

COMPUTATIONAL THINKING PATTERN ANALYSIS:
A PHENOMENOLOGICAL APPROACH TO COMPUTE
COMPUTATIONAL THINKING

by

KYU HAN KOH

B. E., Soongsil University, 2004

M.S., Auburn University, 2007

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirement for the degree of
Doctor of Philosophy
Department of Computer Science
Institute of Cognitive Science
2014

UMI Number: 3621357

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3621357

Published by ProQuest LLC (2014). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

This thesis entitled:

*Computational Thinking Pattern Analysis: A Phenomenological Approach to
Compute Computational Thinking*

written by Kyu Han Koh

has been approved for the Department of Computer Science
and the Institute of Cognitive Science

Alexander Repenning, Committee Chair

Clayton Lewis

David Webb

Michael Klymkowsky

Tom Yeh

Date _____

The final copy of this thesis has been examined by the signatories, and we
find that both the content and the form meet acceptable presentation standards
of scholarly work in the above mentioned discipline.

IRB protocol # 0808.21 and #11-0379

Abstract

Koh, Kyu Han (Ph.D., Computer Science & Cognitive Science)

Computational Thinking Pattern Analysis: A Phenomenological Approach to Compute Computational Thinking

Thesis directed by Professor Alexander Repenning

Since the 1990s there have been multiple efforts to fix the broken pipeline at the K-12 level in computer science education, and most of those efforts have focused on the student motivational factor. The results of many studies in computer science education indicate that student motivation in computer science has been successfully increased by those efforts (Perrone et al., 1995; Walter et al., 2007; Kelleher et al., 2005; Kelleher et al., 2007; Maloney et al., 2008; Resnick et al., 2009), but most of them have failed to address educational benefits of these efforts. I believe that this biased tendency of CS education research has been caused by the lack of an adequate instrument to measure students' achieved skills with learning objectives at the semantic level. In other words, the right assessment instrument should be able to assess not only student learning skills but also achieved learning objectives: what kinds of knowledge students have learned through their activities in the class. Student learning skills may be measured with existing tools such as grading rubrics, but they are extremely time consuming and have a limited functionality to provide necessary educational feedback such as student learning progression.

I developed a learning data analysis tool to measure student-learning skills and represent students' learning achievements at the semantic level through phenomenological analysis in real-time. This concept uses a LSA (Landauer, 2003)

inspired technique, multiple high dimensional cosine calculations to analyze semantic meanings of the pre-defined subject/phenomena in a given. Theoretically, this idea can be applied to several different domains such as natural language processing and visual end user programming. Therefore, this idea can be employed to build a learning assessment tool for computer science (CS) and/or computational thinking (CT) education where visual programming is widely adopted.

As a semantic assessment tool for CS/CT learning, I propose a method, Computational Thinking Pattern Analysis (CTPA) in which nine canonical computational thinking patterns (Koh et al., 2010) work as pre-defined phenomena within a programmed artifact's context. The CTPA measures students' learning of skills (how well they have learned a skill) and students' learning of objectives (how well they have learned certain objectives) at the semantic level through phenomenological analysis from student-programmed artifacts in real-time. The outcomes of CTPA can be used to provide valid and useful educational feedback to educators and learners in CS/CT education such as measuring and tracking student learning outcomes.

Semantic assessment in CS/CT education would be able to provide better individual feedback and faster learning assessment to students and teachers by measuring student skills and challenges and analyzing learning objectives at the semantic level. This kind of feedback can be used to determine when and how teachers can expand students' learning capability in accordance with the theories of the Zone of Proximal Development and Flow (Basawapatna et al., 2013). A validated CTPA will contribute to the study of learning theory, professional development, and educational data mining by providing

empirical data in order to refine the current conceptual framework of educational systems.

This research suggests a method that can assess students' learning skills, provide effective learning guidelines, and compute students' learning outcomes. This type of method, which cannot be found widely, can be used to create real cyberlearning systems that help large numbers of teachers and students to learn computational thinking.

Dedications

To my wife, Jeehye, and my son, Andrew Yunsung.

Acknowledgments

I would like to thank,

My wife, Jeehye, for her unwavering support and encouragement through all these years.

My son, Andrew Yunsung, who entered my life in the middle of this process. He was the reason for me to find new purpose in life and finish my Ph.D. program;

My advisor, Alexander Repenning for encouraging and guiding me through the Ph.D. process and for exposing me to countless educational and research experiences along the way;

My dissertation committee members, Alexander Repenning, Clayton Lewis, David Webb, Michael Klymkowsky, Kris Gutierrezand, and Tom Yeh for all their vital advice;

My parents Dong Hee Koh and Mi Yeon Kim for all their constant support and encouragement to pursue my Ph.D.;

Everyone at AgentSheets Inc. and my research group who helped me, including Nadia Repenning, Fred Gluck, Andri Ioannidou, Vicki Bennett, Ashok Basawapatna, Hilarie Nickerson, Yasko Endo, and Michael Minerva;

My dear friends, Jeff Hoehl, Jeeun Kim, Hyunjoo Oh, Inki Jin, and Junho Ahn for their support.

Table of Contents

CHAPTER 1. INTRODUCTION	1
1.1. RESEARCH OBJECTIVES AND RESEARCH QUESTIONS	5
1.2. DISSERTATION OUTLINES	7
CHAPTER 2. CONCEPTUAL FRAMEWORK.....	9
CHAPTER 3. RELATED RESEARCH.....	13
3.1. COMPUTATIONAL THINKING.....	13
3.2. COMPUTATIONAL THINKING PATTERNS	16
3.3. SOCIAL CREATIVITY THROUGH END-USER PROGRAMMING	20
3.4. ASSESSMENT IN VISUAL END-USER PROGRAMMING.....	23
CHAPTER 4. COMPUTATIONAL THINKING PATTERN ANALYSIS.....	25
4.1. SCALABLE GAME DESIGN (SGD) PROJECT	25
4.2. SCALABLE GAME DESIGN ARCADE (SGDA).....	26
4.3. COMPUTATIONAL THINKING PATTERN ANALYSIS.....	31
4.4. COMPUTATIONAL THINKING PATTERN ANALYSIS GRAPH	33
4.5. THE SENSITIVITY OF CTPA	35
4.5.1. CTPA SIMILARITY VS. CODE SIMILARITY	36
4.5.2. PROGRAMMING DIVERGENCE CALCULATION.....	37
CHAPTER 5. RESEARCH VALIDATION	46
5.1. COMPUTATIONAL THINKING SKILL PROGRESSION	46
5.2. ASSESSMENT VALIDATION	49
5.2.1. 2012 CLASS CONCURRENT VALIDITY RESULTS	50
5.2.2. DEMONSTRATED SKILL FOR INDIVIDUAL GAMES.....	52
5.2.3. COMPREHENSIVE SKILL ACROSS SEVERAL GAMES.....	53
5.2.4. INTER-RATER AGREEMENT	54
5.2.5. 2013 CLASS CONCURRENT VALIDITY RESULTS	54
5.2.6. PREDICTIVE VALIDITY RESULTS.....	55
5.2.7. SUMMARY OF CONCURRENT AND PREDICTIVE VALIDATION EVALUATION.....	57
5.3. FACE AND CONSTRUCT VALIDITY EVALUATIONS	58
5.3.1. EVALUATION WITH PROGRAMMING CODES.....	58
5.3.2. EVALUATION RESULTS	60
5.3.3. EVALUATION WITH PROPORTIONAL PROGRAMMING.....	64
5.3.4. EVALUATION RESULTS WITH PROPORTIONAL PROGRAMMING.....	67
CHAPTER 6. APPLICATIONS OF COMPUTATIONAL THINKING PATTERN ANALYSIS.....	69

6.1. EARLY INDICATOR OF TRANSFER	69
6.1.1. SUMMARY OF ASSESSING STUDENT PRODUCTS USING THE CTPA	75
6.2. LEARNING SKILL ASSESSMENT.....	76
6.2.1. FINDINGS: MIDDLE SCHOOL VS. COLLEGE STUDENT COMPARISON	79
6.2.2. SUMMARY OF ASSESSING STUDENT PRODUCTS USING THE CTPA	82
6.3. DIVERGENCE IN PROGRAMMING.....	83
6.3.1. THREE CLASS CONDITIONS.....	83
6.3.2. FINDINGS	84
6.3.3. DIVERGENCE CALCULATION GRAPH	85
6.3.4. SUMMARY OF ASSESSING STUDENT PRODUCTS USING THE CTPA	87
6.4. ONLINE ASSESSMENT OF COMPUTATIONAL THINKING (ONACT).....	88
6.4.1. SUMMARY OF ASSESSING STUDENT PERFORMANCE USING ONACT	92
CHAPTER 7. DISCUSSION	94
CHAPTER 8. CONCLUSION	99
CHAPTER 9. REFERENCES	101
APPENDIX A. CTPA GRAPHS FOR EACH CT PATTERNS	A-106
APPENDIX B. CTPA EVOLUTION CHART	B-111
APPENDIX C. FACE AND CONSTRUCT VALIDITY SURVEY QUESTIONNAIRE.....	C-125
APPENDIX D. FACE VALIDITY WITH PROPORTIONS SURVEY QUESTIONNAIRE.....	D-140
APPENDIX E. FOUR STANDARD GAMES WITH 13 DIMENSIONAL CTPA	E-141

Table of Figures

FIGURE 1: FOUR ELEMENTS OF LEARNING ENVIRONMENTS (BRANSFORD ET AL., 1999).....	4
FIGURE 2. ZONES OF PROXIMAL FLOW: CT CURRICULUM THAT BALANCES CT CHALLENGES WITH CT SKILLS (BASAWAPATNA ET AL., 2013).	11
FIGURE 3. RELATIONSHIPS BETWEEN THE FIVE COMPUTATIONAL THINKING CATEGORIES IN THE NRC REPORT	16
FIGURE 4. GHOSTS USE HILL CLIMBING ON PACMAN'S DIFFUSED SCENT (PICTURED AROUND PACMAN) TO TRACK DOWN PACMAN	19
FIGURE 5 SCRATCH WEBSITE	21
FIGURE 6 AN EXAMPLE PROJECT OF CYBERMOD	23
FIGURE 7: THE ASSIGNMENT GALLERY OF SGDA	28
FIGURE 8: THE MAIN PAGE OF SGDA.....	29
FIGURE 9: THE INDIVIDUAL PAGE OF SGDA. INDIVIDUAL PAGE ILLUSTRATES THE SCREENSHOT OF THE GAME (UPPER LEFT), RUN AND DOWNLOAD BUTTON (UPPER RIGHT), THE GAME'S SIMILARITY SCORE COMPARED TO FOUR TUTORIAL GAMES (MIDDLE RIGHT), A SIMILARITY SCORE MATRIX SHOWING GAMES PROGRAMMED SIMILARLY TO THE SUBMITTED GAME, AND THE CTPA GRAPH (BOTTOM RIGHT & LEFT).....	30
FIGURE 10: COMPUTATIONAL THINKING PATTERN ANALYSIS USING MULTIPLE HIGH DIMENSIONAL COSINE CALCULATION	31
FIGURE 11: COMPUTATIONAL THINKING PATTERN ANALYSIS GRAPH OBTAINED BY CTPA	32
FIGURE 12: A CTPA GRAPH FROM AN EXAMPLE FROGGER GAME.....	34
FIGURE 13 CTPA RESULTS OF AN AGENTSHEETS PROJECT PROGRAMMED WITH NON-COMPUTATIONAL THINKING PATTERN CONSTRUCTORS ONLY. IT RESULTS IN A VALUE OF 0 FOR ALL CT PATTERNS.....	35
FIGURE 14: CTPA DIVERGENCE IN TERMS OF COMPUTATIONAL THINKING PATTERN. X-AXIS REPRESENTS THE SUBMISSION SEQUENCE BASED ON TIME. Y-AXIS REPRESENTS THE CTPA DIVERGENCE VALUE. ...	39
FIGURE 15 FROG AGENT IN GAME 1	40
FIGURE 16 ADDITIONAL PROGRAMMING OF FROG AGENT IN GAME 2	41

FIGURE 17 CTPA GRAPH COMPARISON OF GAME 1 (GREEN) AND GAME 2 (BROWN).....	43
FIGURE 18 CTPA GRAPH COMPARISON OF GAME 1 (GREEN) AND GAME 3 (BROWN).....	44
FIGURE 19 CTPA GRAPH COMPARISON OF GAME 1 (GREEN) AND GAME 4 (BROWN).....	44
FIGURE 20 CTPA GRAPH COMPARISON OF GAME 3 (GREEN) AND GAME 4 (BROWN).....	45
FIGURE 21 2012 CLASS SPEARMAN RANK CORRELATION CHART	53
FIGURE 22 2013 CLASS SPEARMAN RANK CORRELATION CHART	55
FIGURE 23. PREDICTIVE VALIDITY EVALUATION FROM 2012 CLASS.....	57
FIGURE 24: CURSOR CONTROL PROGRAMMING (CORRECT METHOD)	59
FIGURE 25: CURSOR CONTROL PROGRAMMING (INCORRECT METHOD)	59
FIGURE 26: CTPA GRAPH OF CURSOR CONTROL PATTERN	60
FIGURE 27: A CTPA GRAPH OF A GAME OF FROGGER.....	66
FIGURE 28 AN EXAMPLE OF ABSORPTION PATTERN PROGRAMMING.....	66
FIGURE 29 AN EXAMPLE OF COLLISION PATTERN PROGRAMMING	66
FIGURE 30 A SCREENSHOT OF SOKOBAN	71
FIGURE 31 A CTPA GRAPH OF SOKOBAN	72
FIGURE 32 A SCREENSHOT OF SIMS	72
FIGURE 33 A CTPA GRAPH OF SIMS.....	73
FIGURE 34 A SCREENSHOT OF CHAOS THEORY SIMULATION	73
FIGURE 35 A CTPA GRAPH OF CHAOS THEORY SIMULATION	74
FIGURE 36 COMPARISON OF CTPA GRAPHS: DEPICTS THE SIMS-SOKOBAN COMBINATION	74
FIGURE 37 1ST TO 3RD GAMES IN A MIDDLE SCHOOL AND IN A COLLEGE CLASS	78
FIGURE 38 5 GAMELET MADNESS PROJECTS IN A COLLEGE CLASS.....	79
FIGURE 39 SKILL PROGRESSION COMPARISON BETWEEN COLLEGE AND MIDDLE SCHOOL STUDENTS WITH COMPREHENSIVE SKILL SCORES.....	80
FIGURE 40 SCATTERED DIVERGENCE CALCULATION GRAPH.....	85
FIGURE 41 EXAMPLE OF ONACT ASSESSMENT DASHBOARD SHOWING EVERY STUDENT’S PERFORMANCE IN A GIVEN CLASSROOM.	90
FIGURE 42 INDIVIDUAL GAME SUBMISSION PAGE OF SCALABLE GAME DESIGN ARCADE	91

FIGURE 43 THE COMPUTATIONAL THINKING PATTERN ANALYSIS FORENSICS GRAPH	92
FIGURE 44 FROGGER SUBMISSIONS IN THE 2012 CLASS	95
FIGURE 45 FROGGER DESIGN LAYOUT OF GAME 1	97
FIGURE 46 FROGGER DESIGN LAYOUT OF GAME 2	97
FIGURE 47 BEHAVIOR PROGRAMMING OF GAME 1.....	98
FIGURE 48 BEHAVIOR PROGRAMMING OF GAME 2.....	98
FIGURE 49 CURSOR CONTROL PATTERN	A-106
FIGURE 50 GENERATION PATTERN.....	A-106
FIGURE 51 ABSORPTION PATTERN	A-107
FIGURE 52 COLLISION PATTERN.....	A-107
FIGURE 53 TRANSPORTATION PATTERN.....	A-108
FIGURE 54 PUSH PATTERN	A-108
FIGURE 55 PULL PATTERN.....	A-109
FIGURE 56 DIFFUSION PATTERN.....	A-109
FIGURE 57 HILL CLIMBING PATTERN	A-110
FIGURE 58: PROGRAMMING BEHAVIOR IN AGENT 1.....	C-129
FIGURE 59 PROGRAMMING BEHAVIOR IN AGENT 2.....	C-130
FIGURE 60 PROGRAMING BEHAVIOR IN AGENT 1.....	C-131
FIGURE 61 PROGRAMMING BEHAVIOR IN AGENT 2.....	C-132
FIGURE 62 PROGRAMING BEHAVIOR IN AGENT 1.....	C-133
FIGURE 63 PROGRAMING BEHAVIOR IN AGENT 2.....	C-134
FIGURE 64 PROGRAMING BEHAVIOR IN AGENT 1.....	C-135
FIGURE 65 PROGRAMING BEHAVIOR IN AGENT 2.....	C-136
FIGURE 66 FROGGER WITH 13 DIMENSIONAL CTPA	E-141
FIGURE 67 EXAMPLE SOKOBAN WITH 13 DIMENSIONAL CTPA	E-142
FIGURE 68 TUTORIAL SOKOBAN WITH 13 DIMENSIONAL CTPA.....	E-142
FIGURE 69 SPACE INVADERS WITH 13 DIMENSIONAL CTPA.....	E-143
FIGURE 70 SIMS WITH 13 DIMENSIONAL CTPA.....	E-143

Table of Tables

TABLE 1: COMPUTATIONAL THINKING PATTERNS IN CURRICULUM GAMES.....	19
TABLE 2 FOUR GAMES' CTPA AND PROGRAMMING INFORMATION.....	39
TABLE 3: EXAMPLE OF LEARNING SKILL SCORE CALCULATION.....	49
TABLE 4 FOUR BASIC GAMES AND SPEARMAN RANK CORRELATION CHARTS FOR THE 2012 CLASS.....	52
TABLE 5: EVALUATION RESULTS	63
TABLE 6 EVALUATION RESULTS	67
TABLE 7: DIVERGENCE CALCULATION SCORE IN EACH CLASS.....	86
TABLE 8 FOUR STANDARD GAMES AND FOUR DIFFERENT CTPAS.....	B-113
TABLE 9 FROGGER WITH FOUR CTPAS.....	B-116
TABLE 10 SPACE INVADERS WITH FOUR CTPAS.....	B-118
TABLE 11 SIMS WITH FOUR CTPAS	B-120
TABLE 12 TUTORIAL SOKOBAN WITH FOUR CTPAS.....	B-122
TABLE 13 EXAMPLE SOKOBAN WITH FOUR CTPAS.....	B-124

Chapter 1. Introduction

Since the early 1990's there have been multiple efforts to use end-user visual programming with video game creation in an effort to teach programming. Examples of these visual programming tools include AgentSheets, Alice, and Scratch (Perrone et al., 1995; Repenning et al., 2000; Kelleher et al., 2005; Kelleher et al., 2007; Maloney et al., 2008; Resnick et al., 2009). The inherent appeal of video games to students gives teachers an entertaining way to introduce the otherwise technical practice of programming (Sturtevant et al., 2008; Squire et al., 2003). These tools have multiple advantages over conventional programming languages such as C++ or Java (Peppet et al., 2007). For instance, to create a simple game, complete with graphics, in C++ or Java can take weeks or even months of learning. In contrast, according to my research, AgentSheets allows students with no prior programming experience to create their first game in five hours, in numerous classes at different levels ranging from middle school to graduate school (Basawapatna et al., 2010). As I observed, end-user visual programming with video game creation allows the students to make a program/game in a simpler fashion, and it gives a motivational benefit to the students (Basawapatna et al., 2010). From this observation, we can derive two different benefits from end-user programming approach in computer science education: educational benefits and motivational benefits. Those two benefits were a focus of much computer science education research in the 1990s.

Besides the motivational and the educational benefits of end-user programming, Web 2.0 technologies help the end-users enjoy the benefits of web collaboration and

social creativity. For example, Scratch has a website in which the end-users can share their game/project ideas and actual game/project codes, feedback, etc. Thanks to this advantage of Web 2.0, Scratch users are able to update, modify, and recreate video games, story-telling animations and/or science simulations based on other people's projects. The Scratch website has been able to collect around 615,700 Scratch projects ranging from video games to science simulations (Monroy-Hernández, 2009). This webpage is a great exhibition displaying open-ended projects without spatial and temporal limitations. Additionally, several Scratch projects have been designed, programmed and posted as the results of web-based collaboration by multiple users without geographical limitations (Monroy-Hernández, 2009). Also, through the Behavior Exchange (Repenning et al., 1997), Scratch users can easily build up their own projects based on the projects they downloaded from the Scratch website, which has brought a significant increase in project submissions (Monroy-Hernández, 2009). Thus, we can say Web 2.0 has brought two new features, web collaboration and social creativity, to the end-user programming in computer science education.

It is still difficult to measure the educational benefit of end-user programming that end-users can get from the Scratch website. There are some educational benefits from the collaborative learning by sharing students' projects and remixing them (Monroy-Hernández, 2009; Davis et al., 2013), but still there is a need to quantify educational benefits from the projects themselves.

As I stated above, we have invested enough efforts to increase students' motivation in programming education, but we do not have any tool or infrastructure to support visual programming learning that would show the tangible educational benefits to

the end-users. The tangible educational benefits could be articulated with the items below.

- A semantic assessment tool of the games or simulations that students created.
- A learning progress indicator that can show how the students' knowledge has evolved.
- A tutoring system/ learning aid tool where the students can get feedback/help on site.

Considering those benefits, we might need to think about perspectives on learning environments (Bransford et al., 1999). Bransford et al. describe four elements of learning environments (see Figure 1). These perspectives can be applied to a cyberlearning infrastructure to build up an effective learning space. Unfortunately, the current end-user programming infrastructure or learning environment cannot support those elements fully. For example, there should be two different assessment tools: one for formative assessment and one for summative assessment. However, it is rare to see those tools or even any embedded tool on any end-user programming infrastructure. Currently, the knowledge-centered perspective (Bransford et al., 1999), which aims to bring the occurrence of knowledge transfer (Argotea et al., 2000), can only be partially supported by collaborative learning tools/infrastructures.

Bransford et al. argue that technologies need to work as a scaffold to extend students' learning knowledge and help students acquire more advanced problem solving skills. This statement is rooted in Vygotsky's Zone of Proximal Development (ZPD) which describes the difference between what a learner can do without help and what he

or she can do with help (Vgotsky, 1978). Even though it is important that a cyberlearning infrastructure should be able to foster the knowledge of programming that provides scaffolds for students to explore new ideas, learn things, and expand their current ZPD to a new ZPD, any current technology being used on end-user programming learning infrastructure is not fully ready for this job. Currently, students are expected to learn Computational Thinking skills through programming progressively more complex games. Students would be able to move their ZPD to the next new ZPD more rapidly and effectively if there were learning aid tools that could act as scaffolding on the site.

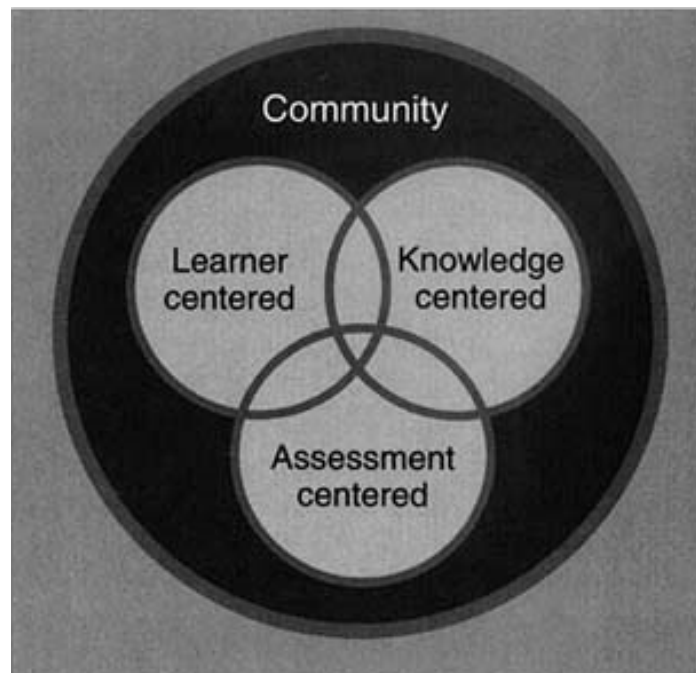


Figure 1: Four Elements of Learning Environments (Bransford et al., 1999)

To address the previously mentioned problems, automated assessment tools and learning aid tools should be included in the current computer science education research with end-user programming. Data visualization (Naps et al., 2002) can support two perspectives, Knowledge-Centered and Assessment-Centered learning environments

(Bransford et al., 1999), by showing the knowledge and skills the students have gained and how successfully they have accomplished their learning goals. Learning aid tools could work as scaffolds to let the students move up to the next learning level with moderate help, and learning aid tools with diagnostic teaching could support the Learner-Centered aspect of the learning environment.

1.1. Research Objectives and Research Questions

Based on my field observation, I believe that there are several challenges to fulfilling these requirements and to supporting teachers' needs. I have devised my research objectives based on these challenges. My research objectives are to create the following:

- **A self-assessment of a desired learning goal** – With the current technology, it is difficult for the students to assess how successfully they have accomplished the target-learning objective inside and outside of class. For example, how can students tell what they have learned and how successfully they have achieved it when he/she has finished making a Frogger game? If the students fail to finish a complete version of Frogger, then how can they figure out what they have missed to complete the game? Using Web 2.0 technology (e.g. Scratch) allows the students to collaborate, inspire each other, and get some educational benefits from collaborative learning, but there is also a low chance of comparing each other's artifacts in the current educational setting.
- **A skill progress indicator** – Within the current K-12 curricula it is not easy to lead a class that accommodates individual students' skill progression.

Generally, K-12 classes are designed based on the average students in the class. Consequently, the advanced students may be bored and the non-advanced students frustrated. To solve this issue, we need to provide a self-guiding tool to the students so that the students can decide whether they are ready to go on to their next learning goal or if they need to stay to complete the current learning objective. This is related to a self-assessment tool. The students need a tool that can indicate how successfully they fulfilled the requirement of the learning object by comparing their artifacts and the tutorial artifact. The students need to know how they have progressed in their programming knowledge over time by creating a game or a simulation. This learning-progress indicator can work with the two above tools to give the next learning objective to the students.

In this dissertation, I propose a method, Computational Thinking Pattern Analysis (CTPA), to analyze, compute and interpret a visual language-programmed artifact in computational thinking. The CTPA analyzes and visualizes the semantic meaning and computational thinking patterns of the submitted games in a cyberlearning infrastructure. The Latent Semantic Analysis technique, as applied to CTPA, analyzes the implemented computational thinking patterns (CTP) in a given game. CTPA compares a specific game/simulation with nine pre-defined canonical computational thinking patterns using an LSA inspired technique: user control, generation, absorption, collision, transportation, push, pull, diffusion, and hill climbing. Those nine patterns are the most common and popular patterns for building video games and science simulations. To perform CTPA, a given AgentSheets project should be converted and expressed as a vector. The interpreted

AgentSheets project vectors are computed with high dimensional cosine calculations to show the AgentSheets project's semantic meanings.

As I stated in the previous paragraph, my research objective is proposing a method to assess students' programmed artifacts and visualize their learning in computational thinking. To determine whether I have achieved the research objective of my dissertation, I devised two research questions as below:

- Research Question 2: Is CTPA effective as a skill progress indicator to predict students' future learning performance?
- Research Question 1: Is CTPA effective in interpreting the computational thinking knowledge a student has learned through making a game/simulation?

Those research questions are answered in Chapter 6, Research Validation.

1.2. Dissertation Outlines

Chapter 2 introduces the conceptual framework of this dissertation, including material previously published in (Basawapatna et al., 2013). Chapter 3 presents research literature in the areas of computational thinking, end-user programming, and assessment, including material reproduced from (Koh et al., 2010; Ioannidou et al., 2011; Basawapatna et al., 2011). Chapter 4 illustrates the Scalable Game Design Arcade and the Computational Thinking Pattern Analysis, which are the main research products from my dissertation research, including material reproduced from (Koh et al., 2010; Basawapatna et al., 2010; Ioannidou et al., 2011). Chapter 5 evaluates the result of Computational Thinking Pattern Analysis validity evaluation, including material adapted from (Koh et

al., 2014). Chapter 6 presents the capabilities of Computational Thinking Pattern Analysis, including material reproduced from (Bennett et al., 2013; Koh et al., 2014). Chapter 7 discusses the findings from the validity studies that evaluated Computational Thinking Pattern Analysis as an automated assessment tool for computer science education. Lastly, Chapter 8 concludes the dissertation by presenting a summary of the current Computational Thinking Pattern Analysis's research contribution and possible future research.

Chapter 2. Conceptual Framework

There are several methods and instruments that are used to assess student motivation in computer science education. Class observations, student interviews, and surveys are good examples of motivation assessment instruments. However, it is hard to find any tool or infrastructure to support Computer Science (CS) and Computational Thinking (CT) (Wing, 2009) education that would assess students' learning of skills and students' learning of objectives at the semantic level. Many existing attempts have been limited in nature to skill investigations mostly at the syntactic level (Les et al., 2008; Lewis 2010) or the functional level. At the syntactic level, student skills are assessed by the number of code lines, the number of certain functions, or the number of warnings and errors (Lewis, 2010). At the functional level, student skills can be measured using a functionality checklist. These existing attempts are extremely difficult and time-consuming because visual and textual languages may not match up very well. Also, usually, students do not get any individual feedback to know what kinds of knowledge they have actually learned through making visual programmed artifacts with most visual end-user programming tools.

A student learning assessment tool at the semantic level can provide tangible educational benefits to students and teachers. This assessment tool would consist of

- A semantic meaning illustrator that interprets what kind of knowledge a student has learned through making a game/simulation
- A learning progress indicator that can show how the student's knowledge has evolved

- A tutoring system/ learning aid tool that can give students feedback/help on site.

The student's learning of skills and the student's encounter of challenge levels are measured by a student's Zones of Proximal Flow (ZPF) as illustrated in Figure 2. The Zones of Proximal Flow (Basawapatna et al., 2013) is a combined theory of Csikszentmihályi's Flow (Csikszentmihalyi, 1990) and Vygotsky's Zone of Proximal Development (ZPD) conceptualization (Vygotsky, 1978). Flow was proposed by Mihaly Csikszentmihalyi (Csikszentmihalyi, 1990), which means a completely motivated and engaged state by measuring student skills and challenges.

To support the theory of the Zones of Proximal Flow in Figure 2, a Computational Thinking (CT) learning course module, which was designed by the Scalable Game Design research team at University of Colorado at Boulder, is structured along a difficulty-oriented organizational pipeline. In other words, the second game in the curriculum is more difficult than the first, and the third game increases in difficulty from the second. The course difficulty continuum also deliberately builds on the acquired knowledge and skills from previously learned games. In the theory of the Zones of Proximal Flow, the students would be able to expand their Flow zone to the next Flow zone via their Zone of Proximal Development with scaffolding learning aid tools on site. This learning aid tool should be able to evaluate and quantify student learning outcomes and challenges by using formative and summative approaches (Bennett et al., 2011).

This Zones of Proximal Flow framework could provide balanced CT curricula for teachers to develop students' CT learning by moving along predictable trajectories

toward more advanced game design or STEM simulation building challenges. This CT learning requires the right assessment tool to support its efficacy.

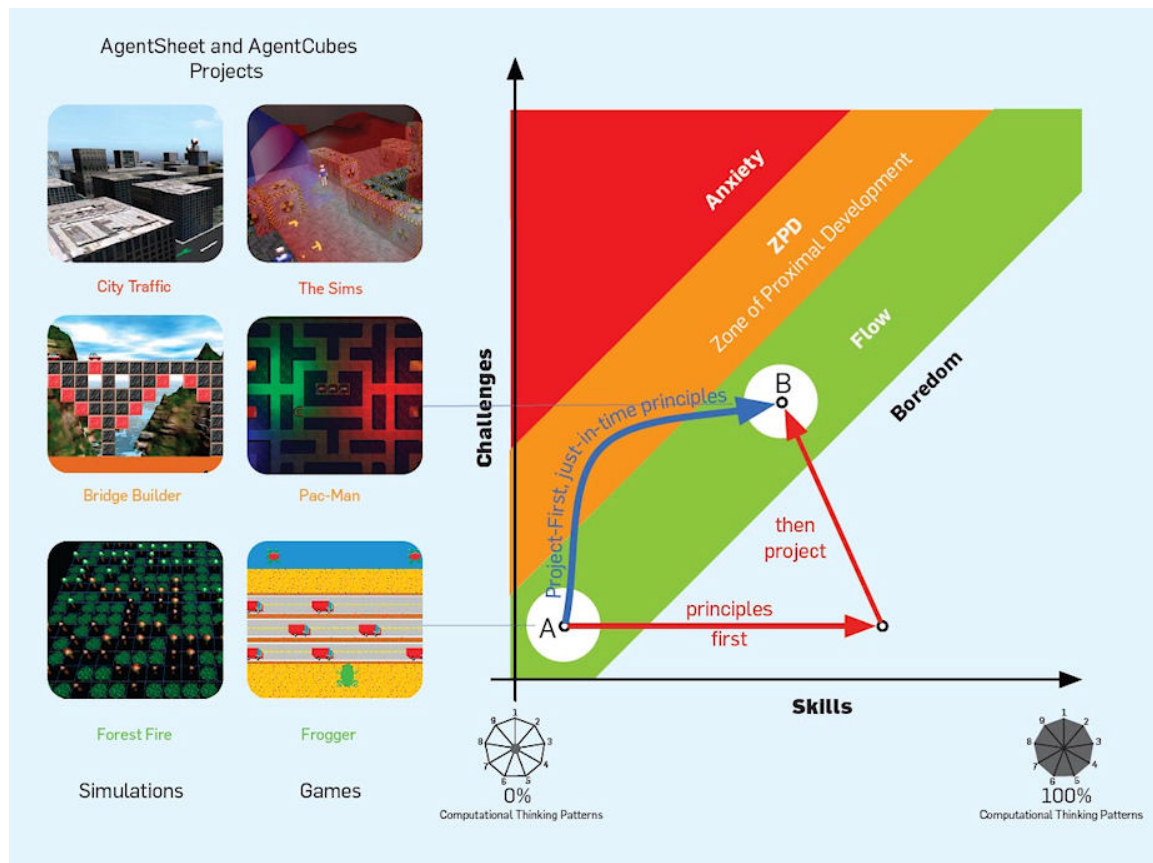


Figure 2. Zones of Proximal Flow: CT curriculum that balances CT challenges with CT skills (Basawapatna et al., 2013).

To measure the efficacy of this approach, student learning is evaluated by employing a tool called Computational Thinking Pattern Analysis (CTPA) (Koh et al., 2010). Every game and simulation produced by students—more than 10,000 over the last 4 years— is not only collected in the Scalable Game Design Arcade (Koh et al., 2010; Koh et al., 2013), but is also analyzed with respect to CT thinking skills expressed by students. CTPA is not looking for constructs such as IF and LOOP statements at the programming level but, instead, is looking for more general object interactions such as

collisions and diffusion at a phenomenistic (Michotte, 1963) level through the use of Latent Semantic Analysis (Landauer, 2003) inspired methods to find code patterns. These phenomenistic patterns, Computational Thinking Patterns (Ioannidou et al., 2011; Basawapatna et al., 2011), can be understood as a subset of universal CT skills that are relevant to game design as well as to other applications including the creation of science simulations.

