

FROM LIVED EXPERIENCES TO GAME CREATION:
HOW SCAFFOLDING SUPPORTS ELEMENTARY SCHOOL STUDENTS LEARNING COMPUTER SCIENCE
PRINCIPLES IN AN AFTER SCHOOL SETTING

by

IAN HER MANY HORSES

B.S., University of Colorado Boulder, 2006

A dissertation 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
School of Education
2016

ProQuest Number: 10108772

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.



ProQuest 10108772

Published by ProQuest LLC (2016). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

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

This dissertation entitled:
From Lived Experiences to Game Creation: How Scaffolding Supports Elementary School Students
Learning Computer Science Principles in an After School Setting
written by Ian Her Many Horses
has been approved for the School of Education, University of Colorado Boulder

Dr. Valerie Otero, Chair

Dr. Kris Gutiérrez

Dr. William Penuel

Dr. Alexander Repenning

Dr. David Webb

Date 4/12/2016

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 # 0610.13

Her Many Horses, Ian (Ph.D., Education [Curriculum & Instruction])

From Lived Experiences to Game Creation: How Scaffolding Supports Elementary School Students

Learning Computer Science Principles in an After School Setting

Dissertation directed by Professor Valerie Otero

Abstract

The world, and especially our own country, is in dire need of a larger and more diverse population of computer scientists. While many organizations have approached this problem of too few computer scientists in various ways, a promising, and I believe necessary, path is to expose elementary students to authentic practices of the discipline. Through a design research study I worked toward developing an effective method to engage elementary students (grades 3-5) in computer science (CS) through video game creation at an after school program called El Pueblo Mágico. To carry out this goal I implemented, and refined, two scaffolding tools over two iterations of the study to help students design and create their own games. These scaffolding tools were meant to support students in organizing and refining their own game ideas, while also assisting them in accessing the CS principles of design and algorithms. The students were to then use their designs to create their games using an agent-based programming environment. In the first iteration of the study I asked students to design their games using a pencil-and-paper *planning document* and then create the games using the AgentSheets programming environment. I found that the pencil-and-paper version of the scaffolding tool was too open for students and they never finished their designs or accessed information that would have helped them make their games. As a result, the students in the first study needed considerable help creating their games from me. In the second iteration of the study I asked students to use a new web-based scaffolding tool that I developed, called AgentDesign, to do design and then create their games using AgentCubes Online. The AgentDesign planning tool was successful in guiding most of the students through the process of design,

and these students not only completed the design process but also accessed much of the information that would help them to make their games. However, these students also did not completely create their games, despite completing the design process and having most of the information they would need at their fingertips.

Dedication

For my girls. Becky, Carmen, and Sonya.

Acknowledgements

I am grateful to Dr. Valerie Otero, for taking a chance on advising a computer science education student and supporting me in trying to make sense of just about everything. She truly modeled what it means to be an advisor, teacher, and learner. Dr. Kris Gutiérrez, Dr. Alex Repenning, and Dr. David Webb, gave me the encouragement, support, space, and time to try out something different. Dr. Bill Penuel, took the time to discuss issues with me that were challenging and meaningful for the work presented here. I am also grateful to Dr. Jim Curry and the Applied Math Department, for giving me my first opportunity to teach, and then my first opportunity to do research, this had a dramatic impact on the decisions I made and who I have become as a scholar. Dr. Vinnie Basile, Dr. Adam York and Henry Suarez provided significant support and encouragement throughout this process and helped me stay connected to the real world. Dr. Ben Van Dusen, Dr. Mike Ross, Dr. Deb Morrison, Dr. Sarah Heredia, and everyone else in the office created an important space for sense-making, deep thought, and intense growth.

Contents

Introduction	1
Literature Review	3
Goals of Computer Science	3
The Broken Pipeline of CS Education	13
Engagement & Learning	26
Scaffolding	28
Learning Theory	30
Design Research Studies	37
Study 1	39
Conceptual Framework	40
The “Make Your Own Game” Activity	48
Study Design & Analysis	60
Findings	68
Discussion & Implications	76
Study 2	79
Research Questions	80
Study Design	80
Conceptual Framework	89
The AgentDesign Planning Tool	98
Study Analysis	107
Findings	113
Study 2 Closing	128
Discussion	129
Transfer from Design to Programming Environment	129
Implications for CS and Informal Learning	131
Implications for the “Make Your Own Game” Activity	133
Implications for Programming Environments	135
Implications for the Design Scaffolding Tool	135
Preparing Computer Science Elementary Teachers	141
References	143
Appendix A	148

Coded Samples of Student Design Summaries	148
Appendix B	150
The AgentCubes Online Programming Environment.....	150

List of Tables

Table 1. Examples of Computational Thinking Patterns	11
Table 2. Study 1 Conjectures	40
Table 3. Practices for designing a video game using the <i>planning document</i>	53
Table 4. School demographic information.....	61
Table 5. Design practices coding scheme	63
Table 6. Example of coded design data with accompanying transcript excerpts.....	65
Table 7. Group 1 coding results	69
Table 8. Group 2 coding results	70
Table 9. Group 3 coding results	72
Table 10. Group 4 coding results	73
Table 11. Study 2 Conjectures	90
Table 12. AgentDesign <i>Planning Tool</i> Design Practices	99
Table 13. AgentDesign <i>Planning Tool</i> Sections and Design Practices.....	99
Table 14. AgentDesign <i>Design Practices</i> Coding Scheme	109
Table 15. Counting Categories for Student Design Summaries	110
Table 16. Sample of Raw Interaction Counting Categories of Students' Design Summaries	111
Table 17. Student Design Practices using AgentDesign	114
Table 18. Correctness of all Students' Designs using AgentDesign	115
Table 19. Individual Counts of the Students Identification of Agents using AgentDesign	117
Table 20. Individual Counts of each Student's Correct Identification of Interactions using AgentDesign.....	120
Table 21. Individual Counts of the Students Correct Use of CTPs using AgentDesign	122
Table 22. Transfer of the Students' Design to the Programming Environment	123

List of Figures

Figure 1. Nature of the computer science discipline	3
Figure 2. Top-down design process	5
Figure 3. The Formative Assessment Model.....	30
Figure 4. Vygotsky's Theory of Concept Formation and the Iceberg Model	32
Figure 5: Pilot Study Conjecture Map	42
Figure 6: Pencil-and-Paper <i>Planning Document</i>	44
Figure 7. 5th Dimension Activity System	47
Figure 8. AgentSheets Game Board	49
Figure 9. AgentSheets Gallery, Depiction Editor, & Behavior Editor	51
Figure 10. Planning Document Pages and Work Flow	54
Figure 11. Project Description.....	54
Figure 12. Agents	55
Figure 13. Game Board	56
Figure 14. Patterns.....	57
Figure 15. The Game activity process in relation to the <i>planning document</i>	59
Figure 16. Group 1 timeline of planning process.....	70
Figure 17. Group 2 timeline of planning process.....	71
Figure 18. Group 3 timeline of planning process.....	72
Figure 19. Group 4 timeline of planning process.....	73
Figure 20. Refined Vygotsky's Theory of Concept Formation and the Iceberg Model	76
Figure 21. Areas needing improved scaffolding for the Game activity	79
Figure 22. Focus of Pathway Beginner Level for Game Activity Process	85
Figure 23. iRemix Pathway - Beginner Level.....	86

Figure 24. Focus of Pathway Advanced Level for Game Activity Process.....	86
Figure 25. iRemix Pathway - Advanced Level	87
Figure 26. Focus of Pathway Expert Level for Game Activity Process	88
Figure 27. iRemix Pathway - Expert Level	89
Figure 28. Study 2 Conjecture Map	91
Figure 29. Disconnected Learning in Study 1.....	93
Figure 30. AgentDesign <i>Planning Tool</i> - Project Description	100
Figure 31. AgentDesign <i>Planning Tool</i> - Agents	101
Figure 32. AgentDesign <i>Planning Tool</i> - Selecting Interactions Window.....	103
Figure 33. AgentDesign <i>Planning Tool</i> - Building Interactions Window	104
Figure 34. AgentDesign <i>Planning Tool</i> - Summary.....	106
Figure 35. Comparison of Design Practices for Studies 1 & 2.....	113
Figure 36. Counts of Identified and Non-Identified Agents in AgentDesign Summaries for All Students	116
Figure 37. Students' Identification of Agents using AgentDesign.....	117
Figure 38. Counts of Alignment between Interactions and Descriptions in AgentDesign Summaries.....	118
Figure 39. Students' Interaction Correctness using AgentDesign.....	119
Figure 40. Counts of CTP Alignment with Descriptions in AgentDesign Summaries	121
Figure 41. Students' CTP Alignment with Interactions and Descriptions using AgentDesign	122
Figure 42. Areas needing improved scaffolding after study 2	130
Figure 43. AgentCubes Online programming environment.....	150

Introduction

The purpose of this design research study was to develop an effective method to engage elementary students in computer science (CS) through video game creation. This was done through an activity where students designed, created, and tested games based on their own experiences and ideas in an after school setting.

The project this study is based on was an activity at El Pueblo Mágico (EPM), a 5th Dimension after school site, and was an effort to incorporate CS into elementary grade-level activities. The activity, called the “Make Your Own Game” activity, was introduced by having students create video games. Initially this was done through students following a tutorial to create the game Frogger, but was transitioned to students *designing* and *creating* their own games. The games were initially created at EPM through the use of a *planning document* created by Middle School teachers, that were part of the general Scalable Game Design projectⁱ, and modified by me, the researcher. My goal for the activity was to help the students to be successful in creating a game they designed while being introduced to the CS discipline.

This dissertation is divided into two parts. Part one presents the results of a pilot study, of which I was trying to understand what elementary students were capable of when creating video games. The research questions I proposed for the pilot study were:

1. What are the students’ behaviors when trying to create video games using an agent-based programming environment?
2. How do the students’ behaviors align with the intended “Make Your Own Game” activity process?

The findings from the pilot study were that the activity, particularly the document that scaffolded the students’ designing of games, did not work as intended. This breakdown resulted in a

new intervention for the activity where the document scaffolding design was transitioned to a web application.

In the following sections, I provide a literature review that elaborates on the terms CS, design, algorithms, and programming languages; provides an overview of CS Education; explains why I chose to focus on elementary grade-level students for this study; and a discussion of relevant learning theory. I then present the results of the pilot study (study 1) and follow-up study (study 2) using the intervention of a computer-assisted *planning tool* called AgentDesign that I created.

Literature Review

Goals of Computer Science

CS is a discipline that uses computational theories and tools to solve problems. One disciplinary practice that computer scientists employ is to *create* software to address problems. The creation aspect of problem solving in CS requires the use of design, which is a thoughtful way of approaching the creating, testing, and releasing of software. Tools that computer scientists use to design and then create software are programming languages and algorithms (Figure 1).

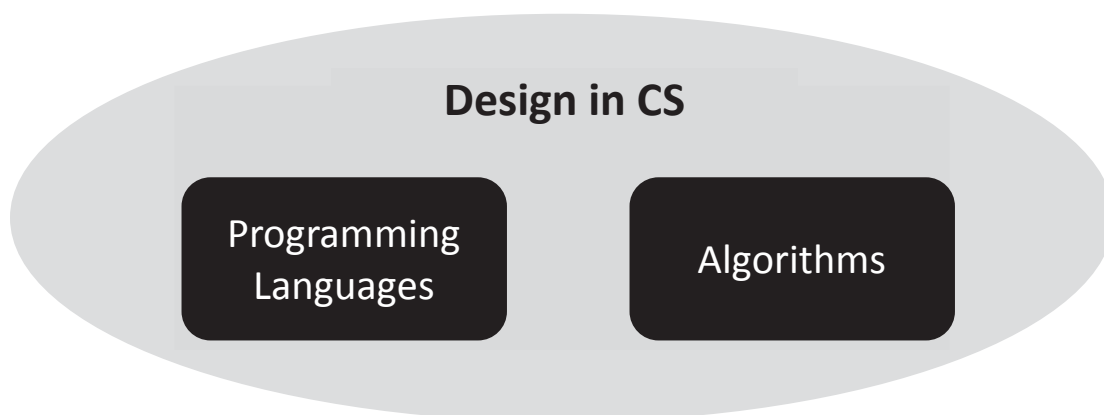


Figure 1. Nature of the computer science discipline

Programming languages and algorithms used together allow for very complex software to be made. But simply knowing many languages and their rules, or memorizing algorithms, is not enough to participate in CS disciplinary practices. Choices must be made by a computer scientist about what language to use and how the program can be structured using algorithms to create an acceptable solution. These choices are *design* choices.

Design

Design is a process through which things (software, buildings, cars, etc.) are created. The process of design initially starts with an analysis of the problem that the design is trying to address. Prior knowledge and experience are leveraged in an analysis of the problem. This analysis can reduce the design to smaller, and easier to handle, tasks. The choices made during the identification of smaller tasks require an understanding of how the smaller tasks can be carried out and how they will work together in the design as a whole. For the smaller tasks, some will be able to use existing solutions and

others will demand that new solutions be developed. The power in being methodical about the design process is that existing solutions can be leveraged and needed solutions will not be so complex.

There are two simple ways to approach design. One is *top-down design* and the other is *bottom-up design*. Top-down design is what most experts use when creating software (Cross, 2004) and is represented in Figure 2. This approach is very methodical and is meant to lead to an optimal solution by examining the problem and decomposing it in order to establish the problem's structure (Ho, 2001) and then using that structure to develop solutions. Bottom-up design involves addressing design issues as they arise in the process of creating software (Cross, 2004). This approach may lead to a similar product being created by the designer, but is also likely to take longer to develop. These two design practices are discussed further in the following sections.

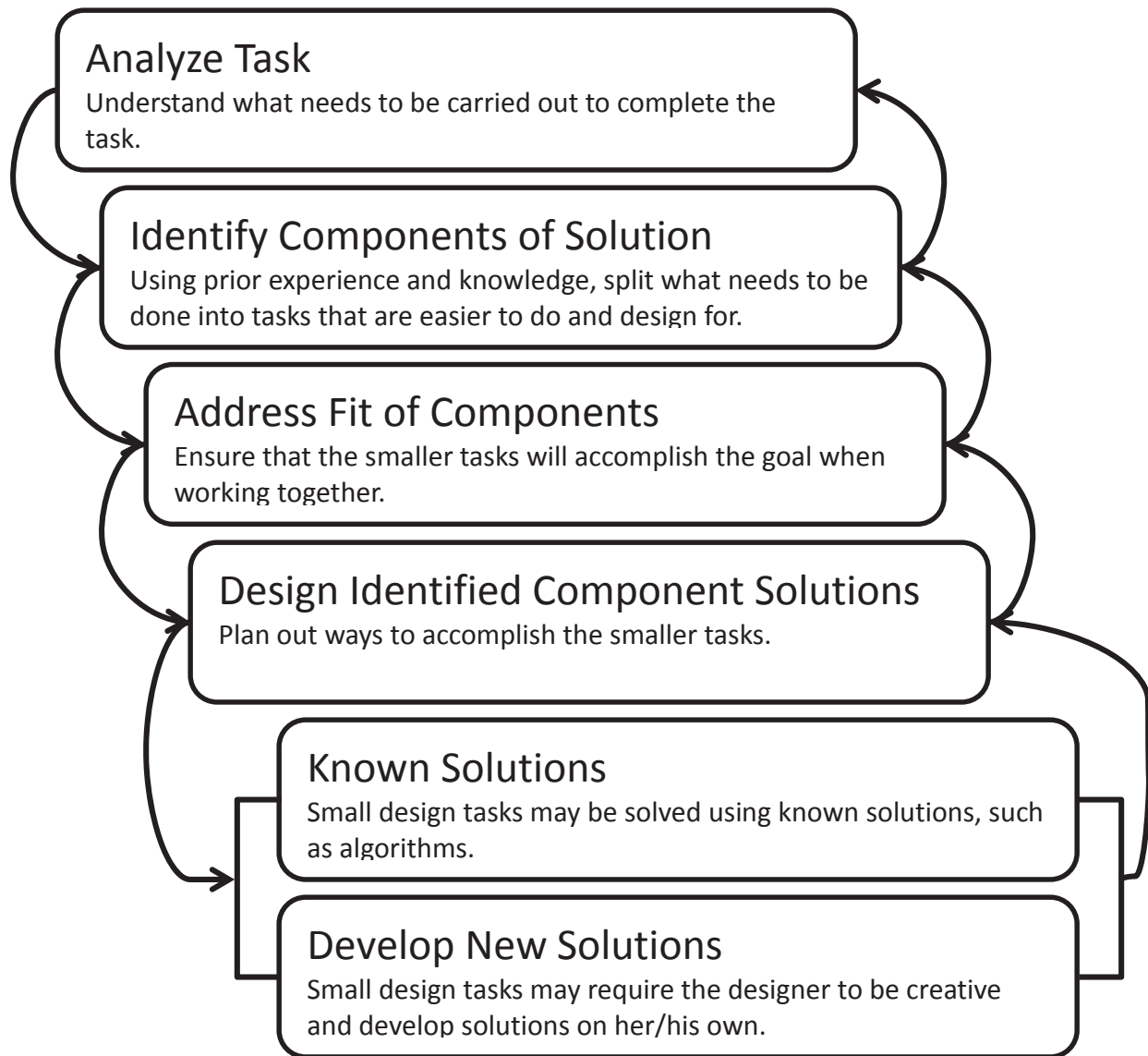


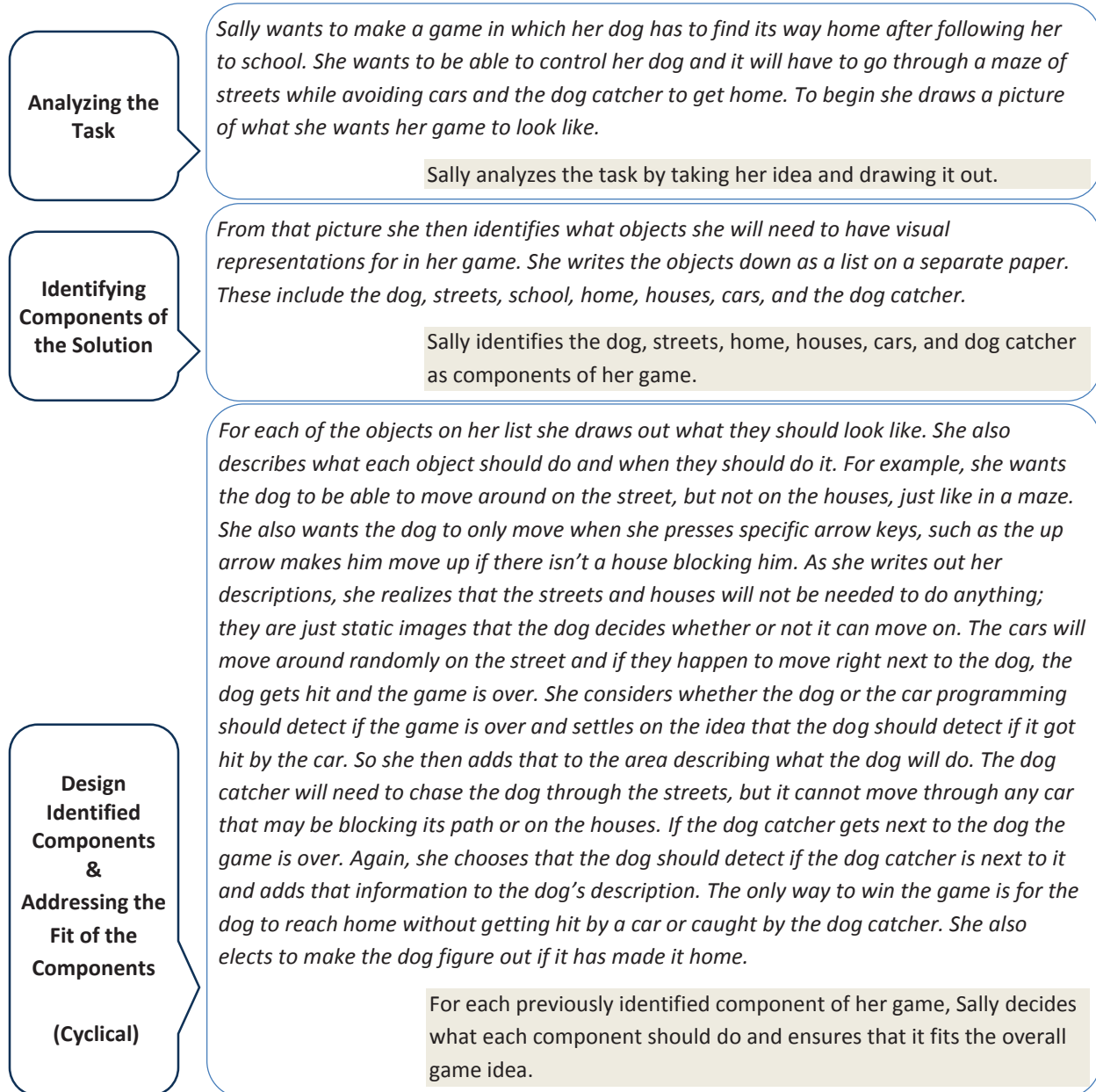
Figure 2. Top-down design process

Top-Down Design

Figure 2 represents a *top-down* design process. When a computer scientist is using a top-down design process she is trying to find an optimal solution for the problem through problem scoping and information gathering *before* she attempts to create it in a programming environment. This type of approach is most closely associated with how experts solve problems (Atman et al., 2007; Cross, 2004). The advantage of the top-down approach is that a computer scientist will have a good idea of what needs to be done to solve the problem so she can be efficient during the creation process. Top-down design also provides the best opportunity for the final program to be efficient in the way it runs. The

disadvantage of this process is that the computer scientist will need to rely on prior knowledge to guide the design, and if her prior knowledge is limited in this area she will need to be provided support materials that can disaggregate her ideas into smaller and approachable tasks. The following is a narrative for how this process may play out for a student creating a game of her own design using a top-down approach for an agent-based programming environment.

