Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

October 2013

Reinforcement learning with motivations for realistic agents

Jacquelyne T. Forgette
The University of Western Ontario

Supervisor
Mike Katchabaw
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

© Jacquelyne T. Forgette 2013

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Artificial Intelligence and Robotics Commons, and the Other Computer Sciences Commons

Recommended Citation

Forgette, Jacquelyne T., "Reinforcement learning with motivations for realistic agents" (2013). *Electronic Thesis and Dissertation Repository*. 1651.
https://ir.lib.uwo.ca/etd/1651

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact tadam@uwo.ca.

المنارة للاستشارات

REINFORCEMENT LEARNING WITH MOTIVATIONS FOR REALISTIC AGENTS

(Thesis Format: Monograph)

by

Jacquelyne T. Forgette

Graduate Program in Computer Science

Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science

University of Western Ontario
London, Ontario
October 6, 2013

© Jacquelyne T. Forgette 2013

Abstract

Believable virtual humans have important applications in various fields, including computer based video games. The challenge in programming video games is to produce a non-player controlled character that is autonomous, and capable of action selections that appear human. In this thesis, motivations are used as a basis for learning using reinforcements. With motives driving the decisions of the agents, their actions will appear less structured and repetitious, and more human in nature. This will also allow developers to easily create game agents with specific motivations, based mostly on their narrative purposes. With minimum and maximum desirable motive values, the agents use reinforcement learning to maximize their rewards across all motives. Results show that an agent can learn to satisfy as many as four motives, even with significantly delayed rewards, and motive changes that are caused by other agents. While the actions tested are simple in nature, they show the potential of a more complicated motivation driven reinforcement learning system. The game developer need only define an agent's motivations, based on the game narrative, and the agent will learn to act realistically as the game progresses.

Acknowledgements

To all my teachers, professors, and supervisors, from the past, present, and future.

للاستشارات المنارة

Contents

Abstract **ii**

Acknowledgements **iii**

1 Introduction **1**

1.1 Artificial Intelligence in Games 1

1.2 Believability 3

1.3 Proposed Method 4

1.4 Thesis Outline 5

2 Reinforcement Learning **6**

2.1 Overview 6

2.2 Agent-Environment 6

2.3 Policy 7

2.4 Expected Reward 8

2.5 Markov Decision Process 8

2.6 Value Function 9

2.7 Temporal-Difference 9

2.7.1 Policy Evaluation 10

2.7.2 On-Policy Control: Sarsa 10

2.7.3 Off-Policy Control: Q-Learning 12

2.8 Eligibility Traces 13

2.8.1 Theoretical Basis 13

2.8.2 Algorithmic Implementation 14

2.8.3 Sarsa(λ) 15

2.8.4 Q(λ) 15

2.9 Generalization 15

2.10 Modelling and Planning 16

2.11 Hierarchical RL 17

2.11.1 Scaling Problem 19
2.11.2 Incorporating Options 20
2.11.3 Option Discovery 20
2.12 Summary 21

3 Realistic Agents **22**
3.1 Personality 22
3.2 Desire Theory 23
3.3 Emotion RL 24
3.4 Motivated RL 26
3.5 Social Believability 27
3.6 Summary 28

4 Method **30**
4.1 Overview 30
4.2 Approach 30
4.2.1 Environment 31
4.2.2 Motives 31
4.2.3 Reinforcement Function 32
4.2.4 Function Approximation 33
4.2.5 State Representation 34
4.3 Game Scenarios 35
4.4 Evaluative Measures 35
4.5 Testing Approach 36
4.5.1 Parameters 36
4.5.2 Exploration Rate 37
4.5.3 Training and Testing 38
4.5.4 Discount and Trace-Decay 38
4.5.5 Multiple Motives 38
4.5.6 Graphing Results 39
4.5.7 Testing Outline 39
4.6 Summary 39

5 Game One **41**
5.1 Testing Outline 41
5.2 Sarsa 41
5.2.1 Exploration Rate 42

5.2.2 Reinforcement Learning Rate 43
5.2.3 Q-Function ANN Learning Rate 44
5.2.4 Individual Motive Reward 44
5.2.5 Q-Function Hidden Neurons 45
5.3 Q-Learning 46
5.3.1 Exploration Rate 46
5.3.2 Reinforcement Learning Rate 47
5.3.3 Q-Function ANN Learning Rate 47
5.3.4 Individual Motive Reward 48
5.3.5 Q-Function Hidden Neurons 49
5.4 Dyna-Q 50
5.4.1 Model-Function Hidden Neurons 50
5.4.2 Model-Function Learning Rate 51
5.4.3 Dyna-Q Planning Steps 51
5.5 Discussion 52
5.5.1 Parameters 52
5.5.2 Consistency 53
5.5.3 Training Time of Optimal Policy 54
5.5.4 Qualitative Analysis 55
5.6 Summary 56

6 Game Two **57**
6.1 Testing Outline 57
6.2 Sarsa 58
6.2.1 Exploration Rate 58
6.2.2 Reinforcement Learning Rate 59
6.2.3 Q-Function Learning Rate 59
6.2.4 Individual Motive Reward 60
6.2.5 Q-Function Hidden Neurons 61
6.3 Q-Learning 61
6.3.1 Exploration Rate 62
6.3.2 Reinforcement Learning Rate 62
6.3.3 Q-Function Learning Rate 63
6.3.4 Individual Motive Reward 64
6.3.5 Q-Function Hidden Neurons 64
6.4 Dyna-Q 65

    6.4.1 Model-Function Hidden Neurons 65
    6.4.2 Model-Function Learning Rate 66
    6.4.3 Dyna-Q Planning Steps 67
  6.5 Discussion 67
    6.5.1 Parameters 67
    6.5.2 Consistency 68
    6.5.3 Discount and Trace-Decay 69
    6.5.4 Training Time of Optimal Parameters 70
    6.5.5 Qualitative Analysis 70
  6.6 Summary 71

7 Game Three 72
  7.1 Testing Outline 73
  7.2 Sarsa 73
    7.2.1 Exploration Rate 73
    7.2.2 Reinforcement Learning Rate 74
    7.2.3 Q-Function Learning Rate 75
    7.2.4 Individual Motive Reward 76
    7.2.5 Q-Function Hidden Neurons 77
  7.3 Q Learning 78
    7.3.1 Exploration Rate 79
    7.3.2 Reinforcement Learning Rate 79
    7.3.3 Q-Function Learning Rate 80
    7.3.4 Individual Motive Reward 81
    7.3.5 Q-Function Hidden Neurons 82
  7.4 Dyna-Q 83
    7.4.1 Model Function Hidden Neurons 83
    7.4.2 Model Function Learning Rate 84
    7.4.3 Dyna-Q Planning Steps 85
  7.5 Discussion 86
    7.5.1 Parameters 87
    7.5.2 Consistency 87
    7.5.3 Discount and Decay-Trace 88
    7.5.4 Training Time 89
    7.5.5 Qualitative Analysis 90
  7.6 Summary 93

8 Game Four 94

8.1 Testing Outline 94
8.2 Sarsa with ε-greedy 95
8.3 Sarsa with Softmax 99
8.4 Q-Learning with ε-greedy 103
8.5 Q-Learning with softmax 108
8.6 Dyna-Q with ε-greedy 111
8.7 Dyna-Q with Softmax 113
8.8 Discussion 119
  8.8.1 Parameters 119
  8.8.2 Consistency 121
  8.8.3 Training Time 122
  8.8.4 Qualitative Analysis 123
8.9 Summary 124

9 Game Five 125

9.1 Testing Outline 126
9.2 Sarsa and ε-greedy 126
9.3 Discussion 130
  9.3.1 Parameters 130
  9.3.2 Consistency 130
  9.3.3 Training Time 131
  9.3.4 Qualitative Analysis 132
9.4 Summary 135

10 Discussion and Conclusion 136

10.1 Discussion 136
10.2 Main Contributions 139
10.3 Future Work 139

Bibliography 141

Vita 145

List of Tables

3.1 Hypotheses of Reiss's theory of 16 basic desires [34]. 25

3.2 Motives in Reiss's Theory of 16 Basic Desires [34]. 25

3.3 Reduced list of appraisals that categorize emotional reactions. 26

4.1 Spilt up into the different RL algorithms, the table describes what parameters are necessary in each aspect of the system. The parameter mR defines the scaling factor of a motive's individual reward, Hn is an array that defines the number of hidden neurons in each hidden layer of an ANN, pS defines the number of planning steps during Dyna-Q, η defines the learning rate of an ANN, λ defines the decay-trace value, γ defines the discount value, ε_s defines the starting exploration rate, ε_e defines the ending exploration rate, τ_s defines the starting temperature value, τ_e defines the ending temperature value, α_s defines the starting RL learning rate, and finally, α_e defines the ending RL learning rate. 37

4.2 Testing combinations for discount and trace-decay values, 26 combinations. 38

4.3 Testing outline specifying where to find results of specific tests. 39

5.1 Outline of fixed variables used in the testing configurations for Game One using Sarsa, and ε-greedy. 42

5.2 How ε_s and ε_e vary for different testing configurations in Game One using Sarsa, and $\mathcal{E}$-Greedy. 42

5.3 How α_s and α_e vary for different testing configurations in Game One with Sarsa, and ε-Greedy. 43

5.4 How η_Q changes for different testing configurations in Game One with Sarsa, and ε-Greedy. 44

5.5 How mR changes for different testing configurations in Game One with Sarsa, ε-Greedy. 44

5.6 How Hn_Q is changed for different testing configurations in Game One with Sarsa, and ε-greedy. 45

5.7 Outline of fixed variables used in the testing configurations for Game One using Q-learning, and ε-greedy. 46

5.8 How ε_s and ε_e vary for different testing configurations in Game One using Q-learning, and $\mathcal{E}$-Greedy. 46
5.9 How α_s and α_e vary for different testing configurations in Game One with Q-learning, and ε-Greedy. 47
5.10 How η_Q changes for different testing configurations in Game One with Q-learning, and ε-Greedy. 48
5.11 How mR changes for different testing configurations in Game One with Q-learning, ε-Greedy. 48
5.12 How Hn_Q is changed for different testing configurations in Game One with Q-learning, and ε-greedy. 49
5.13 Outline of fixed variables used in the testing configurations for Game One using Dyna-Q, and ε-greedy. 50
5.14 How Hn_M is changed for different testing configurations in Game One with Dyna-Q, and ε-greedy. 50
5.15 How η_M changes for different testing configurations in Game One with Dyna-Q, and ε-Greedy. 51
5.16 How pS changes for different testing configurations in Game One with Dyna-Q, and ε-Greedy. 51
5.17 Outline of the variables used to test the consistency of Game One's optimal policy, using Dyna-Q and ε-greedy. 54

6.1 Description of Tarzan's actions and their resulting impact on the game world. 57
6.2 Outline of fixed variables used in the testing configurations for Game Two using Sarsa, and ε-greedy. 58
6.3 How ε_s and ε_e vary for different testing configurations in Game Two using Sarsa, and $\mathcal{E}$-Greedy 58
6.4 Outlines how the starting and ending learning values (α_s and α_e) vary for training games (trainN) as well as testing games (testN), in Game Two with Sarsa and ε-greedy. 59
6.5 How η_Q changes for different testing configurations in Game Two with Sarsa, and ε-Greedy. 59
6.6 How mR changes for different testing configurations in Game Two with Sarsa, and ε-Greedy. 60
6.7 How Hn_Q is changed for different testing configurations in Game Two with Sarsa, and ε-greedy. 61
6.8 Outline of fixed variables used in the testing configurations for Game Two using Q-learning, and ε-greedy. 62

6.9 Outlines how ε_s and ε_e vary for different testing configurations in Game Two using Q-learning, and $\mathcal{E}$-Greedy. 62
6.10 Outline how the starting and ending learning values (α_s and α_e) vary for training games (trainN) as well as testing games (testN), in Game Two with Q learning, and ε-greedy. 63
6.11 Outlines how η_Q changes during testing, in Game Two with Q-learning, and ε-greedy. 63
6.12 Outlines how mR changes during testing, in Game Two with Q-learning and ε-greedy. 64
6.13 Outlines how Hn_Q changes during testing, in Game Two with Q learning and ε-greedy. 65
6.14 Outline of fixed variables used in the testing configurations for Game Two using Dyna-Q, and ε-greedy. 65
6.15 Outlines how Hn_M changes with different testing configurations in Game Two with Dyna-Q, and ε-greedy. 66
6.16 Outline how η_M changes during different testing configurations in Game Two with Dyna-Q, and ε-greedy. 66
6.17 Outlines how pS changes for different testing configurations in Game Two with Dyna-Q, and ε-Greedy. 67
6.18 Outline of the variables used to test the consistency of Game Two's optimal parameters. 69

7.1 Game Three agent motive thresholds. 72
7.2 Outline of fixed variables used in the testing configurations for Game Three using Sarsa, and ε-greedy. 73
7.3 Outlines how ε_s, and ε_e, vary during testing configurations in Game Three using Sarsa, and $\mathcal{E}$-Greedy. 73
7.4 Outline how the starting, and ending learning values (α_s and α_e), vary for training games (trainN), as well as testing games (testN), in Game Three with Sarsa, and ε-greedy. 74
7.5 Outlines how η_Q changes during testing configurations in Game Three with Sarsa, and ε-Greedy. 75
7.6 Outlines mR changes for different testing configurations in Game Three with Sarsa, and ε-Greedy. 76
7.7 Outlines how Hn_Q changes for different testing configurations in Game Three with Sarsa, and ε-greedy. 77
7.8 Outline of the fixed variables used in the testing configurations for Game Three using Q-learning, and ε-greedy. 78
7.9 Outline how ε_s and ε_e vary during testing configurations for Game Three tests with Q-learning and ε-greedy. 79

7.10 Outline how the starting, and ending learning values (α_s and α_e), vary for training games (trainN), as well as testing games (testN), in Game Three with Q learning, and ε-greedy. 79
7.11 Outlines how η_Q changes during testing configurations in Game Three with Q-learning, and ε-greedy. 80
7.12 Outlines how mR changes during testing, in Game Three with Q-learning and ε-greedy. 81
7.13 Outlines how Hn_Q changes during testing in Game Three using Q-learning, and ε-greedy. 82
7.14 Outline of fixed variables used in the testing configurations for Game Three using Dyna-Q, and ε-greedy. 83
7.15 Outlines how Hn_M changes with different testing configurations in Game Three with Dyna-Q, and ε-greedy. 84
7.16 Outline how η_M changes during different testing configurations in Game Three with Dyna-Q, and ε-greedy. 85
7.17 Outlines how pS changes for different testing configurations in Game Three with Dyna-Q, and ε-Greedy. 85
7.18 Outline of the variables used to test the consistency of Game Three's optimal parameters. 88
7.19 This table outlines exactly what discount rate, and trace-decay parameter, were used to train the policies chosen for qualitative analysis. 90

8.1 Description of Tarzan's actions, and the resulting changes in the game world, for Game Four. 94
8.2 Testing outline for Game Four. 95
8.3 Parameter values tested in Game Four with Sarsa, and ε-greedy. 96
8.4 Parameter values tested for Game Four with Sarsa, and Softmax. 99
8.5 Parameter values tested in Game Four with Q-learning, and ε-greedy. 104
8.6 Parameter values tested in Game Four with Q-learning, and Softmax. 109
8.7 Parameter values tested in Game Four using Dyna-Q, and ε-greedy. 112
8.8 Parameter values tested in Game Four with Dyna-Q, and Softmax. 114
8.9 Outline of the variables used to test the consistency of the results. 121
8.10 Statistics measuring consistency of Game Four results (percent of reward). 121
8.11 Statistics measuring consistency of Game Four results (percent of steps in bounds). . 122

9.1 Description of motivations for all agents, in the Game Five scenario. 125
9.2 Description of how object actions change agent's motives, in the Game Five scenario. 125
9.3 Parameter values tested in Game Four with Sarsa, and ε-greedy. 126

9.4 Outline of the variables used to test the consistency of the results. 130

9.5 Statistics measuring consistency of Game Five results (percent of steps in bounds for Tarzan). 131

9.6 Statistics measuring consistency of Game Five results (percent of reward for Tarzan). 131

List of Figures

2.1 Basic Agent-Environment Interaction in Reinforcement Learning. 7

2.2 On-policy control methodology for updating Q^π while following the policy π. 11

2.3 Off-policy control methodology for updating Q^π without following policy π. 12

2.4 Backup diagrams of TD methods ranging from one-step backups to the full termination backups of Monte Carlo methods, where T is the final time step in the episode [40]. . 13

2.5 Backup diagrams of TD(λ). When $\lambda = 0$, the backup is reduced to TD(1-step) whereas when $\lambda = 1$, the backup becomes the same as Monte Carlo [40]. 14

2.6 Agent-Environment-Model Interaction in the DynaQ Architecture. 17

2.7 Options replace a series of low-level actions. a) A search tree outlining the task of searching for a series of actions that result in a favorable state. b) A search tree using options instead of actions. The task of searching through the search space is simplified when groups of actions are replaced with an option. [6] 19

4.1 Finite Motive Scale. 32

4.2 For a motive m, its current value is $m_{current}$, maximum threshold is m_{max}, minimum threshold is m_{min}, and the distance between $m_{current}$, and the nearest threshold is defined by d. The minimum scaling factor is $s_{min} = m_{max}$, and the maximum scaling factor is $s_{max} = 100 - m_{max}$. 32

4.3 Features that are used as input into the Q-function approximation ANN, where M is the number of motives, P is the number of places in the game world, A is the number of agents in the game world excluding the current agent, and O is the total number of object types. 34

5.1 Comparing the results of varying ε_s and ε_e during final policy tests, where t-n is the n^{th} testing configuration according to Table 5.2. The testing configurations are run in Game One using Sarsa, and ε-greedy. 43

5.2 Comparing the results of varying α_s and α_e during final policy tests, where t-n defines the n^{th} testing configuration according to Table 5.3. The testing configurations are run in Game One using Sarsa, and ε-greedy. 43

5.3 Comparing the results of varying η_Q during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.4. The testing configurations are run in Game One using Sarsa, and ε-greedy. 44

5.4 Comparing the results of varying mR during final policy tests, where t-n defines the n^{th} test according to Table 5.5. The testing configurations are run in Game One using Sarsa, and ε-greedy. 45

5.5 Comparing the results of varying Hn_Q during final policy tests, where t-n defines the n^{th} test according to Table 5.6. The testing configurations are run in Game One using Sarsa, and ε-greedy. 45

5.6 Comparing the results of varying ε_s and ε_e during final policy tests, where t-n is the n^{th} testing configuration according to Table 5.8. The testing configurations are run in Game One using Q-learning, and ε-greedy. 47

5.7 Comparing the results of varying α_s and α_e, where t-n defines the n^{th} test according to Table 5.9, consisting only of the final policies reached by each parameter configuration (130 results). The agent follows the policy with absolutely no exploration. The testing configurations are run in Game Two using Q-learning, and ε-greedy. 48

5.8 Comparing the results of varying η_Q during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.10. The testing configurations are run in Game One using Q-learning, and ε-greedy. 48

5.9 Comparing the results of varying mR during final policy tests, where t-n defines the n^{th} test according to Table 5.11. The testing configurations are run in Game One using Q-learning, and ε-greedy. 49

5.10 Comparing the results of varying Hn_Q during final policy tests, where t-n defines the n^{th} test according to Table 5.12. The testing configurations are run in Game One using Q-learning, and ε-greedy. 49

5.11 Comparing the results of varying Hn_M during final policy tests, where t-n defines the n^{th} test according to Table 5.14. The testing configurations are run in Game One using Dyna-Q, and ε-greedy. 50

5.12 Comparing the results of varying η_M during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.15. The testing configurations are run in Game One using Dyna-Q, and ε-greedy. 51

5.13 Comparing the results of varying pS during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.16. The testing configurations are run in Game One using Dyna-Q, and ε-greedy. 52

5.14 Compare the consistency of the final Game One policies learned by the RL agent from ten repeated tests, where t-n defines the n^{th} repeated test. 54

5.15 Showing the percent of reward received after every game (1000 game steps), comparing two contrasting values of η_M. 54

5.16 Showing the percent of reward received after every game (1000 game steps), comparing two contrasting values of pS 55

5.17 Qualitative analysis of best and worst policies found by the set of parameters in Table 5.17. 55

6.1 Comparing the results of varying ε_s and ε_e during final policy tests, where t-n is the n^{th} testing configuration according to Table 6.3. These Game Two testing configurations are run using Sarsa, and ε-greedy. 58

6.2 Comparing the results of varying α_s, and α_e, where t-n defines the n^{th} test according to Table 6.4, consisting only of (test10), the final policy reached by each parameter configuration (130 results). The agent follows the policy with no exploration, and no learning. The testing configurations are run in Game Two using Sarsa, and ε-greedy 59

6.3 Comparing the results of varying η_Q, where t-n defines the n^{th} test according to Table 6.5, consisting of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Sarsa, and ε-greedy. 60

6.4 Comparing the results of varying mR, where t-n defines the n^{th} test according to Table 6.6, consisting only of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Sarsa, and ε-greedy. 60

6.5 Comparing the results of varying Hn_Q during final policy tests, where t-n defines the n^{th} test according to Table 6.7. The testing configurations are run in Game One using Sarsa, and ε-greedy. 61

6.6 Comparing the results of varying ε_s and ε_e, where t-n defines the n^{th} test according to Table 6.9, consisting of the final policies. The testing configurations are run in Game Two using Q-learning, and ε-greedy. 62

6.7 Comparing the results of varying α_s and α_e, where t-n defines the n^{th} test according to Table 6.10, consisting only of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Q-learning, and ε-greedy. 63

6.8 Comparing the results of varying η_Q, where t-n defines the n^{th} test according to Table 6.11, consisting of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Q-learning, and ε-greedy. 64

6.9 Comparing the results of varying mR, where t-n defines the n^{th} test according to Table 6.12, consisting of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Q-learning, and ε-greedy. 64

6.10 Comparing the results of varying Hn_Q, where t-n defines the n^{th} test according to Table 6.13, consisting of the final policies reached by each parameter configuration (130 results). The testing configurations are run in Game Two using Q-learning, and ε-greedy. 65

6.11 Comparing the results of varying Hn_M during final policies, where t-n defines the n^{th} test according to Table 6.15. The testing is run in Game Two using Dyna-Q, and ε-greedy. 66

6.12 Comparing the final policies while varying η_Q, where t-n defines the n^{th} test according to Table 6.16. The testing configurations are run in Game Two using Dyna-Q, and ε-greedy. 66

6.13 Comparing the results of varying pS during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 6.17. The testing configurations are run in Game Two using Dyna-Q, and ε-greedy. 67

6.14 Comparing the consistency of the final policies, trained using optimal parameters outlined in Table 6.18, where t-n defines the n^{th} repeated test. 69

6.15 Comparing the impact of γ and λ on the percent of total reward given the set of optimal parameters found in Table 6.18. 69

6.16 Comparing the impact of training time on the percent of reward, for policies using optimal parameters seen in Table 6.18. 70

6.17 Actual motive values observed during a game following one of the optimal policies found using parameters described in Table 6.18. The red lines indicate the boundaries for minimum, and maximum desirable values, that are different for each motivation. 71

6.18 Actual motive values observed during a game following one of the sub-optimal policies found using parameters described in Table 6.18. The red lines indicate the boundaries for minimum, and maximum desirable values, that are different for each motivation. 71

7.1 Comparing the results of varying ε_s, and ε_e, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.3. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 74

7.2 Comparing the results of varying ε_s, and ε_e, for the agent Jane, where j-n defines the n^{th} test according to Table 7.3. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 74

7.3 Comparing the results of varying α_s, and α_e, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.4. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 75

7.4 Comparing the results of varying α_s, and α_e, for the agent Jane, where j-n defines the n^{th} test according to Table 7.4. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 75

7.5 Comparing the results of changing η_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.5. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 76

7.6 Comparing the results of changing η_Q, for the agent Jane, where j-n defines the n^{th} test according to Table 7.5. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 76

7.7 Comparing the results of changing mR, from the agent Tarzan, where t-n defines the n^{th} test according to Table 7.6. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 77

7.8 Comparing the results of changing mR, from the agent Jane, where j-n defines the n^{th} test according to Table 7.6. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 77

7.9 Comparing the results of changing Hn_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.7. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 78

7.10 Comparing the results of changing Hn_Q, for the agent Jane, where j-n defines the n^{th} test according to Table 7.7. The testing configurations are run in Game Three using Sarsa, and ε-greedy. 78

7.11 The results of varying ε, where t-n and j-n are the n^{th} test from Table 7.9. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 79

7.12 Comparing the results of varying α_s, and α_e, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.10. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 80

7.13 Comparing the results of varying α_s, and α_e, for the agent Jane, where j-n defines the n^{th} test according to Table 7.10. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 80

7.14 Comparing the results of changing η_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.11. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 81

7.15 Comparing the results of changing η_Q, for the agent Jane, where j-n defines the n^{th} test according to Table 7.11. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 81

7.16 Comparing the results of changing mR, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.12. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 82

7.17 Comparing the results of changing mR, for the agent Jane, where j-n defines the n^{th} test according to Table 7.12. The testing configurations are run in Game Three using Q-learning, and ε-greedy. 82

7.18 Comparing the results of changing Hn_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.13, consisting of the final policies reached by each parameter configuration (130 results). The testing configurations are run in Game Three using Q-learning, and ε-greedy. 83

7.19 Comparing the results of changing Hn_Q, for the agent Jane, where t-n defines the n^{th} test according to Table 7.13, consisting of the final policies reached by each parameter configuration (130 results). The testing configurations are run in Game Three using Q-learning, and ε-greedy. 83

7.20 Comparing the results of changing Hn_M, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy. 84

7.21 Comparing the results of changing Hn_M, for the agent Jane, where j-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy. 84

7.22 Comparing the results of changing η_M, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy. 85

7.23 Comparing the results of changing η_M, for the agent Jane, where j-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy. 85

7.24 Comparing the results of changing pS, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy. 86

7.25 Comparing the results of changing pS, for the agent Jane, where j-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy. 86

7.26 Comparing the percent of reward from policies trained using optimal parameters outlined in Table 7.18. 88
7.27 Comparing the percent of steps in bounds from policies trained using optimal parameters outlined in Table 7.18. 88
7.28 Comparing the impact of γ and λ on the percent of reward for Tarzan, and Jane, given the optimal set of parameters in Table 7.18. 89
7.29 Comparing how of training time influences the percent of reward, and the percent of steps in bounds, for Tarzan, and Jane, using the optimal Game Three parameter in Table 7.18. 90
7.30 Actual motive values for the agent Tarzan, where the optimal policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values. 91
7.31 Actual motive values for the agent Jane, where the optimal policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values. 91
7.32 Actual motive values for the agent Tarzan, where the terrible policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values. 92
7.33 Actual motive values for the agent Jane, where the terrible policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values. 92

8.1 Comparing the percent of reward of α, and ε, during testing for Game Four with Sarsa, and ε-greedy. 96
8.2 Comparing the percent of reward of $\gamma\lambda$, and mR, during testing for Game Four with Sarsa, and ε-greedy. 96
8.3 Comparing the percent of reward of η_Q, and Hn_Q, during testing for Game Four with Sarsa, and ε-greedy. 97
8.4 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Sarsa, and ε-greedy. 97
8.5 Results (percent in bounds) from training games 58, 59, and 60, during Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$ are included. 98
8.6 Comparing the percent of steps in bounds of ε, and α, during testing for Game Four with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.05$, are included. 98

8.7 Comparing the percent of steps in bounds of η_Q, and Hn_Q, during testing for Game Four with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.05$, are included. . . . 98

8.8 Comparing the percent of reward of τ, and $\gamma\lambda$, during testing for Game Four with Sarsa, and Softmax. . . . 99

8.9 Comparing the percent of reward of Hn_Q, and mR, during testing for Game Four with Sarsa, and Softmax. . . . 100

8.10 Comparing the percent of reward of η_Q during testing for Game Four with Sarsa, and Softmax. . . . 100

8.11 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax. . . . 100

8.12 All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax. . . . 101

8.13 Results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$ are included in this scatter plot. . . . 101

8.14 Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$ are included in this scatter plot. . . . 102

8.15 Comparing the percent of reward, and the percent of steps in bounds, of τ during testing for Game Four with Sarsa, and Softmax. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$. . . . 102

8.16 Comparing the percent of reward, and the percent of steps in bounds, of η_Q during testing for Game Four with Sarsa, and Softmax. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$. . . . 103

8.17 Comparing the percent of reward, and the percent of steps in bounds, of Hn_Q during testing for Game Four with Sarsa, and Softmax. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$. . . . 103

8.18 Comparing the percent of reward of α, and ε, during testing for Game Four with Q-learning, and ε-greedy . . . 104

8.19 Comparing the percent of reward of $\gamma\lambda$, and mR, during testing for Game Four with Q-learning, and ε-greedy. . . . 104

8.20 Comparing the percent of reward of η_Q, and Hn_Q, during testing for Game Four with Q-learning, and ε-greedy. . . . 105

8.21 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy. . . . 105

8.22 All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy. 106

8.23 Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy. The results are limited to those with $\gamma = 0.9$, and $\lambda = 0.9$. 106

8.24 Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$. 107

8.25 Comparing the percent of steps in bounds of α, and ε, during testing for Game Four with Q-learning, and ε-greedy . The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$. 107

8.26 Comparing the percent of steps in bounds of η_Q, and Hn_Q, during testing for Game Four with Q-learning, and ε-greedy. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$. 108

8.27 Comparing the results of ε, during testing for Game Four with Q-learning, and ε-greedy. The results are limited to those with$\gamma = 0.9$, $\lambda = 0.9$, $mR = -0.1$, $\eta_Q = 0.7$, $Hn_Q = [100]$, $mR = -0.1$, $\alpha_s = 0.9$, and $\alpha_e = 0.5$. 108

8.28 Comparing the percent of reward of τ, and α, during testing for Game Four with Q-learning, and Softmax. 109

8.29 Comparing the percent of reward of Hn_Q, and mR, during testing for Game Four with Q-learning, and Softmax. 109

8.30 Comparing the percent of reward of η_Q during testing for Game Four with Q-learning, and Softmax. 110

8.31 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Q-learning, and Softmax. 110

8.32 All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and Softmax. 111

8.33 Comparing the percent of steps in bounds of Hn_Q, and τ, during testing for Game Four with Q-learning, and Softmax. The results are limited to those with $mR = -0.1$, $\eta_Q = 0.9$, $\alpha_s = 0.9$, and $\alpha_e = 0.9$. 111

8.34 Comparing the percent of reward of ε, and mR, during testing for Game Four with Dyna-Q, and ε-greedy. 112

8.35 Comparing the percent of reward of Hn_Q, and Hn_M, during testing for Game Four with Dyna-Q, and ε-greedy. 112

8.36 Comparing the percent of reward of pS during testing for Game Four with Dyna-Q, and ε-greedy. 113

8.37 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and ε-greedy. 113
8.38 Comparing the percent of reward of ε, and mR, during testing for Game Four with Dyna-Q, and Softmax. 114
8.39 Comparing the percent of reward of Hn_M, and pS, during testing for Game Four with Dyna-Q, and Softmax. 115
8.40 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. 115
8.41 All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. 116
8.42 All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. 116
8.43 All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. 117
8.44 Results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. These results do not include those with $\tau_s = 0.01$, and $\tau_e = 0.00001$. 117
8.45 Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. These results do not include those with $\tau_s = 0.01$, and $\tau_e = 0.00001$. 118
8.46 Comparing the performance of pS, during testing for Game Four with Dyna-Q, and Softmax. The results are limited to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$. 118
8.47 Comparing the performance of Hn_M, during testing for Game Four with Dyna-Q, and Softmax. The results are limited to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$. 119
8.48 Comparing the performance of mR, during testing for Game Four with Dyna-Q, and Softmax. The results are limited to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$. 119
8.49 Showing the percent of reward after every training game to compare the influence of training time. Each run number includes results from the 20 repeated tests from Table 8.9. 123
8.50 Showing the percent of steps in bounds after every training game to compare the influence of training time. Each run number includes results from the 20 repeated tests from Table 8.9. 123
8.51 Qualitative analysis of the policies found using parameters outlines in Table 8.9. . . . 124

9.1 Comparing the performance of α in Game Four testing with Sarsa, and ε-greedy. . . . 126
9.2 Comparing the performance of η_Q in Game Four testing with Sarsa, and ε-greedy. . . 127

9.3 Comparing the performance of ε in Game Four testing with Sarsa, and ε-greedy. . . 127

9.4 Comparing the performance of γ, and λ, in Game Four testing with Sarsa, and ε-greedy. 127

9.5 All results (percent of steps in bounds) from training games 88, 89, and 90, during Game Four testing with Sarsa, and ε-greedy, separated by ε and $\gamma\lambda$. 128

9.6 All results (percent of reward) from training games 88, 89, and 90, during Game Four testing with Sarsa, and ε-greedy, separated by η_Q, and $\gamma\lambda$. 128

9.7 Comparing the performance of α in Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$, are included in the graph. 129

9.8 Comparing the performance of η_Q in Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$, are included in the graph. 129

9.9 Comparing the performance of ε in Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$, are included in the graph. 129

9.10 Comparing the impact of training time on the percent of reward, in Game Five. Every box includes 10 repeated results, found with parameter values from Table 9.4. 132

9.11 Comparing the impact of training time on the percent of steps in bounds, in Game Five. Every box includes 10 repeated results, found with parameter values from Table 9.4. 132

9.12 Qualitative analysis of the policy with the highest percent of reward for Tarzan, found using parameters outlined in Table 9.4. 133

9.13 Qualitative analysis of the policy with the least amount of percent of reward for Tarzan, found using parameters outlines in Table 9.4. 134

Chapter 1

Introduction

The application of believable virtual humans is important for many fields of research, with varying expectations, definitions and requirements of realism. Building a full virtual human is a complex task, including visual non-verbal emotional responses, cognition, behaviour and conversational skills [11]. Each facet of a human's response may be influenced by underlying personality traits, emotions and motivations [5, 35, 32] which are also influenced by sociological connections and perception [16, 37, 3].

In video games, realistic agents have long been a problem to developers and gamers. For large immersive game worlds, players have come to expect the presence of computer generated agents as either necessary for the narrative, or for populating a scene. While human players would be ideal, they cannot be controlled in a way that is conducive to the game narrative. For this reason, non-player characters (NPC) will always be an essential part of video games.

When developing a game, a developer will choose from a wide range of artificial intelligence (AI) techniques to ensure the NPC will behave as realistically as possible, given the requirements and constraints of the game.

1.1 Artificial Intelligence in Games

The application of AI in video games differs from most other AI applications such as robotics, military defence or data mining. The main distinction between video game AI and the traditional academic definition, is their differing goals. Contrary to academic AI, video game AI seeks to create computer controlled agents that provide a challenge, but are ultimately defeated, in an entertaining manner [31]. The goal of game AI is to have players so immersed in the game world, that they momentarily forget about the real world [12]. In essence, game AI is the illusion of intelligence on behalf of NPCs [19]. Despite their usefulness, complex and computational intensive AI should only be used when its solution serves to further entertain and engage the player.

An NPC is defined as a computer controlled character that senses, thinks and acts. The agent continuously loops through a cycle of sensing its environment, evaluating the data gathered, making a decision (optional learning and remembering) until finally acting out the selected behaviour. Sensing could include data gathered from simulated vision, hearing and even inter-agent communications. The thinking stage requires the agent to evaluate the current knowledge of its environment, and is usually what is considered *true* Game AI [31]. The thinking problem is often solved by utilizing pre-coded expert knowledge, or search algorithms.

Expert knowledge can be transmitted in many ways, including finite state machines, logical inference and decision trees. It is simple and natural to program, making it an attractive solution. Developers simply write a series of *if-then* statements, enabling the agent to make *good* decisions. Given that expert knowledge requires many hand crafted rules, they do not typically scale successfully to more complex problems. Also, the designed systems are incomplete solutions and require game testers to uncover unspecified behaviours. When bugs are uncovered, they are patched by adding more rules, making the system fragile. Only with narrow problem domains is expert knowledge sufficient as a solution [31].

Search algorithms can also be used to address agent *thinking*, by discovering sequences of steps that will result in an ideal state, or a complete solution. The most common search technique is path-finding, when the agent must decide its next move while considering a final destination [31].

When learning must occur, expert knowledge or search techniques are insufficient. Without learning or remembering, the agent will never learn from its mistakes, make better decisions over time, or adapt to a particular player. The majority of existing game AI implementations are limited by predefined decisions and static control systems, which lead to predictable and repetitive behaviours [42].

Machine learning is a sub-set of AI concerned with creating algorithms that enable the computer to learn and improve its performance through experience [28]. Learning consists of remembering specific outcomes, and generalizing to unknown situations [31]. It consists of a diverse set of algorithms that provide many different solutions, each suitable for specific types of applications. Such diversity, has advantages and disadvantages. There are often many algorithms that can be used in a particular situation, creating uncertainty as to the best algorithm for the given task. While care must be taken in this regard, machine learning techniques are critical in addressing dynamic and emergent game-play.

Research into machine learning began with perfect knowledge games (Chess, Checkers, Othello...) with large but finite state spaces. Perfect information games have been conquered by machine learning researchers by the end of the 1990s. In 1994, the Checkers World Champion Marion Tinsley, was defeated by the program Chinook. In 1997, the World Chess Champion Garry Kasparov was defeated by the program Deep Blue. Finally, the World Othello Champion Takeshi Murakami was defeated by the program Logistello, in 1997 [14].

Most games do not include perfect knowledge and often have hidden states, probabilistic game play, or even multiple opponents. Games such as Poker and Backgammon were the new target of machine learning research. The Backgammon program was sufficiently developed and attained good performance; however Poker still remains a challenge, with current programs incapable of competing at the world level [14].

Interactive video games included new types of game-play with complex environments, a human controlled protagonist and computer controlled NPCs. Computer controlled game agents require realistic behaviour, to immerse the gamer into the game-world. Even though machine learning techniques are promising, they are not often implemented in commercial video games, requiring years of experience and in depth knowledge to implement correctly. Most games do not require such advanced algorithms, as their AI tasks could be completed with less complex techniques, resulting in similar performance with easier programming, tuning and testability [31].

1.2 Believability

The distinction of believability, related to video game character behaviour, is more accurately defined as the appearance of rational behaviour. While video game literature use the term believability to define a video game character that appears to act as a human, this in no way indicates that a believable game agent is the complete solution to the complex problem of human behaviour. Psychologists would even argue that completely rational behaviour is unbelievable.

The requirements for achieving a believable game agent are extensive and non-trivial, but the reward in terms of player immersion and satisfaction make this an important problem to solve [2]. As written in [37], “agents are considered believable when they are viewed by an audience as endowed with thoughts, desires, and emotions, typical of different personalities”. This definition of believability is not necessarily a definition of character but an illusion of life, permitting the audience’s suspension of disbelief. While this idea of believability has long been studied in all types of arts including literature, theatre, films, etc., its difficulty in relation to video games concerns the required interactivity of a game agent [20]. This level of interactivity requires autonomous and flexible behaviour that is not defined *a priori*.

With computer games, the NPC and human controlled character appear identical in appearance, somewhere along the scale between ultra-realistic or highly stylized. Given facial expressions or body language are not (yet) directly translated from human player to their game world counterpart, players critique a character’s speech and actions when judging if that character is computer or human controlled. In a first person shooter (FPS), NPCs were often very easy to identify, given they sometimes did not move while being fired upon. Large immersive game worlds are even more complex than an FPS and require more complex decision making by NPCs.

Personality has become a term that is used in everyday conversation, in describing the nature of a person. It is often the case where a person's actions are said to be consistent with their personality. Trait personality models propose that certain persistent traits have an influence on behaviour. Trait models use relatively independent personality traits, as dimensions, where any particular personality can be explained using the personality trait axes [5]. Trait theory shows how large groups of people are similar, showing the dynamics of personality across people [32].

While personality is said to be fixed, behaviour is often considered to be highly dependent upon the current situation, making personality traits a poor predictor for human behaviour.

Intra-individual structure of personality occurs when goals and motivations are used to explain the dynamics of human personality, instead of personality traits [32]. A person can be distinctly represented through varying motives and motive intensities. Given that an individual's motives and goals are changing in regards to different situations, they are a better predictor for human behaviour than traits [35].

In video games, a human player's actions are driven by their motivations, exhibiting cooperative, competitive or simply chaotic action selections. To determine what a player will do, find out their desires and predict that they will try and satisfy those desires. A desire may include economic, exploratory, military, etc. Motivation is used to define the urge to perform certain actions based on internal needs relating to survival and self-sufficiency [7]. To emulate this human realism in NPCs, motivations would have to drive the learning and reasoning systems.

1.3 Proposed Method

The purpose of this thesis is to create a real-time learning NPC whose personality is defined with motivations. In this way, game developers would be able to develop game characters by directly referencing the game narrative. The NPC will learn what actions it must take to benefit its motives, in real-time.

The theory behind this approach is based on work done by Reiss [35, 34] where motives are the reason in which a person initiates and performs a voluntary behaviour. Motives are said to affect a person's perception, cognition, emotion and ultimately the resultant behaviours. A person must have a motive to have performed any particular action even if that person is directly unaware of the motive.

When the amount of any particular motive is less than desired, the agent will be motivated to raise the amount of that motive through actions. Similarly, when the amount of motive is much more than desired, the agent will be forced to take actions that compensate and avoid actions that would raise the amount of motive even more.

In this thesis, actions are fully defined in terms of how they affect other objects, agents and places in the game world, including how they influence an agent's set of motivations. In this case, the game developer plays the role of teacher by making complicated actions available to the agents.

1.4 Thesis Outline

In this introductory chapter, the concepts of believable agents were introduced along with the psychological and algorithmic theories behind them. Chapter 2 presents the basic RL theory along with more recent relevant advances, like planning and hierarchical skill learning. Chapter 3 explores the psychological theories behind behaviour with a particular highlight on the theories that inspired this thesis. Chapter 4 explains the use of an agent's motivations as a guide for RL, and the general testing approach. Results for Game tests follow in Chapters 5 to 9. Each successive game scenario introduces more complexity in terms of delayed rewards, increased motivations and inter-agent interactions. After the results are presented, Chapter 10 covers what was learned, and proposes interesting possibilities for future research.

Chapter 2

Reinforcement Learning

2.1 Overview

Reinforcement learning (RL) is a sub-field of machine learning, focused on goal directed learning through trial and error interactions between an agent, and an environment. Though many flavours of RL algorithms exist, their goals revolve around developing an action-selection policy that maximizes the reward received across multiple steps. The agent learns from its environment by taking actions, receiving rewards, and adapting its action-selection policy based on past, present and future rewards. Value-based approaches attempt to learn the expected value of each state, with the purpose of arriving at an optimal action-selection policy. More rarely used are the policy search algorithms, that attempt to search directly in the policy parameter space to find the optimal action-selection policy. Any optimization algorithm can be used for policy search, including genetic algorithms and simulated annealing. It must be noted that policy search RL techniques do not attempt to learn a value function.

RL is a very popular learning technique because of its unsupervised nature. One must simply define the reward function, commence the learning algorithm, and an action-selection policy is refined that will maximize the rewards received over time. While this is the best possible case, the process is rarely so simple in reality. If the rewards are not clear, the state is not properly represented, or the learning parameters are wrong for the given task, the learned policy will be sub-optimal.

2.2 Agent-Environment

For RL to learn properly, the environment must be partially or fully observable to the agent. The actions available to the agent may be low level (e.g. turn right, turn left) or more complex (e.g. move to specific position while avoiding obstacles). If the agent observes perfectly all information in the environment, then the RL agent chooses an action based upon the *true state* of the environment. This ideal state representation is not always possible, but is necessary for the theoretical basis of

RL [17].

The goals of an RL agent are defined by a reinforcement function. By carefully setting the necessary positive or negative reinforcements, the goal is outlined to the system in a way that is understandable and achievable. The reinforcement function defines a mapping from state-action pairs to rewards, thereby determining what action is *good* in the *short-term* [40]. RL reward functions must remain unchanged by the RL system, and instead serve as a basis to change the policy. The RL system designer is responsible for defining the reinforcement function.

Figure 2.1 shows the basics of the agent-environment interaction, where the agent chooses an action based upon the current state of the environment. This action produces a change in the environment, either to agents, objects or places. The reinforcement function evaluates the changes caused by the action, and assigns a reward based upon the agents goals.

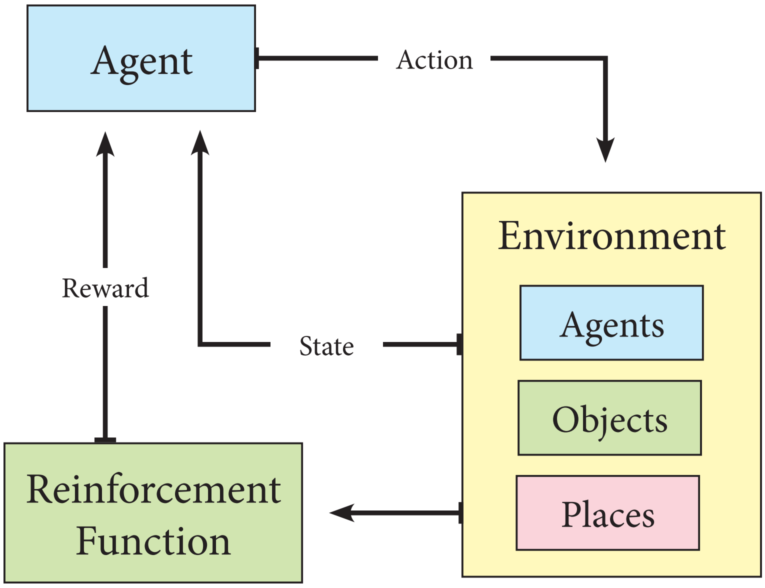


Figure 2.1: Basic Agent-Environment Interaction in Reinforcement Learning.

2.3 Policy

The policy defines how the agent behaves at any given time in the game. It maps the agent's perceived state of the environment to actions. The policy must be learned if it is to be effective. The reinforcement learning process seeks to discover the optimal policy, given the current goals and environment [40, 17].

At each step, there is one action that leads to the greatest estimated future cumulative reward, as determined by the current learned policy. This action is the greedy action, and it is selected if the learning algorithm chooses to exploit the knowledge it currently posses. If an action that is non-greedy is selected, the algorithm is choosing to explore, potentially obtaining greater knowledge that can ultimately be used to achieve greater overall reward. It is not possible to both explore and exploit with one single action selection; proper balance is needed between the two extremes.

The action selection algorithm is the manner in which exploration and exploitation is balanced. A very simple action selection algorithm would always select the greedy action, focusing on maximizing the immediate reward, as opposed to exploring inferior actions that could lead to greater reward [40]. There exist many types of action selection algorithms, the most common are $\varepsilon - greedy$ and Softmax. The $\varepsilon - greedy$ method simply chooses an exploratory action with probability ε, and greedy actions with probability $1 - \varepsilon$. Softmax action selection weights the probability of taking an action by its expected value. The Gibbs, or Boltzmann, distribution is used most commonly in Softmax, choosing an action a on the t^{th} step, with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_b e^{Q_t(b)/\tau}}, \tag{2.1}$$

where τ is the *temperature* parameter, and $\tau \geq 0$.

2.4 Expected Reward

The goal of RL, is to maximize the expected cumulative reward from the current state, until the end of the game. Formally defined, the expected return R_t at time step t is the sequence of rewards received after t, defined as $R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_N$ where N is the last step of the game. This holds true during games that have a final time step.

In the case where the game does not have a set ending, the final time step would be $T = \infty$, potentially making the expected return itself infinite. Discounting is needed to determine the present value of a future reward, for the case of a continuous task with no set ending. The expected return at time t of a discounted task is defined in Equation 2.2, where γ is the discount rate and $0.0 \leq \gamma \leq 1.0$ [40].

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

2.5 Markov Decision Process

The state represents the information about the environment, that is available to the agent at any time. The state signal can include raw sensory measurements or a complex structural representation. However, the state signal should not include information about everything. The ideal state signal should summarize the relative past information in a compact way, where only pertinent information is retained. This idea brings back the definition of the Markov property. A state signal is said to have the Markov property if it succeeds in retaining all relevant information, where the next state can be completely determined by the current state. If the state signal is non-Markov, it can still be thought of as an approximation to a Markov state. A task that satisfies the Markov property is said to be a Markov Decision Process (MDP) [40].

2.6 Value Function

While the reinforcement function gives the agent *short-term* feedback, the agent must have a way of considering the *long-term* consequences of current actions. The value function combines the *short-term* rewards with *long-term* values of future states [40].

Value functions can measure how *good* it is for an agent to be in a particular state, or more specifically, the value in performing a particular action in a given state. Assuming a MDP, the value of the state $s \epsilon S$, while following the policy π, denoted $V^{\pi}(s)$, is the expected return given the starting state s, and subsequently following π. The state-value function $V^{\pi}(s)$ is formally defined as

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}, \tag{2.3}$$

where $E_{\pi}\{\}$ is the expected value if the agent follows the policy π at any time step t.