Chapter 3. Related Research

3.1. Computational Thinking

Computational Thinking (Wing, 2009) is not a clearly defined term, but there have been many researchers trying to conceptualize it or at least utilize it. Following Jeannette Wing, who popularized this idea in computer science education research, Computational Thinking is “a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science” (Wing, 2009). She made an example of a DNA code sequence to show how computer science techniques could be applied to other disciplines (Wing, 2009). However, this is only one interpretation of Computational Thinking.

The National Research Council report on computational thinking outlines five different computational thinking categories (National Research Council, 2010):

1. A Range Of Concepts, Applications, Tools, And Skill Sets

In this category, computational thinking is defined as a way to solve problems, design systems, and understand human behaviors. Also, computational thinking can expand knowledge and be applied to multiple knowledge domains by analyzing the current problems.

2. Language and the Importance of Programming

In the second category, people believe that computational thinking becomes a fundamental intellectual skill just as reading, writing, speaking, and arithmetic. Just as we use our human language, English, to express our thought, programming is the language to

express computational thinking. In other words, as we write a novel or poem with English, programming is writing, and computational thinking is literacy.

3. The Automation of Abstractions

In the third category, computational thinking is close to physics and mathematics, but automation makes computational thinking unique. Like physics and mathematics, computational thinking uses abstraction to represent and explain complexity, but this abstraction and explanation come with automation for computational thinking.

4. A Cognitive Tool

In the fourth category, it is believed that computational thinking does not require formal education to acquire; many people are learning on their own and learning from each other. Computational thinking in this category is considered as a tool for problem solving.

5. Contexts Without Programming a Computer

In the last category, some researchers argue that computational thinking does not need to be associated with computers or programming to be learned. Moreover, they say that computational thinking can be found in many non-IT contexts. Journalism classes could be a good example. Students learn how to write a good article with the circular process: write, edit, send, and re-write it. If you can notice that this process looks similar to a software development model, then it would not be hard to find computational thinking there.

Two of those categories conflict with each other: Computational Literacy and Context Without Programming. The first category mainly focuses on programming. It

says that programming is a fundamental intellectual skill just like reading, speaking and writing. However, the second category argues that computational thinking doesn't need to be associated with programming, and computational thinking can be and should be taught without using programming or a computer.

These two disparate categories present a conflict, but we can derive some common themes from all five categories.

- Computational thinking is not about using specific programs or programming languages.
- Computational thinking is not computer science, but part of it.
- Computational thinking was the outcome of a natural evolution in our understanding of computer science.

In addition, the fundamental arguments of all five ideas seem to suggest that computational thinking is a way to express a perception of certain domains or knowledge. This might be connected to a certain idea, which is called phenomenism (Michotte, 1963).

So I believe that the common idea from the five different categories in the NRC report is "Representation", which can be applied to any given domain. For example,

- Representation of the given situation => Computational Literacy
- Representation of the problem => Abstract
- Representation of the answer => Problem Solving

The below diagram depicts how different concepts of computational thinking in the NRC report overlap each other or share each other's concepts.

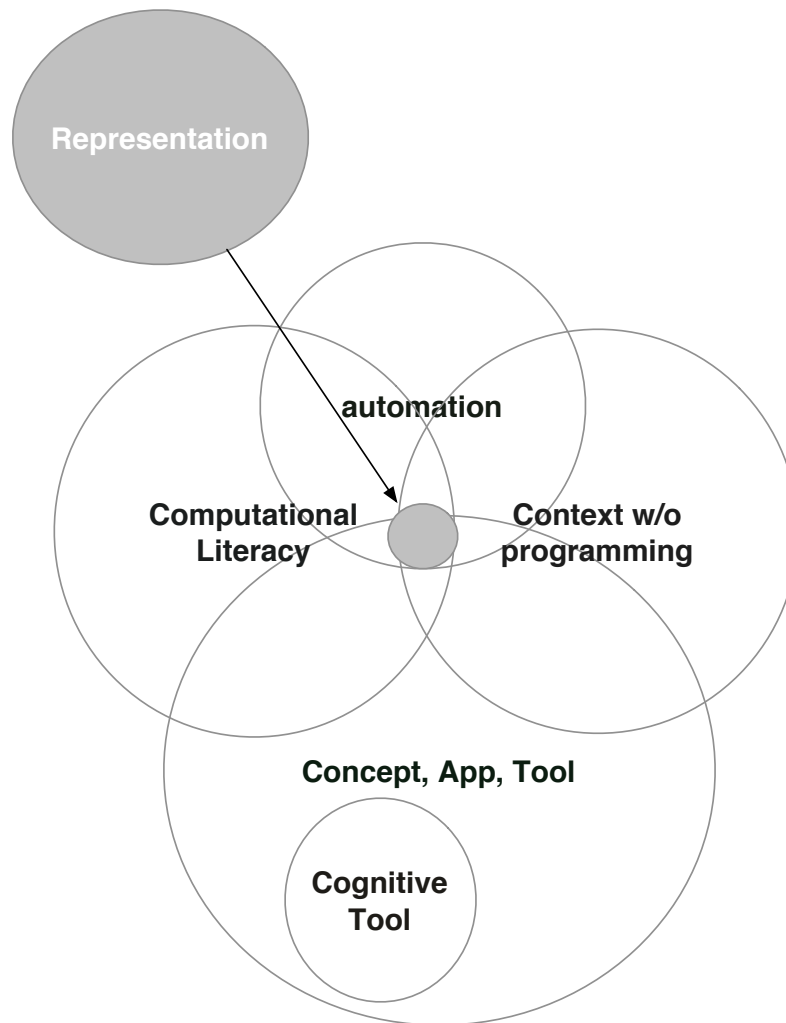


Figure 3. Relationships between the five Computational Thinking categories in the NRC report

3.2. Computational Thinking Patterns

Computational Thinking (CT) is a buzzword in computer science education nowadays, but its description, function and objective in computer programming for educational purposes are still vague. The concept of Computational Thinking Patterns (CTP) was developed as a subset of CT. Computational thinking patterns or agent interactions are commonly observed in other programming contexts and other disciplines (i.e. science and mathematics) (Ioannidou et al., 2011; Basawapatna et al., 2011). For example, the Generation CTP represents one agent creating another agent (e.g., wolf

generating offspring in an ecosystem). In this way, each CTP represents one complete phenomenon or behavioral concept in a game or science simulation design.

Examples/Descriptions of common computational thinking patterns:

- Absorption: An agent programmed to absorb erases a given agent when the given agent moves into a given range. Usually this is done in conjunction with generation to simulate an ongoing action like traffic movement, or a turtle swimming up stream.
- Collision: When a collision occurs in a game or simulation, it represents a situation where two agents physically run into each other. One of the agents is programmed to show the collision visually, perhaps as an explosion. Then one (or both) of the agents erases itself. A good example in Space Invaders is when a missile collides with an alien ship; both agents explode and erase themselves as a result of the collision.
- Generation: An agent programmed to generate must create an ongoing flow of another agent when a specific area near the original agent is empty. A good example of this is the creation of a flow of cars emerging from a tunnel. The tunnel would be the generating agent.
- Transportation: the Transportation pattern is unique and crucial for creating games. This pattern represents the situation that one agent carries another agent as it moves. For example, a dropship in StarCraft can carry marines, or a turtle in Frogger can carry a frog on it. This pattern happens when one agent is stacked over another agent and the stacked agent is carried with the bottom agent as the bottom moves.

- **Push:** the Push pattern is the pattern we see in the game of Sokoban. A player in Sokoban is supposed to push boxes to cover targets. As the player pushes the box in Sokoban, the box moves in the direction (up, down, right or left) it is pushed. The box in Sokoban should move only if it is pushed toward a regular ground tile or a target spot. Therefore, if an agent that is supposed to be moved is blocked by other agents in the direction of the push then neither the agent being pushed nor the pusher agent should move.
- **Pull:** This pattern is the opposite pattern of push. An agent can pull another adjunct agent or any number of agents serially connected to the puller. For example, you can imagine that a locomotive pulls a large number of railroad cars. In this situation one agent, the pulling agent, can pull one or more agents that are connected to itself.
- **Diffusion:** You can diffuse a certain value of an agent through neighboring agents with a diffusion pattern. For example, a torch agent can diffuse the value of heat through neighboring floor tile agents. The closest eight neighboring floor tile agents to the torch agent will have the highest value of heat, and tile agents that are further away from the torch agent will have a lower heat value.
- **Hill Climbing:** Hill climbing is a searching algorithm in computer science. A hill-climbing agent will look at neighboring values and move toward the one with the largest value. Hill climbing can be found in the game of Sims or Pacman. In the game of Pacman, Ghosts chase Pacman by following the

highest value of Pacman's scent that is diffused throughout the level. This is depicted in Figure 4. As with the torch above, the floor tiles around where Pacman is currently has the greatest scent value. Therefore, Ghosts hunt Pacman using the hill climbing technique by following the greatest value of Pacman's diffused scent.



Figure 4. Ghosts use hill climbing on Pacman's diffused scent (pictured around Pacman) to track down Pacman

Games	Computational Thinking Pattern
Frogger	Cursor Control, Generation, Absorption, Collision, Transportation
Sokoban	Cursor Control, Push, Pull
Centipede	Cursor Control, Generation, Absorption, Push, Pull
Space Invaders	Cursor Control, Generation, Absorption, Collision, Choreography
Sims	Diffusion, Hill Climbing

Table 1: Computational Thinking Patterns in Curriculum Games

3.3. Social Creativity through End-User Programming

Several end-user programming tools have attempted to foster creative learning through programming games and STEM simulations since the early 1990s (Basawapatna et al., 2014; Koh et al., 2014). All end-user programming tools have shown the current possibility and the future potential of end-user programming software as an alternative way to teach computer science and programming concepts to K-12 students. The end-user programming tools have shared a common approach: creating simulations and games to teach and learn programming and computer science concepts, including problem-solving skills. This approach has been successful. As new abilities to collaborate through social interactions are blended into software engineering, it is important to integrate those abilities into computer science education in which end-user programming is used.

At this point, visual programming tools including AgentSheets (Repenning et al., 1997) and Scratch (Resnick et al., 2009) could be a stepping-stone for bringing end-user programming to social networks. Research papers on Scratch and AgentSheets have shown how successful those tools have been in promoting web sharing, teaching collaboration skills and fostering creativity in young children (Maloney et al., 2008; Koh et al., 2010). Particularly, more than 20,000 students in public schools have registered and used AgentSheets and AgentCubes with the Scalable Game Design Arcade (Koh et al., 2010) to create educational and non-educational simulations and games since the Scalable Game Design Arcade release in 2010. From 2010 to 2013, the Scalable Game Design Arcade collected more than 30,000 AgentSheets and AgentCubes projects.

Thanks to the ability to share AgentSheets and AgentCubes projects on the web, AgentSheets and AgentCubes users are able to inspire other people and be inspired by them (Koh et al., 2010). In other words, AgentSheets and AgentCubes users are able to learn programming concepts or algorithms and take actual codes from other people's work to make their own games. For AgentSheets programming, this activity is called "Behavior Exchange" (Repenning et al., 2007), and Scratch researchers define this ability as "Remixing" (Monroy-Hernández, 2009).

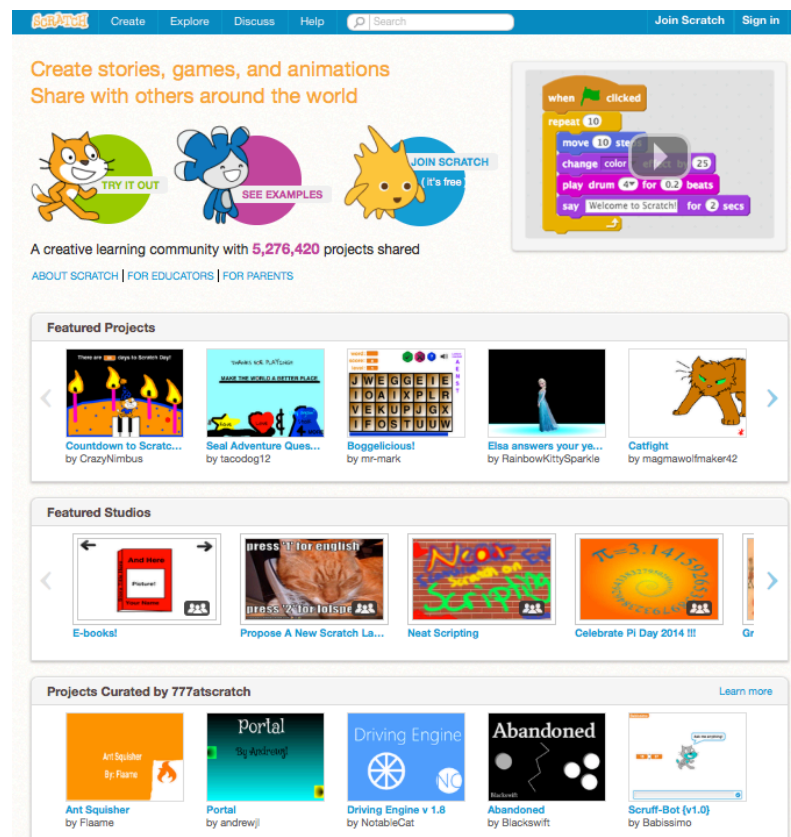


Figure 5 Scratch Website

Remixing is one of the key factors that makes the Scratch website active and special. Around 30% of all Scratch projects are remixed projects, which means one third of Scratch users did not need to build their projects from scratch (Monroy-Hernández,

2009). They simply needed to add something to an existing project to create their projects.

One of the other key factors that keeps the Scratch website active is a section called “Galleries” for Scratch version 1.4 or “Studios” for Scratch version 2.0 in which users are able to take full advantage of the nature of remixing. Galleries and Studios are the place where users can see on-line collaboration happen regardless of spatial, cultural and age differences. Users make clusters of projects and connect them to create a single game or simulation. Users who do not know each other collaborate, share feedback and communicate with peers to accomplish mutual goals. However, this activity is not always successful in producing a meaningful artifact; because of the character of online collaboration, some projects cannot be finished and are abandoned. Nevertheless, this gallery shows that online collaboration can overcome the obstacles of time and spatial limitation that we experience in off-line collaboration.

Just as Remixing is one of the key factors for Scratch, Behavior Exchange has had a substantial role in increasing/fostering creativity within the implementation structure of the SGD project (Basawapatna et al., 2009; Koh et al., 2010).

The Scalable Game Design Arcade does not have a function corresponding to the Galleries or Studios in Scratch, and it is certainly not unique in its function as a creative resource. However, the integral part it plays within the SGD project structure, coupled with the flexibility of AgentSheets programming, certainly provides the students with a unique experience that appears to foster creativity (Bennett et al., 2011).

In addition, CyberMod, a massive online collaboration tool, was released in 2012 for AgentSheets and AgentCubes programmers (Repenning et al., 2013). CyberMod will

be integrated into the next generation of the Scalable Game Design Arcade. CyberMod is a novel and effective method to replace current end-user programming systems. Every programming activity in CyberMod is a synchronized action. Similar to the Google Drive products, anyone can edit and save AgentCubes projects while other people are working on the same project. The nature of CyberMod would break the constraints of current online collaboration systems and bring synchronous web collaboration. Scratch does not have this functionality yet. Scratch users create one artifact through collaboration, but whole transactions cannot be done simultaneously.

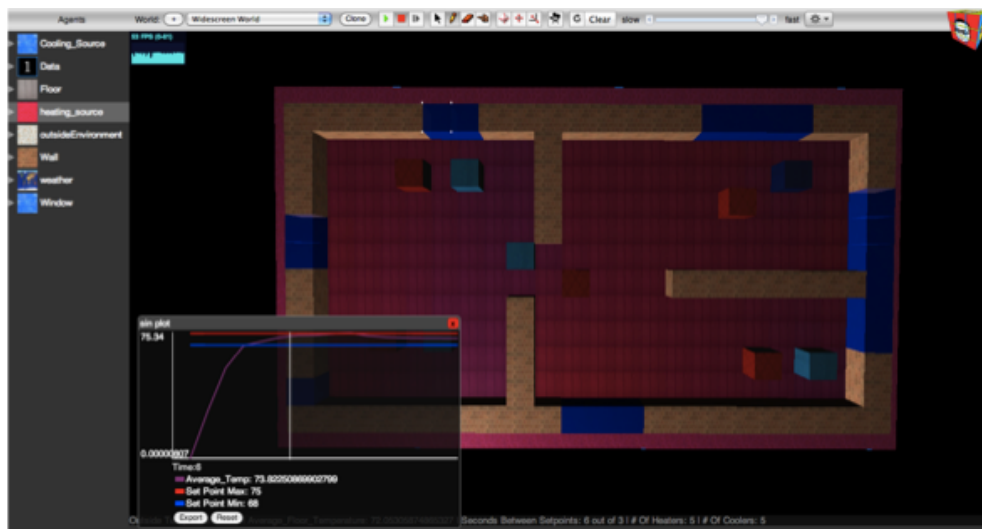


Figure 6 An example project of CyberMod

3.4. Assessment in Visual End-User Programming

Related research, such as Lewis (Lewis, 2010), compares two visual language programs (Scratch and Logo) for student game authoring. For this example, much of this research protocol centers on motivational questions to determine if there was any difference between each of the two software programs. Only knowledge of individual programming pieces, apart from the larger context of the program, were used to compare

the two programs. Studies such as Lewis's research that evaluate on a purely syntactic level might show knowledge of individual concepts but do not evaluate the student's ability to use that knowledge within multiple contexts. Syntactic evaluations are not useful for detecting high-level computational thinking knowledge or learning transfer. A semantic evaluation tool could be very useful for more accurately indicating the transfer of learned knowledge in this respect.

One way to compare and profile code on more of a semantic level, as an alternative to just counting program primitives such as loops, is to look for higher level patterns that could be indicative of the meaning of a program. A similar approach called Latent Semantic Analysis (LSA) is used to find semantic information in natural languages by comparing text (Landauer, 2003). Computer languages, including visual languages, can be subjected to the same idea. Just like natural languages, computer languages are based on the notion of statements consisting of grammatical structure. On one hand, computer languages should be simpler to deal with as their syntactic rules tend to be less irregular. In the LSA, stemming is a fundamental problem which is not relevant to computer language because verb conjugations are non-existent. Functions and primitives of computer languages are comparatively simple.

Chapter 4. Computational Thinking Pattern

Analysis

For my dissertation research, I've created several research tools including Computational Thinking Pattern Analysis (CTPA), the world's first automated real time programming assessment tool through phenomenology, and Scalable Game Design Arcade (SGDA), a cyberlearning infrastructure where middle school to grad school students share their own game and simulation projects. SGDA has collected more than 10,000 educational game/simulation projects from 7,000 students. CTPA and SGDA are designed to substantiate the theory of the Zones of Proximal Flow wherein the Zone of Proximal Development lies between the regions of Flow and anxiety.

4.1. Scalable Game Design (SGD) Project

The SGD project was funded by the National Science Foundation (NSF) to improve computer science interest in middle schools. The goal of SGD is to use visual language programming software (AgentSheets) to introduce a more positive image of computer science through required game design courses. For this purpose, SGD researchers recruited middle school teachers from several Colorado school districts. After recruitment, participant teachers were taught the AgentSheets software during a 2-week long course at the University of Colorado, Boulder. During the following semesters they taught at least two class sections of the Scalable Game Design curriculum in their own classrooms. The games are taught in a sequence/progression of learning difficulty, with students using their individually acquired knowledge from previous games to build the new games (Koh et al., 2010).

SGD participant middle school students submit individually created games and/or simulations at the end of each class unit. Usually students learn one game in each class module. After uploading, each submission is evaluated and displayed in comparison to the tutorial standard with the Computational Thinking Pattern Analysis (CTPA) graphic tool (Koh et al., 2010).

4.2. Scalable Game Design Arcade (SGDA)

In support of the Scalable Game Design project, I designed and operated a cyberlearning infrastructure, Scalable Game Design Arcade (SGDA). The Scalable Game Design Arcade (SGDA) is an educational online infrastructure that facilitates a more user-friendly homework submission format for the Educational Game Programming class (Basawapatna et al., 2010; Koh et al., 2010; Bennett et al., 2011). Through the SGDA, students can play classmates' games and download game programming. They are also able to directly submit their games to SGDA, rate other students' games, and in share feedback on each other's work. Without any time lag, students can benefit from each other's game ideas before and after the submission deadline. Also, SGDA works as an online repository for collecting games/simulations that are created by middle school and high school students from participating schools.

The SGDA consists of three parts: a main page, an assignment gallery, and an individual page.

- The assignments gallery (Figure 7) shows multiple submitted games/simulations that are submitted by participant school students in one unit class. This assignment gallery contains a table with meta- information

for each game including the game title, screen dump, author name, game summary/playing instructions, and submission time. The screen dump and instructions give the potential player a preview of how the game looks, making game selection easy. The column for the game author's name is a direct link to their individual page.

- The main page (Figure 8) displays recently submitted games, the most downloaded games, and the most played games.
- The individual page (Figure 9) displays a screenshot of a submitted game/simulation, the Computational Thinking Pattern Analysis (CTPA) graph (Koh et al., 2010), links for playing and downloading games/simulations, and a similarity score between a given game/simulation and four tutorial games. Any user who accesses SGDA can play, download, and/or rate a game without a time lag after the submission is made (Koh et al., 2010; Ioannidou et al., 2011).



Name	Title	Description	Rate	Date Added	Date Modified	Views
tyler	Tyler SOKOBAN 	Working Sokoban game with four levels. Fixed so levels work [comments : 0]	Rating: 3.0 /5 (1 vote cast) 	11.01.24 : 21:39:42	11.02.15 : 10:34:23	1135
jiewu1234	Name Your Own Project! 	Sorry, Hyuhan(Dont be mad if I spell it wrong). I apologize for the late submission. I just realized myself submit my game to the CSCI7000 2010 which i ..more [comments : 0]	Rating: 2.5 /5 (2 votes cast) 	11.01.31 : 21:38:46	11.01.31 : 21:38:46	820
jmwbarnett	Sokoban 	Three levels. Sorry for annoying backgrounds. Levels 1 and 3 are original. [comments : 0]	Rating: 2.0 /5 (2 votes cast) 	11.01.25 : 21:20:12	11.01.25 : 21:42:48	1232

Figure 7: The Assignment Gallery of SGDA

SCALABLE GAME DESIGN arcade

SCALABLE GAME DESIGN


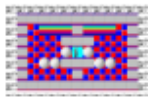

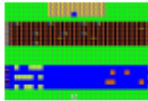
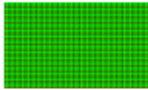
SCALABLE GAME DESIGN wiki

Username: Password:

[\[Forgot Password?\]](#) Not registered? [Sign-Up!](#)

[Home](#) [Class Selection](#) [Open Arcade](#) [Support](#) [About](#)

Recently Submitted Games

				
Space Invaders by mstreet1	iefroggersamplescienc esimulationsampleetc by bv11529	Frogger by eh9975	Frogger by eh9969	thisisaforestfiresimula tion by og10111

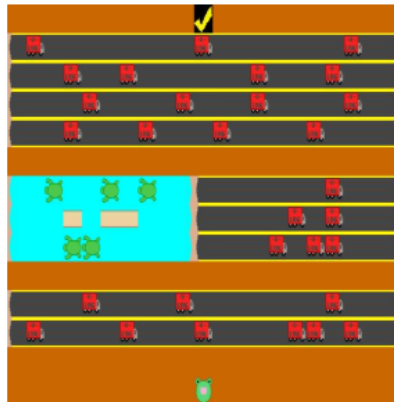
Most Downloaded Games

				
antsnohoming by andri	frogger by theone20001	frogger by theone20001	opensokoban by kyuhan	openarcade15puzzlew ithwindetectionandtim er by fredgs

Most Played Games

				
frogger by oasheim	pacman by d709464	pacman by d709474	centipede by oasheim	sims by cichen

Figure 8: The Main Page of SGDA



Frogger

Probably won't refine this version. This has two levels, with lots of movers and their -U- Lugs trucks, turtles, and logs varying in length. Use the directional arrows on the keyboard to move the frog. - Katherine Pham

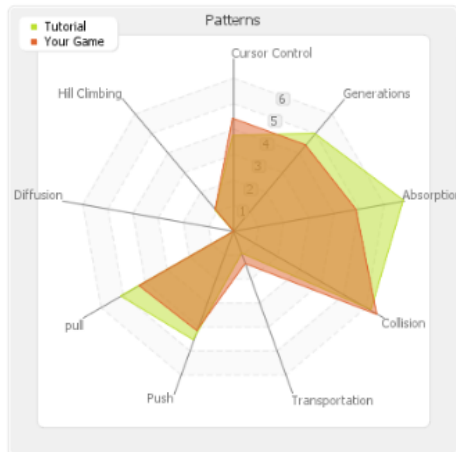
[Run](#)
[Download](#)

Similarity Score to Four Tutorial Games

This score shows how much your game structure is similar to the tutorial games. Max value is 1

- This game's similarity score to Frogger:0.849
- This game's similarity score to Sokoban:0.503
- This game's similarity score to Space Invaders:0.599
- This game's similarity score to Sims:0.132

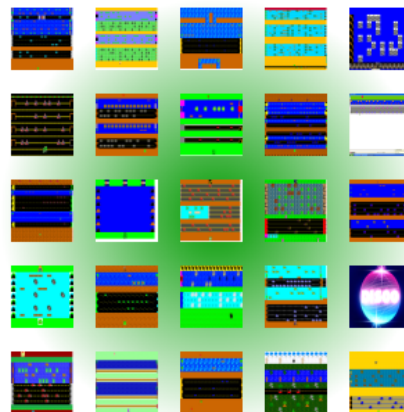
Computational Thinking Patterns



Rating: **2.0/5**
(3 votes cast)
★☆☆☆☆

Similarity Score Matrix

Below Matrix shows other AgentSheets projects sharing similar programming structure.



Submission Time
11/01/16 : 15:54:20

Figure 9: The Individual Page of SGDA. Individual page illustrates the screenshot of the game (upper left), Run and Download button (upper right), the game’s similarity score compared to four tutorial games (middle right), a similarity score matrix showing games programmed similarly to the submitted game, and the CTPA graph (bottom right & left).

4.3. Computational Thinking Pattern Analysis

Computational Thinking Pattern Analysis (CTPA) is designed to evaluate the semantic meaning of the games/simulations submitted to the Scalable Game Design Arcade. The Latent Semantic Analysis (Landauer, 2003) inspired technique as applied to CTPA analyzes the implemented computational thinking patterns (CTP) in a given game. CTPA compares a specific game/simulation with nine pre-defined canonical computational thinking patterns using a LSA inspired technique. The patterns are: user control, generation, absorption, collision, transportation, push, pull, diffusion, and hill climbing. These particular nine patterns are the most common patterns used in the construction of video games and science simulations (Koh et al., 2010; Basawapatna et al., 2011). In the future I could include more CT patterns in CTPA, but to date my research has focused on these nine, as shown in Figure 10.

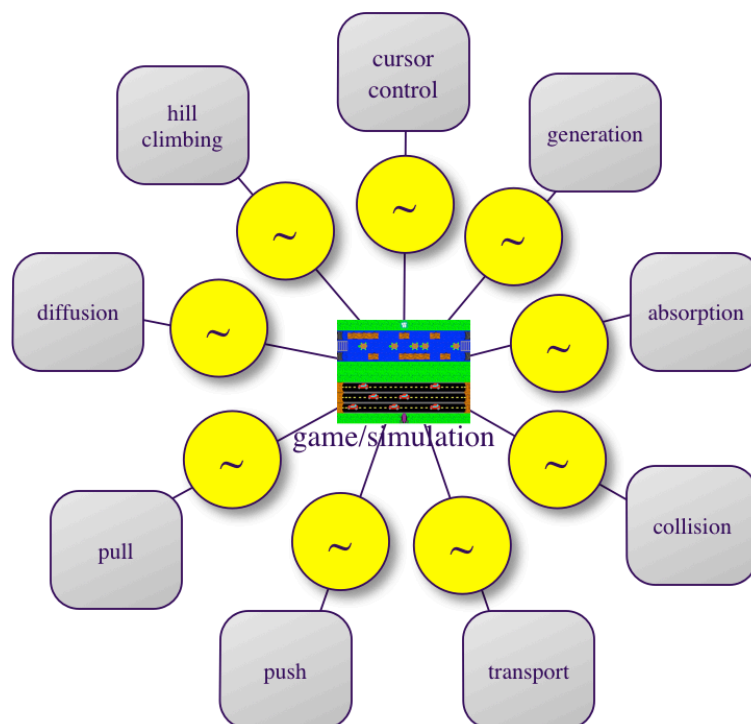


Figure 10: Computational Thinking Pattern Analysis using Multiple High Dimensional Cosine Calculation

Using the LSA-inspired technique denoted as ☺ in Figure 10, a game submitted to the Scalable Game Design Arcade gets compared to nine canonical Computational Thinking Patterns. A CTPA graph showing the similarity values for each pattern is produced, as shown in Figure 11.

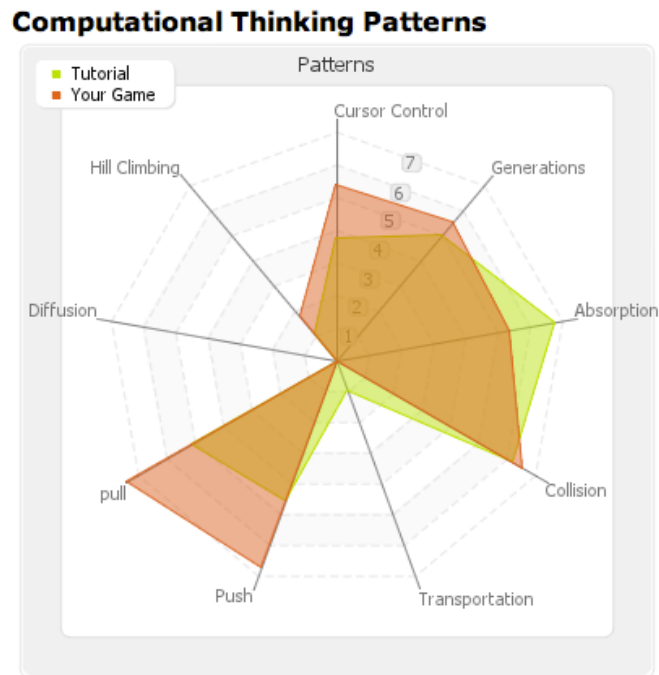


Figure 11: Computational Thinking Pattern Analysis Graph obtained by CTPA

Computational Thinking Patterns are high-level programming concepts, and each pattern requires multiple rules and/or programming primitives to be implemented.

To perform CTPA, a given AgentSheets project is converted and expressed as a vector.

The interpreted AgentSheets project vectors are calculated with the equation below to show vector semantic meaning (Koh et al., 2010; Bennett et al., 2013).

$$\text{CTPA} (m) = \left[\frac{\sum_{i=1}^n u_i v_i}{\sqrt{\sum_{i=1}^n u_i^2} \sqrt{\sum_{i=1}^n v_i^2}} \right]_1^m$$

Equation 1: Computational Thinking Pattern Analysis Equation

In this equation, u and v respectively represent a given game/simulation and one canonical computational thinking pattern. Also, n is the vector size of a game/simulation or CT pattern, and m is the number of computational thinking patterns that are applied to CTPA (currently 9). The calculated result of CTPA through CTPA (1) to CTPA (m) can thus be represented as an m dimension vector.

4.4. Computational Thinking Pattern Analysis Graph

The Computational Thinking Pattern Analysis (CTPA) graph visualizes the semantic meaning and computational thinking patterns of the submitted games within SGDA, which were calculated through CTPA. The computational thinking patterns implemented in each given game are depicted through graphic analysis (Figure 12).

This research implementation used regular class curriculum and was assessed using official game tutorials provided by the Scalable Game Design project researchers and educators. The CTPA Graph automatically overlaps a tutorial graph (brown in Figure 12) with a graph of a submitted game (green in Figure 12) only if an official tutorial of that game was pre-loaded. If the uploaded game is a free submission, which does not have an official and/or unofficial tutorial, then only the submitted game analysis is displayed. Each CT pattern axis is aligned by its implementation difficulty level and its

significance to the relationship between adjacent axes. For example, Generation, Absorption, and Collision usually happen in sequence or are highly relevant to each other.

This graphic analysis can work as a self-assessment tool and/or a learning path indicator through a semantic comparison of the submitted project to that specific submission's tutorial standard. In the absence of a comparative tutorial, standardized information can be programmed into the graphic analysis tool to serve as an appropriate comparison.

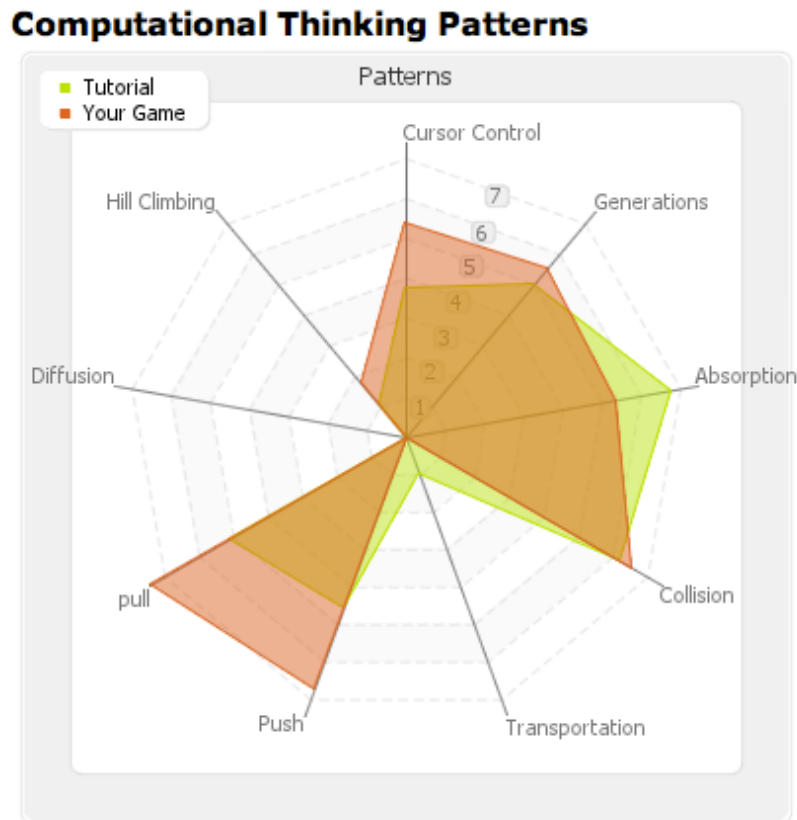


Figure 12: A CTPA Graph from an Example Frogger Game

4.5. The Sensitivity of CTPA

Variances in implementation of computational thinking patterns will result in differences in CTPA. For example, the existence of transport action makes a small or even a big difference for certain CTPA types (See Appendix B). Programming components (conditions and actions) such as See, Move, and Make can affect CTPA if they are constructors of computational thinking patterns such as Generation, Push and Pull. If those programming components are key constructors of computational thinking patterns, then their influences are greater than other non-key constructors' influences. A key constructor means a dominant programming component of a computational thinking pattern such as Make action for Pull or Push pattern.