TOP-DOWN NARRATIVE:



Using Known Solutions

Next, Sally figures out if she knows any algorithms that will help her program the behavior she had described earlier for each component. She knows how to detect if two things are next to one another, so she can write the code for winning and losing. She also knows how to program movement using the arrow keys and how to make the object only move on certain types of ground. For her game she wants the dog to only move onto places that are the street, and not the houses.

Using an algorithm for collision detection that Sally already knows, she can write the code for the winning/losing situations. She also knows how to use key input to control the dog's movement.

Identifying Where New Solutions Will Need to be Developed

The two things she does not know how to program for her game are making the cars move randomly on the street and making the dog catcher chase down the dog. She decides to figure out how to do this when she gets to the point of programming the cars and dog catcher. With these two exceptions, she starts creating her game in the programming environment using everything she has planned out so far.

Sally does not know how to make things move around randomly or chase, she will have to figure these out when she is at that point in the programming.

Bottom-Up Design

In opposition to using top-down design is the bottom-up approach, and is generally associated with novices of design (Cross, 2004). A bottom-up design process involves a computer scientist immediately starting to build a solution in a programming environment and addressing design needs as they come up (Rist, 1991). This allows for the computer scientist to experiment with the programming environment and her ideas to create a solution for a piece of software. Relating the bottom-up design process to Figure 2, a computer scientist will start by addressing the smaller tasks and then move up the chain as needed.

If Sally had chosen this approach in creating her game she may have made different choices and her game would have turned out differently. While this is not necessarily a negative situation, as long as the game works the way she wants, the bottom-up approach is likely to take her longer to create a working game. The following is a narrative of the process Sally may have used to create a game of her own design using a bottom-up approach for an agent-based programming environment.

BOTTOM-UP NARRATIVE:

Analyzing the Task	<p><i>Sally wants to make a game in which her dog has to find its way home after following her to school. She wants to be able to control her dog and it will have to go through a maze of streets while avoiding cars and the dog catcher to get home.</i></p> <p>Sally wants to make a game, so she takes a minute to think about what she wants in the game.</p>
Identifying Components	<p><i>To begin she opens up her programming environment and starts making the dog, which is her main character. She draws a representation of it first.</i></p> <p>She identifies the dog as a component she will need.</p>
Using Known Solutions	<p><i>Then she programs its movement to correspond with the arrow keys.</i></p> <p>Sally knows how to make the dog move using the keys.</p>
Identifying Components	<p><i>Next, she makes the world that the dog is going to move around in. So she creates the road and draws it in the programming environment. The other basic pieces of her world environment are the houses, so she creates the houses and places them in her game playing environment.</i></p> <p>Sally identifies the road and houses as necessary pieces.</p>
Addressing the Fit of the Components	<p><i>At this point the game is starting to look like a game. The game play environment has houses, roads, and there is one dog. Sally decides to see how everything looks when she moves her dog around and notices that it goes everywhere, including over the houses.</i></p> <p>Sally tests her game to see if it's working correctly so far.</p>
Design Identified Components & Using Known Solutions	<p><i>This is not what she wanted, so she goes back and programs the dog to not be able to move on the houses.</i></p> <p>Sally's game did not work correctly, she has to redesign the dog to not walk on the houses and then reprogram it.</p>
Identifying Components & Design Identified Components	<p><i>Next, Sally makes her cars. She wants them to move around randomly on the street, but doesn't know how to do that.</i></p> <p>Sally identifies that she needs cars and designs how they will move.</p>
Developing New Solution	<p><i>Just to get started, she draws out a representation of her cars and places them in the game play environment. Then she starts to create the programming and does testing as she goes. At first, she makes the car move right using a 50% probability, and then move left anytime it doesn't move right. It takes her a while, but she eventually figures out how to make the car randomly choose a direction to move towards while staying on the road.</i></p> <p>Since Sally doesn't know how to enact her design of the cars moving randomly, she has to develop a new solution.</p>

Addressing the Fit of the Components	<p><i>Now Sally has her dog, houses, roads, and cars. She decides to test out her game so far and finds that the cars hitting the dog don't end the game. She forgot to program that behavior.</i></p> <p>Sally tests if the game is working the way she wants so far.</p>
Design Identified Components & Using Known Solutions	<p><i>To make the game end in this case, she programs the dog to end the game when it is next to a car. She knows how to do this, so it was an easy issue to fix.</i></p> <p>Sally designs the dog to detect a game ending situation, and since she knows how to do collision detection, she programs it.</p>
Identifying Components	<p><i>The next piece Sally wants to add is a dog catcher.</i></p> <p>Sally identifies the dog catcher as an object she needs to add.</p>
Design Identified Components	<p><i>The game needs to end if the dog catcher gets to the dog. She wants the dog catcher to track down the dog, so the player has to pick a path that will avoid the dog catcher. However, she doesn't know how to do this.</i></p> <p>Sally decides that the dog catcher should track down the dog.</p>
Developing New Solution	<p><i>Tracking is something Sally has seen other people do, so she looks up the programming in other games and uses that to figure out how to make that happen in her game. It takes some experimenting, but she finally gets it to work.</i></p> <p>Sally has to learn how to make tracking happen in her game, since she doesn't already know how to do the programming.</p>
Addressing the Fit of the Components	<p><i>Sally tests her game again and finds that when the dog catcher gets to the dog nothing happens.</i></p> <p>Sally tests the game now that the dog catcher has been programmed.</p>
Using Known Solutions	<p><i>So she programs the dog to end the game when it meets the dog catcher, just like with the cars. She then tests it again and it works.</i></p> <p>Sally uses code that she already knows to make the dog to detect when the dog catcher is next to it and end the game.</p>
Identifying Components & Design Identified Components	<p><i>Finally, Sally is ready to program the winning scenario, which is when the dog gets home. She knows how to do this so she programs the dog to check if it has reached the home, except she realizes that she forgot to make the home.</i></p> <p>Sally identifies that she needs to make the home and designs that the game should end when the dog gets to the home.</p>
Using Known Solutions	<p><i>She quickly makes the home, programs the dog, and tests to make sure it's all working. The programming works and the game is complete.</i></p> <p>Sally knows how to program collision detection, so she makes the dog end the game if it is at home.</p>

In using a bottom-up design process, Sally needed to program and then test over time. This process became very messy, and resulted in Sally commonly needing to go back and reprogram objects

after testing. By not being deliberate about what she thought was needed for her game to work, she missed things and had to go back and redesign components of her game. While the bottom-up design approach allows for experimentation, it also increased the amount of work Sally needed to go through as she learned of things she forgot to do. The resulting program from the bottom-up design process is harder to decipher and usually patched together rather than being an easily understood organization of thoughts and code.

This section explained two types of design, top-down and bottom-up, and gave examples of how each type may be used to create a game. While either type of design practice may lead to the same result, top-down design encourages a more thoughtful process and is likely to lead to a successful and more enjoyable outcome. The goal of CS is for students to learn to be thoughtful about what they are creating, meaning they should practice using top-down design, so that the product is created efficiently and is effective.

Programming Languages

Within CS, programming languages provide a means through which a solution to a problem can be expressed. They are the mediators for communicating with a computer to make it do what the computer scientist wants.

Programming languages provide both constraints and affordances for the development of a solution to a problem. They are almost always strictly defined, but that strictness provides a structured environment within which a computer scientist can work. In any given situation where a problem needs to be solved, these constraints and affordances must be accounted for to not only choose a language that will allow for a solution to be created, but also one for which the solution will be efficient.

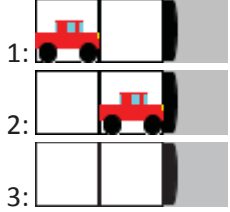
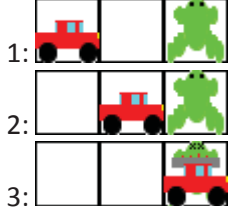
Algorithms

Algorithms allow computer scientists to focus on the conceptual solution to a problem instead of the implementation of the solution (Solomon, 2007). They also provide efficient and well known solutions to certain problems in CS. If a large problem can be analyzed and broken down in a way where

there are known solutions to the smaller pieces, then algorithms become a huge help. For example, problems that require the use of mathematical formulas, such as numerical integration, can utilize known algorithms for those portions of the solution. Computer scientists are able to extend their capabilities using algorithms that were developed and tested by others.

One way to introduce algorithms to novice students is to leverage the algorithmic aspects of *Computational Thinking Patterns (CTPs)* (A. Basawapatna, Koh, Repenning, Webb, & Marshall, 2011). CTPs are, “abstract programming patterns that enable agent interactions not only in games but also in science simulations,” (p. 245). Another way to conceptualize CTPs is that they are common phenomena that occur within games and simulations that are recognizable to users/players. The CTPs used in this dissertation are generally limited to spatially oriented environments. An example is the *collision* CTP, which occurs when two agents collide and cause something else to happen within the game or simulations. The phenomenon of two agents colliding to cause an effect is very recognizable within gaming, and so CTPs become useful because the underlying idea behind each pattern is generally understood for any novice programmer. Useful CTPs include, but are not limited to, the list in Table 1, which provides a description, visual example, and pseudocode example for each.

Table 1. Examples of Computational Thinking Patterns

Description	Visual Example	Pseudocode Example
<p><i>Absorption:</i> This pattern can be conceptualized as the meeting of two agents that results in an agent being erased.</p>		<p>IF the car sees a tunnel to its right</p> <p>THEN the car gets erased</p>
<p><i>Collision:</i> The collision pattern occurs when two agents have a specific, and close, spatial relationship that results in something happening in the game.</p>		<p>IF the car is on top of the frog</p> <p>THEN the frog dies</p>

<p>Generation: The generation pattern occurs when one agent creates another agent while the game is being played.</p>		<p>IF one second has passed</p> <p>THEN make a new truck to the right of the tunnel</p>
<p>Push: The push pattern has one agent push another agent in a specific direction.</p>		<p>IF the pusher wants to move right AND there is a box to the right</p> <p>THEN the box moves to the right AND the pusher moves right</p>
<p>Transport: The transport pattern occurs when one agent carries another agent. Transporting also implies movement, so the carrying agent will both move and carry whatever agents are above it in the game.</p>		<p>IF the frog is stacked above the log</p> <p>THEN the log moves to the right AND the frog moves to the right</p>

As you may have noticed in Table 1, each CTP has a set of operations to enact it within a programming environment. In the examples shown, these operations are specific to the specific agents, but they can also be generalized. This algorithmic nature of CTPs, along with the idea that CTPs are recognizable phenomena, is what I intend to leverage within this dissertation to help students develop an understanding of what algorithms are and how they can be used.

This section defined and discussed the discipline of CS and its purpose of creating solutions to problems using computing. The three major principles that students of CS need to learn are (1) design practices, (2) programming languages, and (3) algorithms. For each of the principles, the learners will need to be supported in a way that their current developmental skills can be extended to a maximum level. Vygotsky (1978) calls the difference between what a student can do on her own and what she can

do with help the *zone of proximal development (ZPD)*. Using a scaffolded tool to help the students do design beyond telling a story can address the design principle. Requiring game creation through the use of an accessible agent-based programming environment that makes writing instructions easy can address the programming languages principle. By framing algorithms as CTPs, which will be phenomenon recognizable to novices, the algorithms principle can be addressed and learned. However, given that most novices will have had prior experience with telling stories and following/giving instructions, but not necessarily with the idea that common phenomena in programs can be considered algorithms, helping students learn the algorithms principle will likely be the most difficult.

The Broken Pipeline of CS Education

CS education has fallen short of recruiting and retaining students into the field, and in the case of non-white, non-male groups it has essentially failed. Women are grossly underrepresented in the discipline (Frenkel, 1990; Montanelli Jr & Mamrak, 1976; Pearl et al., 1990; Todd, Mardis, & Wyatt, 2005), as are minorities (Aspray & Bernat, 2000; Stockard, Klassen, & Akbari, 2004), and this underrepresentation occurs from K-12 to working professionals (ACM Education Policy Committee, 2014; Frenkel, 1990). Within CS education, this phenomenon is often called the *broken pipeline* and it occurs from people either having a lack of access to CS or choosing to stop studying CS when they do have access.

Many projects and studies over the years have elected to address the problem of people from underrepresented groups choosing to leave CS. Much of this work has been done at the post-secondary level and is not simply a matter of encouraging interest. Many women tend to leave the CS pipeline due to an environment that is not only unwelcoming to them, but can also be considered hostile (Teague, 2002). While many women have a high interest in working as professionals in CS, the male dominated culture can easily push them away (Binkerd & Moore, 2002). Binkerd & Moore (2002) suggest that to address the problem of pushing students out, universities and faculty must become aware of and remedy the ways in which they interact with students, especially women.

Another way that educators and researchers have worked to fix the CS pipeline problem is to provide better access to groups that are not typically exposed to CS at the K-12 level. Many of these access points are in the form of after school programs for students and schools that do not offer a CS curriculum. These after school programs seek to expose students to CS in a way where they feel successful and see a CS profession as a legitimate possibility for their future (Denner, Werner, & Ortiz, 2012; Sivilotti & Laugel, 2008). A limitation of the after school approach is that it targets a small portion of the underrepresented groups, and the students that do participate typically self-select into the programs. Ideally, all students would be given an opportunity to learn about CS in their K-12 classrooms and know what their options are when it comes time to choose a career.

A recent development in CS education has been the creation and implementation of the AP Computer Science Principles course. The purpose of this course is to promote, “deep learning of computational content, [develop] computational thinking skills, and [engage] students in the creative aspects of the field,” (p. 1, College Board, 2014). Unlike the previous AP CS course, the Computer Science Principles course is not programming-centric, although it does incorporate programming into the students’ activities. Instead, this new course focuses on helping students develop six computational thinking practices and learning about seven big conceptual ideas of CS. These computational thinking practices and big ideas are intended to provide students with a more authentic disciplinary experience. In regards to the six computational thinking practices, students should develop the abilities to:

- connect computing to everyday life and understanding it’s impact on society,
- create computational artifacts with a purpose and using proper programming practices,
- identify and/or using data abstractly for modeling,
- analyze problems or artifacts to develop or evaluate solutions,
- communicate problems, solutions, and analysis using appropriate methods for a given context,

- and effectively collaborate with others to solve computational problems.

And the seven big conceptual ideas that the course focuses on are that:

- “Computing is a creative activity,” (p. 6).
- “Abstraction reduces information and detail to facilitate focus on relevant concepts,” (p. 8).
- “Data and information facilitate the creation of knowledge,” (p. 13).
- “Algorithms are used to develop and express solutions to computational problems,” (p. 16).
- “Programming enables problem solving, human expression, and creation of knowledge,” (p. 20).
- “The Internet pervades modern computing,” (p. 25).
- “Computing has global impact,” (p. 28).

In comparison to what I am trying to do with elementary students, many of my goals align with this new AP course’s goals. However, its drawback is that many students may not have the opportunity to pursue taking this course by the time they are in high school, or they may simply not be interested. It is imperative that this approach be used with students at an earlier age than high school.

Some projects and organizations have pursued including CS in the general K-12 curriculum earlier than the high school level. The Computer Science Teachers Association (CSTA) released a Model Curriculum for K-12 Computer Science (Tucker et al., 2006) which suggested CS topics for students to learn as well as provided grade-level breakdowns that culminated in students taking AP CS, a projects based course, or receiving industry certification. Three notable projects and organizations have been actionable in exposing students to CS before and during high school in regular classrooms. These include **ECSITE**, **Scalable Game Design**, and **MIT Scratch**. I should note that each of these projects and organizations approach CS exposure to students in classrooms in very different ways, which are described below.

The ECSITE project incorporates computing into a broad spectrum of courses, from art to biology, by pairing non-CS K-12 teachers with CS graduate students to develop curriculum that meets standards but will also benefit from a computing approach. Their findings indicate that by introducing computer science into non-CS classrooms both teachers and K-12 students developed higher levels of interest in CS disciplinary approaches while learning CS practices (Goldberg, Grunwald, Lewis, Feld, & Hug, 2012).

Scalable Game Design trains middle school teachers to introduce CS into their classrooms through creating games, and then simulations, using an agent-based programming environment. The project also develops curriculum for the teachers to use and modify within their classrooms. Their findings show a high participation rate of girls, due the implementation in everyday classrooms, as well as a high rate of enjoyment for both boys and girls; with 85% of the female participants enjoying the activity and 78% saying they would take another game making class (A. R. Basawapatna, Koh, & Repenning, 2010; Repenning, Webb, & Ioannidou, 2010).

Scratch is a free visual programming environment and is easily accessible to novice users (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010; Resnick et al., 2009). It is incredibly popular in K-12 schools, and to date (June 27, 2015) there have been almost 10 million projects shared through the Scratch website. Through its ease of use and accessibility, Scratch has opened the door to CS for many students that would have never been given the opportunity otherwise.

As mentioned earlier, each of these groups have taken different approaches toward including CS in K-12 education. However, a common drawback of each group is that their focus has been primarily on classrooms at the middle school level or above and they haven't moved much into the realm of elementary CS education.

In earlier years of CS education research the focus was not only on helping elementary students learn about programming structures and practices, but to also understand and facilitate their abilities to

design programs (Kafai, 1996; Pea, Kurland, & Hawkins, 1985) and improve their problem solving processes using programming (Papert, 1980).

In his book “Mindstorms,” Papert (1980) describes the LOGO programming environment and how it can assist student learning. He emphasized the idea of teaching without a curriculum and letting students learn a new way of thinking through programming with LOGO. History has proven that this is not necessarily the case, as many teachers brought CS into their classrooms at the behest of his writing and using LOGO alone did not produce the implied shifts in thinking. However, in his introduction to the second edition of “Mindstorms,” Papert does acknowledge that he did not mean to push the idea that programming alone could change student thinking, but that using the LOGO programming environment could support such a shift in students. Like all new experiences for students, the activities they undergo must be supported if any positive change is to be expected, and the shift in thinking was not supported in this case from programming alone.

Kafai (1996) examined elementary students’ design practices while creating games using the LogoWriter programming environment. Her research articulates that design manifests for elementary students, given the option to create open projects, as a mix of “bricolage” and “planning.” “Bricoleurs,” approach problems as time goes on, and they redesign their solutions as they test them (i.e. building from the bottom-up). “Planners” approach problems from the top-down, meaning they consider the relevant pieces of a problem and then focus on each piece to develop a solution. While a bottom-up approach may work for completing certain design tasks, she found that it was not effective for a large task.