The value of taking an action $a \epsilon A(s)$ in the state s, while following the policy π, denoted $Q^{\pi}(s, a)$, is the expected return, given the starting state s, taking the action a, and subsequently following π, formally defined in equation 2.4.

$$Q^{\pi}(s, a) = E_{\pi}\{R_t | s_t = s, a_t = a\} \tag{2.4}$$

2.7 Temporal-Difference

Time is an important consideration for action-selection, given that some actions have no apparent consequences until some time has passed. This delayed reward makes assigning blame to a particular action very difficult. When an undesirable result occurs, is it correct to penalize the most recent action, or an action that was taken many steps before? Choosing which action to assign the blame makes reinforcement learning a particularly difficult task [17].

The solution to this learning problem is a mixture of dynamic programming and Monte Carlo methods. Dynamic programming (DP) is a collection of algorithms that are used to compute the optimal policy, given a perfect model of a Markov Decision Process (MDP). Monte Carlo (MC) methods do not require complete knowledge of the environment, and only require *experience* (state-action pairs with their rewards). Temporal difference (TD) learning combines the advantages of dynamic programming with the advantages of Monte Carlo methods in an elegant way. TD does not require a model of the environment's dynamics, but relies on raw experiences for learning. TD also updates its value estimates based upon other learned estimates, without waiting for the final outcome of the task, similar to DP methods [40].

2.7.1 Policy Evaluation

The policy evaluation or *prediction* problem is how to estimate the value function V^{π} for a policy π. TD methods use experience to solve the policy evaluation. For every non terminal state s_t visited at time t, $V(s_t)$ is updated once the return is known. The policy evaluation is formally defined as

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right], \tag{2.5}$$

where R_t is the return following step t, and α is the constant step-size learning rate. With MC methods, R_t is only known after the last time step. However, TD methods update their estimate of $V(s_t)$ based on another estimate of future values, a method called *bootstrapping*. The simplest TD update called TD(1-step), updates $V(s_t)$ with the estimated value of next state $V(s_{t+1})$, defined as

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right], \tag{2.6}$$

where the return is calculated after the next state is reached. The value R_t can also be calculated after 2 steps, 3 steps, ..., N-steps, where N is the final step. This concept of using multiple steps is explored more during the discussion of eligibility traces.

2.7.2 On-Policy Control: Sarsa

The control problem, in which the policy is refined, can be solved in different ways depending on the trade-off between exploration and exploitation.

On-policy control continuously estimates Q^{π} for the current policy π while also modifying π with respect to Q^{π} in a greedy manner. See Figure 2.2.

Algorithm 2.1 Pseudo-code for the on-policy TD control algorithm, Sarsa.

```
Initialize $Q(s,a)$ for all $s$ and $a$
For each episode
    Initialize $s$
    Choose $a$ from $s$ using derived policy (e.g. $\varepsilon - greedy$)
    For each step of the episode
        Take action $a$, observe $r$ and $s$'
        Choose next action $a'$ from $s'$ using derived policy (e.g. $\varepsilon - greedy$)
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    end
end
```

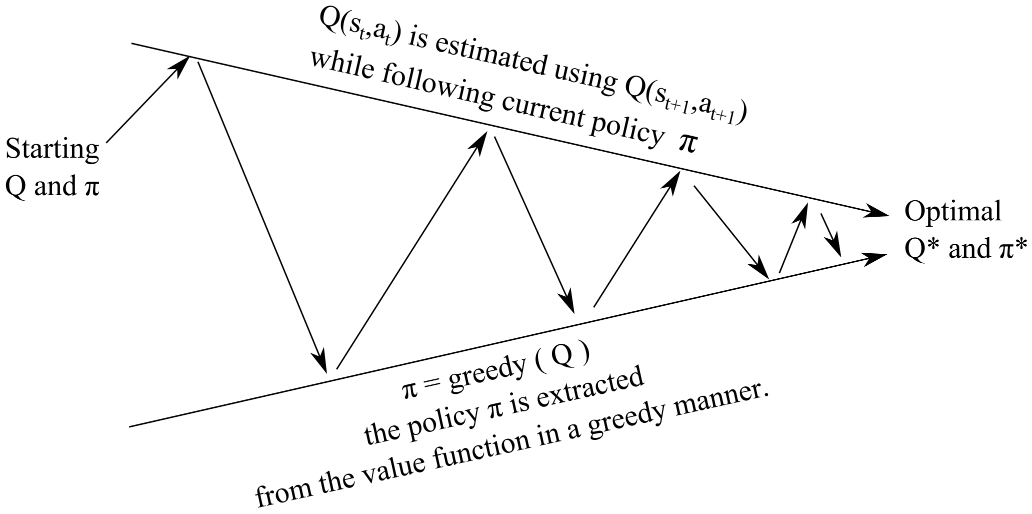

Q(s_t,a_t) is estimated using Q(s_{t+1},a_{t+1})
while following current policy π
Starting
Q and π
Optimal
Q* and π*
π = greedy (Q)
the policy π is extracted
from the value function in a greedy manner.

Figure 2.2: On-policy control methodology for updating Q^{π} while following the policy π.

The initial state s_t transitions to the state s_{t+1} by the action a_t, yielding the reward value r_{t+1}. Once in the state s_{t+1} the next action a_{t+1} is selected. The update to $Q(s_t, a_t)$, after every transition from a non-terminal state, is defined as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right], \tag{2.7}$$

where γ is the amount to discount the return and α is the learning rate. The update to $Q(s_t, a_t)$ uses a quintuple of events $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_{t+1}, \mathbf{s}_{t+1}, \mathbf{a}_{t+1})$, consisting of every element that makes up the transition from one state-action pair to the next. This quintuple gives rise to the name Sarsa [40]. Pseudo-code for the Sarsa control algorithm can be seen in Algorithm 2.1.

Algorithm 2.2 Pseudo-code for the off-policy TD control algorithm, Q-Learning.

```
Initialize $Q^{\pi}(s,a)$ for all $s$ and $a$
For each episode
    Initialize $s$
    For each step of the episode
        Choose $a$ from $s$ using derived policy from Q (e.g. $\varepsilon - greedy$)
        Take action $a$, observe $r$ and $s'$
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[ r + \gamma \underset{a'}{max} Q(s',a') - Q(s,a) \right]$
        $s \leftarrow s'$
    end
end
```

2.7.3 Off-Policy Control: Q-Learning

Off-policy TD control (Q-learning), directly estimates the action-value without following the policy. Q-learning is one of the most important breakthroughs in reinforcement learning as it separates the action-value function Q^{π} from the policy π, see Figure 2.3.

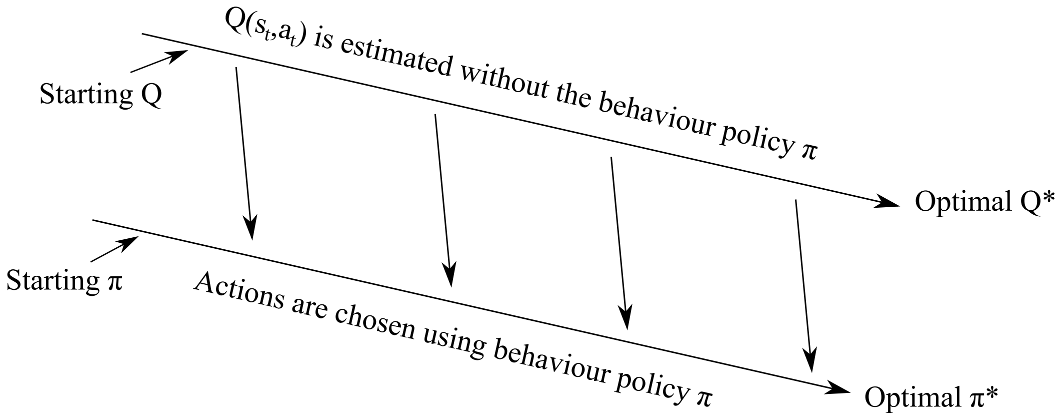

Q(s_t,a_t) is estimated without the behaviour policy π
Starting Q
Optimal Q*
Starting π
Actions are chosen using behaviour policy π
Optimal π*

Figure 2.3: Off-policy control methodology for updating Q^{π} without following policy π.

The initial state s_t transitions to the state s_{t+1} by the action a_t, yielding the reward value r_{t+1}. Once in the state s_{t+1} the update to $Q(s_t,a_t)$, after every transition from a non-terminal state, is defined as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \underset{a}{max} Q(s_{t+1},a) - Q(s_t,a_t) \right] \tag{2.8}$$

where the action with the maximum value is selected as the potential next action a. Pseudo-code for off-policy TD control can be seen in Algorithm 2.2 [40]. Note that the action that is used to update $Q(s_t,a_t)$ is not necessarily the next action to be taken as the update may change all value estimates.

2.8 Eligibility Traces

Eligibility traces are used to improve learning time by enabling past actions to benefit from current reward, allowing for sequences of actions to be learned. This is particularly important for applications in which key behaviours are combinations of basic actions. Eligibility traces can modify most TD methods to produce more generalized and efficient learning methods [40].

2.8.1 Theoretical Basis

From a theoretical stand-point, eligibility traces bridge the gap between TD methods and Monte Carlo methods, unifying both concepts. If more than one step is used to calculate the return, TD methods begin to resemble Monte Carlo methods, that wait until the end of the episode to calculate the return. Backup diagrams are used to illustrate these concepts, where the large circles are states and the small circles are rewards. These backup diagrams are used extensively in [40] as a simple and powerful illustrative tool for understanding RL algorithms. Figure 2.4 shows the backup diagrams for a range of methods between a simple one-step TD backup, and a full termination backup as used in Monte Carlo methods. The arrows are added to show when the backup occurs and to what state the backup applies. The *n-step* return at time t is defined as

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}), \tag{2.9}$$

where the return is truncated after n steps with an added estimate of the value of the n^{th} next state.

Backups can be done with any average of *n-step* returns, given that the weights on the individual returns are positive and sum to 1. For example, a *complex backup* can be done towards a return that is half of a *two-step* return and half of a *three-step* return, given that both halves are positive numbers that sum to 1. *Complex backup* diagrams include backup diagrams for each weighted backup, with a horizontal line above them and the weight fractions below. TD using eligibility traces, TD(λ), can be thought of as a particular way of averaging *n-step* backups, seen in Figure 2.5.

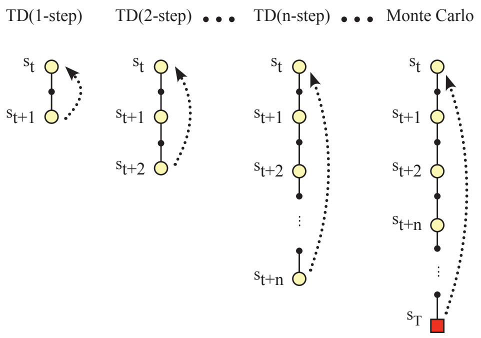


Figure 2.4: Backup diagrams of TD methods ranging from one-step backups to the full termination backups of Monte Carlo methods, where T is the final time step in the episode [40].

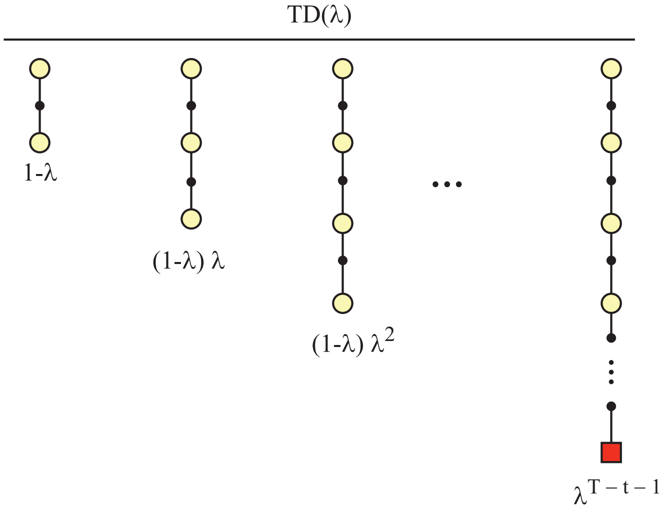


Figure 2.5: Backup diagrams of TD(λ). When $\lambda = 0$, the backup is reduced to TD(1-step) whereas when $\lambda = 1$, the backup becomes the same as Monte Carlo [40].

2.8.2 Algorithmic Implementation

From an algorithmic perspective, eligibility traces keep temporary information about the occurrence of events, such as visiting a state or performing an action. When a TD error occurs, only the relevant states are assigned blame or credit. Eligibility traces provide a basic mechanism for further temporal credit assignment [40].

Eligibility traces require that an additional variable be kept in memory for each state-action pair. This extra memory does not need to be saved and loaded for reuse. The trace value indicates how recent the state-action pair was encountered. Rewards can be assigned to past state-action values according to how recently they were updated. At each step, the trace decays for all previous state-action pairs, while the trace for the current state-action pair is incremented by 1, see Equation 2.10.

$$e_t(s,a) = \begin{cases} e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases} \quad \text{for all } s,a \tag{2.10}$$

The variable λ is the trace-decay parameter and γ is the discount parameter [40]. Notice, when $\gamma = 0$, any value for λ will produce the same results. The TD error (δ) is calculated using the difference between the current action value $Q_t(s_t, a_t)$ and the decayed estimate of the new action value, $Q_t(s_{t+1}, a_{t+1})$, and adding the current reward value, see Equation 2.11 [40].

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \tag{2.11}$$

Algorithm 2.3 Sarsa with eligibility traces, Sarsa(λ).

```
Initialize $Q(s,a)$ and $e(s,a)$ for all $s$ and $a$
For each episode
    Initialize $s, a$
    Repeat for each episode
        Take action $a$, observe $r$ and $s$'
        Choose next action $a'$ from $s'$ using derived policy (e.g. $\varepsilon - greedy$)
        $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        For all $s, a$
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            $e(s,a) \leftarrow \gamma\lambda e(s,a)$
        end
        $s \leftarrow s'$
        $a \leftarrow a'$
    end
end
```

The updated action value $Q_{t+1}(s_t, a_t)$, is calculated by adding the old estimate $Q_t(s_t, a_t)$ to a fraction of the error that was calculated in Equation 2.11, see Equation 2.12.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\delta_t e_t(s_t, a_t), \quad \text{for all } s \text{ and } a \tag{2.12}$$

2.8.3 Sarsa(λ)

The Sarsa algorithm can be extended with eligibility traces, resulting in the algorithm Sarsa(λ), see Algorithm 2.3 [22, 40]. When function approximation methods are used, the advantages of using eligibility traces often decrease, and as a result, eligibility traces are mostly used with a tabular function approximation approach [40].

2.8.4 Q(λ)

There are a few different ways with which to incorporate eligibility traces into the Q-learning algorithm, however only Watkins's method is described below. Because the policy that is learned is not the policy that is used to select actions, special care is taken when calculating *n-step* returns. Pseudo-code for the Q(λ) algorithm can be seen in Algorithm 2.4.

2.9 Generalization

In most of the available literature, RL is applied to problems with small discrete state-spaces. Making an effort to minimize the size of the state space permits the RL algorithm to examine the space more easily. While this approach is sufficient for most simpler applications, it is too restrictive in many real-world applications, especially in video game AI.

Algorithm 2.4 Pseudo-code for Q-learning with eligibility traces using Watkin's Q(λ) algorithm.

```
Initialize $Q(s,a)$ and $e(s,a) = 0$ for all $s$ and $a$
For each episode
    Initialize $s$, $a$
    Repeat for each episode
        Take action $a$, observe $r$ and $s$'
        Choose $a'$ from $s'$ using derived policy from Q (e.g. $\varepsilon - greedy$)
        $a^* \leftarrow max_b Q(s', b)$ (if $a'$ ties for max, then $a^* \leftarrow a'$)
        $\delta \leftarrow r + \gamma Q(s',\, a^*) - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        For all $s, a$
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                else $e(s,a) \leftarrow 0$
        end
        $s \leftarrow s'$
        $a \leftarrow a'$
    end
end
```

Function approximation is required when extending RL to a high-dimensional state-space. The information learned by the system must be generalized to other similar states thereby reducing the amount of information that needs to be collected in order to find a near-optimal policy. Generalization is particularly important for continuous state-spaces as the number of states is realistically infinite and the probability of returning to exactly the same state is very small[40].

Luckily, the concept of generalization and function approximation has already been extensively explored. These function approximation methods use samples of input and output pairs to learn an approximation of an unknown function. Using function approximation with RL would use value function updates as the input and output samples needed to train the value function approximation. The state of the game is represented more compactly with a feature vector that may include continuous or discrete information. This feature vector is used as the state representation when using function approximation methods [40].

2.10 Modelling and Planning

Traditionally, reinforcement learning has been solely considered a technique for learning, but more recently, it is being used in effective planning. When using RL for planning, the agent must keep a model of the environment, to predict how the environment will react to any given action. Given a state and an action, the model must predict the next state and the next reward.

Planning and learning are very similar; learning uses real experience generated by the environment, while planning uses simulated experience generated by a model. Online planning introduces interesting interactions between planning and learning. New interactions with the environment are

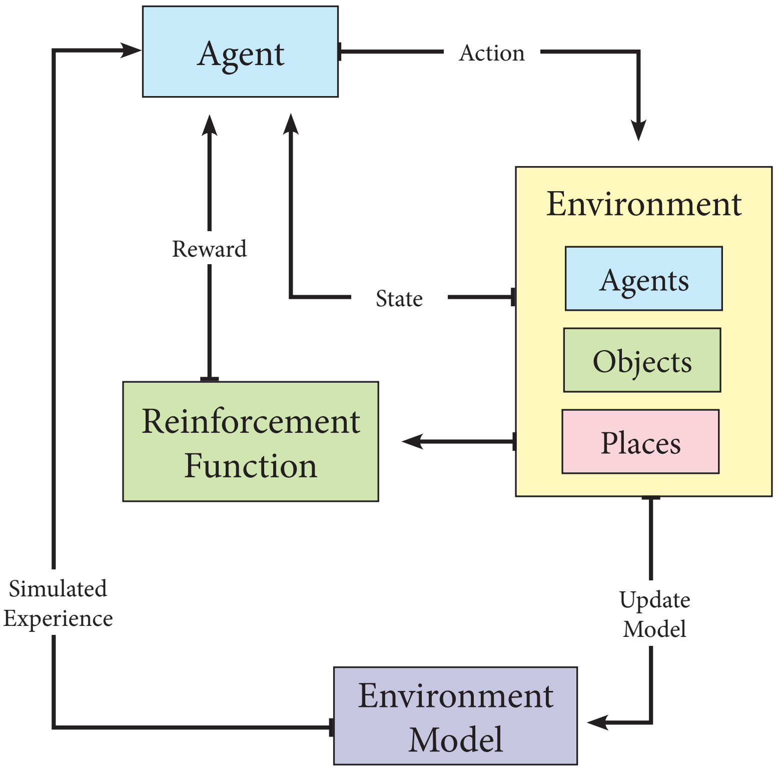

Agent
Action
Environment
Agents
Objects
Places
Reward
State
Reinforcement Function
Simulated Experience
Update Model
Environment Model

Figure 2.6: Agent-Environment-Model Interaction in the DynaQ Architecture.

used to alter the model thereby affecting the planning.

Planning and learning can be computationally expensive and as a result, computational resources may need to be divided between them. The most intuitive approach is for planning and learning to occur in parallel, utilizing shared resources between both processes [40].

The DynaQ method is applied to Watkin's Q(λ) algorithm in Algorithm 2.5, where s_{sim} and a_{sim} are state-action pairs that are randomly selected from previous experience, s'_{sim} and r_{sim} are the simulated next state and next reward, and a'_{sim} is the simulated next action.

2.11 Hierarchical RL

It is common practice within psychology and neuroscience to explore the applicability of ideas from a machine learning perspective. Computational reinforcement learning has had a profound and prolonged impact on the psychological and neuroscience research communities [6]. The initial impact of RL techniques helped to interpret certain brain activities by providing a similar framework. In return, advancements in RL-based techniques, particularly the distinction between habitual and goal-directed behaviours, were motivated by the aforementioned psychological advances.

Algorithm 2.5 Pseudo-code for Dyna-Q learning and planning algorithm based on Watkin's Q(λ) algorithm.

```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \epsilon S$ and $a \epsilon A(s)$
For each episode
    Initialize $s$, $a$
    Repeat for each episode
        Take action $a$, observe $r$ and $s$'
        Choose $a'$ from $s'$ using derived policy from Q (e.g. $\varepsilon - greedy$)
        $a^* \leftarrow max_b Q(s',b)$ (if $a'$ ties for max, then $a^* \leftarrow a'$)
        $\delta \leftarrow r + \gamma Q(s',\ a^*) - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        For all $s, a$
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
            If $a' = a^*$then $e(s,a) \leftarrow \gamma \lambda e(s,a)$
                else $e(s,a) \leftarrow 0$
        end
        $Model(s,a) \leftarrow r, s'$
        for $i = 1$ to $pS$
            Get random state $s_{sim}$, and action $a_{sim}$ from previous experience
            Get $s'_{sim}$ and $r_{sim}$ from $Model(s_{sim}, a_{sim})$
            $Q(s_{sim}, a_{sim}) \leftarrow Q(s_{sim}, a_{sim}) + \alpha \left[ r_{sim} + \gamma \max_{a'_{sim}} Q(s'_{sim}, a'_{sim}) - Q(s_{sim}, a_{sim}) \right]$
        end
        $s \leftarrow s'$
        $a \leftarrow a'$
    end
end
```

The development of RL within computer science focuses on factors that limit its applicability. In particular, RL methods are unable to cope with large state-action domains, called the scaling problem. When the space of possible actions or the space of possible states is too large, the RL system cannot cope [6].

The scaling problem is particularly relevant in interactive video games, with continuous complex environments. Temporal abstraction is a particular approach to the scaling problem that groups together interrelated actions as a single higher-level action. New representations are abstracted across sequences of potentially variable low-level behaviours.

To illustrate the importance and the need for complex sequences of behaviours, imagine the various actions that might be required when wanting to turn on the TV. One possible sequence of actions would be to pickup the remote, point the remote towards the TV, and then press the power button. RL techniques that use temporal abstraction often assume that options can be assembled into high-level skills in a hierarchical manner. The option for *turn on TV* might be used to form the option *watch Movie* or *play XBOX*.

RL techniques that use hierarchical temporal abstraction are called hierarchical reinforcement learning (HRL). HRL addresses the issue of how reusable sets of skills can emerge through learning, an important topic for behaviour research [6]. This paper follows [41] where sequences of actions are called *options*. Other HRL paper exists with similar frameworks [29].

2.11.1 Scaling Problem

RL agents learn adaptive policies by trial and error interactions with their environment. As the state-action space increases, more time is needed to sample all possible actions in all possible states. The relationship between training time and state-action space size can be defined as a positive acceleration function. As the problem size increases, standard (flat) RL becomes impractical [6, 40].

The state space can be minimized through elimination of irrelevant state distinctions (state abstraction). The agent is given a minimal state representation that still holds all required information needed for learning to occur [40]. State abstraction will not help the scaling problem if there exists a very large number of actions. HRL solves the scaling problem not through state abstraction, but through action abstractions. Actions are temporally abstracted and grouped together to form an option that consists of a sequence of low-level actions or even other options. These options rather than specifying individual actions to execute, specify a whole policy that should be followed [6].

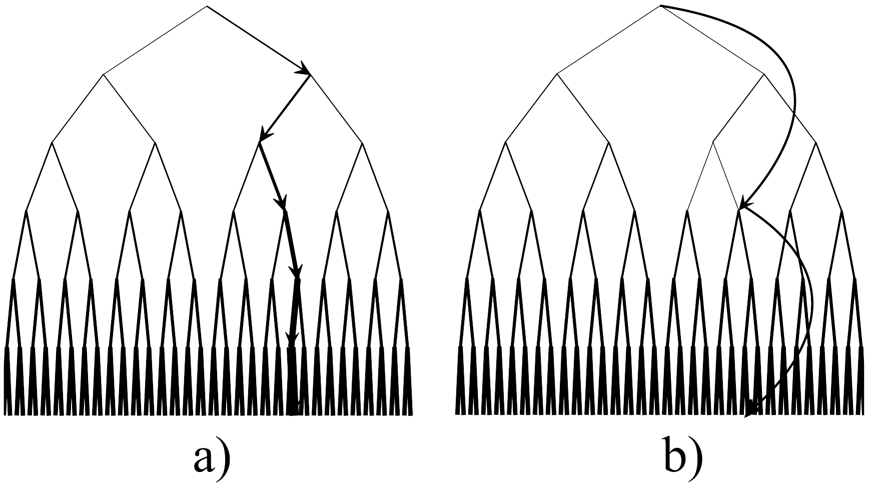


Figure 2.7: Options replace a series of low-level actions. a) A search tree outlining the task of searching for a series of actions that result in a favorable state. b) A search tree using options instead of actions. The task of searching through the search space is simplified when groups of actions are replaced with an option. [6]

Temporally abstracted actions (options) can alleviate the problems present in the scaling problem, by introducing structure into the exploration process. Options are used to guide exploration down partial action paths in the search tree. This potentially allows for earlier discovery of high-value sequences of actions (traversal). Options effectively reduce the size of the search space making the agent learn optimal policies faster than when using low-level actions. Options also enable the agent to use past experiences more effectively [6].

2.11.2 Incorporating Options

An option can be seen as a *mini-policy*, defined by starting conditions, a policy, and a termination function. The initiation set determines the states in which the option will be available. The policy determines the closed-loop behaviour to follow, mapping states to actions or even *other options*. The termination function determines what set of states will allow the option to terminate. The important fact, is that options can contain other options, allowing the agent to build complex high-level skills[6].

In a traditional RL policy using Softmax action selection, the actions are weighted depending on the suitability of performing the action in a certain state. Just like actions, options also have associated weights. If an option is selected, further actions or options are selected depending on the current option's policy. The option policy continues to be followed until the option terminates. Only when the option terminates, is the prediction for the entire option calculated[6].

Option-specific policies must first be learned, in order to be effective. Options are defined in terms of sub-goal states. When the current option being followed reaches its desired sub-goal state, all actions leading up to the state are reinforced. This reinforcement is different than external rewards and is therefore attaining a sub-goal results in *pseudo-reward*. The HRL must maintain option specific value functions that predict the cumulative reward and *pseudo-reward* that will be received. Option-specific policies are learned just as normal policies are learned. At each step of the option execution, a prediction error is calculated and used to update the option's value function. Through repeatedly updating the option's value function, the policy is refined to result in behaviour towards the desired sub-goal [6].

2.11.3 Option Discovery

Options can be thought of as a means to achieve sub-goals. One such sub-goal could be the desire to leave a room. These options could be perceived through evolutionary methods, whereby the options are genetically specified and are shaped across generations through natural selection. Some action sequences in animals, like animal grooming sequences, are possibly genetically specified. These genetically specified action sequences, act as built-in options. Evolution plays an important role in building the basis for animal (including human) behaviour [6].

In addition to instinctual behaviours, humans learn more complex behaviour routines through learning. Options can be formed within a development period in the absence of external goals. The agent learns from intrinsic rewards that are generated by the occurrence of unexpected salient events. The unexpected occurrence of salient events while exploring will trigger an effort to make the event reoccur, thus learning options with these events as their sub-goals. This type of sub-goal discovery can lead to iterative development of hierarchies of skills. Psychology supports this approach, as it suggests that human behaviour is motivated by either exploration or mastery that is independent of extrinsic goals [6].

According to psychology, unexpected events are only one way to trigger intrinsic rewards. An individual's social environment can potentially be another source of sub-goals. Humans will spontaneously infer goals and sub-goals from watching the behaviour of others. Options can be taught deliberately by parents or school teachers, or options can be learned indirectly through observation of behaviour from others. The process of raising a child is done through deliberately transferring knowledge in the form of useful action sequences. Teaching a child what happens when the light switch is pressed or showing them how blocks can be stacked one on top of the other [6].

There are typically two types of actions that a human can take, habitual actions that are directly influenced by a given stimuli, or actions that are based on the results of planning. Habits are based on established stimulus-response links and are learned without a model of the environment relying on cached action-values. Model-based RL provides the means to look ahead based on an internal model of the environment that relates actions to their likely results. HRL can be extended with models whereby each option is associated with an option-model, that indicates the likely outcome, reward and the amount of time needed to execute the option. With option-models, the agent can *look ahead*, enabling the agent to evaluate potential courses of action. The search process can skip-over large sequences of primitive actions thereby reducing the search tree [6].

With HRL, it is difficult to recognize overlap between various options such as spreading jam on toast and spreading icing on a cake. Execution of sub-tasks is very context sensitive, where the sequence of actions needed is dependent on the initial starting state [6].

2.12 Summary

Reinforcement learning is a useful tool for online agent-learning through interactions with an environment, including objects, places, and other agents. RL is more useful than ever before, with temporal credit assignment and temporally abstracted action sequences. With eligibility traces, credit for current rewards is distributed to previous state-action pairs, updating their values for better or worse. HRL allows for sequences of actions to be grouped into options, which they themselves can be combined into even more complicated options.

With the basics of RL established, the following chapter will explain the psychological aspects of a human's personality, and what person model most accurately predicts behaviour.

Chapter 3

Realistic Agents

Large story driven game genres, such as role playing games, and action-adventure games, consist of many individual characters that are important to the overall player experience. Game characters contribute to the player's feeling of being so immersed in the virtual game world that they are unaware it is artificial. These types of game genres would benefit from the development of realistic NPCs. Because of the interactivity of video games, a game agent must behave and interact with a player realistically, without their actions conflicting with their apparent personality. Even if nothing is explicitly revealed about a game character's personality, or motives, a player builds their own model of a character through observation. This model drives player's expectations of the NPC's actions, during game play. While this model will not be wholly accurate, an NPC that conflicts with too many expectations would result in disbelief on behalf of the player. This type of disbelief is to be avoided at all costs, because interferes with player immersion.

Many approaches have been taken in the direction of believable game agents. Some techniques specifically target key aspects of believability, including social behaviour [37, 16, 3, 30], personality traits [5, 32, 9] and even emotional responses [21]. Other works explore perception, focusing on how the NPC can detect human player intentions [10], and how to project emotion and intention to the game player [38].

This chapter will cover the basics of personality psychology, and desire theory, along with recent related advances in reinforcement learning research.

3.1 Personality

The mission behind personality psychology theory, is to use mechanisms to explain a person's thought patterns, emotions, and behaviours. The most notable problem in personality psychology, is the fragmented nature of its many solutions. Integrating the specialized research topics, like developmental, social, cognitive, and biological, into a unifying understanding of a person, is a particularly difficult challenge [13].

The historical basis of personality psychology include many different approaches, resulting in different paradigms like psychoanalytic, trait, behaviourist, and humanistic. Through developments in these main fields, other new paradigms were created. The trait and behaviourist paradigms evolved into social-cognitive and biological approaches. There is also evolutionary psychology, a completely new paradigm [13].

Regardless of the paradigm followed, in order to empirically study personality, three elements are key: person, situation, and behaviour [13]. The personality triad is imbalanced, because proper attention has not been given to the situation variable. While the person variable has been explored by all the various personality paradigms, the situation variable has not received the same attention [13]. While this is a problem in real-life situations, it should not be a problem in video games. In video games, the situation consists of sensory representation of the game world, as it relates to the agent in question. The problem becomes how to define the person, or in this case, game agent. This assumes that all pertinent information is available.

All the research into personality dynamics, while diverse, can be grouped roughly into two main categories. The first studies the statistical structure of personality across large groups of people, and is also called inter-individual personality structure. The second group, intra-individual personality structure, studies the dynamics of personality within a person, to explain the processing systems responsible for their personality [32].

3.2 Desire Theory

Voluntary behaviour is influenced most heavily by a person's motives. The motives themselves affect a person's perception, cognition, emotion, and behaviour. Motives are said to be either *intrinsic,* or *extrinsic* depending on their purpose. Psychologists have defined *intrinsic motivation* as being moved to do a particular behaviour, because it is inherently enjoyable. *Extrinsic motivation* is defined as wanting to do a particular action, because of a specifically desired outcome.

Most people have heard of the term "ends vs. the means", which divides a person's motives based on the purpose of performing the behaviour [1]. An end motive is when an individual will perform behaviour for no apparent reason, other than it is what they desire to do. A motive is considered a means, if it is only needed to full-fill another end motive. A student may be motivated to get high grades in order to please their parents. In this example, the motive to get high grades is a means, and the end motive is to please their parents. A *behaviour chain* is a series of means that ultimately, by definition, finishes with an end [34].

There are infinite ways in which a person can seek to accomplish an end. The number of possible means is also infinite, constrained only by the imagination. In contrast to means, ends are finite by human nature. The classification of ends is important, given that they reveal an individual's ultimate goals.

The classification of end goals is split into two different perspectives. Multifaceted theory postulates that end goals are unrelated to each other, even genetically distinct with different evolutionary paths. Unitary theory, in contrast, proposes that end goals can be roughly grouped into a manageable number of categories, based on common characteristics [34].

The separation of end goals, into drives and intrinsic motives (IMs), is currently popular with psychologists. Drive theory defines reward as reducing a state of deprivation. When an agent's hunger drive is in a state of deprivation, food becomes a powerful reward, increasing motivation to learn actions that produce food. In [18], Hull identifies four types of drives: hunger, thirst, sex, and escape from pain. A large disadvantage to drive theory's validity, is the fact that it does not explain exploration (curiosity), autonomy, and play.

Unitary IM theory is an alternative to drive theory that explains the motives that drive theory could not. It theorizes that competence is the origin of curiosity, and autonomy. While there may be a correlation, it is unrealistic to say that if a person has an above average amount of curiosity, then they must also have an above average autonomy[34].

The multifaceted theory of end goals has been explored by many different angles throughout the years. Very early on, Aristotle defined 12 end motives: confidence, pleasure, saving, magnificence, honour, ambition, patience, sincerity, conversation, social contact, modesty, and righteousness. Evolution may also play an important role in multifaceted theory. Different IMs may originate from distinct survival needs, embedded in different genes. The need to build nests for survival from weather or predators, suggests the evolutionary motivation efficacy. Autonomy being the desire for freedom, motivates an animal to leave the *nest* and *search* for food in a large area[34].

Reiss's theory of 16 basic desires is an important multifaceted model of IM, outlined in Tables 3.2 and 3.1. Studies have shown that the theory can be used to describe religious beliefs [33], athleticism [36], and lack of scholastic achievement [34]. For example, religious motivation was found to be motivated by above average honour and family, and below average vengeance and independence.

3.3 Emotion RL

Emotions can influence human behaviour by altering our perception of people and events. Beyond altering perception, the concept of feelings can be used to drive reinforcement learning, towards quicker learning and better policies. The agent learns behaviours that make it feel *good,* and avoids behaviours that make it feel *bad.* In this context, specific goals can lead to a *good* feeling, to make the agent want to reach the goal through intrinsically-motivated reward [21].

Following appraisal theory, emotional reactions can be characterized by an evaluation of the stimulus with respect to a number of dimensions, mostly relating to the current goal. These appraisals

Hypothesis 1	Each of the basic desires is a trait motive.
Hypothesis 2	The 16 basic desires motivate animals as well as people (except maybe idealism and acceptance).
Hypothesis 3	The 16 basic desires are considered genetically distinct with different evolutionary histories.
Hypothesis 4	Satisfying a basic desire produces an intrinsically valued feeling of joy. Each desire produces a different feeling of joy.
Hypothesis 5	Individuals prioritize each desire differently.
Hypothesis 6	Each basic desire is a continuous range between opposite values.
Hypothesis 7	What is motivating are the discrepancies between the amount of an intrinsic satisfier that is *desired* and the amount that was recently *experienced*.
Hypothesis 8	Basic desires influence: attention, cognition, feelings and behaviour into a coherent action.

Table 3.1: Hypotheses of Reiss's theory of 16 basic desires [34].

Name	Motive	Intrinsic Feeling
Power	Desire to influence	Efficacy
Curiosity	Desire for knowledge	Wonder
Independence	Desire to be autonomous	Freedom
Status	Desire for social standing including desire for attention	Self-importance
Social contact	Desire for peer companionship (desire to play)	Fun
Vengeance	Desire to get even (including desire to compete to win)	Vindication
Honour	Desire to obey a traditional moral code	Loyalty
Idealism	Desire to improve society (including altruism and justice)	Compassion
Physical exercise	Desire to exercise muscles	Vitality
Romance	Desire for sex (including courting)	Lust
Family	Desire to raise own children	Love
Order	Desire to organize (including desire for ritual)	Stability
Eating	Desire to eat	Satiation
Acceptance	Desire for approval	Self-confidence
Tranquillity	Desire to avoid anxiety, fear	Safe, relaxed
Saving	Desire to collect, value of frugality	Ownership

Table 3.2: Motives in Reiss's Theory of 16 Basic Desires [34].

Suddenness	Amount that the stimulus is abrupt or highly intense.
Unpredictability	Amount the stimulus could not have been predicted.
Intrinsic Pleasantness	How pleasant the stimulus is, regardless of the goal?
Relevance	How important the stimuli is compared to the goal?
Causal Agent	Stimulus was caused by who?
Causal motive	What was the motivation behind the agent that caused the stimulus?
Outcome Probability	Probability of the stimulus actually occurring.
Discrepancy from Expectation	Compare what was expected, to the actual stimulus.
Conduciveness	Is the stimulus good, or bad for the goal? To what extent?
Control	Is the stimulus easily influenced by anyone?
Power	Is the stimulus easily influenced by the agent in question?

Table 3.3: Reduced list of appraisals that categorize emotional reactions.

are described by Scherer [39], but are reduced to a sub-set of appraisals by [21] described in Table 3.3.

Emotions are fleeting and often change in response to new stimuli. In Marinier and Lard's paper [21], mood is defined as a "moving history of emotion". As appraisals determine an emotion, mood's value changes incrementally over time towards the most recent emotion. If an agent were to be ruled by its emotions, its interpretation of a situation would be skewed. Mood provides the emotional history needed for more levelled interpretations. The combination of mood and emotion is described as *feeling*. Though *feeling* is represented by a set a appraisal values similar to emotion, it can be measured by a single valenced intensity value that is dependent on conduciveness.

With emotion driven RL, the appraisal process would be considered the critic in a typical RL system. The feeling's intensity value is the reward signal for the emotional reinforcement learning agent [21]. Rules generate the appraisal values by matching patterns in perception and the internal state. Appraisal values change with the situation, altering the current emotion, mood, and ultimately, the feeling state of the agent.

This approach to RL has shown to produce improved learning in limited test cases in a maze type environment [21].

3.4 Motivated RL

Intrinsic motivations allow autonomous organisms to explore their environments without explicit rewards, promoting play and exploration. These curiosity driven activities favour the development of widespread competence, rather than specialized behaviours. This directly contradicts the purpose of most machine learning techniques, whose systems are designed to respond to specific problems [8, 4]. General competence can be used to solve a wide range of specific problems, making the approach more versatile than specific goal learning.

Despite the wide-spread applicability, and relative power of machine learning techniques, they are typically only applied to specific isolated problems. The type of algorithm must be carefully chosen, and the parameters must be hand-tuned, to provide the necessary learning capabilities. Also, effective training sets must be constructed, or acquired, to find a good solution to the problem.

The solution may lie in a developmental approach, based on cognition, neuroscience, and psychology. An agent would experience a development period, where it learns a set of reusable high-level skills. This period of development is experienced in humans, when children learn basic skills through exploration, and playing. These complex skills are later used to overcome larger challenges. What we learn through intrinsically motivated behaviour is essential to our ability to solve a wide range of problems [4].

The traditional reinforcement learning technique has been extended with intrinsic motivations to promote autonomous development of skill hierarchies, called motivated reinforcement learning (MRL) [25, 23, 26, 27, 24, 4, 8]. MRL uses a motivation function based on interest, to calculate the reinforcement function. The agent calculates the differences, between past and present states, to compute the reward signal that is responsible for directing learning [25].

The motivation function is not dependent on the domain, instead it is based on the concept of interest to calculate the intrinsic motivation signal. The skills the agent develops depend on the environment and its experiences. A single agent model can be applied to different NPCs, with different agents learning different skills. The skills that are developed over time and can adapt to changes in the environment [25]. High-level skills are learned during initial competence training, making specific goal learning quicker. To solve specific goals, the agent selects from high-level skills, rather than low-level actions [4].

The motivational profiles discussed in [27] show how learning sets of skills is affected by broad motivational tendencies for achievement, motivation, and power. During a risk-taking test, power motivated agents would select goals with high-incentives, whereas achievement motivated agents would select goals with regards to higher difficulty.

3.5 Social Believability

Humans in general are social animals where our relationships, culture, and values, all play important roles in how we interact with others. The ways in which social factors influence an individual's thoughts and behaviours are an important part of personality psychology. Work has been done to incorporate the psycho-socially influenced behaviour into an NPC, with the goal believability[15, 2, 3].

In work by Bailey et al.[3], a unique role based approach was used for social modeling in addition to a multi-model approach to personality and emotion. A goal-oriented utility-based planning system was used with an extension of utility-driven psychosocial behaviour and appraisal theory. The NPC will encounter an event, appraise it, cope with the consequences, select a goal in response, formulate a plan, and finally, select an action. Notice that learning is not a part of this method.

During coping, an NPC must update its internal state in response to the event. In this case, this adjustment would reflect the NPC's physical and psychosocial state, which includes mood and emotional memory. Goal selection is not only based on the NPC's surroundings in the game world, but also its current social context. The goals themselves are mostly selected from the NPC's associated roles, which include values, goals, emotional traits, actions, and personality traits.

The results of this approach, are NPCs with personality, emotion and social awareness, capable of creating engaging and immersive game play for the player [3].

3.6 Summary

In every day life, it comes as a shock when a person acts outside the realm of their personality. Once the shock subsides, the new information is used to help us refine our model of that person. In video games, if this type of shocking behaviour occurs often, players are puzzled, and often think of these behaviours as *glitches*. This type of distraction interrupts the carefully thought out narrative, and breaks the player's immersion into the game world. To keep action selection believable, the NPCs must make decisions that are in line with what players expect.

To reliably predict an individual's behaviour, both the situation, and the model of the person, must be accurate. In video games, the situation is not much of a problem, given that the world is already entirely represented by the computer. The model of a person, however, is more difficult. There are numerous different theories on what affects a model of a person, from personality traits, motivations, evolution, social network, and even biology. An individual's motivations and desires, likely have the most direct impact on their behaviours.

Emotion and motivated reinforcement learning use psychological principles to improve learning in a broad sense. Emotions are used to guide the learning process by encouraging the agent to take actions that make it feel good. Also, the introduction of motivated learning into hierarchical reinforcement learning helps to solve the problem of automatic option construction (sequences of actions). This use of motivation or emotion, while loosely influencing the action selections, does not provide specific direction for the agent with the purpose of realistic action selection. The goal of MRL and emotional RL, is for the agent to more quickly learn effective policies.

In contrast to emotional and motivated RL, the method discussed in this thesis uses intrinsic motivations to directly guide the reinforcement learning and action selection of the agent with the express purpose of believability. The focus is not on the efficiency of the RL method, but on how the agent's motivations should be incorporated into the situation, guiding the agent's action selection with believability in mind.

Chapter 4

Method

4.1 Overview

This chapter covers the theories behind this approach, and what concessions had to be made during its implementation.

4.2 Approach

In the literature, a personality triad defines a strong relationship between a person, a situation, and the resulting behaviour. To more easily solve this problem, the relationship is approximated by a mathematical operation. In the case of action selection policies, behaviour is unknown, and it is assumed that the parameters defining a person are fixed. In this case, the relationship between the personality triad variables is most accurately defined by

$$Behaviour = person(situation), \tag{4.1}$$

where the *behaviour* is the dependent variable, the *situation* is the independent variable, and *person* is a fixed predefined function. The *person* function, includes all necessary information of a person, and how that person responds to a situation. The problem now becomes a question of function approximation. What characteristics of a person must be included in the person function, to make the approximation as close as possible to the true function? Since the person function is not what is being learned, this problem cannot be solved using function approximation methods.

In real-life situations, the structural representation of a person is very complex. Models for understanding an individual's personality include traits, motives, and desires. These models can also depend on psychosocial, evolutionary, or biological factors. In a computer controlled video game, such complexity may not be required by the performing NPC. Selecting the strongest behaviour indicator for estimating the person function, allows for a simpler, less computationally intensive approach to realistic agents.

Rather than having motivation drive skill learning as in MRL, this implementation tracks the state of an agent's motivations, to learn actions that will benefit an agent's self-interest. The necessary minimum and maximum desirable motive values remain fixed, representing the *person* function. The current desire values are constantly changing, and should therefore be a part of the situation variable. The implemented system, while drawing inspiration from Reiss's theory of 16 basic desires [35], is not constrained by any particular motivational model. In this way, any motivation (mean or end) can be created to suit the developers needs.

The actions themselves directly affect any number of the agents motivations, and must be specifically defined in the action definition. The agent will learn to perform actions that benefit itself, according to its internal motivation values and thresholds. Each agent will have distinct thresholds for individual motivations. For example, if an agent's social interaction falls below its minimum threshold, the agent should learn to choose actions that will increase its social interaction. Conversely, if the agent's social interaction value is raised above the maximum threshold, the agent should seek the opposite, isolation, to return within acceptable limits. Setting different thresholds for different motivations, and different agents, allows for a unique representation of a person.

4.2.1 Environment

The game world is defined through an XML file that is parsed at the beginning of the game. The game world is a collection of agents, objects and places. Places contain both agents and objects, while objects can only contain other objects (fridge contains food). Agents can be defined with any number of motives. All actions are defined globally, but an action instance is created for each agent at the beginning of the game. Actions include the information needed to affect changes to objects (position, ownership, erased, etc.), places (add an agent or object), or agents (position, gaining a possession, change in motive values, etc.). If an action includes changes to a motive that is not present in the current agent, the motive changes are ignored. While loading the game world, if the agent's actions have already been created in a previous game, with no changes to its motives, the learned action policies are simply loaded from memory.

4.2.2 Motives

A motive consists of a current value, $m_{current}$, a maximum desired value $m_{max} \leq 100$, and a minimum desired value $m_{min} \geq 0$, where $m_{min} < m_{max}$. The minimum and maximum desired values are also called thresholds, or bounds. A motive value will naturally decay at a linear rate over time, until an action is taken that increases or decreases the value. See Figure 4.1 for a graphical representation of a motive.

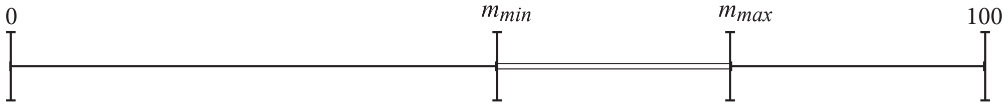


Figure 4.1: Finite Motive Scale.

4.2.3 Reinforcement Function

The rewards are a means to guide the learning process towards the goals of the developer. With incorrectly defined rewards, the learning system will not reach the desired goal. For this application, the overall goal is for the agent to have its motive values within threshold, as much as possible. The goal can be translated into rewards in many ways.

The simplest reinforcement function would assign a reward of $+1$ when all motives are within their thresholds, and a reward of -1 otherwise. This reinforcement function does not take into account the situations in which it is impossible to have all motives within thresholds. Even if it were always possible to achieve a perfect solution, the learning system must be robust enough to handle several motives, and actions that have no immediate impact. This simple reinforcement function would take much too long to train, as there are no guiding rewards aiding the learning process. The ideal reinforcement function should assign rewards that are weighted according to their importance, to help the learning process find the best possible policy.

For the reinforcement function used in this implementation, the reward r, depends on the distance between the current motive value, and its nearest threshold value, defined by d, see Figure 4.2.

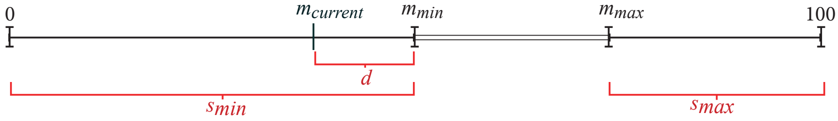


Figure 4.2: For a motive m, its current value is $m_{current}$, maximum threshold is m_{max}, minimum threshold is m_{min}, and the distance between $m_{current}$, and the nearest threshold is defined by d. The minimum scaling factor is $s_{min} = m_{max}$, and the maximum scaling factor is $s_{max} = 100 - m_{max}$.

For the cases where $m_{min} \leq m_{current} \leq m_{max}$, then the distance is $d = 0$. The distance between the minimum threshold m_{min} and 0, is defined as s_{min}, and the distance between the maximum threshold m_{max}, and 100, is defined as s_{max}. The reward associated with the motive m, r_m, is defined as

$$r_m = \begin{cases} (d/s_{min}) * mR & \text{if } |m_{current} - m_{max}| > |m_{current} - m_{min}| \\ (d/s_{max}) * mR & \text{if } |m_{current} - m_{max}| < |m_{current} - m_{min}| \end{cases}, \tag{4.2}$$

where mR is the reward scale value. As $m_{current}$ leaves the threshold, d increases, and the negative reward associated with the motive approaches mR. The value mR is the minimum possible reward that can be received by the agent, from one motive. The total reward for a state is determined by a summation of rewards associated with all motives, defined in Equation 4.3, where N is the total number of motives. In this way, every motive has a contribution to the overall reward, and addressing multiple motives results in greater reward (less negative). Because s_{min} does not have to be the same as s_{max}, the same d will produce different r_m values depending on whether $m_{current}$ is below m_{min}, or above m_{max}.

$$r = \sum_{m=0}^{m=N} r_m \tag{4.3}$$

4.2.4 Function Approximation

Approximating the Q value function is the essence of reinforcement learning. Implementation of RL in the literature is most often achieved through look-up tables. While using a tabular function approximation is simple and intuitive to understand, it often does not generalize to unfamiliar states (particularly with large state spaces) and cannot handle continuous state information. In this implementation, the state is represented by continuous and discrete variables. ANNs were chosen to approximate the Q and Model functions, because of their ability to handle both continuous and discrete input.