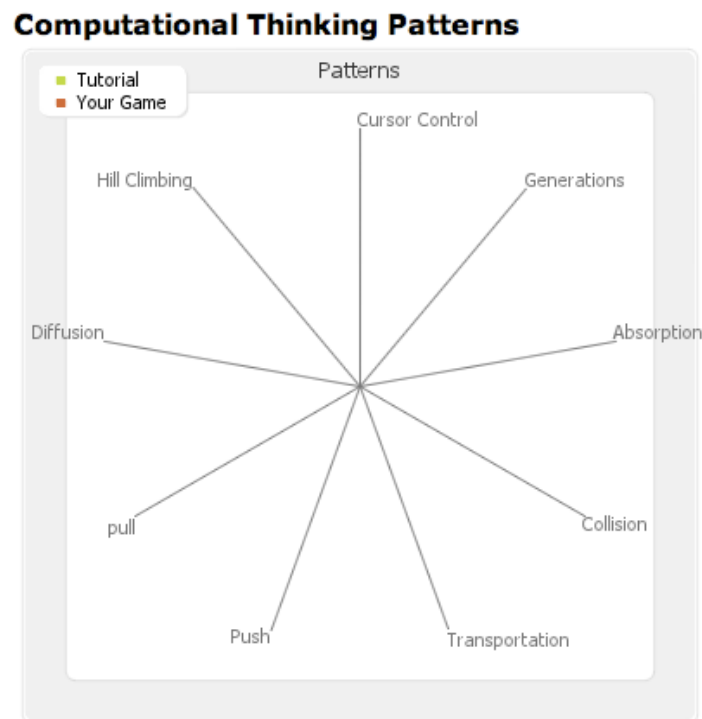


Figure 13 CTPA results of an AgentSheets Project programmed with non-computational thinking pattern constructors only. It results in a value of 0 for all CT patterns.

On the other hand, programming components that are not constructors of a pre-defined computational thinking pattern do not cause any change in CTPA. For example, if one makes a project with Hear condition and Text action only, (neither of which is a constructor of any computational thinking pattern), then the CTPA results in a value of 0 for all computational thinking patterns (Figure 13). In other words, the CTPA is not changed/affected by a small or a big change in non-computational thinking pattern constructors.

4.5.1. CTPA similarity vs. Code similarity

Usually, CTPA similarity and programming similarity are highly correlated. However, even if two projects appear similar from a CTPA point of view, those two projects might be dissimilar from a programming point of view. For instance, if one made a Frogger project and named it Frogger A, added non-computational thinking pattern constructors to Frogger A and then saved it under the name Frogger B, Frogger A and Frogger B are identical from a CTPA point of view since the CTPA is not affected by any non-computational thinking pattern constructors for its analysis. However, they are different from a programming point of view.

Figure 14 illustrates all Frogger games from Aspen Creek K-8 School between 2011 and 2012. One data point indicates one student-made Frogger game. Figure 14 displays the CTPA divergence distribution (Koh et al., 2011; Bennett et al., 2013), and lower values indicate higher CTPA similarity to the official Frogger tutorial. All data in Figure 14 were obtained with the CTPA Divergence equation in Equation 2 (Koh et al., 2011; Bennett et al., 2013).

4.5.2. Programming Divergence Calculation

Figure 12 depicts a student-submitted game in comparison to the standard SGD online tutorial (Bennett et al., 2013). In Figure 12, the difference or space between the tutorial (green) and the submitted artifact (brown) visually displays the divergence of a student's Frogger game from the tutorial "norm." Since each axis on the CTPA graph represents one element in a vector, the CTPA graph represents a nine-element vector, where each element represents a CT programming pattern that could be chosen by the student as part of his/her programming solution. Thus the divergence of a student-created artifact from the tutorial "norm" can be calculated as the difference between two nine-element vectors, one from the tutorial and the other from the submitted artifact. The Divergence Score is calculated from the difference of the two nine-element vectors. The equation of the Divergence Calculation is depicted below.

$$Divergence(x) = \frac{\sqrt{\sum_{i=1}^n (u_i - v_i)^2}}{\sqrt{n}}$$

Equation 2: Divergence in Programming Equation that calculates the differences from the norm (tutorial)

In this equation, u and v represent a tutorial and a given game respectively, and n represents the number of computational thinking patterns. Presently, nine computational thinking patterns are used in the Scalable Game Design project curriculum.

For example in Figure 12, the student-submitted game and the tutorial can be represented as nine dimensional vectors, respectively: (0.525, 0.557, 0.432, 0.641, 0, 0.687, 0.721, 0, 0.197) and (0.373, 0.499, 0.679, 0.623, 0.096, 0.455, 0.51, 0, 0.106). The difference of those two vectors is (0.152, 0.058, -0.247, 0.018, -0.096, 0.232, 0.211, 0, 0.091). The normalized (divided by the value of rooted n) length value of that vector is 0.15, and this is the value of the divergence score of the given game.

When a student-submitted game is exactly the same as the tutorial, there is no difference between those graphs. For example, the submitted game and the tutorial can be represented as following vectors, (0.373, 0.499, 0.679, 0.623, 0.096, 0.455, 0.51, 0, 0.106) and (0.373, 0.499, 0.679, 0.623, 0.096, 0.455, 0.51, 0, 0.106). The difference of those two vectors, of course, is (0, 0, 0, 0, 0, 0, 0, 0, 0). The value of 0 represents no difference between a given game and the tutorial, so the game's programming is functionally identical.

Therefore, it is possible to compare CTPA similarity and programming similarity with examples. I chose four Frogger games and colored them in the graph to explain the relationships between CTPA divergence and programming difference (Table 2). Table 2 shows the four games' CTPA divergences and programming differences to the tutorial (lower value means higher similarity to the tutorial).

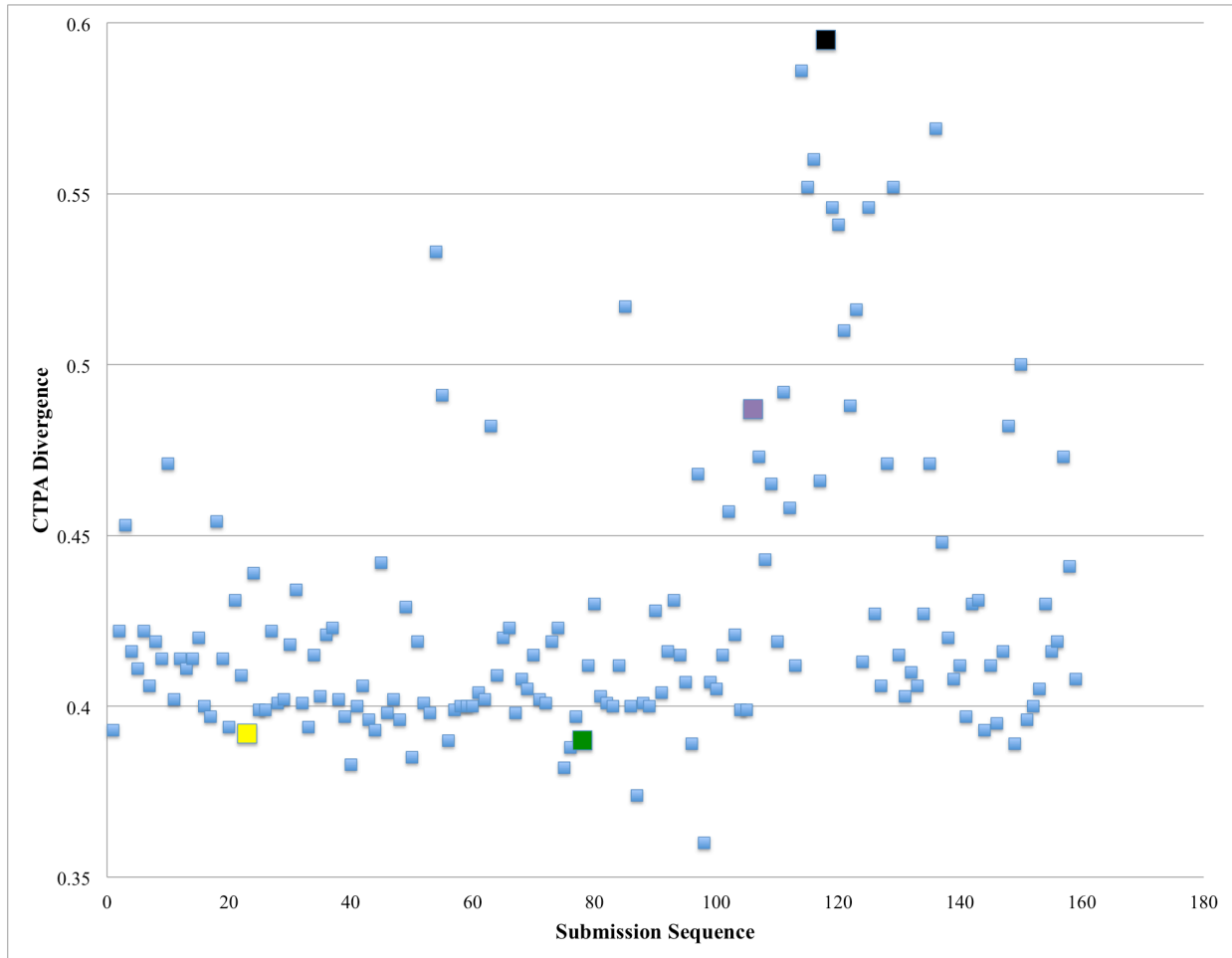


Figure 14: CTPA Divergence in terms of Computational Thinking Pattern. X-axis represents the submission sequence based on time. Y-axis represents the CTPA divergence value.

	Color in the Graph	Number of Agents	Number of Conditions	Number of Actions	Divergence from the tutorial
Game 1	Yellow	12	30	25	0.392
Game 2	Green	12	42	44	0.39
Game 3	Purple	10	16	36	0.487
Game 4	Black	22	37	104	0.595

Table 2 Four Games' CTPA and Programming Information

As Table 2 depicts, Game 1 and Game 2 are more similar to the tutorial, and Game 3 and Game 4 are more different from the tutorial. Game 1 and 2 are programmed

in a similar manner, but Game 2 has more programming components. For example, the Frog agent in Game 1 is programmed as illustrated in Figure 15 (5 rules), and the Frog agent in Game 2 has the exact same programming components as Game 1 but adds more programming components as shown in Figure 16 (7 additional rules). More precisely Game 1 fails to implement the Absorption pattern for drowning and Game 2 has more cursor control options. This variance results in a small difference in CTP divergence and programming difference.



Frog

Number of methods: 1

- [WHILE-RUNNING \(\)](#)

^WHILE-RUNNING ()

Put text here to explain what this method does!

Number of rules: 5

```

If
  KEY (126)
Then
  MOVE (↑)

If
  KEY (125)
Then
  MOVE (↓)

If
  KEY (124)
Then
  MOVE (→)

If
  KEY (123)
Then
  MOVE (←)

If
  SEE (← , ■)
Then
  PLAY-SOUND (explode), and
  CHANGE (■ , ●), and
  WAIT (3.0), and
  ERASE (■), and
  RESET-SIMULATION ()

```

Figure 15 Frog Agent in Game 1

```

If
  STACKED (immediately above , 🟦)
Then
  PLAY-SOUND (kaboom), and
  CHANGE (🟦 , 🟡), and
  WAIT (0.5), and
  SHOW_MAKE (you drowend), and
  ERASE (🟦), and
  RESET-SIMULATION ()

If
  STACKED (immediately above , 🟩)
Then
  PLAY-SOUND (kaboom), and
  CHANGE (🟦 , 🟡), and
  WAIT (0.5), and
  SHOW_MAKE (you die), and
  ERASE (🟦), and
  RESET-SIMULATION ()

If
  STACKED (immediately above , 🟨)
Then
  PLAY-SOUND (hahaehi), and
  CHANGE (🟦 , 🟡), and
  WAIT (0.5), and
  ERASE (🟦), and
  SWITCH-TO-WORKSHEET (nwxt level)

If
  KEY (14)
Then
  MOVE (🟦)

If
  KEY (1)
Then
  MOVE (🟦)

If
  KEY (0)
Then
  MOVE (🟦)

If
  KEY (12)
Then
  MOVE (🟦)

```

Figure 16 Additional programming of Frog Agent in Game 2

There are some differences between Game 1 and 2, but they are programmed in a similar way. Figure 17 illustrates comparisons of CTPA graphs for Game 1 and Game 2.

I have argued that if there is a big divergence between a given game and a tutorial, the given game is either incomplete or more advanced than the tutorial. Games 3 and 4 are examples of both situations. Game 3 has missing agents (Log and Turtle agents or equivalent). The CTPA graphs of Game 1 and Game 3 are compared in Figure 18. On the other hand, Game 4 has more agents than the other 3 games and extra features such as a cheating buster (Frogger cannot bypass the river). The CTPA graphs of Game 1 and Game 4 are compared in Figure 19.

Even though they are programmed differently, Game 3 and Game 4 look similar from the CTPA point of view, in Figure 20. The value of the CTPA divergence between them is 0.18, and the value of the programming difference between them is 0.14. In the CTPA divergence calculation, games with missing features and games with extra features seem to be assessed similarly. However, there is a noticeable CTPA graph difference between incomplete games and advanced games (Bennett et al., 2013). An advanced game has a tutorial-like graph shape in a smaller size (Figure 19), but an incomplete game has the same size graph as the tutorial graph but in a different shape (Figure 18). This phenomenon occurs because the score for each vector of the CTPA graph represents how much a certain computational thinking pattern is employed in a given game. So, if a game has features, which are not in the CTPA graph structure, the CTPA graph will not analyze those features. Therefore, the remaining computational thinking patterns would be a smaller portion of the CTPA graph. Consequently, undetected features will lower the

CTPA process score of those computational thinking patterns. As a result, the CTPA graph for that game will be smaller than a game that employs only the computational thinking patterns within the CTPA graph structure.

In conclusion, the CTPA is affected by programming difference only if the difference comes from computational thinking pattern constructors. Programming difference in non-computational thinking pattern constructors cannot affect CTPA results. High similarity in CTPA divergence cannot guarantee high similarity in programming, but generally they are highly correlated.

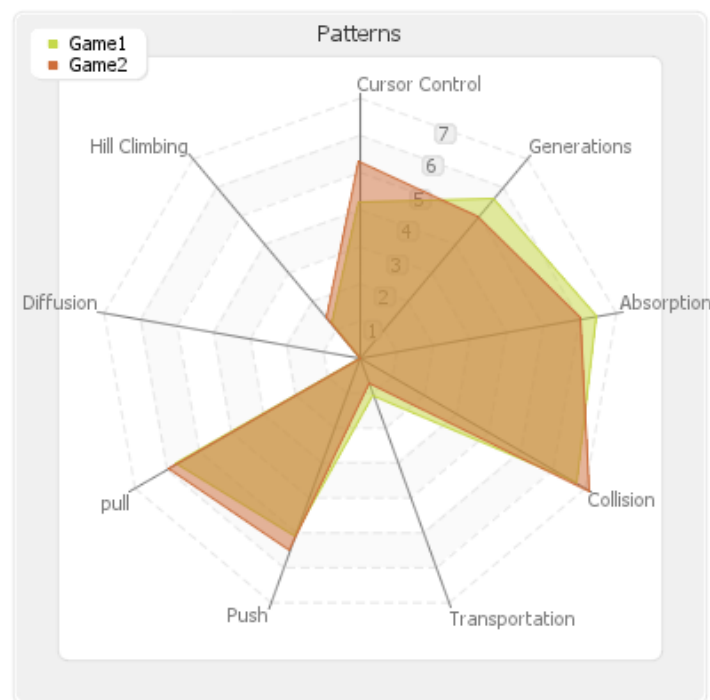


Figure 17 CTPA graph comparison of Game 1 (Green) and Game 2 (Brown)

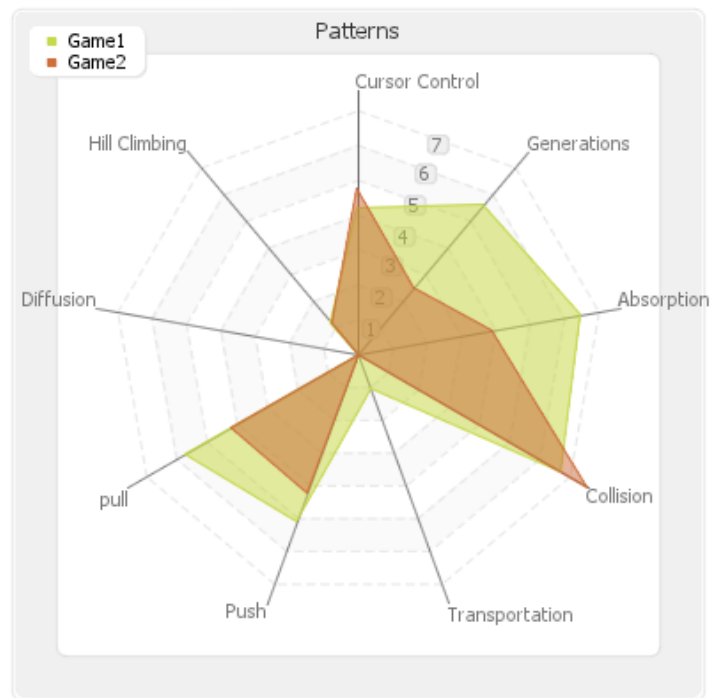


Figure 18 CTPA graph comparison of Game 1 (Green) and Game 3 (Brown)

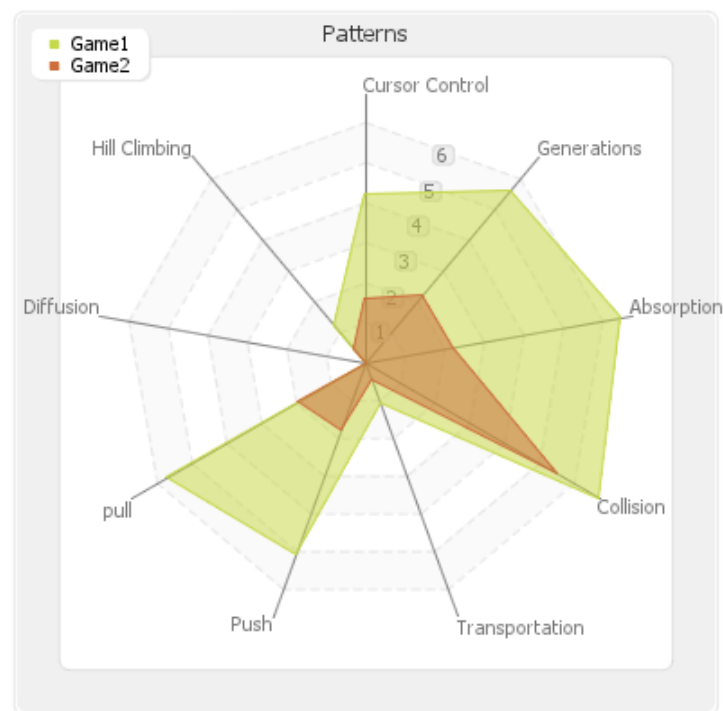


Figure 19 CTPA graph comparison of Game 1 (Green) and Game 4 (Brown)

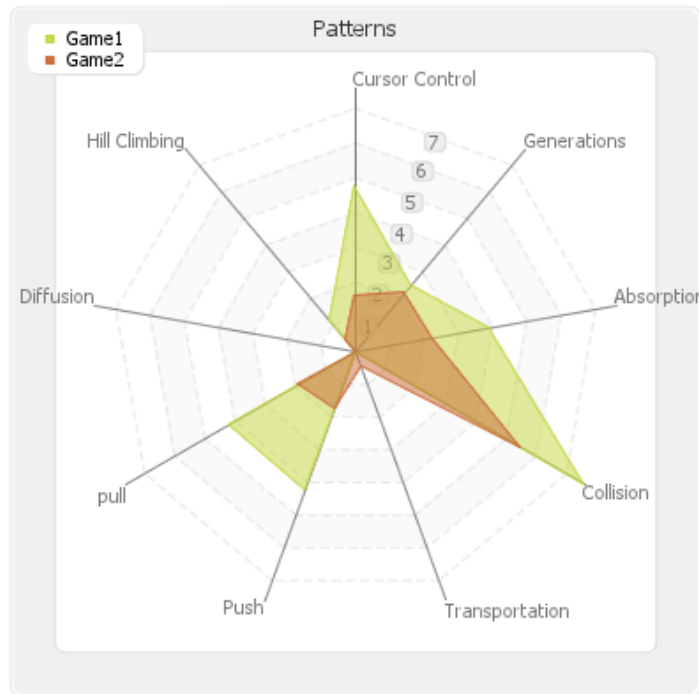


Figure 20 CTPA graph comparison of Game 3 (Green) and Game 4 (Brown)

Chapter 5. Research Validation

Since the early stage of CTPA development, the CTPA has been validated formally and informally with traditional Computer-Automated Scoring (CAS) system validation approaches and other educational validation approaches under the two IRB permissions that were previously approved (0808.21 and 11-0379).

CTPA can be considered as not only a CAS system but also a program phenomenological analysis tool. So the CTPA should be validated using several different validation approaches. For the following validation studies, teachers from the project's participating schools, students and researchers from University of Colorado at Boulder were recruited.

My validation studies followed the CAS system validation approaches that Yang et al. classified as follows (Yang et al., 2002).

- Approaches focusing on relationships among scores generated by different scorers
- Approaches focusing on scoring processes

I added one additional category to the above CAS system validation approaches:

- Approaches focusing on predictive validity

5.1. Computational Thinking Skill Progression

While the semantic information from individual games/simulations is only a piece of student learning development, it could provide a measurement for a student's entire skill progress. Representing semantic meaning in measureable units to visually

demonstrate student learning trends can benefit students and teachers directly. This approach could also indicate possible curriculum failings at a fundamental level.

The value of each axis on the CTPA Graph represents the proportion of implemented knowledge for a given computational thinking pattern within a game/simulation. The sum or average of these values is interpreted as the student's skill in designing the game/simulation, much as an average GPA from several classes is understood to represent a student's overall learning. That is, the nine computational thinking patterns are target-learning categories. A sample game accompanies each tutorial, and I created a target score for each CT pattern in each game by running the sample game through CTPA. Thus, the CTPA Graph illustrates how well students meet the target-learning goal in each assignment or group of assignments. Within the CTPA, a one-time assignment analysis is referred to as a Demonstrated Skill Score. Learning that takes place over time through several assignments is referred to as a Comprehensive Skill Score. Both Demonstrated and Comprehensive Skill Scores are calculated from the length (norm) of a vector of the nine computational thinking patterns reduced to one dimension (unit).

The Demonstrated and Comprehensive Skill Scores are calculated using the following equations.

$$\text{Demonstrated Skill Score } (n) = \frac{\sqrt{\sum_{i=1}^n (P_i)^2}}{\sqrt{n}}$$

Equation 3. Demonstrated Skill Score

$$\text{Comprehensive Skill Score } (m) = \frac{\sqrt{\sum_{i=1}^n [\max_{j=1}^m (P_{i,j})]^2}}{\sqrt{n}}$$

Equation 4. Comprehensive Skill Score

In these equations, P is a computational thinking pattern, n is the number of computational thinking patterns on the CTPA Graph, and m is the number of submitted assignments. Those equations are derived from the formula for the length of a vector.

As an example, the learning skill scores of two consecutive game submissions from one student (Sokoban and Pacman) are shown in the table below (Table 3). The bold and italic styled values are the maximum computational thinking pattern values that the student has received through two game submissions. Those maximum values are used to calculate the student's second game's comprehensive skill score (Pacman).

Computational Thinking Pattern	CTP value from 1 st game Submission (Sokoban)	CTP value from 2 nd game Submission (Pacman)
Cursor Control	<i>0.52</i>	0.47
Generation	0.35	0.36
Absorption	0.48	0.55
Collision	0.56	<i>0.70</i>
Transportation	<i>0.64</i>	0.50
Push	<i>0.82</i>	0.60

Pull	<i>0.75</i>	0.60
Diffusion	0.16	0.18
Hill Climbing	0.43	0.36
Demonstrated Skill Score	<i>0.56</i>	<i>0.50</i>
Comprehensive Skill Score	<i>0.56</i>	<i>0.58</i>

Table 3: Example of Learning Skill Score Calculation

The Demonstrated Skill Score shows a student's programming skill as of when the game was submitted, while the Comprehensive Skill Score shows a student's progress in skill acquisition over time. Each Skill Score is the normalized size of the value on each axis of the CTPA Graph. For the Comprehensive Skill Score calculation, I make the following assumption to track students' skill progression: if there is a skill that a student has learned and demonstrated accurately at least once, then that skill is available for the student to use for the entire duration of the course even if it is not used again. In other words, a maximum value of any given game represents its creator's (student) best achieved level in CT pattern implementation. Consequently the maximum value is selected in this equation.

5.2. ASSESSMENT VALIDATION

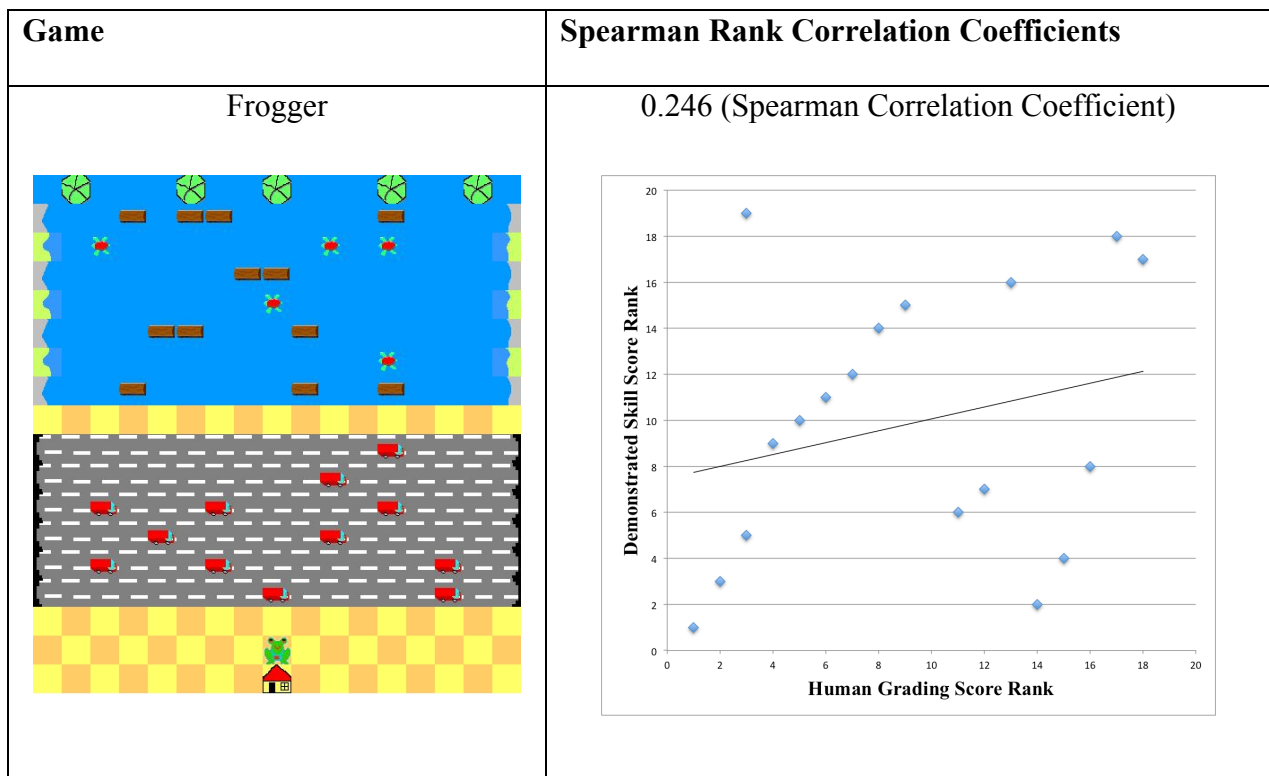
To gauge the value of CTPA as a Computational Thinking assessment tool, I conducted the early stages of concurrent validity and predictive validity evaluation using data from CU undergraduate and graduate students who took an Educational Game Design class in 2012 and 2013. For concurrent validity I compared 39 students' grades (19 students for the 2012 class and 20 students for the 2013 class) with CTPA-measured

skills for four basic games: Frogger, Sokoban, Centipede, and the Sims. To assess predictive validity I computed students' comprehensive skill scores based on the four basic games and compared them to the demonstrated skill scores of their final projects.

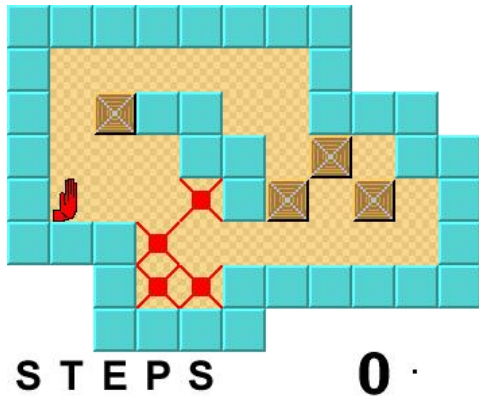
5.2.1. 2012 Class Concurrent Validity Results

For the 2012 class (19 students), I hired two graders for this research who were asked to provide grades based on the official grading rubric for each game. I also used CTPA to calculate a demonstrated skill score for each game.

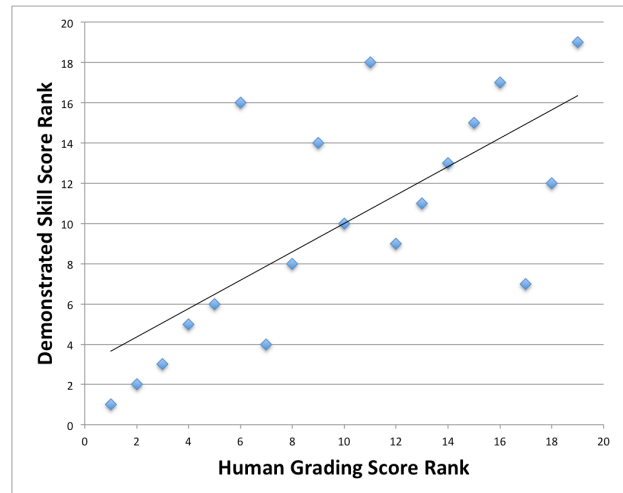
The human grades and the demonstrated skill scores are not normally distributed. Instead, they are skewed negatively. Therefore, I calculated the Spearman rank correlation coefficient to measure the statistical dependence between the CTPA-measured skills of students and the grades that they actually received.



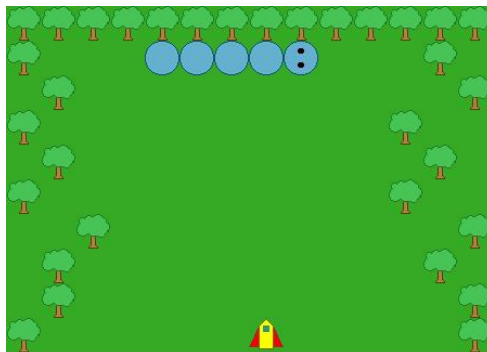
Sokoban



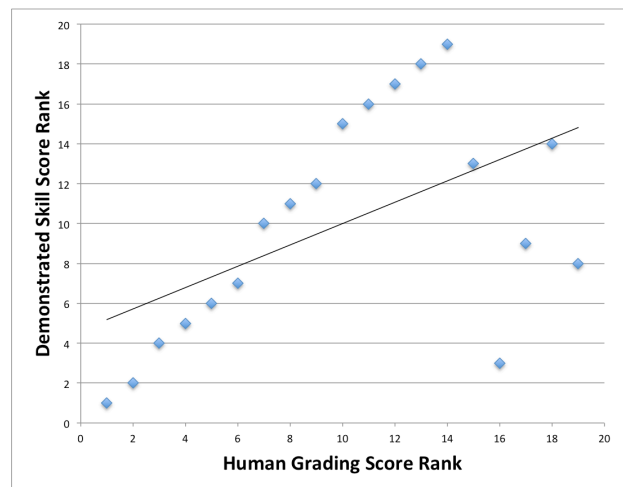
0.705 (Spearman Correlation Coefficient)



Centipede



0.535(Spearman Correlation Coefficient)



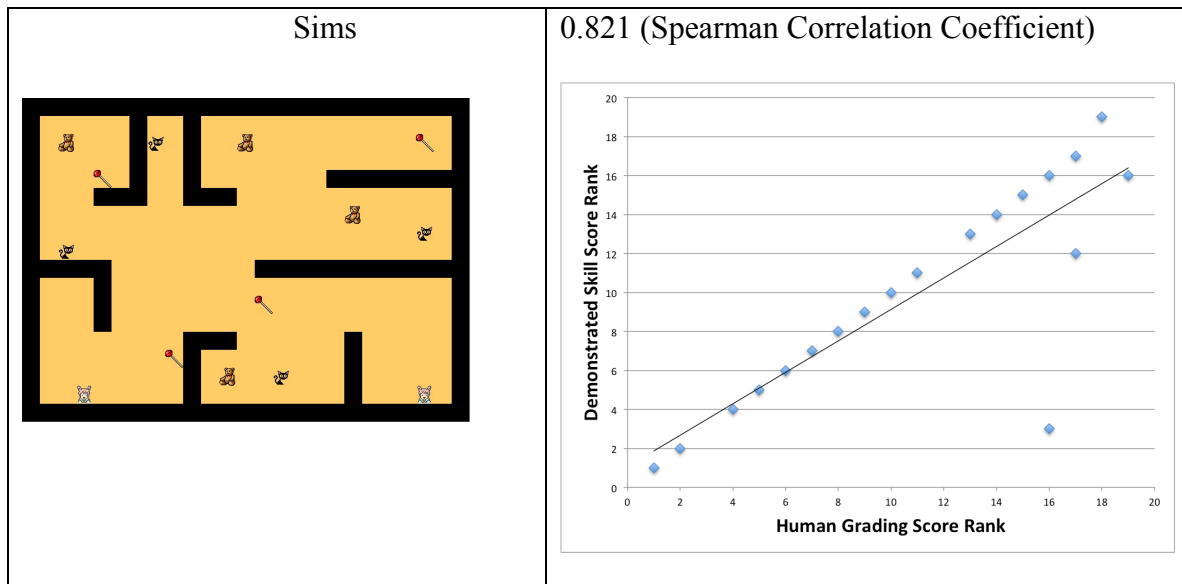


Table 4 Four Basic Games and Spearman Rank Correlation Charts for the 2012 Class

5.2.2. Demonstrated Skill for Individual Games

As Table 4 shows, the Spearman rank correlation coefficients for three of the four basic games are high enough to demonstrate a correlation between human graded scores and CTPA-measured skills. These results indicate that CTPA is capable of measuring students' skills, and its measured results connect well with the human grades.