Pea et al. (1985) examined elementary students’ planning abilities and the transfer of those abilities outside of programming by examining 3rd/4th and 5th/6th grade classrooms over a two year period. Similar to the findings of Kafai (1996), Pea et al. (1985) found that the elementary students did not focus on planning out their programs. Instead, the students focused on the immediate needs of any

task they were trying to accomplish (they used a bottom-up approach). In examining whether or not programming improved the students' general planning abilities compared to students that did not do programming, Pea et al. (1985) found no improvement of student planning abilities outside of the programming activity after one year of experience. In year two of their study, they used a different approach to measure students' planning abilities outside of programming that was more closely associated with the practice of programming. However, their results were the same in that they did not find a definitive link between programming and planning abilities of the students they worked with. Reflecting on their results, Pea et al. (1985) believed that the teachers they had worked with did not explicitly focus on design practices enough. They found that the teachers had only begun to do this toward the end of their study.

Given the outcomes of the early work with elementary students doing CS, it seems apparent a new way of thinking is not magically developed, nor are the necessary design practices used by computer scientists. For any new student of a discipline, learning needs to be supported. Students are able to explore and learn within programming environments, but they need to be guided in ways to organize their thinking and ideas to develop those processes.

Unlike the work done in the 80s and 90s, CS education organizations have all but removed the link between programming and problem solving for elementary students. The CSTA Model Curriculum (Tucker et al., 2006) suggests that elementary students focus on skills using technology and not necessarily on the thinking that is required in CS. Other suggestions include elementary students taking up the CS Unpluggedⁱⁱ curriculum (Fletcher & Lu, 2009) in which no computers are used and students work on conceptual CS problems (Dwyer, Hill, Carpenter, Harlow, & Franklin, 2014). Despite the current popular beliefs around how to expose elementary students to CS, research is being done that pushes against the idea that elementary students focus only on technological skills or CS without a computer.

These studies, in which elementary students pursued programming tasks through scientific modeling contexts or simply learning to program, are discussed below.

Louca (2005) investigated 5th grade students' approaches to scientific modeling using two different visual programming environments, MicroWorlds Logo and Stagecast Creator. The findings by Louca showed that students used different approaches for the two programming environments. Students using MicroWorlds LOGO discussed the structure of their programs while planning in relation to the programming environment. In contrast, students using Stagecast Creator discussed the scenario they were modeling while planning, so their focus was more on the conceptual idea. These two approaches are very different and the findings imply that educators need to be aware of the implications of choosing one programming environment over another and how that choice will affect student behavior.

Sánchez-Ruíz and Jamba (2008) used a two year study to examine 4th and 5th grade students' understanding of the computer and programming. In the first year of their study, 4th grade year for the students, they used 1 hour, weekly sessions to teach about the workings of a computer. The students were given presentations and worked through activities that mimicked a computer. In year two of their study they followed the students to 5th grade and had the students learn about programming using an environment called Squeak. After being provided presentations, the students pursued programming tasks that involved drawing out shapes and letters. This involved the students using a "turtle" (cursor agent) to draw within a 2D coordinate system. Most of the students were successful in completing the activities, but were also found to dislike the "formulas, equations, and weird language," (p. 29) that were necessary to complete the tasks.

Sengupta and Farris (2012) helped 3rd and 4th grade students learn about kinematics through modeling using an agent-based programming environment called ViMAP. They found that the students

were not distracted by the aspect of doing programming, and that it considerably helped them to understand a difficult concept.

What is interesting about these three studies is that examining how students used the programming environment, or its effect on their learning, was a major focus of the findings. Since there are many options for programming environments now, understanding which ones work well for young students, and why, is important since the environment appears to influence student behavior (Louca, 2005). However, programming is only one part of CS. CS education can't constantly focus on what language or environment works well without also looking at the activities and practices that students take up. How can students be influenced in their design practices? What approaches help students to learn algorithms? Can young students learn to do more than just the technical aspect of CS (programming)? CS education needs to shift towards a problem solving approach, rather than a knowledge based approach.

Generally speaking, all 21st century citizens will be confronted with both age old problems and new problems that have never been considered. Many solutions to these problems will be developed using technology, which makes it imperative that students at all levels learn how to solve problems using technology, such as CS, to do so. For students to learn how to develop solutions, they must understand how to use design to inform their problem solving process (Fischer & Scharff, 2000; The Computer Science Teachers Association, 2012). The Next Generation Science Standards (NGSS, 2013) echo this call to action by including Engineering Design as a topic for which elementary students should be able to:

- “3-5-ETS1-1. Define a simple design problem reflecting a need or a want that includes specified criteria for success and constraints on materials, time, or cost.
- 3-5-ETS1-2. Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem.

- 3-5-ETS1-3. Plan and carry out fair tests in which variables are controlled and failure points are considered to identify aspects of a model or prototype that can be improved,” (p. 32).

These three abilities that elementary students should be able to show are considered disciplinary core ideas within NGSS. Within the grade band of 3rd through 5th grade the NGSS (2013) also acknowledges that students should be able to enact the science and engineering practices of:

- “Asking Questions and Defining Problems
- Planning and Carrying out Investigations
- Constructing Explanations and Design solutions,” (p. 32).

Also within the Next Generation Science Standards (2013), there is guidance as to how students in 3rd through 5th grade can make connections between disciplines through “cross-cutting concepts,” that are present in various fields of science, engineering, and technology. These cross-cutting concepts bridge disciplinary boundaries (Committee on Conceptual Framework for the New K-12 Science Education Standards & National Research Council, 2011) and the cross-cutting concepts that apply to the work presented here are:

- Patterns: Students recognize and learn to use common phenomena that occur in games and simulations in the form of algorithms (specifically CTPs).
- Cause and Effect: Students use design to create programs that have a desired *effect* or outcome.
- Scale: Students use measurements and scalable values within programs to determine functionality or program state.
- Systems and System Models: Within object-oriented programming languages and agent-based languages, programs function as a system of components that form a whole. Additionally, games and simulations function as a model of an observed or imagined system.

- Structure and Function: Programming environments have a rigid structure from which functionality can be built. Students must understand the structure of a programming environment to design for and enact desired functionality within that environment.

One way that CS education has recently approached design is through the notion of *computational thinking*, which can be defined as using a certain form of logical thinking to solve problems (Lu & Fletcher, 2009; Wing, 2006, 2008). An operational definition of computational thinking developed by the ISTE and CSTA (2011) states that it, “is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem solving process to a wide variety of problems.”

Computational thinking has become the focus of CS education. If students are to take up this way of approaching problems it is imperative that they learn not only about programming languages, but also the algorithms that can make programming languages powerful. Without understanding both programming languages and algorithms, it would not be feasible for students to truly think computationally as it is defined above.

In regards to design, two studies have pursued understanding how young students can do design in a CS context. One approach is for students to pursue design as a form of storytelling (Kelleher & Pausch, 2006), and another is for students to be methodical and supported in a more traditional

practice of top-down design (Robertson & Nicholson, 2007). Both of the studies discussed below focus on young students, some of which were young enough to be in elementary school, but they were not limited to only the elementary grade levels.

Kelleher & Pausch (2006) were interested in finding a method to attract more women to CS. They focused on girls that were 10-16 years old and asked them to create an animated movie for the study. Using a modified version of the Alice programming environment, called Story Alice, their activity presented, “programming as a means to the end of storytelling.” Although their research was primarily focused on motivating girls to find programming interesting through storytelling, they also offer important insights on how students can be guided to do design. In the study, the girls used a scaffolded design process by refining storyboards to develop their ideas. The researchers initially found that the girls’ first story boards were too high-level to create appropriate programming code, and had to urge the girls to be as detailed as possible in their descriptions. Another finding about the storyboarding process was that if the girls had access to the programming environment before they had finished developing their story, the girls would change their story based on a perception of what they thought would be difficult to program. Given that the primary focus of the research was motivating the girls, restricting access to the programming environment in order to build excitement about the end product makes sense. However, it is also interesting to note that when the girls were aware of the programming environment, they used that information to inform their story (design).

Robertson & Nicholson (2007) focused on young students’, ages 10 to 16, creative practices while they designed and made 3D games in both an informal and classroom context. Their main objective was to understand the process students go through to create games and then develop a scaffolded, interactive software tool to assist future students in designing and making games. The data collected included student games, design plans, and interviews discussing what the students accomplished each day of the activity. To guide the activity, the researchers theorized that the creative

process consists of six stages: *exploration, idea generation, game design, game implementation, game testing, and evaluation*. Their findings show that students in both contexts enjoyed the activity and worked very hard to complete their games. However, the classroom context, classroom and informal, did not allow for an adequate amount of exploration for the students, resulting in students having a limited understanding of the programming environment. Alternatively, the classroom context was beneficial to the students because it was favorably structured for sharing ideas and letting students learn from one another. Another finding was that students developed game ideas from three general areas: other creative products, social influences, and exploration of the game toolkit. So it is apparent that exploration, along with students' prior experiences, are important aspects of the researchers' theorized creativity process. When examining students' generation of ideas, they found that students had difficulty organizing and refining what they wanted to make. Game ideas also evolved or were discarded as the students worked with the programming environment, or because they simply forgot about them. Using the results of the research, Robertson & Nicholson outline their intended initial design of the scaffolded software they were developing. Since exploration played a prominent role in students' generating ideas and understanding the programming environment, a workspace must be available for students to play around with small aspects of programming. A separate space must also be available for students to organize their ideas and to evaluate fit and coherence of multiple ideas. The game design and game implementation aspects of the creative process must provide wizards to support common and complex tasks. Students were found to have scaled back their designs from initial ideas because they weren't able to find a solution with their limited experience and needed support. The design and implementation supports must be able to guide students in a way that they do not become frustrated or disappointed with their product. To support the testing phase, students need a list of things to check for in their game as well as support to fix buggy aspects of their programming. The final aspect of the software is that students need a simple way of accessing and addressing feedback

(evaluation in the creative process). Allowing students to interpret and decide what is, or isn't, acceptable feedback will push against the idea that there is a *correct* way for them to have made a game and will give the students independence and final ownership of their product.

From these two studies there are two important takeaways. One is that knowledge of the programming environment can influence design, and the resultant program, in both positive and negative ways. If students are aware of how difficult certain functionality may be to program, they will not design the game they may really want to make. In contrast, students will use the knowledge that they have of the programming environment to inform their design, which is an important CS practice. A balance needs to be struck so that students can have grand ideas for their design, but also be supported when it comes to turning that design into a working product. The second takeaway is that students need to be pushed to think through their designs. Both studies shared that students did not initially develop their designs well enough that they could be used to help with the programming phase. Supporting the coordination of ideas and refining the design is necessary for novice programmers, and both studies indicate that this can be done through a well-designed activity.

Up to this point in the literature review, I have elaborated on how I define the goals of CS and the ways that design, algorithms, and programming languages are used and connected within the discipline. I also discussed the literature around the “broken pipeline” of the CS profession by outlining the historic and current issues with the overall lack of diversity of CS students and professionals. Additionally, I have discussed the work of various groups to improve both the quality and accessibility of CS education to K-12 students, and noted that much of this work is being done above the elementary grade levels. Finally, I discussed how CS education is beginning to focus on helping students to develop design practices, and then relating those practices to programming. In the following section I will discuss literature covering student engagement and learning and how it pertains to the two studies presented in this dissertation.

Engagement & Learning

Engagement and Classroom Context

Any activity designed to assist students in learning must be both engaging and effective for producing desired learning outcomes. This section presents literature discussing student engagement and theories of learning. The work of Stipek (1996) and Ames (1992) is used to describe how instructional strategies and environmental structures are integrated to influence student motivation and engagement. Although this work is often not associated with sociocultural theories of learning, the term engagement is used instead of motivation to point to the sociocultural applications of this work. In this section, theories of learning are also discussed with special attention to scaffolding, formative assessment, Vygotsky's Theory of Concept Formation, and the Freudenthal Iceberg model.

Student engagement is influenced by learning contexts that can position students in terms of their performance on various tasks versus in terms of their engagement within the task and within the classroom community (Dweck, 1986). How the goal of a task is perceived by both the student and the teacher can influence student engagement. Goals that encourage students to perform a task for an unrelated reward or social status are not as effective for learning as goals that encourage mastery of a task (Ames, 1992). Performance goals tend to draw the students' focus toward the reward or completion of the task, which then draws the focus away from what is being learned. Mastery goals connect the reward to the learning being done through both implementing and completing the task itself. Both Stipek (1996) and Ames (1992) discuss practical strategies for encouraging mastery of tasks below.

Designing an activity in a way that students perceive effort as a legitimate path to success has been shown to be very effective (Ames, 1992; Stipek, 1996). Student engagement is tied to a belief that success is possible (Stipek, 1996). Stipek (1996) discusses the, "use of rewards," "the nature of tasks," and the, "criteria for evaluation," (p. 105) as influential instructional practices of student engagement.

The relationship of these three practices within a learning context is important for any educator to take

into account. She emphasizes that rewards should be used as little as possible and focus on the informational, rather than the controlling, purpose of the reward. She also states that students should recognize that rewards can be earned through effort towards the mastery of a task and that the perceived causality of rewards should be directed away from an external cause. Stipek (1996) says that tasks should be moderately difficult (achievable through effort), vary in format and nature, and be personally meaningful. It is also important that students be provided choice in their tasks, which allows for the development of intrinsic motivation for completing the task and gives the student more ownership over the result. The evaluation aspect of instructional practices discussed by Stipek (1996) encourages the ability for students to self-evaluate their progress and mastery of tasks, and that the focus of evaluation be on mastery and not social comparisons (competition). In Stipek's (1996) view, designing instructional strategies using these suggestions will encourage students to have a, "perception of self-determination," and, "feelings of mastery and competence," (p. 102). The work discussed in this dissertation uses the information provided by Stipek (1996) by building opportunities for success through effort, choice in the type of game the students can make, and the ability to self-evaluate their progress as they work through the game making activity.

When students feel that a particular ability or skill is necessary to solve a difficult problem they are less likely to try to solve it, especially if they do not feel that they have that skill or ability. Ames (1992) calls an effort based evaluation for success a *mastery goal orientation*, and it is associated with motivation. She focuses on the use of instructional strategies for the *task, authority, and evaluation* aspects of a classroom structure to distinguish between performance and mastery goals. For mastery goals, strategies for the *task* structure should focus on the activity being meaningful and reasonably challenging. The *task* structure should also help students establish goals that are short-term and self-referenced, as well as support students in developing and using effective learning strategies. The structure of *authority* for the learning environment should allow for students to have decision making

abilities, develop responsibility and independence, and develop and use self-managing skills. The *evaluation* aspect of the learning environment should allow for students to self-monitor their improvement and mastery in a private manner, provide opportunities for improvement, and perceive mistakes as part of the learning process. Through structuring the learning environment and activity in this manner students can focus on their own effort and learning, be engaged, and have an ability to tolerate failure. The studies presented take into account Ames' (1992) suggestions for setting up the task, authority, and evaluation aspects of the activity. Students are encouraged to develop a mastery goal orientation by being provided short-term goals while designing and creating games, having ownership over all decisions for the game design, and being able to self-monitor their progress and learning while working through the activity.

High student engagement is necessary due to the self-directed nature of the activities used for the studies presented in this dissertation. However, while students are engaged, they must also be supported in their learning throughout the activity. This support is accomplished through the use of scaffolding.

Scaffolding

Employing scaffolding (and associated learning theories) to support student learning and ability to accomplish a task is critical in learning environments, especially if it is enacted in a way that students can use their own knowledge and skills in a way leading to mastery goals. Vygotsky (1978) refers to the Zone of Proximal Development (ZPD) as the "distance" between what a student can do on her own, and what she can do with help, and it is a very important concept when discussing scaffolding. If a student is supported (by others or by tools) in a way that she is working on tasks that would be difficult for her to accomplish on her own, but can still be successful, she will develop skills necessary to do this work on her own. Utilizing the theory of the ZPD, instructional scaffolds are put into place for individual students in a way that takes into account the individual student's capabilities and the activity's short and longer term goals. The goal of scaffolding is to help students extend their current capabilities to a point where

the scaffold is not needed. At the point where the scaffolding is no longer needed it is removed, or “faded,” as the student learns and gains skills (McNeill, Lizotte, Krajcik, & Marx, 2006). However, scaffolding that isn’t removed, and has a constant presence in the learning environment, can also be very useful. White et al. (2002) used a constant form of scaffolding to help students learn to process information and become metacognitive about their learning and what it means to do inquiry. This non-faded form of scaffolding was done using computer support, with the goal being to develop a practice amongst the students. Using scaffolding in this way, students do not need to interact as often with a teacher but are still given the support to be independent, motivated, and successful while working on an activity.

The scaffolding process relies heavily on the knowledge and experiences that students bring to the learning environment and builds on these over time. Using student knowledge and experience to inform instruction is aligned with the formative assessment model (Black & Wiliam, 2009), which promotes the idea that the teacher first assesses where students are coming from in terms of their experiences, knowledge, and practices. Then, with expectations of where students should be going to, based on activity goals, the teacher determines which activities can help them get there (Figure 3). Expectations for the participants in the studies presented here are that the students will have experiences playing video games and may have also created a video game before. These students will have many experiences on which to draw and will likely want to recreate games they have seen, or make innovations to games with which they are familiar. Through the development of a video game, together with scaffolding, it is expected that the students using will begin to develop principles that are involved with CS, including algorithms (as CTPs) and design.

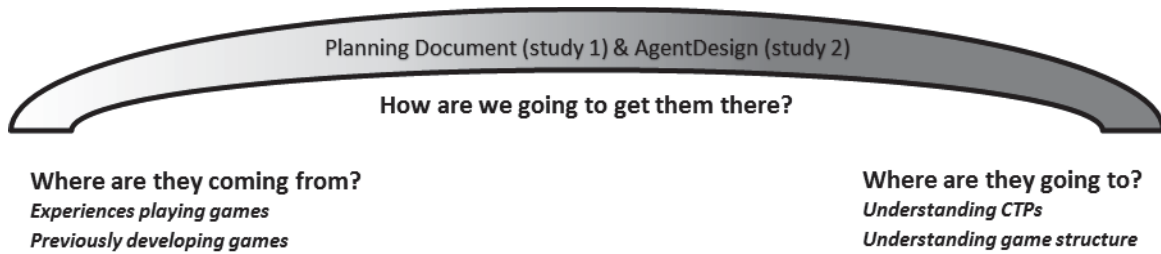


Figure 3. The Formative Assessment Model

Scaffolding is provided by pencil-and-paper *planning documents* in study 1 and an online *planning tool* called AgentDesign in study 2.

Learning Theory

Vygotsky's theory of concept formation (Vygotsky, 1986), from which the idea of the ZPD was developed, operates under the notion that words, tools, and symbols mediate thinking. How people develop a concept is guided by their cultural and historical interactions and activities around that concept, and over time that concept is refined as more experiences are gained that influence its meaning. Following Vygotsky, Otero & Nathan (2008) distinguish between a person's experience-based concepts (drawn informally from their personal experiences in the world) and academic concepts (principles and practices that are a part of a broader community, such as CS or physics). The learning process then consists of scaffolding that facilitates students in the process of abstracting experience-based concepts from the specific concrete episodes to which they are tied and, at the same time, academic concepts (typically introduced through schooling) being introduced and increasingly tied to concrete experiences, specifically to elements of experience that are being abstracted from the complex experiences of the learner. These processes mediate one another, and are mediated through classroom instruction and other forms of scaffolding. Vygotsky's theory of concept formation makes clear that the learning process is not complete until various ideas are formalized and tied to experiences, thus being consistent both with, in this case, the computer science community and with the students' everyday and classroom experiences.

Nasir, Hand, and Taylor (2008) and others compare “practice-linked mathematical knowledge” to “school-linked mathematical knowledge.” They discuss differences in the way people express their mathematical knowledge when applied to everyday versus school-based situations. They argue that sometimes students’ mathematical performance is stronger when they use unconventional methods to solve problems in out of school contexts such as the basketball court. They encourage readers to value both types of knowledge but do not discuss the mediating process that schooling can play in mediating between everyday and formal knowledge structures. Nasir, Hand, and Taylor’s work reveals the dangers of instruction that focuses solely on disciplinary principles and fails to provide scaffolding to make constant connections to students’ knowledge and experiences.