There are a separate Q-function and Model-function approximations, for each action. Having multiple action functions results in a simpler implementation, but it also prevents actions from benefiting from experience involving other actions. Every game step, the Q-function is updated only for the action in question. During the RL algorithm, the value function update $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$, is an update to the ANN Q-function approximation, using a value that is estimated using the same Q-function ANN.

Fast Artificial Neural Network library (FANN) was used for the implementation of ANNs. The activation function for the hidden layers and the output layer is sigmoid symmetric, with a span of $-1 < y < 1$ for any input. The desired training error for the network is -0.000001, the activation steepness for hidden neurons and output neurons is 0.5. The learning algorithm used is incremental.

Given that the rewards must be $r \leq 0$, the result of the Q function approximation should be $Q(s,a) \leq 0$ for any $s \epsilon S$ and $a \epsilon A(s)$. If the Q-function ANN approximation is initialized with random weights, the resulting approximation could be anywhere in the span of $-1 < y < 1$. For this reason, when an ANN is first created, it is initialized with a training set of 200 random inputs, all with the desired output of 0. The same initially trained ANN is used for all Q-function action approximations, enabling all action-value estimates to be identical, before learning begins. If initial action estimates are very different, more time is needed for the RL system to learn a good policy.

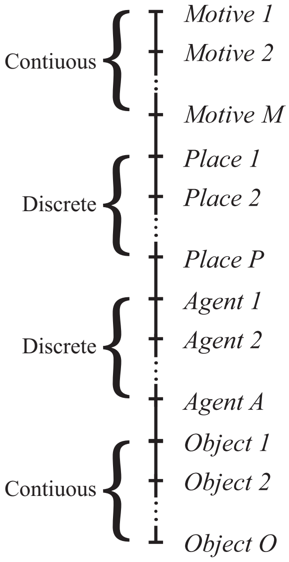


Figure 4.3: Features that are used as input into the Q-function approximation ANN, where M is the number of motives, P is the number of places in the game world, A is the number of agents in the game world excluding the current agent, and O is the total number of object types.

4.2.5 State Representation

For input to the RL algorithm, the motive values must be transformed into a floating point value, that is readable by the ANN function approximation. This mapping will directly affect how the agent will learn its policy. In this case, it was logical to directly map 0 to -1.0 and 100 to 1.0.

If the state of an agent consists only of floating point motive values, then the agent does not have knowledge of where it is, what other objects and agents are nearby, and what are the states of its possessions (does its fridge contain food?). For these reasons, other state variables were added to include this information.

An additional state variable is added for each possible place in the game world. The variable for the place in which the agent inhabits is set to 1, and set to 0 otherwise. Also, an additional state variable is added for every other game agent in the game-world. For example, the state variable associated with $agent_1$ is set to 1, when $agent_1$ resides in the same place as the current agent, and set to 0 otherwise.

A floating point state variable is added for each object that may contain a certain number of objects, and belongs to the agent in question. The floating point variable shows the state of the object, 1 being full, and 0 being empty. The state of the object variable is calculated using the capacity of the object, and the current number of containing objects. This variable was added for situations that require knowledge of the *fullness* of an object. For example, the agent should learn to go to the grocery store to buy food, when their fridge is empty.

4.3 Game Scenarios

For experimental purposes, several different game scenarios were designed to fully explore the potential and shortcomings of this approach. The first game, Game one, is a simplistic environment including only one agent, one motivation, one object, and one place. All actions include immediate consequences with no delayed rewards. This game is used to demonstrate how easily an agent can learn the basics. The second game, Game Two, increased the number of motivations, and also introduces an action (open fridge) that has delayed reward, where the value of opening the fridge is only found when the food is eaten. This game is used to demonstrate the ability to learn multiple motives, even with delayed rewards. The third game, Game Three, introduced the concept of inter-agent interaction, where an agent performs an action that affects another agent's motivations. The fourth game, Game Four, introduced an action with a 5-step delayed reward. This game is used to demonstrate how sequences of actions can be learned. The fifth and final game, Game Five, combined the concepts of inter-agent interaction, 5-step delayed reward, and multiple motive values, for a comprehensive learning challenge.

4.4 Evaluative Measures

Experimental results of this implementation were carried out in a game world with no graphical interface, and no user input. Quantitative results are obtained through observing and recording interactions between singular or multiple computer controlled game agents, in different environments. In this way, thousands of game variations are tested within minutes, allowing the exploration of multiple algorithms and parameters.

To fully test this approach would require a fully developed game, with multiple human users, and computer controlled game agents. A game of the genre role-playing game (RPG), action/adventure/RPG or even a full massively multiplayer online RPG (MMORPG) would be ideal. While these types of games would allow for a more complete set of testing data, access to code for these types of games is limited by the gaming industry. For this reason, a fully functional game would have to be developed for the purpose of testing and experimentation. Such a large scale game development project takes years for a team of developers, and therefore is not feasible to produce for the purpose of this thesis.

With the purpose of this work being a more believable game agent, qualitative analysis should include the degree to which this has been achieved. Since the formation of extensive test groups is not possible, a hybrid quantitative-qualitative analysis will be done in a variety of different scenarios. Various significant final policies are analyzed in terms of actions sequences, and interactions with other agents, with a real world perspective.

The ultimate goal of the agent is to stay as close as possible to motivational equilibrium, with all motivations within maximum and minimal thresholds. During a game, a number of statistics are gathered in regards to each individual game agent. The metric for direct evaluation is the sum of the rewards received throughout the game. As reward is the parameter with which reinforcement learning is enforced, this metric allows the most accurate evaluation of the learning capabilities of this approach. Though maximizing the total reward is the goal of the RL, the secondary purpose is to have the maximum number of game steps with $r = 0$ (all motives are within bounds). Both metrics are recorded as percentages (percent of total reward and percent of steps within bounds) allowing for direct comparison between two runs with different numbers of game steps.

Random variations in data are expected because of the randomly initialized weights for Q-function and M-function ANNs. Each repeated test begins with a different set of action value estimators (ANNs). This random variation is measured using the repeated test runs and graphically illustrated during the analysis of consistency in each Game scenario.

4.5 Testing Approach

Built into this implementation is the means to use different RL algorithms (Sarsa, Q-learning and Dyna-Q), with different action selection methods (ε-greedy and Softmax). The various game scenarios are tested with different combinations of methods, and various parameter values.

4.5.1 Parameters

With this implementation, there are a large number of parameters and algorithms from which to choose. Parameter and algorithm choices are outlined in Table 4.1. One of the purposes of the motivation based RL is to facilitate the development of realistic computer controlled characters. A set of variables must be found that will perform well in most situations, thereby not complicating the possible implementation of this method. For this reason, the purpose of extensive testing is to understand the impact of each variable, on the agent's learning. Given the large number of variables involved in the testing process, it is impossible to test every variable combination. Instead, an incremental approach was used to determine what variables produced the best widespread results. This manner of testing was repeated for each Game scenario, with the understanding that each game introduces a more complex learning task, potentially requiring certain parameters to be different. Understanding the parameters needed for each increasingly complex situation, enables the possible discovery of widespread optimal values.

<table>
<tr><td rowspan="3">Sarsa</td><td colspan="4">Action Selection</td><td>Reward</td><td colspan="4">Learning</td></tr>
<tr><td colspan="2">ε-greedy</td><td colspan="2"><i>Softmax</i></td><td rowspan="2">mR</td><td>α_s</td><td>α_e</td><td colspan="2">Q(s,a)</td></tr>
<tr><td>ε_s</td><td>ε_e</td><td>τ_s</td><td>τ_e</td><td>γ</td><td>λ</td><td>Hn</td><td>η</td></tr>
<tr><td rowspan="3">Q</td><td colspan="4">Action Selection</td><td>Reward</td><td colspan="4">Learning</td></tr>
<tr><td colspan="2">ε-greedy</td><td colspan="2"><i>Softmax</i></td><td rowspan="2">mR</td><td>α_s</td><td>α_e</td><td colspan="2">$Q(s,a)$</td></tr>
<tr><td>ε_s</td><td>ε_e</td><td>τ_s</td><td>τ_e</td><td>γ</td><td>λ</td><td>Hn</td><td>η</td></tr>
<tr><td rowspan="3">Dyna-Q</td><td colspan="4">Action Selection</td><td>Reward</td><td colspan="4">Learning</td><td colspan="3">Planning</td></tr>
<tr><td colspan="2">ε-greedy</td><td colspan="2"><i>Softmax</i></td><td rowspan="2">mR</td><td>α_s</td><td>α_e</td><td colspan="2">$Q(s,a)$</td><td rowspan="2">pS</td><td colspan="2">$Model(s,a)$</td></tr>
<tr><td>ε_s</td><td>ε_e</td><td>τ_s</td><td>τ_e</td><td>γ</td><td>λ</td><td>Hn</td><td>η</td><td>Hn</td><td>η</td></tr>
</table>

Table 4.1: Spilt up into the different RL algorithms, the table describes what parameters are necessary in each aspect of the system. The parameter mR defines the scaling factor of a motive's individual reward, Hn is an array that defines the number of hidden neurons in each hidden layer of an ANN, pS defines the number of planning steps during Dyna-Q, η defines the learning rate of an ANN, λ defines the decay-trace value, γ defines the discount value, ε_s defines the starting exploration rate, ε_e defines the ending exploration rate, τ_s defines the starting temperature value, τ_e defines the ending temperature value, α_s defines the starting RL learning rate, and finally, α_e defines the ending RL learning rate.

4.5.2 Exploration Rate

The exploration rate is used to balance how often an agent takes exploratory actions, rather than actions that are known to produce the best results. For video games waiting an extended amount of time for a policy to converge is not an option. The theoretical optimal exploration rate is of little use in a real-time application with time constraints. Also, reward functions with delayed rewards and non-stationary game worlds with constantly evolving characters necessitate more exploration due to changing or unclear action values. For simple learning tasks, a high exploration rate is not needed, as the best actions can be discovered rather quickly, with very little exploration. With more complicated tasks, a higher learning rate would allow for the agent to discover less obvious, but more beneficial actions. For this reason, it is also common to start the exploration rate at a higher value, incrementally decreasing it over time, as the agent learns its environment. Towards the end of the game, the agent should exploit much more than it explores. A moderate learning rate should be maintained to allow for the discovery of new solutions in a changing environment. The rate of exploration needed to learn a particular task is directly related to complexity, in terms of delayed reward, and number of objectives. For this reason, the degree of exploration needed is expected to increase as the Game scenarios become more complex.

4.5.3 Training and Testing

During training games, exploratory actions are an important part of learning. Without exploration, the optimal action selection policy might not be found. The current policy is obtained, at any time, by setting the exploration rate to 0. To clearly illustrate how training time influences the learning process, in some game scenarios, the agent goes through alternating states of training and testing. The game run $TrainN$ is the N^{th} training game of a particular testing configuration. The game run $TestN$ follows the policy obtained through the N^{th} training game run. There are 1000 game steps in each individual Game run (training or testing). For example, a training game, $Train10$, begins with 9000 game steps of training from previous games. The testing runs maintain an RL learning rate of $\alpha = 0$, and an exploration rate of $\varepsilon = 0$ for ε-greedy, or $\tau = 0.00001$ for softmax ($\tau > 0$). While the final policies obtained through testing are insightful, they serve little purpose beyond analyzing the training process. Real agents should maintain a low level exploration rate to promote the discovery of better policies, given sudden shifts, or changes in the game world.

4.5.4 Discount and Trace-Decay

For each game testing configuration, a large number of discount and trace-decay combinations are tested, leading to 26 results, for one game execution (training or testing). The performance of a set of variables will always include every combination of γ and λ seen in Table 4.2, unless otherwise specified.

	Eligibility Trace λ					
Discount γ	**0.0**	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
0.0	00	-	-	-	-	-
0.1	-	11	31	51	71	91
0.3	-	13	33	53	73	93
0.5	-	15	35	55	75	95
0.7	-	17	37	57	77	97
0.9	-	19	39	59	79	99

Table 4.2: Testing combinations for discount and trace-decay values, 26 combinations.

4.5.5 Multiple Motives

With multiple motivations, the percent of total reward may not be the best metric. For example, given an agent with three motives, the results show an average of 70% of total reward, for a particular set of testing configurations. While this number may appear satisfactory, it indicates the possibility that one motivation is being ignored entirely, in favour of satisfying the other two motivations.

For this reason, both the percent of total reward and the percent of steps in bounds are used in determining the best possible variable configurations.

4.5.6 Graphing Results

The percent of total reward and percent of steps within bounds are graphed using box plots, to clearly illustrate the distribution of results. In a box plot, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers encompass the furthest data points that are not outliers, and outliers are plotted separately. Raw results may also be plotted using scatter plots, to show exactly how the results are distributed.

4.5.7 Testing Outline

Testing any particular game is separated by the three main RL algorithms: Sarsa, Q-learning and Dyna-Q. Testing in some Games is further separated into types of action selection methods, either Softmax or ε-greedy. In the Table 4.3 below, game testing sections are outlined along with their associated Chapter sections. See the discussion section of each game for the optimal game policy, in-depth qualitative analysis of the best and worst policies of the game, and overall lessons learned for each individual Game.

Game	Sarsa		Q Learning		DynaQ		Discussion
	$\varepsilon - greedy$	$Softmax$	$\varepsilon - greedy$	$Softmax$	$\varepsilon - greedy$	$Softmax$	
1	5.2		5.3		5.4		5.5
2	6.2		6.3		6.4		6.5
3	7.2		7.3		7.4		7.5
4	8.2	8.4	8.2	8.5	8.6	8.7	8.8
5	9.2						9.3

Table 4.3: Testing outline specifying where to find results of specific tests.

4.6 Summary

This method incorporates the status of an agent's motivations into the agent's state representation of the game world. Rewards are given by comparing the change in motivation values, for better or worst. In this way, reinforcement learning is used to learn an action selection policy that seeks to satisfy all of the agent's motives.

Experiments were designed to test the impact of each individual parameter associated with RL, and the ANNs that are used. The goal is to find what values give consistently good results, and whether the ideal values change with added game complexity. The following successive Chapters cover the experimental results of each game scenario.

Chapter 5

Game One

Testing begins with a very simple learning scenario. The game world consists of a house that is occupied by an agent named *Tarzan*, and an object named *food*. Every game step, Tarzan has the option to *eat food*, or *do nothing*, with the goal of ultimately satisfying his only motive, *hunger*. The action *eat food* increases Tarzan's *hunger motive* by 5 units, and takes 1 game step to complete. If Tarzan decides to *do nothing*, all motives decay according to a linear decay function. The learning task for Game One is to develop an action selection policy that knows when it is best to *eat*, and when it is best to *do nothing*.

Testing for Game One will concentrate on determining the impact of the learning parameters, and comparing the results of different RL methods (Sarsa, Q-learning, and Dyna-Q). Given that ε-greedy is the most intuitive, and most common, method of managing exploration, it is the only action selection method used during Game One testing. Softmax action selection is used to guide exploration to avoid the worst case actions, and will only be tested with more complicated Game scenarios (Game Four and Game Five).

5.1 Testing Outline

Testing began with Sarsa in Section 5.2, and continued with Q-learning in Section 5.3. Finally, optimal values found during Q-learning are used as a base for testing Dyna-Q in Section 5.4. The Section 5.5 discusses parameter significance, qualitative analysis, and addresses the consistency of the optimal policies. All testing configurations in Game One are repeated 5 times to measure consistency.

5.2 Sarsa

In the following sections, Sarsa is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 5.1.

Sub-Section	Testing	Hn_Q	mR	η_Q	RL Learning Rate	Exploration
5.2.1	ε	[100]	-0.1	0.7	$\alpha_s = 0.2$ $\alpha_e = 0.1$	
5.2.2	α	[100]	-0.1	0.7		$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
5.2.3	η_Q	[100]	-0.1		$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
5.2.4	mR	[100]		0.99	$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
5.2.5	Hn_Q		-0.1	0.99	$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$

Table 5.1: Outline of fixed variables used in the testing configurations for Game One using Sarsa, and ε-greedy.

5.2.1 Exploration Rate

The testing configurations outlined in Table 5.2, show how the starting and ending exploration rates (ε_s and ε_e) impact learning. All other variables are held constant, seen in Table 5.1, to clearly interpret the results of varying ε_s and ε_e. The distributions of results are shown in Figure 5.1, and discussed in Section 5.5.

Test				1	2	3	4	5
trainN	α_s	0.2	ε_s	0.9	0.9	0.9	0.9	0.7
	α_e	0.1	ε_e	0.7	0.5	0.3	0.1	0.5
testN	α_s	0.0	ε_s	0.0	0.0	0.0	0.0	0.0
	α_e	0.0	ε_e	0.0	0.0	0.0	0.0	0.0
Test				6	7	8	9	10
trainN	α_s	0.2	ε_s	0.7	0.7	0.5	0.5	0.3
	α_e	0.1	ε_e	0.3	0.1	0.3	0.1	0.1
testN	α_s	0.0	ε_s	0.0	0.0	0.0	0.0	0.0
	α_e	0.0	ε_e	0.0	0.0	0.0	0.0	0.0

Table 5.2: How ε_s and ε_e vary for different testing configurations in Game One using Sarsa, and $\mathcal{E}$-Greedy.

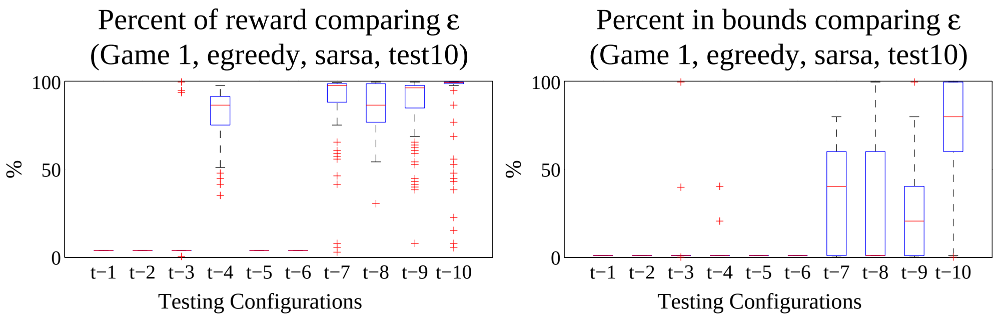


Figure 5.1: Comparing the results of varying ε_s and ε_e during final policy tests, where t-n is the n^{th} testing configuration according to Table 5.2. The testing configurations are run in Game One using Sarsa, and ε-greedy.

5.2.2 Reinforcement Learning Rate

The testing configurations, outlined in Table 5.3, show how the starting and ending reinforcement learning rates (α_s and α_e) impact the learned policies. To clearly interpret the results of varying α_s and α_e, all other variables are held constant, seen in Table 5.1. The distributions of results are shown in Figure 5.2, and discussed in Section 5.5.

Test				1	2	3	4	5	6	7	8	9	10	11
trainN	ε_s	0.3	α_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3	1.0
	ε_e	0.1	α_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1	1.0
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5.3: How α_s and α_e vary for different testing configurations in Game One with Sarsa, and ε-Greedy.

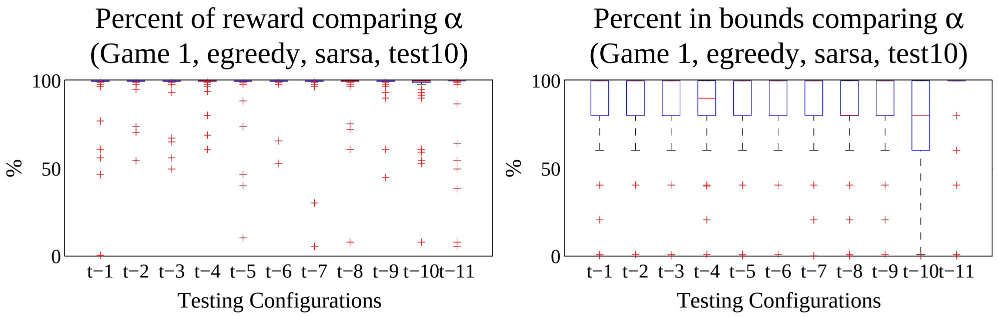


Figure 5.2: Comparing the results of varying α_s and α_e during final policy tests, where t-n defines the n^{th} testing configuration according to Table 5.3. The testing configurations are run in Game One using Sarsa, and ε-greedy.

5.2.3 Q-Function ANN Learning Rate

A number of testing configurations, outlined in Table 5.4, show how the Q-function approximation ANN learning rate (η_Q) influences the learned policies. To clearly interpret the results of varying η_Q, all other variables are held constant, seen in Table 5.1. The distributions of results are shown in Figure 5.3, and discussed in Section 5.5.

Test	1	2	3	4	5	6	7	8	9	10
η_Q	0.99	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Table 5.4: How η_Q changes for different testing configurations in Game One with Sarsa, and ε-Greedy.

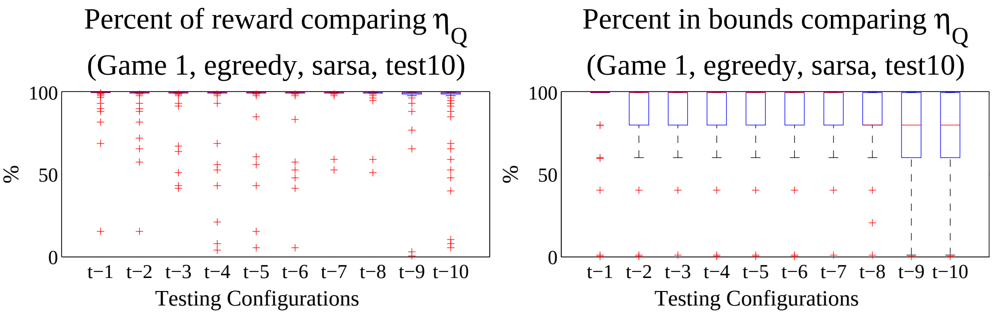


Figure 5.3: Comparing the results of varying η_Q during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.4. The testing configurations are run in Game One using Sarsa, and ε-greedy.

5.2.4 Individual Motive Reward

A number of testing configurations, outlined in Table 5.5, show how the motive reward scaling factor (mR) influences learning. Other key variables are held constant, see Table 5.1, to clearly interpret the results of varying mR. The distributions of results are shown in Figure 5.4, and discussed in Section 5.5.

Test	1	2	3	4
mR	-1.0	-0.5	-0.1	-0.01

Table 5.5: How mR changes for different testing configurations in Game One with Sarsa, ε-Greedy.

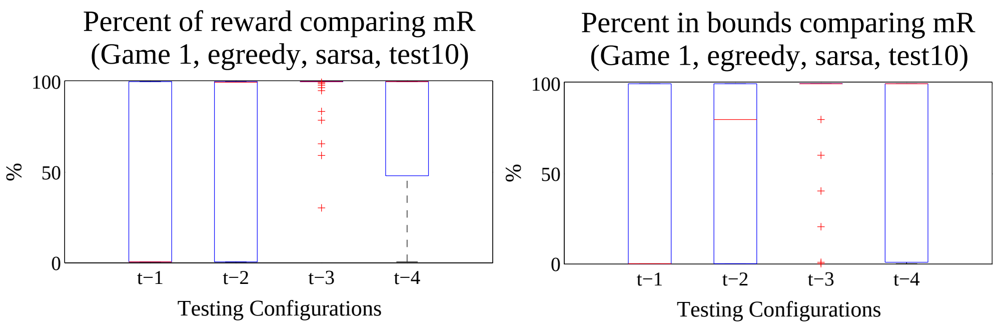


Figure 5.4: Comparing the results of varying mR during final policy tests, where t-n defines the n^{th} test according to Table 5.5. The testing configurations are run in Game One using Sarsa, and ε-greedy.

5.2.5 Q-Function Hidden Neurons

A number of testing configurations, outlined in Table 5.6, were used to show how the Q-function approximation ANN's hidden neuron configuration (Hn_Q) influences learning. Other key variables are held constant, see Table 5.1, to clearly interpret the results of varying Hn_Q. The distributions of results for the percent of reward, and the percent of steps in bounds, are shown in Figure 5.5, and discussed in Section 5.5.

Test	1	2	3	4
Hn	[3]	[10]	[50]	[100]
Test	5	6	7	8
Hn	[3 3]	[10 10]	[50 50]	[100 100]
Test	9	10	11	12
Hn	[3 3 3]	[10 10 10]	[50 50 50]	[100 100 100]

Table 5.6: How Hn_Q is changed for different testing configurations in Game One with Sarsa, and ε-greedy.

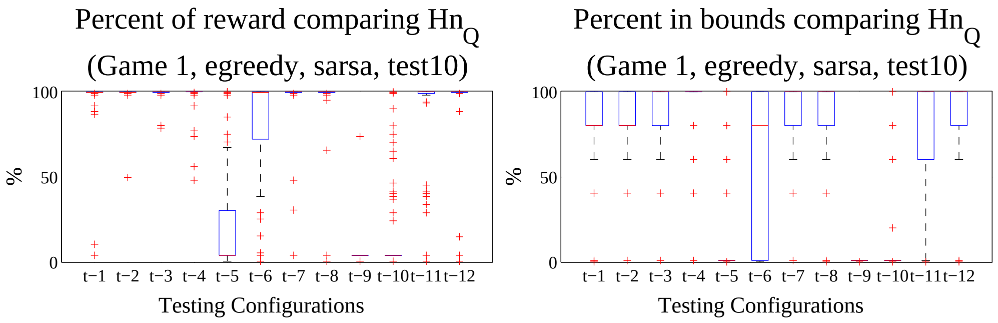


Figure 5.5: Comparing the results of varying Hn_Q during final policy tests, where t-n defines the n^{th} test according to Table 5.6. The testing configurations are run in Game One using Sarsa, and ε-greedy.

5.3 Q-Learning

In the following sections, Q-learning is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 5.7.

Sub-Section	Testing	Hn_Q	mR	η_Q	RL rate	Exploration
5.3.1	ε	[100]	-0.1	0.7	$\alpha_s = 0.2$ $\alpha_e = 0.1$	
5.3.2	α	[100]	-0.1	0.7		$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
5.3.3	η_Q	[100]	-0.1		$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
5.3.4	mR	[100]		0.99	$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
5.3.5	Hn_Q		-0.1	0.99	$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$

Table 5.7: Outline of fixed variables used in the testing configurations for Game One using Q-learning, and ε-greedy.

5.3.1 Exploration Rate

A number of testing configurations, outlined in Table 5.7, show how the starting and ending exploration rates (ε_s and ε_e) impact learning. All other variables are held constant, seen in Table 5.7, to clearly interpret the results of varying ε_s and ε_e. The distributions of results are shown in Figure 5.6, and discussed in Section 5.5.

Test				1	2	3	4	5
trainN	α_s	0.2	ε_s	0.9	0.9	0.9	0.9	0.7
	α_e	0.1	ε_e	0.7	0.5	0.3	0.1	0.5
testN	α_s	0.0	ε_s	0.0	0.0	0.0	0.0	0.0
	α_e	0.0	ε_e	0.0	0.0	0.0	0.0	0.0
Test				6	7	8	9	10
trainN	α_s	0.2	ε_s	0.7	0.7	0.5	0.5	0.3
	α_e	0.1	ε_e	0.3	0.1	0.3	0.1	0.1
testN	α_s	0.0	ε_s	0.0	0.0	0.0	0.0	0.0
	α_e	0.0	ε_e	0.0	0.0	0.0	0.0	0.0

Table 5.8: How ε_s and ε_e vary for different testing configurations in Game One using Q-learning, and $\mathcal{E}$-Greedy.

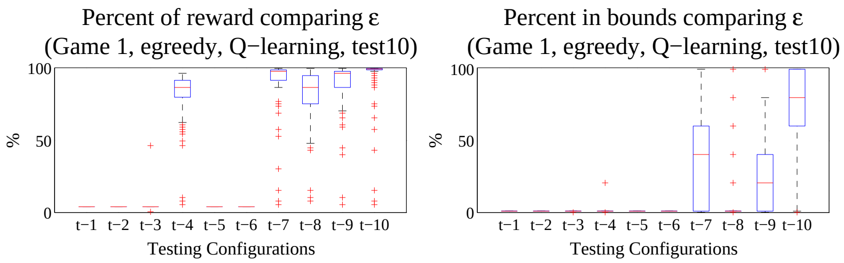


Figure 5.6: Comparing the results of varying ε_s and ε_e during final policy tests, where t-n is the n^{th} testing configuration according to Table 5.8. The testing configurations are run in Game One using Q-learning, and ε-greedy.

5.3.2 Reinforcement Learning Rate

The testing configurations, outlined in Table 5.9, show how the starting and ending reinforcement learning rates (α_s and α_e) impact the learned policies. To clearly interpret the results of varying α_s and α_e, all other variables are held constant, seen in Table 5.7. The distributions of results are shown in Figure 5.7, and discussed in Section 5.5.

Test				1	2	3	4	5	6
trainN	ε_s	0.3	α_s	0.9	0.9	0.9	0.9	0.7	0.7
	ε_e	0.1	α_e	0.7	0.5	0.3	0.1	0.5	0.3
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	0.0
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	0.0
Test				7	8	9	10	11	
trainN	ε_s	0.3	α_s	0.7	0.5	0.5	0.3	1.0	
	ε_e	0.1	α_e	0.1	0.3	0.1	0.1	1.0	
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	

Table 5.9: How α_s and α_e vary for different testing configurations in Game One with Q-learning, and ε-Greedy.

5.3.3 Q-Function ANN Learning Rate

The testing configurations outlined in Table 5.10, show how the Q-function approximation ANN learning rate (η_Q) influences the learned policies. To clearly interpret the results of varying η_Q, all other variables are held constant, seen in Table 5.7. The distributions of results are shown in Figure 5.8, and discussed in Section 5.5.

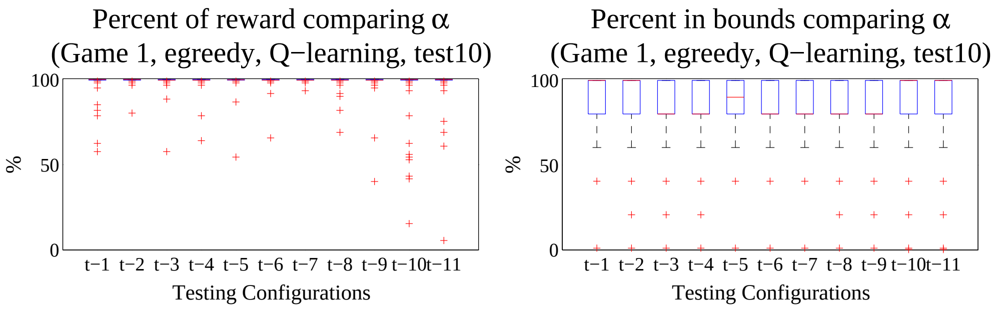


Figure 5.7: Comparing the results of varying α_s and α_e, where t-n defines the n^{th} test according to Table 5.9, consisting only of the final policies reached by each parameter configuration (130 results). The agent follows the policy with absolutely no exploration. The testing configurations are run in Game Two using Q-learning, and ε-greedy.

Test	1	2	3	4	5	6	7	8	9	10
η_Q	0.99	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Table 5.10: How η_Q changes for different testing configurations in Game One with Q-learning, and ε-Greedy.

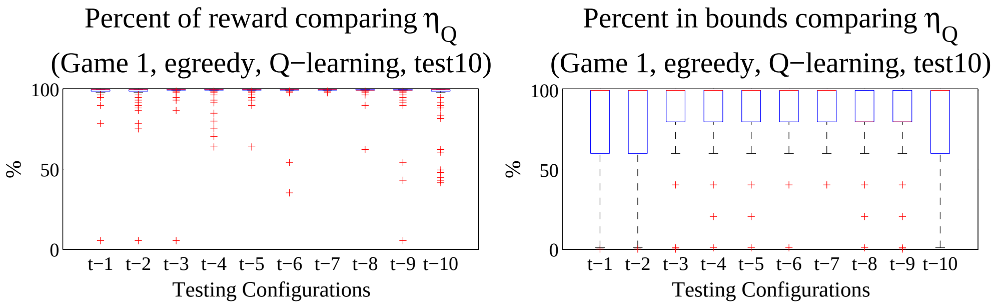


Figure 5.8: Comparing the results of varying η_Q during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.10. The testing configurations are run in Game One using Q-learning, and ε-greedy.

5.3.4 Individual Motive Reward

The number of testing configurations outlined in Table 5.11, show how the motive reward scaling factor (mR) influences learning. Other key variables are held constant, see Table 5.7, to clearly interpret the results of varying mR. The distributions of results are shown in Figure 5.9, and discussed in Section 5.5.

Test	1	2	3	4
mR	-1.0	-0.5	-0.1	-0.01

Table 5.11: How mR changes for different testing configurations in Game One with Q-learning, ε-Greedy.

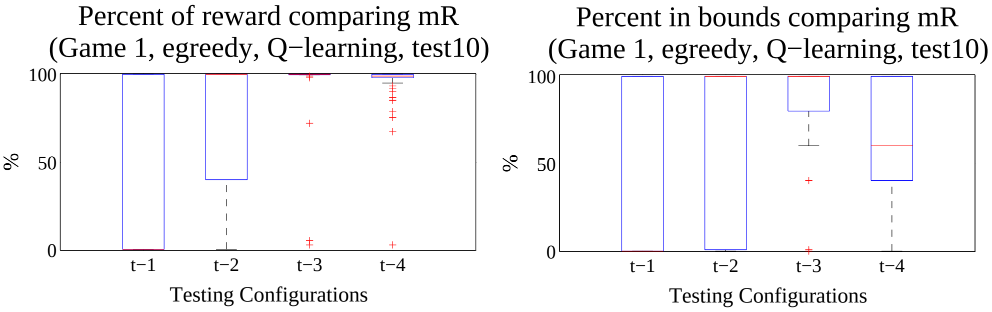


Figure 5.9: Comparing the results of varying mR during final policy tests, where t-n defines the n^{th} test according to Table 5.11. The testing configurations are run in Game One using Q-learning, and ε-greedy.

5.3.5 Q-Function Hidden Neurons

The testing configurations outlined in Table 5.12, are used to show how the Q-function approximation ANN's hidden neuron configuration (Hn_Q) influences learning. Other key variables are held constant, see Table 5.7, to clearly interpret the results of varying Hn_Q. The distributions of results for the percent of reward, and the percent of steps in bounds, are shown in Figure 5.10, and discussed in Section 5.5.

Test	1	2	3	4
Hn	[3]	[10]	[50]	[100]
Test	5	6	7	8
Hn	[3 3]	[10 10]	[50 50]	[100 100]
Test	9	10	11	12
Hn	[3 3 3]	[10 10 10]	[50 50 50]	[100 100 100]

Table 5.12: How Hn_Q is changed for different testing configurations in Game One with Q-learning, and ε-greedy.

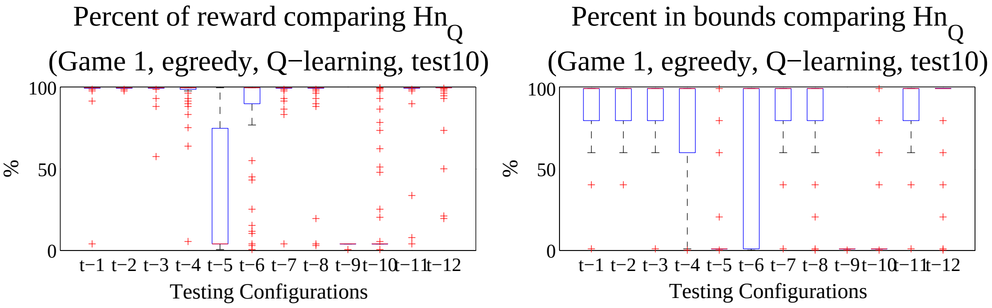


Figure 5.10: Comparing the results of varying Hn_Q during final policy tests, where t-n defines the n^{th} test according to Table 5.12. The testing configurations are run in Game One using Q-learning, and ε-greedy.

5.4 Dyna-Q

In the following sections, Dyna-Q is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 5.13.

Sub-Section	Test	Hn_Q	η_Q	mR	RL rate	Exploration	Hn_M	η_M	pS
5.4.1	Hn_M	[50]	0.4	-0.1	$\alpha_s = 0.7$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$		0.4	3
5.4.2	η_M	[50]	0.4	-0.1	$\alpha_s = 0.7$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$	[3]		3
5.4.3	pS	[50]	0.4	-0.1	$\alpha_s = 0.7$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$	[3]	0.4	

Table 5.13: Outline of fixed variables used in the testing configurations for Game One using Dyna-Q, and ε-greedy.

5.4.1 Model-Function Hidden Neurons

A number of testing configurations, outlined in Table 5.14, were used to show how the Model-function approximation ANN's hidden neuron configuration (Hn_M) influences learning. Other key variables are held constant, see Table 5.13. The distributions of results are shown in Figure 5.11, and discussed in Section 5.5.

Tests	1	2	3	4
Hn	[3]	[10]	[50]	[100]
Tests	5	6	7	8
Hn	[3 3]	[10 10]	[50 50]	[100 100]
Tests	9	10	11	12
Hn	[3 3 3]	[10 10 10]	[50 50 50]	[100 100 100]

Table 5.14: How Hn_M is changed for different testing configurations in Game One with Dyna-Q, and ε-greedy.

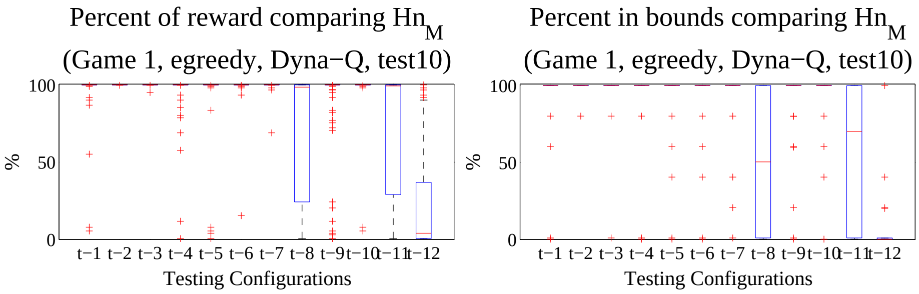


Figure 5.11: Comparing the results of varying Hn_M during final policy tests, where t-n defines the n^{th} test according to Table 5.14. The testing configurations are run in Game One using Dyna-Q, and ε-greedy.

5.4.2 Model-Function Learning Rate

A number of testing configurations, outlined in Table 5.15, show how the Model-function approximation ANN learning rate (η_M) influences the learned policies. To clearly interpret the results of varying η_M, all other variables are held constant, seen in Table 5.13. The distributions of results are shown in Figure 5.12, and discussed in Section 5.5.

Test	1	2	3	4	5	6	7	8	9	10
η_M	0.99	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Table 5.15: How η_M changes for different testing configurations in Game One with Dyna-Q, and ε-Greedy.

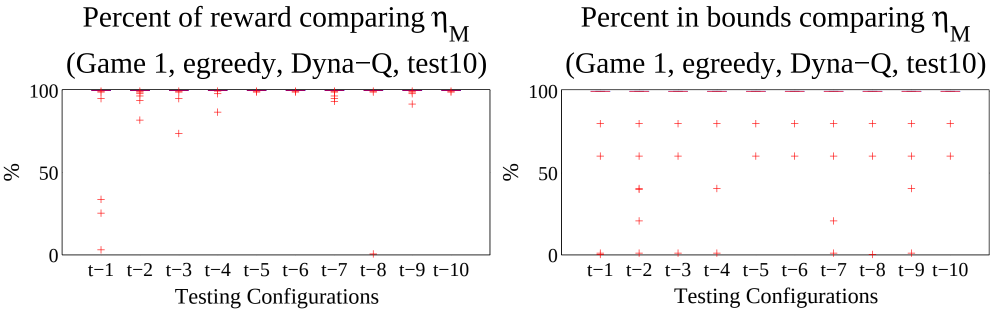


Figure 5.12: Comparing the results of varying η_M during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.15. The testing configurations are run in Game One using Dyna-Q, and ε-greedy.

5.4.3 Dyna-Q Planning Steps

A number of testing configurations, outlined in Table 5.16, show how the number of planning steps (pS) influences the learned policies. To clearly interpret the results of varying pS, all other variables are held constant, seen in Table 5.13. The distributions of results are shown in Figure 5.13, and discussed in Section 5.5.

Test	1	2	3	4
pS	10	20	50	100

Table 5.16: How pS changes for different testing configurations in Game One with Dyna-Q, and ε-Greedy.

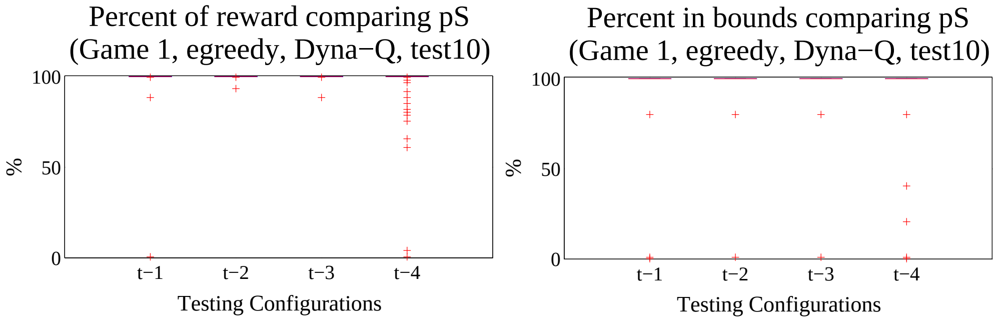


Figure 5.13: Comparing the results of varying pS during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 5.16. The testing configurations are run in Game One using Dyna-Q, and ε-greedy.

5.5 Discussion

In the real-world implementation of desire driven reinforcement learning, the developer should not worry about setting a large number of variables. This section evaluates the most significant variables that are important for the overall performance of this method.

5.5.1 Parameters

Exploration Rate The exploration rate needed for this simple game scenario was expected to be low. Results for Sarsa (Figure 5.1), and Q-learning (Figure 5.6), show the the best policies were learned when the exploration rate was reduced to 0.1 over time.

Reinforcement Learning Rate The RL learning rate (α) does not show significant difference in results, given nearly equal median percent of total reward for Sarsa (Figure 5.2), and Q-learning (Figure 5.7).

Q-Function ANN Learning Rate The Q-function ANN learning rate (η_Q) has the most impact on training time, given it controls how quickly the Q-function is approximated. The results show no significant difference in percent of total reward for the final testing game, for Sarsa (Figure 5.3), or Q-learning (Figure 5.8).

Motive Reward Factor The motive reward scaling factor (mR), is the minimum reward that can be associated with one of the agent's motives, at any time step in the Game. The variable mR directly influences the magnitude of the rewards throughout a game. The results for Sarsa (Figure 5.4), and Q-learning (Figure 5.9) show that best results occur when $mR = -0.1$. However, in this case, Q-learning shows better results than Sarsa when $mR = -0.5$, and $mR = -0.01$.

Q-Function ANN Hidden Neurons The hidden neuron configuration for the Q-function ANN should reflect the complexity of the function being approximated. Results from Sarsa (Figure 5.5), and Q-learning (Figure 5.10), show that this simple game scenario requires very little hidden neurons to achieve a good policy. The test t-2 with $Hn = [10]$ has one hidden neuron layer, and 10 hidden neurons, but still results in close to 100% of total rewards received for the final learned policy. Adding additional hidden layers increases the complexity with no added benefits. Adding additional hidden layers has a detrimental effect on the results when the number of neurons in those hidden layers is small.

Model-Function ANN Hidden Neurons The complexity of the game world has a direct impact on the complexity needed for the model-function. Results from testing Hn_M show the simplicity of this game scenario, given that only 3 hidden neurons are needed to achieve a good model of the environment, seen in Figure 5.11. The results also show how a large number of hidden neurons, and hidden layers, have a detrimental impact on the Model-function approximation.

Model-Function ANN Learning Rate The learning rate of the Model-function approximation, has a direct impact on the quality of the approximation. If the learning rate is too high, the ANN will not be able to accurately represent the true model of the environment. However, if the learning rate is too low, the ANN will need more time to learn the model-function approximation. The Figure 5.12 shows the results of varying η_M, with lower values showing slightly better results given this simple learning task.

Dyna-Q Planning Steps With an accurate model of the game world, increasing the number of planning steps directly increases the speed at which the optimal policy is learned. If the model of the game world is inaccurate, increasing the number of planning steps will negatively affect the performance of the agent. Results from testing pS, seen in Figure 5.17, show that even with $pS = 50$ the performance is near perfect.

5.5.2 Consistency

Testing the consistency of the results involves repeating tests under identical conditions, and comparing the results. In this case, the values in Table 5.17 are used in 10 separate game tests, with results in Figure 5.14. The resulting policies have equal medians in both percent of total rewards and percent of steps in bounds, confirming consistency.

Hn_Q	η_Q	mR	RL Rate	Exploration Rate	Hn_M	η_M	pS
[50]	0.4	-0.1	$\alpha_s = 0.7$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$	[50]	0.2	50

Table 5.17: Outline of the variables used to test the consistency of Game One's optimal policy, using Dyna-Q and ε-greedy.

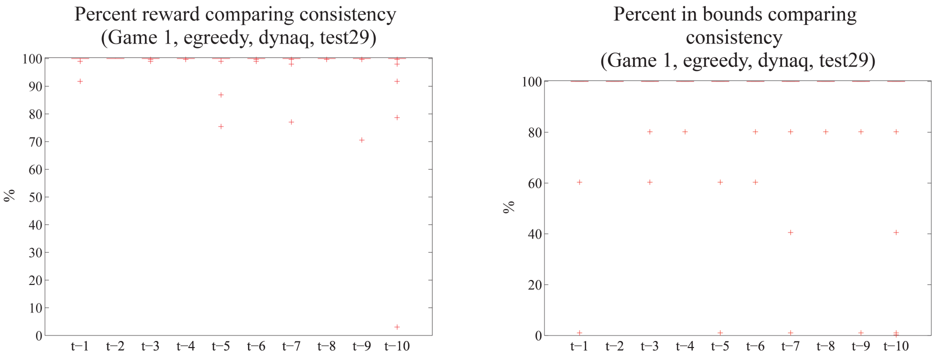


Figure 5.14: Compare the consistency of the final Game One policies learned by the RL agent from ten repeated tests, where t-n defines the n^{th} repeated test.

5.5.3 Training Time of Optimal Policy

The rate at which the policy is learned should be as high as possible, without compromising the end result. Some variables may produce better policies given a shorter training time, however, slower training in most cases will produce better results when given longer training periods. In the case of this Game One scenario, the model-function accurately models the game world, resulting in quicker training time given more planning steps, see Figure 5.16. The model function ANN more accurately models the game world when its learning rate is lower, see Figure 5.15.

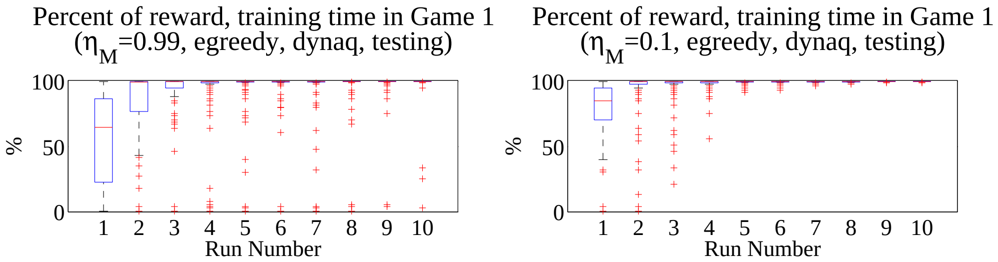


Figure 5.15: Showing the percent of reward received after every game (1000 game steps), comparing two contrasting values of η_M.