Although the originality and the design of the game were part of human grading, CTPA measures only programming skills. So for the tied scores, the person who received a higher grade in programming is ranked higher than the person who got a higher grade in originality and design. For example, there are two students who received 100 points where student A received 90 points for basic programming and 10 points for advanced design and student B received 80 points for basic programming and 20 points for advanced design. In this case, student A is ranked higher than student B. If students received exactly the same scores for basic and advanced programming, then they are

ranked based on their programming completeness (i.e., avoiding undeclared variables/methods or unnecessary programming components).

5.2.3. Comprehensive Skill Across Several Games

I also calculated students' comprehensive skill scores to reflect the correlation between the average student grades and CTPA-measured skill scores when students finished making all four basic games.

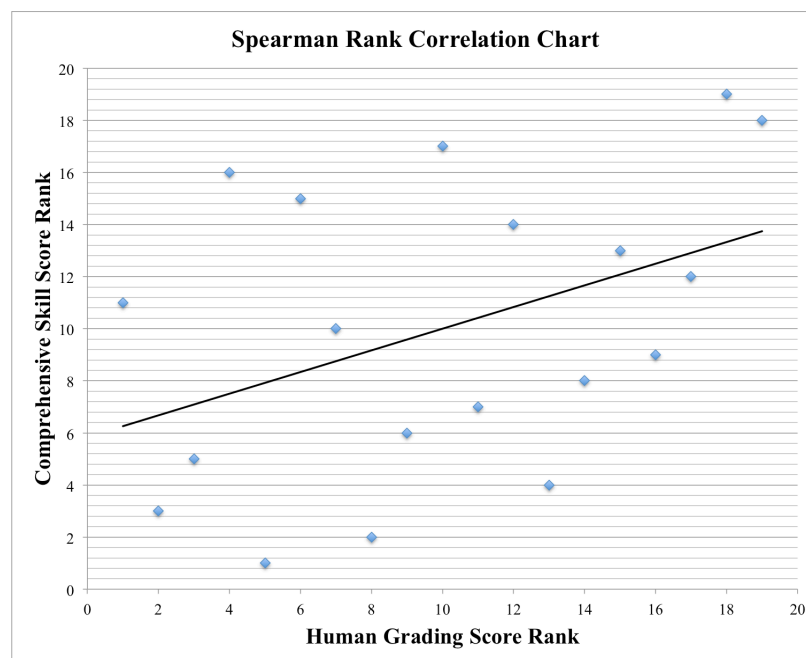


Figure 21 2012 Class Spearman Rank Correlation Chart

The Spearman rank correlation coefficient value between students' grades and their CTPA-measured skill scores is 0.415 (Figure 21). This number indicates a moderate level of positive correlation between students' grades and their CTPA-measured skill scores. Due to the small sample size, I verified its significance with critical values for the Spearman rank correlation coefficient. The critical value for $N=19$ with a significance level of 0.05 is 0.391, which is lower than the calculated correlation coefficient, 0.415.

This calculation indicates that there is a greater than 95% chance of the correlation being truly significant. This result offers another positive indication of the CTPA's validity as a programming assessment tool, suggesting that it would be usable in a real classroom situation.

5.2.4. Inter-Rater Agreement

To check the inter-rater agreement between the two graders, I converted the original 1 to 100 scale scores to letter grades from A to F. In a 1 to 100 scale, there are 100 options for grades, and it was difficult to get high inter-rater agreement percentages since there were so many scores that are close but not exactly the same (i.e. 93 vs. 95). I converted the scores above 90 to A, the scores above 80 to B, the scores above 70 to C, the scores above 60 to D, and the scores below 60 to F.

The inter-rater agreement percentage between the two graders was 95% on average for the four basic game grades.

5.2.5. 2013 Class Concurrent Validity Results

For the 2013 class (20 students), I hired one of the two graders who graded the 2012 class. The same rubric was provided for grading consistency. As for the 2012 class, the students' comprehensive skill scores were calculated as the basis for determining the correlation between average student grades and CTPA-measured skill scores when students finished making four basic games.

The Spearman rank correlation coefficient value between students' grades and their CTPA-measured skills is 0.476 (Figure 22). This number indicates a moderate level of positive correlation between students' grades and their CTPA-measured skill scores.

Due to the small sample size, I again verified its significance with critical values for the Spearman rank correlation coefficient. The critical value for $N=20$ with a significance level of 0.025 is 0.447, which is lower than the calculated correlation coefficient, 0.476. This calculation indicates that there is a greater than 97.5% chance of the correlation being truly significant. This result illustrates the reliability of CTPA-measured skill scores over two consecutive classes.

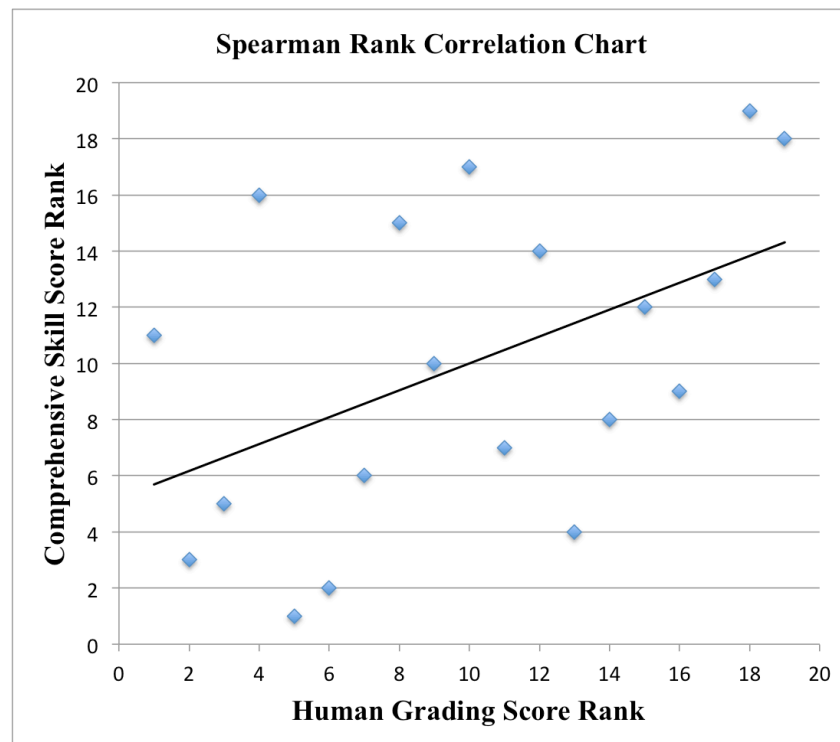


Figure 22 2013 Class Spearman Rank Correlation Chart

5.2.6. Predictive Validity Results

I then performed a predictive validity test to confirm CTPA's validity as a programming assessment tool. In contrast to the four basic games, the final project was graded based on originality, educational facts, engagement, and student presentation

skills rather than programming skills. Thus, for predictive validity, it was not adequate to compare CTPA-measured student skills and student grades.

However, it is possible to use a pure programming comparison to predict students' future achievements based on their previous skills. In other words, if a student has shown high achievement through previous assignments, then s/he is expected to show high achievement in the final project, too. I therefore computed student CTPA-measured skills to show their correlation between pre-final projects and the final project. As Figure 23 illustrates, those who showed better performance through pre-final assignments tended to show better performance in the final project also. For the 2012 class, the Pearson correlation coefficient value between pre-final projects and the final project is 0.676, and there is a 99.5% chance of this correlation being truly significant. For a better correlation calculation, I excluded two students who missed more than three assignments and one student who didn't submit his final project.

This high correlation between Skill scores from pre-final projects and the final project implies that CTPA is able to predict a student's future learning performance and skill trajectory. This capability of CTPA can be applied to build a cyberlearning infrastructure including automated tutoring systems.

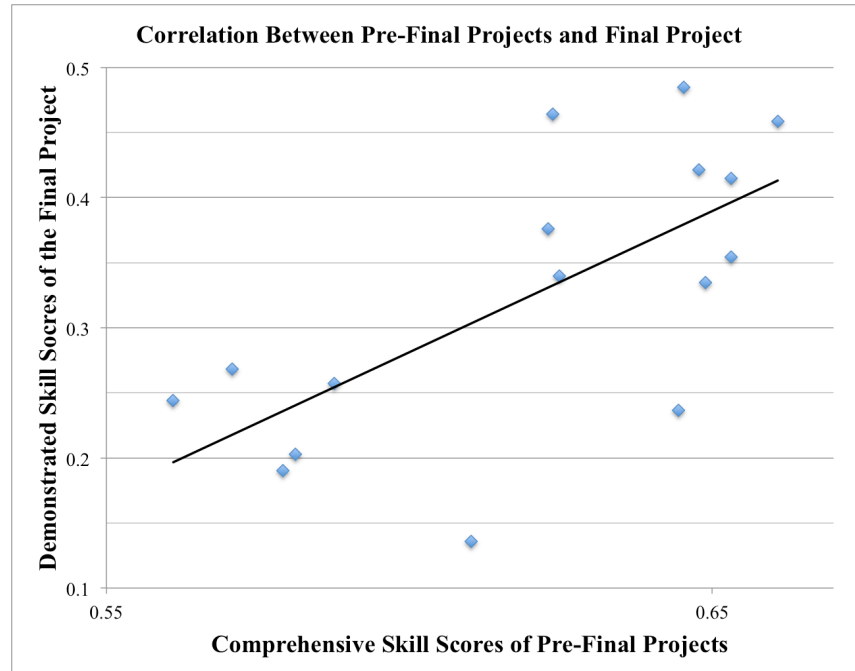


Figure 23. Predictive Validity Evaluation from 2012 Class

5.2.7. Summary of Concurrent and Predictive Validation Evaluation

In this initial foray into CTPA validation, I found satisfactorily strong positive correlations between scores given by human graders and students' comprehensive skill scores calculated by CTPA, giving me confidence about proceeding with further validation activities. Several factors suggest that the correlations described here are lower than those I might expect to find during additional validation, including the small size of the samples. The current human grader scoring rubric includes both programming skill items, which are closely related to the characteristics examined through CTPA, and other, less related items. For example, the graders checked for the presence of expected computational thinking pattern implementation, and also looked for what users should experience while the game is played. Therefore, human graders are evaluating game

design skill along with programming skill. A revised rubric with greater emphasis on programming would be expected to lead to higher correlations, but it might be a better idea to change or update the current CTPA to better match what human graders do than vice versa. See the next section on face/construct validity.

5.3. Face and Construct Validity Evaluations

Converting CTPA into a holistic score is not the only way to validate CTPA. CTPA can be validated with approaches focusing on scoring processes. For example, when a game of Frogger is graded, a grader will look at the functionalities of Frogger: Frogger can be moved with key controls (cursor control), a tunnel creates cars or trucks (generation), etc. Thus, CTPA also should be able to check the functionalities of computational thinking patterns as a human scorer does. To do so, each canonical pattern of CTPA should be matched to the basic requirement of a human grader's or a student's perception.

I conducted face and construct validity evaluations of CTPA with Scalable Game Design project researchers and teachers along with graders who have graded an Educational Game Design class where AgentSheets or AgentCubes programming were taught. All participants were able to detect a certain pattern from AgentSheets or AgentCubes programming codes.

5.3.1. Evaluation with Programming Codes

All participants were asked to describe which pattern they could detect by examining AgentSheets programming codes for nine cases: Cursor Control, Generation, Absorption, Collision, Transportation, Push, Pull, Diffusion, and Hill Climbing. For

example, participants were asked whether they could see a Cursor Control pattern by examining the following AgentSheets programming codes (Figure 24 and 25). For each pattern, two programming questions were asked, except the Diffusion pattern, for which 3 programming questions were asked. For example, the two programming questions for the Cursor Control pattern are illustrated in Figures 24 and 25, depicting the correct method and the incorrect method respectively. A complete questionnaire with 18 questions (two per pattern) can be found in Appendix C.

“Using the following AgentSheets programming codes, can you detect the Cursor Control pattern?”



Figure 24: Cursor Control Programming (Correct Method)

“Using the following AgentSheets programming codes, can you detect the Cursor Control pattern?”

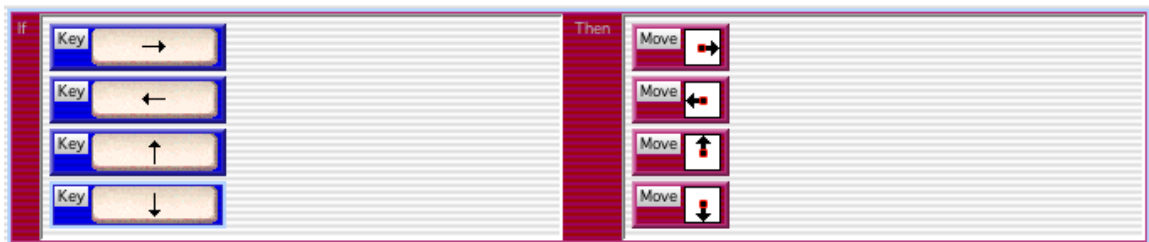


Figure 25: Cursor Control Programming (Incorrect Method)

CTPA shows the same graph (Figure 26) for those two above programming codes because CTPA cannot detect where the condition and action components lie.

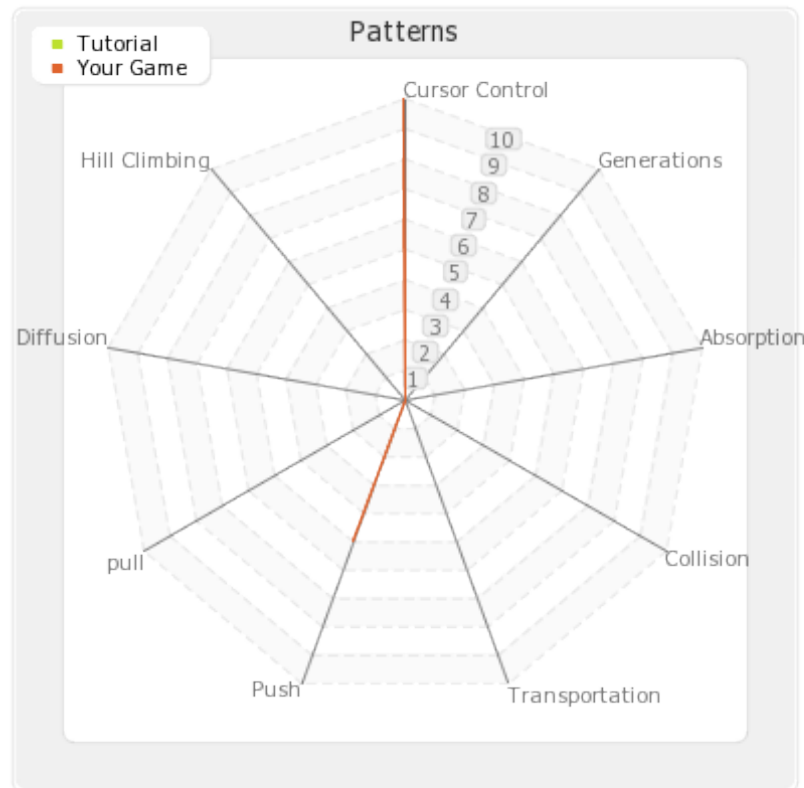


Figure 26: CTPA Graph of Cursor Control Pattern

The CTPA graph in Figure 26 shows some false positive values of the Push pattern, which originate from the Key (Keyboard) condition components in the Cursor Control pattern. This occurs because the Key condition is used in the implementation of both the Push and the Cursor Control patterns.

5.3.2. Evaluation Results

There were in total ten participants in this evaluation: five participants from the SGD research team, three SGD project teachers, and two SGD class graders. Only the

first question for each pattern shows the correct method to implement a given pattern, and the second or the third question shows an incorrect method to program the given pattern with the same programming (condition and action) components as in the correct method. So the CTPA graphs of the correct method and the incorrect method are identical to each other. The CTPA graph for each pattern can be found in Appendix A.

The participants' responses are recorded in the following table to compare human recognition and CTPA recognition of the given pattern. CTPA detected a given CT pattern in both correct and incorrect methods because both programming codes are identical as far as CTPA is concerned. All participants were able to detect a given pattern in the correct method implementation, but there were notable differences among several incorrect method implementations.

All participants responded yes for the Cursor Control, Generation, Absorption, Collision, Transportation, Diffusion, and Hill Climbing patterns in the correct implementation, but two participants responded no for the Push and Pull patterns in the correct implementation and answered yes for those two patterns in the incorrect implementation. Those participants are the SGD teachers including one participant in the "I don't know" category.

The interesting result was found in the questions with the incorrect methods. I could categorize the rationale behind the participants' judgment into two categories: one is judgments based on conditions and the other is judgments based on actions. Two SGD project teachers, two SGD researchers, and two graders perceived a given pattern based on conditions, and one SGD project teacher and three SGD researchers recognized a given pattern based on actions. In other words, the participants who answered no for the

given pattern in the incorrect method justified their responses with the disconnect between conditions and actions, and the participants who replied yes explained their responses with the existence of specific actions. For example, for the Generation pattern, most justifications for the “No” response indicated the disconnection between the See and Chance conditions and the New action while others for the “Yes” response classified the New action as a valid Generation pattern. I quoted two justifications from the “Yes” response and the “No” response for the Generation pattern in the incorrect method.

Yes. I would still call this generation, although not a well done generation. An agent is still visibly GENERATED here.

Technically, the action will look like generate visually, but it seems unlikely that the code expresses the user’s intent. A new agent will be created if there is no gray agent to the right AND the 50% chance happens to be false. To me, this set of rules does not express the generation pattern.

Interestingly, experienced participants in the SGD project tend to follow the action-based judgment. There are four participants who followed the action-based judgment for the CT pattern recognition, and all of them have at least four years experience in the SGD research.

Computational Thinking Pattern	Survey Question Number	CTPA	Humans (10 people)		
			Yes	No	I don’t know/ I’m not sure
Cursor Control	1 (Correct Method)	Yes	10	0	0
	2	Yes	1	9	0
Generation	3 (Correct Method)	Yes	10	0	0
	4	Yes	4	6	0
Absorption	5 (Correct Method)	Yes	10	0	0

	6	Yes	5	5	0
Collision	7 (Correct Method)	Yes	10	0	0
	8	Yes	3	7	0
Transportation	9 (Correct Method)	Yes	10	0	0
	10	Yes	0	10	0
Push	11 (Correct Method)	Yes	9	0	1
	12	Yes	1	7	2
Pull	13 (Correct Method)	Yes	7	2	1
	14	Yes	4	5	1
Diffusion	15 (Correct Method)	Yes	10	0	0
	16	Yes	0	8	2
	17	No	0	8	2
Hill Climbing	18 (Correct Method)	Yes	10	0	0
	19	Yes	0	10	0

Table 5: Evaluation Results

Also, this evaluation could be used to assess the quality of the SGD teacher training. For example, two SGD teachers were not able to explain their responses to the incorrect Diffusion implementations (Questions 16 and 17). They were confused by the wrong equations for the Diffusion pattern, and they didn't understand what the diffusion equation which is shown in the correct method expresses. This result could indicate that the SGD teacher training should be revised to help teachers understand the logic and the algorithms behind programming implementations.

In conclusion, even though CTPA didn't fail to detect the existence of a given CT pattern in which human participants were able to recognize the CT pattern, the CTPA cannot differentiate the correct method and the incorrect method to implement a given pattern if same programming components are used in both methods. This is one of the limitations of the current CTPA method, and this limitation can be found in other vector

space model approaches that the CTPA is inspired by. Additionally, this evaluation result possibly indicates the need for refining the current canonical computational thinking patterns. More than 30% of participants responded yes to the incorrect implementation for four CT patterns, and those participants are the most experienced ones among the ten participants. This means that the current canonical computational thinking patterns do not earn complete human expert agreements for certain CT patterns.

5.3.3. Evaluation with Proportional Programming

The proportion of each computational thinking pattern in a given AgentSheets or AgentCubes project is used to draw the CTPA graph. The pattern with the highest value on the CTPA graph is the most dominant pattern in a particular AgentSheets or AgentCubes project. To validate its proportional illustration, I designed an experiment to compare human perception of the pattern proportions in a specific game programmed using AgentSheets.

For example, a CTPA graph of a Frogger game shows Cursor Control, Generation, Absorption, Collision, Transportation, Push, Pull, and Hill Climbing patterns. Among those eight patterns, Push, Pull, and Hill Climbing patterns are false positive patterns. Those false positive values occur because there are several commonly used condition and action components in the Generation, Absorption, Collision, Transportation, Push, Pull, and Hill Climbing patterns. For example, the Move condition component is used in the Cursor Control, Push, Pull, and Hill Climbing pattern implementations.

Besides the false positive patterns, I asked participants to rank the proportion of each pattern in a game of Frogger as the below paragraphs. The complete survey questionnaire for this evaluation can be found in Appendix D.

“Please open and play the sample Frogger project, which is under the Projects folder of AgentSheets. Please list the Computational Thinking Patterns in a proportional rank in the Frogger programming using your understanding of Computational Thinking Patterns. The sample Frogger uses Cursor Control, Generation, Absorption, Collision, and Transportation patterns. (i.e. 1. Transportation 2. Cursor Control 3. Generation 4. Absorption 5. Collision. A smaller number means a higher proportion in programming)”

The participants ranked Cursor Control, Generation, Absorption, Collision, and Transportation in a proportional rank using their understanding of Computational Thinking Patterns.

Following the CTPA graph of our tutorial Frogger game in Figure 27, Absorption is the most dominant pattern, and Collision, Generation, Cursor Control, and Transportation follow. In programming, the Absorption pattern (Figure 28) is a subset of the Collision pattern (Figure 29) so their proportions are highly correlated; the value of the Absorption pattern is always higher than the value of the Collision pattern. However, it was unclear whether or not human evaluators were able to recognize patterns' subset relation, total independence, or exclusivity to each other.

Computational Thinking Patterns

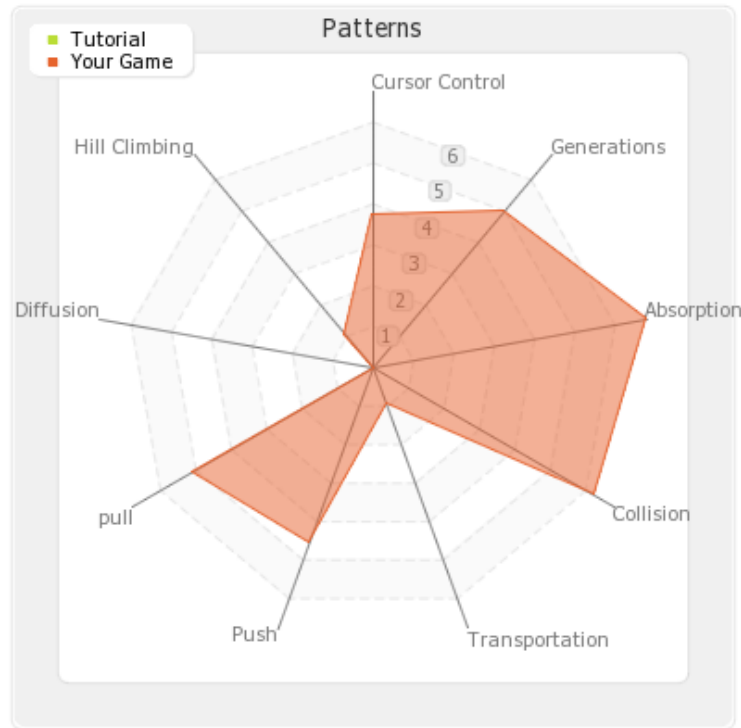


Figure 27: A CTPA graph of a game of Frogger

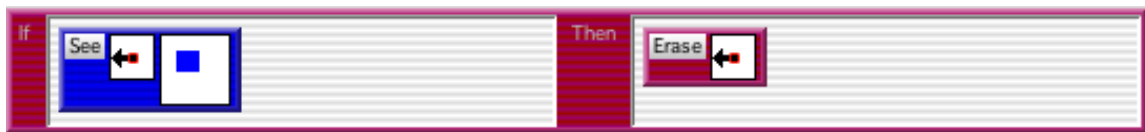


Figure 28 An example of Absorption pattern programming

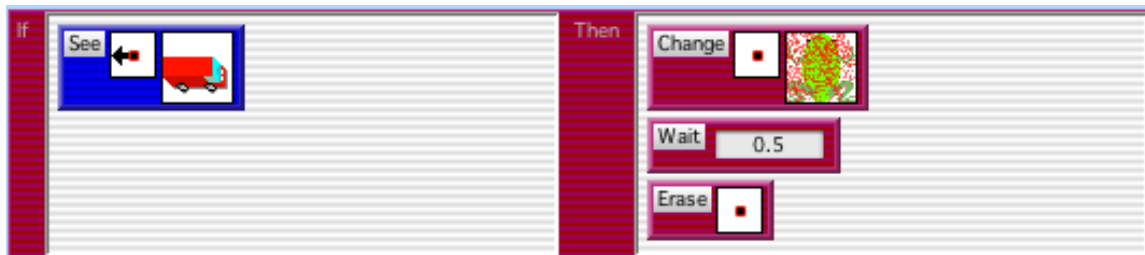


Figure 29 An example of Collision pattern programming

5.3.4. Evaluation Results with Proportional Programming

This evaluation was designed to compare people's perception of the Computational Thinking Patterns and CTPA's Computational Thinking Pattern detection in a given game. I asked the participants not to access the codes of the games but to play the games to respond the survey questions.

The evaluation result illustrates that the CT pattern detection by CTPA and human perception of CT patterns in a given game are not highly correlated, and the individual responses between participants are not highly correlated either. Nevertheless, there is one pattern per game with high or low ranking agreement among participants: the Transportation pattern of Frogger, the Pull pattern of Sokoban, and the Choreography pattern of Space Invaders. Besides the Transportation pattern, the CT pattern detection of CTPA corresponds to the participants' responses.

Game	CT Patterns ranked by CTPA	CT Patterns ranked by Participants
Frogger	1. Absorption	1. Generation
	2. Collision	2. Collision
	3. Generation	3. Absorption
	4. Cursor Control	4. Cursor Control
	5. Transportation	5. Transportation
Sokoban	1. Pull	1. Push
	2. Push	2. Cursor Control
	3. Cursor Control	3. Pull
Space Invaders	1. Choreography	1. Choreography
	2. Absorption	2. Collision
	3. Collision	3. Generation
	4. Cursor Control	4. Absorption
	5. Generation	5. Cursor Control

Table 6 Evaluation Results

The game of Space Invaders has five Computational Thinking Patterns, and the Choreography pattern, one of those five CT patterns, is not covered by the current CTPA. So for the evaluation of Space Invaders, I compared participants' responses with a 13 dimensional version of CTPA which has four more Computational Thinking Patterns than the current CTPA, including the Choreography pattern. The 13 dimensional CTPA graphs with four standard games can be found in Appendix E.

As I mentioned in the previous chapter, the Absorption pattern is a subset of the Collision pattern, so the Absorption pattern happens whenever the Collision pattern happens. Thus, the Absorption pattern should be ranked higher than the Collision pattern. However, in the evaluation result, nine participants ranked the Collision pattern higher than the Absorption pattern. One participant who ranked the Absorption pattern higher than the Collision pattern was informed about the subset relation between the Absorption pattern and the Collision pattern before she took this evaluation.

In conclusion, the Computational Thinking Pattern representation of CTPA does not show a high correlation with human perceptions of the Computational Thinking Patterns, but from this evaluation, at least I can conclude that CTPA is better than human perception at identifying CT patterns' subset relations.

Chapter 6. Applications of Computational Thinking Pattern Analysis

All AgentSheets and AgentCubes projects collected through SGDA were analyzed by CTPA. The analyzed results of CTPA have shown promise in providing educational feedback in the areas of learning transfer (Koh et al., 2010), learning trajectory (Bennett et al., 2011), and programming divergence (Koh et al., 2011; Bennett et al., 2013).

6.1. Early Indicator of Transfer

Bransford et al. (Bransford et al., 1999) describe knowledge transfer as the most common method for human beings to learn the necessary components of life. Transfer is defined as the ability to extend or use what has been learned in one context into a new context or to solve a new problem. Knowledge transfer can be aided by using multiple contexts (the more diverse settings, the better) to demonstrate new concepts to students. The new knowledge can then be retained by the student in a more abstract form. When new types of future situations occur, this knowledge can then be accessed by the student. Students are not normally able to transfer purely conceptual information to real world situations without help (Bransford et al., 1999; Vgotsky 1978). Linking any concept to a single setting or context can also cause difficulty with transferring knowledge to new situations. So, although transfer is our preferred mode of learning and retaining new information, transfer cannot be assumed in any given context. Previous knowledge that students build upon can also enhance or deter the effort to assimilate new information.

Consequently, the ability to detect possible knowledge transfer could benefit researchers in many disciplinary areas (Koh et al., 2010).

Since learning and knowledge from the field of Computer Science in general can potentially be integrated and used productively in many other disciplines, promoting the transfer of computer science knowledge into these areas could substantially enhance learning and research within the computer science field. If there were a tool which could detect the potential transfer of computer science knowledge to other disciplines, the tool would tend to increase the breadth and validity of computer science research, and contribute to the growth of the field. Graphs generated by CTPA could potentially demonstrate the existence of knowledge transfer, not just within related computer science fields, but also across disciplinary lines.

The CTPA was first developed as a means to offer feedback to students uploading their games to the Scalable Game Design Arcade (SGDA). The SGDA served as a submission format for the CU Boulder educational game design course using AgentSheets. During the semester, students are exposed to simple computational thinking patterns; as the class progresses they are introduced to more complex and diverse computational thinking patterns. Towards the end of the class, students are given open-ended assignments. For these assignments students are encouraged to build on their initial knowledge from the class in order to create their games. For the final project, students often choose to create simulations that depict some natural phenomena. Semantically analyzing a given student's games from the beginning of the semester as compared to their final project (especially a science simulation), could offer an opportunity to discover potential knowledge transfer.

For instance, a chaos theory simulation created by one student (Figure 34) with the accompanying CTPA graph (Figure 35) shows how he mixed and combined computational thinking patterns that he had learned and used when previously programming Sokoban (Figure 30) and Sims (Figure 32). The CTPA graph (Figure 35) of his science simulation is very similar to the combined CTPA graphs of Sokoban and Sims (Figure 36). Consequently, for this student, the CTPA graphs indicate that knowledge transfer has occurred.

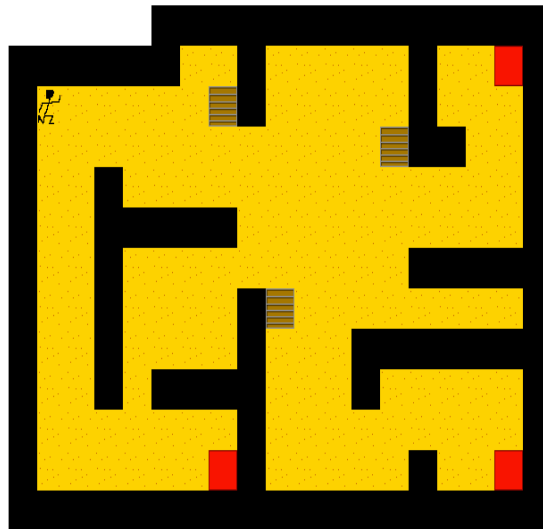


Figure 30 A Screenshot of Sokoban

Computational Thinking Patterns

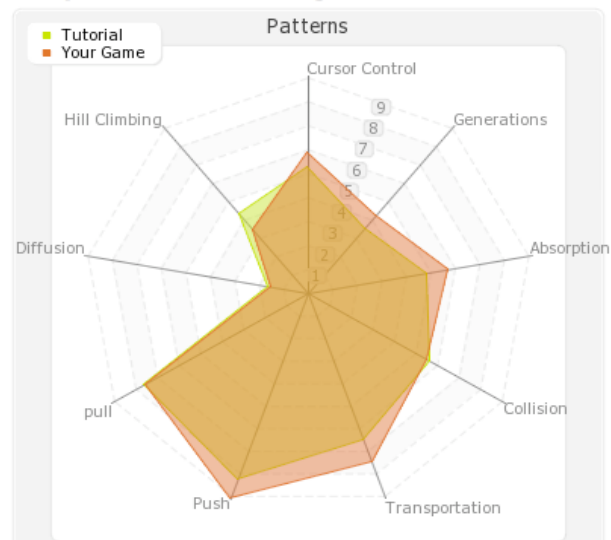


Figure 31 A CTPA Graph of Sokoban

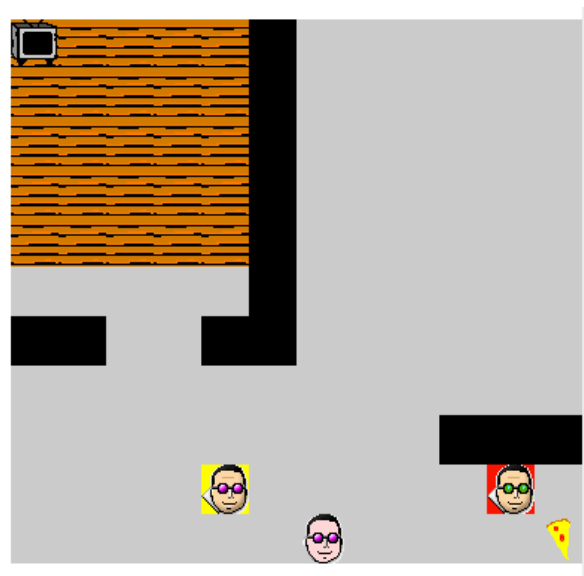


Figure 32 A Screenshot of Sims

Computational Thinking Patterns

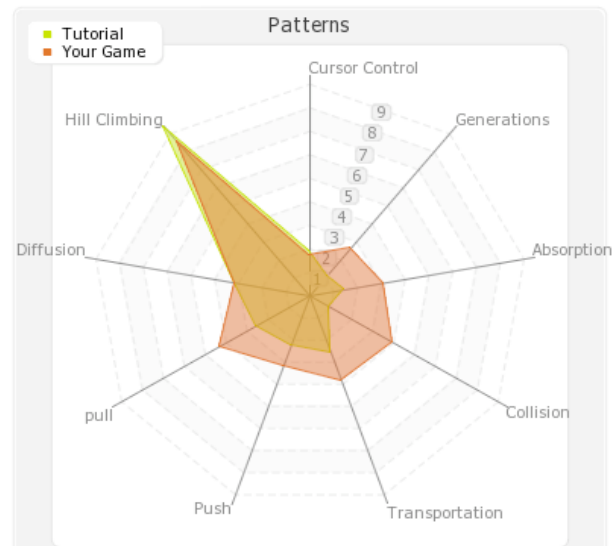


Figure 33 A CTPA Graph of Sims

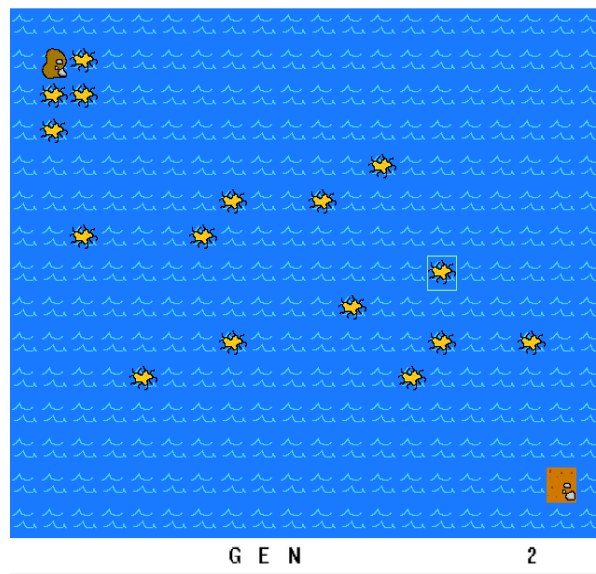


Figure 34 A Screenshot of Chaos Theory Simulation

Computational Thinking Patterns

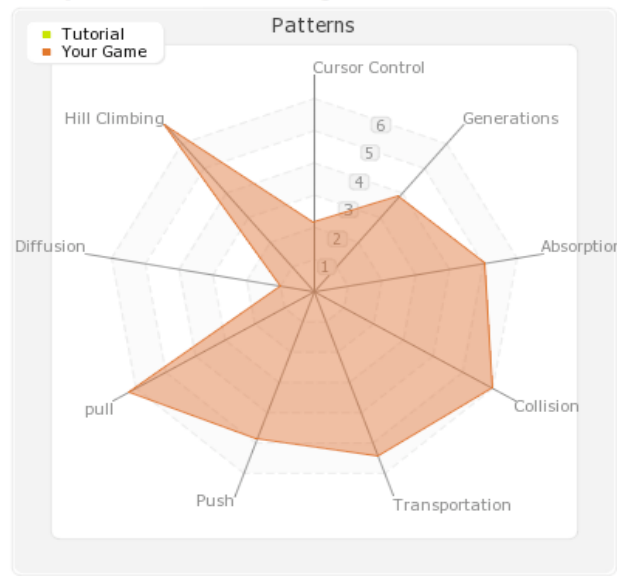


Figure 35 A CTPA Graph of Chaos Theory Simulation

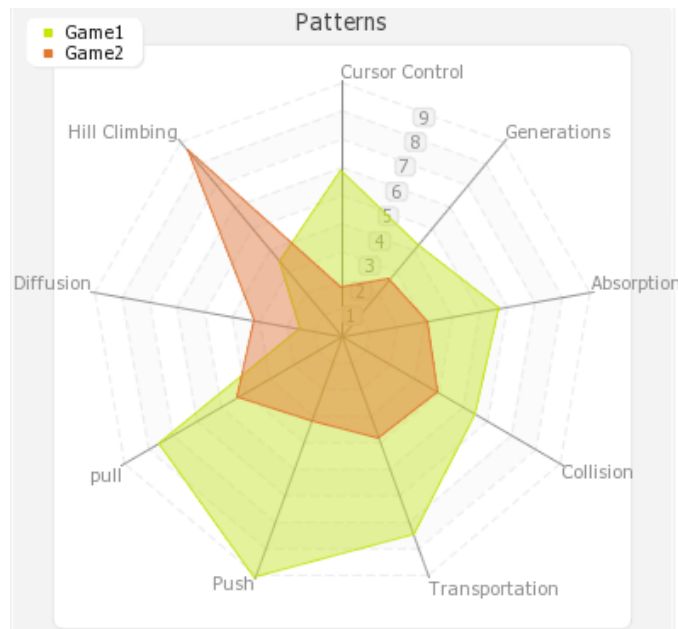


Figure 36 Comparison of CTPA Graphs: Depicts the Sims-Sokoban combination

6.1.1. Summary of Assessing Student Products Using the CTPA

For the last decade, several visual programming languages have provided easy ways for young children to learn programming concepts and skills. Many of these visual languages successfully motivate students. However, visual language research has not focused on what kind of knowledge students have actually learned from creating these games. The CTPA provides an initial way to assess specific knowledge accumulated by students within a given class.