The *iceberg* model used by the Freudenthal Institute for Science and Mathematics Education (Doorman & Gravemeijer, 2009; Webb, Boswinkel, & Dekker, 2008) emphasizes how informal, experiential knowledge mediates and is mediated by formal concepts, symbols, and terminology. The name, *iceberg* model, is a metaphor used to present the idea that conceptual development is processed through a collection of representations. The focus of conceptual development is always on the most formal representation, meaning the tip of the *iceberg*. However, all of the informal and pre-formal representations of that concept are always under the water, supporting the *iceberg*’s ability to float. When a new context of use for the formal concept is given, those informal and pre-formal representations may be called upon again to further refine the formal concept. In the *iceberg* model, students will rely on many pieces of knowledge and experiences in order to develop a formal concept. For example, if a student is learning about long division she may begin to understand this concept by utilizing what she understands about grouping. A conceptual understanding of long division will not occur if the student does not have something real and concrete to relate to in order to abstract general rules or principles. Likewise, a student learning about algorithms, such as CTPs, and design will need concrete experiences using those algorithms and doing design. Students creating games need to be

given access to examples where they can see programming code, its relationship to overarching algorithms like CTPs, and the relationship of both to the game being developed.

Figure 4 combines the ideas of Vygotsky’s theory of concept formation and the *iceberg* model to emphasize the relationship of the experiences and knowledge that a person may have to how they will influence that person’s development of a formal concept/context-free principle.

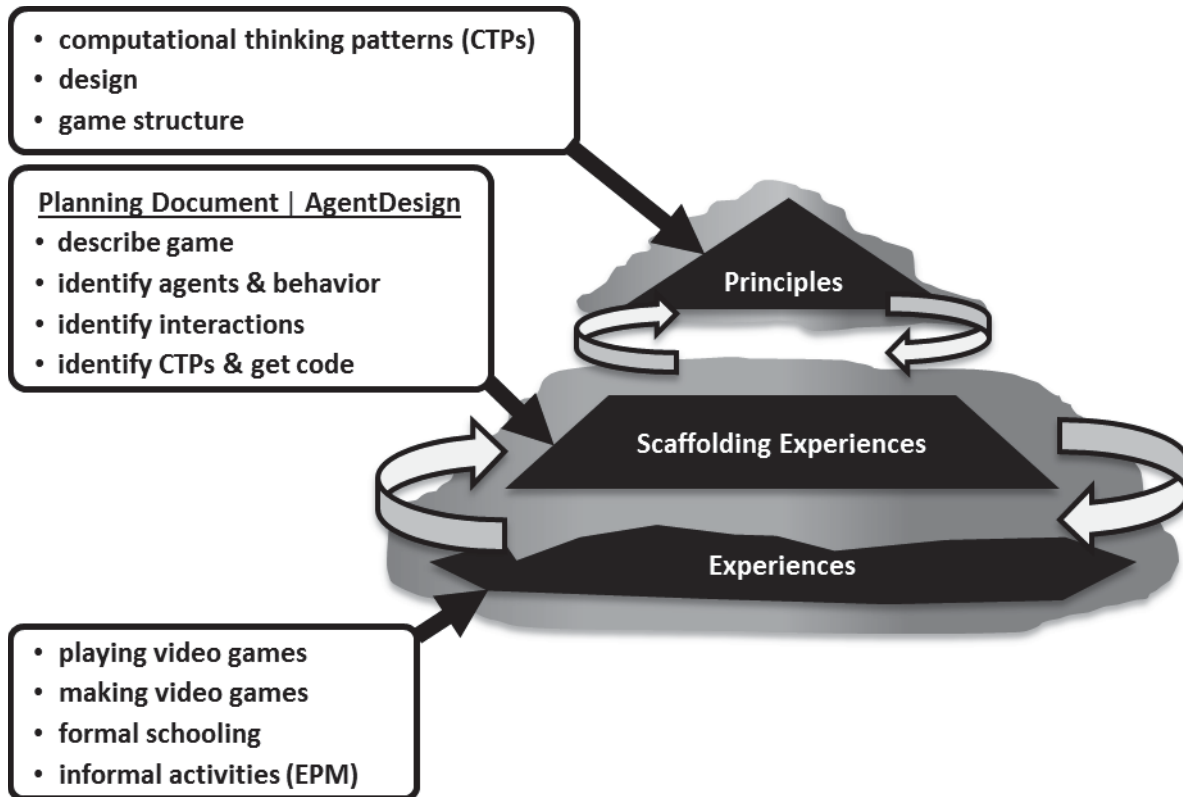


Figure 4. Vygotsky's Theory of Concept Formation and the Iceberg Model

Figure 4 emphasizes the relationship between experiential knowledge and formalized principles according to Vygotsky’s theory of concept formation (1986) and the *iceberg* model (Webb et al., 2008). As a student gains experience, they develop pre-formal ideas about a principle that are bound to that context, such as a specific inference from a specific observation in science. As the learner makes many of these inferences, she is able to see a pattern or rule that can be developed into or linked to a disciplinary principle. As these ideas become reinforced with more experiences, the principle she is developing is both changed and strengthened. The formal principle gives the student a new way to

interpret experiences and further develop her conceptual understanding of that topic as the experience-based concepts provide an anchor for principles introduced through teaching. These two processes, experiential knowledge developing a formal principle and a formal principle making sense of an experience, mediate one another.

The boxes in Figure 4 relate this theoretical formulation of conceptual development to the context and activities of the studies presented in this dissertation. Students will enter the activity with experiences playing games, and some may even have made a game in the past. Additionally, they will bring knowledge of how formal schooling works, as well as the processes involved in EPM informal activities. These experiences will influence their interaction with the scaffolding experiences established in the Game activity, the *planning document* for study 1 and the *AgentDesign planning tool* for study 2. Through the scaffolding experiences, the students will develop pre-formal ideas about game descriptions, defining agents, agent interactions, and CTPs. These pre-formal ideas can then be drawn upon in their own conceptual development of what CTPs are, what it means to do design, and how video games are structured.

Computer science consists of a set of formal symbols, language, and practices, as noted earlier. Thus, according to the learning theories presented here, there is a need for a scaffolding tool to mediate between students' everyday knowledge, their ways of solving problems, their understandings of the world around them and the programming language(s) that they are learning in school. CS involves designing a solution to a problem, then implementing that solution using a programming environment. Pane, Ratanamahatana, & Myers (2001) state that, "[a] large part of the programming task is to take a mental plan for solving a problem and transform it into the particular programming language being used," (p. 262). However, they find that the transformation aspect of that process has been shown to be particularly challenging for learners of CS. Pane et al. (2001) examined problem solutions written by non-programmers, focused on the language and structure of those solutions, and then contrasted the

those solutions with the affordances and constraints of popular programming environments. Their goal was to provide evidence-based recommendations for developing future programming environments that better suit how people naturally design and communicate solutions to problems. In one study, 5th graders were asked to use words and pictures to explain how they would instruct a computer to accomplish various scenarios in Pacman. In another study, adults and fifth graders were asked to use words and pictures to describe how they would access data from a database. In both studies, Pane et al. (2001) found that when the participants were asked to develop solutions, they had difficulty being specific about their solutions. Both the students and adults were able to develop solutions using natural language, but their solutions lacked the specificity to be easily translated into a programming language. This is because the solutions that the participants developed did not contain enough detail and also because their use of logical operators differed from the standard used in programming languages. This gap between initial solution designs using natural language and programming languages is a common occurrence for novice CS students, and emphasizes the need for future programming environments to help students translate between how they naturally communicate problem solutions and the programming languages they are using. The AgentDesign scaffolding tool attempts to mediate this transition between students' natural language and the programming environment in the work presented in this dissertation.

Other studies in CS have shown that CS programming is associated with a particular way of thinking and findings suggest that students need specific scaffolding in those ways of thinking. In the first publication in a series titled "Commonsense Computing," Simon, Chen, Lewandowsky, McCartney, & Sanders (2006) examined how college students solve problems, prior to learning specific CS practices, by studying the ways in which they approached a sorting task. They asked groups of beginning CS students (taking a CS1 course) and non-CS students (not taking a CS course) to use everyday language to describe how they would arrange 10 numbers in ascending, sorted order. Their work found that

approximately two-thirds of the CS students could correctly describe a process to sort the numbers at the beginning of the course, while only one-third of the non-CS students could also do so. Other findings included that the students commonly thought of numbers as strings, and not as a type of data, meaning that their representation of numbers in the problem solving process would not fit the computational process that is commonly used to correctly implement a sorting algorithm. Additionally, the authors found that the students did not utilize the most efficient methods to move through the data. In the study, the students preferred post-test loops to iterate through the set of numbers, as opposed the “while” loop that would more likely be used by experienced computer scientists. The most surprising finding was that the students taking the CS1 course were *less* likely to develop a correct algorithm *after* 10 weeks of instruction in the course, meaning that the students’ methods for enacting sorting regressed after spending time in the educational context that was supposed to help them to be better at the task of sorting. This finding encourages CS educators to seriously examine the ways that CS, particularly thinking through problem solving, is taught. CS Educators need to take into account what students are bringing into the learning environment, and acknowledge that their ways of thinking and understanding are likely different than that of the educators. The research presented in this dissertation seeks to address this issue by leveraging students’ own language and helping them structure their ideas in a way that will lead to connections to higher level CS practices and principles, such as algorithms (CTPs).

Research that discusses students’ types of thinking when working in a computing environment has shown that representations of a problem solution can influence the type reasoning that students carry out. Parnafes & diSessa (2004) examined how representations available to students influenced the students’ types of reasoning. They used an application called NumberSpeed, which allowed students to adjust three turtles’ initial speed, velocity, and acceleration along a one-dimensional track and then have the turtles “race”. NumberSpeed allowed the students to view each turtle’s position over time as

either a number-list or as a visual moving image. The number-list of turtle positions over time and the motion of the turtles were the two types of representations provided to the students. The two types of reasoning they observed were constraint-based reasoning and model-based reasoning. Constraint-based reasoning occurred when the students' choices for developing a solution were guided by the constraints of the problem, and then restricted the student's thinking to address only the constraints, and not the bigger picture, of the problem. The bottom-up design process discussed earlier is similar to constraint-based reasoning in that students address issues (constraints) as they come. Model-based reasoning occurred when the students approached a problem holistically by thinking through the problem solution and then adjusting the model if the underlying assumptions did not seem to work. The top-down design process is similar to how the authors described model-based reasoning in that both address the problem as whole, and encourage the process of thinking through the problem solution. Parnafes & diSessa (2004) found that there was a correlation between students' use of imagery (motion) and enacting model-based reasoning, and that when the students' used the number-list of turtle positions over time there was a correlation with enacting constraint-based reasoning. These findings suggest that students are able to approach a problem as a whole and are more likely to use model-based reasoning if they are able to visualize the problem solution. Alternately, if students are given information that is difficult to visualize, such as a list of numbers, they are more likely to address the constraints of a problem one at a time, and not consider the best overall solution. This work shows that the types of representations of information available to students is important. For this dissertation, both tools used for studies 1 and 2 are meant to guide students through the process of thinking through their game designs holistically. The visual nature of the programming environments, AgentSheets and AgentCubes, will also likely influence students to enact a more model-based reasoning approach to developing their own games, while also being mediated by the scaffolding tools.

The nature of planning out a problem solution is to develop a model that represents what the final product will look like and how it will function. The two studies discussed later in this dissertation are meant to examine the effectiveness of the two scaffolding tools, the pencil-and-paper *planning document* and the AgentDesign *planning tool*, and how they assisted students in refining their own game ideas so that they could be implemented in an agent-based programming environment. The nature of the programming environments, and the activity, are very visual, which will likely encourage a more holistic approach to the design process (Parnafes & diSessa, 2004). Additionally, the programming environments and scaffolding tools take into account the natural language ways in which the students may describe the behavior in their games. Pane et al. (2001) encouraged future programming environments to take into account how learners of CS may understand CS syntax given the ways that they already know how to organize instructions and information, and the scaffolding tools, particularly AgentDesign, are meant to address the development of the connection between natural language and CS practices and principles. Finally, students need assistance in developing detailed solutions to problems (Simon et al., 2006), and the scaffolding tools directly address this issue. The activities discussed in this dissertation are intended to assist students in connecting their own ways of describing and representing a game idea to CS practices and principles so that the game can be created in a programming environment. Assisting students in developing these connections between informal and formal representations is imperative for CS learners and is at the core of designing solutions to CS problems.

Design Research Studies

As mentioned earlier, this dissertation is a design research study. Design research studies have particular characteristics that differentiate them from typical intervention type studies (Brown, 1992; Collins, Joseph, & Bielaczyc, 2004). These include an iterative process in which the design of the study is evaluated and improved upon for each iteration, meaning the practice of the intervention is improved over time. Also, the underlying theory guiding the design is refined over iterations of implementation.

Design research studies are also set within learning environments, which can be very complex. Researchers must acknowledge and account for the ways in which the environment affects the design and enact modifications as needed by gathering a lot of data and being aware of how the data can inform future iterations.

The work presented in this dissertation fits the criteria of a design research study in that it is intended to be iterative and inform both the practice and theory of elementary students learning computer science disciplinary principles and practices. The data gathered during each iteration informed future implementations, and was processed in a way to account for the complexity of the learning environment. For this work, the first two iterations of the study are presented. The first iteration of the study involved elementary students using a pencil-and-paper *planning document* to assist them in design and creating their own video games. From analysis of the data gathered during the first study, I decided to introduce an intervention for the second iteration (study 2) to better assist students in the design process. This intervention was for students to use a web-based *planning tool* called AgentDesign instead of the *planning document*.

Study 1

Before I begin explaining the study, I would like to share how I started working with elementary students at the after school site, EPM. My involvement stemmed from a request for the Scalable Game Design (SGD) project, which I work for as a graduate researcher, to provide a CS activity and support for the students at the site. I agreed to help and initially started attending EPM one or two times a week (EPM ran three days a week). The activity that we (SGD) initially decided to use was making a “Frogger” game using a tutorial. The tutorial was commonly used by middle school students in a classroom setting as part of the SGD project. As I worked with the students to make “Frogger” I became interested in understanding how elementary students learn CS. This interest is what led me to develop the activity for the pilot study and pursue my dissertation research at EPM.

The pilot study was an exploratory study to see if the students at the after school site would enjoy making games if they were allowed to design the game and be given proper support to turn the design into a functional program. The reasoning for having students pursue a game “design-and-create” activity stemmed from a previous experience at the after school site. In the semester prior to this study being done, the activity given to students had them follow a tutorial to create a game like “Frogger,” and while the 4th and 5th grade students could complete the activity, they did not seem to enjoy it. Towards the end of that previous semester, two female students approached me about creating their own game, so I took some blank sheets of paper and a stapler and made them a small booklet. I then asked the students to draw out all of their agents on separate pages and describe what each agent would do. These students seemed to enjoy this process and were able to successfully create their own game with some help from me. This experience became my inspiration for developing the pencil-and-paper *planning document*, which is described later, to scaffold the design experience for students.

This study used an activity in which elementary students, working as a group with undergraduates, designed a game using a pencil-and-paper *planning document* and then used their

design to assist the group in creating the game using an agent-based programming environment called AgentSheets (explained later). Previous work in computer science education has not focused on using scaffolds to support elementary students designing and creating games, nor has much research been carried out on how well elementary students use agent-based programming environments. With this in mind, the research questions the pilot study pursued are:

1. What are the students' behaviors when trying to create video games using an agent-based programming environment?
2. How does the students' behaviors align with the intended "Make Your Own Game" activity process?

Conceptual Framework

For the pilot study of this dissertation I knew that the activity I would be providing would need to not only provide ample support, but also be enjoyable for the students in a way that they would be highly engaged. This conceptual framework discusses five conjectures that guided the design of the activity. The conjectures are listed below in Table 2 and the conjecture map is shown in Figure 5.

Table 2. Study 1 Conjectures

Associated Theory	Conjecture	Description
Student Engagement	1:	<i>Having students design, create, and test their own games in groups will engage them in the activity of making a game.</i>
Scaffolding	2:	<i>Providing a structure, the planning document, will assist students in designing and refining their ideas.</i>
Scaffolding	3:	<i>A completed planning document will assist groups in creating their games using AgentSheets.</i>
Scaffolding	4:	<i>The groups will not require a lot of support from the researcher because of the scaffolding (planning document) used in the activity.</i>
Concept Formation	5:	<i>Making games will help students learn CTPs.</i>

As stated earlier, the two students that designed and made their own game together seemed to enjoy the activity more than following a tutorial. From this occurrence I latched on to the idea that

engagement and ownership would be important elements for helping the students complete a video game creation activity and enjoy it.

I developed the *planning document*, as a non-faded scaffold, to support the students during the design process. Similar to White et al. (2002), I used a constant scaffold in the form of the *planning document* (Figure 6) to guide students through a practice of top-down design, as well as to assist them in developing a standard practice of design. Additionally, the scaffold provided prompts for when students would need to look up documentation, which is a typical CS practice. With little to no programming experience, the students needed the scaffold to guide them through connecting their designs to known algorithms (computational thinking patterns) that were described in a separate online resource. The expected outcomes were that the groups would complete the design and successfully use that design to find programming code, learn about CTPS, and create a game using the AgentSheets programming environment.

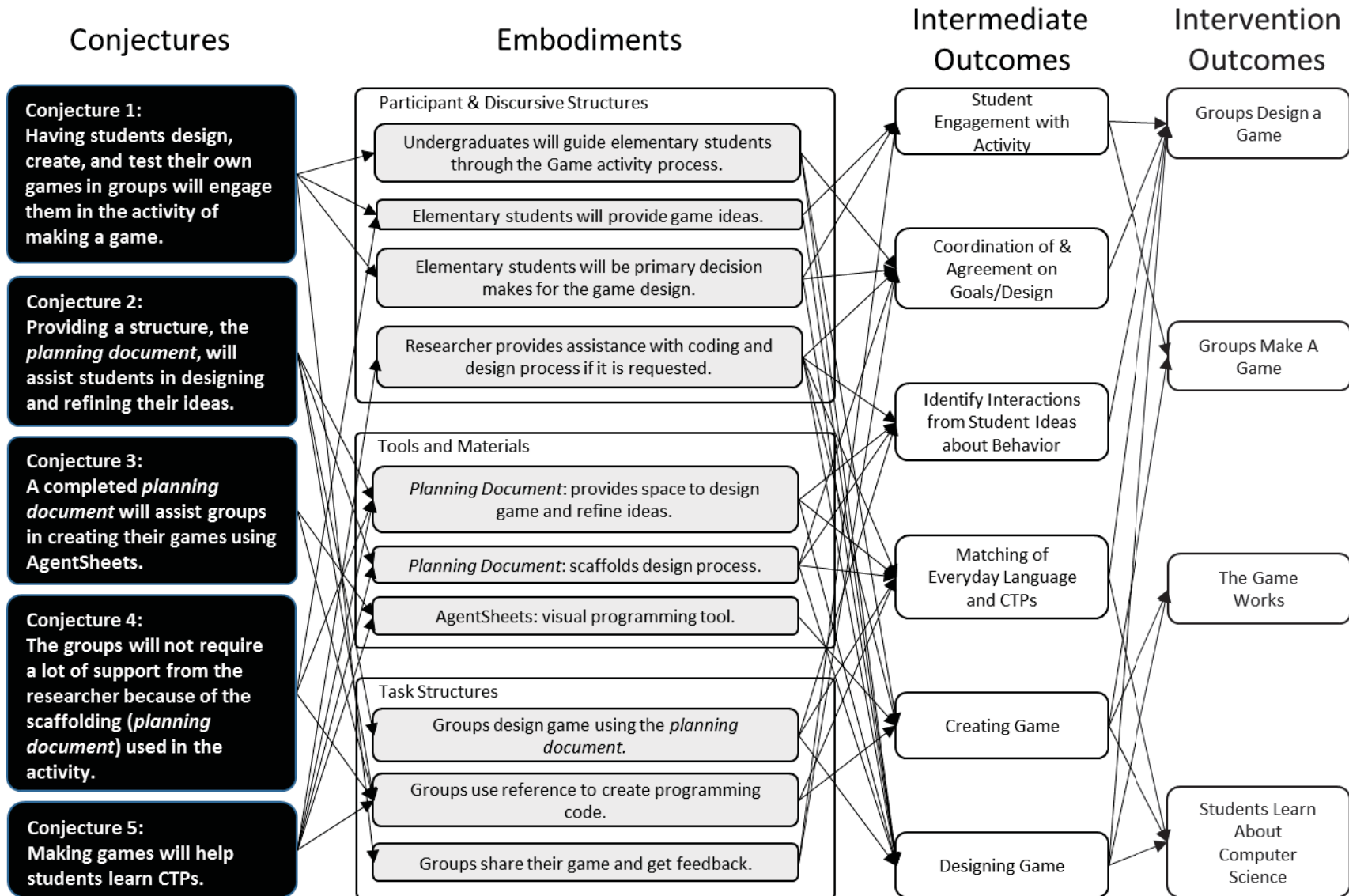


Figure 5: Pilot Study Conjecture Map

Conjecture 1: *Having students design, create, and test their own games in groups will engage them in the activity of making a game.*

This pilot study was guided by a desire to engage the students and assist them in being successful in creating a video game. Structuring the activity and environment in a way where the students could feel like they could create a game through effort (develop a mastery goal orientation) and have ownership over the final product was a way to encourage high engagement (Ames, 1992). By having the activity use student ideas to design their game, they did not need any special abilities, skills, or knowledge to complete the design process. Using their own ideas gave the students *authority* within the activity, which addressed one of Ames's (1992) three structural components. The students were able to develop responsibility and have a level of independence through their decision making power. To address the *task* structure of the activity, the pencil-and-paper *planning document* (shown in Figure 6) was visual, but also provided prompts, so that students could see the progress they were making and what they had left to complete. The prompts for the *planning* document were specific, but gave students space to express their ideas through drawing or writing. Additionally, students could track their own progress in the programming environment by comparing what they had designed for to what was completed. The students were also given control over their own *evaluation* through the ability to decide if the design was what they wanted. During the programming aspect of the activity, the programming environment also allowed for students to test their game as they created it without having any high-stakes failures. Given that students were the evaluators of the design, and the programming environment allowed for incremental testing, the students were provided the opportunity to learn to tolerate failure. And in the cases where they could not overcome that failure on their own, they always had me, the researcher, around for support.