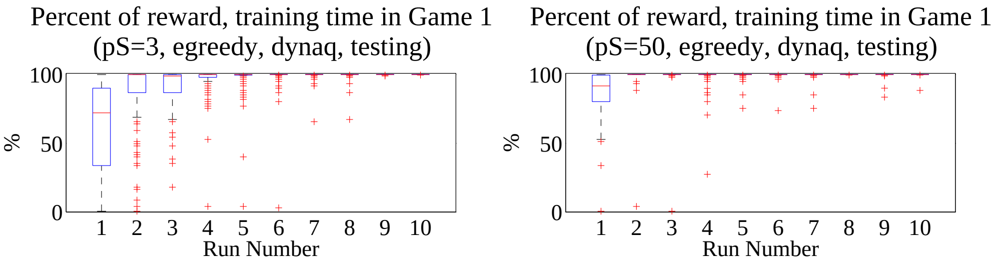
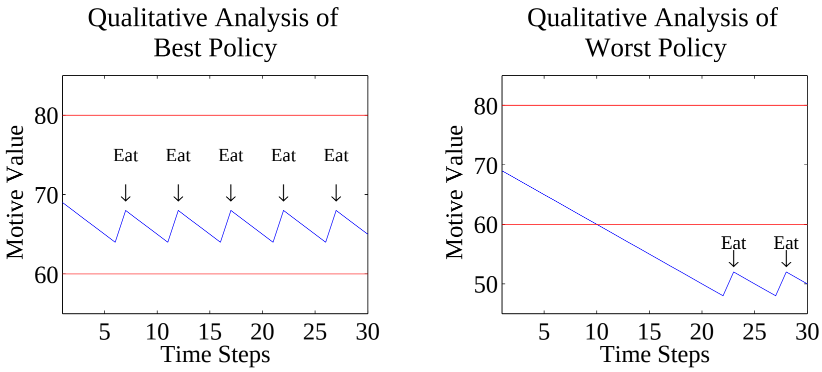


Figure 5.16: Showing the percent of reward received after every game (1000 game steps), comparing two contrasting values of pS .

5.5.4 Qualitative Analysis

A policy dictates to the agent what action to take, given any situation in the game. The optimal policy was found using quantitative methods, where the percent of reward, and the percent of steps in bounds, were used to evaluate the performance of a policy. Quantitative analysis does not examine impact of the learned sequences of states, and actions. For this reason, a policy should also be evaluated through actions, rather than percentages. Qualitative analysis of the optimal policy, seen in Figure 5.17a, shows how the agent's motives are satisfied according to actions, and the resulting motive values. The Figure 5.17b shows an example of the action selections taken by an in-optimal policy. Both policies were found using the parameters in Table 5.17. The in-optimal policy in Figure 5.17b is within the minimum and maximum bounds for only 1% (10 steps) of the game. Eating begins too late, at step 23, making the food motive value closer to the minimum bounds, but never within the desired range. Given that 98.6% of the policies learned during consistency testing had 100% of steps in bounds, the in-optimal policies are most likely caused by an undesirable initial ANN.



(a) Best policy found by the set of parameters in Table 5.17 with 100% of steps in bounds.

(b) Worst policy found by the set of parameters in Table 5.17 with only 1% of steps in bounds.

Figure 5.17: Qualitative analysis of best and worst policies found by the set of parameters in Table 5.17.

5.6 Summary

Game One is a simple scenario that is easily learned by a wide range of parameters, given minimal training time. The observations made about significant parameter values are for this scenario alone. Further testing in the following Chapters will determine if those observations are true for all game complexities. The next Chapter introduces Game two, with an increased complexity through additional actions, and multiple motivations.

Chapter 6

Game Two

The second game world adds complexity through multiple objects, three motives, and actions with delayed reward. The agent Tarzan resides in his house, with a TV, a treadmill, and a fridge. The fridge contains one *healthy food* and one *greasy food.* The fridge must be opened before the food becomes accessible. Tarzan's motives include *hunger*, *health,* and *entertainment.* A detailed outline of objects, and their associated actions, can be seen in Table 6.1. The purpose of this game is to learn combination actions, such as open the fridge to eat food, while trying to satisfy multiple motives.

Testing for Game Two is done the same way as Game One. The goal of testing is to determine the impact of the learning parameters, while also comparing the results between different RL methods (Sarsa, Q-learning, and Dyna-Q), using the $\varepsilon - greedy$ action selection method.

Object	**Action**	**Affects**			
		Hunger	**Healthy**	**Entertainment**	**More Objects**
Healthy food	Eat	+15	+5		
Greasy food	Eat	+10	-5		
Fridge	Open				Access food
Treadmill	Go to	-5	+30		
TV	Go to		-5	+20	

Table 6.1: Description of Tarzan's actions and their resulting impact on the game world.

6.1 Testing Outline

Testing began with Sarsa in Section 6.2, and proceeded with Q-learning in Section 6.3. Finally, optimal values found during Q-learning are used as a base for testing Dyna-Q in Section 6.4. The Section 6.5 discusses the parameter significance with respect to Game One, qualitative analysis, and explores the consistency of the optimal results. All testing configurations in Game Two are repeated 5 times to measure consistency.

6.2 Sarsa

In the following sections, Sarsa is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 6.2.

Sub-Section	Testing	Hn_Q	mR	η_Q	RL Learning Rate	Exploration
6.2.1	ε	[10]	-0.1	0.4	$\alpha_s = 0.7$ $\alpha_e = 0.3$	
6.2.2	α	[10]	-0.1	0.4		$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
6.2.3	η_Q	[10]	-0.1		$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
6.2.4	mR	[10]		0.9	$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
6.2.5	Hn_Q		-0.1	0.9	$\alpha_s = 1.0$ $\alpha_e = 1.0$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$

Table 6.2: Outline of fixed variables used in the testing configurations for Game Two using Sarsa, and ε-greedy.

6.2.1 Exploration Rate

To show the impact of the exploration rate (ε_s and ε_e) on the results, a number variable configurations were tested, see Table 6.3. Other necessary variables are held constant, seen in Table 6.2, to highlight the results of varying ε_s and ε_e. The results are shown in Figure 6.1, and discussed in Section 6.5.

Test				1	2	3	4	5	6	7	8	9	10
trainN	α_s	0.2	ε_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	α_e	0.1	ε_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1
testN	α_s	0.0	ε_s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	α_e	0.0	ε_e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6.3: How ε_s and ε_e vary for different testing configurations in Game Two using Sarsa, and $\mathcal{E}$-Greedy .

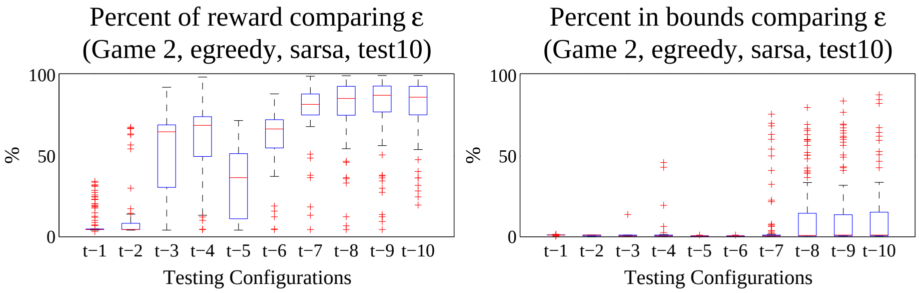


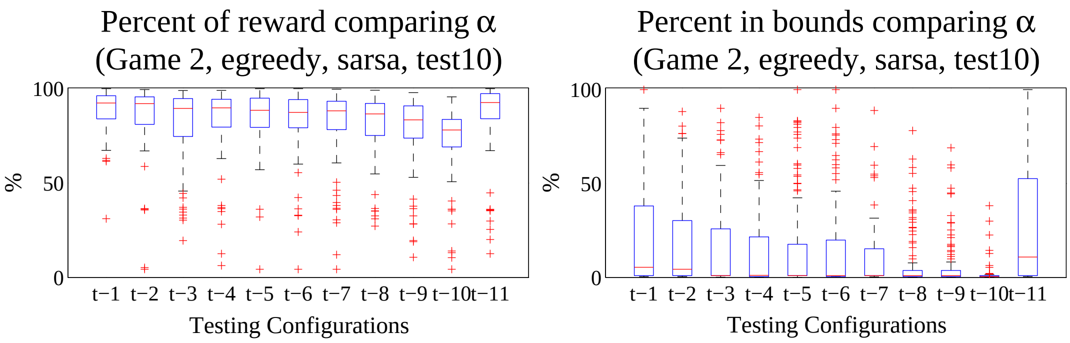
Figure 6.1: Comparing the results of varying ε_s and ε_e during final policy tests, where t-n is the n^{th} testing configuration according to Table 6.3. These Game Two testing configurations are run using Sarsa, and ε-greedy.

6.2.2 Reinforcement Learning Rate

The testing configurations, outlined in Table 6.4, show how the starting and ending reinforcement learning rates (α_s and α_e) impact the learned policies. To clearly interpret the results of varying α_s and α_e, all other variables are held constant, seen in Table 6.2. The distributions of results are shown in Figure 6.2, and discussed in Section 6.5.

Test				1	2	3	4	5	6	7	8	9	10	11
trainN	ε_s	0.3	α_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3	1.0
	ε_e	0.1	α_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1	1.0
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6.4: Outlines how the starting and ending learning values (α_s and α_e) vary for training games (trainN) as well as testing games (testN), in Game Two with Sarsa and ε-greedy.



Percent of reward comparing α (Game 2, egreedy, sarsa, test10)

Percent in bounds comparing α (Game 2, egreedy, sarsa, test10)

Figure 6.2: Comparing the results of varying α_s, and α_e, where t-n defines the n^{th} test according to Table 6.4, consisting only of (test10), the final policy reached by each parameter configuration (130 results). The agent follows the policy with no exploration, and no learning. The testing configurations are run in Game Two using Sarsa, and ε-greedy

6.2.3 Q-Function Learning Rate

A number of testing configurations, outlined in Table 6.5, show how the Q-function ANN learning rate (η_Q) influences the learned policies. To clearly interpret the results of varying η_Q, all other variables are held constant, seen in Table 6.2. The distributions of results are shown in Figure 6.3, and discussed in Section 6.5.

Test	1	2	3	4	5	6	7	8	9	10
η_Q	0.99	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Table 6.5: How η_Q changes for different testing configurations in Game Two with Sarsa, and ε-Greedy.

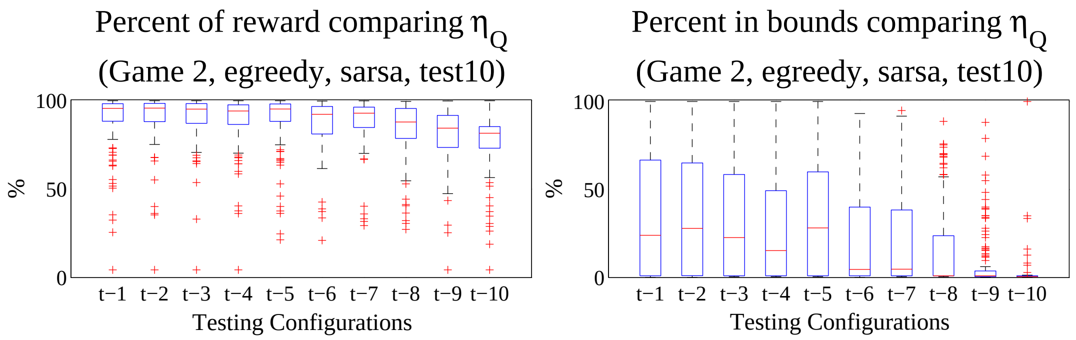


Figure 6.3: Comparing the results of varying η_Q, where t-n defines the n^{th} test according to Table 6.5, consisting of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Sarsa, and ε-greedy.

6.2.4 Individual Motive Reward

A number of testing configurations, outlined in Table 6.6, show how the motive reward scaling factor (mR) influences learning. Other key variables are held constant, see Table 6.2 , to clearly interpret the results of varying mR. The distributions of results are shown in Figure 6.4, and discussed in Section 6.5.

Test	1	2	3	4
mR	-0.2	-0.1	-0.05	-0.01

Table 6.6: How mR changes for different testing configurations in Game Two with Sarsa, and ε-Greedy.

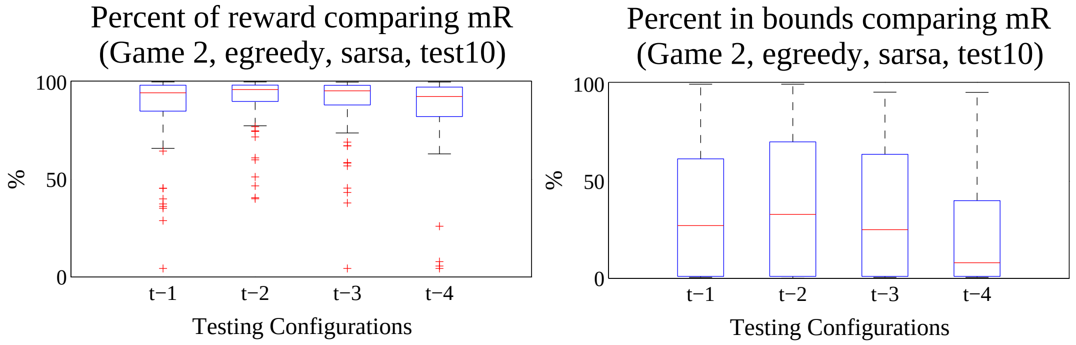


Figure 6.4: Comparing the results of varying mR, where t-n defines the n^{th} test according to Table 6.6, consisting only of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Sarsa, and ε-greedy.

6.2.5 Q-Function Hidden Neurons

To show the impact of the Q-function ANN hidden neuron configuration rate (Hn_Q) on the results, a number variable configurations were tested, see Table 6.7. Other necessary variables are held constant, seen in Table 6.2, to highlight the impact of varying Hn_Q. The results are shown in Figure 6.5, and discussed in Section 6.5.

Test	1	2	3	4
Hn	[3]	[10]	[50]	[100]
Test	5	6	7	8
Hn	[3 3]	[10 10]	[50 50]	[100 100]
Test	9	10	11	12
Hn	[3 3 3]	[10 10 10]	[50 50 50]	[100 100 100]

Table 6.7: How Hn_Q is changed for different testing configurations in Game Two with Sarsa, and ε-greedy.

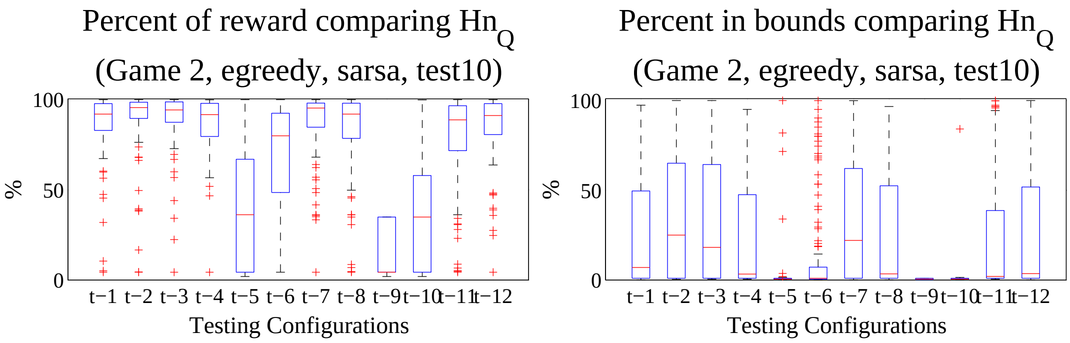


Figure 6.5: Comparing the results of varying Hn_Q during final policy tests, where t-n defines the n^{th} test according to Table 6.7. The testing configurations are run in Game One using Sarsa, and ε-greedy.

6.3 Q-Learning

In the following sections, Q-learning is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 6.8.

Sub-Section	Testing	Hn_Q	mR	η_Q	RL rate	Exploration
6.3.1	ε	[50]	-0.1	0.5	$\alpha_s = 0.9$ $\alpha_e = 0.7$	
6.3.2	α	[50]	-0.1	0.5		$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$
6.3.3	η_Q	[50]	-0.1		$\alpha_s = 0.9$ $\alpha_e = 0.5$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$
6.3.4	mR	[50]		0.7	$\alpha_s = 0.9$ $\alpha_e = 0.5$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$
6.3.5	Hn_Q		-0.1	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.5$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$

Table 6.8: Outline of fixed variables used in the testing configurations for Game Two using Q-learning, and ε-greedy.

6.3.1 Exploration Rate

A number of testing configurations, outlined in Table 6.9, show how the starting and ending exploration rates (ε_s and ε_e) impact learning. All other variables are held constant, seen in Table 6.2, to clearly interpret the results of varying ε_s and ε_e. The results are shown in Figure 6.6, and discussed in Section 6.5.

Test				1	2	3	4	5	6	7	8	9	10
trainN	α_s	0.2	ε_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	α_e	0.1	ε_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1

Table 6.9: Outlines how ε_s and ε_e vary for different testing configurations in Game Two using Q-learning, and $\mathcal{E}$-Greedy.

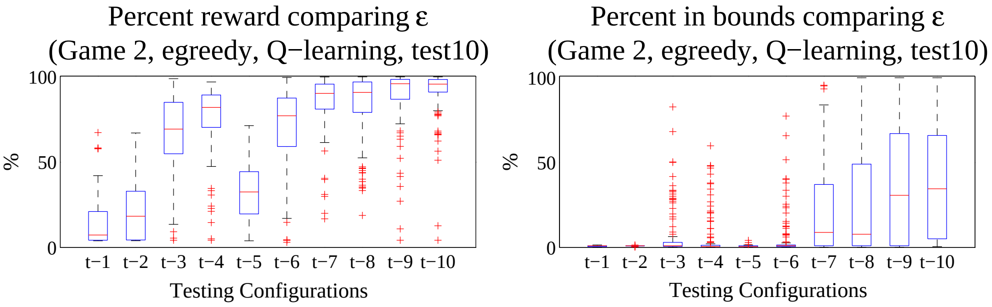


Figure 6.6: Comparing the results of varying ε_s and ε_e, where t-n defines the n^{th} test according to Table 6.9, consisting of the final policies. The testing configurations are run in Game Two using Q-learning, and ε-greedy.

6.3.2 Reinforcement Learning Rate

The testing configurations in Table 6.10 show the impact of the RL learning rate (α) on the percent of total reward, and the percent of steps in bounds. To clearly interpret the results of varying α_s and

α_e, all other variables are held constant, seen in Table 6.2. The distributions of results are shown in Figure 6.7, and discussed in Section 6.5.

Test				0	1	2	3	4	5	6	7	8	9	10
trainN	ε_s	0.3	α_s	1.0	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	ε_e	0.1	α_e	1.0	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6.10: Outline how the starting and ending learning values (α_s and α_e) vary for training games (trainN) as well as testing games (testN), in Game Two with Q learning, and ε-greedy.

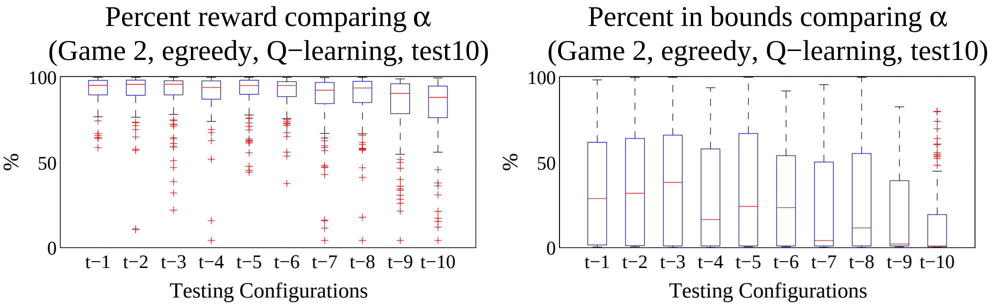


Figure 6.7: Comparing the results of varying α_s and α_e, where t-n defines the n^{th} test according to Table 6.10, consisting only of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Q-learning, and ε-greedy.

6.3.3 Q-Function Learning Rate

The testing configurations outlined in Table 6.11, show how the Q-function approximation ANN learning rate (η_Q) influences the learned policies. To clearly interpret the results of varying η_Q, all other variables are held constant, seen in Table 6.2. The distributions of results are shown in Figure 6.8, and discussed in Section 6.5.

Test	1	2	3	4	5
$\eta_{Q(s,a)}$	0.9	0.7	0.5	0.3	0.1

Table 6.11: Outlines how η_Q changes during testing, in Game Two with Q-learning, and ε-greedy.

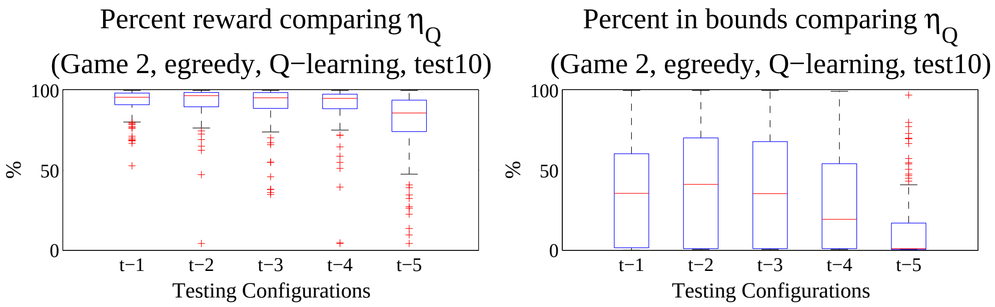


Figure 6.8: Comparing the results of varying η_Q, where t-n defines the n^{th} test according to Table 6.11, consisting of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Q-learning, and ε-greedy.

6.3.4 Individual Motive Reward

The number of testing configurations outlined in Table 6.12, show how the motive reward scaling factor (mR) influences learning. Other key variables are held constant, see Table 6.2, to clearly interpret the results of varying mR. The distributions of results are shown in Figure 6.9, and discussed in Section 6.5.

Test	1	2	3	4
mR	-0.12	-0.1	-0.08	-0.05

Table 6.12: Outlines how mR changes during testing, in Game Two with Q-learning and ε-greedy.

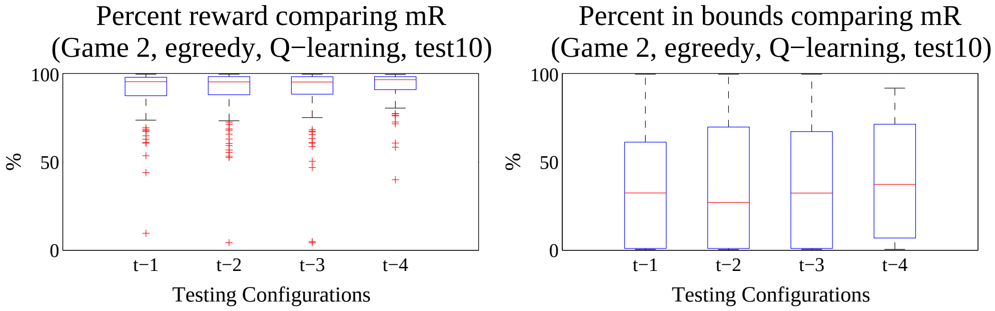


Figure 6.9: Comparing the results of varying mR, where t-n defines the n^{th} test according to Table 6.12, consisting of the final policies reached by each parameter configuration (130 results). The agent follows the policy with no exploration and no learning. The testing configurations are run in Game Two using Q-learning, and ε-greedy.

6.3.5 Q-Function Hidden Neurons

The testing configurations outlined in Table 6.13, are used to show how the Q-function approximation ANN's hidden neuron configuration (Hn_Q) influences learning. Other key variables are held constant, see Table 6.2, to clearly interpret the results of varying Hn_Q. The results are shown in Figure 6.10, and discussed in Section 6.5.

Test	1	2	3	4	
Hn	[3]	[10]	[20]	[40]	
Test	5	6	7	8	
Hn	[60]	[80]	[100]	[150]	
Test	9	10	11	12	13
Hn	[200]	[20 20]	[40 40]	[100 100]	[200 200]

Table 6.13: Outlines how Hn_Q changes during testing, in Game Two with Q learning and ε-greedy.

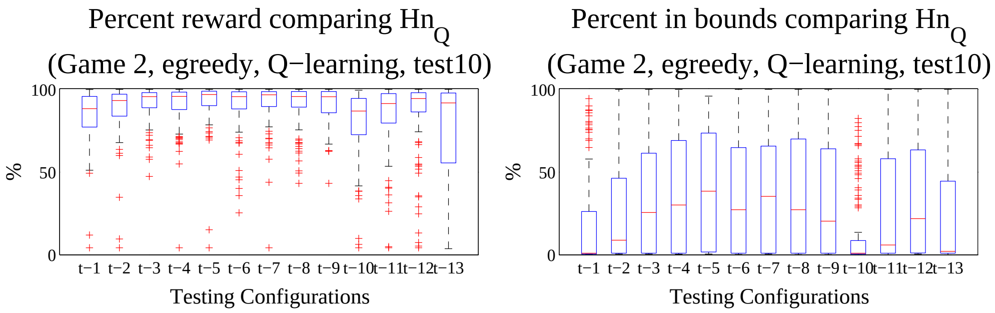


Figure 6.10: Comparing the results of varying Hn_Q, where t-n defines the n^{th} test according to Table 6.13, consisting of the final policies reached by each parameter configuration (130 results). The testing configurations are run in Game Two using Q-learning, and ε-greedy.

6.4 Dyna-Q

In the following sections, Dyna-Q is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 6.14.

Section	Test	Hn_Q	mR	η_Q	RL rate	Exploration	pS	η_M	Hn_M
6.4.1	Hn_M	[40]	-0.1	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.5$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$	3	0.6	
6.4.2	η_M	[40]	-0.1	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.5$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$	3		[3]
6.4.3	pS	[40]	-0.1	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.5$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$		0.9	[3]

Table 6.14: Outline of fixed variables used in the testing configurations for Game Two using Dyna-Q, and ε-greedy.

6.4.1 Model-Function Hidden Neurons

A number of testing configurations, outlined in Table 6.14, were used to show how the Model-function approximation ANN's hidden neuron configuration (Hn_M) influences learning. Other key variables are held constant, see Table 6.14. The distributions of results are shown in Figure 6.11, and discussed in Section 6.5.

Tests	1	2	3	4	5	6	7
Hn	[3]	[10]	[15]	[20]	[50]	[100]	[150]

Table 6.15: Outlines how Hn_M changes with different testing configurations in Game Two with Dyna-Q, and ε-greedy.

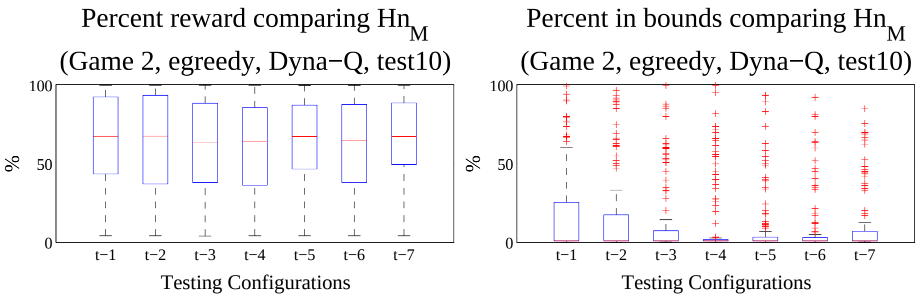


Figure 6.11: Comparing the results of varying Hn_M during final policies, where t-n defines the n^{th} test according to Table 6.15. The testing is run in Game Two using Dyna-Q, and ε-greedy.

6.4.2 Model-Function Learning Rate

The number of testing configurations, outlined in Table 6.16, show how the Model-function approximation ANN learning rate (η_M) influences the learned policies. To clearly interpret the results of varying η_M, all other variables are held constant, seen in Table 6.14. The distributions of results are shown in Figure 6.12, and discussed in Section 6.5.

Test	1	2	3	4	5
η_M	0.9	0.7	0.5	0.3	0.1

Table 6.16: Outline how η_M changes during different testing configurations in Game Two with Dyna-Q, and ε-greedy.

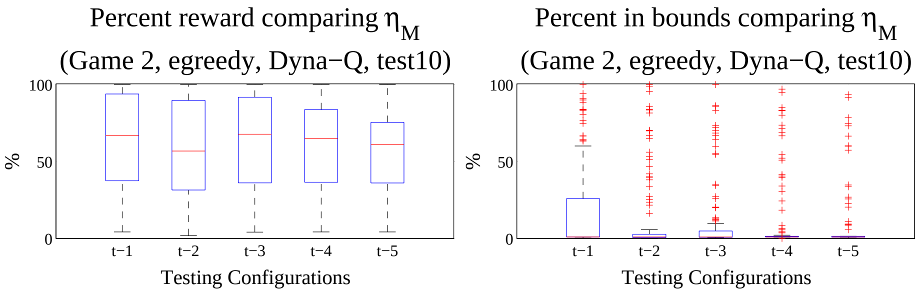


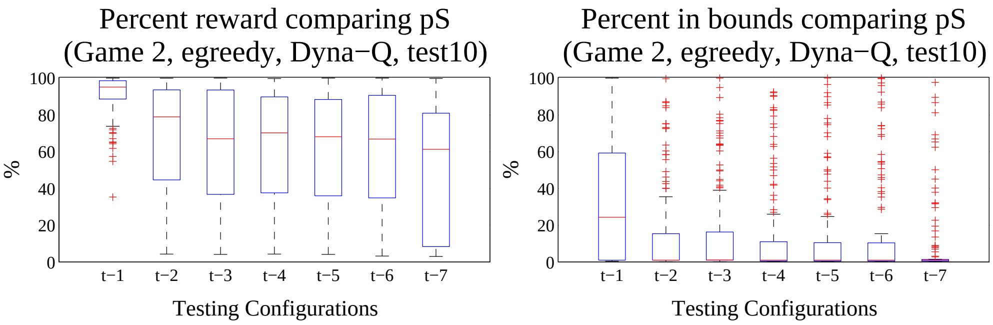
Figure 6.12: Comparing the final policies while varying η_Q, where t-n defines the n^{th} test according to Table 6.16. The testing configurations are run in Game Two using Dyna-Q, and ε-greedy.

6.4.3 Dyna-Q Planning Steps

The testing configurations outlined in Table 6.17, show how the number of planning steps (pS) has an impact on the learned policies. To clearly interpret the results of varying pS, all other variables are held constant, seen in Table 6.14. The results are shown in Figure 6.13 and the implications of the results are discussed in Section 6.5.

Test	1	2	3	4	5	6	7
pS	0	1	2	5	10	20	50

Table 6.17: Outlines how pS changes for different testing configurations in Game Two with Dyna-Q, and ε-Greedy.


Percent reward comparing pS (Game 2, egreedy, Dyna−Q, test10)

Percent in bounds comparing pS (Game 2, egreedy, Dyna−Q, test10)

Figure 6.13: Comparing the results of varying pS during final policy tests, where t-n is defined as the n^{th} testing configuration according to Table 6.17. The testing configurations are run in Game Two using Dyna-Q, and ε-greedy.

6.5 Discussion

This section evaluates the most significant variables, and comparing RL methods (Sarsa, Q-learning, and Dyna-Q).

6.5.1 Parameters

Exploration Rate Exploration rates that end with 0.1, and 0.3 over time, show higher percent of reward, and higher percent of steps in bounds, for Sarsa (Figure 6.1), and Q-learning (Figure 6.6). The optimal exploration rate was found to be $\varepsilon_s = 0.3$, and $\varepsilon_s = 0.1$.

Reinforcement Learning Rate Unlike Game One, the RL learning rate (α) for Game Two shows significant differences between high values ($\alpha_s = 1.0$ and $\alpha_e = 1.0$), and low values ($\alpha_s = 0.3$ and $\alpha_e = 0.1$), where higher values produce higher percent of reward, and higher percent of steps in bounds, see Figures 6.2, and 6.7.

Q-Function ANN Learning Rate The Q-function ANN learning rate (η_Q) has the most impact on training time, given it controls how quickly the Q-function is approximated. The results show a significant difference in percent of total reward for the final testing game, for Sarsa (Figure 6.3), and Q-learning (Figure 6.8). Performance is better when η_Q is larger. This is a strong indicator that more training is needed.

Motive Reward Factor The variable mR directly influences the magnitude of rewards received by the agent throughout a game. For Game Two, $mR = -0.1$ is the optimal value for both Sarsa (Figure 6.4), and Q-learning (Figure 6.9). The same conclusion was reached for Game One, with $mR = -0.1$ being the optimal value.

Q-Function ANN Hidden Neurons Given the more complex game scenario of Game Two, the resulting function approximation should require more hidden neurons than Game One. This is true for Q-learning results, where the optimal value is $Hn_Q = [60]$, seen in Figure 6.10. Results for Sarsa in Figure 6.5, show the optimal value is close to $Hn_Q = [10]$, the same as Game One.

Model-Function ANN Hidden Neurons The Figure 6.11 shows optimal results with $Hn_M = [3]$. The results (percent of reward, and percent of steps in bounds) are much less than those from the basic Q-learning tests previously run, indicating that the learned model does not accurately predict the game world.

Model-Function ANN Learning Rate Contrary to Game One results, testing the model function ANN learning rate η_M in Game Two showed that a higher learning rate produces a more accurate model of the environment, seen in Figure 6.12.

Dyna-Q Planning Steps As the number of planning steps increases, the performance of the agent decreases, see Figure 6.13, indicating that the model-function does not accurately approximate the game world.

6.5.2 Consistency

Testing the consistency of the results involves comparing results from repeated tests, under identical conditions. In this case, the values in Table 6.18 are used in 5 separate game tests, with results in Figure 6.14. The resulting policies have equal medians in both percent of total rewards and similar medians for percent of steps in bounds. The consistency of the results is a reflection of the optimality of the parameters being tested. In this case near optimal parameters were found.

RL Algorithm	Hn_Q	η_Q	mR	**RL Rate**	**Exploration Rate**
Q-Learning	[60]	0.7	-0.1	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.5$ $\varepsilon_e = 0.1$

Table 6.18: Outline of the variables used to test the consistency of Game Two's optimal parameters.

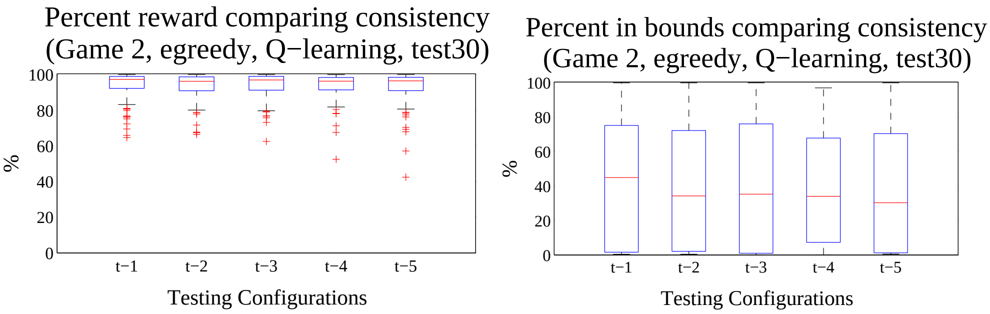


Figure 6.14: Comparing the consistency of the final policies, trained using optimal parameters outlined in Table 6.18, where t-n defines the n^{th} repeated test.

6.5.3 Discount and Trace-Decay

The trace-decay rate, and discount value, are compared given the set of optimal parameters described in Table 6.18. Graphical results in Figure 6.15 show that a compromise between long term, and short term reward, $\gamma = 0.5$, increases the percent of reward. Propagating the rewards to previous actions as much as possible ($\lambda = 0.9$) also shows an increase in percent of reward.

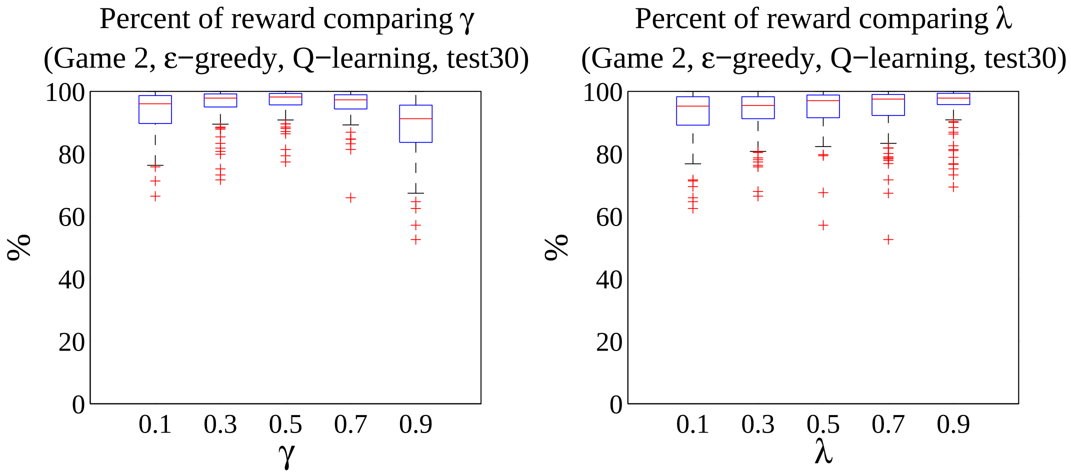


Figure 6.15: Comparing the impact of γ and λ on the percent of total reward given the set of optimal parameters found in Table 6.18.

6.5.4 Training Time of Optimal Parameters

From the results in Figure 6.16, 10 training games would have been sufficient for attaining a good policy.

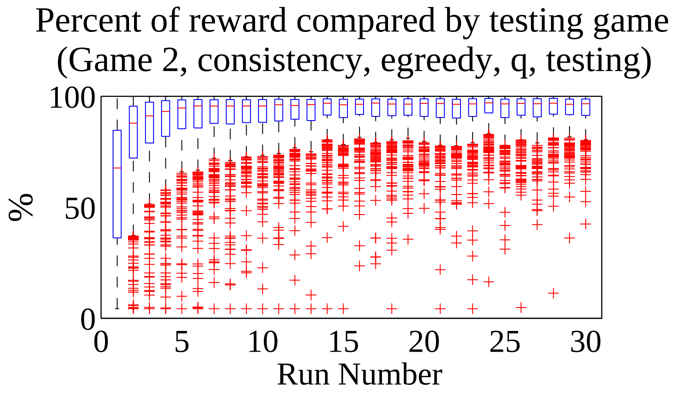


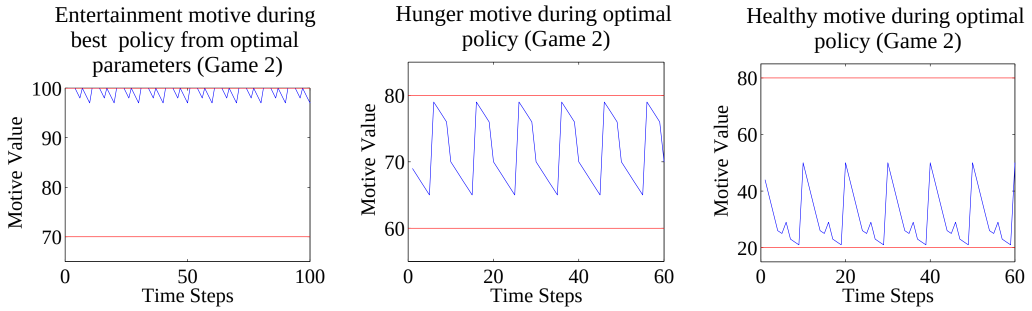
Figure 6.16: Comparing the impact of training time on the percent of reward, for policies using optimal parameters seen in Table 6.18.

6.5.5 Qualitative Analysis

Significant policies (optimal, and in-optimal) found using the parameters in Table 6.18 are analyzed for their action selection sequences. Motive values, and thresholds, are graphed for every motivation, see Figures 6.17 and 6.18.

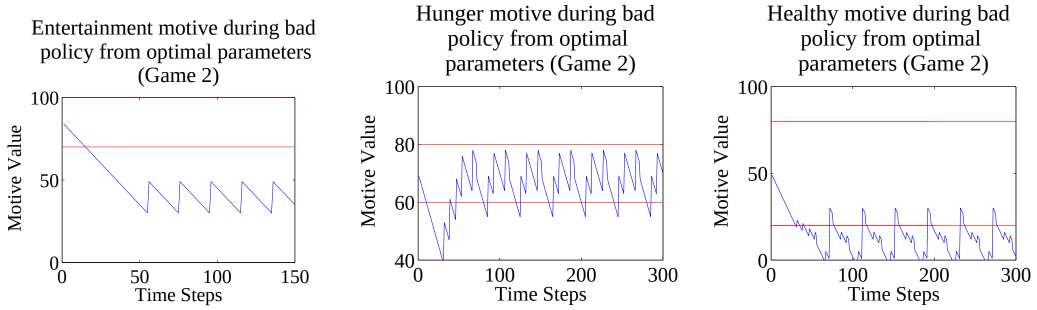
The optimal policy selected for qualitative analysis, was trained with $\gamma = 0.3$, and $\lambda = 0.1$. The agent spends most of the game watching TV. Since the entertainment motive has no maximum threshold (maximum threshold is 100), the agent can watch a lot of TV without going above its maximum desired entertainment value. However, watching TV does decrease the healthy motive value. When the healthy motive value is close to the desired minimum, the agent will stop watching TV, and instead, use the treadmill. When the agent's hunger motive is close to the minimum desirable amount, the agent will open the fridge, and eat healthy food. The agent does not eat any greasy food during the entire game. Greasy food satisfies the hunger motive less than healthy food, and also decreases the healthy motive value. With no incentive in eating greasy food (tastes good), it is understandable that the agent chooses to eat healthy food instead.

The poor policy selected for qualitative analysis, was trained with $\gamma = 0.9$, and $\lambda = 0.3$. With such a high discount rate, the agent will consider long-term rewards much more important than short-term rewards. The only action with delayed reward is *opening the fridge*, and the reward is only delayed by 2 game steps when food is eaten. In fact, the agent is continuously opening the fridge door, ignoring the other motive values that need to be satisfied. This type of behaviour is interesting, but not surprising, as there are no negative consequences associated with opening the fridge (e.g. shortening the life span of the food or using electricity).



(a) (b) (c)

Figure 6.17: Actual motive values observed during a game following one of the optimal policies found using parameters described in Table 6.18. The red lines indicate the boundaries for minimum, and maximum desirable values, that are different for each motivation.



(a) (b) (c)

Figure 6.18: Actual motive values observed during a game following one of the sub-optimal policies found using parameters described in Table 6.18. The red lines indicate the boundaries for minimum, and maximum desirable values, that are different for each motivation.

6.6 Summary

The increase in motivations for Game Two did present more of a challenge for learning, compared to those of Game One, reflected in the slightly poorer worst case results. Keep in mind overall 100% of total reward is desirable, however, it is not a requirement for the success of this method. Most of the tests include all discount values, and trace-decay rates. The results do suggest that an increase in training time would benefit the performance of less ideal parameters, particularly with lower values of η_Q. The next Chapter will introduce the concept of inter-agent interaction, and actions that impact more than one agent.

Chapter 7

Game Three

This game scenario introduces the possibility of inter-agent interactions. There are three agents: Tarzan, Jane, and Bob. Tarzan and Jane each have their own private houses. Bob is the bar tender at the local bar, and does not have the ability to leave. Both Tarzan and Jane have a *social* motivation, and a *hunger* motivation. However, Bob's only motivation is *social*. Jane and Tarzan must go to the bar for social interaction, and return to their respective houses to eat. In order to eat food, the fridge needs to be opened first.

Managing social interactions is the most difficult aspect of this game scenario. When any of the agents talk to another agent, they receive an increase of 10 in their social motivation, and also increase the social motivation of the other agent by 5. An agent must learn that *doing nothing* is not enough to decrease their social interaction level. Since other agents contribute to the increase in social interaction, agents must learn to remove themselves from the social situation, and return home where no social interaction can take place.

All agents have different minimum, and maximum thresholds, for their social motivation. This allows for a direct comparison of how different motivational profiles would learn using the same existing RL algorithm. See Table 7.1 for a more detailed description of motive thresholds for all agents.

Agent	Social		Food	
	Min	Max	Min	Max
Tarzan	20	60	40	70
Jane	50	80	40	70
Bob	95	100		

Table 7.1: Game Three agent motive thresholds.

The purpose of this learning task is to learn inter-agent interactions, compare significant variable values to Games One and Two, and to compare qualitative results from two different motivation profiles (Tarzan vs. Jane).

7.1 Testing Outline

Testing once again begins with Sarsa in Section 7.2, and continues with Q-learning in Section 7.3. Finally, Dyna-Q is explored in Section 7.4. Results and important insights are discussed in Section 7.5. All testing configurations in Game Three are repeated 5 times to measure consistency.

7.2 Sarsa

In the following sections, Sarsa is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 7.2.

Sub-Section	Testing	Hn_Q	mR	η_Q	RL Rate	Exploration
7.2.1	ε	[50]	-0.1	0.5	$\alpha_s = 0.9$ $\alpha_e = 0.7$	
7.2.2	α	[50]	-0.1	0.5		$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
7.2.3	η_Q	[50]	-0.1		$\alpha_s = 0.9$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
7.2.4	mR	[50]		0.7	$\alpha_s = 0.9$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
7.2.5	Hn_Q			0.7	$\alpha_s = 0.9$ $\alpha_e = 0.1$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$

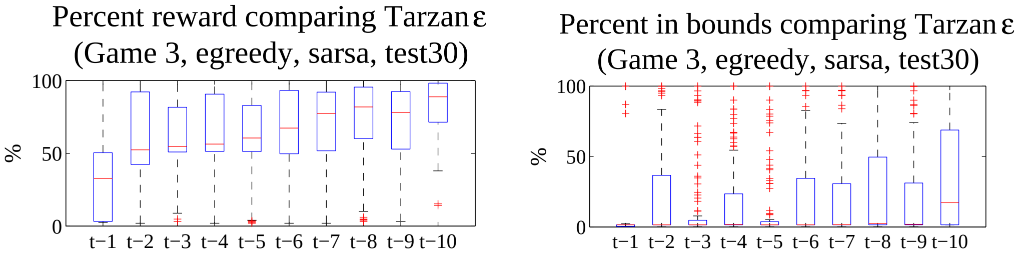
Table 7.2: Outline of fixed variables used in the testing configurations for Game Three using Sarsa, and ε-greedy.

7.2.1 Exploration Rate

The results of testing configurations outlined in Table 7.3, show how the starting, and ending exploration rates (ε_s and ε_e), impact learning. All other variables are held constant, see Table 7.2, to clearly interpret the results of varying ε_s, and ε_e. The results are shown in Figure 7.1 for Tarzan, and Figure 7.2 for Jane. Overall results are discussed in Section 7.5.

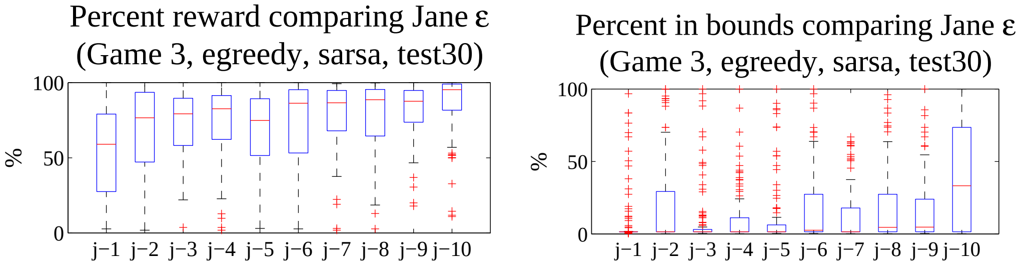
Test				1	2	3	4	5	6	7	8	9	10
trainN	α_s	0.9	ε_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	α_e	0.7	ε_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1

Table 7.3: Outlines how ε_s, and ε_e, vary during testing configurations in Game Three using Sarsa, and $\mathcal{E}$-Greedy.



(a) (b)

Figure 7.1: Comparing the results of varying ε_s, and ε_e, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.3. The testing configurations are run in Game Three using Sarsa, and ε-greedy.



(a) (b)

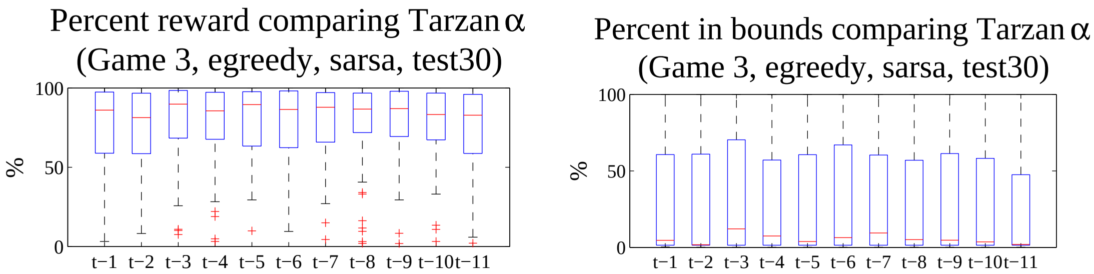
Figure 7.2: Comparing the results of varying ε_s, and ε_e, for the agent Jane, where j-n defines the n^{th} test according to Table 7.3. The testing configurations are run in Game Three using Sarsa, and ε-greedy.

7.2.2 Reinforcement Learning Rate

The testing configurations outlined in Table 7.4 are meant to show the impact of the RL learning rate (α) on the percent of total reward, and the percent of steps in bounds. To clearly interpret the results of varying α_s, and α_e, all other variables are held constant, see Table 7.2. The results are shown in Figure 7.3 for Tarzan, and Figure 7.4 for Jane. Overall results are discussed in Section 7.5.