As I have observed, the ability to detect computational thinking patterns is important for school teachers and students using visual languages for education. The CTPA provides us with an initial means to answer the question “Now that the student can program Space Invaders, can the student program a science simulation?” Furthermore, the CTPA has the ability to enable Human Centric Computing as teachers can get immediate feedback on their student’s progress.

Analyzing computational thinking patterns in multiple combinations could improve our ability to assess the depth and breadth of students’ knowledge. The semantic nature of the CTPA graph allows us to evaluate and visualize a program’s actual underlying meaning. A syntactic evaluation of a student’s learning (Lewis, 2010) only shows the student’s knowledge in a very limited context. Moreover, the implementation of a given student’s previously learned computational thinking patterns in a scientific context gives us a clearer picture of how the student transferred new knowledge to a new situation, demonstrating through the CTPA graph comparison that knowledge transfer exists.

Although in most learning scenarios, knowledge transfer is often assumed, this transfer cannot be guaranteed to have actually taken place. The graph generated by CTPA is a better tool for evaluating knowledge transfer because the graph represents the CTP combinations as an observable and definable outcome. The ability to detect knowledge transfer through the CTPA graph, over the duration of a semester course, is a positive first step towards measuring transfer in other areas, and possibly other forms of learning.

6.2. Learning Skill Assessment

While the semantic information from individual games/simulations is only a piece of student learning development, the semantic analysis of individually created games or simulations could provide a measurement for a student's entire skill progress. Representing semantic meaning in measureable units to visually demonstrate student-learning trends, and to represent the students' knowledge and skill, can benefit students and teachers directly. This approach could also indicate possible curriculum failings at a fundamental level.

A learning skill score (Bennett et al., 2011; Koh et al., 2014), which I described in the previous chapter, can be used to track an individual student's learning progression or the entire class's learning progression. For this project, I explored two entire classes' learning progressions (one middle school class and one college class) as computed with CTPA. Teachers need to know how students are accomplishing individual assignments, how students' learning progresses over an entire class year or semester, and how the class as a whole is learning. The Demonstrated Skill Score shows a student's programming skill score as of when the game was submitted, while the Comprehensive Skill Score shows a student's progressed learning over time. Each Skill Score is a normalized

maximum vector length of the value on each axis of the CTPA Graph over the entire class section for easy use as a grading tool or learning progress indicator tool. Through this vector normalization, CTPA information in nine dimensions is represented in one dimension for easy human understanding and cognition, similar to taking an average score from multiple quiz scores.

To track a student's learning progression, analyzing the individual student created artifacts (demonstrated skill score) does not appear to be effective. The below demonstrated skill score graphs (Figure 37 and 38) show that students' demonstrated skill scores are not highly related over time. This result is not unique for AgentSheets and the Scalable Game Design project. A recent study using Scratch shows similar results (Scaffidi et al., 2010). Each specific game/simulation requires different sets of programming skills, and the use of a specific programming skill is not necessarily increased over time for the implementation of different learning target projects. This is especially true when students have choices for building their own projects. This tendency becomes more obvious as students use their personal skill set more frequently. Figure 38 shows the average demonstrated skill scores in a college class, when students were asked to build any game/simulation with their programming preferences. The last game shows the lowest demonstrated skill score, but this does not mean that the students' skills decreased. Students chose the most practiced strategies and skills to build their final projects so they didn't need to include other unnecessary CT patterns. Moreover, each CT pattern has a different difficulty level of implementation, so a lower demonstrated skill score or a smaller proportion of CT patterns in a project does not necessarily mean a decrease in student programming skills.

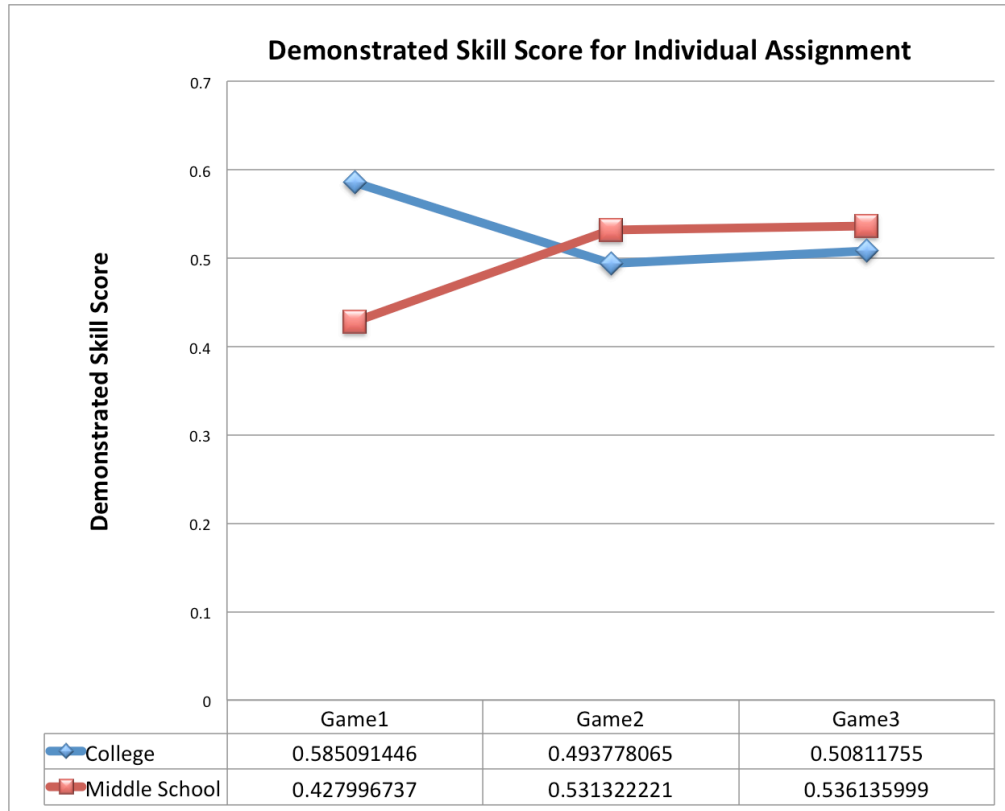


Figure 37 1st to 3rd Games in A Middle School and in A College Class

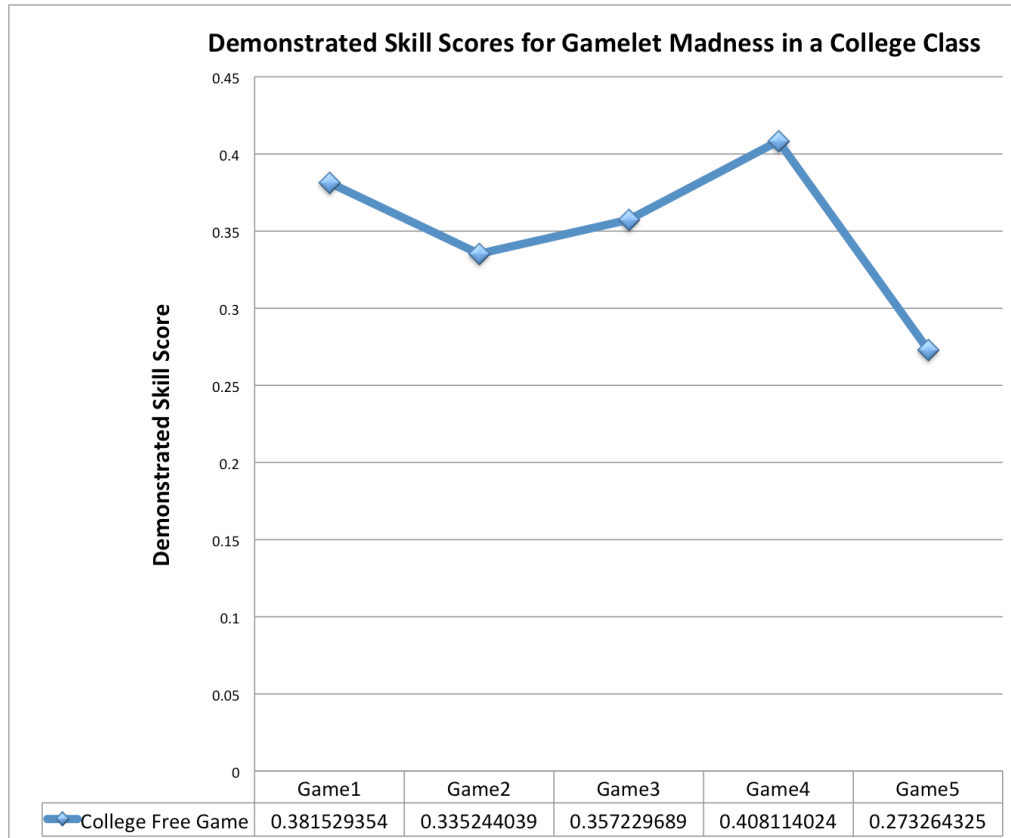


Figure 38 5 Gamelet Madness Projects in A College Class

6.2.1. Findings: Middle School Vs. College Student Comparison

I tested 268 AgentSheets games and simulations from 30 college students and 73 games from 33 middle school students. From these AgentSheets projects, I calculated the average learning skill scores of each game/simulation and computed learning progressions using comprehensive skill score for each game/simulation for the entire class in one academic semester. Figure 39 shows these learning progressions and the comparison between them.

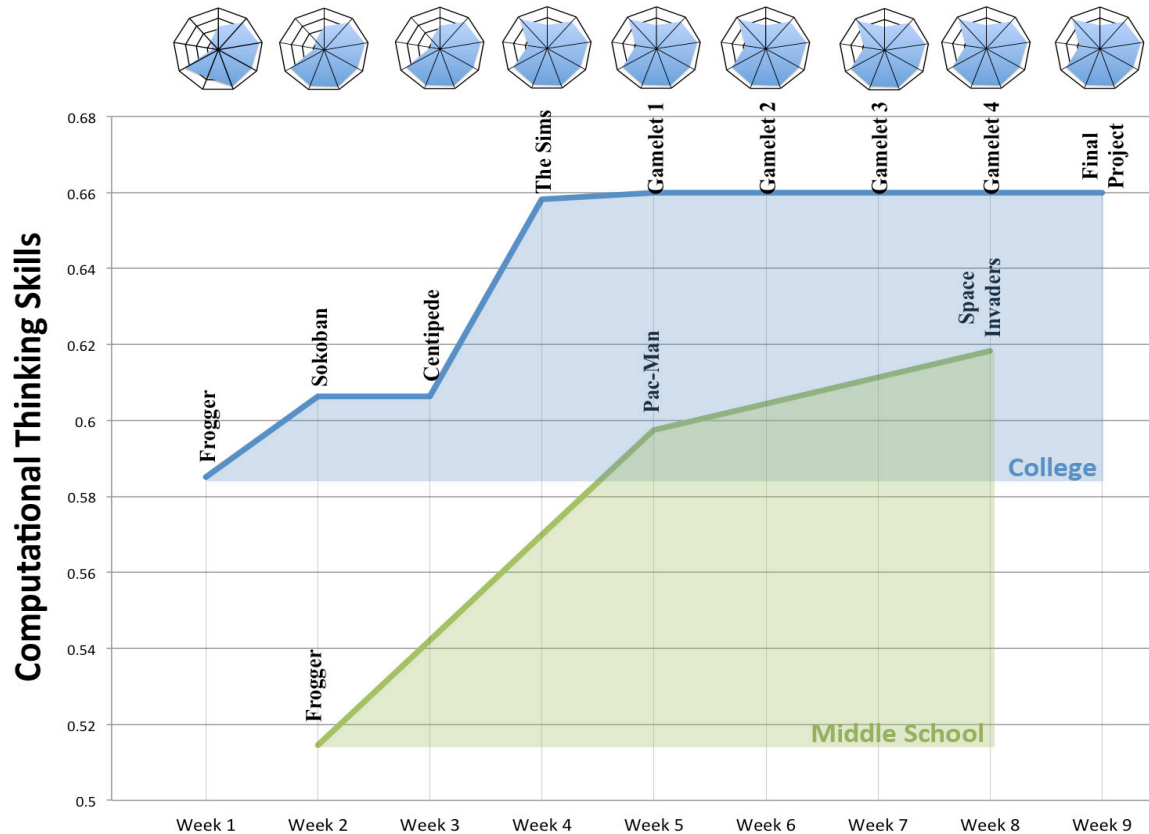


Figure 39 Skill Progression Comparison between College and Middle School Students with Comprehensive Skill Scores

Students from the middle school (MS) class units (usually 2-4 weeks long) and the semester-long college class increased their knowledge and understanding of programming and game design. These improvements are reflected in their calculated learning skill scores over time. Both MS and college students are expected to learn each new game's computational thinking patterns. Therefore, learning skill scores tend to increase with newly learned games. Students' learning skill score progressions are illustrated in Figure 39.

Figure 39 shows all classes gradually increasing their learning skill scores over time by making games. The high increases in the college students' learning skill scores

between Centipede and Sims can be explained by the higher-level complexity required to create the Sims game. Both Sims and Pacman required two main computational thinking patterns, diffusion and hill climbing. These patterns are two of the most difficult of the nine computational thinking patterns that must be learned to successfully complete these assignments. In contrast, college class learning skill scores barely increase between Sokoban and Centipede. Sokoban and Centipede share the same computational thinking patterns, which explains why students' learning progression appears steady. The college students were required to learn all nine computational thinking patterns as they progressed from Frogger to The Sims, and then used their previously learned skills to design their own unique games. Since no new skills were required, previously learned skills were simply practiced for retention.

As Figure 39 illustrates, the two classes overlap through time. Learning skill scores are quite different for each class. For example, the game of Frogger as created by college students had a learning skill score value of 0.59, while the same game created by middle students had a value of 0.52. This gap between their learning skill scores can be explained by the completeness of the different Frogger implementations. College students completed Frogger entirely, but more than half of the middle school students failed to upload a complete Frogger game (i.e. missing transportation or some other patterns). The same completeness factor holds for the uploaded Sokoban, Pacman and Space Invaders games.

Nevertheless, middle school students reached the entry level learning skills of the college class. After their second game submission the learning skill score of the middle school students reached 0.6; the college class entry-level score was 0.59.

6.2.2. Summary of Assessing Student Products Using the CTPA

I believe that knowledge acquisition or transfer of educational content over time has been shown through Computational Thinking Pattern Analysis within the Scalable Game Design class curriculum. These skill progressions also revealed some interesting differences between middle school students and the college class.

For instance, the time the middle school students used to move from one game to the next was over three times as long as that for the college students (Figure 39). Since the game learning progressions and sequence between the groups are very similar, the learning timeframe would appear to be a significant difference between the two groups. However, the game curriculum imparted enough programming skill and comprehension to allow the middle school students to complete their design submission goal at a similar level as the college students despite substantial differences in educational level and age—the middle school students simply took longer. The point is not that it takes middle school students longer to accomplish the design of a similar video game as college students, but that it is possible to track students' skill progressions over time with demonstrated and comprehensive skill scores using CTPA. This comparison begins to shed some important light on previous assumptions about the value and scope of enjoyable visual language curriculum lessons, especially in areas linked to computer science education. Using the previously mentioned Flow model (Csikszentmihalyi, 1990), I could hypothesize that this game design curriculum appears to balance the enjoyment factor with an increasing skill level, thereby promoting increased learning across widely spaced ages and grades. Facilitating Flow in middle schools could therefore help increase positive attitudes towards computer science and programming in middle school students.

In addition, CTPA's ability to compute learning progressions by measuring computational thinking pattern knowledge acquisition over time is a substantial step forward in showing the worth of game/simulation-programming courses and their associated visual language.

6.3. Divergence in Programming

Divergence in programming could be assessed through the evaluation of the differing approaches each student employed within the specified design parameters. In other words, when faced with a difficult challenge for agent behavior design, each student defined that programming challenge in a way that specified an accurate programming solution (Bennett et al., 2013). From this I devised the divergence calculation which was introduced in the chapter 5.1 to demonstrate student-programming creativity as a divergence calculation from the "norm," or the SGD online tutorial standard (Bennett et al., 2013). The difference between a student's game and the tutorial can be detected by visual inspection using the CTPA graphs. Through this divergence calculation, this difference can be mathematically calculated.

6.3.1. Three Class Conditions

Sheryle (pseudonym), the teacher selected for this study, taught three unique class conditions. Initially she taught the project class based on the SGD online tutorial. Subsequently, she designed her own tutorial for her regular in-class students and then transferred her tutorial adaptation to an online version of the project class. This offered a rare opportunity to compare three unique class conditions without having to consider teacher influence as a random variable. In all three classes, which were taught by one

teacher, Sheryle, the uploaded student games were noticeably divergent from the SGD tutorial “norm.”

Common factors identified from all three class conditions taught by Sheryle are as follows:

- Sheryle is the teacher of record for all classes
- She alone helps the students complete their games
- She followed the project curriculum content parameters
- Frogger is the first project game taught to all classes
- Frogger is the only uploaded game analyzed

6.3.2. Findings

For this investigation I chose three unique class conditions that were taught by a single teacher. This allowed us to keep the experimental focus on the divergence calculation using Equation 2 as an indication of creativity, as opposed to teacher influence. Findings show a marked difference between the three class conditions in regards to programming divergence, as represented in Figure 40.

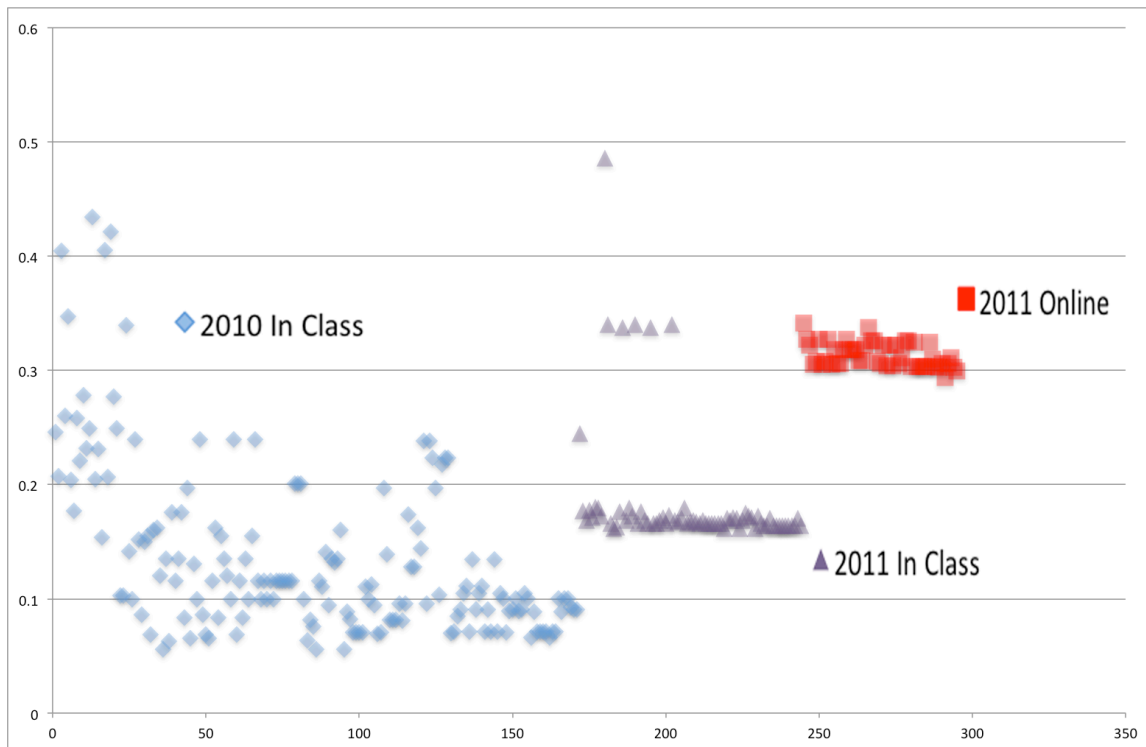


Figure 40 Scattered Divergence Calculation Graph

The graph in Figure 40, the X-axis represents time by order of submissions. The Y-axis represents the Divergence Score. Each dot means an individual submission. 296 Frogger games are displayed.

6.3.3. Divergence Calculation Graph

Figure 40 uses the divergence calculation (Bennett et al., 2013) to depict the collected data of all Frogger games during academic years 2010-2011 from Sheryle's three class conditions. Each individual student-submitted game is placed on the graph according to the calculated divergence in programming. The three distinguishable clusters accurately represent the three distinct class conditions. The left cluster (blue) displays a sparser more scattered pattern than the middle (purple) and right (red) clusters.

This graph represents 296 Frogger games.

For the 2010 In-Class condition, the students' Frogger games were highly divergent from each other, but the game plots started to converge at the beginning of the 2011 classes. Coincidentally, the first game submission of the 2011 classes occurred when Sheryle started to use her own tutorial in place of the official SGD online tutorial. It appears that although her teaching style was unchanged, her presentation of the material had evolved in some fashion.

I also calculated the class standard deviation, as well as the class divergence average. Those are displayed in Table 6 (below). The “2010 In-class” condition (blue in Figure 40) with the widest pattern spread also shows the largest standard deviation within the class, and the “2011 Online class” (red in Figure 40) with the narrowest spread has the smallest standard deviation within the class. This means that the games in the “2010 In-class” condition are more divergent from each other than the games in “2011 Online” class condition.

Divergence Score	Standard Deviation	Average
In Class 2010	0.074	0.135
In Class 2011	0.057	0.186
Online Class 2011	0.011	0.314

Table 7: Divergence Calculation Score in Each Class

I conjecture that not only is the revised tutorial a significant factor in the represented divergence between class conditions, but that the in-class/online condition

comparison is a significant factor in the divergence calculation for at least two class conditions.

6.3.4. Summary of Assessing Student Products Using the CTPA

I have developed a mathematical measure of divergence for game programming that calculates the difference of each programming sequence of a submitted game from the game tutorial “norm.” The main difference between my measurement of divergence and the traditional creativity measurement methods is the use of a mathematical calculation in place of the subjective appraisal by a trained rater. The “norm” for divergence tests is usually a predetermined standard solution, similar to the SGD online tutorial. Since the SGD tutorial is commonly used by most project teachers, it is the “norm” for the SGD curriculum or standard by which the teachers judge or grade their students’ work.

Although the same teacher conducted the three class conditions discussed here, operationally eliminating teacher influence as a mitigating variable in the divergence calculation, I recognize that a teacher rarely teaches multiple classes that are exactly the same. However, I believe that I have documented compelling support for the initial validation of the divergence calculation as a measurement of programming divergence. Examination of the Scattered Divergence Calculation graph (Figure 40) reveals three obviously separate and distinct clusters that represent these three class conditions mentioned in Table 6. Significantly, figure 40 shows each of the three separate class-learning conditions generating a unique divergence pattern. Since each of the separate

conditions is unique, this first validity test shows that divergence analysis can identify uniqueness in student game programming environments.

These divergence calculations are supported by other data sources, such as the teacher's unsolicited comments about her students' creativity and my observations of student enthusiasm and creativity in physical classroom settings (Bennett et al., 2011; Bennett et al., 2013). As a fail-safe, I also manually inspected the programming code supporting the divergence calculation within a random sample of uploaded games.

6.4. ONline Assessment of Computational Thinking (ONACT)

ONACT is an embedded, formative, real-time graphical assessment tool that quickly gives teachers insight into student mastery of computational thinking constructs as they create games and simulations. ONACT provides teachers with a useful representation of class and individual progress, allowing them to make effective instructional decisions. The ONACT system breaks down all collectable student project information and records it in the ONACT database. ONACT analyzes the student project information stored in this database through Computational Thinking Pattern Analysis in real time. This analysis extracts semantic meaning out of the code by interpreting which Computational Thinking Patterns have been implemented by students. The analyzed data are illustrated through three different levels of visualization: Computational Thinking Pattern Analysis Graph (Koh et al., 2010), Assessment Dashboard (Figure 41), and Computational Thinking Pattern Analysis Forensics (Figure 43).

The Assessment Dashboard indicates to teachers where students are in their programming tasks. The dashboard visualizes the programming progression for each

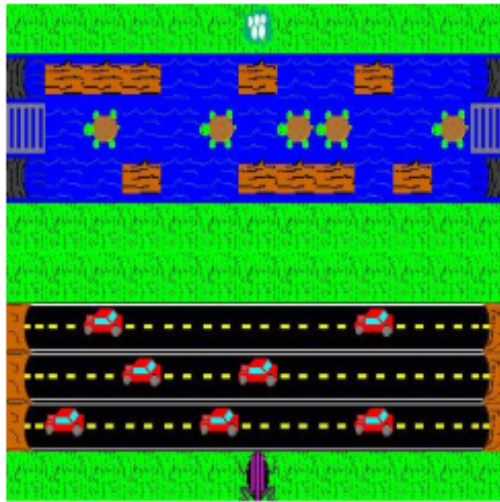
student in the class through CTPA (Koh et al., 2010; Bennett et al., 2011; Bennett et al., 2013). In Figure 41, green indicates students who are completing the program correctly, orange indicates students who may need some help with their program, and red indicates students who are in significant need of scaffolding. The Dashboard clearly shows students who might be in trouble. By selecting a specific student in the Dashboard, a teacher is directed to an individual game submission page (Figure 42) where the teacher can see in-depth representations of that student's progression in their Computational Thinking Pattern Analysis Graph (Koh et al., 2010; Bennett et al., 2011; Bennett et al., 2013) and Computational Thinking Pattern Analysis Forensics (Figure 43). The Computational Thinking Pattern Analysis Forensics graph can be reached by clicking the Computational Thinking Pattern Analysis Graph in the individual game submission page (left bottom in Figure 42).

The Computational Thinking Pattern Analysis Forensics graph explains how a student has progressed in his/her computational thinking pattern implementations by programming with AgentSheets. Each dot in the CTPA Forensics graph indicates a value of a certain computational thinking pattern in each step when any single condition or action is added to the AgentSheets project. If a student has followed the tutorial or the teacher's instruction, then the student's CTPA Forensics graph will be same as the CTPA Forensics graph of the tutorial or the teacher's project.

University of Colorado Boulder
 2011 CSCI7000
 Sokoban



Figure 41 Example of ONACT Assessment Dashboard showing every student’s performance in a given classroom.



FroggerMcKeevek

Classic Frogger! Get Frogger home to his/her eggs!

[Run](#)
[Download](#)

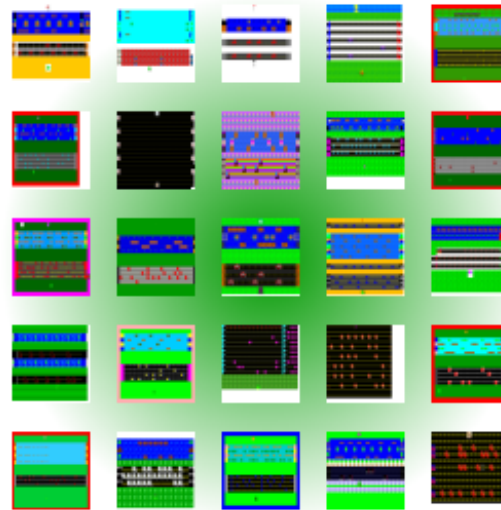
Similarity Score to Four Tutorial Games

This score shows how much your game structure is similar to the tutorial games. Max value is 1

- This game's similarity score to Frogger:0.819
- This game's similarity score to Sokoban:0.623
- This game's similarity score to Space Invaders:0.628
- This game's similarity score to Sims:0.141

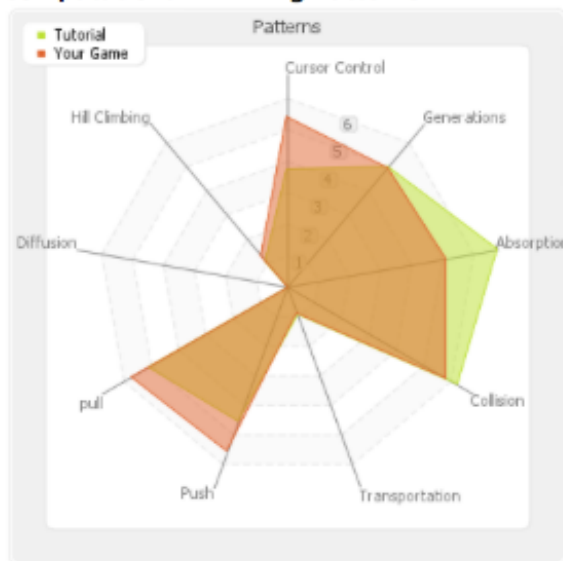
Similarity Score Matrix

Below Matrix shows other AgentSheets projects sharing similar programming structure.



Submission Time
11/01/25 : 12:57:27

Computational Thinking Patterns



Rating: **2.0/5**
(2 votes cast)



Figure 42 Individual Game Submission Page of Scalable Game Design Arcade

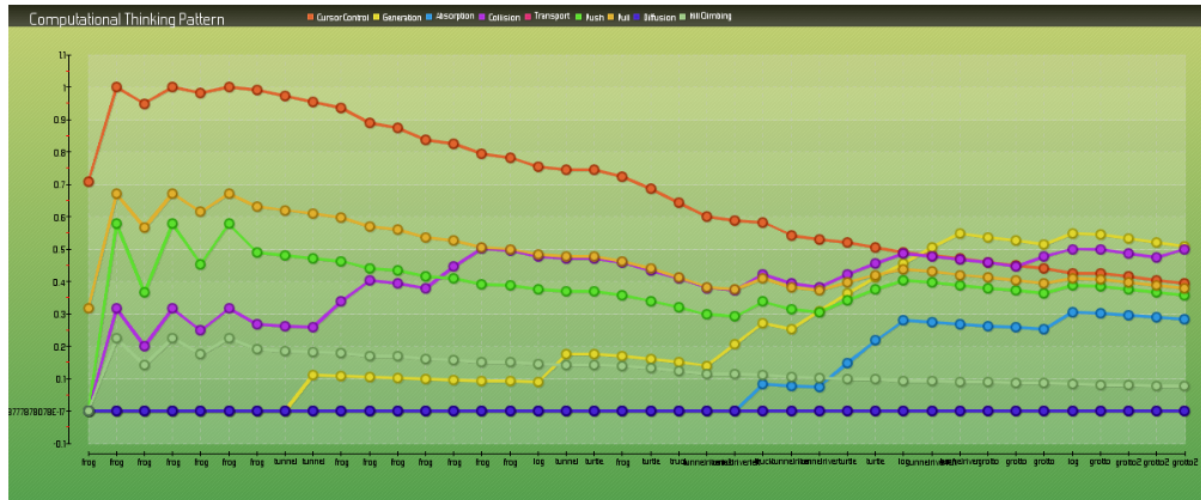


Figure 43 The Computational Thinking Pattern Analysis Forensics graph

6.4.1. Summary of Assessing Student Performance Using ONACT

Semantic analysis of computational thinking in visual programming learning promises better individual feedback and faster learning assessment to students and teachers. Additionally, this kind of feedback can be used to determine when and how teachers can expand students' learning capability in accordance with two theories: First, the Zone of Proximal Development proposed by Lev Vygotsky (Vygotsky, 1978), which describes the difference between what a learner can do without help and what he or she can do with help; and secondly, Flow proposed by Mihaly Csikszentmihalyi (Csikszentmihalyi, 1990), which means a completely motivated and engaged state that is derived by measuring student skills and challenges. A validated CTPA will contribute to the study of learning theory, professional development, and educational data mining by providing empirical data in order to make it possible to refine the current conceptual framework of educational systems.