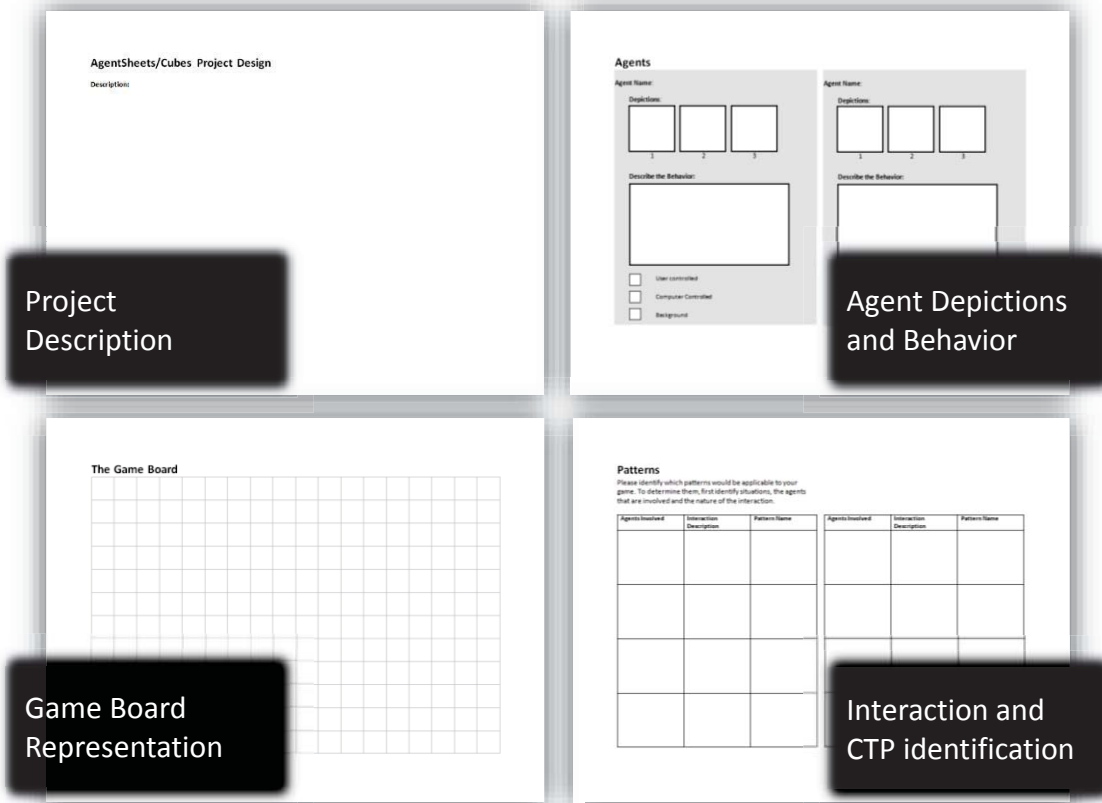


Figure 6: Pencil-and-Paper *Planning Document*

The embodiment of this conjecture in the activity was primarily facilitated through the participant, discursive, and task structures. Students gained ownership through the participant and discursive structural components by having the power of choice for the design. The undergraduates working within the group were participants as facilitators of the activity. In contrast, the elementary students were the members of the group providing the game ideas and were also the decision makers for the group. Additionally, through the task structure of having the game shared after it was created the students also had a motivation to make something great for their peers.

Conjecture 2: *Providing a structure, the planning document, will assist students in designing and refining their ideas.*

As discussed earlier by Robertson & Nicholson (2007) and Kelleher & Pausch (2006), novices typically do not have fully developed ideas when they design. However, the students would have initial

ideas that they could bring into the activity. Using those ideas as a starting point, the *planning document* provided a scaffold to assist the students in refining their ideas. Students could use the project description as a starting point, and the scaffold would help them to identify agents, define individual agent behavior, and then identify interactions. The *planning document* would help the students and undergraduates to work within their ZPD (Vygotsky, 1978) to fully design their game.

The embodiments of this conjecture were through both task and tools & material structural components. The tool that supports students is the *planning document*, and it provides the space and prompts for what students needed to complete for the design. And although it is implied, completing the *planning document* was a necessary task for groups. Also, within completing the *planning document*, a necessary and important task was for the groups to use a reference to identify CTPs to access useful programming code.

Conjecture 3: *A completed planning document will assist groups in creating their games using AgentSheets.*

As discussed in conjecture 2, the *planning document* was designed to assist the groups in finding helpful programming code for behavior that they had designed for. Without this part of the scaffold, the students would only have their ideas, written in their own words, to help them create programming code. In some instances, the natural language descriptions have been all that was needed to create code, but in general this was not the case. The *planning document* was meant to help the students move from natural language descriptions to programming code by identifying behavior and then using an online reference to get code. Again, the scaffold extended the students capabilities, but did it in a way that the students could make sense of the higher thinking and learn (Vygotsky, 1978).

The embodiment of conjecture 3 was carried out through the activity task structure and the activity tools & materials structure. The groups were tasked with using a reference to find programming

code using their design. The AgentSheets programming environment was the tool they used to actually do the programming.

Conjecture 4: *The groups will not require a lot of support from the researcher because of the scaffolding (planning document) used in the activity.*

The nature of scaffolding is to help learners extend their capabilities (McNeill et al., 2006). The *planning document* was designed in a way that the groups would be able to complete the activity, and make a game, as long as they completed the document. Since the *planning document* would provide any necessary information as it was needed, any support from the researcher should have been as minimal as possible.

The embodiment of this conjecture was present in all of the structural components of the activity. The students were the primary idea generators and decision makers, and so the researcher would not be very useful during the design process outside side of facilitation. The *planning document* also provided all of the necessary prompts and space necessary for the groups to develop a plan. And finally, the *planning document* combined with the programming code reference would have given the groups the necessary information to turn their design into a working game.

Conjecture 5: *Making games will help students learn CTPs.*

The final conjecture was that students would learn about CTPs. If a student is operating within her ZPD, she is extending what she can do, but she is doing so in a way that she is learning. Scaffolds are intended to be removed (McNeill et al., 2006), and although the scaffold used for this activity was not intended to be removed, it did not ignore the fact that as the students experience CTPs and relate them to their own understanding, they will learn (Vygotsky, 1986).

The embodiments of conjecture 5 were through the *planning document*, the AgentSheets programming environment, the researcher's assistance, and the programming code resource. By working with all of those materials and resources, the students would be able to build connections

between their own ideas and the formal concept of CTPs. Essentially, conjecture 5 is embodied in the process of creating a final product for the Game activity, which must use CTPs.

The activity process that was designed, while informed by my understanding of scaffolding and motivation, was primarily influenced by what I knew of computer scientists' practices. Successful computer scientists design their products first, and commonly rely on documentation to help with unknown information. For students to learn about these practices, and not just about programming, they needed to be given an activity that was motivating and supported them when they needed it.

The 5th Dimension Activity System

The setting for this pilot study, an after school program, also had its own theoretical guidance. The after school program, called El Pueblo Mágico (EPM), employed a 5th dimension activity system (Cole & Engestrom, 2007). The essential aspect of a 5th dimension activity system is that there is a, "joint activity between a university and community institution," (p. 496) as indicated in Figure 7. Most 5th dimension sites are located within the local community, and are commonly held during after school hours at a K-6 school. At these locations university and elementary students participate in activities meant to enhance the children's intellectual and social development. 5th dimension activity systems provide the community institutions with activities that contribute to their students' learning. The university institution benefits by having a location for their students to learn about fieldwork and the development of young minds.

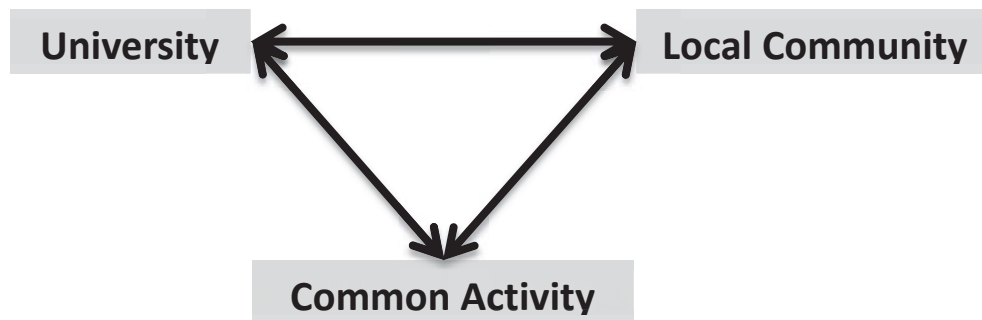


Figure 7. 5th Dimension Activity System

In the case of EPM, the joint activity is between university undergraduates taking an education psychology course and elementary students in 2nd through 5th grade. At EPM, elementary students and undergraduate students worked together in groups to complete various activities offered by the program. The available activities were created so that expertise and knowledge could come from any source, student or undergraduate, and be utilized by the group to complete the activity. These activities were designed for the groups, which ranged in size from 2-6 participants, to involve aspects of creativity and play. The program ran for 2 hours, three days a week, throughout a semester with some elementary students attending multiple days and undergraduates only attending one day a week. If an elementary student attended multiple days, the activities she pursued changed for each day depending on what her group chose for that day. For example, a student attending on a Monday and a Tuesday would have separate projects and groups for each day, so she would only work on a given activity one day a week. The available activities included Board Games, Digital Storytelling, World Maker, Make Your Own Game, and others.

The “Make Your Own Game” Activity

Goal

The overarching anticipated outcome of the “Make Your Own Game” activity, which will now be referred to as the Game activity, was for students to create good games with minimal help from an expert. Within this larger goal was for students to also learn about the purpose of design, understand computational thinking patterns (algorithms), develop ways of evaluating their work, and align their identity with CS.

In the following sections the AgentSheets agent-based programming environment and activity are explained. The ordering is intended to assist you, as the reader, in understanding the language and context of the activity in a way that the process and pencil-and-paper *planning document* make sense.

AgentSheets Programming Environment Overview

The AgentSheets programming environment (Figure 8 and Figure 9) is designed to be an accessible platform for young students to do programming. The premise of the environment is that

there are *agents* and each agent has its own *behavior*. Multiple instances of an agent can be placed within a game play environment called a *worksheet*, which will be referred to as the *game board* from this point forward. The game board is shown in Figure 8 and is where the agents' programmed behavior is the only thing that guides what they will do once the program is running.

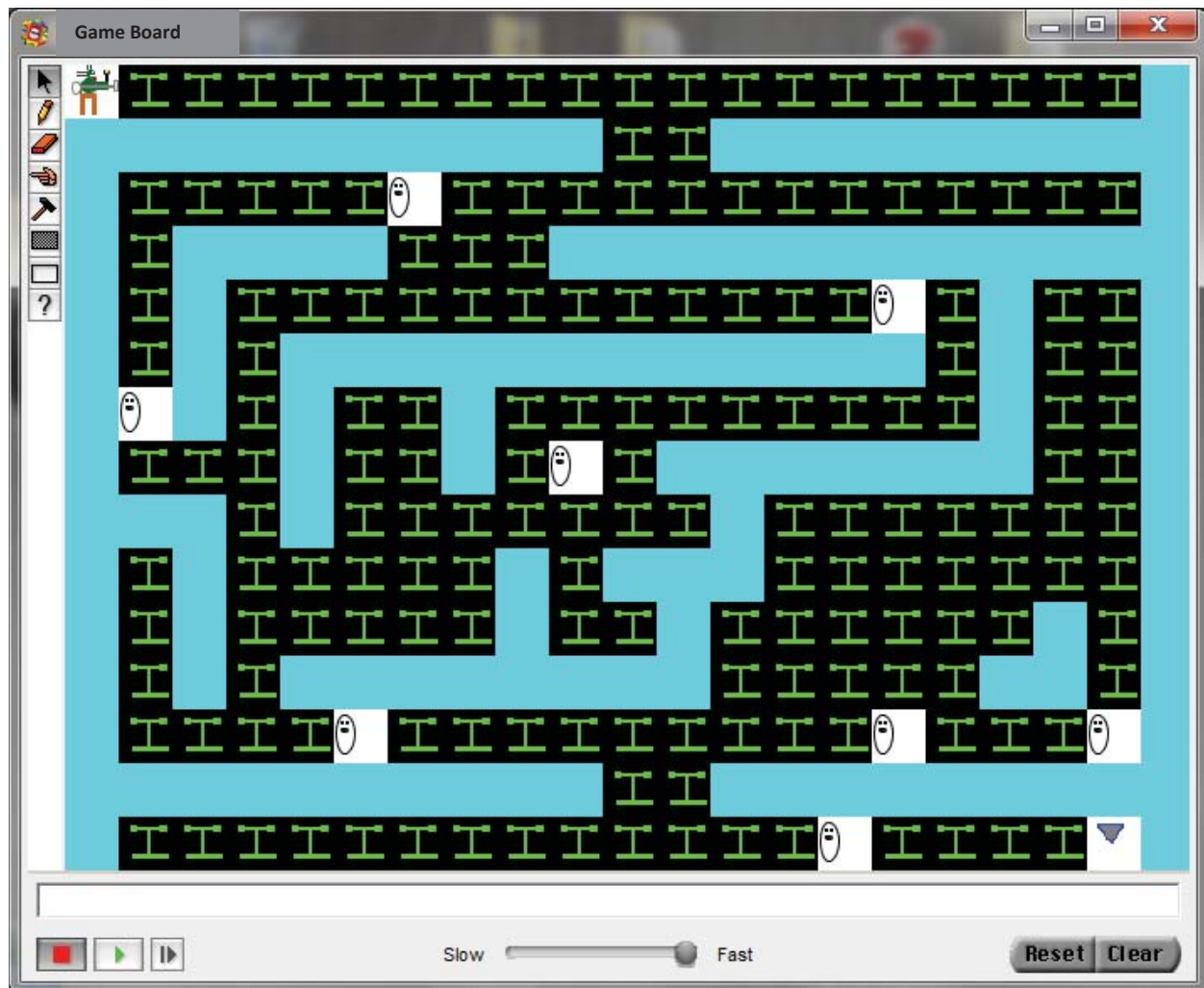


Figure 8. AgentSheets Game Board

Figure 9 shows how agents can be created and programmed in AgentSheets. This is done through accessing the depiction and behavior editors from the project gallery window in the programming environment. Agents can be created and accessed from the *Gallery* window, shown in Figure 9A. The buttons at the bottom of the *Gallery* access both the behavior and the depiction of the selected agent. An agent's behavior can be edited in the *Behavior* window, Figure 9B. Behaviors are

controlled by methods and rules, where rules are composed of IF-ELSE statements. To create a rule, conditions can be added to the IF side of a rule from the *Conditions* window (Figure 9C) and actions can be added to the THEN side of a rule from the *Actions* window (Figure 9D). To edit an agent's visual representation, called a depiction, the *Depiction* window is used (Figure 9E). The *Depiction* window is a simple drawing space that can also import images.

The screenshot displays the AgentSheets software interface. On the left is the 'Gallery: New Project' window (A), which lists various agents like 'Zombie', 'Ghost', and 'Themonster'. The main window shows the 'Behavior: Zombie' editor (B) with a 'While running' method containing IF-THEN rules. Below this are the 'Conditions' editor (C) and the 'Actions' editor (D), both showing a 'basic' scenario with various conditions and actions. On the right is the 'Depiction: zombie' editor (E), which shows a pixelated drawing of a zombie. Arrows indicate the flow of information: a green arrow from the Gallery to the Behavior Editor, a blue arrow from the Behavior Editor to the Conditions Editor, a red arrow from the Behavior Editor to the Actions Editor, and a yellow arrow from the Conditions and Actions Editors to the Depiction Editor.

The behavior of each agent is controlled by *methods* and *rules*. The primary method is called "While Running," and the rules within this method are constantly looped through from top to bottom while the game is running. Other methods can be given names and called on an individual basis.

Rules are formatted as IF-THEN statements, if the conditions inside the IF are true at a given time, the **actions** in the THEN are carried out. When a rule is evaluated as true, all other rules below it are skipped until the next time the method is run.

The gallery is where agents are created. An agent's default and alternate depictions are shown here. The depiction and behavior editors can be accessed here.

The conditions are drag and drop, testable scenarios that will result in a true or false being given.

The actions are drag and drop acts that an agent will carry out if the rule's conditions evaluate as true.

The depiction editor provides a simple drawing tool to create a visual representation of an agent. How the agent will look on the game board is shown in the upper right-hand corner of the window.

Figure 9. AgentSheets Gallery, Depiction Editor, & Behavior Editor

Game Activity Process

For the Game activity, groups composed of elementary and undergraduate students were asked to pursue a *top-down* design approach to creating their game. With this approach, students were to design the game on a pencil-and-paper *planning document* before they got on a computer and worked with the AgentSheets programming environment. This approach served three purposes. One purpose was to provide a model for, and experience with, planning out a solution to a genuine and meaningful computing problem. The second was to ensure that each group understood what they wanted to do before working with the programming environment. From that understanding the students would then be able to learn how to create AgentSheets code from their own language and thinking. The third purpose was to give the groups a sense of ownership and progress over what they were creating, which would then lead to a higher level of motivation and interest in the activity.

The intended process of this activity consists of three parts: (1) design a game using the *planning document*, (2) create the designed game using AgentSheets, and then (3) test out their game with other people and possibly make changes. This process is further explained below.

Designing the Game

The task of designing a game was facilitated by a *planning document*. The *planning document* was a pencil-and-paper, structured, non-fading scaffolded guide for each group to analyze their initial idea for a game, identify its important components, and then refine those important pieces. This work was done primarily on paper using the *planning document* and included sections to be filled out (Figure 11-Figure 14). Table 3 gives a general description of how Figure 11 - Figure 14 relate to the necessary activities to design a video game.