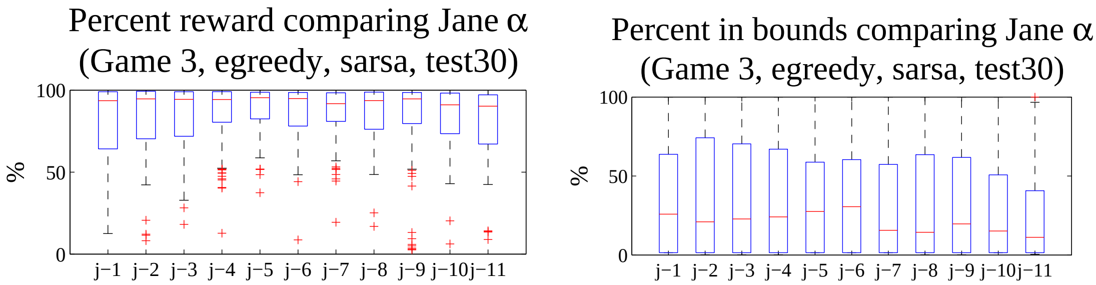
Test				1	2	3	4	5	6	7	8	9	10	11
trainN	ε_s	0.3	α_s	1.0	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	ε_e	0.1	α_e	1.0	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 7.4: Outline how the starting, and ending learning values (α_s and α_e), vary for training games (trainN), as well as testing games (testN), in Game Three with Sarsa, and ε-greedy.



(a) (b)

Figure 7.3: Comparing the results of varying α_s, and α_e, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.4. The testing configurations are run in Game Three using Sarsa, and ε-greedy.



(a) Percent of reward for Jane. (b) Percent in bounds for Jane.

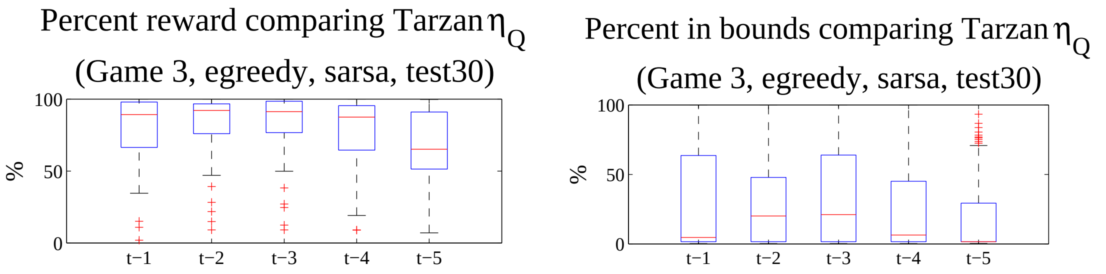
Figure 7.4: Comparing the results of varying α_s, and α_e, for the agent Jane, where j-n defines the n^{th} test according to Table 7.4. The testing configurations are run in Game Three using Sarsa, and ε-greedy.

7.2.3 Q-Function Learning Rate

The set of testing configurations seen in Table 7.5, are designed to show how the Q-function ANN learning rate (η_Q) changes the learned policies. To clearly interpret the results of varying η_Q, all other variables are held constant, seen in Table 7.2. The results are shown in Figure 7.5 for Tarzan, and Figure 7.6 for Jane. Overall results are discussed in Section 7.5.

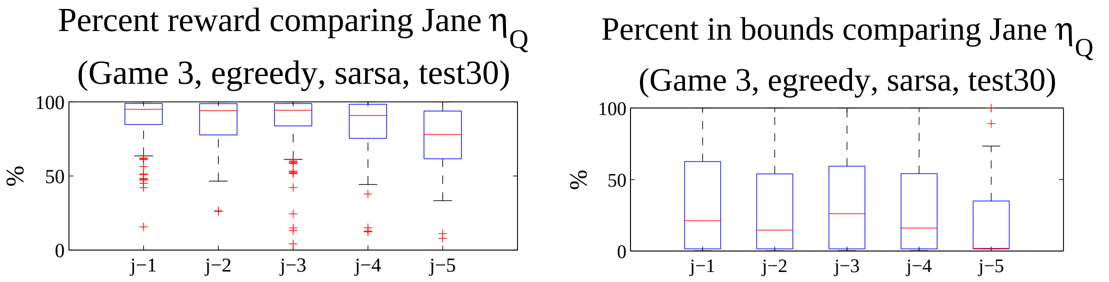
Test	1	2	3	4	5
η_Q	0.9	0.7	0.5	0.3	0.1

Table 7.5: Outlines how η_Q changes during testing configurations in Game Three with Sarsa, and ε-Greedy.



(a) (b)

Figure 7.5: Comparing the results of changing η_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.5. The testing configurations are run in Game Three using Sarsa, and ε-greedy.



(a) (b)

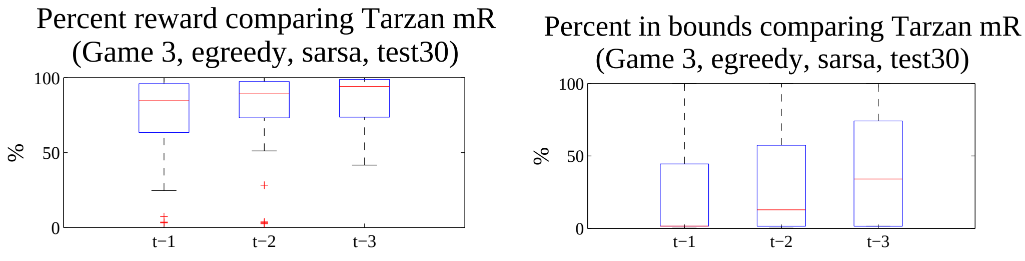
Figure 7.6: Comparing the results of changing η_Q, for the agent Jane, where j-n defines the n^{th} test according to Table 7.5. The testing configurations are run in Game Three using Sarsa, and ε-greedy.

7.2.4 Individual Motive Reward

The set of testing configurations outlined in Table 7.6, are intended to show how the motive reward scaling factor (mR) influence learning. Other key variables are held constant, see Table 7.2. The results are shown in Figure 7.7 for Tarzan, and from in Figure 7.8 for Jane. Overall results are discussed in Section 7.5.

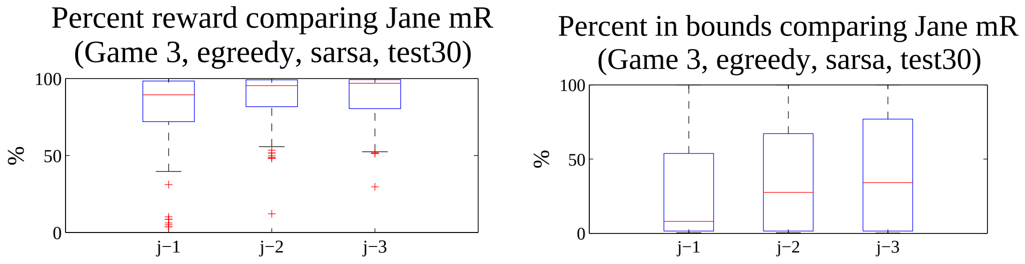
Test	1	2	3
mR	-0.15	-0.1	-0.05

Table 7.6: Outlines mR changes for different testing configurations in Game Three with Sarsa, and ε-Greedy.



(a) (b)

Figure 7.7: Comparing the results of changing mR, from the agent Tarzan, where t-n defines the n^{th} test according to Table 7.6. The testing configurations are run in Game Three using Sarsa, and ε-greedy.



(a) (b)

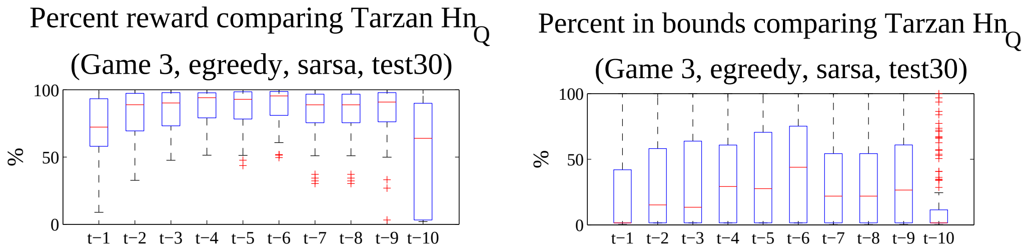
Figure 7.8: Comparing the results of changing mR, from the agent Jane, where j-n defines the n^{th} test according to Table 7.6. The testing configurations are run in Game Three using Sarsa, and ε-greedy.

7.2.5 Q-Function Hidden Neurons

To show the impact of the Q-function ANN hidden neuron configuration rate (Hn_Q) on the results, a number variable configurations were tested, see Table 7.7Other necessary variables are held constant, seen in Table 7.2. The results are shown in Figure 7.9 for Tarzan, and Figure 7.10 for Jane. Overall results are discussed in Section 7.5.

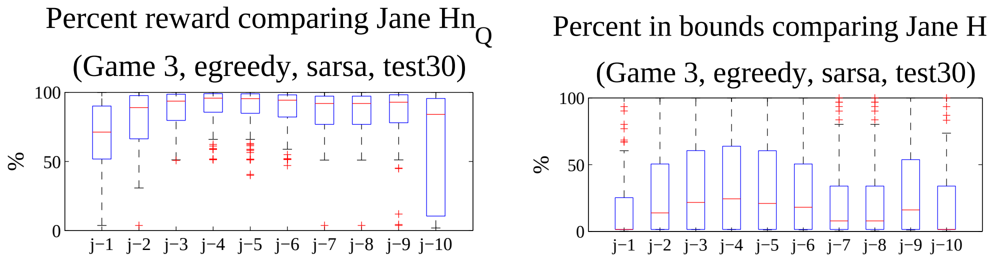
Test	1	2	3	4	5	6	7
Hn	[3]	[10]	[25]	[50]	[75]	[100]	[250]
Test	8		9		10		
Hn	[50 50]		[100 100]		[250 250]		

Table 7.7: Outlines how Hn_Q changes for different testing configurations in Game Three with Sarsa, and ε-greedy.



(a) (b)

Figure 7.9: Comparing the results of changing Hn_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.7. The testing configurations are run in Game Three using Sarsa, and ε-greedy.



(a) (b)

Figure 7.10: Comparing the results of changing Hn_Q, for the agent Jane, where j-n defines the n^{th} test according to Table 7.7. The testing configurations are run in Game Three using Sarsa, and ε-greedy.

7.3 Q Learning

In the following sections, Q-learning is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 7.8.

Sub-Section	Testing	Hn_Q	mR	η_Q	RL Rate	Exploration Rate
7.3.1	ε	[50]	-0.1	0.5	$\alpha_s = 0.9$ $\alpha_e = 0.7$	
7.3.2	α	[50]	-0.1	0.5		$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
7.3.3	η_Q	[50]	-0.1		$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
7.3.4	mR	[50]		0.7	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$
7.3.5	Hn_Q		-0.05	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$

Table 7.8: Outline of the fixed variables used in the testing configurations for Game Three using Q-learning, and ε-greedy.

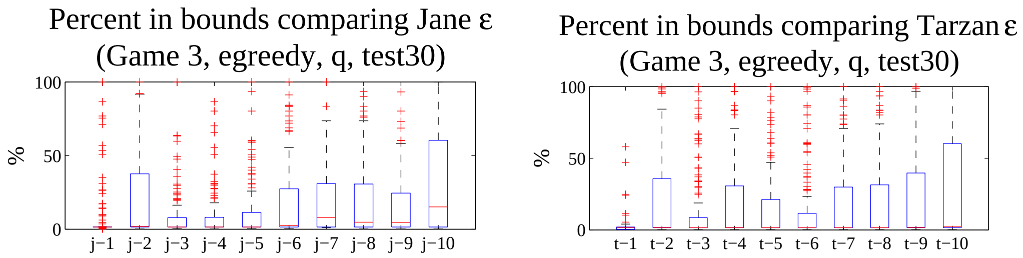
للاستشارات المنارة

7.3.1 Exploration Rate

A number of testing configurations, outlined in Table 7.9, are intended to show how the starting, and ending exploration rates (ε_s and ε_e), influence learning. All other variables are held constant, seen in Table 7.8, to clearly interpret the results of varying ε_s and ε_e. The results are shown in Figure 7.11b for Tarzan, and Figure 7.11a for Jane. Overall results are discussed in Section 7.5.

Test				1	2	3	4	5	6	7	8	9	10
trainN	α_s	0.9	ε_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	α_e	0.7	ε_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1

Table 7.9: Outline how ε_s and ε_e vary during testing configurations for Game Three tests with Q-learning and ε-greedy.



(a) (b)

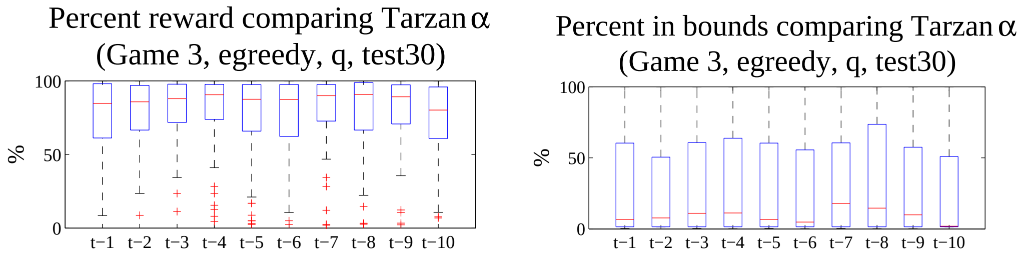
Figure 7.11: The results of varying ε, where t-n and j-n are the n^{th} test from Table 7.9. The testing configurations are run in Game Three using Q-learning, and ε-greedy.

7.3.2 Reinforcement Learning Rate

The testing configurations in Table 7.10 are meant to show the impact of the RL learning rate (α) on the percent of total reward, and the percent of steps in bounds. To clearly interpret the results of varying α_s, and α_e, all other variables are held constant, seen in Table 7.8. The results are shown in Figure 7.12 for Tarzan, and from Jane in Figure 7.13. Overall results are discussed in Section 7.5.

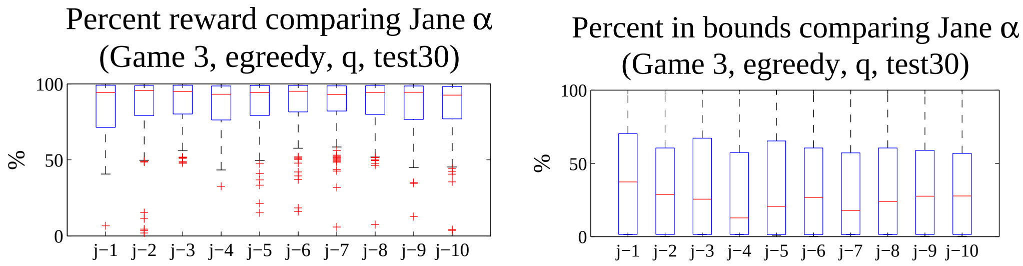
Test				1	2	3	4	5	6	7	8	9	10
trainN	ε_s	0.3	α_s	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.5	0.5	0.3
	ε_e	0.1	α_e	0.7	0.5	0.3	0.1	0.5	0.3	0.1	0.3	0.1	0.1
testN	ε_s	0.0	α_s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ε_e	0.0	α_e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 7.10: Outline how the starting, and ending learning values (α_s and α_e), vary for training games (trainN), as well as testing games (testN), in Game Three with Q learning, and ε-greedy.



(a) (b)

Figure 7.12: Comparing the results of varying α_s, and α_e, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.10. The testing configurations are run in Game Three using Q-learning, and ε-greedy.



(a) (b)

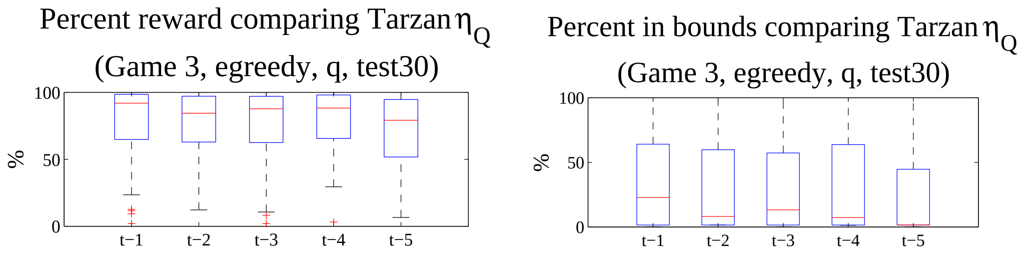
Figure 7.13: Comparing the results of varying α_s, and α_e, for the agent Jane, where j-n defines the n^{th} test according to Table 7.10. The testing configurations are run in Game Three using Q-learning, and ε-greedy.

7.3.3 Q-Function Learning Rate

The testing configurations outlined in Table 7.11, are meant to show how the Q-function approximation ANN learning rate (η_Q) influences the learned policies. To clearly interpret the results of varying η_Q, all other variables are held constant, seen in Table 7.8. The results are shown in Figure 7.14 for Tarzan, and Figure 7.15 for Jane. Overall results are discussed in Section 7.5.

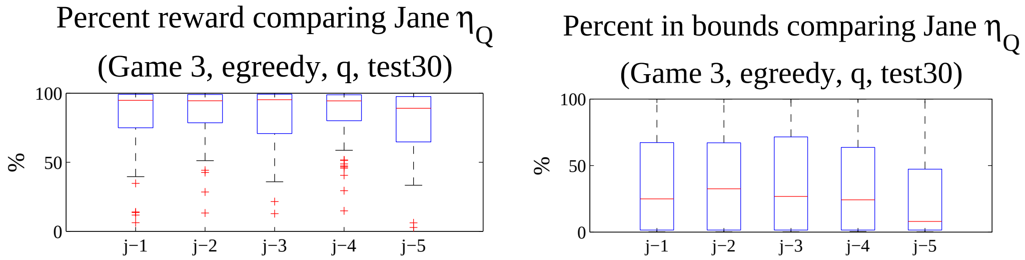
Test	1	2	3	4	5
η_Q	0.9	0.7	0.5	0.3	0.1

Table 7.11: Outlines how η_Q changes during testing configurations in Game Three with Q-learning, and ε-greedy.



(a) (b)

Figure 7.14: Comparing the results of changing η_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.11. The testing configurations are run in Game Three using Q-learning, and ε-greedy.



(a) (b)

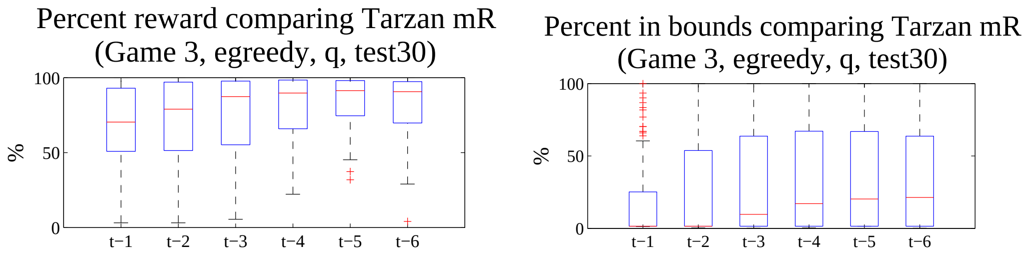
Figure 7.15: Comparing the results of changing η_Q, for the agent Jane, where j-n defines the n^{th} test according to Table 7.11. The testing configurations are run in Game Three using Q-learning, and ε-greedy.

7.3.4 Individual Motive Reward

The number of testing configurations outlined in Table 7.12, are meant to show how the motive reward scaling factor (mR) influences learning. Other key variables are held constant, see Table 7.2, to clearly interpret the results of varying mR. The results are shown in Figure 7.16 for Tarzan, and Figure 7.17 for Jane. Overall results are discussed in Section 7.5.

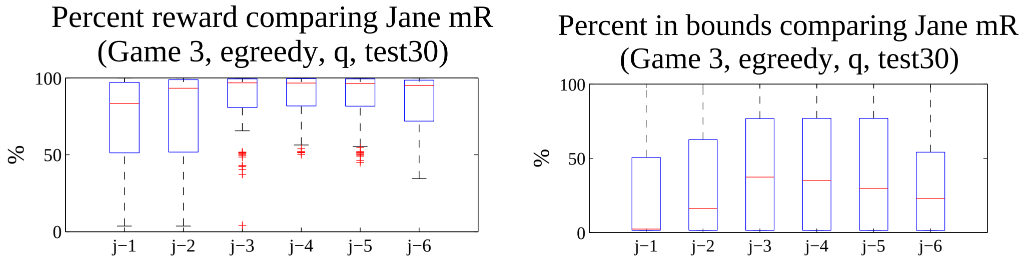
Test	1	2	3	4	5	6
mR	-0.3	-0.2	-0.1	-0.05	-0.01	-0.005

Table 7.12: Outlines how mR changes during testing, in Game Three with Q-learning and ε-greedy.



(a) (b)

Figure 7.16: Comparing the results of changing mR, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.12. The testing configurations are run in Game Three using Q-learning, and ε-greedy.



(a) (b)

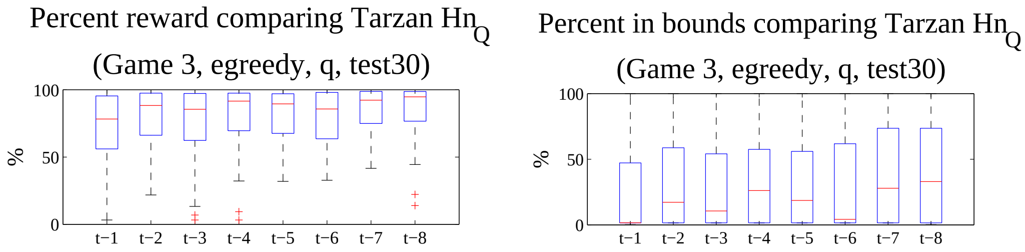
Figure 7.17: Comparing the results of changing mR, for the agent Jane, where j-n defines the n^{th} test according to Table 7.12. The testing configurations are run in Game Three using Q-learning, and ε-greedy.

7.3.5 Q-Function Hidden Neurons

The testing configurations outlined in Table7.13, are used to show how the Q-function approximation ANN's hidden neuron configuration (Hn_Q) influences learning. Other key variables are held constant, see Table 7.18, to clearly interpret the results of varying Hn_Q. The results are shown in Figure 7.19 for Tarzan, and Figure 7.17 for Jane. Overall results are discussed in Section 7.5.

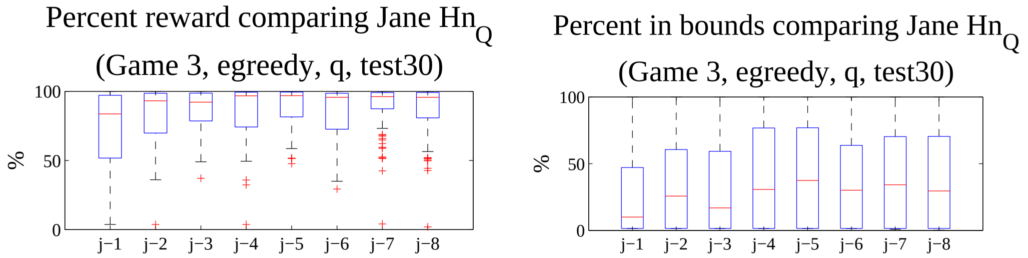
Test	1	2	3	4
Hn	[3]	[10]	[20]	[30]
Test	5	6	7	8
Hn	[40]	[50]	[75]	[100]

Table 7.13: Outlines how Hn_Q changes during testing in Game Three using Q-learning, and ε-greedy.



(a) (b)

Figure 7.18: Comparing the results of changing Hn_Q, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.13, consisting of the final policies reached by each parameter configuration (130 results). The testing configurations are run in Game Three using Q-learning, and ε-greedy.



(a) (b)

Figure 7.19: Comparing the results of changing Hn_Q, for the agent Jane, where t-n defines the n^{th} test according to Table 7.13, consisting of the final policies reached by each parameter configuration (130 results). The testing configurations are run in Game Three using Q-learning, and ε-greedy.

7.4 Dyna-Q

In the following sections, Dyna-Q is tested with the goal of determining what parameters produce significant results. The variables that remain fixed during testing are outlined in Table 7.14.

Section	Test	Hn_Q	mR	η_Q	RL Rate	Exploration Rate	pS	η_M	Hn_M
7.4.1	Hn_M	[40]	-0.05	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$	10	0.5	
7.4.2	η_M	[40]	-0.05	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$	10		[5]
7.4.3	pS	[40]	-0.05	0.7	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$		0.9	[5]

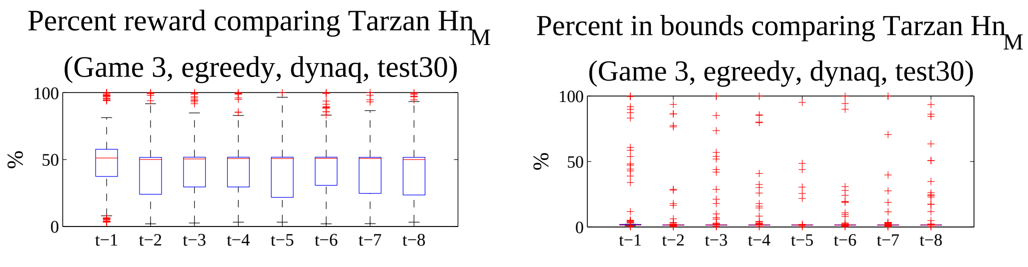
Table 7.14: Outline of fixed variables used in the testing configurations for Game Three using Dyna-Q, and ε-greedy.

7.4.1 Model Function Hidden Neurons

The testing configurations outlined in Table 7.15 are used to show how the Model-function approximation ANN's hidden neuron configuration (Hn_M) influences learning. Other key variables are held constant, see Table 7.14. The results are shown in Figure 7.20 for Tarzan, and Figure 7.21 for Jane. Overall results are discussed in Section 7.5.

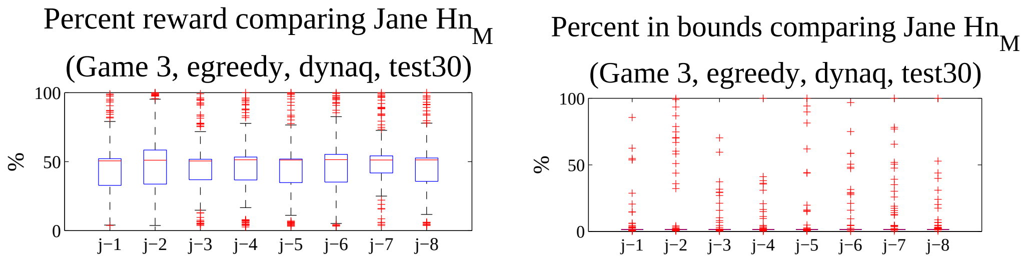
Test	1	2	3	4	5	6	7	8
Hn	[3]	[5]	[10]	[15]	[25]	[50]	[75]	[100]

Table 7.15: Outlines how Hn_M changes with different testing configurations in Game Three with Dyna-Q, and ε-greedy.



(a) (b)

Figure 7.20: Comparing the results of changing Hn_M, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy.



(a) (b)

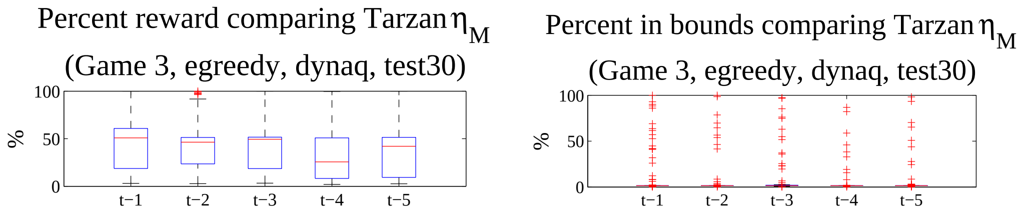
Figure 7.21: Comparing the results of changing Hn_M, for the agent Jane, where j-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy.

7.4.2 Model Function Learning Rate

The number of testing configurations outlined in Table 7.16, are meant to show how the Model-function approximation ANN learning rate (η_M) influences the learned policies. To clearly interpret the results of varying η_M, all other variables are held constant, seen in Table 7.14. The results are shown in Figure 7.22 for Tarzan, and from in Figure 7.23 for Jane. Overall results are discussed in Section 7.5.

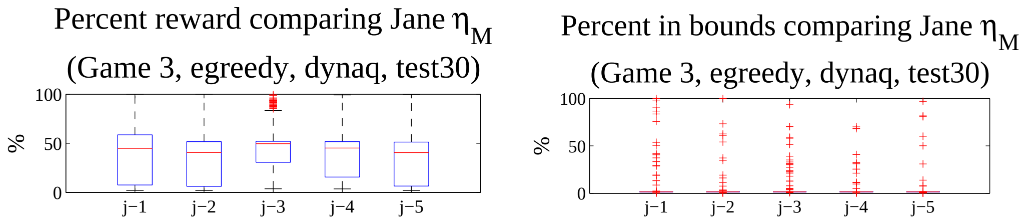
Test	1	2	3	4	5
η_M	0.9	0.7	0.5	0.3	0.1

Table 7.16: Outline how η_M changes during different testing configurations in Game Three with Dyna-Q, and ε-greedy.



(a) (b)

Figure 7.22: Comparing the results of changing η_M, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy.



(a) (b)

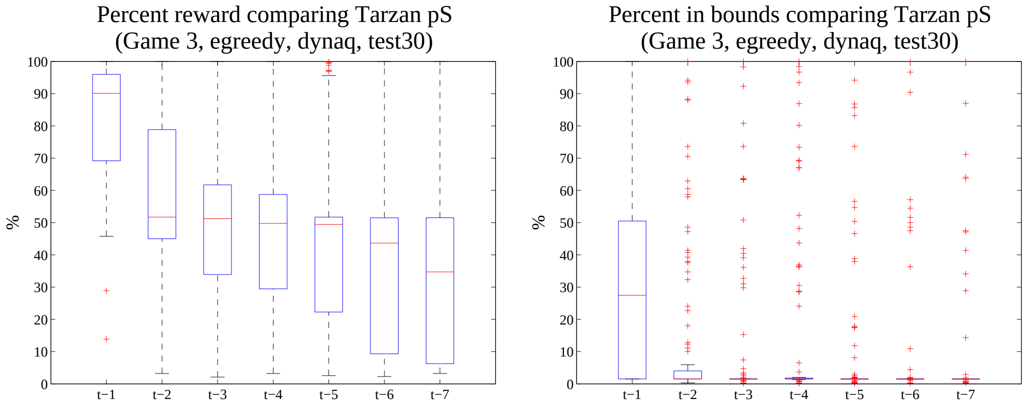
Figure 7.23: Comparing the results of changing η_M, for the agent Jane, where j-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy.

7.4.3 Dyna-Q Planning Steps

The testing configurations outlined in Table 7.17, are meant to show how the number of planning steps (pS) has an impact on the learned policies. To clearly interpret the results of varying pS, all other variables are held constant, seen in Table 5.13. The results are shown in Figure 7.22 for Tarzan, and Figure 7.23 for Jane. Overall results are discussed in Section 7.5.

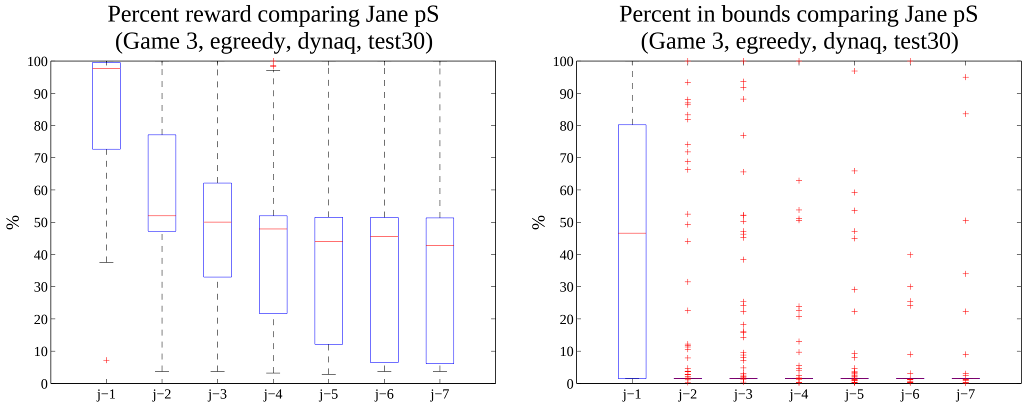
Test	1	2	3	4	5	6	7
pS	0	1	2	5	10	20	50

Table 7.17: Outlines how pS changes for different testing configurations in Game Three with Dyna-Q, and ε-Greedy.



(a) (b)

Figure 7.24: Comparing the results of changing pS, for the agent Tarzan, where t-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy.



(a) (b)

Figure 7.25: Comparing the results of changing pS, for the agent Jane, where j-n defines the n^{th} test according to Table 7.15. The testing configurations are run in Game Three using Dyna-Q, and ε-greedy.

7.5 Discussion

The added complexity of Game Three over Games One, and Two, make a direct comparison hard to make. This section will evaluate the most significant variables, compare the results of different RL methods (Sarsa, Q-learning, and Dyna-Q), and compare the difference in results between agents.

The difficulty in this scenario is not the delayed reward associated with opening the fridge, but rather the fact that an agent could be doing nothing, and still gain social interaction through other agent's actions.

7.5.1 Parameters

Exploration Rate The ε-greedy action selection algorithm showed predictable results with the optimal value being $\varepsilon_s = 0.3$ and $\varepsilon_e = 0.1$, for Sarsa (Figure 7.2) and Q-learning (Figure 7.11). The results from the agent Jane, show slightly higher performance over the results from the agent Tarzan, regardless of reinforcement learning algorithm. The difference in results between agents is caused by different threshold values.

Reinforcement Learning Rate The results comparing the learning rate for Sarsa, shown in Figures 7.3, and 7.4, and Q-learning, shown in Figures 7.12, and 7.15, show very little performance difference between values of α_s, and α_e. In these tests, the agent Jane shows higher performance than the agent Tarzan, in both Sarsa, and Q-learning.

Q-Function ANN Learning Rate Results from using Sarsa (Figure 7.5 for Tarzan, and Figure 7.6 for Jane) show near equal median percent of reward, for $\eta_Q = 0.9$, $\eta_Q = 0.7$, $\eta_Q = 0.5$, and $\eta_Q = 0.3$. Results from using Q-learning show similar results, with slightly more variation (Figure 7.14 for Tarzan, and Figure 7.15 for Jane). The results for Jane consistently have higher performance (percent of reward, and percent of steps in bounds) than results for Tarzan.

Motive Reward Factor The minimum reward received by the agent from one of their motivations does significantly affect the resulting percentages if $mR < -0.1$, see Figures 7.8 and 7.17. Note that Tarzan and Jane have different optimal values, caused by their different motivation thresholds.

Q-Function ANN Hidden Neurons Testing has shown that it is best to use only one hidden neuron layer. With this game scenario, good policies are found with at least 10 hidden neurons, and up to 100 hidden neurons.

Dyna-Q All the results associated with Dyna-Q, in Figures 7.20, 7.21, 7.22, 7.23, 7.24, and 7.25, show worst performance than those of Q-learning, indicating that only a few of the parameter configurations were able to accurately train the model-function approximation. There are still certain parameter configurations that reach 100% of total reward with Dyna-Q, indicating the model function approximation was correct in some cases.

7.5.2 Consistency

Testing the consistency of the results involved repeating tests under identical conditions, and comparing the results. In this case, the values in Table 7.18 are used in 5 separate game tests, with results in Figures 7.26, and 7.27. The results show similar median percent of rewards, but with a larger variation in result for percent of steps in bounds.

RL Algorithm	Hn_Q	η_Q	mR	**RL Rate**	**Exploration Rate**
Q learning	[100]	0.7	-0.05	$\alpha_s = 0.9$ $\alpha_e = 0.7$	$\varepsilon_s = 0.3$ $\varepsilon_e = 0.1$

Table 7.18: Outline of the variables used to test the consistency of Game Three's optimal parameters.

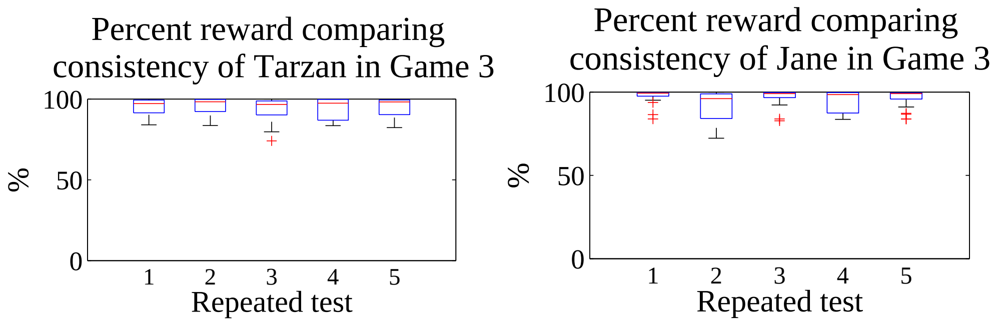


Figure 7.26: Comparing the percent of reward from policies trained using optimal parameters outlined in Table 7.18.

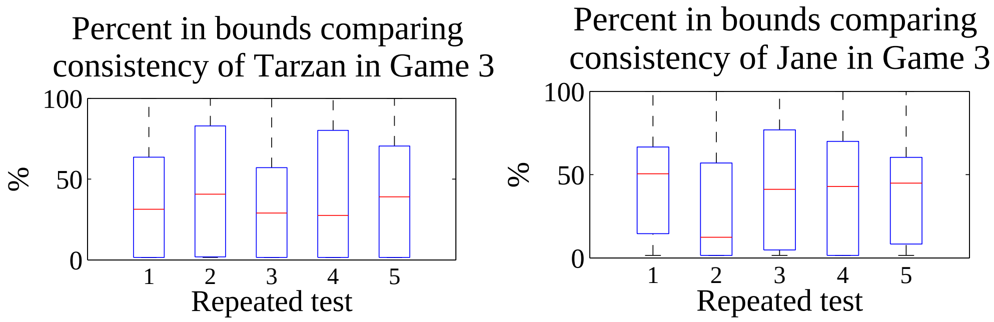
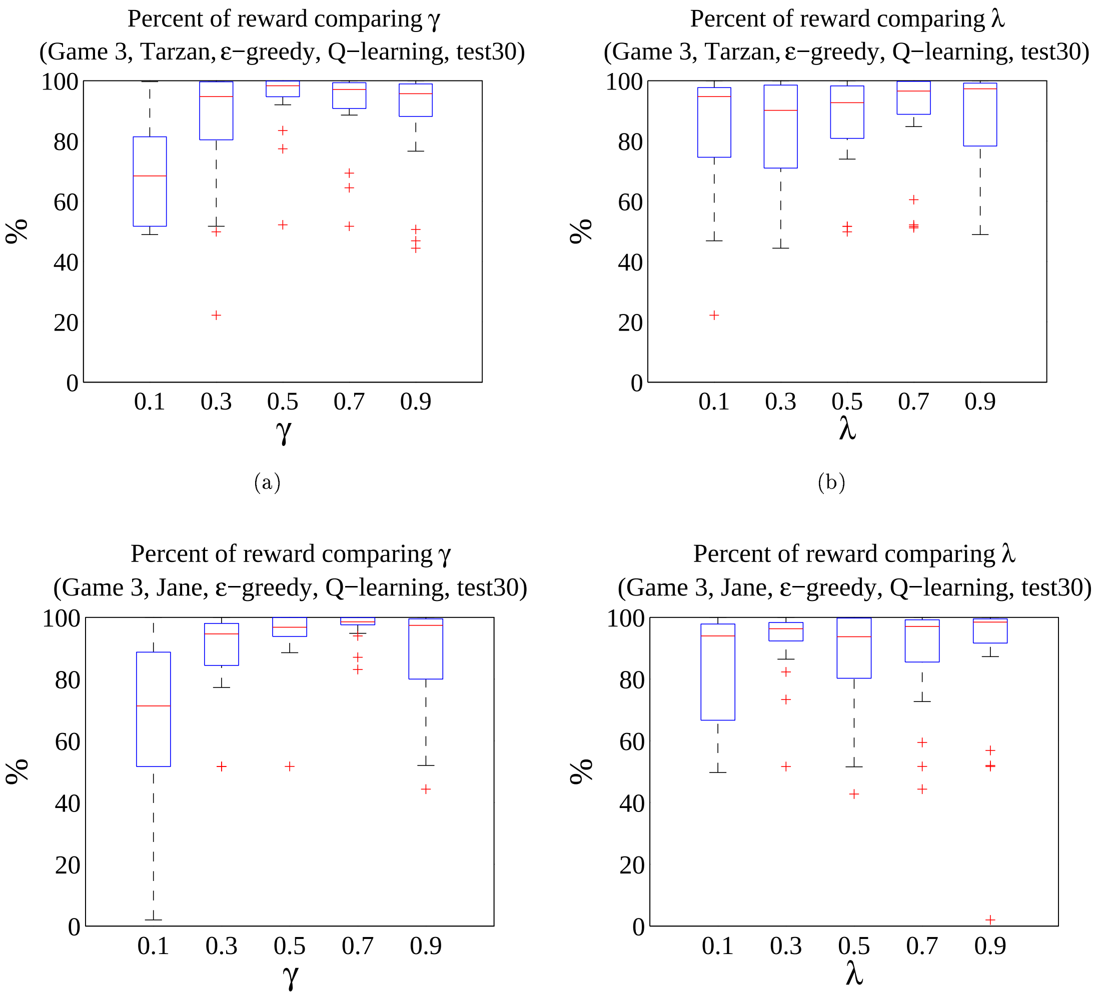


Figure 7.27: Comparing the percent of steps in bounds from policies trained using optimal parameters outlined in Table 7.18.

7.5.3 Discount and Decay-Trace

The policies trained using optimal parameters outlined in Table 7.18 are used to compare the decay-trace rate, and discount values. This game scenario is fairly simple with regards to delayed rewards and number of motives. Notice that all trace-decay values, and discount rates, show a maximum of 100% percent of reward. For Tarzan, and Jane, $\lambda = 0.9$ produces the higher median percent of reward. The highest median percent of reward is $\gamma = 0.5$ for Tarzan, and $\gamma = 0.7$ for Jane.



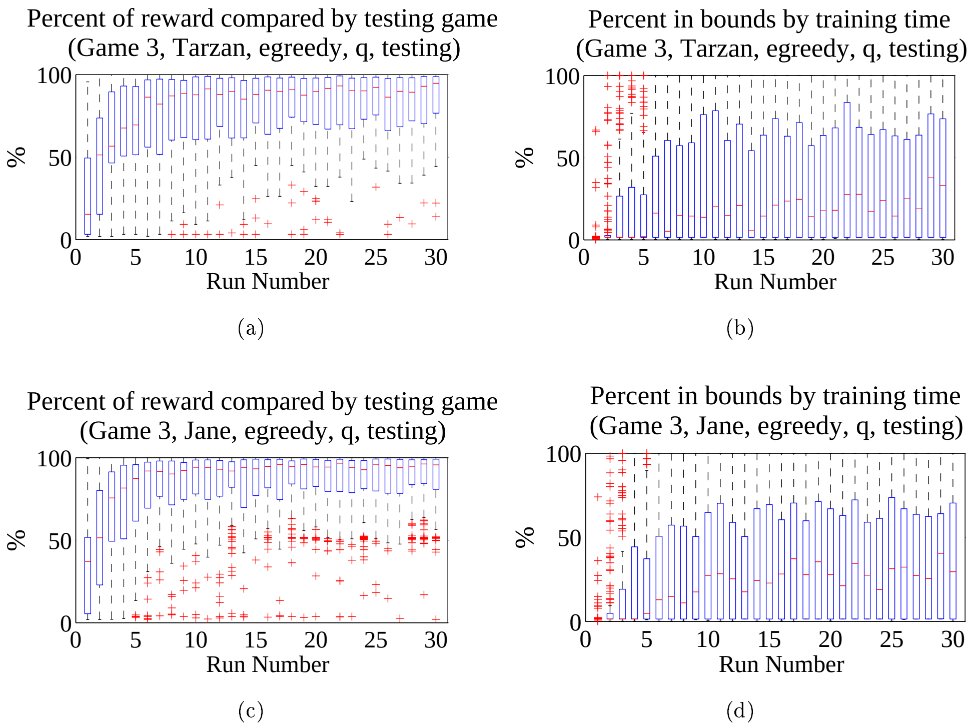
(a) (b)

(c) (d)

Figure 7.28: Comparing the impact of γ and λ on the percent of reward for Tarzan, and Jane, given the optimal set of parameters in Table 7.18.

7.5.4 Training Time

The policies found using the parameters in Table 7.18 are graphed according to training game. From the results in Figure 7.29, the policies do not appear to improve past the 15th training game.



Percent of reward compared by testing game (Game 3, Tarzan, egreedy, q, testing)

Percent in bounds by training time (Game 3, Tarzan, egreedy, q, testing)

(a)

(b)

Percent of reward compared by testing game (Game 3, Jane, egreedy, q, testing)

Percent in bounds by training time (Game 3, Jane, egreedy, q, testing)

(c)

(d)

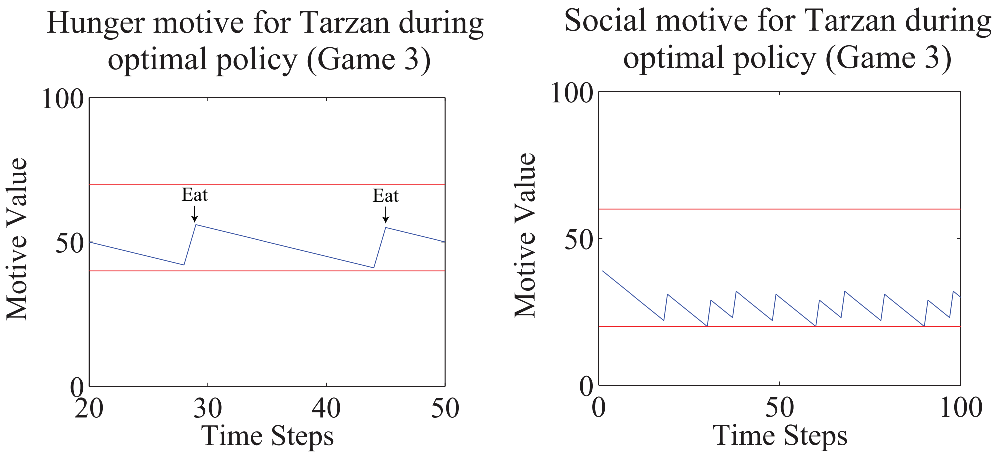
Figure 7.29: Comparing how of training time influences the percent of reward, and the percent of steps in bounds, for Tarzan, and Jane, using the optimal Game Three parameter in Table 7.18.

7.5.5 Qualitative Analysis

Significant results (optimal, and non-optimal), from parameters listed in Table 7.18, are analyzed by their action selection sequences, rather than percent of reward.

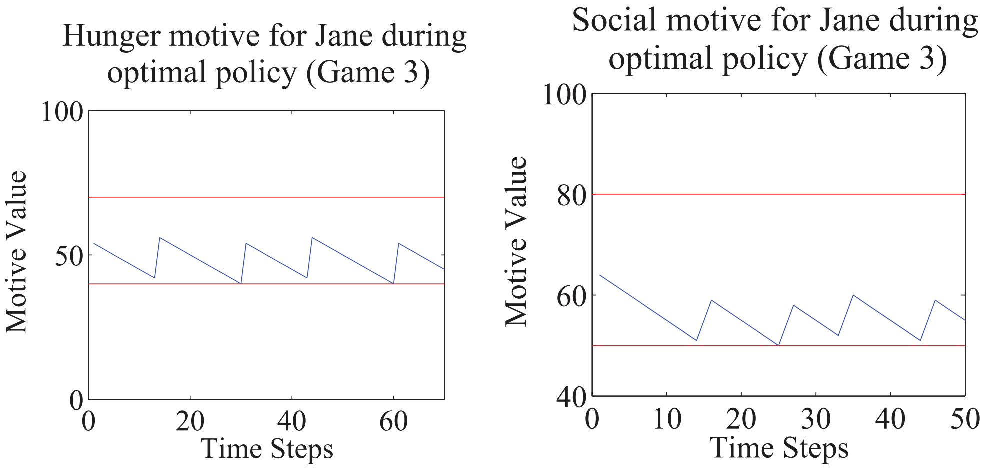
		Optimal	Non-Optimal
Tarzan	γ	0.5	0.7
	λ	0.9	0.1
Jane	γ	0.5	0.1
	λ	0.9	0.7

Table 7.19: This table outlines exactly what discount rate, and trace-decay parameter, were used to train the policies chosen for qualitative analysis.