The CTPA can measure educational benefits of visual programming learning that we couldn't quantify before. Each visual programming artifact can be computed and measured semantically in terms of computational thinking. So far I have explored CTPA primarily with AgentSheets and AgentCubes. However, it could be applied to other visual programming language artifacts and/or other research domains. When CTPA is applied to other visual programming languages or other research domains, this research would suggest a valuable and effective method that can assess students' learning performance, provide effective learning guidelines, and compute students' learning outcomes. This can be used to create real cyberlearning systems that help large numbers of teachers and students learn computational thinking.

Chapter 7. Discussion

The Computational Thinking Pattern Analysis has shown many promising results in providing valuable educational feedback, but also several limitations with its current approach. I discuss three limitations of the current CTPA that I found during my PhD study in this chapter.

First, the identified Computational Thinking Patterns in the current CTPA are limited. Currently, the CTPA covers only nine CT patterns. Those nine patterns are the most popular CT patterns in game and science simulation designs (Koh et al., 2010; Basawapatna et al., 2011), and they are the first patterns that students and teachers will learn during the SGD curriculum. However, through the SGD research, we have identified additional CT patterns in game and simulation designs, and the current CTPA cannot detect those new CT patterns. Also, the current CTPA cannot detect possible new CT patterns if they are constructed with non-identified CTPA constructors. This is the challenge that the current CTPA should overcome.

Second, there is a limitation which comes from the false positives and the false negatives. This limitation can explain the outliers in Chapter 6.2. For example, there are two outliers (the red and green data points) in the below graph (Figure 44). The red data point, Game 1, is ranked third in the human grading score rank and nineteenth in the Demonstrated Skill Score rank while the green data point, Game 2, is fourteenth in the human grading score rank and third in the Demonstrated Skill Score rank.

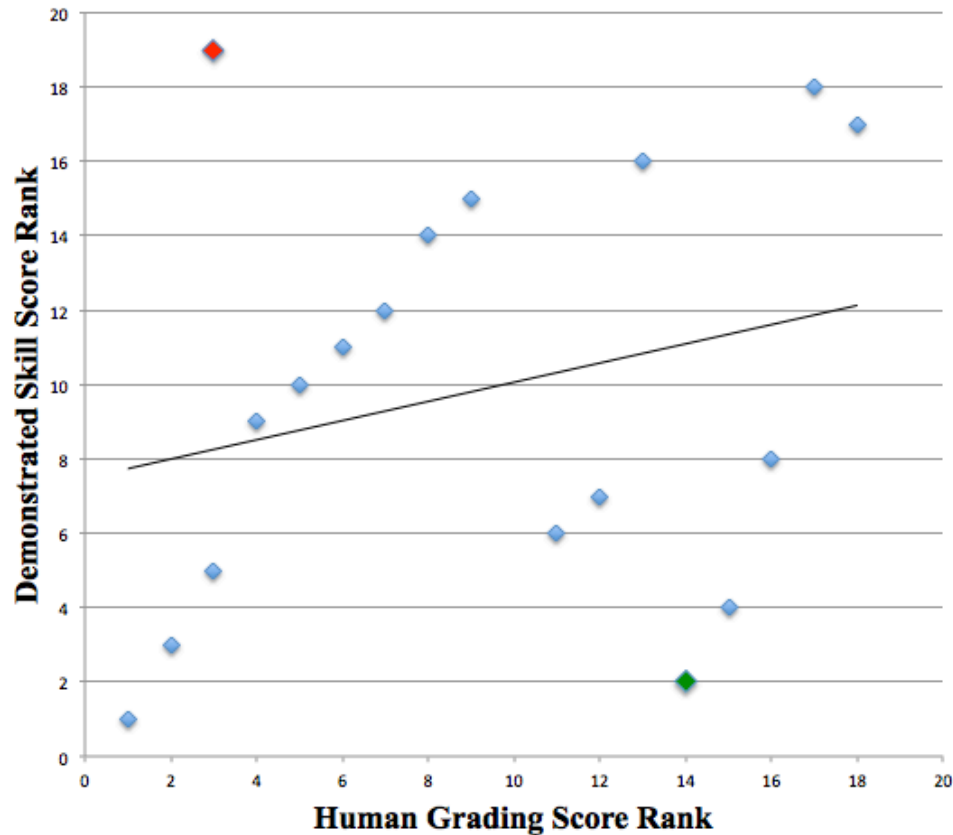


Figure 44 Frogger Submissions in the 2012 class

Both games are similar to each other in their game play, but they are different from each other in game design and programming. Figures 45 and 46 show the design layouts of two games, and Figures 47 and 48 depict the behavior programming of the car agent in Game 1 and the truck agent in Game 2. The car and the truck agents play the same role in this game of Frogger.

In Figure 45, Game 1 does not have any generator or absorber for the car, the turtle, and the log agents, but they are generated and absorbed at the edges of the game layout. The correct methods of the Generation and the Absorption patterns create the Generator and Absorber agents and use the See condition for the two CT patterns, but Game 1 uses the Stack condition which is not a CTPA constructor of the current CTPA.

This non-CTPA constructor usage decreased the Generation and the Absorption patterns' CTPA values, and it lowered Game 1's Demonstrated Skill Score value. This limitation is called a false negative, which indicates that the CTPA cannot detect the existing CT pattern or lower the CTPA value of the existing CT pattern in a given game or simulation. In contrast, a false positive means that the CTPA shows a CTPA value of a non-existing CT pattern due to the use of CTPA constructors in other CT pattern implementations in a given game or simulation. The values of the Push and the Pull patterns in a CTPA graph of Frogger games are examples of the false positives (See Appendix B).

Unlike Game 1, Game 2 followed the correct methods for its Generation and Absorption pattern implementations. Except for its poorly designed game layout, Game 2 is very similar to the SGD Frogger tutorial. However, a game similar to the tutorial does not guarantee a good score when a grader grades it. So Game 2 is ranked high in the Demonstrated Skill Score rank, but it is ranked low in the human grading score rank.

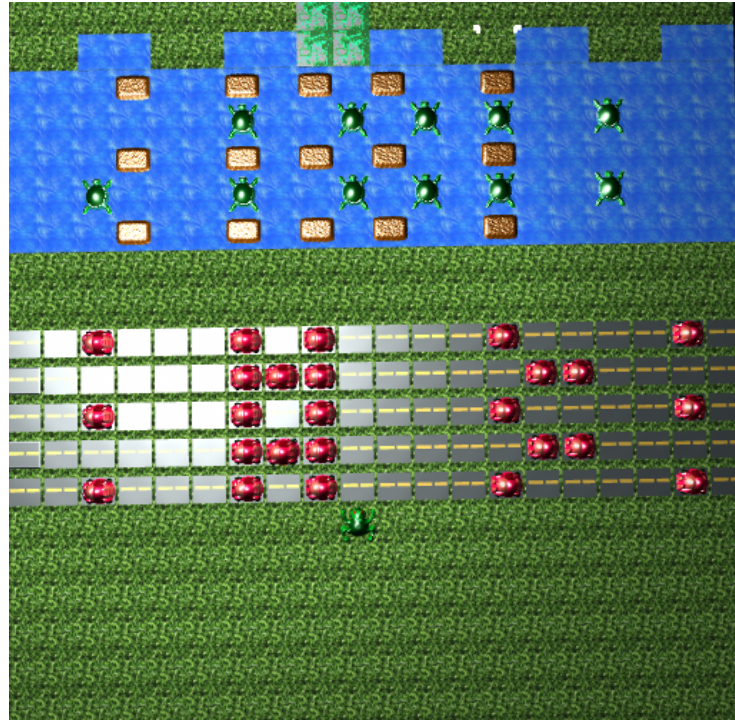


Figure 45 Frogger Design Layout of Game 1

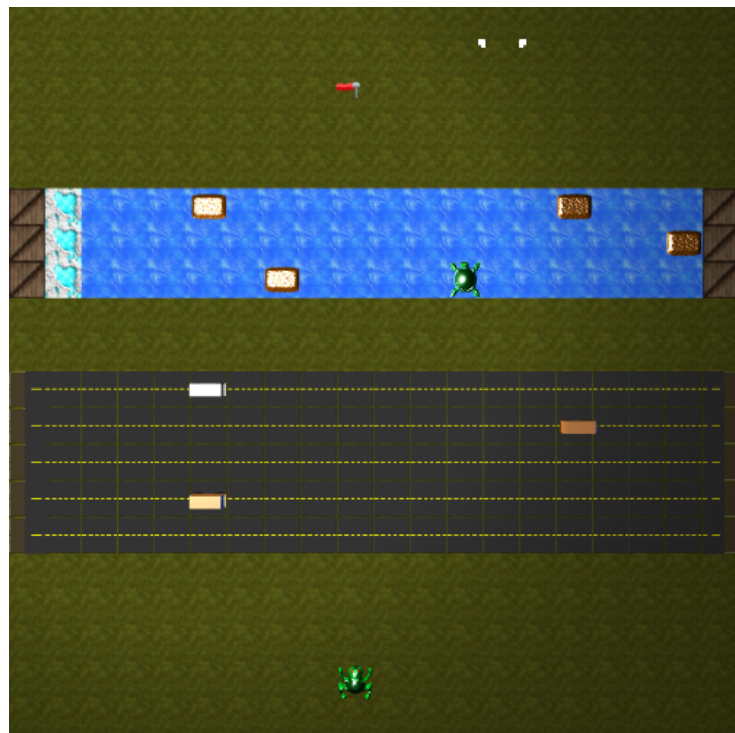


Figure 46 Frogger Design Layout of Game 2



Figure 47 Behavior Programming of Game 1

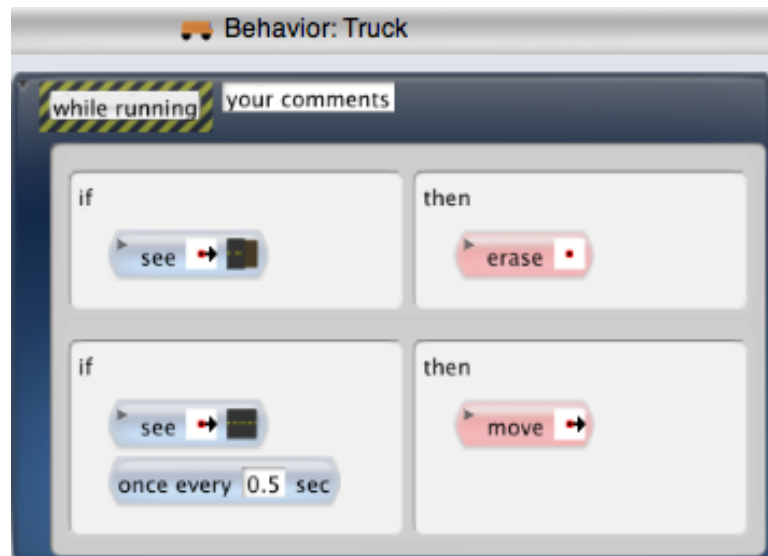


Figure 48 Behavior Programming of Game 2

Third, the CTPA cannot detect the difference between the correct method and the incorrect method in the CT pattern implementation with the same CTPA constructors.

This limitation is covered in Chapter 6. 3.

Chapter 8. Conclusion

Many previous attempts in CT learning assessment have been limited in nature to skill investigations mostly at the syntactic level (Koh et al., 2010; Lewis 2010). At the syntactic level, these attempts are extremely difficult because visual and textual languages may not match up very well. Also, students usually do not get any individual feedback to know what kinds of knowledge they have actually learned through making programmed artifacts.

The CTPA suggests a new notion of learning assessment in computer science and computational thinking education. Teachers can assess student-learning performance at an individual level, a whole class level, and even a school level. Teachers will be able to know which students struggled to follow a class lesson and which students moved ahead. Students will get direct and instant feedback from CTPA to understand what they have learned through making games and simulations and how their learning skills have evolved over time.

Semantic analysis of computational thinking in visual programming learning promises better individual feedback and faster learning assessment to students and teachers. Additionally, this kind of feedback can be used to determine when and how teachers can expand students' learning capability in accordance with the theory of the Zones of Proximal Flow (Basawapatna et al., 2013). A validated CTPA will contribute to the study of learning theory, professional development, and educational data mining by providing empirical data in order to make it possible to refine the current conceptual framework of educational systems.

Despite certain limitations of the current CTPA, the validation results with concurrent and predictive validities for the CTPA are promising, though further exploration with a larger data set is warranted. Beyond demonstrating that CTPA and human grader performance are well correlated when assessing foundational games, I showed the predictive value of this analysis tool for assessing students' skill in designing their own games. I anticipate that it will be possible to use the CTPA in the future to provide trustworthy educational feedback, especially given the consistency of the findings using data from two consecutive years.

The CTPA can measure educational benefits of visual programming learning that we couldn't quantify before. Each visual programming artifact can be computed and measured semantically in terms of computational thinking. So far I have explored the CTPA with primarily AgentSheets and AgentCubes. However, it could be applied to other visual programming language artifacts and/or other research domains. When the CTPA is applied to other visual programming languages or other research domains, this research will potentially suggest a valuable and effective method that can assess students' learning performance, provide effective learning guidelines, and compute students' learning outcomes. This can be used to create real cyberlearning systems that help large numbers of teachers and students learn computational thinking.

Chapter 9. References

- Argotea, L., Ingramb, P., Knowledge Transfer: A Basis for Competitive Advantage in Firms, *Organizational Behavior and Human Decision Processes*, Volume 82, Issue 1, May 2000, Pages 150–169
- Basawapatna, A., Koh, K.H., Repenning, A., Using Scalable Game Design to Teach Computer Science From Middle School to Graduate Schools, In *Proc. ITICSE '10 Annual Conference on Innovation and Technology in Computer Science Education*, Ankara, Turkey, 2010.
- Basawapatna, A., Koh, K. H., Repenning, A., Webb, D., Marshall, K., Recognizing Computational Thinking Patterns, In *Proceedings of ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2011)*, Dallas, Texas, USA, March 9-12, 2011.
- Basawapatna, A., Repenning, A., Koh, K. H., Nickerson, H., The Zones of Proximal Flow: Guiding Students through a Space of Computational Thinking Skills and Challenges, *ICER '13: International Computing Education Research Conference*, San Diego, California, USA, August 12- 14, 2013.
- Basawapatna, A., Repenning, A., Koh, K. H., Saviganano, M., The Consume - Create Spectrum: Balancing Convenience and Computational Thinking in STEM Learning, In *Proceedings of ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2014)*, Atlanta, Georgia, USA, March 5-8, 2014.
- Bennett, V., Koh, K. H., Repenning, A., CS Education Re-Kindles Creativity in Public Schools, In *Proceedings of ITiCSE '11: Annual Conference on Innovation and Technology in Computer Science Education*, Darmstadt, Germany, June 27-29, 2011.
- Bennett, V., Koh, K. H., Repenning, A., Can Learning Acquisition be Computed?, In *Proceedings of IEEE International Symposium on Visual Languages and Human-Centric Computing 2011*, Pittsburgh, PA, USA, September 18-22, 2011.
- Bennett, V., Koh, K. H., Repenning, A., Computing Creativity: Divergence in Computational Thinking, In *Proceedings of ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2013)*, Denver, Colorado, USA, March 6-9, 2013.

- Bransford, J., Brown, A., and Cocking, R., How People Learn, *National Academy Press*, 1999
- Csikszentmihalyi, M. Flow: The Psychology of Optimal Experience. Harper Collins Publishers, New York, 1990.
- Davis, R., Kafai, Y., Vasudevan, V., and Lee. E., 2013. The education arcade: crafting, remixing, and playing with controllers for Scratch games. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 439-442. DOI=10.1145/2485760.2485846 <http://doi.acm.org/10.1145/2485760.2485846>
- Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., Basawapatna, A., Computational Thinking Patterns, In *Proceedings of the 2011 Annual Meeting of the American Educational Research Association (AERA)*, New Orleans, Louisiana, USA, April 8-12, 2011.
- Kelleher, C., Pausch, R., and Kiesler. S., 2007. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 1455-1464. DOI=10.1145/1240624.1240844 <http://doi.acm.org/10.1145/1240624.1240844>
- Kelleher, C., and Pausch, R., 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2 (June 2005), 83-137. DOI=10.1145/1089733.1089734 <http://doi.acm.org/10.1145/1089733.1089734>
- Koh, K. H., Bennett, V., Repenning, A., Inspiring Collaborative Benefits: An Interaction between a Virtual and a Physical Group Learning Infrastructure, In *Proceedings of Western Canadian Conference on Computing Education (WCCCE 2010)*, Okanagan, B.C., Canada, May 7-8, 2010.
- Koh, K. H., Basawapatna, A., Bennett, V., Repenning, A., Towards the Automatic Recognition of Computational Thinking, *IEEE International Symposium on Visual Languages and Human-Centric Computing*, Leganés-Madrid, Spain, September 21-25, 2010.
- Koh, K. H., Bennett, V., Repenning, A., Computing Indicators of Creativity, In *Proceedings of ACM Creativity & Cognition 2011*, The High Museum of Art · Atlanta, Georgia, USA, November 3-6, 2011.

- Koh, K. H., Basawapatna, A., Nickerson, H., Repenning, A., Real Time Assessment of Computational Thinking, In *Proceedings of IEEE International Symposium on Visual Languages and Human-Centric Computing*, Melbourne, Australia, July 28-Aug 1, 2014
- Koh, K. H., Nickerson, H., Basawapatna, A., Repenning, A., Early Validation of Computational Thinking Pattern Analysis, In *Proceedings of ITiCSE '14: Annual Conference on Innovation and Technology in Computer Science Education*, Uppsala, Sweden, June 23-25, 2014
- Landauer, T. K., 2003. Pasteur's quadrant, computational linguistics, LSA, education. In *Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing - Volume 2 (HLT-NAACL-EDUC '03)*, Vol. 2. Association for Computational Linguistics, Stroudsburg, PA, USA, 46-52. DOI=10.3115/1118894.1118901
<http://dx.doi.org/10.3115/1118894.1118901>
- Lewis, C. M., How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch, *Proc. SIGCSE '10*, ACM Press, WI, USA, 2010.
- Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., and Rusk. N., 2008. Programming by choice: urban youth learning programming with scratch. *SIGCSE Bull.* 40, 1 (March 2008), 367-371. DOI=10.1145/1352322.1352260
<http://doi.acm.org/10.1145/1352322.1352260>
- Michotte, A. (1963). *The Perception of Causality* (T. R. Miles, Trans.). London: Methuen & Co. Ltd.
- Monroy-Hernández, A., Designing a website for creative learning. In *Proc. the WebSci'09: Society On-Line*, March 18-20, 2009, Athens, Greece
- Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Velázquez-Iturbide. J. A., 2002. Exploring the role of visualization and engagement in computer science education. In *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '02)*. ACM, New York, NY, USA, 131-152. DOI=10.1145/782941.782998
<http://doi.acm.org/10.1145/782941.782998>

- National Research Council, Report Of A Workshop On The Scope And Nature Of Computational Thinking, *The National Academies Press*, Washington, D.C., 2010.
- Peppler, K. & Kafai, Y. B., Collaboration, Computation, and Creativity: Media Arts Practices in Urban Youth Culture. In C. Hmelo- Silver & A. O'Donnell (Eds.), In *Proc. Computer Supported Collaborative Learning*, New Brunswick, NJ, USA, 2007.
- Perrone C., and Repenning. A., 1995. Remote exploratoriums: combining network media with design environments. In *Conference Companion on Human Factors in Computing Systems (CHI '95)*, I. Katz, R. Mack, and L. Marks (Eds.). ACM, New York, NY, USA, 117-118. DOI=10.1145/223355.223458 <http://doi.acm.org/10.1145/223355.223458>
- Repenning, A., and Ambach, J, The agentsheets behavior exchange: supporting social behavior processing, In *Proc. CHI '97 extended abstracts on Human factors in computing systems: looking to the future*, ACM, 1997, pp 26-27
- Repenning A. and Perrone. C., 2000. Programming by example: programming by analogous examples. *Commun. ACM* 43, 3 (March 2000), 90-97. DOI=10.1145/330534.330546 <http://doi.acm.org/10.1145/330534.330546>
- Repenning, A., Basawapatna, A. and Klymkowsky, M. Making Educational Games That Work in The Classroom. In *Proceedings of the IEEE Games Innovation conference (IGIC 2013)*, Vancouver, British Columbia, Canada, September 23-25, 2013.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai. Y., 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (November 2009), 60-67. DOI=10.1145/1592761.1592779 <http://doi.acm.org/10.1145/1592761.1592779>
- Scaffidi C., and Chambers C., 2010, Skill progression demonstrated by users in the Scratch animation environment. *International Journal of Human-Computer Interaction*, 28:6, 383-398.
- Sturtevant, N. R., Hoover, H. J., Schaeffer, J., Gouglas, S., Bowling, M. H., Southey, F., Bouchard, M., and Zabaneh, G. 2008. Multidisciplinary students and instructors: a second-year games course. In *proc 39th SIGCSE Technical Symposium on Computer Science Education*, Portland, OR, USA, 2008.

Squire, K., Video games in education. *International Journal of Intelligent Simulations and Gaming*, (2) 1. 2003

Vgotsky, L.S, *Mind in Society: Development of Higher Psychological Processes*, Harvard University Press, 1978

Walter, S. E., Forssell, K., Barron, B., and Martin. C., 2007. Continuing motivation for game design. In *CHI '07 Extended Abstracts on Human Factors in Computing Systems (CHI EA '07)*. ACM, New York, NY, USA, 2735-2740.
DOI=10.1145/1240866.1241071 <http://doi.acm.org/10.1145/1240866.1241071>

Wing. J., 2009. Computational thinking. *J. Comput. Small Coll.* 24, 6 (June 2009), 6-7.

Yang, Y., Buckendahl, C., Juskiewicz, P., & Bhola, D. (2002). A review of strategies for validating computer-automated scoring. *Applied Measurement in Education*, 15(4), 391-412

Z. Les., and M. Les., Thinking, Visual Thinking, and Shape Understanding, *Studies in Computational Intelligence*, 86, 2008, pp. 1-45.

Appendix A. CTPA Graphs for each CT patterns

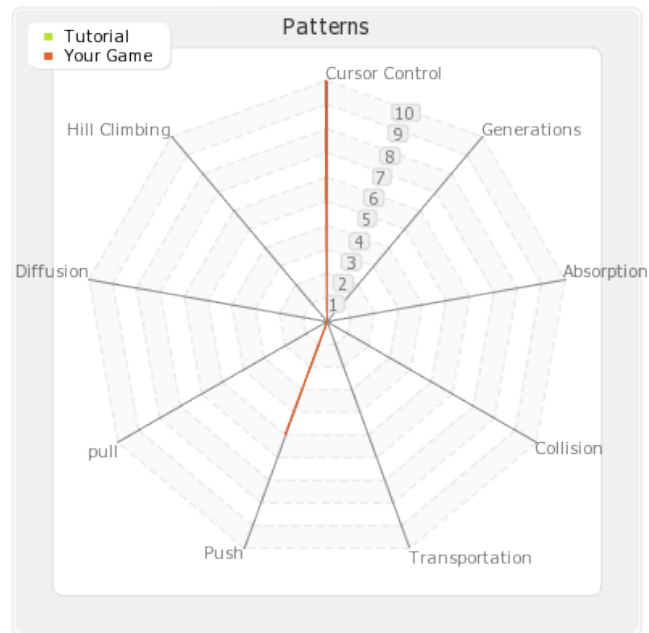


Figure 49 Cursor Control Pattern

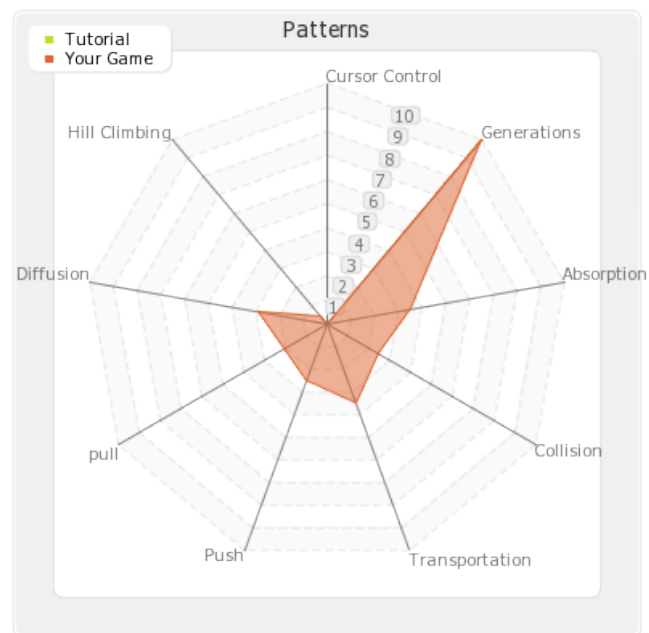


Figure 50 Generation Pattern

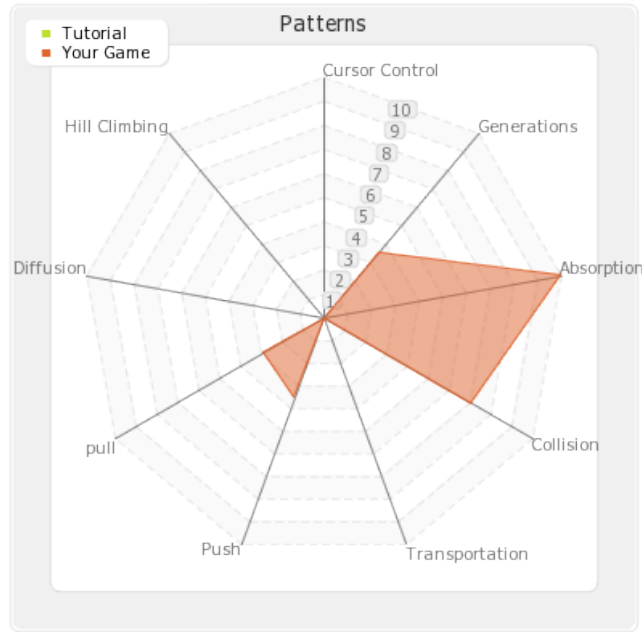


Figure 51 Absorption Pattern

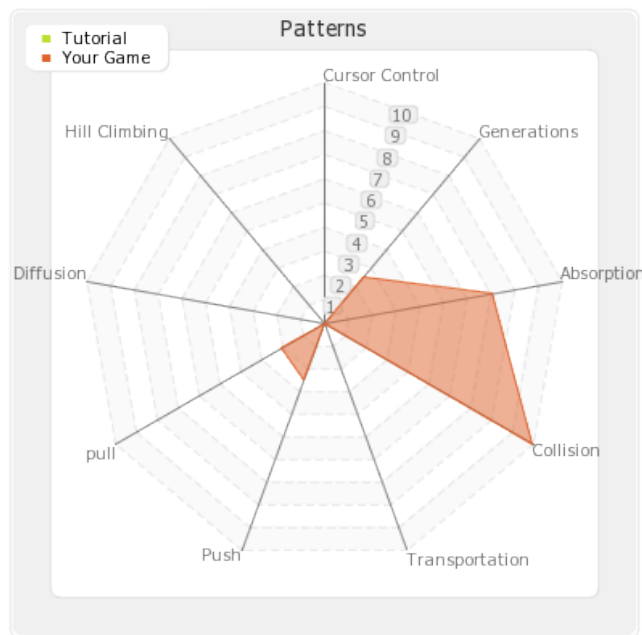


Figure 52 Collision Pattern

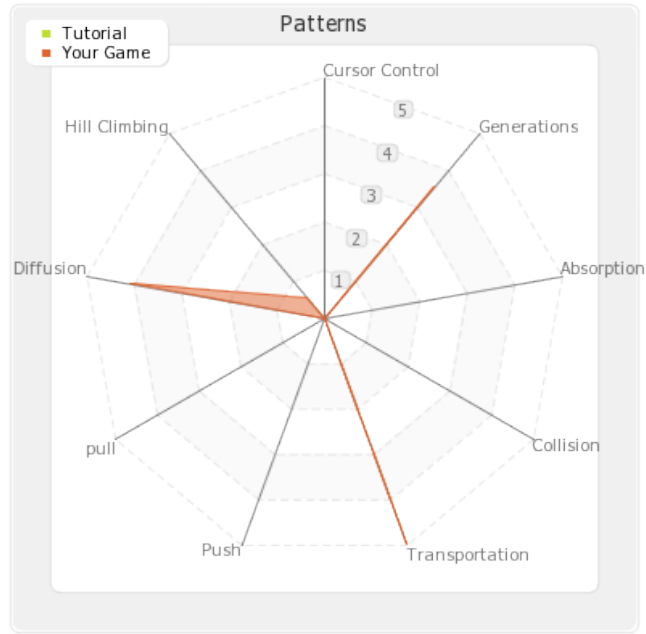


Figure 53 Transportation Pattern

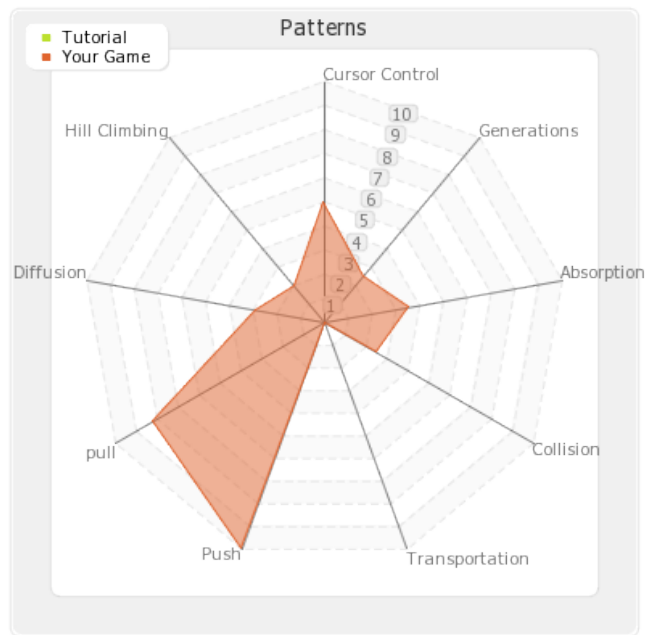


Figure 54 Push Pattern

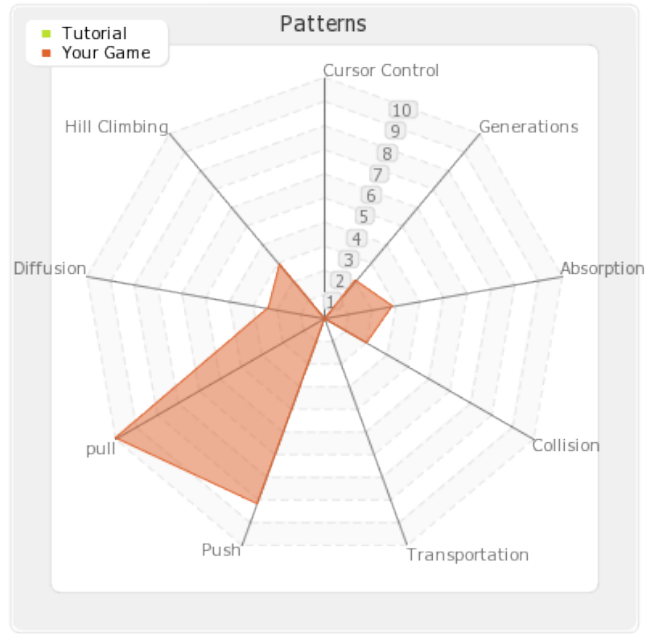


Figure 55 Pull Pattern

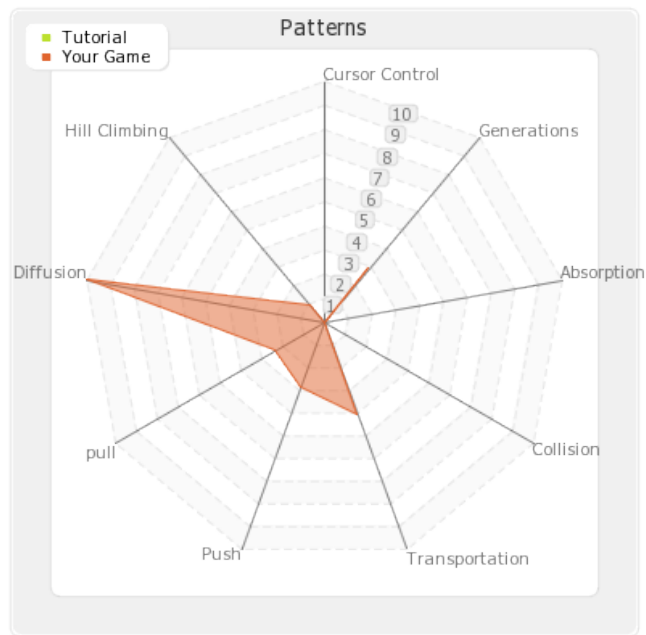


Figure 56 Diffusion Pattern

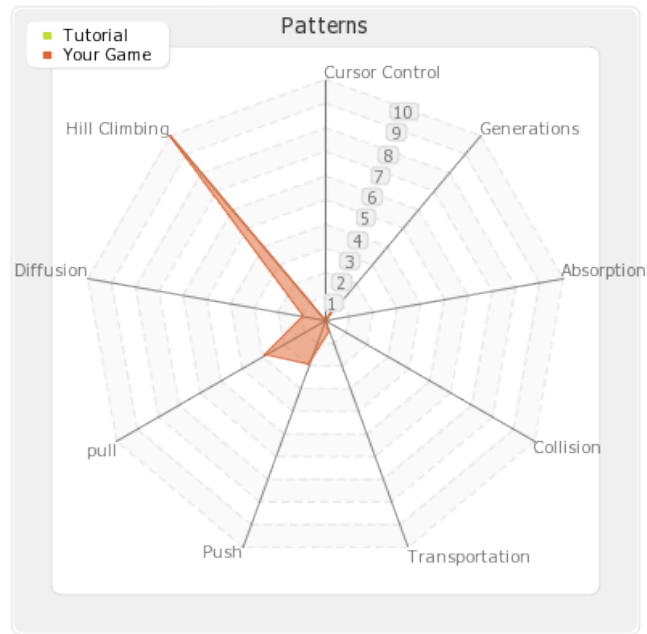


Figure 57 Hill Climbing Pattern

Appendix B. CTPA Evolution Chart

Over the last 3 years, the CTPA has been updated using expert agreement, which is one form of validation of CAS systems. When I first designed the CTPA graph in 2009, the graphs were like the below charts.