Table 3. Practices for designing a video game using the *planning document*

	Design Practices	Design Practice Details	Figures
A	Project Description	i. Describing the game idea ii. Recording a general statement about the game	11
B	Agents	i. Identifying agents ii. Describing and recording agent behavior iii. Drawing representations of the agents iv. Choosing if agents were human, computer, or not controlled	12
C	Game Board	i. Drawing a representation of what the game will look like	13
D	Patterns	i. Identifying interactions between agents ii. Associating interactions with computational thinking patterns	14

The *planning document* (Figure 10) provided space for all of the necessary work of designing a game to be done, including drawing and writing. The initial ideas of any game could be described or drawn in an open area on the Project Description page (Figure 11). Agents could have their visual representations sketched out and their behavior described on the Agents pages (Figure 12). A visual representation of the game could be drawn on the Game Board page (Figure 13), which had a grid that modeled how AgentSheets organizes agents. Interactions between agents could be identified, described, and associated with computational thinking patterns on the Patterns page (Figure 14). Figure 11 through Figure 14 in the following sections show the (8.5x11 inch) pages of the planning document meant to guide students through the four activities shown in Table 3. Figure 10, below, shows the process that a student or group may go through when designing a game using the *planning document* pages.

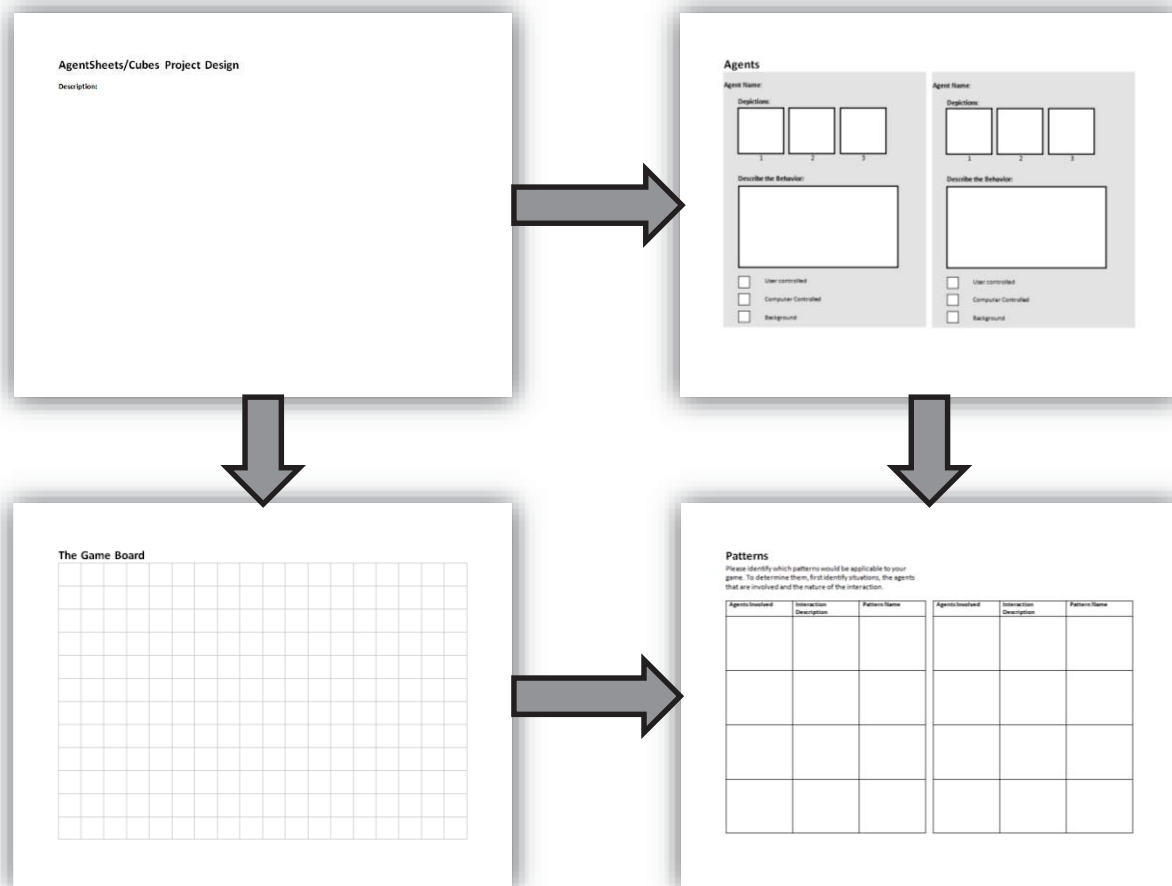


Figure 10. Planning Document Pages and Work Flow

Project Description Page

The project description, discussed in Table 3A, provided the initial prompt to describe the game.

This was essentially a blank page that could be written or drawn on in any way that was needed to express the game idea (Figure 11).

AgentSheets/Cubes Project Design

Description:

Figure 11. Project Description

Agents Pages

Multiple pages of Figure 12 were provided in the *planning document* for the identification of agents, where two agents could be designed on each page. Agents needed to be named, drawn, and have their behavior described during the design process. The name of an agent could be written at the top of each grey space on the page (Table 3B-i). There were three spaces available for different representations of an agent to be drawn, in case the agent would have a different look at different times in the game (Table 3B-iii). A free space was provided for describing the behavior in which drawings or natural language descriptions could be recorded (Table 3B-ii). Also, three checkboxes were available to identify if an agent was “user controlled,” “computer controlled,” or was supposed to be in the “background,” (Table 3B-iv).

The figure shows two identical grey rectangular forms side-by-side, each titled "Agents". Each form contains the following elements from top to bottom: a label "Agent Name:" followed by a blank line; a section labeled "Depictions:" containing three small square boxes numbered 1, 2, and 3; a section labeled "Describe the Behavior:" containing a large empty rectangular box; and three checkboxes at the bottom, each with a label: "User controlled", "Computer Controlled", and "Background".

Figure 12. Agents

Game Board Page

Another task that the *planning document* prompted was for the game play environment to be drawn out (Table 3C). AgentSheets has visual representations of agents organized within a grid-based

gameplay environment, so a blank table of squares was used here (Figure 13). These blank squares were to be filled in using sketches of agents that were already drawn on the Agents page in Figure 12.

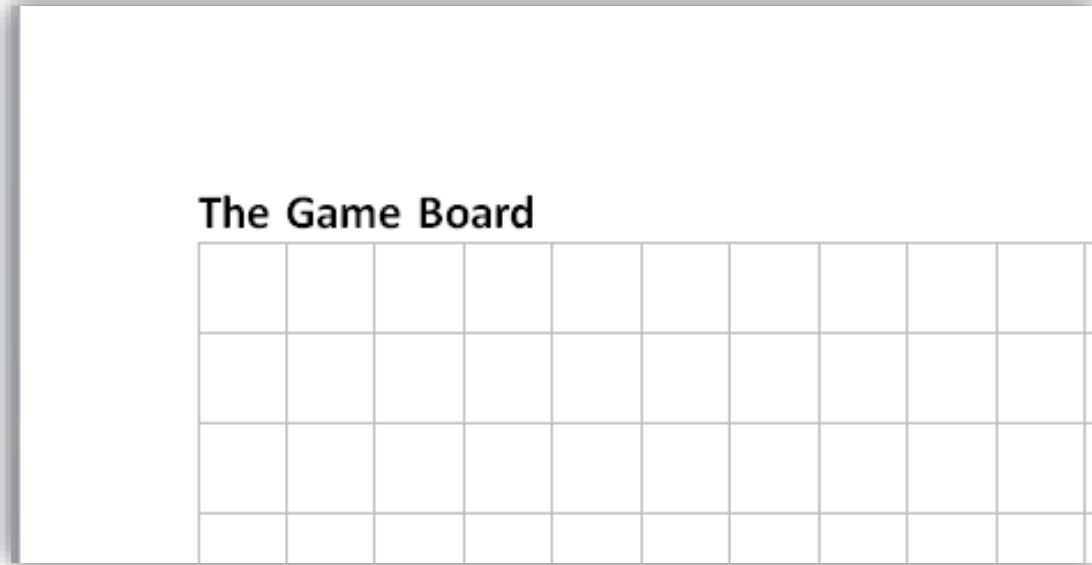


Figure 13. Game Board

Patterns Page

The final task prompted by the planning process was to develop an association between the agent interactions and computational thinking patterns (CTPs) (Table 3D). CTPs are common phenomena that happen in spatially oriented games, such as collision or tracking. The *planning document* prompted this task through the page shown in Figure 14. On the Patterns page, three columns were available to describe what agents would have an interaction with one another (Table 3D-i), what the natural language description of that interaction was (Table 3D-i), and the CTP that fit the interaction (Table 3D-ii). A list of patterns was available on a websiteⁱⁱⁱ, which also provided sample code, that would assist in the creation process using AgentSheets.

Patterns

Please identify which patterns would be applicable to your game. To determine them, first identify situations, the agents that are involved and the nature of the interaction.

Agents Involved	Interaction Description	Pattern Name	Agents Involved	Interaction Description

Figure 14. Patterns

Building the Game

Once the design was finalized, the groups were expected to build the game on the computer using the AgentSheets programming environment. This activity required creating and drawing the agents identified in the *planning document* in AgentSheets and translating the everyday language descriptions of behaviors to AgentSheets code. All of the information needed to do this was intended to have been worked out in the *planning document*. Most of the behavior in games occurs where there is an interaction between two agents, so examples of needed code would be given when CTPs were matched to interactions using the online references.

Testing

The final aspect of the Game activity process was to have the game be tested by other people. Testing was meant to serve two purposes, one was to see if the game operated the way the group wanted it to and the other was to see whether or not it was a good game. This was due to the possibility that the choices the group made in creating the game would make it not fun to play. From this point, any changes that needed to be made based off of feedback could be carried out.

Figure 15 expresses the relationship between the *planning document* and the general cyclical planning process of design, building, and making changes based on testing. The figure associates the pages of the *planning document* with the Game activity process and access points for where a redesign could focus.

In Figure 15, a student wanting to make a game would start by describing and recording their game idea (step 1), then would identify and record the needed agents based on that description (step 2). From the list of identified agents the student would draw a representation of each agent (step 3). Next, the student would record each agent's behavior (step 4) and figure out which agents will interact with one another in the game (step 5). The student would then need to associate the behaviors and interaction descriptions to computational thinking patterns (CTPs) (step 6). It is at this point that the student may be done working with the *planning document*, unless a redesign is needed.

If the design developed in steps 1-6 seems as robust as possible, the student then creates the game using AgentSheets (step 7). The final step in the cycle is for the student to test their game and then possibly make changes based on how the game plays or feedback that was received (step 8).

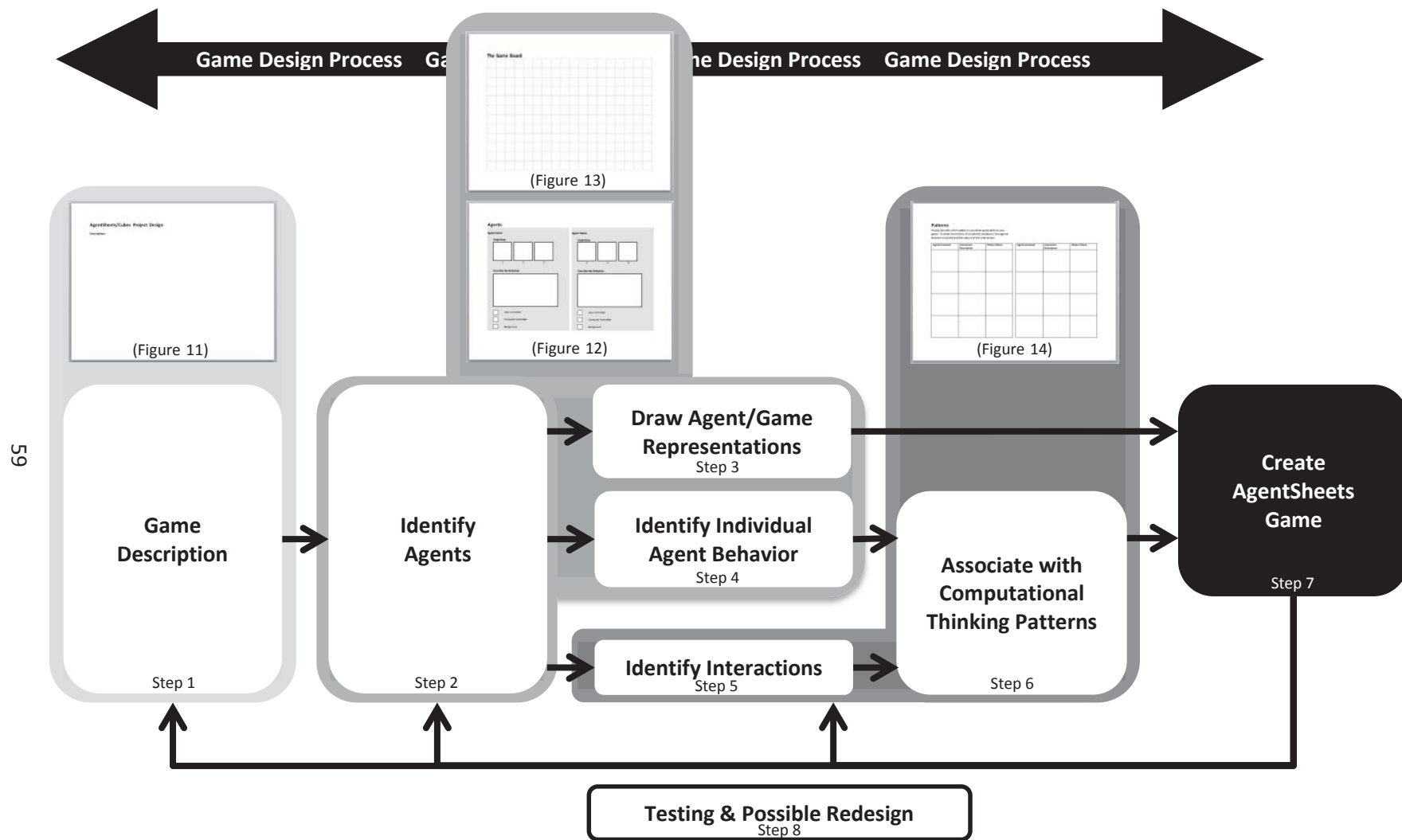


Figure 15. The Game activity process in relation to the *planning document*

Study Design & Analysis

This study was designed to be a participant-observation, ethnographic study (LeCompte & Schensul, 1999). The researcher played the role of an expert at EPM for students taking part in the Game activity (Figure 15) and participated heavily in helping the groups of elementary and undergraduate students to make their games work. Group activity was video recorded (explained later) with the purpose of creating profiles of student activity from analysis. The profiles of activity from video were then used to compare to the intended activities of the design process and provided a general sense of what students chose to focus on when designing.

Participants

Undergraduates & Training

As stated earlier, undergraduate students worked cooperatively with elementary students to select and complete the activities available at EPM. Their participation in the after school program was required as part of an education psychology course they were enrolled in. Most of the undergraduate students were pre-service elementary teachers and the population was comprised of almost all women.

It was not expected for the undergraduate students to have strong content knowledge about CS; they were required to attend training for the Game activity. The training was divided into two parts and was given prior to the start of the after school program for the semester. The first training focused on CS content knowledge and lasted two hours. This first training also provided an introduction to the AgentSheets programming environment. In the training, the undergraduates individually created a simple maze game where a “Hero,” agent needed to get to the end of a maze and avoid a “Bad,” agent. This was carried out through direct instruction by the researcher, who was an AgentSheets expert, where the undergraduates followed along individually using their own computers. The second training focused on the pedagogical content knowledge needed to carry out the activity at EPM. This involved learning about the *planning document* that they would use to design a game with their groups, see Figure 11-Figure 14. The second training lasted approximately an hour and consisted of the researcher explaining the purpose of the *planning document* and having the undergraduates go through the

process of designing a game. This was done in small groups while the researcher provided support as they went through the process. The undergraduates were given instructions that the designs could be as elaborate as possible, as long as they were clear and complete about what should happen in the game. They were also made aware that the *planning document* would assist them in finding the necessary code through the identification of interactions and computational thinking patterns that were explained on a separate websiteⁱⁱⁱ.

Elementary Students

The elementary student participants in this research were in grades 2 through 5. This participant population was representative of the school's demographics, shown in Table 4. Some of the elementary students had past experience creating games using AgentSheets from previous attendance at EPM, but none had experienced the design approach used in this research.

Table 4. School demographic information^{iv}

<i>Grade Level</i>		2ND GRADE	3RD GRADE	4TH GRADE	5TH GRADE	TOTAL
American Indian or Alaskan Native	F	0	0	1	0	3
	M	1	0	0	1	
Asian	F	0	0	0	2	4
	M	0	1	1	0	
Black or African American	F	0	0	0	0	1
	M	0	0	1	0	
Hispanic or Latino	F	17	10	11	15	116
	M	16	16	17	14	
White	F	2	12	6	2	43
	M	4	10	3	4	
Two or More Races	F	1	0	0	1	6
	M	1	0	1	2	
TOTAL		42	49	41	41	173

Role of the Researcher

A third type of participant in the Game activity was myself, the researcher, and I was the expert for AgentSheets at EPM. My role at EPM was to assist the groups working on the activity but not act as a teacher. I have a degree in CS and had the most CS content knowledge at EPM. Besides providing

assistance at EPM, I was also responsible for training the undergraduate students to carry out the activity.

Data Sources

The data sources for this research included video, the planning document (Figure 11 - Figure 14), and the resultant games. Additionally, there are two types of video data. One type of video data is a stationary camera that focused on a group throughout their design and game creation tasks. The other type of video collected was from a head cam that the researcher wore while providing assistance at EPM.

Groups were selected to be recorded by the stationary camera on the basis that they were pursuing their first attempt of the Game activity, had an early interest in creating a video game, and the researcher believed that they would complete the activity. This opinion was based on the group having a general idea of what they wanted to make and all of the members of the group agreeing that they wanted to make a game. The recordings of these groups started within the first two weeks of the semester. In total, four groups were recorded throughout their Game activity process (Figure 15). Three of these groups completed a playable game.

Planning documents and completed games were collected from all students. The data was used to triangulate what students intended to do with their processes of activity and what the results of that activity ended up being.

Design Practices Coding Scheme

The development of the coding scheme was guided by tasks that students were asked to complete during the design phase, which precedes the creation and testing phases, of making a video game. In the design portion of the Game activity, students were prompted to complete a set of tasks intended to assist their development of what they wanted their game to look like and do. The codes shown in Table 5 are a priori codes and activities that I was interested in seeing if the elementary

students were able to demonstrate. Table 3 (reproduced below) shows all of the practices that students were expected to enact while they designed their games.

Reproduction of Table 3. Practices for designing a video game using the *planning document*

	Design Practices	Design Practice Details	Figures
A	Project Description	iii. Describing the game idea iv. Recording a general statement about the game	11
B	Agents	v. Identifying agents vi. Describing and recording agent behavior vii. Drawing representations of the agents viii. Choosing if agents were human, computer, or not controlled	12
C	Game Board	ii. Drawing a representation of what the game will look like	13
D	Patterns	iii. Identifying interactions between agents iv. Associating interactions with computational thinking patterns	14

Table 5 emphasized the relationship of these practices to the coding scheme. The purpose of using this coding scheme was to understand what top-down design practices the groups were engaged in during the design process.