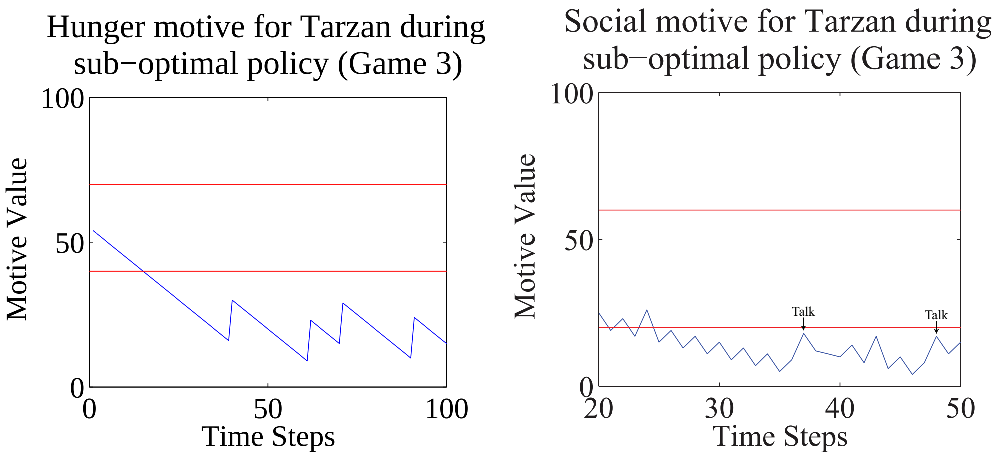
(a) (b)

Figure 7.30: Actual motive values for the agent Tarzan, where the optimal policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values.



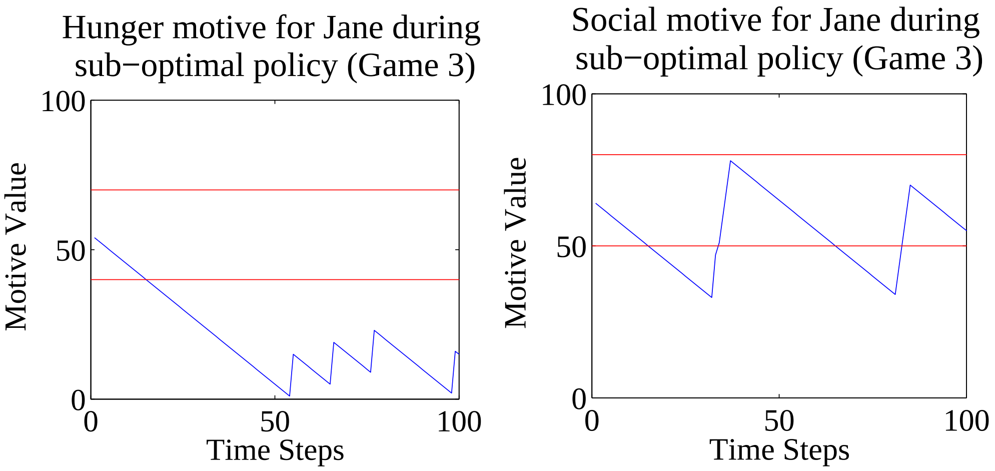
(a) (b)

Figure 7.31: Actual motive values for the agent Jane, where the optimal policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values.



(a) (b)

Figure 7.32: Actual motive values for the agent Tarzan, where the terrible policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values.



(a) (b)

Figure 7.33: Actual motive values for the agent Jane, where the terrible policy was found using parameters described in Table 7.18. The red horizontal lines show the boundaries for minimum, and maximum desirable values.

In the case of the optimal policies, both Jane and Tarzan are not spoken to by another agent, shown in Figures 7.30 and 7.31. Both agents go to the bar, talk to Bob, and immediately return home. Both agents are repeatedly opening the fridge, making food accessible, and allowing the motive values to decay naturally. Since there are no negative consequences associated with opening the fridge (spoiled food), this learned sequence of actions is not surprising.

During the non-optimal policy for Tarzan, he rarely talks to other agents, and instead allows them to talk to him. Where the rise in social motive is particularly high, Bob and Jane talk to Tarzan at the same time, see Figure 7.32b. Tarzan is constantly moving between his house, and the bar, perhaps looking for other agents that will to talk to him, since he won't talk to them.

The non-optimal policy for Jane shows her only going to the bar when her social motive is as low as 33, and then having a conversation with Bob (Jane talking to Bob, and Bob talking to Jane). Jane returns home when her social motive value becomes close to the maximum, see Figure 7.33b. Otherwise Jane spends the majority of her time doing nothing, and occasionally opening her fridge to eat food.

7.6 Summary

The introduction of inter-agent interactions has added a different complexity to the game. The number of other agents occupying the same space is important in learning why a motive is increasing regardless of the agents actions. The small variation in results, between Tarzan, and Jane, suggests that different minimum, and maximum thresholds, have small influence on the required parameters of the agents.

DynaQ planning shows a reduction in the median percent reward, and median percent in bounds, over Sarsa, and Q-learning. The best case results with Dyna-Q have 100% of steps in bounds, even with $pS = 50$, suggesting that Dyna-Q only reduces the performance in cases where the model is learned incorrectly. Using Dyna-Q will not be practical with increasingly complex game scenarios, because the worst case results should be reduced, even at the expense of slightly lower best case results.

Chapter 8

Game Four

The fourth game scenario adds complexity by introducing an action with extremely delayed reward. The agent must go to the grocery store, buy food, return home, open the fridge, and finally, eat the food. All five steps must be completed to increase the current hunger value. Tarzan's hunger motive has a minimum of 60, and maximum of 80. Tarzan can move between his house, and the grocery store. Only one food item can be bought at any time, and a food item must belong to Tarzan (ownership) before he can eat it. Tarzan has access to eight actions that are listed in Table 8.1.

Object or Place	**Action**	**Affects**			
		Food	**Position**	**Ownership**	**More Objects**
Healthy food	Eat	+15	-	-	-
	Buy	-	food to fridge	Tarzan	-
Unhealthy food	Eat	+10	-	-	-
	Buy	-	food to fridge	Tarzan	-
Fridge	Open	-	-	-	Access objects in fridge
Grocery store	Go to	-	agent to store	-	-
Tarzan's home	Go to	-	agent to home	-	-
-	Nothing	-	-	-	-

Table 8.1: Description of Tarzan's actions, and the resulting changes in the game world, for Game Four.

8.1 Testing Outline

Using the insights gained from previous scenarios, Game Four tests complete combinations of a few specifically chosen variables. Testing combinations of values allows the graphical analysis to show interactions between parameters. Given the increased complexity of Game Four, Softmax action selection is tested along with ε-greedy.

Initial results are analyzed with box-plots, showing the distribution of results across multiple parameter values. Further analysis is done using scatter plots, to see the actual results, and interactions between parameters.

Most variables, once set, remain constant throughout the various training games, with the exception of the exploration (ε, and τ), and learning (α) rates. For example, if $\alpha_s = 1.0$, and $\alpha_e = 0.1$, and there are a total of 60 training games, then the value for α will decrease by 0.015 for each successive training game. The true measure of ε, τ, and α, cannot be determined until the final training games, when their current values are close to their desired ending values. For this reason, only the last three training games are taken into consideration during analysis (instead of testing games with no exploration, or learning). An outline of Game Four testing can be seen in Table 8.2.

Section	**RL Algorithm**	**Action Selection**
8.2	Sarsa	ε-greedy
8.3	Sarsa	softmax
8.4	Q learning	ε-greedy
8.5	Q learning	softmax
8.6	DynaQ	ε-greedy
8.7	DynaQ	softmax

Table 8.2: Testing outline for Game Four.

8.2 Sarsa with ε-greedy

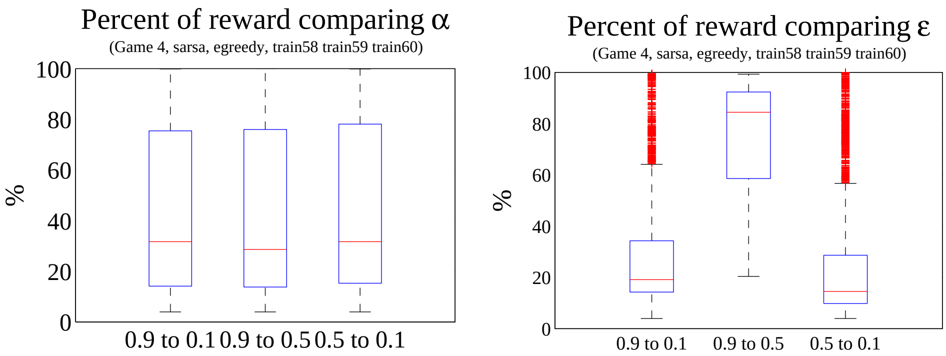
With this complicated game scenario, being able to look ahead for future rewards, and credit past actions with current rewards, are important to learning actions with delayed consequences. The values tested for each parameter are listed in Table 8.3. Each combination of variables is repeated 5 times, completing 60 training games for each repeated testing configuration. In this case, 324000 different results (percent of reward, and percent within bounds) were recorded. Box-plots showing the distribution of results, according to different parameter values, can be seen in Figures 8.1, 8.2, and 8.3. A scatter plot showing the percent of reward can be seen in Figure 8.4.

After initial analysis, the best results occur with a high discount rate, and high trace-decay value. For this reason, the 324000 results are reduced to 64800, thereby isolating the results to those with $\gamma = 0.9$, and $\lambda = 0.9$. These narrowed results are graphed with a scatter plot, in Figure 8.5.

The best value for mR, according to Figure 8.5, is $mR = -0.1$, or $mR = -0.05$. The 64800 results are once again reduced, to 21601, in order to isolate the results with $mR = -0.05$. The results of policies with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.05$, are illustrated with box-plots in Figures 8.6, and 8.7.

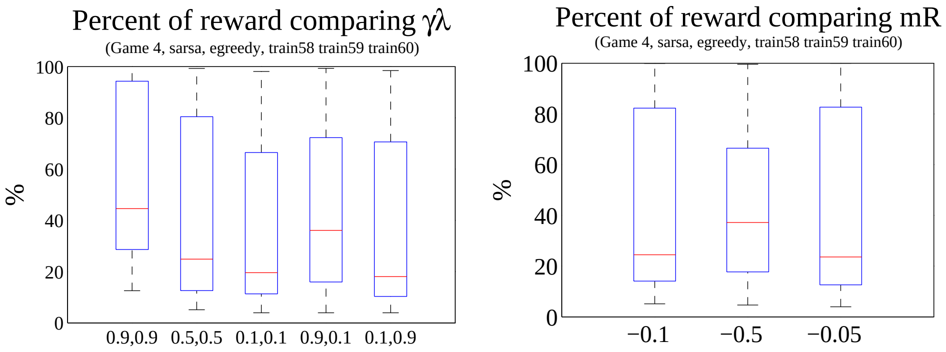
Variable	**Values**				
$[\gamma,\, \lambda]$	[0.9, 0.9]	[0.5, 0.5]	[0.1, 0.1]	[0.9, 0.1]	[0.1, 0.9]
$[\varepsilon_s,\, \varepsilon_e]$	[0.9, 0.1]	[0.9, 0.5]	[0.5, 0.1]		
$[\alpha_s,\, \alpha_e]$	[0.9, 0.1]	[0.9, 0.5]	[0.5, 0.1]		
mR	-0.1	-0.5	-0.05		
η_Q	0.7	0.3			
Hn_Q	[10]	[25]	[50]	[100]	

Table 8.3: Parameter values tested in Game Four with Sarsa, and ε-greedy.



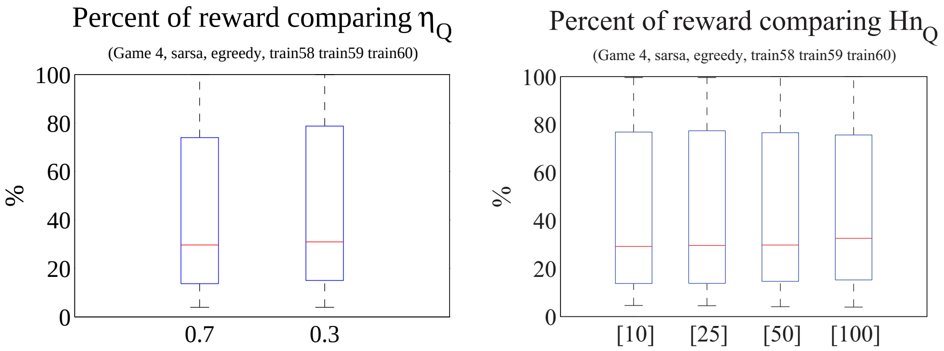
(a) (b)

Figure 8.1: Comparing the percent of reward of α, and ε, during testing for Game Four with Sarsa, and ε-greedy.



(a) (b)

Figure 8.2: Comparing the percent of reward of $\gamma\lambda$, and mR, during testing for Game Four with Sarsa, and ε-greedy.



(a) (b)

Figure 8.3: Comparing the percent of reward of η_Q, and Hn_Q, during testing for Game Four with Sarsa, and ε-greedy.

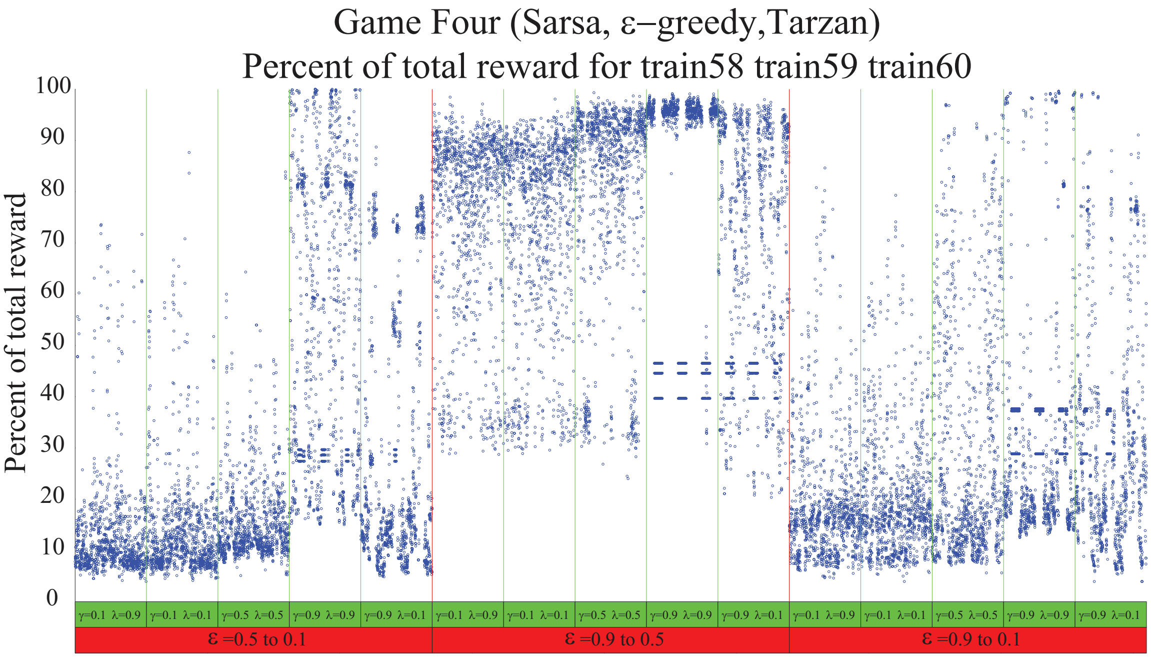


Figure 8.4: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Sarsa, and ε-greedy.

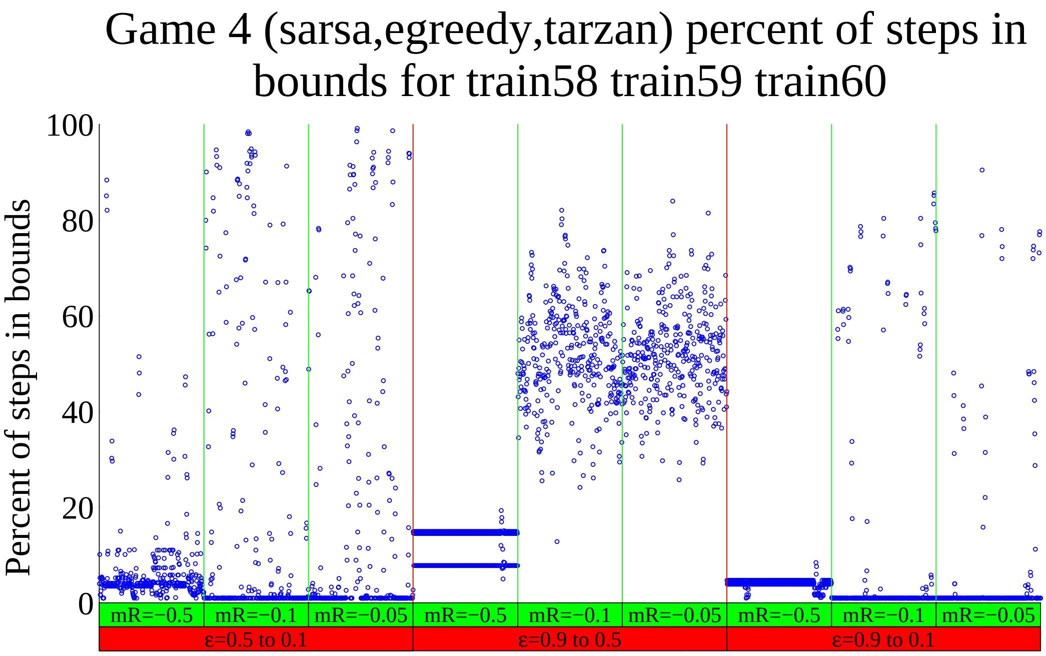


Figure 8.5: Results (percent in bounds) from training games 58, 59, and 60, during Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$ are included.

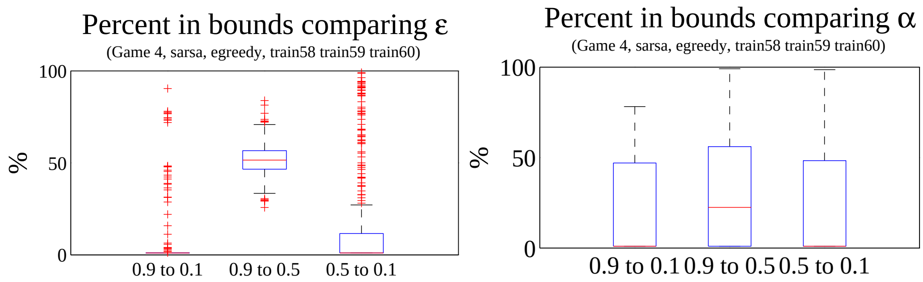


Figure 8.6: Comparing the percent of steps in bounds of ε, and α, during testing for Game Four with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.05$, are included.

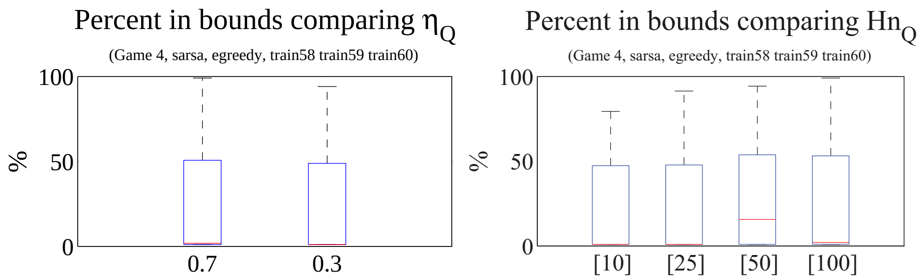


Figure 8.7: Comparing the percent of steps in bounds of η_Q, and Hn_Q, during testing for Game Four with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.05$, are included.

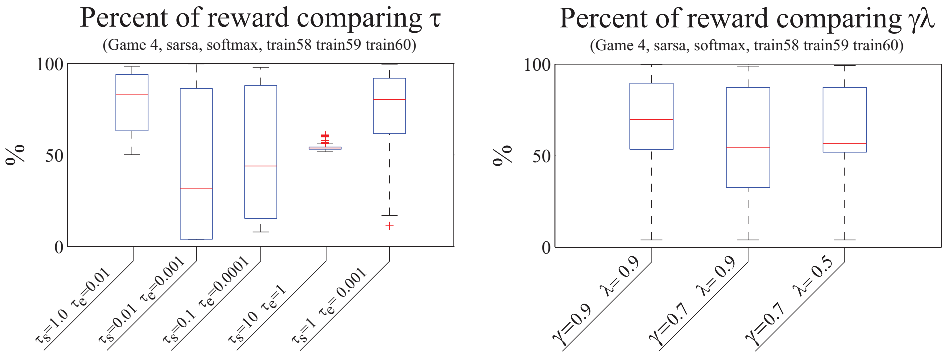
8.3 Sarsa with Softmax

With Game Four being the first scenario to test Softmax action selection, it is important to test many temperature values. The list of tested variables can be seen in Table 8.4. Each combination of values is repeated 5 times, completing 60 training games for each repeated testing configuration. In this case, all combinations of parameter values give 36000 results. Analysis showing distributions of the results can be seen in Figures 8.8, 8.9, and 8.10. The results are also graphed in Figures 8.11, and 8.12.

An initial analysis shows that the best results occur with a high discount rate, and high trace-decay value. For this reason, the 36000 results are reduced to 12000, thereby isolating the results to those with $\gamma = 0.9$, and $\lambda = 0.9$. A scatter plot of the reduced results can be seen in Figures 8.13, and 8.14. Finally, the results reduced to those with $mR = -0.1$, and graphed in Figure 8.15, 8.16, and 8.17.

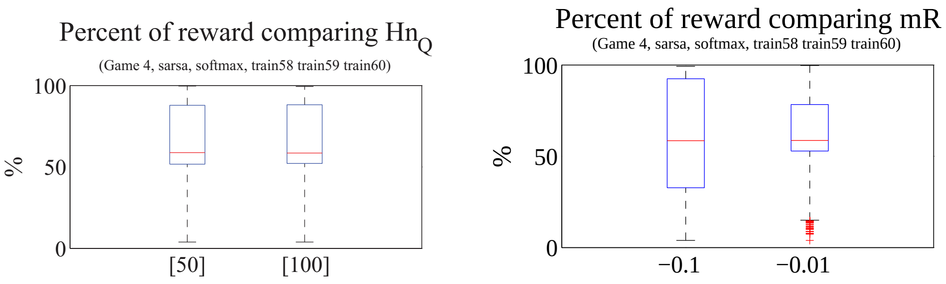
Variable	**Values**				
$[\gamma,\ \lambda]$	[0.9, 0.9]	[0.7, 0.9]	[0.7, 0.5]		
$[\tau_s,\ \tau_e]$	[1.0,0.01]	[0.01, 0.001]	[0.1, 0.001]	[10, 1]	[1.0, 0.001]
$[\alpha_s,\ \alpha_e]$	[0.9. 0.9]				
mR	-0.1	-0.01			
η_Q	0.3	0.9			
Hn_Q	[50]	[100]			

Table 8.4: Parameter values tested for Game Four with Sarsa, and Softmax.



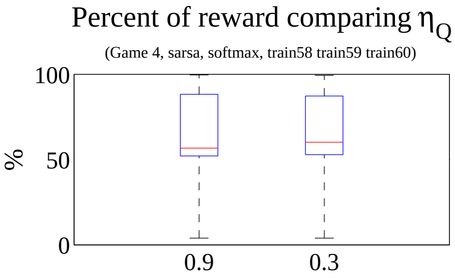
(a) (b)

Figure 8.8: Comparing the percent of reward of τ, and $\gamma\lambda$, during testing for Game Four with Sarsa, and Softmax.



(a) (b)

Figure 8.9: Comparing the percent of reward of Hn_Q, and mR, during testing for Game Four with Sarsa, and Softmax.



(a)

Figure 8.10: Comparing the percent of reward of η_Q during testing for Game Four with Sarsa, and Softmax.

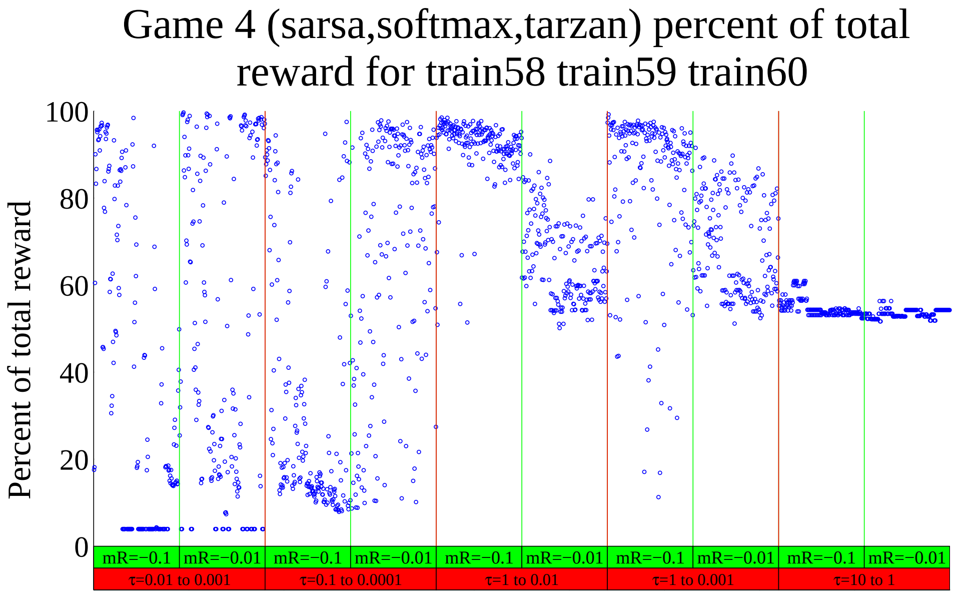


Figure 8.11: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax.

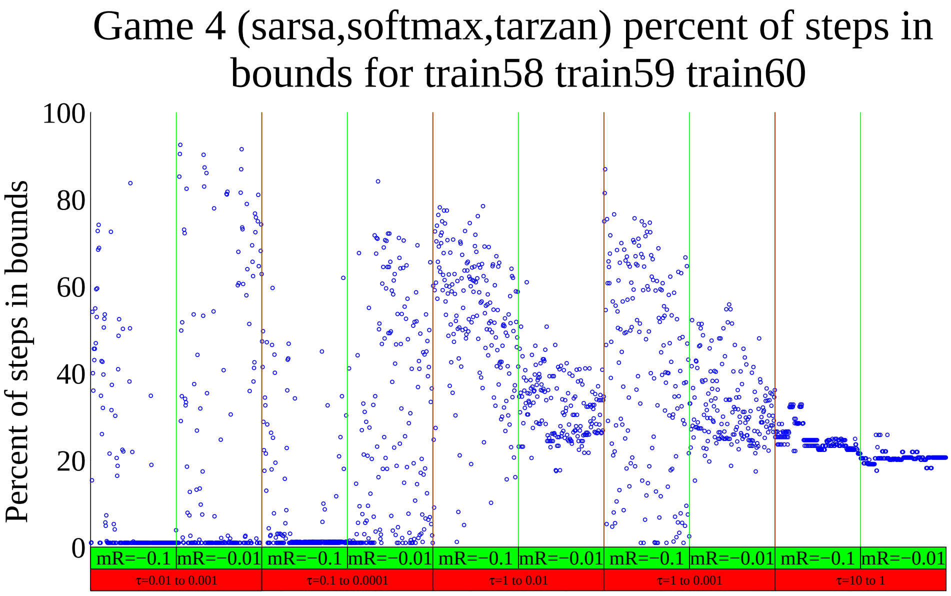


Figure 8.12: All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax.

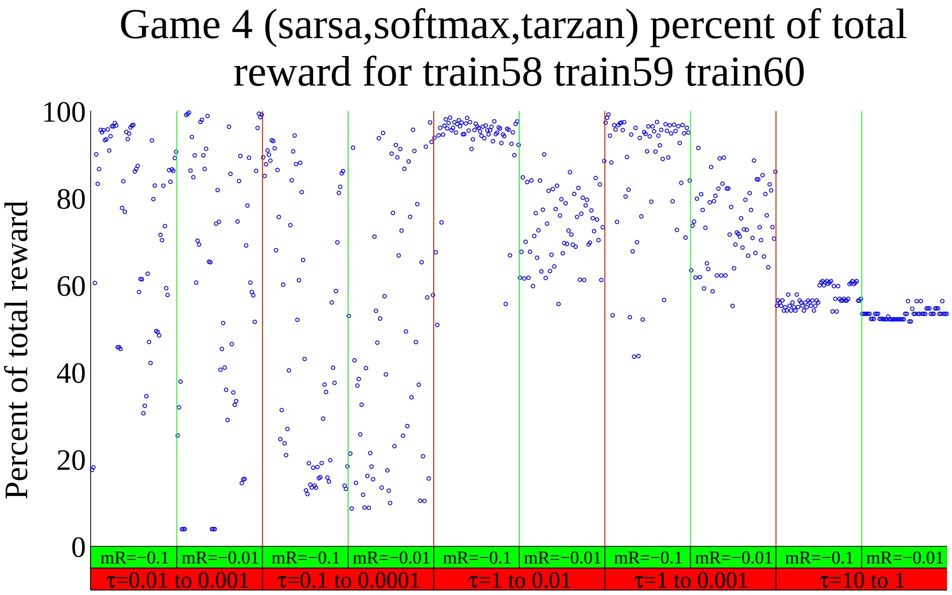


Figure 8.13: Results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$ are included in this scatter plot.

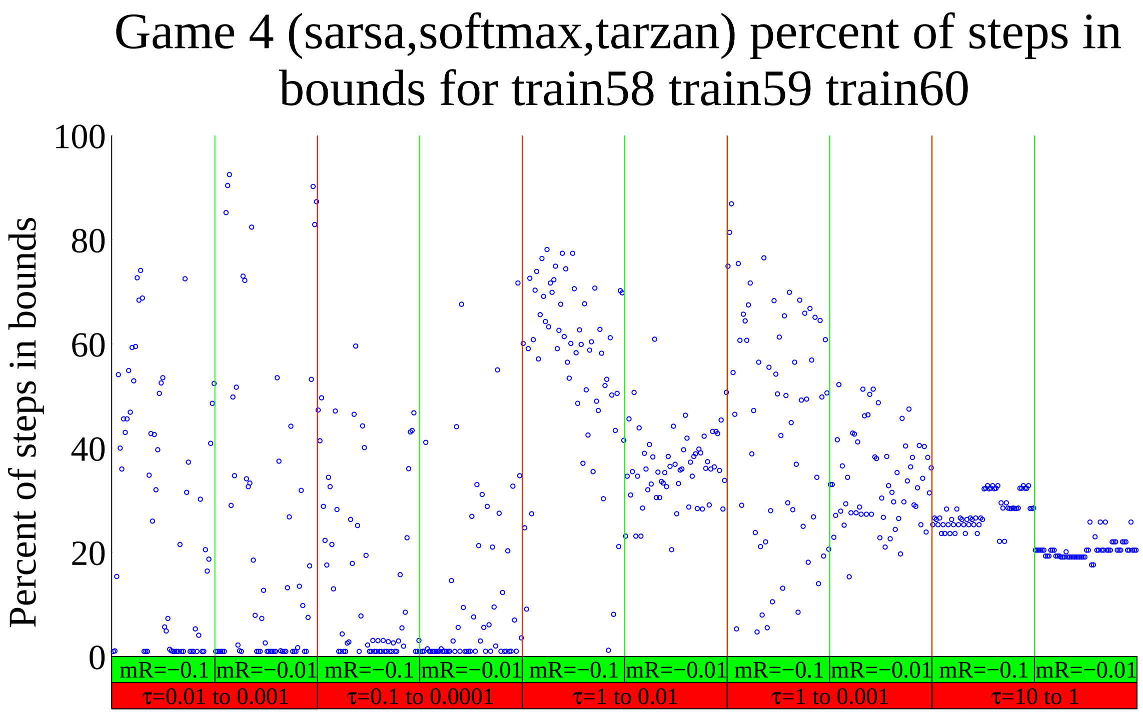
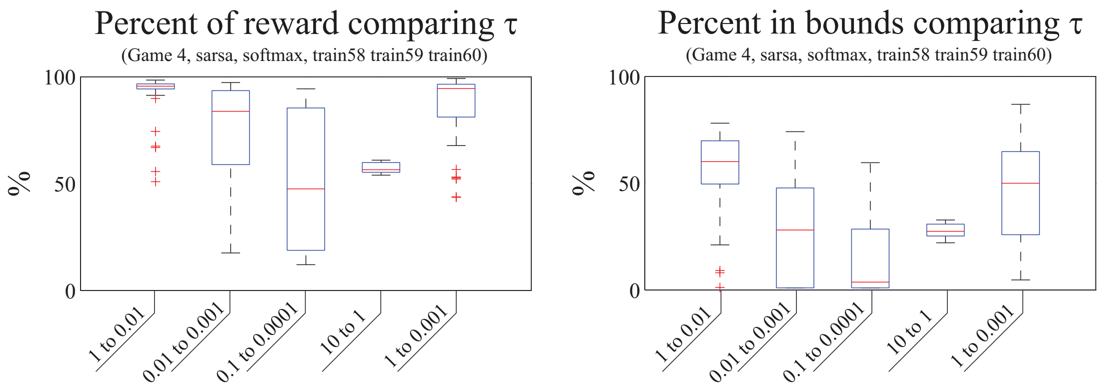


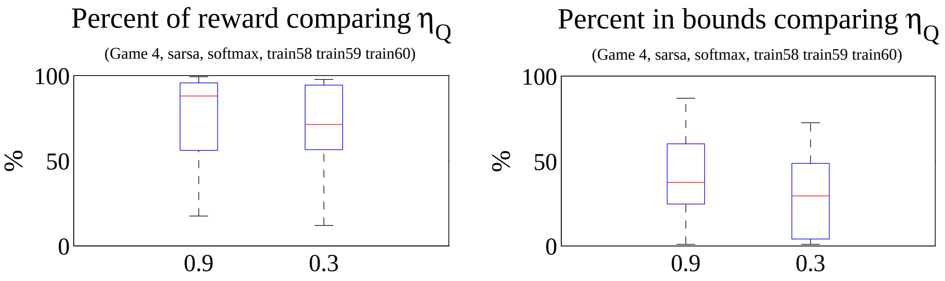
Figure 8.14: Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Sarsa, and Softmax. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$ are included in this scatter plot.



(a)

(b)

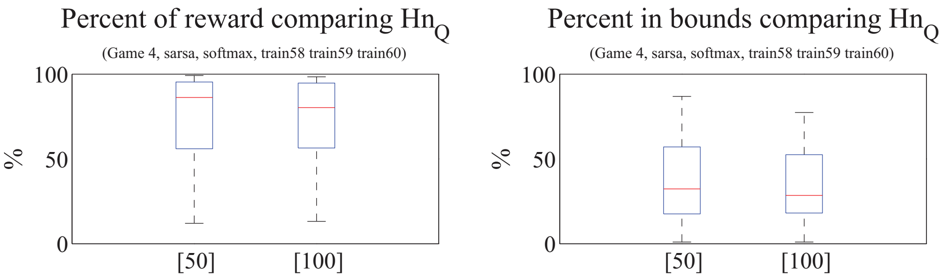
Figure 8.15: Comparing the percent of reward, and the percent of steps in bounds, of τ during testing for Game Four with Sarsa, and Softmax. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$.



(a)

(b)

Figure 8.16: Comparing the percent of reward, and the percent of steps in bounds, of η_Q during testing for Game Four with Sarsa, and Softmax. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$.



(a)

(b)

Figure 8.17: Comparing the percent of reward, and the percent of steps in bounds, of Hn_Q during testing for Game Four with Sarsa, and Softmax. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$.

8.4 Q-Learning with ε-greedy

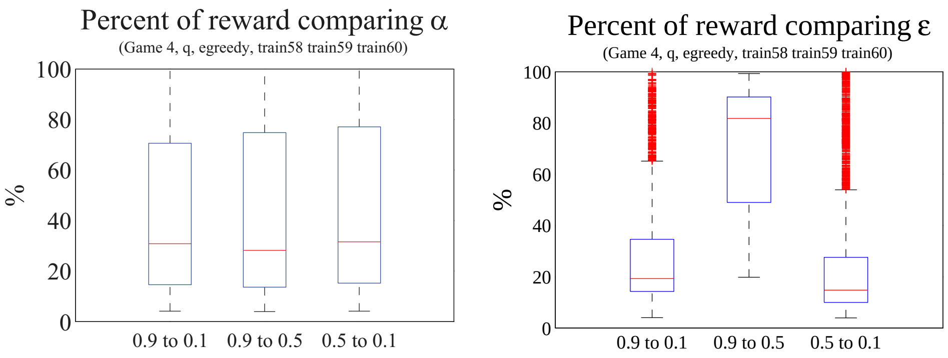
The tested values for Game Four with Q-learning, and ε-greedy, are listed in Table 8.5. Every variable combination is repeated 5 times, completing 60 training games for each repeated testing configuration. An analysis showing distributions of the results can be seen in Figures 8.18, 8.19, and 8.20. All results are also graphed using a scatter plot, in Figures 8.21, and 8.22.

After an initial analysis, the results are reduced to those with $\gamma = 0.9$, and $\lambda = 0.9$, and graphed with a scatter plot, in Figure 8.23. The results are further reduced to those with $mR = -0.1$, and graphed with a scatter plot in Figure 8.24, and box-plots in Figures 8.25, and 8.26.

Finally, results are reduced to those with $\eta_Q = 0.7$, $Hn_Q = [100]$, $\alpha_s = 0.9$, and $\alpha_e = 0.5$, and compared in Figure 8.27.

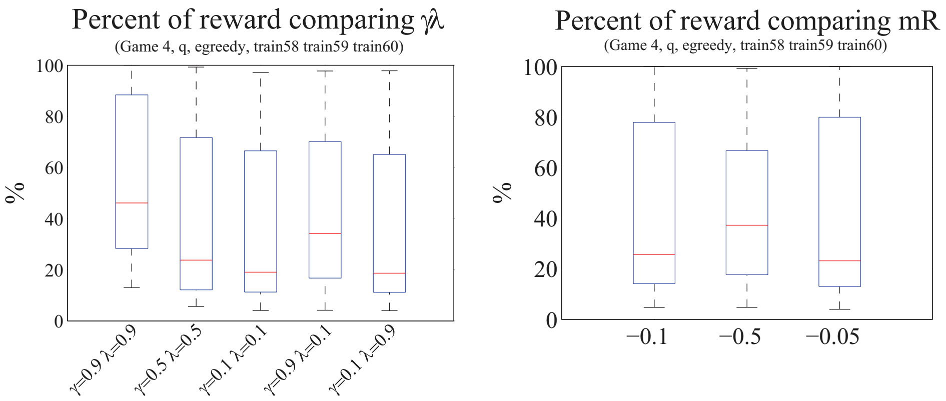
Variable	Values				
$[\gamma, \lambda]$	[0.9, 0.9]	[0.5, 0.5]	[0.1, 0.1]	[0.9, 0.1]	[0.1, 0.9]
$[\varepsilon_s, \varepsilon_e]$	[0.9, 0.1]	[0.9, 0.5]	[0.5, 0.1]		
$[\alpha_s, \alpha_e]$	[0.9, 0.1]	[0.9, 0.5]	[0.5, 0.1]		
mR	-0.1	-0.5	-0.05		
η_Q	0.7	0.3			
Hn_Q	[10]	[25]	[50]	[100]	

Table 8.5: Parameter values tested in Game Four with Q-learning, and ε-greedy.



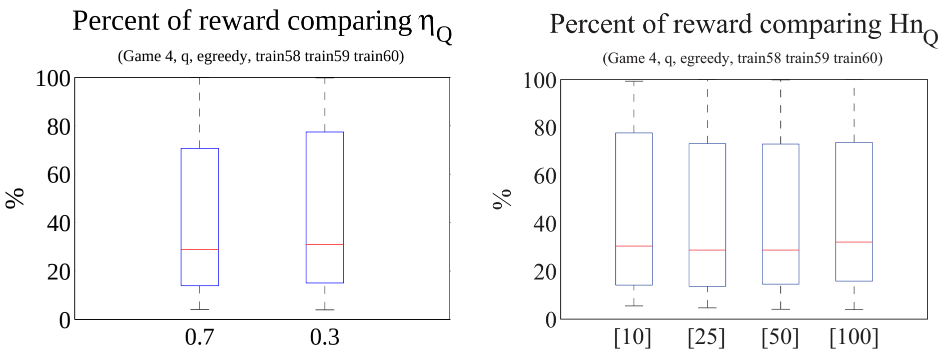
(a) (b)

Figure 8.18: Comparing the percent of reward of α, and ε, during testing for Game Four with Q-learning, and ε-greedy .



(a) (b)

Figure 8.19: Comparing the percent of reward of $\gamma\lambda$, and mR, during testing for Game Four with Q-learning, and ε-greedy.



(a) (b)

Figure 8.20: Comparing the percent of reward of η_Q, and Hn_Q, during testing for Game Four with Q-learning, and ε-greedy.

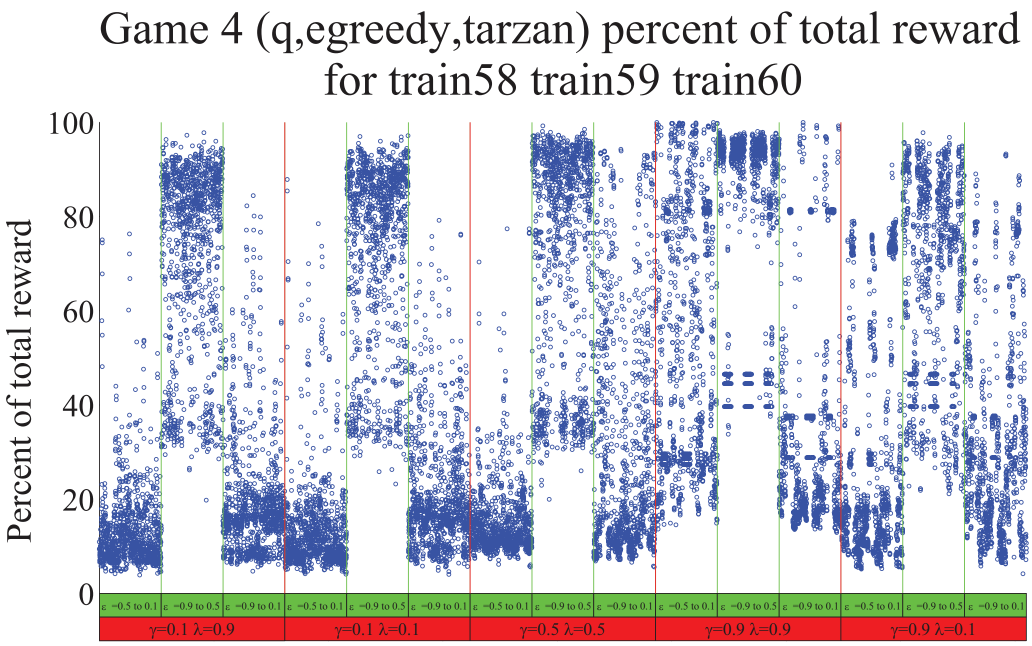


Figure 8.21: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy.

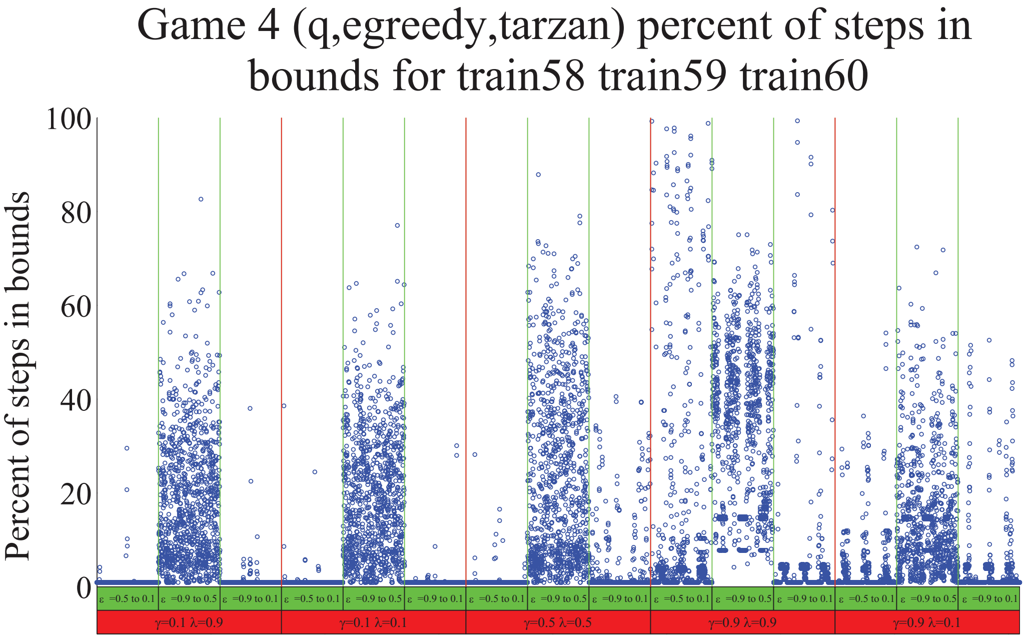


Figure 8.22: All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy.

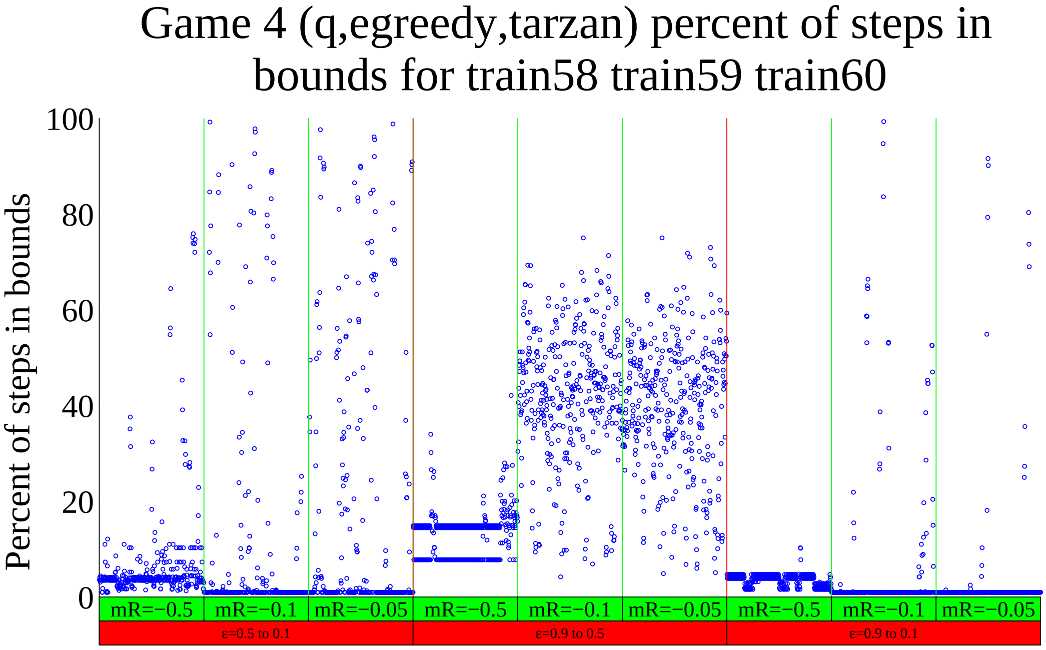


Figure 8.23: Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy. The results are limited to those with $\gamma = 0.9$, and $\lambda = 0.9$.