	Prototype CTPA Graph
Frogger	<p>A radar chart for the game Frogger. The chart has eight axes: Cursor Control (10), Generation, Absorption, Collision, Transportation, Push, Pull, and Diffusion. The scores are approximately: Cursor Control (10), Generation (8), Absorption (7), Collision (7), Transportation (7), Push (7), Pull (7), and Diffusion (7).</p>
Sokoban	<p>A radar chart for the game Sokoban. The chart has eight axes: Cursor Control (10), Generation, Absorption, Collision, Transportation, Push, Pull, and Diffusion. The scores are approximately: Cursor Control (10), Generation (4), Absorption (4), Collision (4), Transportation (4), Push (4), Pull (4), and Diffusion (4).</p>
Space Invaders	<p>A radar chart for the game Space Invaders. The chart has eight axes: Cursor Control (8), Generation, Absorption, Collision, Transportation, Push, Pull, and Diffusion. The scores are approximately: Cursor Control (8), Generation (5), Absorption (5), Collision (5), Transportation (5), Push (5), Pull (5), and Diffusion (5).</p>

After AgentSheets 3.0 was released, I redesigned the CTPA matched to AgentSheets version 3.0, and I proposed four different CTPA types. Currently, Spatial CTPA is used to analyze all AgentSheets and AgentCubes projects. For a certain CTPA type, small changes in programming can bring small changes in CTPA, but for other CTPA types it can result in big changes in CTPA.

	Original	Projected	Spatial	Projected Spatial
Frogger				
Sokoban Tutorial Space Invaders				
Sokoban				
Example				

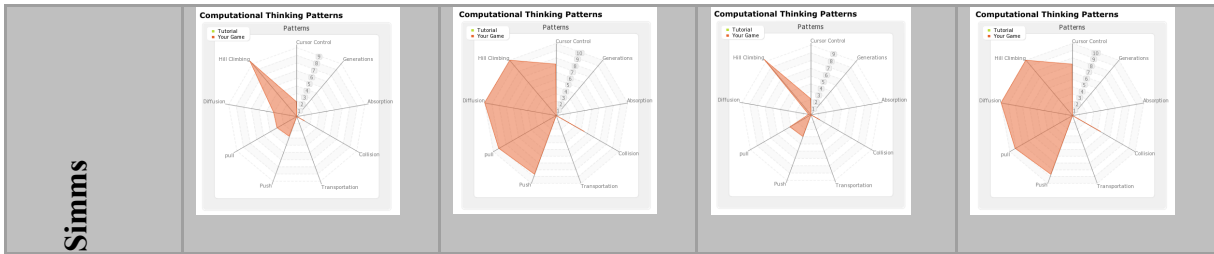
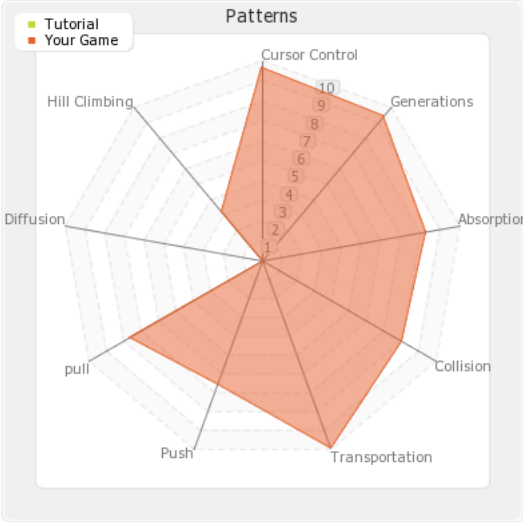
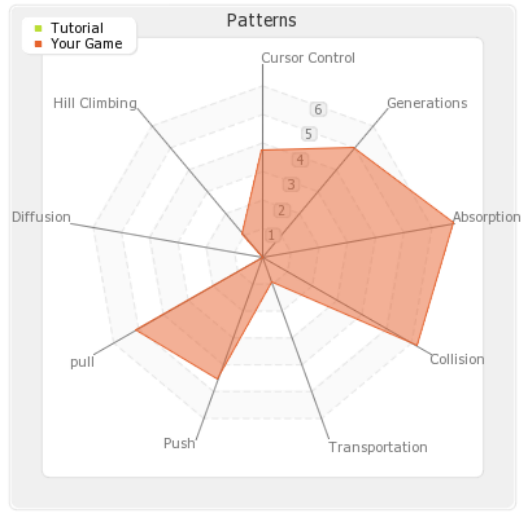


Table 8 Four Standard Games and Four Different CTPAs

- Projected CTPA overcomes the lower indication problem of less dominant patterns in a given game/simulation. Basically, this boosts low indications of less dominant patterns.
- Cons: The projected CTPA basically boosts the indication values, so the false positive values are increased too.
- Spatial CTPA is designed to eliminate false positives of hill climbing and diffusion using the “adding one more bit” approach. False positives of diffusion and hill climbing are dramatically improved. There is 0 value of false positive on diffusion, and small values of false positive on hill climbing still are identified because they are considered movement actions like cursor movement.
- Cons: Like the original CTPA, the low indication problem from less dominant patterns still remains.
- Projected Spatial CTPA is a combination of the projected CTPA and the spatial CTPA.

Simms

Frogger																								
Category	Computational Thinking Pattern Graph	Comment																						
Original CTP	<p>Computational Thinking Patterns</p> <p>The radar chart displays the following approximate values for 'Your Game' (orange) across the ten patterns:</p> <table border="1"> <thead> <tr> <th>Pattern</th> <th>Value (approx.)</th> </tr> </thead> <tbody> <tr> <td>Cursor Control</td> <td>4.5</td> </tr> <tr> <td>Generations</td> <td>4.5</td> </tr> <tr> <td>Absorption</td> <td>4.5</td> </tr> <tr> <td>Collision</td> <td>4.5</td> </tr> <tr> <td>Transportation</td> <td>1.0</td> </tr> <tr> <td>Push</td> <td>2.5</td> </tr> <tr> <td>pull</td> <td>2.5</td> </tr> <tr> <td>Diffusion</td> <td>2.5</td> </tr> <tr> <td>Hill Climbing</td> <td>2.5</td> </tr> <tr> <td>Generations (repeated)</td> <td>2.5</td> </tr> </tbody> </table>	Pattern	Value (approx.)	Cursor Control	4.5	Generations	4.5	Absorption	4.5	Collision	4.5	Transportation	1.0	Push	2.5	pull	2.5	Diffusion	2.5	Hill Climbing	2.5	Generations (repeated)	2.5	<p>This original CTP graph illustrates all implemented CT patterns in the sense of their proportion in game implementation. So more dominant patterns in the given game show higher indications than less dominant patterns on the CTP graph. i.e)</p> <p>Transportation is a pattern of Frogger, but the proportion is small (only one action). So it has a very low indication.</p>
Pattern	Value (approx.)																							
Cursor Control	4.5																							
Generations	4.5																							
Absorption	4.5																							
Collision	4.5																							
Transportation	1.0																							
Push	2.5																							
pull	2.5																							
Diffusion	2.5																							
Hill Climbing	2.5																							
Generations (repeated)	2.5																							

<p>Projected CTP</p>	<p>Computational Thinking Patterns</p> 	<p>To overcome the lower indication of less dominant patterns, a different type of high dimensional cosine calculation is applied to draw this Projected CTP. In the projected CTP, the high dimensional cosine calculation is performed with only relevant conditions and actions between CT patterns and a given game. Consequently, it boosts low indications of less dominant patterns.</p>
<p>Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>The Spatial CTP is designed to eliminate false positives of hill climbing and diffusion. This approach does not do much here.</p>

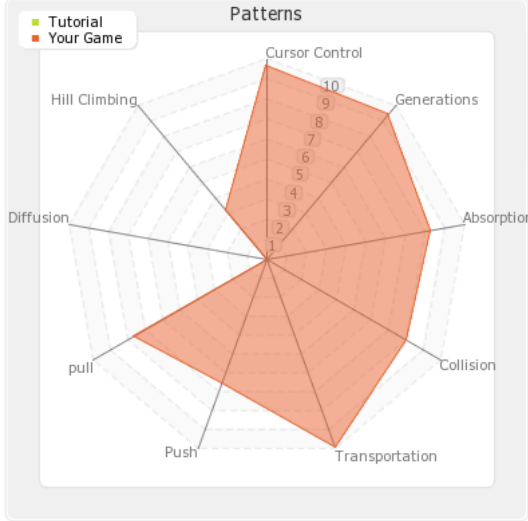
<p>Projected Spatial CTP</p>	<p>Computational Thinking Patterns</p>  <p>The radar chart displays the following approximate scores for 'Your Game' (orange) across the categories:</p> <table border="1"> <thead> <tr> <th>Category</th> <th>Score (Your Game)</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>10</td></tr> <tr><td>Generations</td><td>8</td></tr> <tr><td>Absorption</td><td>7</td></tr> <tr><td>Collision</td><td>6</td></tr> <tr><td>Transportation</td><td>5</td></tr> <tr><td>Push</td><td>4</td></tr> <tr><td>pull</td><td>3</td></tr> <tr><td>Diffusion</td><td>2</td></tr> <tr><td>Hill Climbing</td><td>1</td></tr> <tr><td>Cursor Control</td><td>1</td></tr> </tbody> </table>	Category	Score (Your Game)	Cursor Control	10	Generations	8	Absorption	7	Collision	6	Transportation	5	Push	4	pull	3	Diffusion	2	Hill Climbing	1	Cursor Control	1	<p>The Projected Spatial CTP is a combination of the projected CTP and the spatial CTP.</p>
Category	Score (Your Game)																							
Cursor Control	10																							
Generations	8																							
Absorption	7																							
Collision	6																							
Transportation	5																							
Push	4																							
pull	3																							
Diffusion	2																							
Hill Climbing	1																							
Cursor Control	1																							

Table 9 Frogger with Four CTPAs

Space Invaders																																			
Category	Computational Thinking Pattern Graph	Comment																																	
Original CTP	<p>Computational Thinking Patterns</p> <table border="1"> <caption>Original CTP Data</caption> <thead> <tr> <th>Pattern</th> <th>Tutorial</th> <th>Your Game</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>1</td><td>1</td></tr> <tr><td>Generations</td><td>1</td><td>2</td></tr> <tr><td>Absorption</td><td>1</td><td>3</td></tr> <tr><td>Collision</td><td>1</td><td>4</td></tr> <tr><td>Transportation</td><td>1</td><td>5</td></tr> <tr><td>Push</td><td>1</td><td>6</td></tr> <tr><td>pull</td><td>1</td><td>6</td></tr> <tr><td>Diffusion</td><td>1</td><td>6</td></tr> <tr><td>Hill Climbing</td><td>1</td><td>6</td></tr> <tr><td>Absorption</td><td>1</td><td>6</td></tr> </tbody> </table>	Pattern	Tutorial	Your Game	Cursor Control	1	1	Generations	1	2	Absorption	1	3	Collision	1	4	Transportation	1	5	Push	1	6	pull	1	6	Diffusion	1	6	Hill Climbing	1	6	Absorption	1	6	<p>After upgrading the CTP graph to match AS 3.0, the shape of the CTP graph is a bit different from what we've seen before, but still the same mechanism.</p> <p>False positives on Hill Climbing and Diffusion are shown.</p>
Pattern	Tutorial	Your Game																																	
Cursor Control	1	1																																	
Generations	1	2																																	
Absorption	1	3																																	
Collision	1	4																																	
Transportation	1	5																																	
Push	1	6																																	
pull	1	6																																	
Diffusion	1	6																																	
Hill Climbing	1	6																																	
Absorption	1	6																																	
Projected CTP	<p>Computational Thinking Patterns</p> <table border="1"> <caption>Projected CTP Data</caption> <thead> <tr> <th>Pattern</th> <th>Tutorial</th> <th>Your Game</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>1</td><td>1</td></tr> <tr><td>Generations</td><td>1</td><td>2</td></tr> <tr><td>Absorption</td><td>1</td><td>3</td></tr> <tr><td>Collision</td><td>1</td><td>4</td></tr> <tr><td>Transportation</td><td>1</td><td>5</td></tr> <tr><td>Push</td><td>1</td><td>6</td></tr> <tr><td>pull</td><td>1</td><td>7</td></tr> <tr><td>Diffusion</td><td>1</td><td>8</td></tr> <tr><td>Hill Climbing</td><td>1</td><td>9</td></tr> <tr><td>Absorption</td><td>1</td><td>10</td></tr> </tbody> </table>	Pattern	Tutorial	Your Game	Cursor Control	1	1	Generations	1	2	Absorption	1	3	Collision	1	4	Transportation	1	5	Push	1	6	pull	1	7	Diffusion	1	8	Hill Climbing	1	9	Absorption	1	10	<p>The projected CTP basically boosts the indication values, so the false positive values are increased too.</p>
Pattern	Tutorial	Your Game																																	
Cursor Control	1	1																																	
Generations	1	2																																	
Absorption	1	3																																	
Collision	1	4																																	
Transportation	1	5																																	
Push	1	6																																	
pull	1	7																																	
Diffusion	1	8																																	
Hill Climbing	1	9																																	
Absorption	1	10																																	

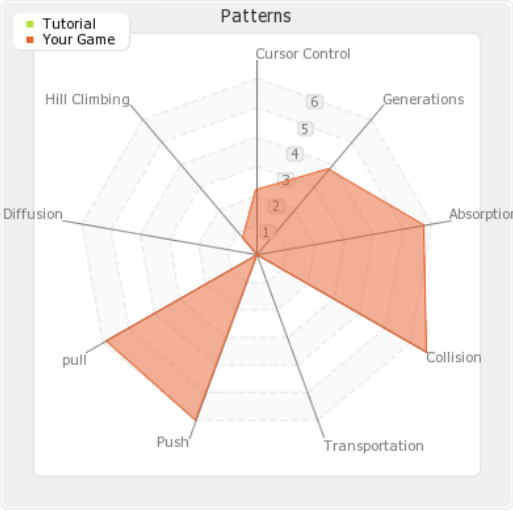
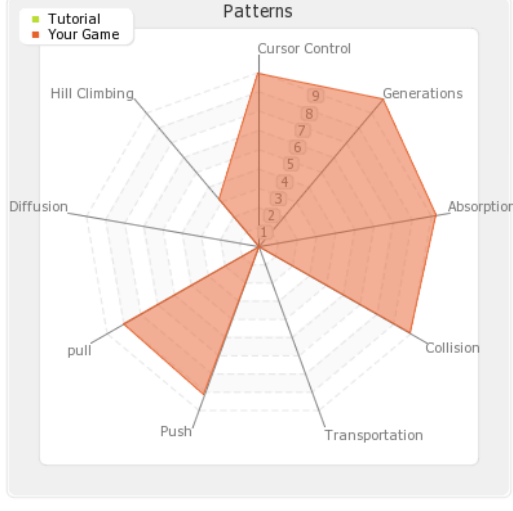
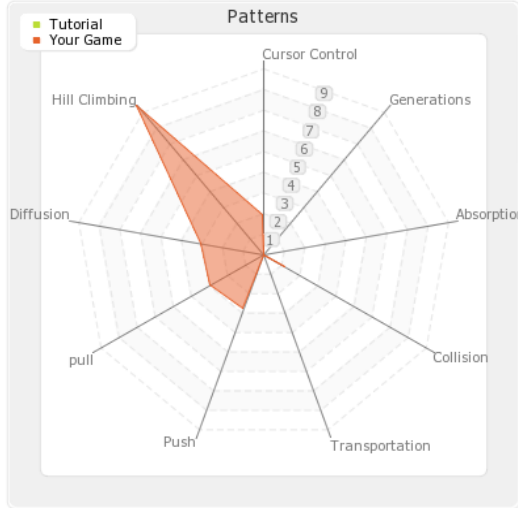
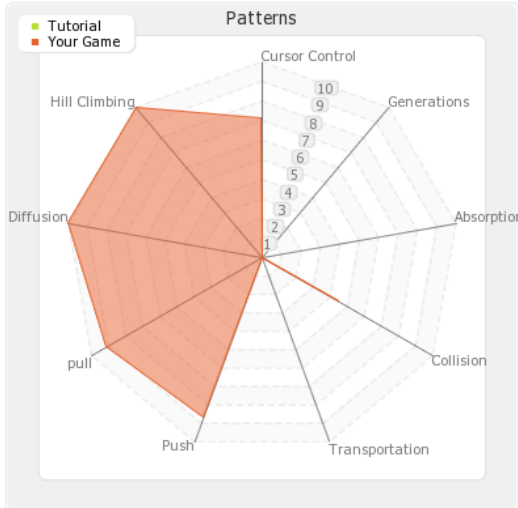
<p>Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>Adding one more bit. To eliminate false positives from diffusion and hill climbing, special code is added to determine spatial reference. There is 0 value of false positive on diffusion, and small values of false positive on hill climbing still are identified because it is considered as a movement action like cursor movement.</p>
<p>Projected Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>The Projected Spatial CTP is a combination of the projected CTP and the spatial CTP.</p>

Table 10 Space Invaders with Four CTPAs

Sims		
Category	Computational Thinking Pattern Graph	Comment
Original CTP	<p>Computational Thinking Patterns</p>  <p>The radar chart displays the following approximate values for 'Your Game' (orange) across the patterns: Hill Climbing (8), Diffusion (4), pull (3), Push (3), Transportation (3), Collision (2), Absorption (2), Generations (2), Cursor Control (1), and Absorption (1). The Tutorial (green) series shows much lower values, generally below 2 for all patterns.</p>	<p>In complete Sims games, hill climbing is a dominant pattern (all AI moving agents), and diffusion is a less dominant pattern (only a tile or a background agent). Push and Pull patterns show because Make and Move actions are used a lot in this Sims.</p>
Projected CTP	<p>Computational Thinking Patterns</p>  <p>The radar chart displays the following approximate values for 'Your Game' (orange) across the patterns: Hill Climbing (9), Diffusion (6), pull (5), Push (5), Transportation (5), Collision (3), Absorption (2), Generations (2), Cursor Control (1), and Absorption (1). The Tutorial (green) series shows much lower values, generally below 2 for all patterns.</p>	<p>The projected CTP basically boosts the indication values, so the false positive values are increased too.</p>

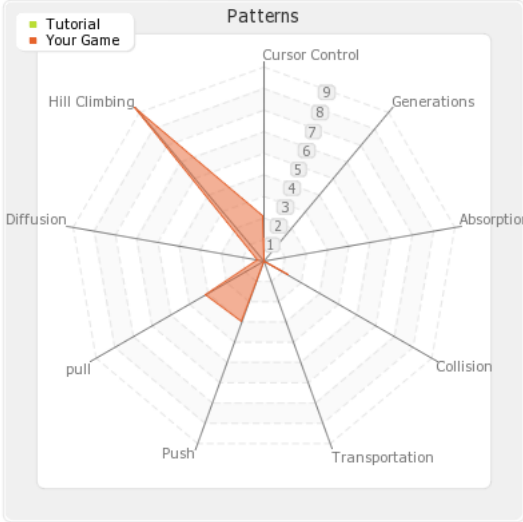
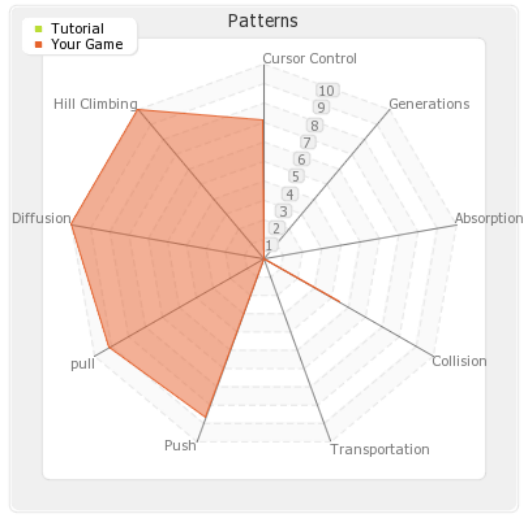
<p>Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>When calculating high dimensional cosine with spatial code, it eliminates all set actions that are not used to diffuse values. Consequently, diffusion becomes a less dominant pattern than before.</p>
<p>Projected Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>The Projected Spatial CTP is a combination of the projected CTP and the spatial CTP.</p>

Table 11 Sims with Four CTPAs

Sokoban (Tutorial Version)																										
Category	Computational Thinking Pattern Graph	Comment																								
Original CTP	<p>Computational Thinking Patterns</p> <p>Legend: Tutorial (Green), Your Game (Orange)</p> <table border="1"> <caption>Original CTP Data</caption> <thead> <tr> <th>Pattern</th> <th>Tutorial</th> <th>Your Game</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>7</td><td>7</td></tr> <tr><td>Generations</td><td>6</td><td>6</td></tr> <tr><td>Absorption</td><td>5</td><td>5</td></tr> <tr><td>Collision</td><td>4</td><td>4</td></tr> <tr><td>Transportation</td><td>3</td><td>3</td></tr> <tr><td>Push</td><td>2</td><td>2</td></tr> <tr><td>Hill Climbing</td><td>1</td><td>1</td></tr> </tbody> </table>	Pattern	Tutorial	Your Game	Cursor Control	7	7	Generations	6	6	Absorption	5	5	Collision	4	4	Transportation	3	3	Push	2	2	Hill Climbing	1	1	<p>After upgrading the CTP graph to match AS 3.0, the shape of the CTP graph is a bit different from what we've seen before, but still the same mechanism.</p> <p>False positives on Hill Climbing and Diffusion are shown.</p>
Pattern	Tutorial	Your Game																								
Cursor Control	7	7																								
Generations	6	6																								
Absorption	5	5																								
Collision	4	4																								
Transportation	3	3																								
Push	2	2																								
Hill Climbing	1	1																								
Projected CTP	<p>Computational Thinking Patterns</p> <p>Legend: Tutorial (Green), Your Game (Orange)</p> <table border="1"> <caption>Projected CTP Data</caption> <thead> <tr> <th>Pattern</th> <th>Tutorial</th> <th>Your Game</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>7</td><td>10</td></tr> <tr><td>Generations</td><td>6</td><td>9</td></tr> <tr><td>Absorption</td><td>5</td><td>8</td></tr> <tr><td>Collision</td><td>4</td><td>7</td></tr> <tr><td>Transportation</td><td>3</td><td>6</td></tr> <tr><td>Push</td><td>2</td><td>5</td></tr> <tr><td>Hill Climbing</td><td>1</td><td>4</td></tr> </tbody> </table>	Pattern	Tutorial	Your Game	Cursor Control	7	10	Generations	6	9	Absorption	5	8	Collision	4	7	Transportation	3	6	Push	2	5	Hill Climbing	1	4	<p>The projected CTP basically boosts the indication values, so the false positive values are increased too.</p>
Pattern	Tutorial	Your Game																								
Cursor Control	7	10																								
Generations	6	9																								
Absorption	5	8																								
Collision	4	7																								
Transportation	3	6																								
Push	2	5																								
Hill Climbing	1	4																								

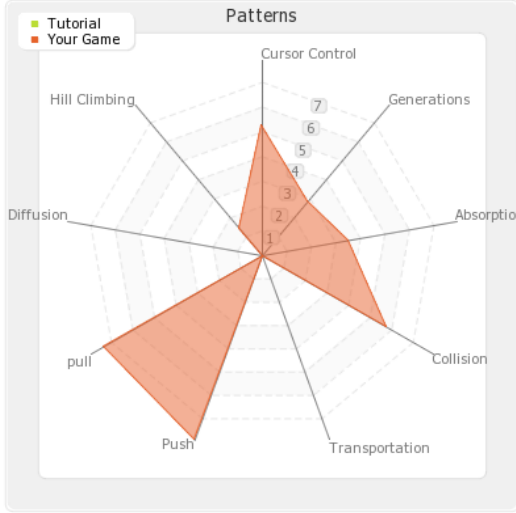
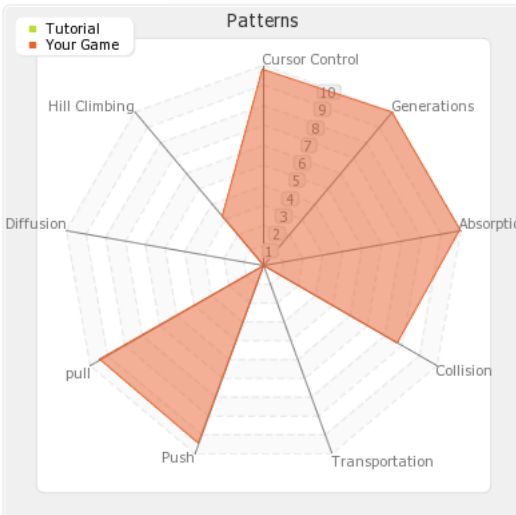
<p>Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>Adding one more bit. To eliminate false positives from diffusion and hill climbing, special code is added to determine spatial reference. There is 0 value of false positive on diffusion, and small values of false positive on hill climbing still are identified because it is considered as a movement action like cursor movement.</p>
<p>Projected Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>The Projected Spatial CTP is a combination of the projected CTP and the spatial CTP. Still it shows some false positive on Generation because of many see conditions.</p>

Table 12 Tutorial Sokoban with Four CTPAs

Sokoban (Example Version in AS)																																
Category	Computational Thinking Pattern Graph	Comment																														
Original CTP	<p>Computational Thinking Patterns</p> <table border="1"> <caption>Approximate values from the 'Original CTP' radar chart</caption> <thead> <tr> <th>Category</th> <th>Tutorial</th> <th>Your Game</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>3</td><td>4</td></tr> <tr><td>Generations</td><td>4</td><td>5</td></tr> <tr><td>Absorption</td><td>5</td><td>6</td></tr> <tr><td>Collision</td><td>6</td><td>7</td></tr> <tr><td>Transportation</td><td>7</td><td>8</td></tr> <tr><td>Push</td><td>8</td><td>9</td></tr> <tr><td>pull</td><td>9</td><td>10</td></tr> <tr><td>Diffusion</td><td>10</td><td>11</td></tr> <tr><td>Hill Climbing</td><td>11</td><td>12</td></tr> </tbody> </table>	Category	Tutorial	Your Game	Cursor Control	3	4	Generations	4	5	Absorption	5	6	Collision	6	7	Transportation	7	8	Push	8	9	pull	9	10	Diffusion	10	11	Hill Climbing	11	12	<p>After upgrading the CTP graph to match AS 3.0, the shape of the CTP graph is a bit different from what we've seen before, but still the same mechanism.</p> <p>False positives on Hill Climbing and Diffusion are shown.</p>
Category	Tutorial	Your Game																														
Cursor Control	3	4																														
Generations	4	5																														
Absorption	5	6																														
Collision	6	7																														
Transportation	7	8																														
Push	8	9																														
pull	9	10																														
Diffusion	10	11																														
Hill Climbing	11	12																														
Projected CTP	<p>Computational Thinking Patterns</p> <table border="1"> <caption>Approximate values from the 'Projected CTP' radar chart</caption> <thead> <tr> <th>Category</th> <th>Tutorial</th> <th>Your Game</th> </tr> </thead> <tbody> <tr><td>Cursor Control</td><td>3</td><td>4</td></tr> <tr><td>Generations</td><td>4</td><td>10</td></tr> <tr><td>Absorption</td><td>5</td><td>9</td></tr> <tr><td>Collision</td><td>6</td><td>8</td></tr> <tr><td>Transportation</td><td>7</td><td>7</td></tr> <tr><td>Push</td><td>8</td><td>8</td></tr> <tr><td>pull</td><td>9</td><td>9</td></tr> <tr><td>Diffusion</td><td>10</td><td>10</td></tr> <tr><td>Hill Climbing</td><td>11</td><td>11</td></tr> </tbody> </table>	Category	Tutorial	Your Game	Cursor Control	3	4	Generations	4	10	Absorption	5	9	Collision	6	8	Transportation	7	7	Push	8	8	pull	9	9	Diffusion	10	10	Hill Climbing	11	11	<p>The projected CTP basically boosts the indication values, so the false positive values are increased too.</p>
Category	Tutorial	Your Game																														
Cursor Control	3	4																														
Generations	4	10																														
Absorption	5	9																														
Collision	6	8																														
Transportation	7	7																														
Push	8	8																														
pull	9	9																														
Diffusion	10	10																														
Hill Climbing	11	11																														

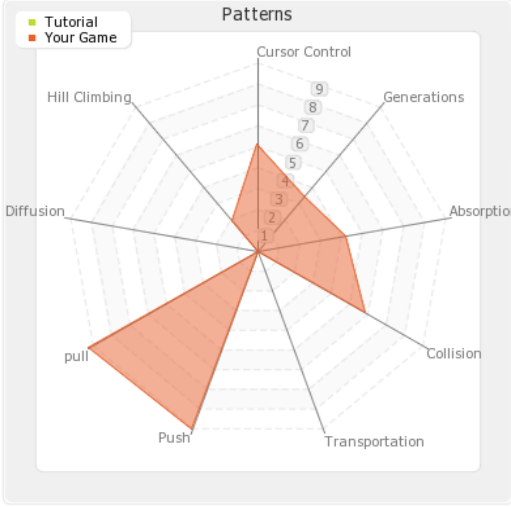
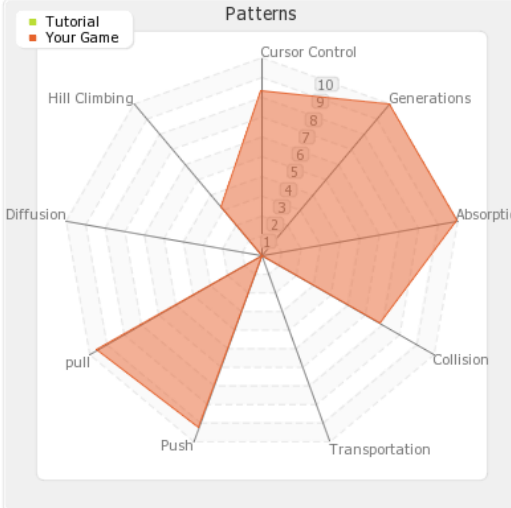
<p>Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>Adding one more bit. To eliminate false positives from diffusion and hill climbing, special code is added to determine spatial reference. There is 0 value of false positive on diffusion, and small values of false positive on hill climbing still are identified because it is considered as a movement action like cursor movement.</p>
<p>Projected Spatial CTP</p>	<p>Computational Thinking Patterns</p> 	<p>The Projected Spatial CTP is a combination of the projected CTP and the spatial CTP.</p>

Table 13 Example Sokoban with Four CTPAs

Appendix C. Face and Construct Validity Survey

Questionnaire

1. Using the following AgentSheets programming codes, can you detect the Cursor Control pattern? Please justify your answer.




2. Using the following AgentSheets programming codes, can you detect the Cursor Control pattern? Please justify your answer.






3. Using the following AgentSheets programming codes, can you detect the Generation pattern? Please justify your answer.




4. Using the following AgentSheets programming codes, can you detect the Generation pattern? Please justify your answer.

If	See 	Then	
If	% chance <input type="text" value="50"/>	Then	
If		Then	New  

5. Using the following AgentSheets programming codes, can you detect the Absorption pattern? Please justify your answer.

If	See  	Then	Erase 
----	---	------	--

6. Using the following AgentSheets programming codes, can you detect the Absorption pattern? Please justify your answer.

If	See  	Then	
If		Then	Erase 

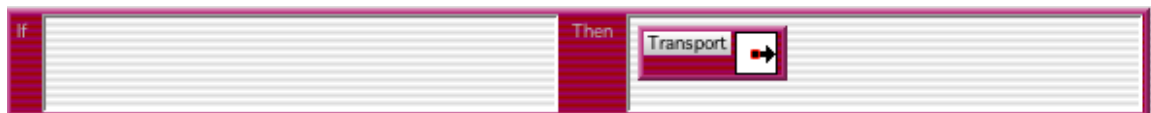
7. Using the following AgentSheets programming codes, can you detect the Collision pattern? Please justify your answer.



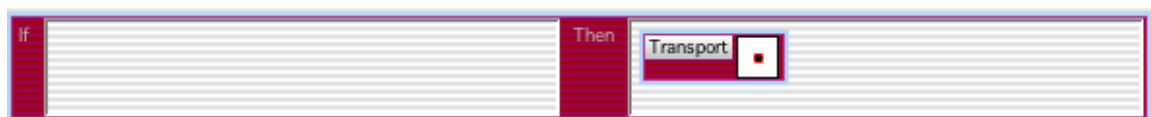
8. Using the following AgentSheets programming codes, can you detect the Collision pattern? Please justify your answer.



9. Using the following AgentSheets programming codes, can you detect the Transportation pattern? Please justify your answer.



10. Using the following AgentSheets programming codes, can you detect the Transportation pattern? Please justify your answer.



11. Using the following AgentSheets programming codes, can you detect the Push pattern? Please justify your answer. The Push pattern requires two adjacent agents' interaction so there are two programming behaviors from two agents in this question.