Table 5. Design practices coding scheme

<i>Code</i>	<i>Description</i>	<i>Design Practice(s)</i>
Discussing Overall Game	Discussion amongst the group about what they want in their game. General ideas/brainstorming that may result in being written down.	Table 3A-i Table 3A-ii
Identifying Agents	Discussing and identifying the agents that will be required for the game to do what they want.	Table 3B-i
Discussing Behavior	Talk about what agents will do outside of brainstorming ideas.	Table 3B-ii
Recording Behavior	Someone writes down what an agent will do during the gameplay.	Table 3B-ii
Discussing Depiction	Discussion of what agents will look like in the game.	Table 3B-iii
Drawing Depiction	Someone is actively drawing a representation of what an agent will look like, can be either digital or on paper.	Table 3B-iii
Selecting Agent Controller	Deciding and recording whether an agent will be user controlled, computer controlled, or just be stationary in the background.	Table 3B-iv
Discussing Game Board	Discussion of what the game environment will look like and how the agents will be placed.	Table 3C-i
Drawing Game Board	Drawing out an example of what the game environment will look like.	Table 3C-i

Discussing Interactions	Talk about what happens because of two agents interacting in some way.	Table 3D-i
Recording Interactions	Someone writes down when two agents interact.	Table 3D-i
Identifying CTPs	Talk about what computational thinking patterns are occurring in the game based on the behavior and interactions they are planning to include.	Table 3D-ii

This coding scheme was used to analyze the video data of the four groups to develop a profile of how each group participated in the design process.

Analysis Methodology

Coding

An analysis was carried out on the design practices of the four groups recorded by the stationary camera using the coding scheme previously described (Table 5). For each groups' video that showed how each group proceeded through the design process, the group was given a code (Table 5) for a timeframe of their video if at least one person in the group was participating in that activity. This coding was done directly from the video. In many cases there were overlapping activities as different members of the groups performed different tasks concurrently. All activities were coded. There were also times during the design process where the group was not performing any of the activities from the coding scheme, which led to gaps of time in the raw coding. "Off-task," behavior was coded, but is not included in the analysis because there was commonly one person in the group that was "off-task," while others were engaging in a codable activity. Notes were also taken to describe what was happening during that timeframe for an activity. Table 6 provides an example of the raw coding that was performed and is shown along with audio transcription excerpts during that timeframe. Please be aware that the game this group was designing was not a school appropriate game. The inappropriateness of the game was addressed with the students outside of the coded examples shown below and it was not completed. The students are designated by S1 and S2, and the undergraduates by U1 and U2. Non-verbal activities are described in square brackets.

Table 6. Example of coded design data with accompanying transcript excerpts

Task	Start	End	Total	Transcription
Drawing Depiction	0:00:00	0:11:05	0:11:05	<p>S2: [Smiles at camera, then continues drawing on a paper. There are already drawings on it.]</p> <p>...</p> <p>S1: What should I draw?</p> <p>U1: So draw, he was drawing the nerd, you want to draw the bully? Is that the bully you want? [directed to S1]</p> <p>...</p> <p>U1: We can color it too.</p> <p>S1: We color in the pants blue and give it like a black [inaudible] around it.</p> <p>U1: Ok.</p> <p>S1: And you know like maybe some ripped up pants.</p> <p>U1: Yeah.</p> <p>S1: That'd be cool.</p>
Discussing Behavior	0:09:40	0:15:30	0:05:50	<p>...</p> <p>Researcher: Alright, so, on our planning sheet write down what the nerd does [motions to location on planning document where behavior goes]. What does the nerd do?</p> <p>S2: Stands there.</p> <p>S1: Just stands there and cries.</p> <p>Researcher: Ok, he stands there...</p> <p>S1: He pees his pants.</p> <p>Researcher: And then what happens?</p> <p>S1: Pees his pants.</p> <p>U2: He gets [inaudible] right? [directed at S2]</p> <p>S1: No, he's scared and he pees his pants.</p> <p>S2: Noooooooo.</p> <p>U2: No? What happens to him then? [directed at S2]</p> <p>S2: [while drawing on a separate paper] [inaudible] they stand like that until they get killed [inaudible].</p> <p>U1: Are we gonna have them moving?</p> <p>Researcher: Yeah, do they move at all?</p> <p>S2: Why would they move?</p> <p>U2: Don't you throw one into the other one?</p> <p>S2: Yes.</p>

				<p>Researcher: And if they move, when do they move?</p> <p>U2: You have to tell'em that they get thrown. One of them gets thrown into the other one.</p> <p>U1: So you wanna have like a crowd of nerds. Here's like a bully, here's the crowd of nerds right? [motioning with his hands]</p> <p>U1: Do you guys swing nerds at the crowd? Is that correct?</p> <p>S2: No. That's not correct.</p> <p>...</p>
Recording Behavior	0:11:55	0:12:21	0:00:26	<p>U2: Right there. [points at S2] [inaudible] the nerds [inaudible]. Write that down. [S2 starts writing in the planning document]</p> <p>U1: So how does, how does angry birds work?</p> <p>S1: Say Whaaat? [high pitched voice]</p> <p>U2: [inaudible] Just say, there's a group of [inaudible] there and [inaudible] throw.</p> <p>Researcher: [brings over some blank papers and puts them in front of S2] So if you want, draw what it will look like. So you have a bunch of like, nerds over here right? [motions to one side of blank page] That they're going to be thrown into? What goes on this side [motions to other side of paper] if it's like angry birds?</p> <p>S1: [makes some funny noises]</p> <p>U1: That looks beautiful. [directed at S1 and her drawing]</p> <p>S1: Thanks.</p> <p>U1: So, so, so. So can you tell me how angry birds is [inaudible]? 'Cause I've never played it.</p> <p>S2: [inaudible] [pushes the planning document away after writing some]</p>
Discussing Game Board	0:12:04	0:15:05	0:03:01	<p>U1: So, so, so. So can you tell me how angry birds is [inaudible]? 'Cause I've never played it.</p> <p>S2: [inaudible] [pushes the planning document away after writing some]</p> <p>S1: But I forgot to draw [inaudible, picks up marker] Well, it's sort just of like a... Where you like, hmm, I'll just draw it</p>

				<p>'cause I can't [inaudible]. It's hard to explain. [S1 starts drawing]</p> <p>U1: [inaudible] Yeah, yeah.</p> <p>S1: I'm making a [inaudible, doll?] on there because [inaudible], I just don't want to.</p> <p>U1: Yeah, no, that's fine.</p> <p>U1: That's a slingshot?</p> <p>S1: Yeah, that's supposed to be the slingshot.</p> <p>U1: Ok.</p> <p>U1: So, in our game do we want, are we gonna have a slingshot? Or are we just gonna have a guy?</p> <p>S1: We're gonna have a guy.</p> <p>S2: We need a guy with a slingshot there.</p> <p>S1: Well you know what...</p> <p>U2: Yeah, we can't, it's gotta like [makes arm motion moving her hand up slowly like a cannon].</p> <p>U1: Ok, [inaudible].</p> <p>...</p>
--	--	--	--	--

From these data a count of instances of coded activity, total amount of time spent on each coded activity, and average amount of time spent on each coded activity during the design process was calculated. A timeline graph was also constructed showing each group's practices throughout the design process. The purpose of coding in this manner was to understand what design practices from Table 3 each group exhibited and how often the practices occurred.

Coding Confirmation

I did a comparison of coded results to confirm that coding directly from video, and not a detailed transcript of group conversation and activity, effectively captured the practices of a group. To do this, I took a segment of video for group 1 that was coded using video, transcribed that video segment, and then coded the textual data. I chose a video segment that contained a diverse set of codes after coding directly from video so that I could have many opportunities to check for overlap. Comparing the two sets of coded data, the consistent overlap of all but one code confirmed the original coding method. The one exception found by doing a comparison was that instances of the code "Discussing

Depiction,” were not always captured during video-only coding while the group members were also drawing agent depictions.

Inter-rater Reliability

Inter-rater reliability was carried out for the coding process just described. An outside coder focused on a 15 minute segment that was previously coded and found to have a diversity of codes from the original coding carried out by myself, the researcher. The outside coder and I were in agreement for all of the coding that I had previously done (19 coded segments). However, the outside coder double coded five additional instances, and added one coded timeframe that I did not have. This provides an agreement rate of 19/25 => 76%.

Three of the five double coded segments, for which the outside coder observed behavior that I did not identify, consisted of one coder using the “discussing overall game” or “drawing overall game” codes and the other coder using a more refined code for the observed behavior, such as “identifying agents.” The other two double coded segments included the outside coder using the “discussing interactions” code where I used the “recording behavior” code and also where the outside coder used the “discussing depiction” code where I used the “identifying agents” code. Both of these codes were understandably used by the outside coder after reviewing the data.

Findings

The primary finding from the data was that the groups focused primarily on what their game would look like during the design process. They did this through either heavily working on the look of the agents or the look of the game board. There was minimal discussion on how agents in their game should behave and almost no discussion of the programming processes or what computational thinking patterns were present in agent interactions.

Design Practices

In this section the results of the four groups’ design process analysis are shown and explained. For the coding results (Table 7-Table 10) all codes are shown, but it should be noted that there were no instances of the following codes for any of the groups: *Selecting Agent Controller, Recording*

Interactions, and *Identifying CTPs*. Also, codes that were observed to consume a lot of the groups' design time are highlighted in the tables to assist the reader.

Group 1: Angry Nerds

Group 1, who worked on designing a game called "Angry Nerds," spent the most amount of time, approximately 34 minutes, discussing and drawing out representations of what their agents would look like. The drawing was primarily facilitated by the elementary students in the group. In contrast, the group only spent about 11 minutes discussing and recording what would happen in their game and that was primarily facilitated by the undergraduates in the group. Group 1 was also the only group to discuss an interaction that would occur in their game during the design process. Table 7 shows how often these practices occurred and how much time was spent on each.

Table 7. Group 1 coding results

Code	Instances	Total Time Spent	Average Time Spent
Discussing Overall Game	3	0:07:07	0:02:22
Discussing Depiction	8	0:03:14	0:00:24
Drawing Depiction	7	0:31:08	0:04:27
Selecting Agent Controller	0	0:00:00	0:00:00
Discussing Game Board	2	0:03:19	0:01:40
Drawing Game Board	1	0:03:01	0:03:01
Identifying Agents	3	0:01:22	0:00:27
Discussing Behavior	4	0:06:59	0:01:45
Recording Behavior	8	0:03:43	0:00:28
Discussing Interactions	1	0:00:49	0:00:49
Recording Interactions	0	0:00:00	0:00:00
Identifying CTPs	0	0:00:00	0:00:00
TOTAL	37		

A timeline for group 1 (Figure 16) shows the emphasis that the group put on drawing what their agents would look like throughout the design process.

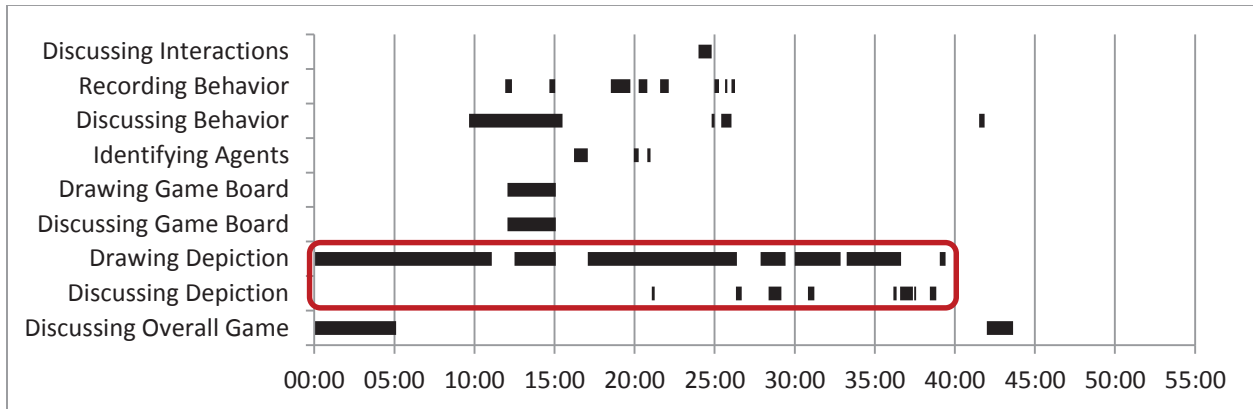


Figure 16. Group 1 timeline of planning process

Group 2: Maze Game

Group 2, which created a maze type game, gave even less attention to what would happen in their game than group 1. This group chose to primarily focus on how their overall game would look. They dedicated almost 24 minutes of the time they spent designing to discussing and drawing representations of the game board. There were also very few instances of the group discussing behavior (4) and then recording their decisions (3), which resulted in approximately 5 minutes of time devoted to deciding what they wanted to happen in their game.

Table 8. Group 2 coding results

Code	Instances	Total Time Spent	Average Time Spent
Discussing Overall Game	4	0:08:09	0:02:02
Discussing Depiction	1	0:00:32	0:00:32
Drawing Depiction	3	0:01:03	0:00:21
Selecting Agent Controller	0	0:00:00	0:00:00
Discussing Game Board	10	0:16:16	0:01:38
Drawing Game Board	9	0:07:40	0:00:51
Identifying Agents	3	0:01:21	0:00:27
Discussing Behavior	4	0:01:45	0:00:26
Recording Behavior	3	0:02:54	0:00:58
Discussing Interactions	0	0:00:00	0:00:00
Recording Interactions	0	0:00:00	0:00:00
Identifying CTPs	0	0:00:00	0:00:00
TOTAL	37		

Figure 17 shows the timeline of group 2's design practices. Throughout the design process, the group switched discussion between what the overall game would be and what it would look like when

played. Very little discussion occurred around the details of their game, such as what their agents would look like and do.

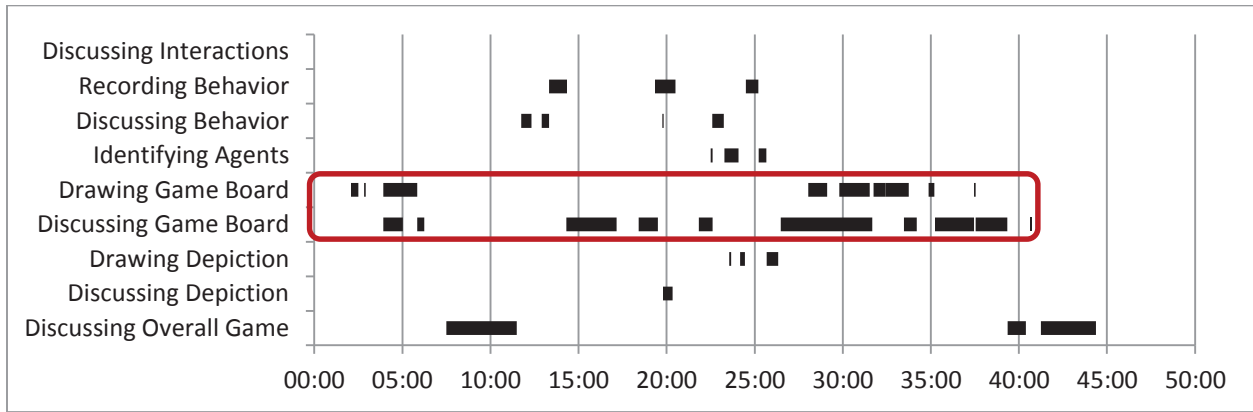


Figure 17. Group 2 timeline of planning process

Group 3: Bear's Night Out

Group 3 created a game called "Bear's Night Out." Their original plan was to have each elementary student in their group design their own game and then they would compile all of the games together as different levels within a final version. The design practices that the group exhibited were fairly distributed with the exception of extreme cases around discussing what the overall game would be and what interactions would occur. Discussing the overall game dominated much of the conversation that this group had during the design process. In contrast, the group did not discuss which agents would interact at all.

Table 9. Group 3 coding results

Code	Instances	Total Time Spent	Average Time Spent
Discussing Overall Game	17	0:17:35	0:01:02
Discussing Depiction	4	0:00:31	0:00:08
Drawing Depiction	7	0:03:46	0:00:32
Selecting Agent Controller	0	0:00:00	0:00:00
Discussing Game Board	6	0:04:32	0:00:45
Drawing Game Board	6	0:03:14	0:00:32
Identifying Agents	10	0:03:58	0:00:24
Discussing Behavior	9	0:08:03	0:00:54
Recording Behavior	7	0:03:41	0:00:32
Discussing Interactions	0	0:00:00	0:00:00
Recording Interactions	0	0:00:00	0:00:00
Identifying CTPs	0	0:00:00	0:00:00
TOTAL	66		

The timeline of group 3’s design practices (Figure 18) shows that the group repeatedly discussed what the overall game would be. They frequently used each student’s game idea as a starting point to refine the individual ideas and the group’s understanding of what they would eventually want to create. This practice of discussing each student’s game in general, then refining the ideas, led to an interesting pattern in the timeline, which is indicated below in Figure 18. This group came close to enacting the full top-down design process that was set up by the Game activity, but stopped short by not considering the interactions that would occur in their game.

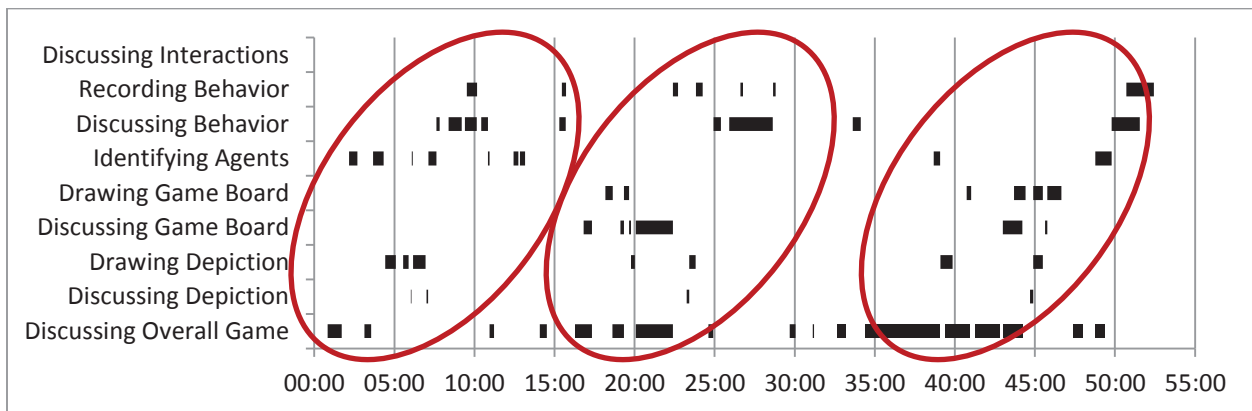


Figure 18. Group 3 timeline of planning process

Group 4: Zombie vs Ghosts

Group 4 created a game called “Zombie vs Ghosts.” Their planning process consisted almost entirely of talking about and drawing individual agents (approximately 32 minutes). At the point that the group felt comfortable getting onto the computers to start creating their game they had not discussed the game board at all, and had only minimally discussed what agents would do within their game outside of their initial brainstorming(1 minute).

Table 10. Group 4 coding results

Code	Instances	Total Time Spent	Average Time Spent
Discussing Overall Game	8	0:07:49	0:00:59
Discussing Depiction	14	0:05:53	0:00:25
Drawing Depiction	7	0:26:00	0:03:43
Selecting Agent Controller	0	0:00:00	0:00:00
Discussing Game Board	0	0:00:00	0:00:00
Drawing Game Board	0	0:00:00	0:00:00
Identifying Agents	3	0:00:27	0:00:09
Discussing Behavior	1	0:01:00	0:01:00
Recording Behavior	0	0:00:00	0:00:00
Discussing Interactions	0	0:00:00	0:00:00
Recording Interactions	0	0:00:00	0:00:00
Identifying CTPs	0	0:00:00	0:00:00
TOTAL	33		

Group 4’s focus on what the agents in their game would look like is apparent in the timeline shown in Figure 19. Once the group had discussed their general ideas for the game, they sporadically discussed design components other than the depictions.