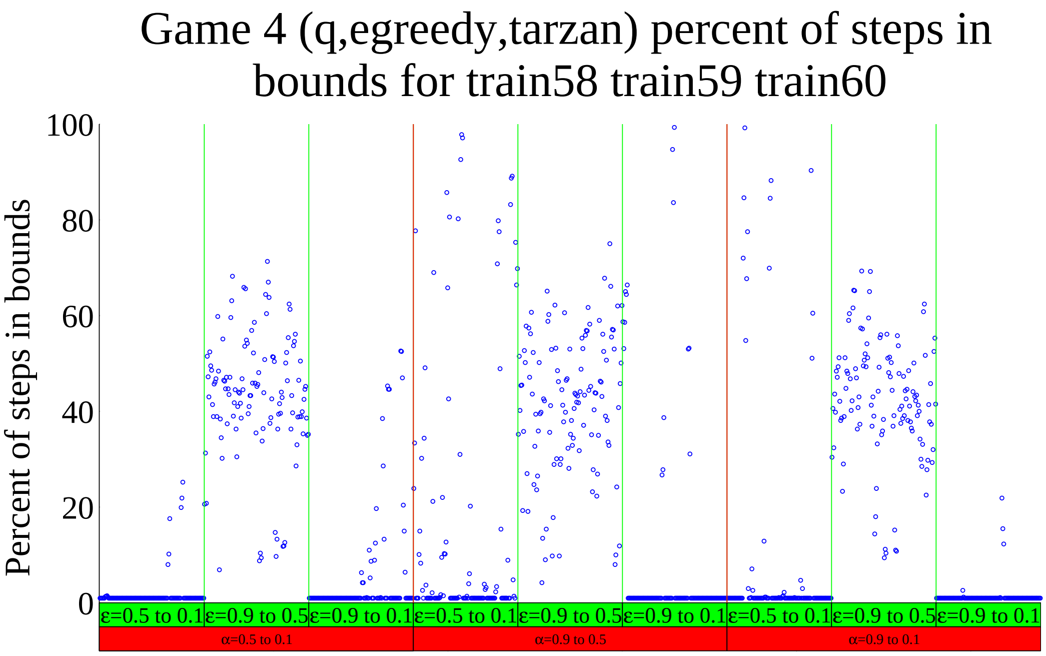
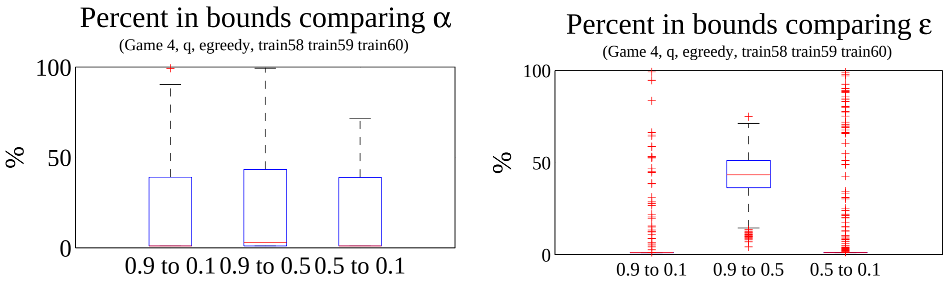


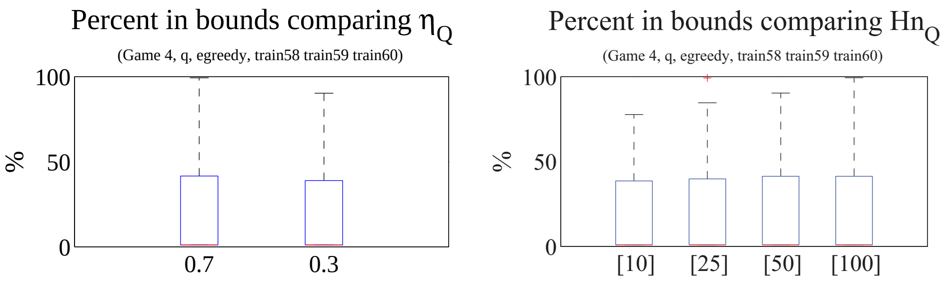
Figure 8.24: Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and ε-greedy. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$.



(a)

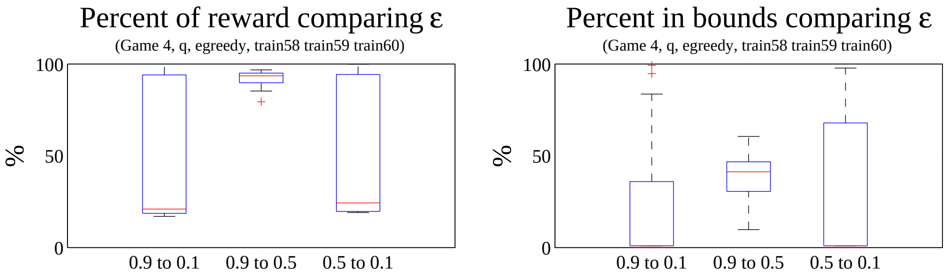
(b)

Figure 8.25: Comparing the percent of steps in bounds of α, and ε, during testing for Game Four with Q-learning, and ε-greedy . The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$.



(a) (b)

Figure 8.26: Comparing the percent of steps in bounds of η_Q, and Hn_Q, during testing for Game Four with Q-learning, and ε-greedy. The results are limited to those with $\gamma = 0.9$, $\lambda = 0.9$, and $mR = -0.1$.



(a) (b)

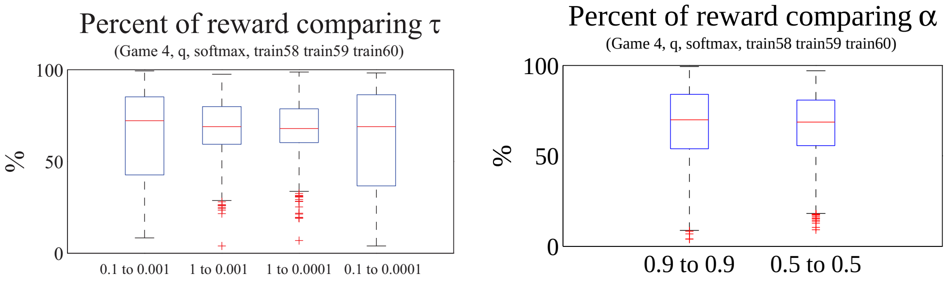
Figure 8.27: Comparing the results of ε, during testing for Game Four with Q-learning, and ε-greedy. The results are limited to those with$\gamma = 0.9$, $\lambda = 0.9$, $mR = -0.1$, $\eta_Q = 0.7$, $Hn_Q = [100]$, $mR = -0.1$, $\alpha_s = 0.9$, and $\alpha_e = 0.5$.

8.5 Q-Learning with softmax

The tested values for Game Four with Q-learning, and Softmax, are listed in Table 8.6. Each combination of variables is repeated 5 times, completing 60 training games for each repeated test configuration. Analysis showing distributions the of results can be seen in Figures 8.28, 8.29, and 8.30. All results from the last three training games (train58, train59, and train60), are graphed using a scatter plot in Figures 8.31, and 8.32. The full combinations of results are reduced to those with $\gamma = 0.9$, $\lambda = 0.9$, $mR = -0.1$, $\eta_Q = 0.9$, $mR = -0.1$, $\alpha_s = 0.9$, and $\alpha_e = 0.9$. The narrow results can be seen in Figure 8.33.

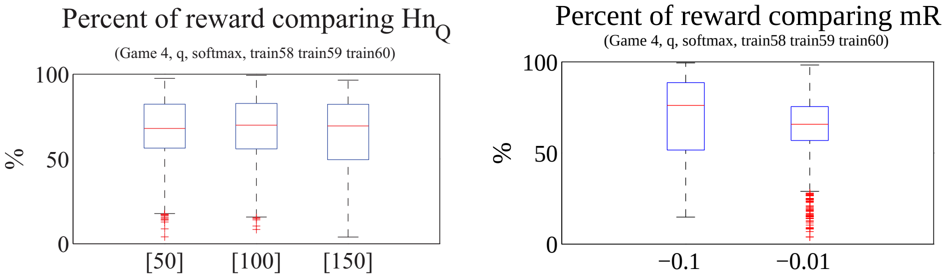
Variable	**Values**			
$[\gamma,\ \lambda]$	[0.9, 0.9]			
$[\tau_s,\ \tau_e]$	[0.1, 0.001]	[1, 0.001]	[1,0.0001]	[0.1, 0.0001]
$[\alpha_s,\ \alpha_e]$	[0.9. 0.9]	[0.5. 0.5]		
mR	-0.1	-0.01		
η_Q	0.9	0.3		
Hn_Q	[50]	[100]	[150]	

Table 8.6: Parameter values tested in Game Four with Q-learning, and Softmax.



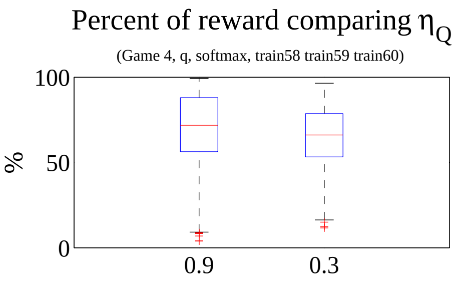
(a) (b)

Figure 8.28: Comparing the percent of reward of τ, and α, during testing for Game Four with Q-learning, and Softmax.



(a) (b)

Figure 8.29: Comparing the percent of reward of Hn_Q, and mR, during testing for Game Four with Q-learning, and Softmax.



(a)

Figure 8.30: Comparing the percent of reward of η_Q during testing for Game Four with Q-learning, and Softmax.

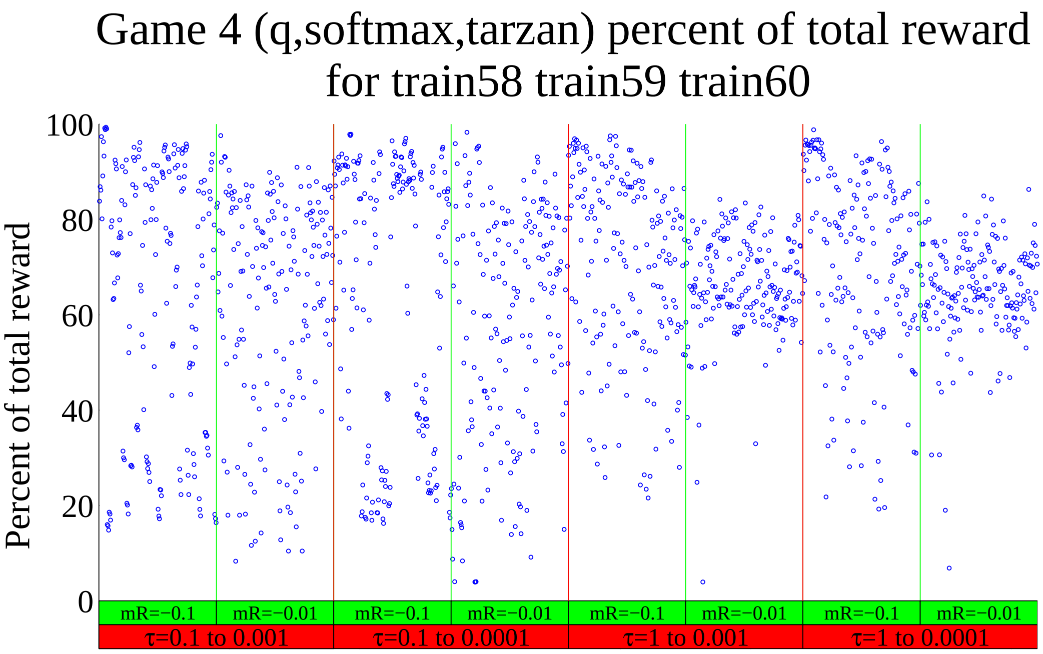


Figure 8.31: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Q-learning, and Softmax.

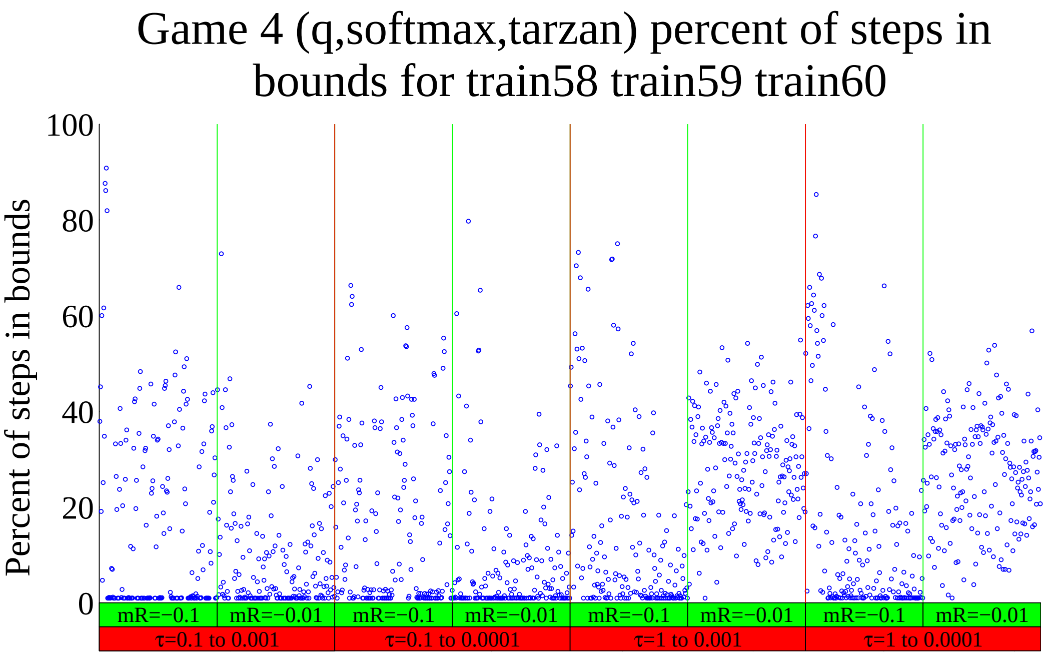
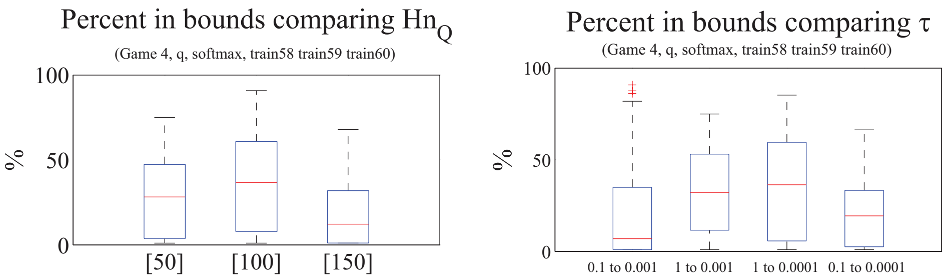


Figure 8.32: All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Q-learning, and Softmax.



(a) (b)

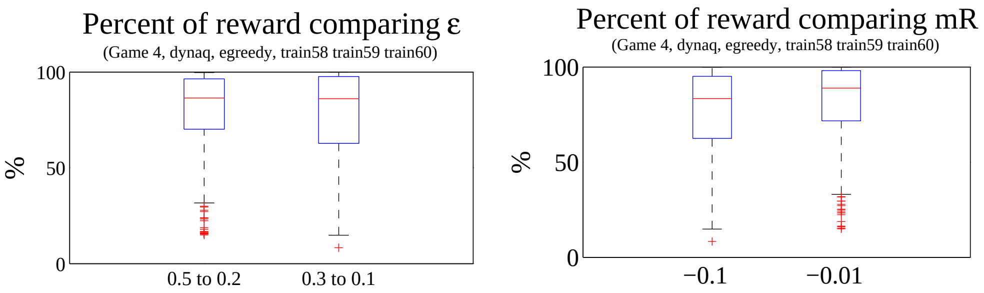
Figure 8.33: Comparing the percent of steps in bounds of Hn_Q, and τ, during testing for Game Four with Q-learning, and Softmax. The results are limited to those with $mR = -0.1$, $\eta_Q = 0.9$, $\alpha_s = 0.9$, and $\alpha_e = 0.9$.

8.6 Dyna-Q with ε-greedy

The tested values for Game Four with Dyna-Q, and ε-greedy, are listed in Table 8.7. Analysis showing distributions of the results can be seen in Figures 8.34, 8.35, and 8.36. All results from the last three training games (train58, train59, and train60), are graphed using a scatter plot seen in Figure 8.37. Every variable combination is repeated 5 times, completing 60 training games steps for each repeated testing configuration.

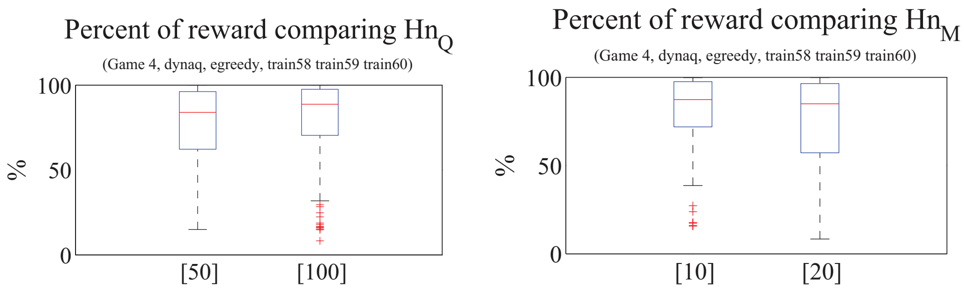
Variable	Values	
$[\gamma,\ \lambda]$	[0.9, 0.9]	
$[\varepsilon_s,\ \varepsilon_e]$	[0.5, 0.2]	[0.3, 0.1]
$[\alpha_s,\ \alpha_e]$	[1.0, 1.0]	
mR	-0.1	-0.01
η_Q	0.9	
Hn_Q	[50]	[100]
η_M	0.7	
Hn_M	[10]	[20]
pS	0	5

Table 8.7: Parameter values tested in Game Four using Dyna-Q, and ε-greedy.



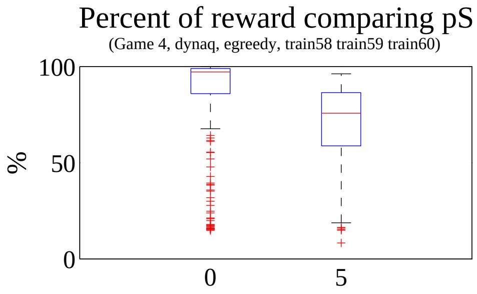
(a) (b)

Figure 8.34: Comparing the percent of reward of ε, and mR, during testing for Game Four with Dyna-Q, and ε-greedy.



(a) (b)

Figure 8.35: Comparing the percent of reward of Hn_Q, and Hn_M, during testing for Game Four with Dyna-Q, and ε-greedy.



(a)

Figure 8.36: Comparing the percent of reward of pS during testing for Game Four with Dyna-Q, and ε-greedy.

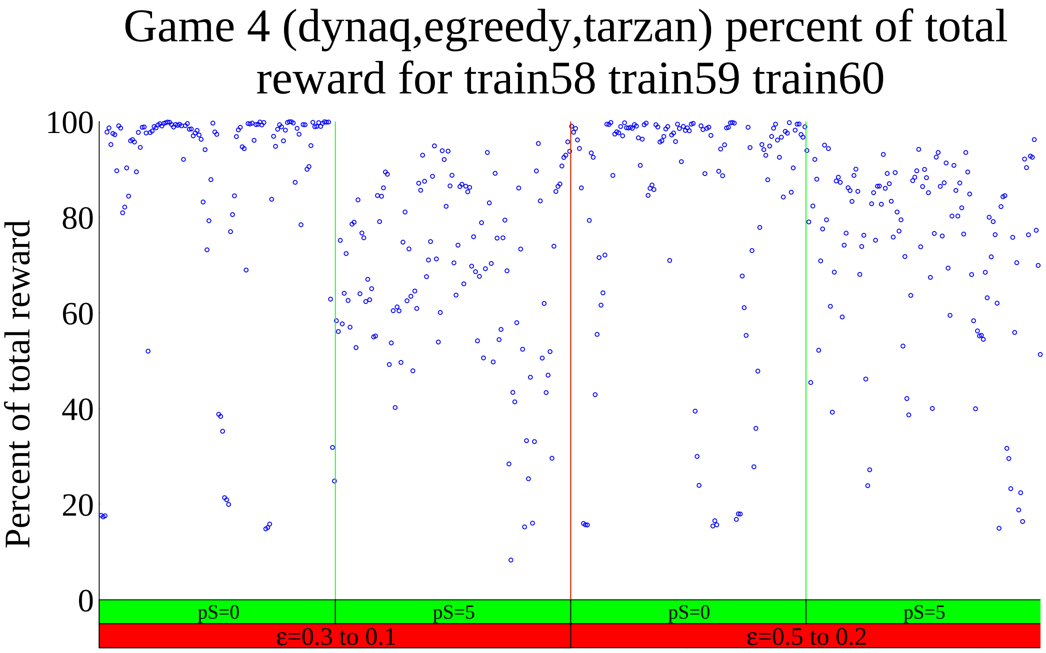


Figure 8.37: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and ε-greedy.

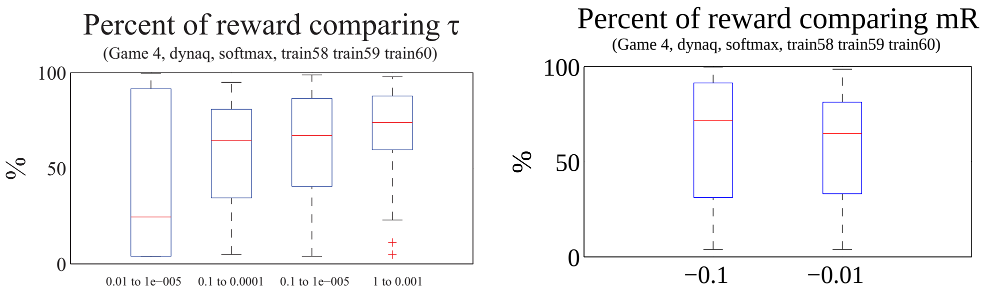
8.7 Dyna-Q with Softmax

The tested values for Game Four with Dyna-Q, and Softmax, are listed in Table8.8. Analysis showing distributions of the results can be seen in Figures 8.38, and 8.39. All results from the last three training games (train58, train59, and train60), are graphed using a scatter plot, in Figures 8.40, 8.41, 8.42, and 8.43. Every variable combination is repeated 5 times, completing 60 training games for each repeated testing configuration.

After analyzing the initial results, the values $\tau_s = 0.01$, and $\tau_e = 0.00001$, produce policies that are split between great, and terrible, percent of reward, and percent of steps in bounds. With the goal of maximizing the performance of all policies, all results with $\tau_s = 0.01$, and $\tau_e = 0.00001$, are eliminated from the overall results. The reduced results show that more planning steps produce policies with better performance, particularly when the temperature value is reduced to $\tau_e = 0.00001$, see Figures 8.44, and 8.45. The results are further reduced to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$, with box-plots in Figures 8.46, 8.47, and 8.48.

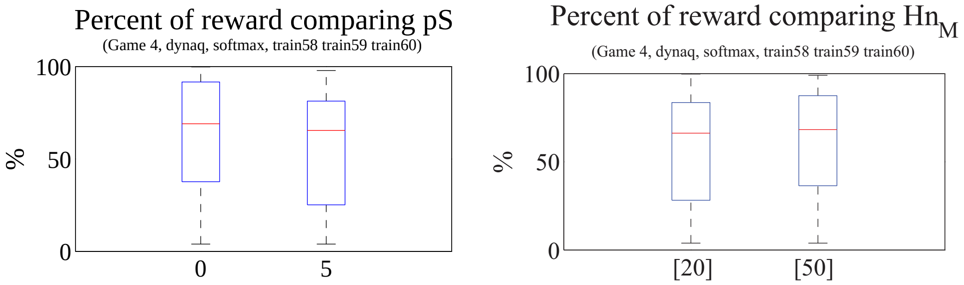
Variable	**Values**		
$[\gamma,\ \lambda]$	[0.9, 0.9]		
$[\tau_s,\ \tau_e]$	[0.01,0.00001]	[0.1,0.0001]	[0.1,0.00001]
$[\alpha_s,\ \alpha_e]$	[1.0, 1.0]		
mR	-0.1	-0.01	
η_Q	0.9		
Hn_Q	[50]	[100]	
η_M	0.7		
Hn_M	[20]	[50]	
pS	0	5	

Table 8.8: Parameter values tested in Game Four with Dyna-Q, and Softmax.



(a) (b)

Figure 8.38: Comparing the percent of reward of ε, and mR, during testing for Game Four with Dyna-Q, and Softmax.



(a) (b)

Figure 8.39: Comparing the percent of reward of Hn_M, and pS, during testing for Game Four with Dyna-Q, and Softmax.

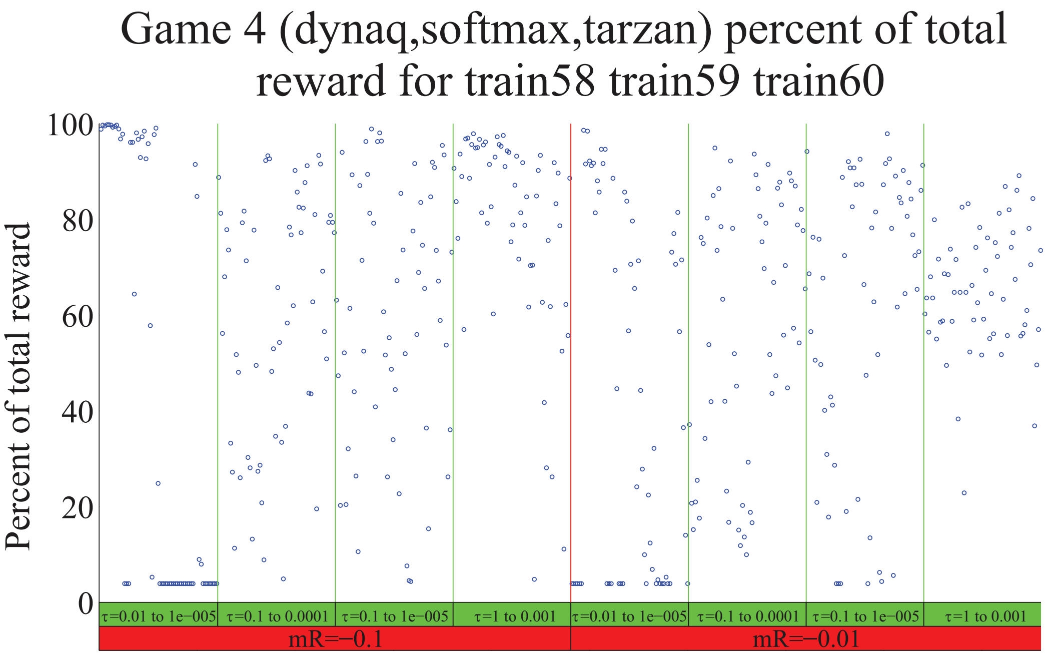


Figure 8.40: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax.

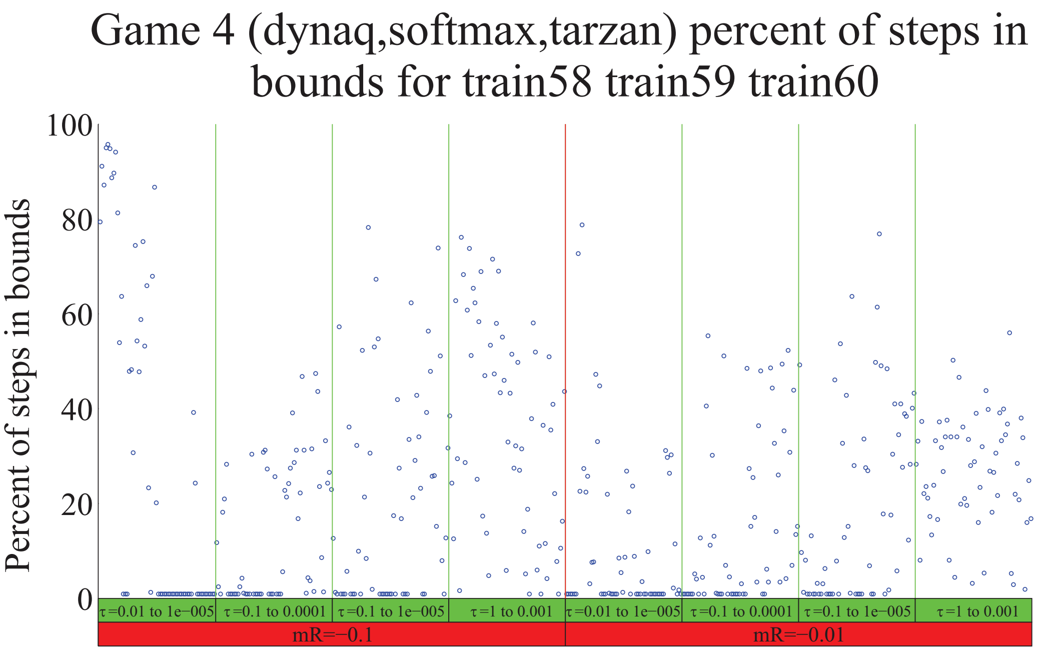


Figure 8.41: All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax.

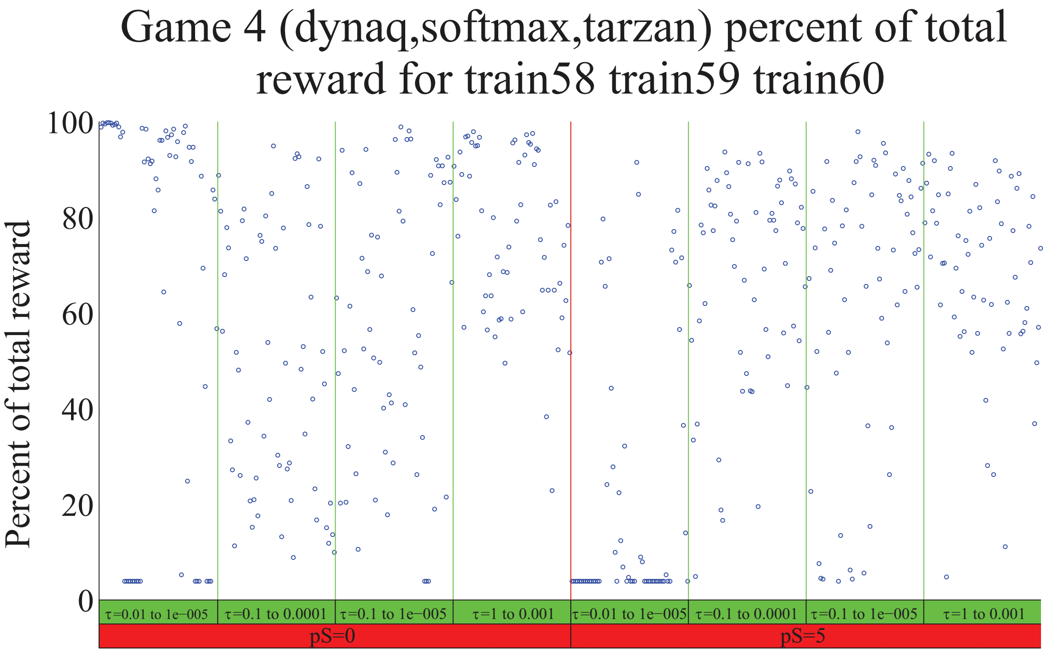


Figure 8.42: All results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax.

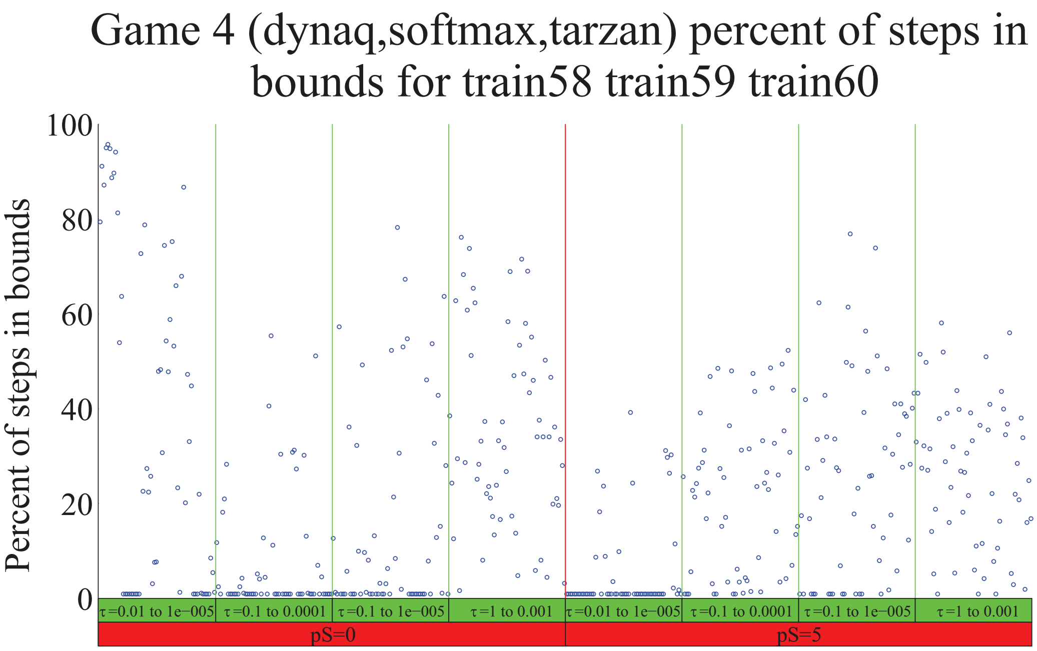


Figure 8.43: All results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax.

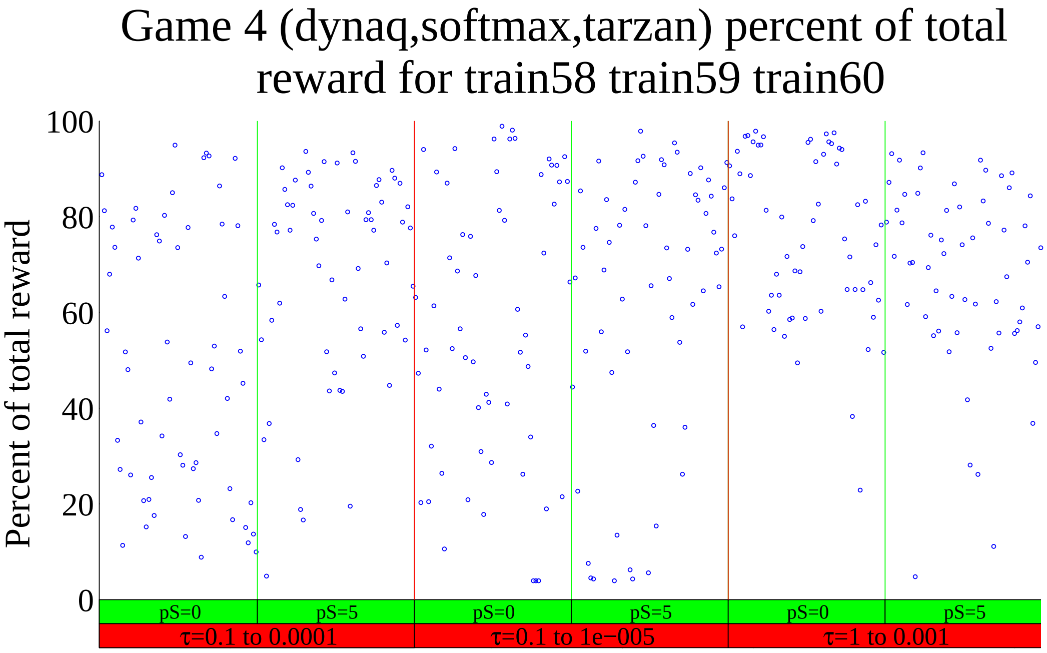


Figure 8.44: Results (percent of reward) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. These results do not include those with $\tau_s = 0.01$, and $\tau_e = 0.00001$.

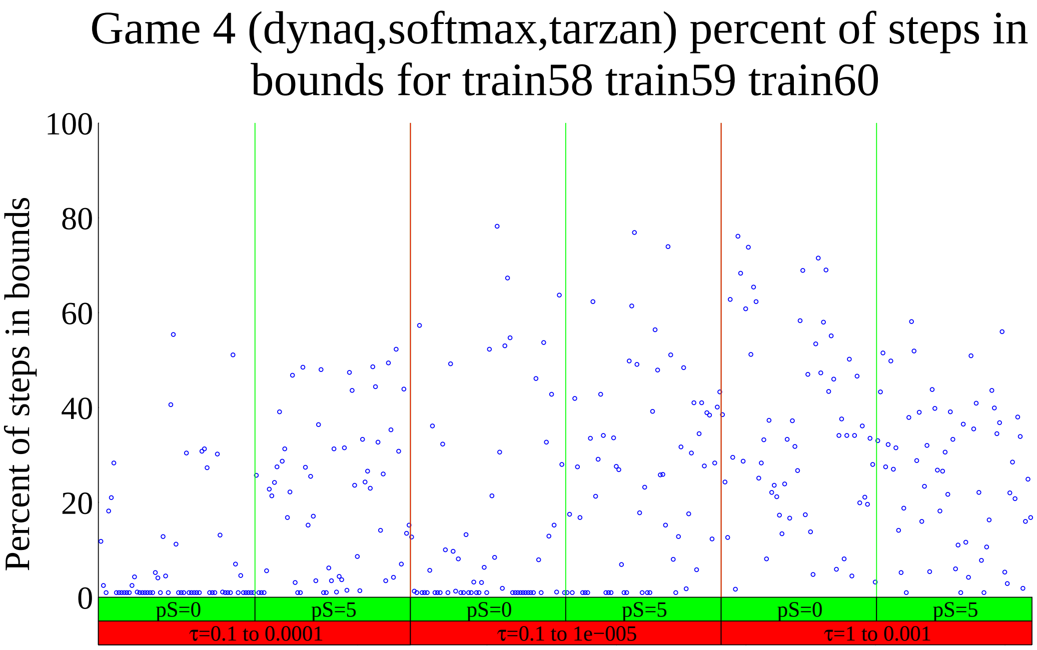
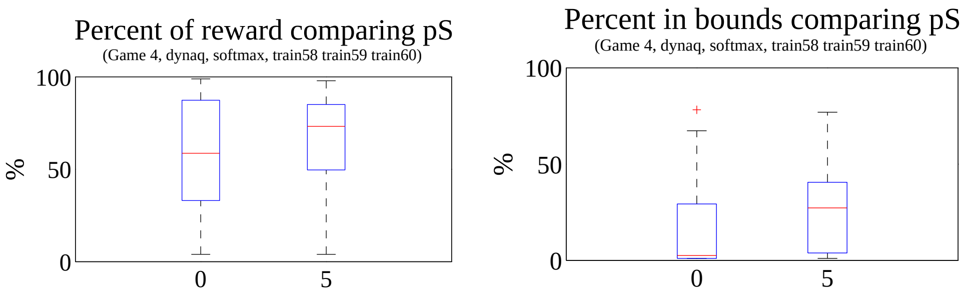


Figure 8.45: Results (percent of steps in bounds) from training games 58, 59, and 60, during Game Four testing with Dyna-Q, and Softmax. These results do not include those with $\tau_s = 0.01$, and $\tau_e = 0.00001$.



(a)

(b)

Figure 8.46: Comparing the performance of pS, during testing for Game Four with Dyna-Q, and Softmax. The results are limited to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$.

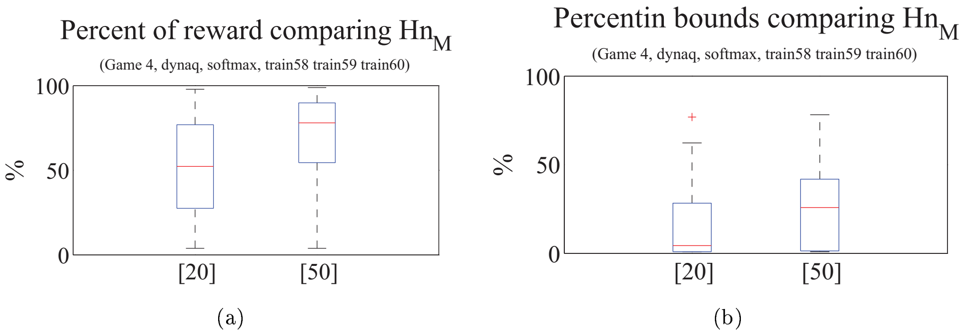


Figure 8.47: Comparing the performance of Hn_M, during testing for Game Four with Dyna-Q, and Softmax. The results are limited to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$.

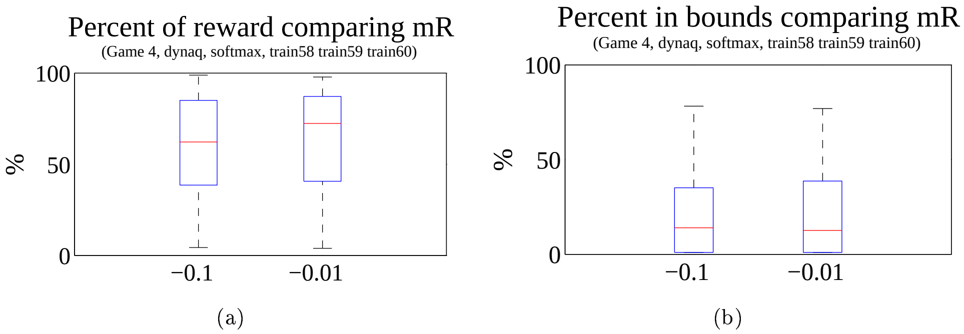


Figure 8.48: Comparing the performance of mR, during testing for Game Four with Dyna-Q, and Softmax. The results are limited to those with $\tau_s = 0.1$, and $\tau_e = 0.00001$.

8.8 Discussion

This section will discuss the testing results for Game Four in terms of optimal parameters, consistency, training time, and selective qualitative analysis.

8.8.1 Parameters

Exploration Rate Since Game Four introduced Softmax action selection for the first time, the exploration rate refers to both ε, and τ. After testing ε-greedy with Sarsa, and Q-learning, the value that maximizes the percent of steps in bounds, was found to be $\varepsilon_s = 0.5$, and $\varepsilon_e = 0.1$, see Figures 8.6, and 8.27. Results also show that values of $\varepsilon_s = 0.9$, and $\varepsilon_e = 0.5$, produce policies with much higher percent of reward, see Figures 8.6, and 8.27. The best possible exploration rate will depend on the importance of reward, and steps in bounds.

After testing the Softmax action selection, similar results were seen for Sarsa, and Q-learning. The values that produce higher percent of reward, and higher percent of steps in bounds, that start with $\tau_s = 1$, and end with $\tau_e = 0.001$, or $\tau_e = 0.001$, see Figures 8.15, and 8.33.

The exploration rates that were successful for Dyna-Q were much smaller than for Sarsa, or Q-learning. In the case of Dyna-Q with ε-greedy, better policies are trained with $\varepsilon_s = 0.3$, and $\varepsilon_e = 0.1$, see Figure 8.34. For Dyna-Q and Softmax, higher performing (percent of reward, and percent of steps) policies were trained with $\tau_s = 0.1$, and $\tau_e = 0.00001$, see Figures 8.44, and 8.45.

Reinforcement Learning Rate Not all variable combinations included multiple values for the reinforcement learning rate (α). When the values were tested, the policies always perform better with higher values of α, see Figures 8.6, and 8.25.

Q-Function ANN Learning Rate For this game scenario, larger values for η_Q produce policies with a higher percent of reward, and a higher percent of steps in bounds, see Figures 8.7, 8.16, 8.26, and 8.30.

Motive Reward Factor For this game scenario, $mR = -0.1$ produced better policies for all RL algorithms, and action selection methods, see Figures 8.5, 8.11, 8.23, and 8.31. There is very little variation in results when changing mR while using ε-greedy action selection. However, because mR directly influences the magnitude of rewards, and the resulting Q-function values, different values for mR require different temperature rates.

Discount and Trace-Decay Throughout all tests for this game scenario, the optimal values are consistently $\gamma = 0.9$ and $\lambda = 0.9$. This suggests that optimal policies are found when rewards are propagated to previous actions as much as possible, and future rewards are considered more heavily than short-term rewards.

Q-Function ANN Hidden Neurons For this game scenario, all values of Hn_Q have only 1 hidden layer. Best results were achieved with $Hn_Q = [50]$, or $Hn_Q = [100]$, see Figure 8.7, 8.17, 8.26, 8.29, and 8.35.

Dyna-Q Given the Game Four scenario is more complex than previous Games, it was expected that Dyna-Q results, with $pS > 0$, would produce terrible policies. This expectation was confirmed in the testing results for Dyna-Q and ε-greedy, see Figure 8.36. The results for Dyna-Q and Softmax are more promising, see Figure 8.46. The median percent of steps in bounds, and the median percent of reward, are higher for $pS = 5$, than for $pS = 0$.

RL	Hn_Q	η_Q	mR	α_S	α_e	ε_s	ε_e	γ	λ
Sarsa	[100]	0.7	-0.1	0.9	0.5	0.9	0.5	0.9	0.9

Table 8.9: Outline of the variables used to test the consistency of the results.

Percent of Reward				
Test Number	**train58**	**train59**	**train60**	**mean**
1	97.70	98.87	98.14	**98.24**
2	97.14	96.67	97.51	**97.11**
3	96.25	96.04	96.86	**96.38**
4	96.66	97.58	97.77	**97.34**
5	96.52	97.41	96.51	**96.82**
6	97.74	97.72	95.48	**96.98**
7	98.28	98.43	96.16	**97.62**
8	97.23	97.45	97.68	**97.45**
9	96.71	97.17	97.92	**97.27**
10	90.25	95.29	97.70	**94.41**
11	96.88	88.73	93.17	**92.93**
12	92.50	96.74	96.33	**95.19**
13	95.83	97.38	96.83	**96.68**
14	93.78	97.21	95.44	**95.48**
15	91.84	95.76	96.30	**94.63**
16	94.75	96.53	96.71	**96.00**
17	97.80	95.80	97.75	**97.12**
18	95.37	97.57	97.81	**96.92**
19	95.22	96.35	94.82	**95.46**
20	97.46	98.29	97.48	**97.74**
Std	**2.19**	**2.09**	**1.26**	**1.33**

Table 8.10: Statistics measuring consistency of Game Four results (percent of reward).

8.8.2 Consistency

Testing the consistency of the results involves repeating a test under identical conditions and comparing the results. To determine consistency, 20 repeated tests were run with identical parameter values (Table 8.9). The results of the last 3 training games (train58, train59, and train60), for every test number, can be found in Table 8.10 for the percent of reward, and Table 8.11 for the percent of steps in bounds. After analyzing the results, it is clear that there is more variation among the percent of steps in bounds, than the percent of reward.

Percent of Steps in Bounds				
Test Number	**train58**	**train59**	**train60**	**mean**
1	71.8	78.8	73.3	**74.63**
2	67.3	60.5	70.7	**66.17**
3	58.4	57.2	57.3	**57.63**
4	56.9	66.9	73.4	**65.73**
5	62.2	68.6	62.8	**64.53**
6	67.5	68.3	45	**60.27**
7	73.6	74.6	59.1	**69.10**
8	66.2	60.3	64.6	**63.70**
9	57.4	62.6	68.1	**62.70**
10	29.1	51.4	67.7	**49.40**
11	56.6	38.1	32.2	**42.30**
12	37.2	58.7	60.1	**52.00**
13	51.2	70.4	63.4	**61.67**
14	47.7	65.3	57.3	**56.77**
15	38.6	61.6	50.1	**50.10**
16	42.7	57	59.7	**53.13**
17	75.5	59.7	72.2	**69.13**
18	57.7	67	71.5	**65.40**
19	51.3	55.7	59.2	**55.40**
20	66.8	73	63.1	**67.63**
Std	**12.83**	**9.05**	**10.24**	**8.12**

Table 8.11: Statistics measuring consistency of Game Four results (percent of steps in bounds).

8.8.3 Training Time

Complexity of the game four scenario necessitates a longer training time. The Figures 8.49 and 8.50 show distributions of the results for the 20 repeated tests, after each training game. The parameter values tested are outlined in Table 8.9. Further training has the potential to improve the percent in bounds given the fact that the values are slowly increasing, even at the 60^{th} training game.

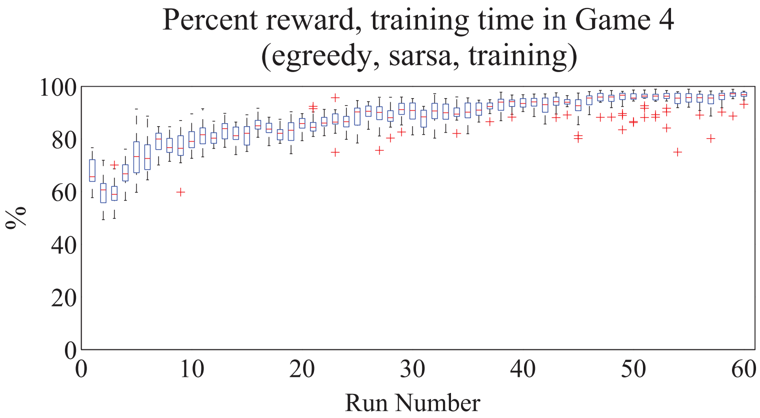


Figure 8.49: Showing the percent of reward after every training game to compare the influence of training time. Each run number includes results from the 20 repeated tests from Table 8.9.

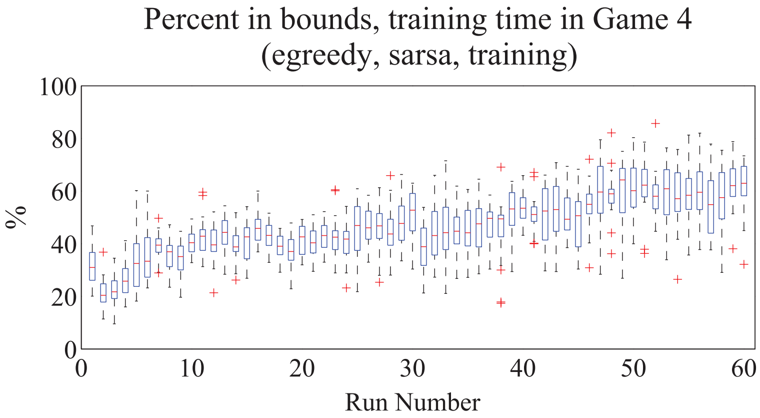
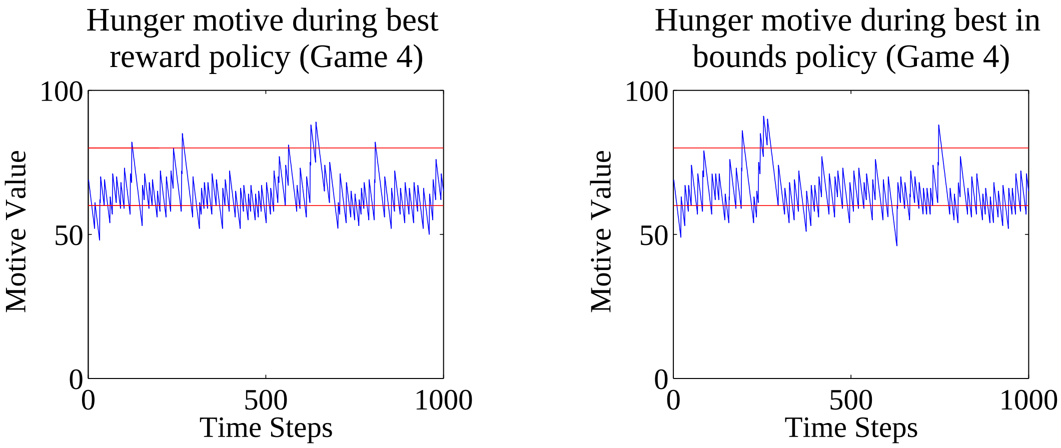


Figure 8.50: Showing the percent of steps in bounds after every training game to compare the influence of training time. Each run number includes results from the 20 repeated tests from Table 8.9.