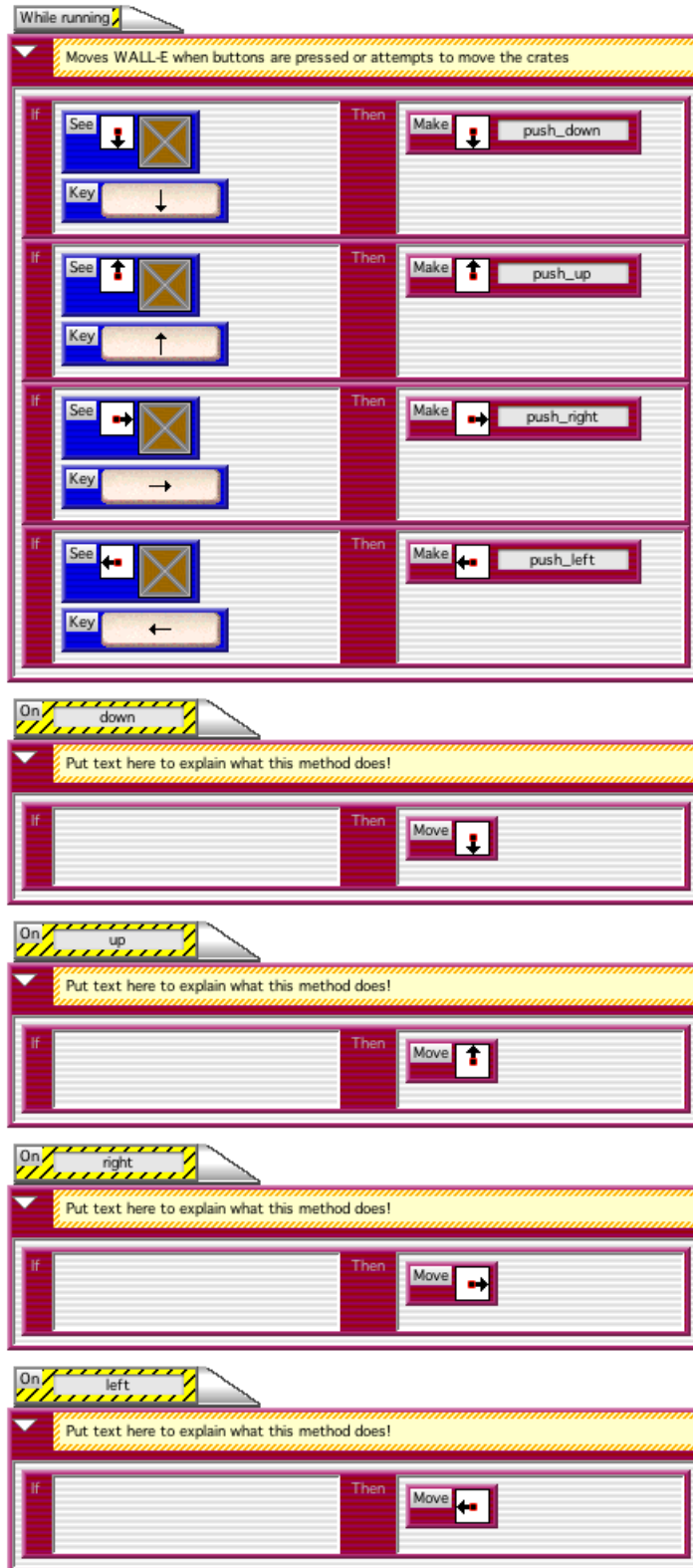


Figure 58: Programming Behavior in Agent 1

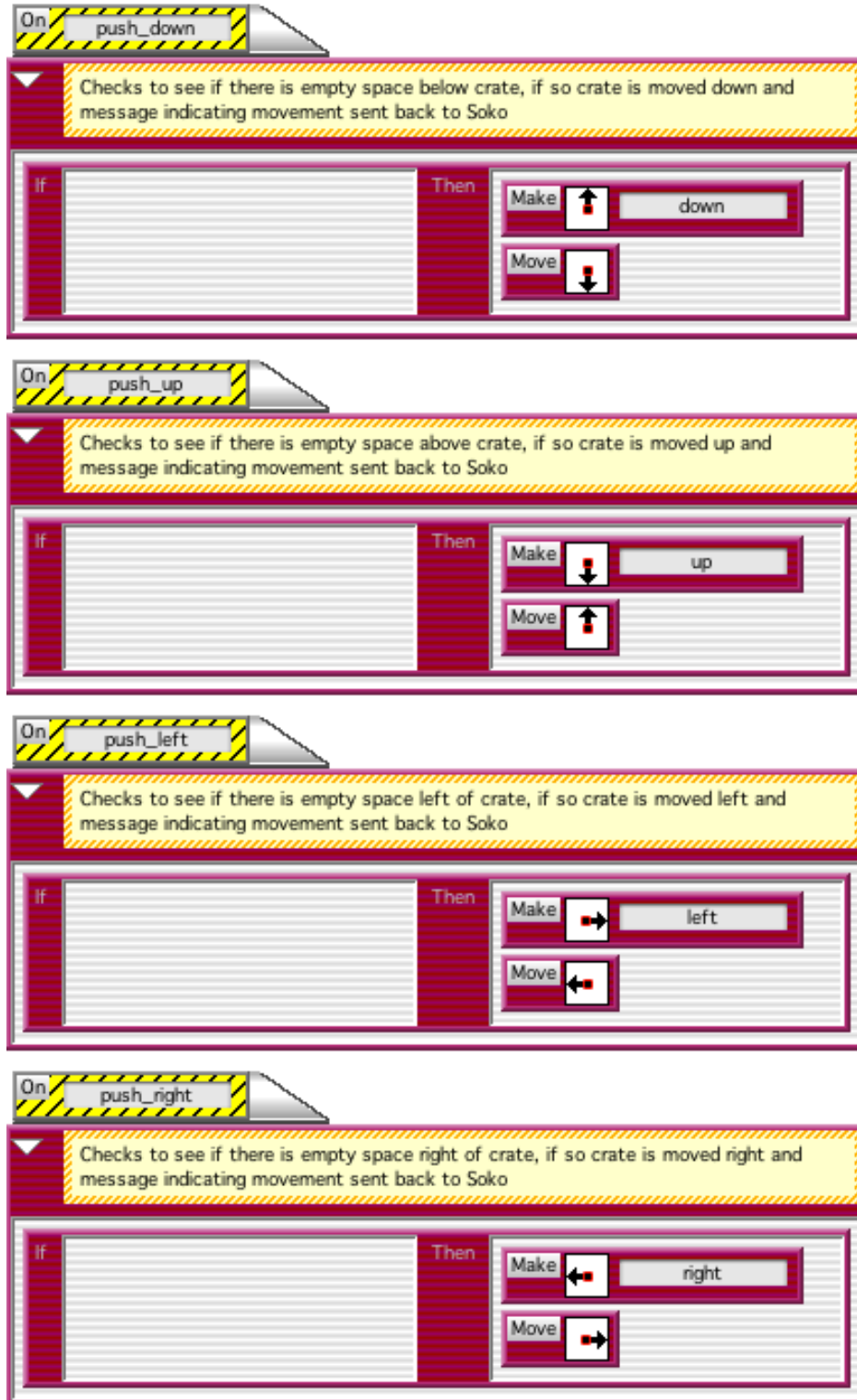


Figure 59 Programming Behavior in Agent 2

12. Using the following AgentSheets programming codes, can you detect the Push pattern? Please justify your answer. The Push pattern requires two adjacent agents' interaction so there are two programming behaviors from two agents in this question.

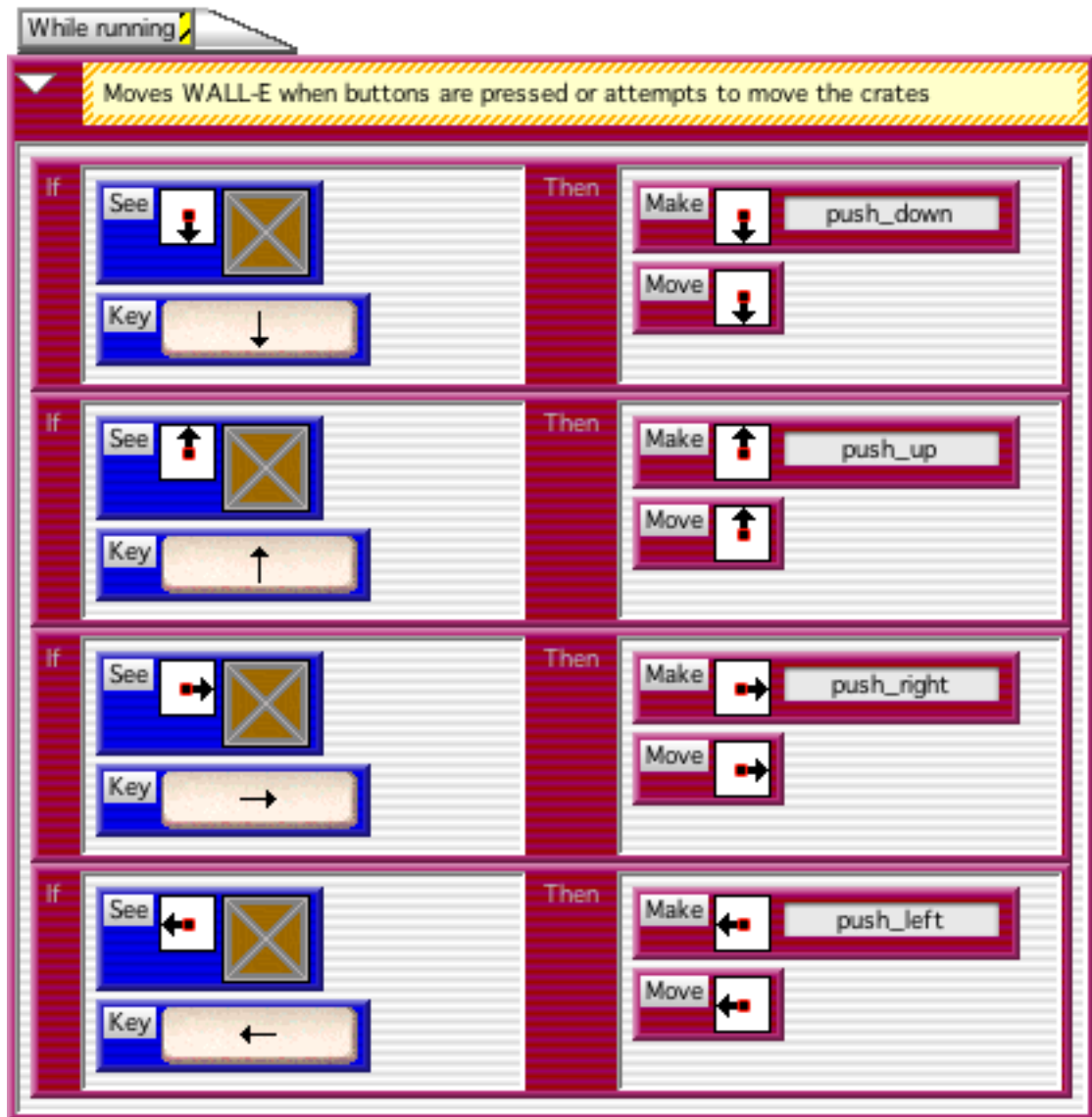


Figure 60 Programing Behavior in Agent 1

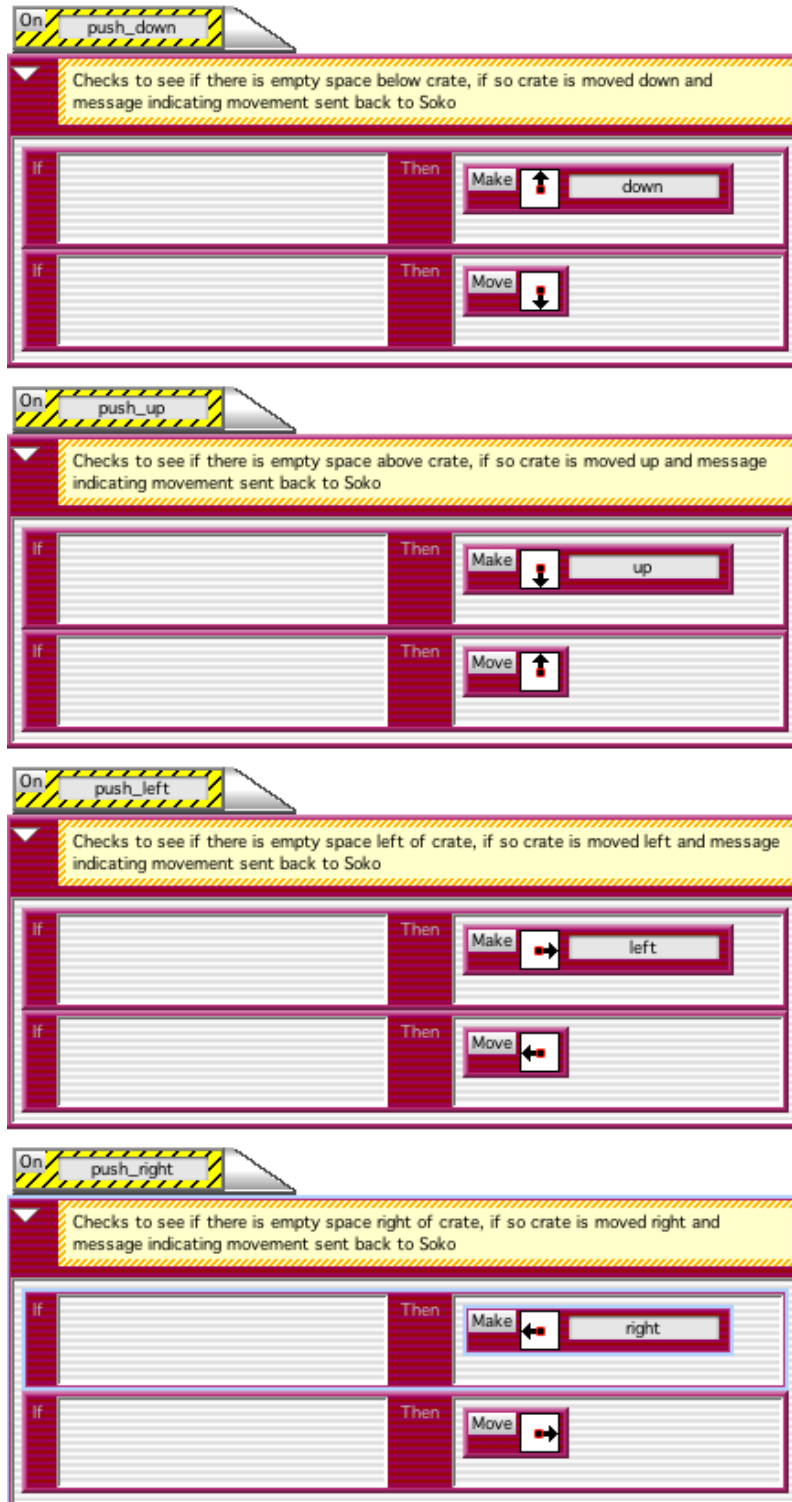


Figure 61 Programming Behavior in Agent 2

13. Using the following AgentSheets programming codes, can you detect the Pull pattern? Please justify your answer. The Pull pattern requires two adjacent agents' interaction so there are two programming behaviors from two agents in this question.



Figure 62 Programing Behavior in Agent 1

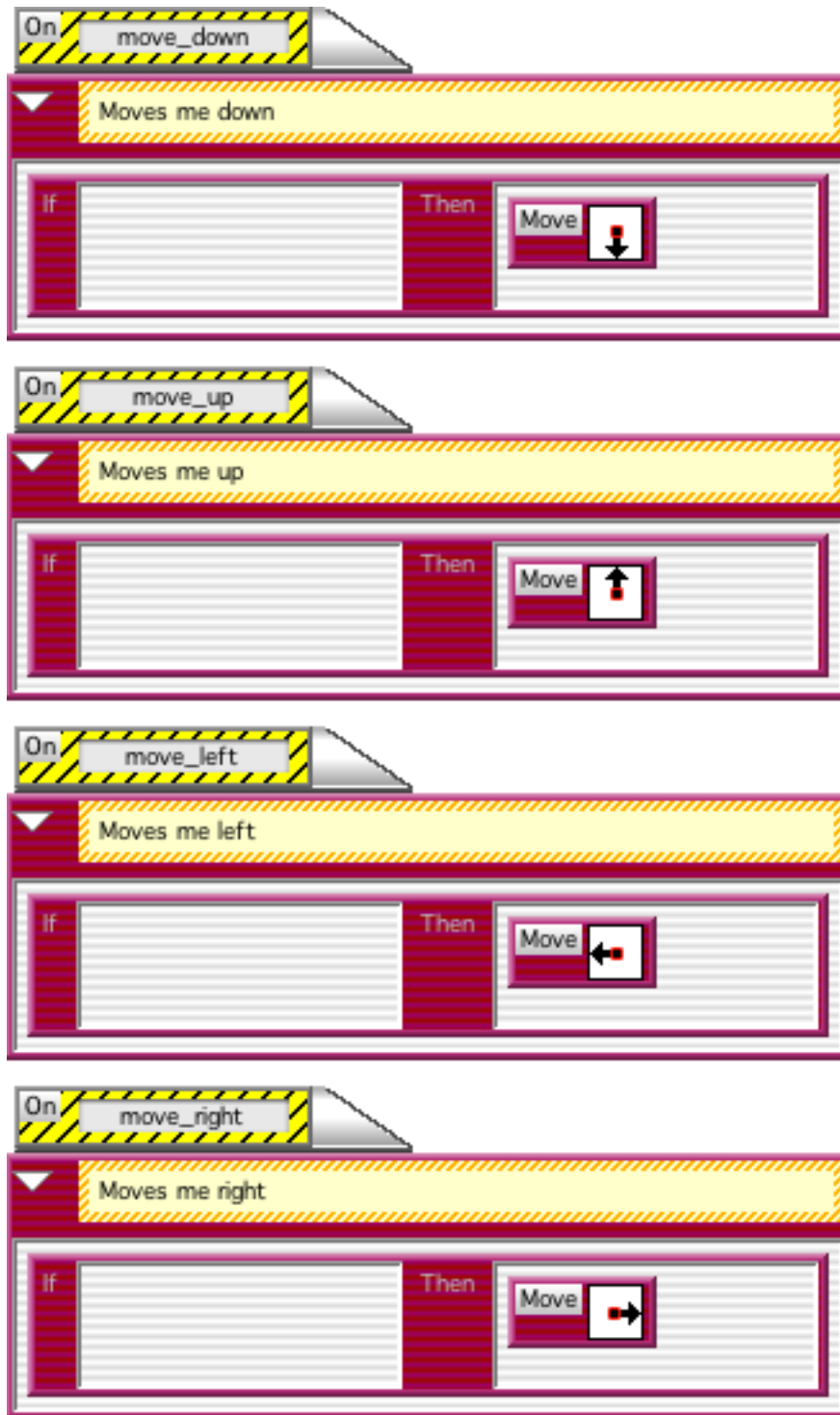


Figure 63 Programing Behavior in Agent 2

14. Using the following AgentSheets programming codes, can you detect the Pull pattern? Please justify your answer. The Pull pattern requires two adjacent agents' interaction so there are two programming behaviors from two agents in this question.

















If	See  	Then	Make  move_down
If	See  	Then	Make  move_right
If	See  	Then	Make  move_left
If	See  	Then	Make  move_up
If		Then	Move 
If		Then	Move 
If		Then	Move 
If		Then	Move 

Figure 64 Programing Behavior in Agent 1

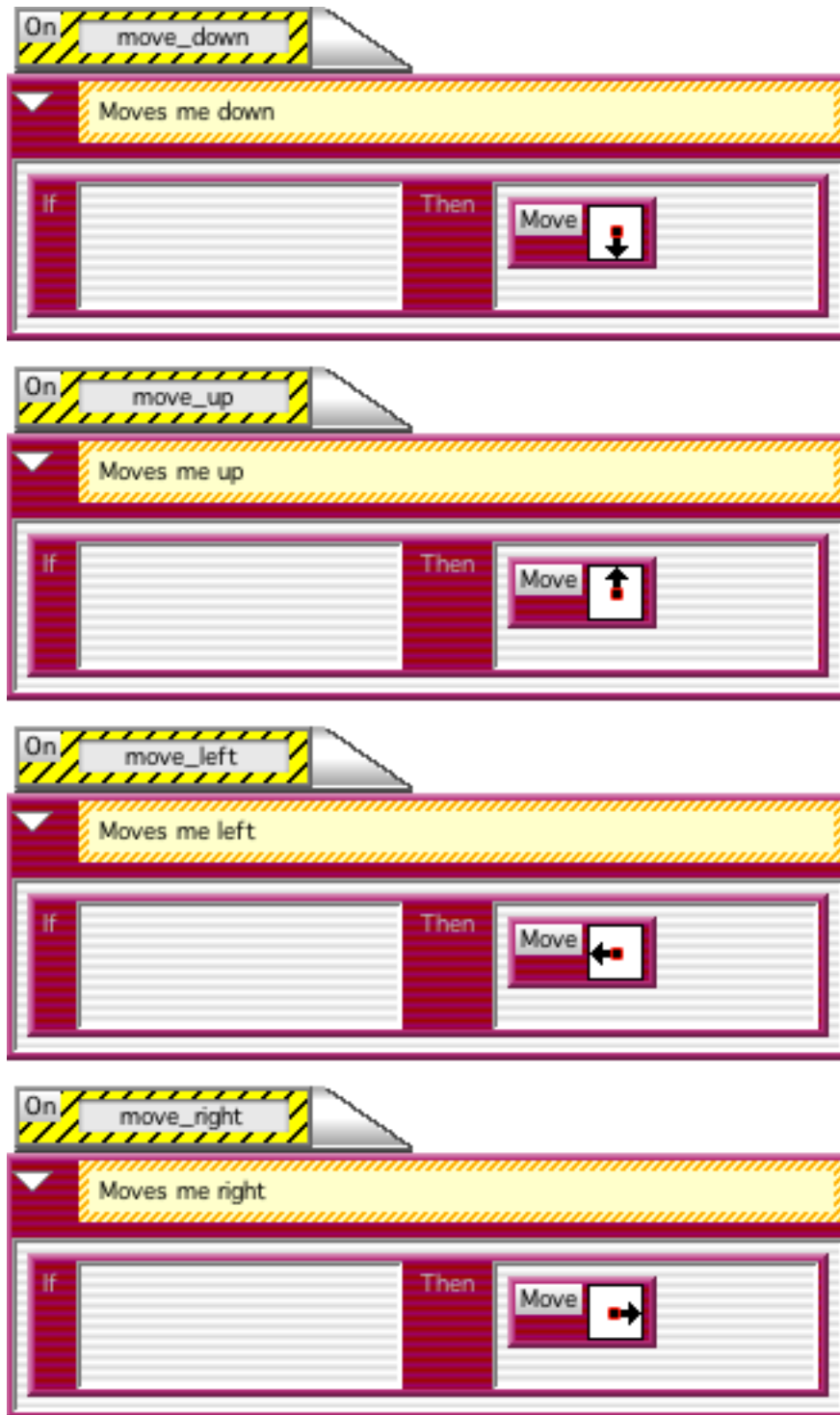
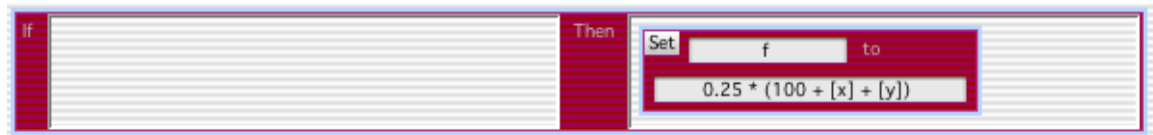


Figure 65 Programing Behavior in Agent 2

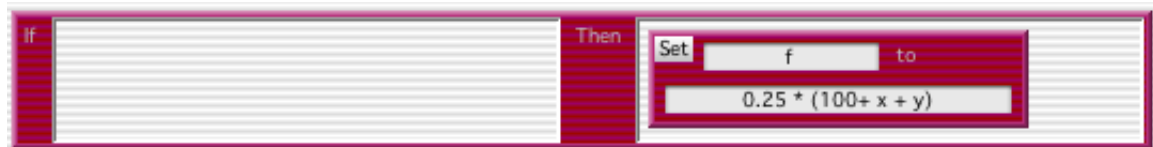
15. Using the following AgentSheets programming codes, can you detect the Diffusion pattern? Please justify your answer. The equation in the box is $0.25 * (f[\text{left}] + f[\text{right}] + f[\text{up}] + f[\text{down}])$.



16. Using the following AgentSheets programming codes, can you detect the Diffusion pattern? Please justify your answer. The equation in the box is $0.25 * (100 + [x] + [y])$.



17. Using the following AgentSheets programming codes, can you detect the Diffusion pattern? Please justify your answer. The equation in the box is $0.25 * (100 + x + y)$.



17. Using the following AgentSheets programming codes, can you detect the Hill Climbing pattern? Please justify your answer.

<p>if</p> <p>is <input type="text" value="n[up]"/> > <input type="text" value="n[left]"/></p> <p>is <input type="text" value="n[up]"/> > <input type="text" value="n[right]"/></p> <p>is <input type="text" value="n[up]"/> > <input type="text" value="n[down]"/></p>	<p>Then</p> <p>Move </p>
<p>if</p> <p>is <input type="text" value="n[down]"/> > <input type="text" value="n[left]"/></p> <p>is <input type="text" value="n[down]"/> > <input type="text" value="n[right]"/></p> <p>is <input type="text" value="n[down]"/> > <input type="text" value="n[up]"/></p>	<p>Then</p> <p>Move </p>
<p>if</p> <p>is <input type="text" value="n[right]"/> > <input type="text" value="n[left]"/></p> <p>is <input type="text" value="n[right]"/> > <input type="text" value="n[down]"/></p> <p>is <input type="text" value="n[right]"/> > <input type="text" value="n[up]"/></p>	<p>Then</p> <p>Move </p>
<p>if</p> <p>is <input type="text" value="n[left]"/> > <input type="text" value="n[right]"/></p> <p>is <input type="text" value="n[left]"/> > <input type="text" value="n[down]"/></p> <p>is <input type="text" value="n[left]"/> > <input type="text" value="n[up]"/></p>	<p>Then</p> <p>Move </p>

18. Using the following AgentSheets programming codes, can you detect the Hill Climbing pattern? Please justify your answer.

<p>if</p> <p>is <input type="text" value="n[up]"/> > <input type="text" value="n[left]"/></p> <p>is <input type="text" value="n[up]"/> > <input type="text" value="n[right]"/></p> <p>is <input type="text" value="n[up]"/> > <input type="text" value="n[down]"/></p>	<p>Then</p> <p><input type="text" value="bit"/></p>
<p>if</p> <p>is <input type="text" value="n[down]"/> > <input type="text" value="n[left]"/></p> <p>is <input type="text" value="n[down]"/> > <input type="text" value="n[right]"/></p> <p>is <input type="text" value="n[down]"/> > <input type="text" value="n[up]"/></p>	<p>Then</p> <p><input type="text" value="chink"/></p>
<p>if</p> <p>is <input type="text" value="n[right]"/> > <input type="text" value="n[left]"/></p> <p>is <input type="text" value="n[right]"/> > <input type="text" value="n[down]"/></p> <p>is <input type="text" value="n[right]"/> > <input type="text" value="n[up]"/></p>	<p>Then</p> <p><input type="text" value="clack"/></p>
<p>if</p> <p>is <input type="text" value="n[left]"/> > <input type="text" value="n[right]"/></p> <p>is <input type="text" value="n[left]"/> > <input type="text" value="n[down]"/></p> <p>is <input type="text" value="n[left]"/> > <input type="text" value="n[up]"/></p>	<p>Then</p> <p><input type="text" value="boing"/></p>

Appendix D. Face Validity with Proportions Survey

Questionnaire

1. Please open and play the sample Frogger project, which is under the Projects folder of AgentSheets. Please list the Computational Thinking Patterns in a proportional rank in the Frogger programming using your understanding of Computational Thinking Pattern. The sample Frogger uses Cursor Control, Generation, Absorption, Collision, and Transportation patterns. (i.e. 1. Transportation 2. Cursor Control 3. Generation 4. Absorption 5. Collision. A smaller number means a higher proportion in programming)
2. Please open and play the sample Sokoban project, which is under the Projects folder of AgentSheets. Please list the Computational Thinking Patterns in a proportional rank in the Sokoban programming using your understanding of Computational Thinking Pattern. The sample Sokoban uses Cursor Control, Push, and Pull patterns. (i.e. 1. Cursor Control 2. Push 3. Pull. A smaller number means a higher proportion in programming)
3. Please open and play the sample Space Invaders project, which is under the Projects folder of AgentSheets. Please list the Computational Thinking Patterns in a proportional rank in the Space Invaders programming using your understanding of Computational Thinking Pattern. The sample Space Invaders uses Cursor Control, Generation, Absorption, Collision, and Choreography patterns. (i.e. 1. Choreography 2. Cursor Control 3. Generation 4. Absorption 5. Collision a smaller number means a higher proportion in programming)

Appendix E. Four Standard Games with 13 dimensional CTPA

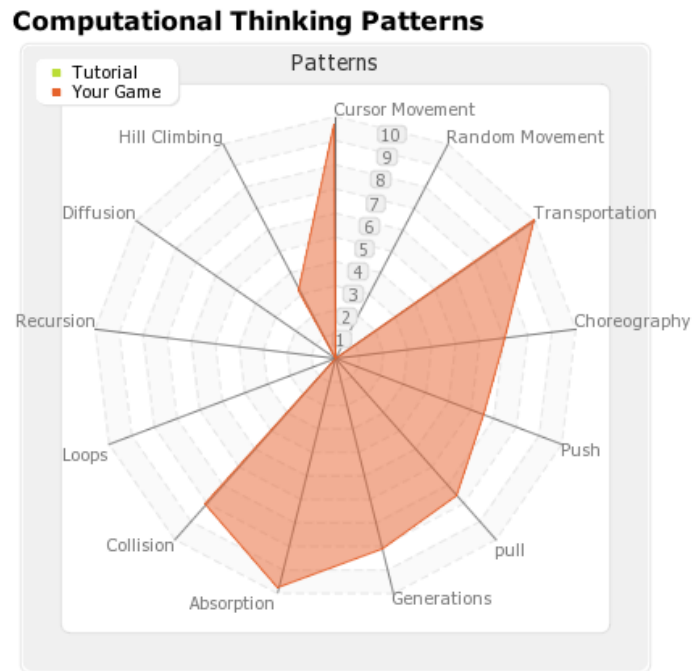


Figure 66 Frogger with 13 dimensional CTPA

Computational Thinking Patterns

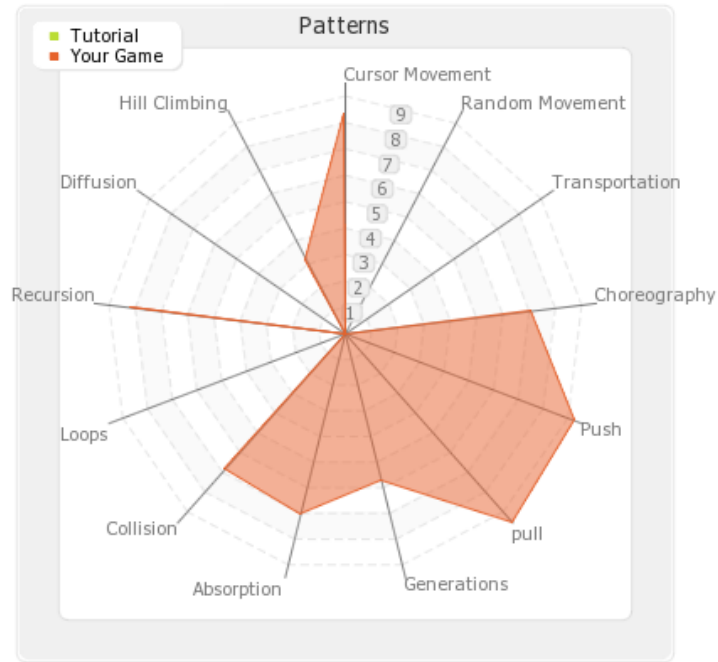


Figure 67 Example Sokoban with 13 dimensional CTPA

Computational Thinking Patterns

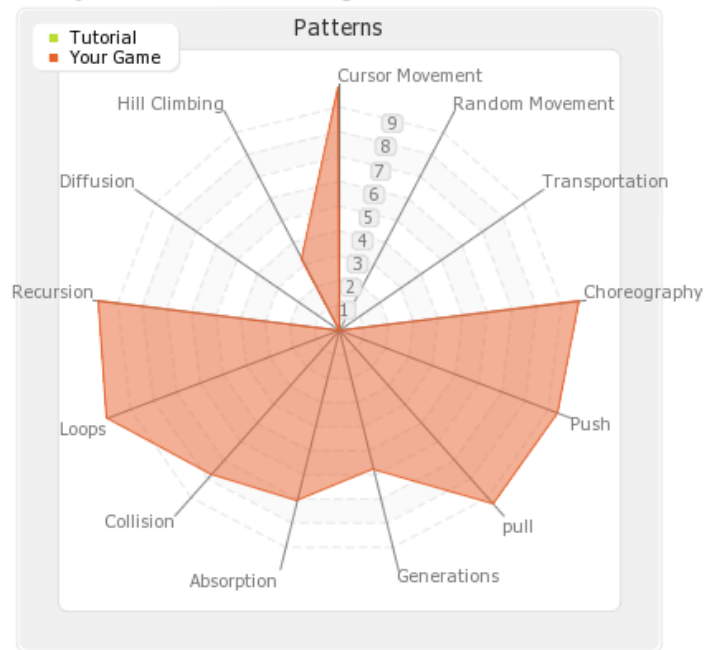


Figure 68 Tutorial Sokoban with 13 dimensional CTPA

Computational Thinking Patterns

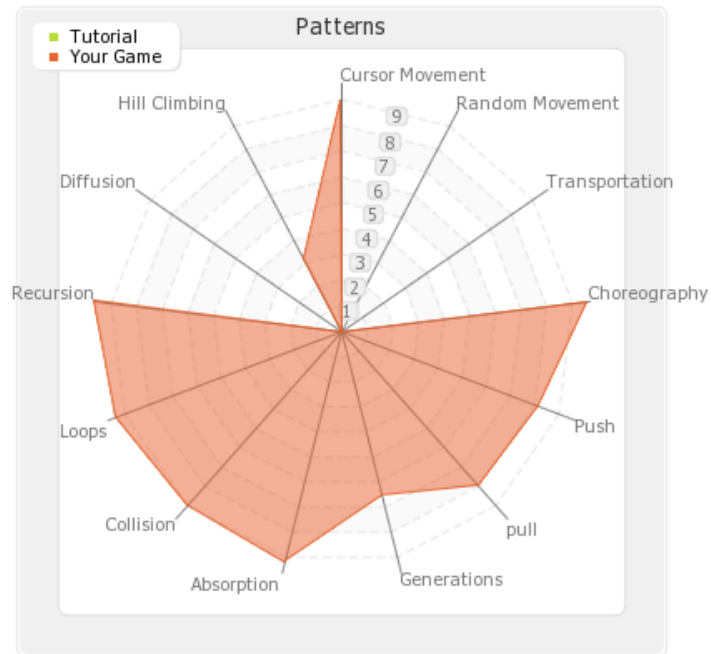


Figure 69 Space Invaders with 13 dimensional CTPA

Computational Thinking Patterns

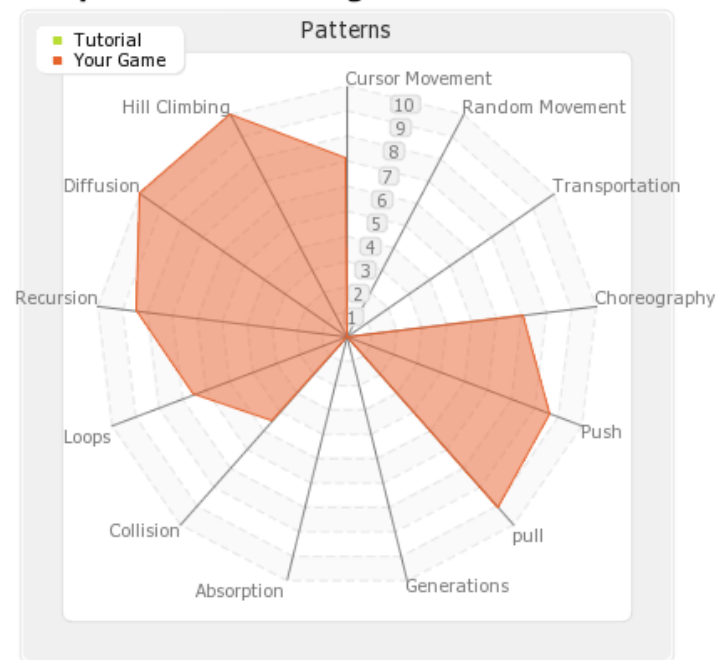


Figure 70 Sims with 13 dimensional CTPA