8.8.4 Qualitative Analysis

Analyzing how motive values change during a game is important to the overall goal of believability. Figure 8.51 shows the motive values from two policies that were found using the variables described in Table 8.9. In both cases, the agent prefers to go below the minimum threshold, rather than go above the maximum. This is easily explained by the bounds themselves. The individual motive reward is scaled by the range between the maximum and 100, or minimum and 0, depending on which side of the bounds the current motive value resides. The smaller the range, the more severe the negative reward. Going over the maximum threshold by 10 received much larger negative reward, than if the motive value were 10 below the minimum threshold. For this reason it makes complete sense that the agent avoids going over the maximum much more than it avoids going under the minimum. It is an accurate result of the learning.



(a) The *hunger motive* values throughout the final training game that maximizes the amount of reward.

(b) The *hunger motive* values throughout the final training game that maximizes the amount of steps in bounds.

Figure 8.51: Qualitative analysis of the policies found using parameters outlines in Table 8.9.

8.9 Summary

The results show that an agent can learn actions even with rewards that are delayed as much as five game steps. The agent Tarzan, learns to go to the grocery store, buy food, return home, open the fridge, and eat the food. In this simplified real life scenario, the food can only be purchased one at a time, and the food instantly goes into the fridge once purchased. The fridge does have a maximum capacity, to prevent the agent from buying too much food at any given time. With the introduction of Softmax action selection, it was interesting to see that mR has such large influence on the best τ value. Softmax uses estimated Q-values to calculate action probabilities, and mR determines the magnitude of the Q-values learned, therefore, mR has a direct impact on the required values for τ. Also, surprisingly, Dyna-Q with Softmax was able to approximate the environment enough to produce better polices with $pS > 0$. Dyna-Q appears to be more effective when trained with smaller rates of exploration, compared to the exploration needed for Sarsa, and Q-learning.

Chapter 9

Game Five

The fifth, and final, game scenario, is the culmination of all previous testing games. It includes inter-agent actions, delayed reward, and multiple motives. The game world consists of four places: Tarzan's house, Jane's House, the bar, and the grocery store. Both Jane, and Tarzan, have the option of going to the bar, or the grocery store, but not to each other's houses. Bob is meant to be the bartender, but he also has the option of going to the grocery store. When an agent talks to another agent, they receive +20 *social*, and +3 *entertainment*, while the agent they talk to receives +5 *social*, and +1 *entertainment*. Since Jane has no entertainment motive, any motive change for entertainment has no effect on her. Table 9.1 shows the motives, and their bounds, for each agent in the game world. Table 9.2 shows actions that are associated with objects, and how the actions change an agent's motive values.

Agent	Social		Food		Health		Entertainment	
	Min	Max	Min	Max	Min	Max	Min	Max
Tarzan	10	50	40	70	50	90	70	100
Jane	50	80	40	70				
Bob	99	100						

Table 9.1: Description of motivations for all agents, in the Game Five scenario.

Object	Action	Affects Motive			
		Social	Food	Healthy	Entertainment
Healthy food	eat		+20	+1	
Greasy food	eat		+15	-1	
Treadmill	use		-2	+25	
TV	watch			-2	+20

Table 9.2: Description of how object actions change agent's motives, in the Game Five scenario.

9.1 Testing Outline

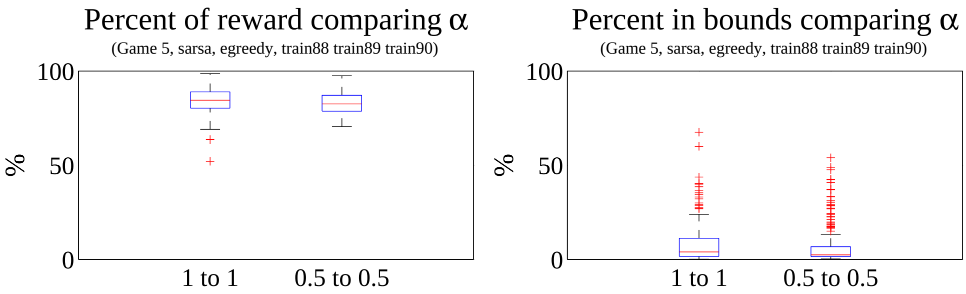
For this particular game scenario, testing is done with Sarsa, and ε-greedy. The goal is to show that this level of complexity is possible to learn. Testing is done with complete combinations of the variables outlined in Table 9.3. Each testing configuration has 90 training games, and is repeated 5 times for consistency measures. Only the last 3 training games (train88, train89 and train90) are used for analysis purposes.

9.2 Sarsa and ε-greedy

Variables	Tested Values			
$[\gamma,\ \lambda]$	[0.9, 0.9]	[0.7, 0.9]	[0.7, 0.7]	[0.5, 0.9]
$[\varepsilon_s,\ \varepsilon_e]$	[0.5, 0.1]	[0.5, 0.2]		
$[\alpha_s,\ \alpha_e]$	[1.0, 1.0]	[0.5, 0.5]		
mR	-0.01			
Hn_Q	[100]			
η_Q	0.9	0.3		

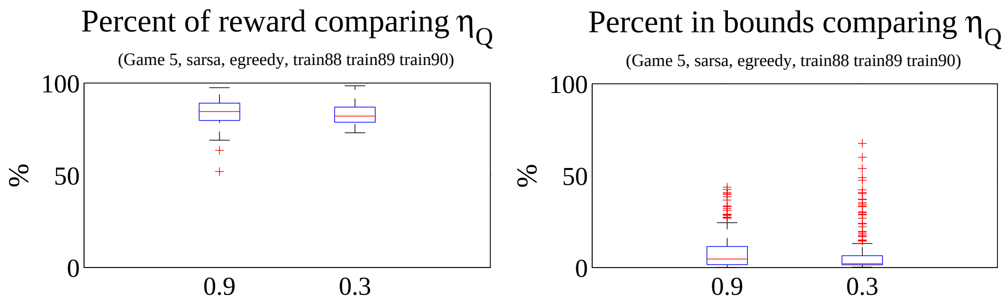
Table 9.3: Parameter values tested in Game Four with Sarsa, and ε-greedy.

The following graphs show the results of the last three training games from all combinations of variables seen in Table 9.3. After analyzing the initial results, it is clear that the policies with $\gamma = 0.9$, and $\lambda = 0.9$, have the highest performance, see Figures 9.4 and 9.5. Creating a separate subset of results, with $\gamma = 0.9$ and $\lambda = 0.9$, allows the variable interactions within that subset to become more clear, see Figures 9.7, 9.8, and 9.9.



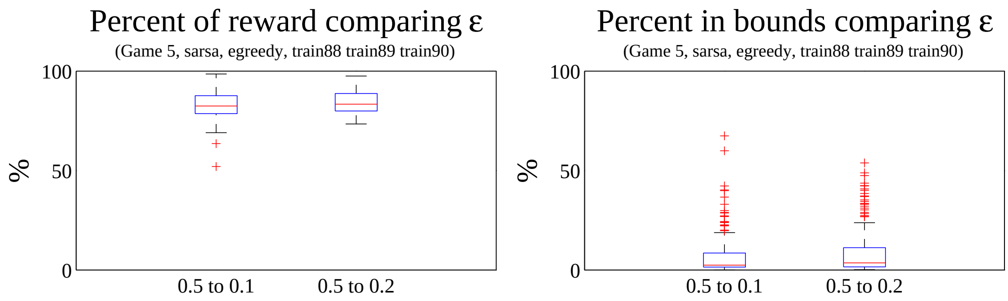
(a) (b)

Figure 9.1: Comparing the performance of α in Game Four testing with Sarsa, and ε-greedy.



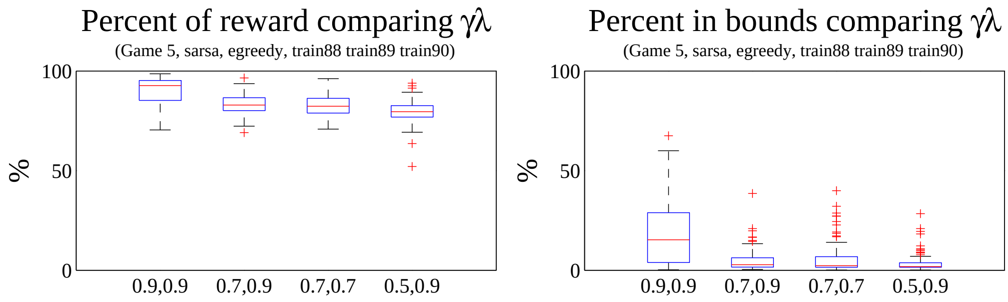
(a) (b)

Figure 9.2: Comparing the performance of η_Q in Game Four testing with Sarsa, and ε-greedy.



(a) (b)

Figure 9.3: Comparing the performance of ε in Game Four testing with Sarsa, and ε-greedy.



(a) (b)

Figure 9.4: Comparing the performance of γ, and λ, in Game Four testing with Sarsa, and ε-greedy.

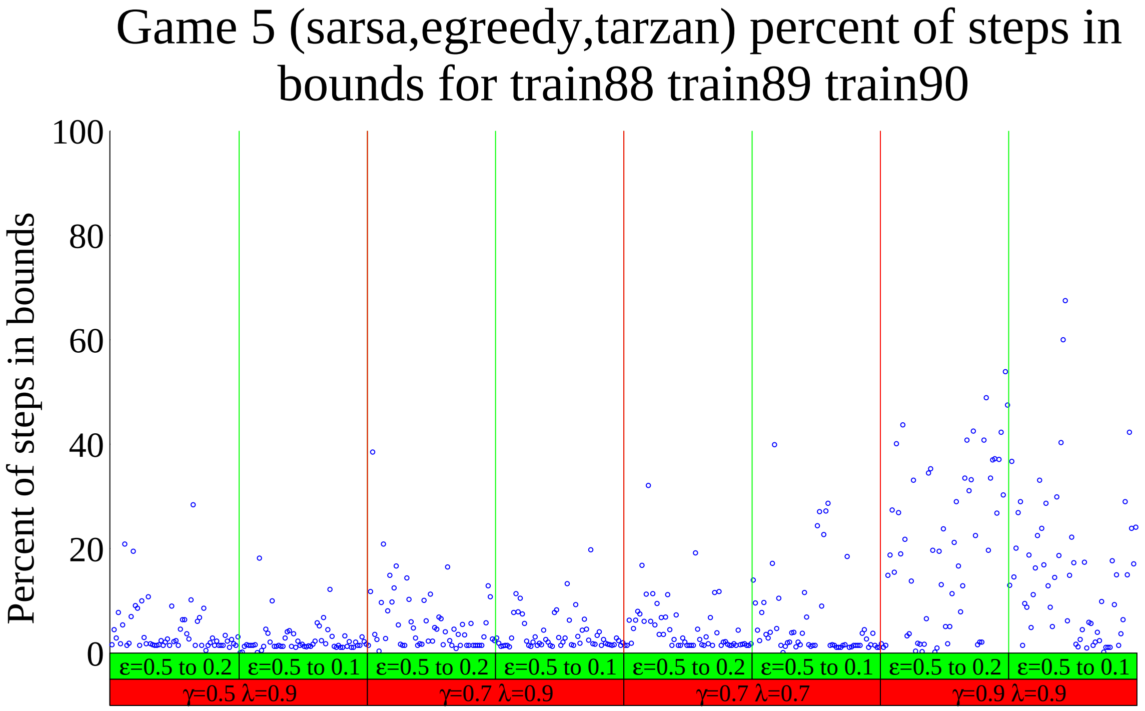


Figure 9.5: All results (percent of steps in bounds) from training games 88, 89, and 90, during Game Four testing with Sarsa, and ε-greedy, separated by ε and $\gamma\lambda$.

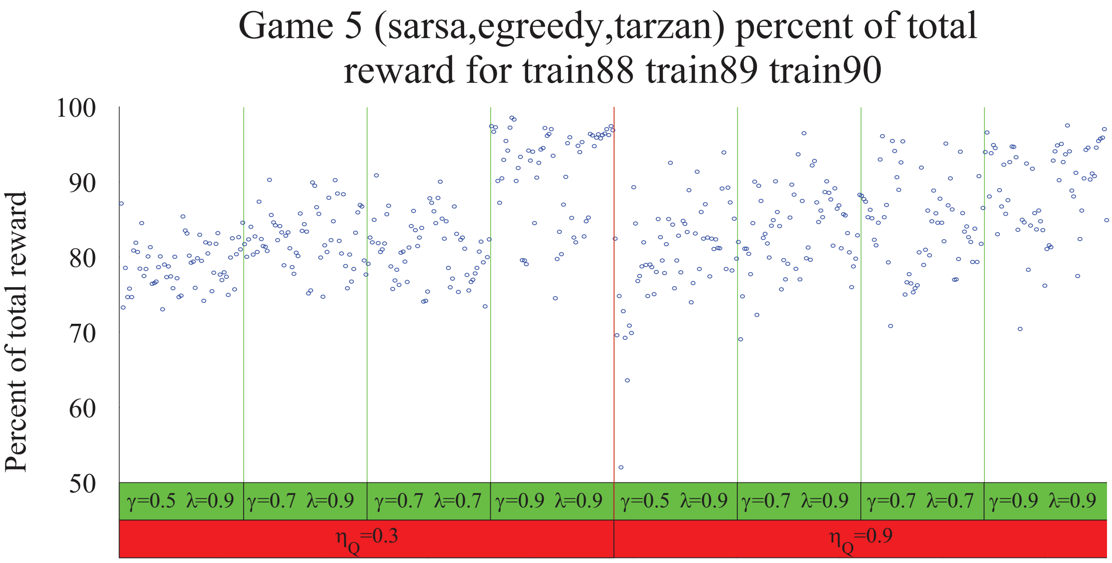
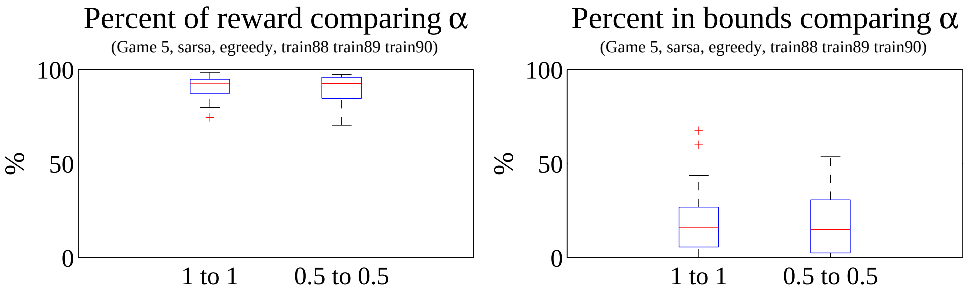
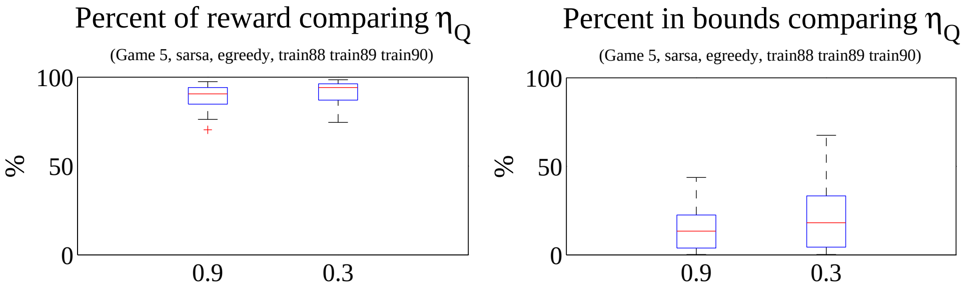


Figure 9.6: All results (percent of reward) from training games 88, 89, and 90, during Game Four testing with Sarsa, and ε-greedy, separated by η_Q, and $\gamma\lambda$.



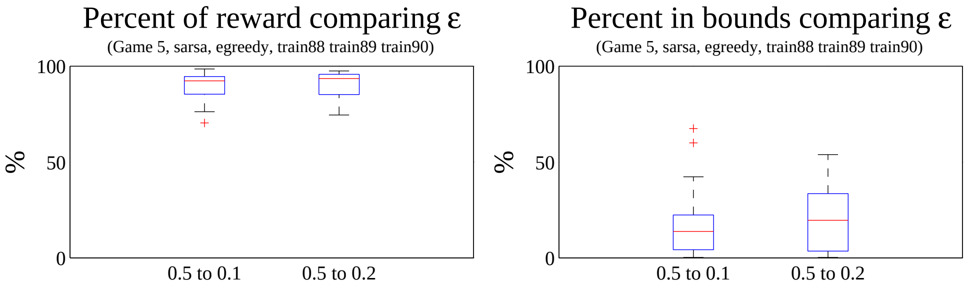
(a) (b)

Figure 9.7: Comparing the performance of α in Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$, are included in the graph.



(a) (b)

Figure 9.8: Comparing the performance of η_Q in Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$, are included in the graph.



(a) (b)

Figure 9.9: Comparing the performance of ε in Game Four testing with Sarsa, and ε-greedy. Only the results with $\gamma = 0.9$, and $\lambda = 0.9$, are included in the graph.

9.3 Discussion

This section will discuss the results of Game 5 in terms of optimal parameters, accuracy of the results, training time and selective qualitative analysis. Tarzan is the only agent being analyzed because Jane, and Bob, both have the same configurations as previous games.

9.3.1 Parameters

The results at a glance in Figures 9.1 to 9.4, show how specific variables perform as a whole, in the last three training games. More specific results can be seen in the scatter plots, showing variable results and interactions. The results in Figure 9.5 are sorted once by ε, and second with $\gamma\lambda$. The resulting graph shows that the best trace-decay value, and discount rate, are $\gamma = 0.9$, and $\lambda = 0.9$, regardless of other parameter values. Sorting the results with $\gamma\lambda$, and then with η_Q, shows that a higher percent of reward can be reached with a lower η_Q, only with $\gamma = 0.9$, and $\lambda = 0.9$, see Figure 9.6.

Further analysis reduces the results to those with $\gamma = 0.9$ and $\lambda = 0.9$. The reduced distributions of results are graphed in Figures 9.7, 9.8, and 9.9.

9.3.2 Consistency

Testing the consistency of the results involves repeating a test under identical conditions and comparing the results. To determine consistency, 10 repeated tests were run with identical parameter values (Table 9.4). The training time is extended from 90 games, to 200 training games. The results of the last 3 training games (train58, train59, and train60), for every test number, can be found in Table 9.5 for the percent of reward, and Table 9.6 for the percent of steps in bounds. Contrary to Game Four, analyzing the consistency for Game Five shows there is larger variation in the percent of reward, than the percent of steps in bounds.

Hn_Q	η_Q	mR	α_S	α_e	ε_s	ε_e	γ	λ
[100]	0.3	-0.1	1.0	1.0	0.5	0.1	0.9	0.9

Table 9.4: Outline of the variables used to test the consistency of the results.

Percent of Reward						
Test	**train196**	**train197**	**train198**	**train199**	**train200**	**mean**
1	95.64	97.65	97.81	89.75	60.51	**88.27**
2	95.62	97.26	96.84	87.41	96.17	**94.66**
3	97.19	97.37	97.48	96.46	98.45	**97.39**
4	94.99	83.72	84.33	91.62	93.76	**89.69**
5	96.57	96.11	96.37	78.11	87.57	**90.95**
6	98.09	85.41	89.77	83.53	85.16	**88.39**
7	97.79	95.04	98.50	96.66	96.45	**96.89**
8	78.35	80.20	84.39	95.90	98.04	**87.38**
9	87.33	76.77	81.22	88.72	86.00	**84.01**
10	94.42	96.62	95.95	96.98	96.21	**96.03**
std	**6.17**	**8.16**	**6.68**	**6.34**	**11.46**	**4.61**

Table 9.5: Statistics measuring consistency of Game Five results (percent of steps in bounds for Tarzan).

Percent of Steps in Bounds						
Test	**train196**	**train197**	**train198**	**train199**	**train200**	**mean**
1	9.56	9.77	9.78	8.97	6.05	**8.83**
2	9.56	9.73	9.68	8.74	9.62	**9.47**
3	9.72	9.74	9.75	9.65	9.84	**9.74**
4	9.50	8.37	8.43	9.16	9.38	**8.97**
5	9.66	9.61	9.64	7.81	8.76	**9.09**
6	9.81	8.54	8.98	8.35	8.52	**8.84**
7	9.78	9.50	9.85	9.67	9.65	**9.69**
8	7.83	8.02	8.44	9.59	9.80	**8.74**
9	8.73	7.68	8.12	8.87	8.60	**8.40**
10	9.44	9.66	9.59	9.70	9.62	**9.60**
std	**0.62**	**0.82**	**0.67**	**0.63**	**1.15**	**0.46**

Table 9.6: Statistics measuring consistency of Game Five results (percent of reward for Tarzan).

9.3.3 Training Time

The results from consistency testing are graphed with respect to training time, in Figures 9.10 and 9.11. Increased training time does improve both the percent of reward, and the percent of steps in bounds, but not significantly compared to the 90 training games tested previously.

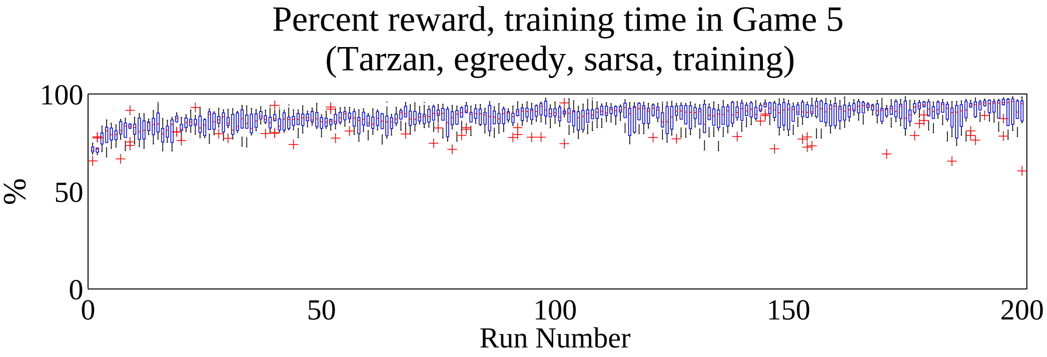


Figure 9.10: Comparing the impact of training time on the percent of reward, in Game Five. Every box includes 10 repeated results, found with parameter values from Table 9.4.

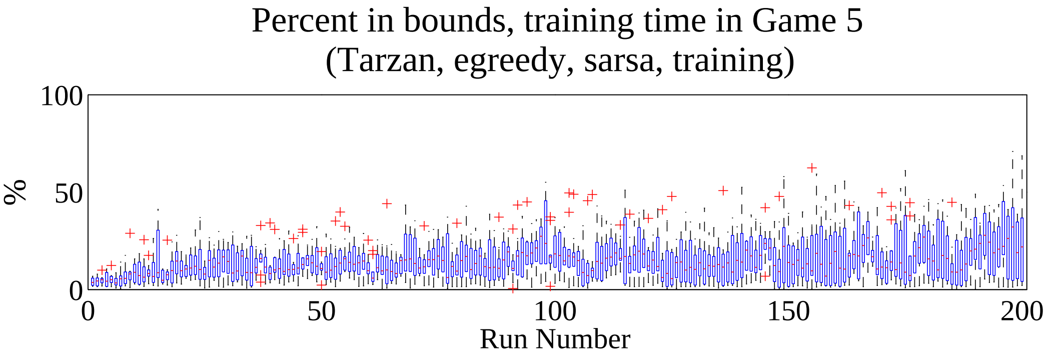
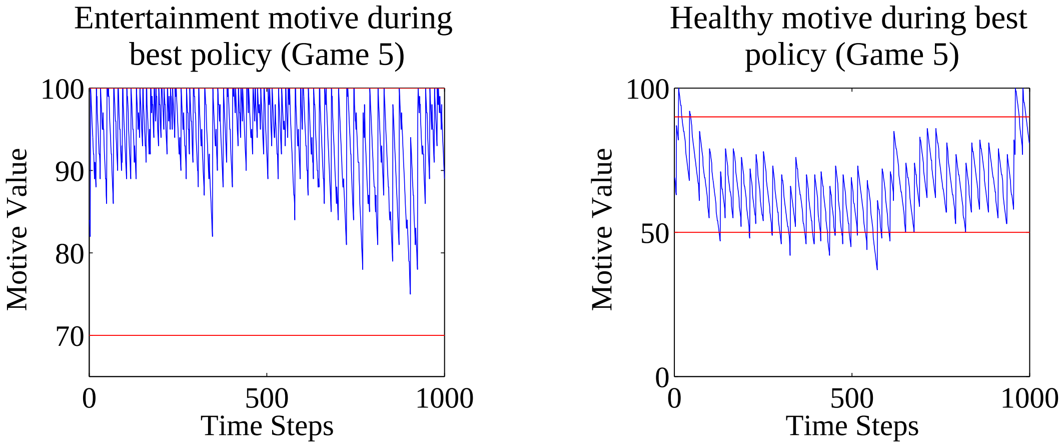


Figure 9.11: Comparing the impact of training time on the percent of steps in bounds, in Game Five. Every box includes 10 repeated results, found with parameter values from Table 9.4.

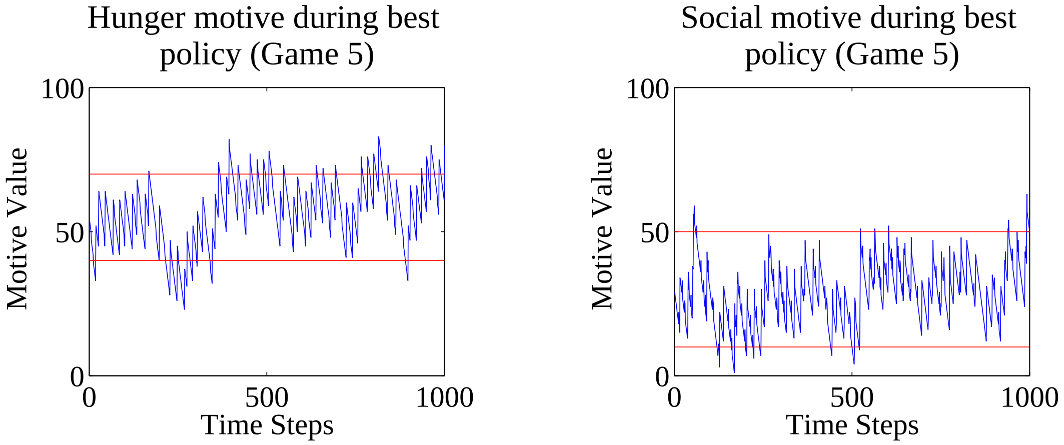
9.3.4 Qualitative Analysis

Taking a closer look at actual action selection sequences shows how the agent learned, or did not learn, how to balance all four motives. The first policy, in Figure 9.12, shows the policy that resulted in the highest percent of reward in the last five training games (*train198* of repeated test number 7). The entertainment motive was easily learned given there are no bounds on the maximum desirable value. Much of the agent's time is spent moving between the grocery store and the bar in search for social interaction from either Bob or Jane. The second policy, in Figure 9.13, shows the policy that resulted in the worst percent of reward in its last five training games (*train200* of repeated test number 1). During the game, Tarzan goes to the grocery store to buy food, but continuously gets talked to by Jane, resulting in much higher social interaction than desired.



(a) The *entertainment motive* values for Tarzan in *train198*, repeated game number 7.

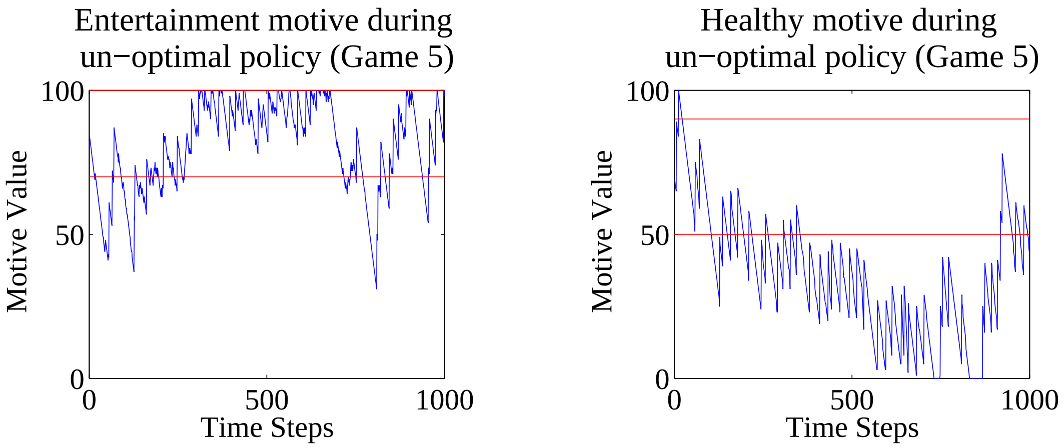
(b) The *healthy motive* values for Tarzan in *train198*, repeated game number 7.



(c) The *hunger motive* values for Tarzan in *train198*, repeated game number 7.

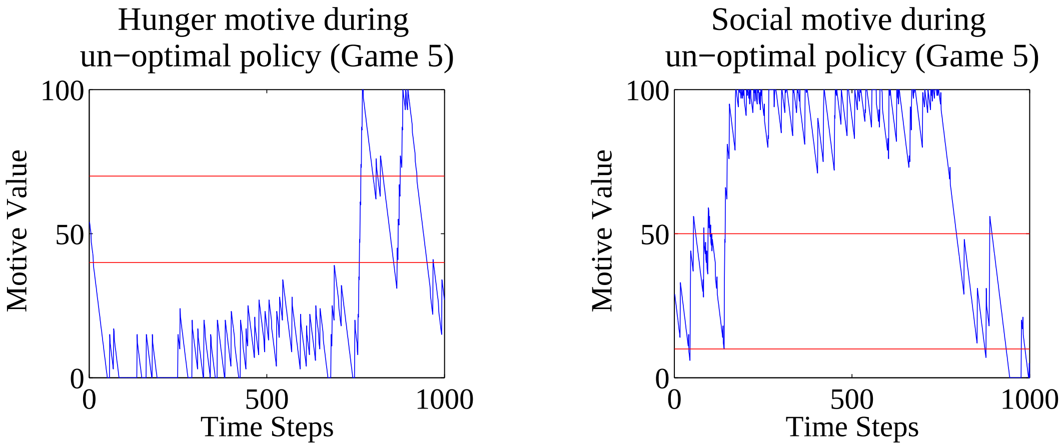
(d) The *social motive* values for Tarzan in *train198*, repeated game number 7.

Figure 9.12: Qualitative analysis of the policy with the highest percent of reward for Tarzan, found using parameters outlined in Table 9.4.



(a) The *entertainment motive* values throughout the final training game with the least amount of reward for Tarzan.

(b) The *healthy motive* values throughout the final training game with the least the amount of reward for Tarzan.



(c) The *hunger motive* values throughout the final training game with the least amount of reward for Tarzan.

(d) The *social motive* values throughout the final training game with the least amount of reward for Tarzan.

Figure 9.13: Qualitative analysis of the policy with the least amount of percent of reward for Tarzan, found using parameters outlines in Table 9.4.

9.4 Summary

Despite having four motivations to satisfy, and having an action with 5 steps of delayed reward, the agent Tarzan can successfully find a policy that achieves up to 98.5% of possible reward. The best results were seen with a lower η_Q, and increased training time.

Chapter 10

Discussion and Conclusion

There were two main goals associated with this work. The first was to create an NPC that learns to take actions that satisfy its motives. The second was to explain how parameter values influenced the agent's learning. After extensive testing, the results show that an RL agent is able to satisfy up to four motives, even when the environment has actions with delayed reward, and inter-agent interactions. After every testing scenario, the optimal parameter values were compared, and explained. This final Chapter explains the most common optimal parameters, the main contributions of this work, and how this work can be extended in the future.

10.1 Discussion

An important factor to consider, for any new game AI, is the ease in which a developer can incorporate the proposed method into their game. Beyond the programming details, it is important to have the modifiable parameters clearly outlined, with guidelines for optimal value selection. This section outlines how every parameter affects the learning, and what values are best to optimize the performance of policies (percent of reward received, and the percent of steps the agent remains in motivational equilibrium).

Balancing between exploration, and exploitation, is an important parameter to consider with any RL task. Exploration is needed for discovering the best possible actions, but exploitation is needed for actually choosing the best possible actions. The needed degree of exploration is dependant upon the complexity of the environment, the required adaptability, and the available training time. The learning task typically begins with a higher exploration rate, which is reduced to a small value over time. RL algorithms behave differently towards exploratory action selections. Sarsa(λ) updates its eligibility trace information, even when a selected action is exploratory in nature. Watkin's Q(λ) will not assign credit, or blame, to exploratory action selections, by clearing all eligibility trace information, after the estimated best action is not selected. The Dyna-Q(λ) based on Watkin's Q(λ), clears the eligibility traces after exploration, but still uses the exploratory action to update its model of the game world.

In general, if an agent has access to multiple motives, and is required to learn actions with delayed reward, starting the exploration rate with a larger value, tends to help with action sequence discovery, in Sarsa(λ), and Q(λ). However, Dyna-Q(λ) performs better when the exploration is kept low throughout the game. The exploration rate should be decreased over time, and the final (smallest) exploration rate should reflect the desired adaptability of the agent. An agent using very little exploration cannot quickly react to changes in the environment, given that it acts on knowledge gained before the changes in the environment were made. Testing shows that decreasing the exploration to a small value will produce policies with a greater number of steps in bounds (learned a good policy), or very few steps in bounds (knowledge is incomplete). Keeping a modest exploration rate after the initial training period, produces policies with smaller total reward, and fewer steps in bounds, but also increases the performance of policies with insufficient knowledge (better worst cases).

Eligibility traces allow credit, or blame, to be assigned to previous state-action pairs. The trace-decay value determines how many past state-action pairs are updated with current value information, and the degree in which they are updated. During testing (Game Two, Three, and Four), a higher trace-decay value produced overall better policies. The larger the number of past state-action pairs updated, the more calculations need to be made every game step. Therefore, higher trace-decay values will increase the time needed to update a game agents logic.

The value of a state action-pair is part current reward, and part future reward. Discounting specifies how much the agent will *look ahead*, considering the future possible reward. The degree of difficulty associated with delayed rewards has an impact on the optimal discount value. For example, Game Two introduced a 2-step delayed reward, with best found policies with $\gamma = 0.5$. Game Three, with the ability to move places, and talk to agents, found best policies with $\gamma \geq 0.5$. Game Four with a large delayed reward, achieved better policies with $\gamma = 0.9$. As a general rule, the more actions that are required to get a reward, the higher the discount rate should be.

With temporal difference methods, the current value of a state-action pair at time t, $Q(s_t, a_t)$, is updated by adding a fraction of the expected value, defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right].$$

This fraction, α, determines how quickly new information will override stored values (old information). Realistically, a reduced the learning rate would require a longer training period, but a high learning rate might produce inferior policies. In this implementation, ANNs are used as the Q-function approximation, with $Q(s_t, a_t)$ as input, and $Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$ as the desired output. The function approximation itself has an associated learning rate (η_Q). If the

ANN learning rate is too large, the function may not be successfully approximated. Lower values of η_Q would take longer to train, but may produce a better approximation of the function. Testing for all Game scenarios showed that higher values of α performed better, or just as well, as lower α values. Testing also shows an increase in performance with a lower η_Q, given longer training times.

The ANN for Q-function approximation, must be configured to match the true Q-function's complexity. During testing, it was determined that only one hidden neuron layer was needed. The optimum number of neurons in the hidden layer varies by Game complexity, but there is a common range that produces good results. In Game One, the optimal number of hidden neurons is between 3, and 100. In Game Two, the optimal number of hidden neurons for Sarsa is between 10, and 100 (optimal at 10), and for Q-learning between 20, and 200 (optimal at 100). In Game Three the optimal number of neurons with Sarsa is between 10, and 250 (optimal at 100), and Q-learning is between 10, and 100 (optimal at 100). Finally, the optimal number of hidden neurons in Game Four is either 50, or 100. For all testing games, the common optimal number of hidden neurons in the only hidden layer, is $Hn_Q = 100$.

The algorithm Dyna-Q uses a model of the environment to simulate real experiences. If the model is wrong, the experiences will also be incorrect. Naturally, when a policy learns from incorrect experience, it performs worst when interacting with the real environment. In most game scenarios tested, Dyna-Q performed worst than Sarsa, and Q-learning, when $pS > 0$. Testing Dyna-Q in Game Four did show improved results, given low exploration rates. Overall, the model function approximation has a hard time accurately approximating the game world, making Dyna-Q less than the ideal choice of RL method.

All the previously mentioned parameters are needed to learn good action selection policies. For every action, there is a ANN that approximates its value, given a current game state. In a real life scenario, there are an infinite number of actions available to an individual, each with their unique consequences. In the game world, there are a finite set of actions available to the agent. Drawing parallels between a game scenario, and a real life situation is convenient. However, it is also important to understand that some consequences for real actions may not be specified in the game world, either because they have no relevance, or they were overlooked. For example, in most of the game scenarios, the agent must open the fridge to make the food items available. In real life, cold air escapes from the fridge for as long as the fridge is open. A person will open the fridge in short amounts of time, to keep the food inside from spoiling. During testing, because there are no negative consequences (reinforcements) associated with opening the fridge, it is completely understandable that the agent will choose to continuously open the fridge, until such a time where the agent becomes hungry. All the necessary reinforcements need to completely define the learning task, for the agent to learn as the developer envisions.

للاستشارات المنارة

10.2 Main Contributions

Programming believable game agents is a non-trivial problem, that involves creating an NPC that acts *realistically* in the eyes of the player. A complete believable agent would have thoughts, desires, and emotions, that are the manifestation of different personalities. They would also incorporate full interactivity, with realistic body language, and natural language processing. In video Game AI, behaviour selection is the most important aspect of believable agents.

There are many approaches that address realistic action selection, and they most often involve personality traits, or social understanding. These methods use facets of human personality to create realistic, and entertaining NPCs. While these methods are emergent, their actions are entirely dependent on models of personality, and even general roles. Emergent behaviour is possible, but learning is not part of the process.

Allowing the agent to learn during a game is an important part of believability. Beyond simple learning, machine learning algorithms, reinforcement learning in particular, have been influenced by concepts of personality. RL has been extended to form emotion driven RL, and motivated RL for hierarchical skill learning. However, these extensions to RL are focused on better learning of general actions, not on creating realistic NPCs.

The agent learning method proposed in this thesis, introduces reinforcement learning guided by an agent's motives, with the purpose of creating realistic NPC behaviour. A person's motives can be split into two parts: the desired values that remain constant, and the current values that change according to situations, and actions. The current motive values are incorporated into the agent's state of the game world. The actual game state used for learning includes information about places, other agents, and the state of the agent's objects. The reinforcements are calculated using the constant desired motive values (minimum, and maximum), in response to the agent's current motive values. As a result, the agent learns what behaviours to take, to fully satisfy its motives.

10.3 Future Work

The method developed in this thesis, is very much a starting point for a more advanced realistic agent. There are many opportunities to improve on the proposed method, including the addition of more advanced reinforcement learning techniques, and the addition of other motive elements.

In theory, an individual's motives describe a continuum of possible values, with no limits in either direction. When motives were adopted for use in this method, the motive continuum had to become a finite scale, to allow learning, and computation of rewards. The range of possible motive values is much smaller in practice (between 0 and 100), than in real-life ($-\infty$ to ∞). All motives decay at the same rate, according to a linear decay function. If different decay functions were available, it would add another way of making motives unique across multiple agents. Variable decay functions could also be used to emulate an infinite motive range, given a slow enough decay function, linear, or otherwise.

Currently, the actions available to the agent are very basic (e.g. open fridge, eat food, watch TV). It is possible for sequences of actions to become skills, or options, through the use of motivation reinforcement learning (MRL). An option would have an associated Q-value, and reward, the same as any action. With the addition of option models, the agent could *look ahead,* and determine the likely value of performing an option, according to the estimated changes to its motive values.

The reinforcement function is meant to provide more positive (less negative) reward when multiple motives have been addressed, but there are no noticeable differences between addressing different motives. The reinforcement function could be modified to assign more importance to the motives that are further from their bounds. If two motives can be modified in the same amount, more reward should be given when the motive that is further from its bounds is addressed, and less to the other. This change might decrease the situation where one motive is completely ignored over satisfying others.

The addition of more motive elements, and more advanced RL techniques, would enable more diverse agent specifications, and more realistic action sequences. Ultimately, this method also requires a more comprehensive qualitative analysis, aimed to determine the actual believability of a desire driven game agent.

Bibliography

[1] Aristotle, J.A.K. Thomson, and H. Tredennick. *The ethics of Aristotle: the Nicomachean ethics*. Penguin classics. Pengüin, 1976.

[2] C. Bailey and M. Katchabaw. An emergent framework for realistic psychosocial behaviour in non player characters. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pages 17–24. ACM, 2008.

[3] C. Bailey, J. You, G. Acton, A. Rankin, and M. Katchabaw. Believability through psychosocial behaviour: Creating bots that are more engaging and entertaining. *Believable Bots: Can Computers Play Like People?*, page 29, 2012.

[4] A. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Developmental Learning (ICDL '04)*, LaJolla CA, 2004.

[5] E. Bevacqua, E. de Sevin, C. Pelachaud, M. McRorie, and I. Sneddon. Building credible agents: Behaviour influenced by personality and emotional traits. In *Proceedings of International Conference on Kansei Engineering and Emotion Research*, 2010.

[6] M. Botvinick, Y. Niv, and A. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113:262–280, 2009.

[7] L. Cañamero. Designing emotions for activity selection in autonomous agents. *Emotions in humans and artifacts*, 115:148, 2003.

[8] N. Chentanez, A. Barto, and S. Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2004.

[9] T. Doce, J. Dias, R. Prada, and A. Paiva. Creating individual agents through personality traits. In *Intelligent Virtual Agents*, pages 257–264. Springer, 2010.

[10] E. Doirado and C. Martinho. I mean it!: detecting user intentions to create believable behaviour for virtual agents in games. In *Proceedings of the 9th International Conference on Autonomous*

Agents and Multiagent Systems: volume 1-Volume 1, pages 83–90. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[11] E. Douglas-Cowie, R. Cowie, C. Cox, N. Amier, and DKJ Heylen. The sensitive artificial listner: an induction technique for generating emotionally coloured conversation. 2008.

[12] B. Ellinger. Artificial Personality: A Personal Approach to AI. *AI Game Programming Wisdom 4*, pages 671–683, 2008.

[13] D. C. Funder. Personality. *Annual Review of Psychology*, 52(1):197–221, 2001. PMID: 11148304.

[14] L. Galway, D. Charles, and M. Black. Machine learning in digital games: a survey. *Artif. Intell. Rev.*, 29:123–161, April 2008.

[15] L. Gruenwoldt, M. Katchabaw, and S. Danton. A realistic reaction system for modern video games. In *the Proceedings of the DiGRA 2005 Conference: Changing Views–Worlds in Play*, 2005.

[16] A. Guye-Vuilleme and D. Thalmann. A high-level architecture for believable social agents. *Virtual Reality*, 5(2):95–106, 2000.

[17] M. Harmon and S. Harmon. Reinforcement Learning: A Tutorial. http://citeseer.ist.psu.edu/harmon96reinforcement.html, 1996.

[18] C. L. Hull. Principles of behavior (1943). *New York Appleton–Century-Crofts.*

[19] M. Joselli and E. Clua. GPUWars: Design and Implementation of a GPGPU Game. In *Games and Digital Entertainment, Brazilian Symposium*, pages 132–140, 2009.

[20] A.B. Loyall. *Believable agents: building interactive personalities.* PhD thesis, Stanford University, 1997.

[21] R. Marinier III and J. Laird. Emotion-Driven Reinforcement Learning. 2009.

[22] M. McPartland and M. Gallagher. Reinforcement Learning in First Person Shooter Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):43–56, 2011.

[23] K. Merrick. Modeling motivation for adaptive nonplayer characters in dynamic computer game worlds. *Computers in Entertainment (CIE)*, 5(4):5:1–5:32, 2008.

[24] K. Merrick. A computational model of achievement motivation for artificial agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 1067–1068, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[25] K. Merrick and M. Maher. Motivated reinforcement learning for non-player characters in persistent computer game worlds. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, ACE '06, New York, NY, USA, 2006. ACM.

[26] K. Merrick and M. Maher. *Motivated Reinforcement Learning: Curious Characters for Multiuser Games*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[27] K. Merrick and K. Shafi. Achievement, affiliation, and power: Motive profiles for artificial agents. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 19(1):40–62, February 2011.

[28] T.M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[29] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *NIPS*, 1997.

[30] R. Prada, G. Raimundo, J. Dimas, C. Martinho, J. F. Peña, M. Baptista, P. A. Santos, and L. L. Ribeiro. The role of social identity, rationality and anticipation in believable agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1175–1176. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[31] S. Rabin. *Introduction to game development*. Game development series. Course Technology, 2010.

[32] S. Read, B. Monroe, A. Brownstein, Y. Yang, G. Chopra, and L. Miller. A neural network model of the structure and dynamics of human personality. *Psychological review*, 117(1):61–92, 2010.

[33] S. Reiss. Why people turn to religion: A motivational analysis. *Journal for the Scientific Study of religion*, 39(1):47–52, 2000.

[34] S. Reiss. Multifaceted nature of intrinsic motivation: The theory of 16 basic desires. *Review of General Psychology*, 8(3):179, 2004.

[35] S. Reiss and S. Havercamp. Toward a comprehensive assessment of fundamental motivation: Factor structure of the Reiss Profiles. *Psychological assessment*, 10(2):97–106, 1998.

[36] S. Reiss, J. Wiltz, and M. Sherman. Trait motivational correlates of athleticism. *Personality and Individual Differences*, 30(7):1139–1145, 2001.

[37] P. Rizzo, M. Veloso, M. Miceli, and A. Cesta. Personality-driven social behaviors in believable agents. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents*, pages 109–114, 1997.

[38] P. Ruijten, C. Midden, and J. Ham. I didn't know that virtual agent was angry at me: Investigating effects of gaze direction on emotion recognition and evaluation. In *Persuasive Technology*, pages 192–197. Springer, 2013.

[39] K. R. Scherer. *Appraisal considered as a process of multilevel sequential checking*, volume 92. 2001.

[40] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. 1998.

[41] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[42] I. Szita, M. Ponsen, and P. Spronck. Effective and Diverse Adaptive Game AI. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):16 –27, march 2009.

VITA

NAME: Jacquelyne T. Forgette

PREVIOUS POSITIONS

University of Western Ontario — September 2010 to May 2012

Teaching Assistant in the Department of Computer Science

Nipissing University — January 2010 to April 2010

Teaching Assistant in the Department of Computer Science

Nipissing University — April 2009 to April 2010

Research Assistant in the Department of Physical Education and English

EDUCATION

University of Western Ontario — September 2010 to present

M.Sc. (Computer Science) expected September 2013,
Supervisor: Mike Katabaw

Nipissing University — September 2006 to April 2010

Hons. B.Science. (Computer Science with Distinction) received June 2010

AWARDS

Natural Sciences and Engineering Research Council of Canada (NSERC) Postgraduate Scholarship (PGS-M) (2010 - 2011)

Western Graduate Research Scholarship (2010 - 2011)

PUBLICATIONS

J. Forgette, R. Wachowiak-Smolikova, and M. Wachowiak. "Implementing Independent Component Analysis in General-Purpose GPU Architectures." In Digital Information Processing and Communications, pp. 233-243. Springer Berlin Heidelberg, 2011.

J. Forgette, R. Wachowiak-Smolikova, and M. Wachowiak. "Scalable parallel implementation of independent components analysis on the graphics processing unit." In Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on, pp. 000912-000916. IEEE, 2011.

J. Forgette, R. Wachowiak-Smolikova, and M. Wachowiak. "Influence maps For Facilitating Tactical Engagement Decisions In Real-Time Strategy Games." International Journal of Intelligent Games & Simulation 6, no. 1 (2010): 8.