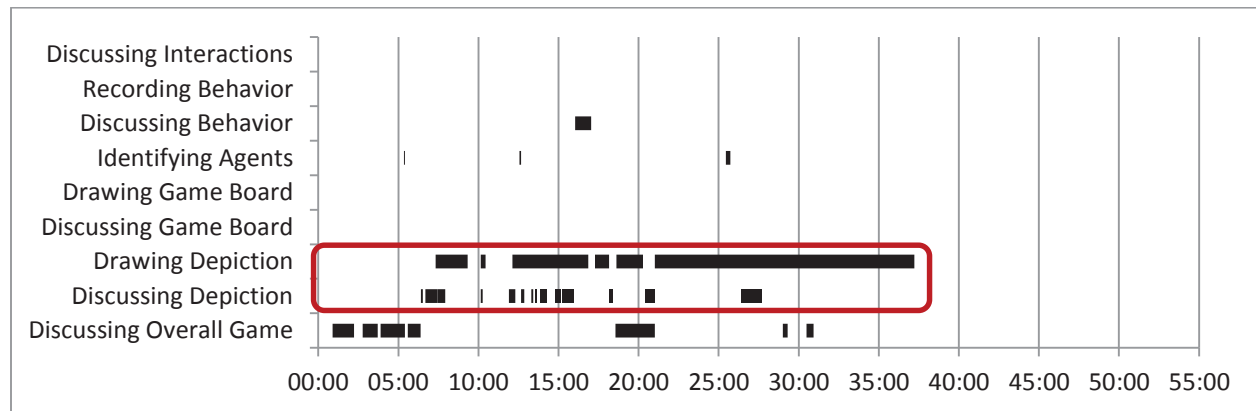


Figure 19. Group 4 timeline of planning process

The intention of having each group design their game first was to make the process of building the game easier and accessible to novices. Once the design was finalized using the *planning document*, the groups would know how their game would look and have example code to draw upon when they needed to program the agent behavior. The following section discusses the common result across all groups when they tried to create their games in AgentSheets and had not fully thought out agent behavior, identified interactions, or researched the CTPs that would help them do the programming.

Creating a Game from the Design

Examining data from the four groups' creation processes showed that the lack of a clear understanding of agent behaviors, and almost no consideration of interactions between agents, resulted in a high need of assistance from the researcher. The following excerpts come from group 4 trying to create their game in the AgentSheets programming environment. There are two undergraduates, U1 and U2, and two elementary students, S1 and S2, in the group. The line numbers pertain to the excerpt only. In the first excerpt the group is trying to figure out how to get their zombie to move using the arrow keys which is the *User Control* computational thinking pattern.

Excerpt 1

Line #	Speaker	Transcript
1	U1	We need our agents to have their movements and stuff.
2	S1	I know how. I know how. [moves AgentSheets computer back to his control]
3		[U2 returns and S2 shifts his focus to his computer and the two pairs seem to separate again.]
4	S1	Watch, I'm gonna make this "ghost dog" move.
5	U1	Let's start with the "zombie," 'cause that's our main guy.
6		[S2 and U2 are having their own conversation to the side. They appear to be trying to open AgentSheets. U2 encourages S2 to participate with the rest of the group.]
7	U1	[shifts his chair to have a better view of the AgentSheets computer] Let's get the "zombie" moving.
8		[S2 still tries to get on the internet, and U2 questions him on what he wants to do on there.]
9	U1	Oh, it has to move onto the background right? [looking over the shoulder of S1 on the AgentSheets laptop] The background... So that's the background.
10		[U1 points at stuff on the screen while S1 works]

11	S1	We need lan.
----	----	--------------

The task in the excerpt above was primarily pursued by Undergraduate 1 (U1) and Student 1 (S1), and ends with an acknowledgement that they can't figure out what to do by playing around with the environment and need the help of the researcher. They have a very general idea of what they want to happen in their game (line 1), but do not understand how to use the arrow keys to do so (lines 2-9), which leads to them needing the researchers help (line 11).

Another episode from later on in group 4's game creation process that resulted in their needing help from the researcher is shown below in excerpt 2. In this excerpt they are trying to make the game end when the "ghost dog," agent is next to the "zombie."

Excerpt 2

Line #	Speaker	Transcript
1	U1	OK, here we go. <i>[S1 looks back at the AgentSheets computer.]</i> When it sees a zombie, it needs to stop and do something, right? Not when it sees it though, when it's next to it. When it's immediately next to it. <i>[inaudible]</i> this. <i>[presses a key]</i> When it's, when it's immediately... When it's next to...
2	U1	Alright, so...
3	S1	<i>[S1 points to something on the screen, U1 is still in control of the computer]</i> Press that, no press that <i>[inaudible]</i> .
4		<i>[U1 presses where S1 points and S1 shows him which option to pick for what the agent should recognize]</i>
5	U1	<i>[inaudible reading of the rule]</i> Um...
6	U1	[S1], so then what? What do I do?
7	U1	This has to be <i>[inaudible]</i> , right? <i>[inaudible]</i> <i>[presses a key]</i>
8	S1	Mm hmm.
9	U1	That's right. What does this zero mean?
10	S1	I don't know. <i>[turns toward rest of room and shouts]</i> lan!
11	U2	lan.

For excerpt 2, Undergraduate 1 (U1) is exploring some of the spatial testing conditions in AgentSheets, such as an agent being, "stacked immediately above," or being, "next to," another agent (line 1). However, he doesn't know which one to use, so U1 and S1 continue to play around with the rules for the agent (lines 2-9). They eventually reach a point where they decide that they need help again and call the researcher over to their group (lines 10 and 11).

These types of “try, then call for help,” episodes were the norm once groups started to create their games in the AgentSheets programming environment. With each group giving so little attention to the behaviors, interactions between agents, and identification of CTPs, they needed considerable assistance from the researcher to get their games to work.

Discussion & Implications

Leveraging students’ previous experiences with games motivated students to design and create a video game, but there was little to draw upon from their designs to get them to complete the activity without considerable help from the researcher. It seems that the students were able to draw on their prior knowledge, but were unable to move to the next step of “making inferences,” or in this case programming *agent behavior, identifying interactions, or identifying CTPs and getting sample code*, as shown in Figure 20 (a refined version of Figure 4 from the literature review).

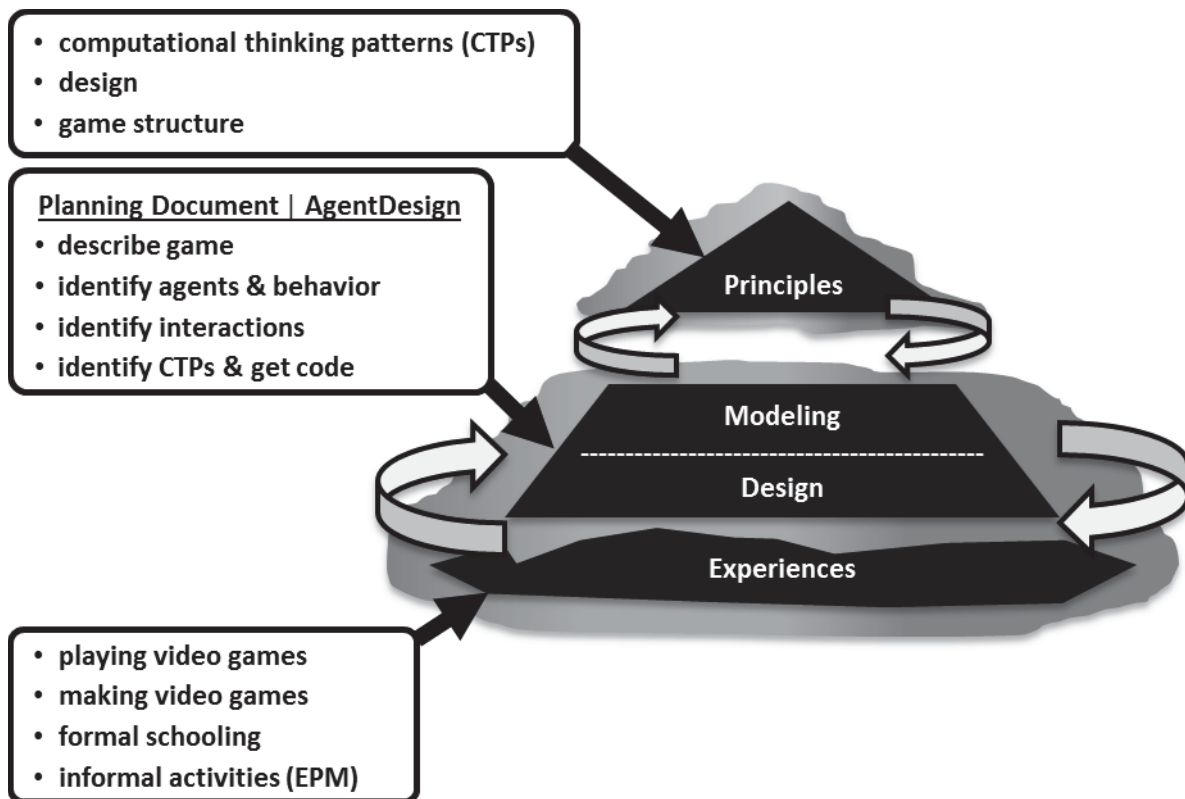


Figure 20. Refined Vygotsky's Theory of Concept Formation and the Iceberg Model

In Figure 20, the terms “modeling,” and “design,” are used to represent the move from concrete experiences to generalizable rules (principles) as well as moving in the other direction—tying formal

language and symbols to concrete experiences. The groups in this study rarely got to the point of fully designing a game and creating an instance of it in AgentSheets, or generalizing from the specific experiences, and instead ended up calling on the researcher to help them with these processes. They were able to draw on their experiences to create depictions of their desired agents and to say some things about the behaviors they wanted their agents to have, but when they got to the point where they had to now move into the programming environment and use CTPs, they consistently called on the researcher. Instead of serving a scaffolding role, to help the group slowly, but iteratively, establish the tools that they would need to do some parts of the Game activity on their own the next time, for whatever reason, I, as the researcher instead ended up actually doing the work for the group.

It is likely that the *mere presence of the researcher* may have prevented students from even imagining, *or identifying with*, the role of programming for themselves. The scaffolding that was necessary to apply principles and to generalize experiences was missing from both the *planning document* and the help that the expert was able to provide the group in this context. The results of this study indicate that a better scaffolding tool is needed and should be present always in the working environment of the students. A better scaffolding tool would provide more assistance in the bidirectional process of moving from concrete experiences to generalizable rules (principles-programming activity) as well as moving in the other direction—tying formal language and symbols (CTPs) to concrete experiences.

Another possibility for why students had difficulty working within the programming environment is that they simply did not see the structure of the game environment as consisting of interactions among agents, behaviors, and game-play environments. Making these three components more transparent could also benefit the students.

In future iterations of this activity, students would benefit from a more explicit mentioning of the specific principles (CTPs) that they are expected to apply when creating agents, behaviors, and

game-play environments. In addition, more transparency about the structure of the games as consisting of interactions between agents, behaviors, and game-play environments would be helpful. Students could also benefit from a more linear structure that would require that they move from one step before going to another. Leveraging students' experiences with the formal language and practices associated with CTPs, will lead to a more robust understanding of programming code and is a necessary aspect of the Game activity.

Study 2

The study presented in this section is intended to create an improved process to engage elementary students in CS through video game development at EPM. From the findings discussed earlier, the major obstacle to students completing the Game activity without help from the researcher stemmed from students moving from their ideas to creating AgentSheets code. Specifically, students did not focus much on discussing agent behaviors or identifying interactions between agents. Most of the focus for designing a game went into how the individual agents or overall game would look.

This iteration of the Game activity incorporated more explicit opportunities to record agent behavior, identify interactions between agents, and associate interactions to CTPs. Figure 21 emphasizes where additional scaffolding is needed to elicit the necessary design practices for elementary students to create their own game.

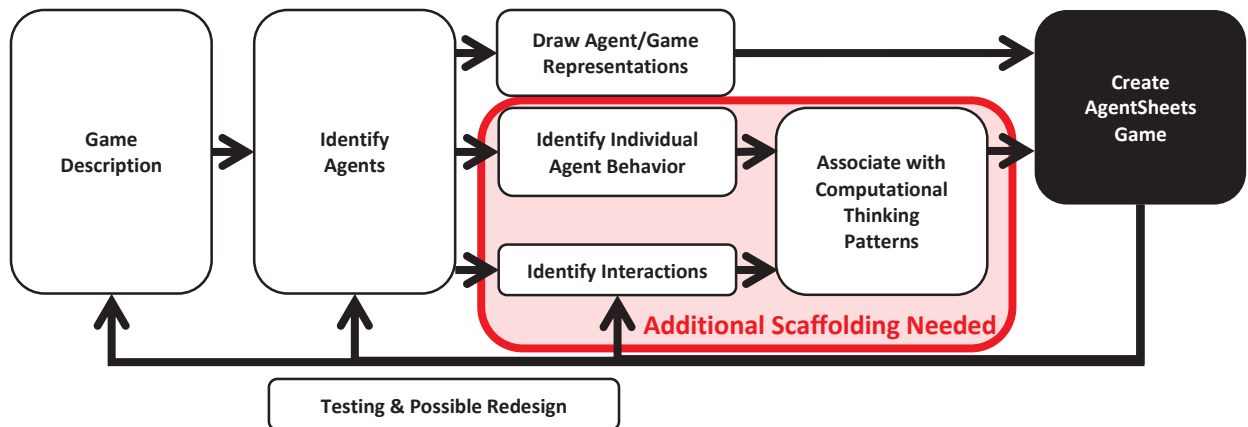


Figure 21. Areas needing improved scaffolding for the Game activity

The areas from Figure 21 that are identified as needing additional scaffolding required that the activity be modified so that sample CTP code was easily accessed from, and relatable to, student descriptions of behavior and interactions. This was accomplished through a scaffolded computer-assisted *planning tool*, called AgentDesign, which I created.

I hypothesized that through the improved AgentDesign *planning tool* the elementary students would be less likely to call out for help and be more likely to use computational thinking patterns to create their games. I further hypothesized that the AgentDesign *planning tool* would help students

understand the structure of a game as consisting of interacting agents, behaviors, and game-play environments. This hypothesis was derived from the notion of scaffolding as it appears in Vygotsky's theory of concept formation and the Iceberg model. The AgentDesign *planning tool* provided additional scaffolding and explicit mention (and choices) of the terminology associated with the principles that students were expected to learn. By applying and reapplying both the principles (in the form of words and symbols) to their experiences with the game design, it was expected that students would be able to build out more generalized rules that could be used and applied both in later parts of their current game design, and future game designs. The scaffolding offered through the AgentDesign *planning tool* provided some guidance, thus alleviating the need for constant calls for help from the researcher.

Research Questions

This study sought to answer the following research questions:

1. What understanding of CTPs do elementary students develop when using the improved scaffolding tool, **AgentDesign**?
2. In what ways does the **AgentDesign** *planning tool* help, or hinder, elementary students' process of creating video games?

Please note that I built **AgentDesign** to address the issues that occurred in study 1.

Study Design

This study focused on the use of the AgentDesign *planning tool* and the design practices of elementary students during the Game activity.

Study Environment

The study environment for this follow-up study continued to be at El Pueblo Mágico (EPM). The theoretical structure of EPM remained the same as what was explained earlier, although there were organizational changes from the previous study that effected how, and who, the elementary students worked with while doing the Game activity.

The first organizational change was that elementary students and undergraduates did not work together in static groups. Instead, the undergraduates were in charge of activity stations and the

elementary students were free to move between these stations as individuals. For example, there were typically 2 or 3 undergraduate students at the Game activity station. For each station, each group of undergraduates at a station moved to a different station every 3 weeks.

The second organizational change for the Game activity was that there were CS experts, and not just the researcher, available to assist with the Game activity. This was partially due to the availability of CS undergraduate students participating as part of an undergraduate research program. The inclusion of other CS experts was also necessary since the researcher was not able to be at EPM throughout the week.

The third organizational change was that the activities were organized by an online social networking platform called iRemix⁹. iRemix was used to organize the activities into pathways and gave students benchmark goals for each activity station. Students were also able to share their work on iRemix through the uploading of videos and pictures or by writing about them in a blog post. An explanation of how the Game activity was facilitated by iRemix is presented in a later section.

How the organizational changes affected the study participants is explained in the following section.

Study Participants

There were four types of study participants for this study; elementary students, undergraduates, CS experts, and the researcher.

Undergraduates

The undergraduate participants continued to be a group of mostly females and pre-service elementary teachers. Their participation in EPM was also still an aspect of their practicum experience for an education psychology course.

At EPM, the undergraduates were in charge of activity stations in groups of 2-4 for three weeks at a time. The undergraduates were given the opportunity to choose their preferred activity stations and were generally assigned to the stations they requested throughout the semester as long as there was

not an abundance of interest for specific stations. Their tasks were to assist the elementary students working through the iRemix pathway for their respective activity station.

All undergraduates were given training on the Game activity prior to the start of EPM for the semester. This training lasted approximately 40 minutes and covered how to use the AgentDesign *planning tool* and the AgentCubes Online programming environment. During the training, they observed me, the researcher, design and create a simple game using the two resources.

Elementary Students

The elementary student participants at EPM, like the undergraduates, had a similar composition to that of the earlier study. They were a representational group of the overall school population and were in grades 2-5 (shown in Table 4). In total 23 students started the design process by creating an AgentDesign account, but many other students participated in the activity station that did not design games. Some of these students simply played games on the AgentCubes Open Arcade, or simply started to make a game without designing it first. Of the 23 students that created an **AgentDesign** account, 18 actually started designing a game, with one student creating two separate designs. The gender distribution of the student participants was 6 girls and 12 boys.

As explained earlier, due to the organizational changes, the elementary students' participated on more of an individual basis for this study. The students were able to move between activity stations as they wished and were not required to work with a group on any activity.

CS Experts

There were 3 CS expert participants at EPM, excluding the researcher, with at least one CS expert at the Game activity station for each day of the after school program. The experts had a background in CS, but not necessarily any prior CS pedagogical content knowledge. One of the experts was a graduate student taking a course on qualitative research, and two others were undergraduate students pursuing degrees in CS. One undergraduate expert had prior experience with the Game activity at EPM from the previous semester, while the other CS experts were new to the activity. All CS experts

participated in the training with the undergraduates to learn about the AgentDesign *planning tool* and AgentCubes Online.

Researcher

The researcher had a similar role as explained earlier for study 1. However, he only attended EPM for one day a week on a consistent basis. If the CS experts felt that they needed assistance, they could have requested that he come on other days, but this only happened due to scheduling issues when the CS experts were not able to attend EPM.

Revisiting the Game Activity

The overarching goal of the activity for study 2 was still for students to create video games with minimal help. The more specific goals, that students will understand the purpose of design; understand CTPs (algorithms); develop ways of evaluating their work; and align their identity with CS, were also still present in the activity. However, due to both technical limitations and in order to introduce an alternative scaffolding tool, the programming environment and design scaffolding tool have been changed from what was used in study 1. Additionally, the guidance and instructions for how to complete the activity was moderated through a social media network, iRemix. These changes are explained below.

The AgentCubes Online programming environment was used instead of AgentSheets for two reasons. The primary reason was that the available technology at EPM, chrome books, would not allow for AgentSheets to be installed. A secondary reason was that students had issues saving their work in the prior years of offering the Game activity at EPM when using AgentSheets. Since AgentCubes Online is web-based, all of the students' work was automatically saved and instantly shareable through the Scalable Game Design Arcade. Although it was new to the activity, the AgentCubes Online programming environment description is located in Appendix B, since it is so similar to AgentSheets in the ways it was used and functions for this study and a complete description here is not necessary.

As the intervention for this study, the scaffolding tool used to support students through the design process was also switched to an interactive, web-based application that I developed. This

scaffolding tool will be referred to as the *AgentDesign planning tool*, and its purpose was to improve the support for students as they went through the design process. The tool is further described in a later section.

Using iRemix, the students accessed activity pathways that they could choose to complete. Each activity station at EPM had a pathway represented on iRemix that offered a beginning, advanced, and expert level task to complete. For the Game activity, I asked students to design a game using AgentDesign at the beginner level, create the game using AgentCubes Online at the advanced level, and then share the game using iRemix, and possibly make changes, at the expert level. The advantage of using iRemix was that the links to the design scaffolding tool and programming environment, as well as helpful videos, could be easily accessed by students with clear instructions on what to do with them. Below are screenshots of the beginner, advanced, and expert level tasks that the student would see while working on the Game activity. Presented with each pathway level are figures showing what parts of the Game activity process from study 1 (see Figure 15) that each level focused on.

The beginner level pathway focused on the design aspect of the Game activity process. Students should have described their games, identified agents, decided on agent representations, described individual agent behavior, and identified interactions along with corresponding CTPs during this part of the activity using a tool that I developed called **AgentDesign**.

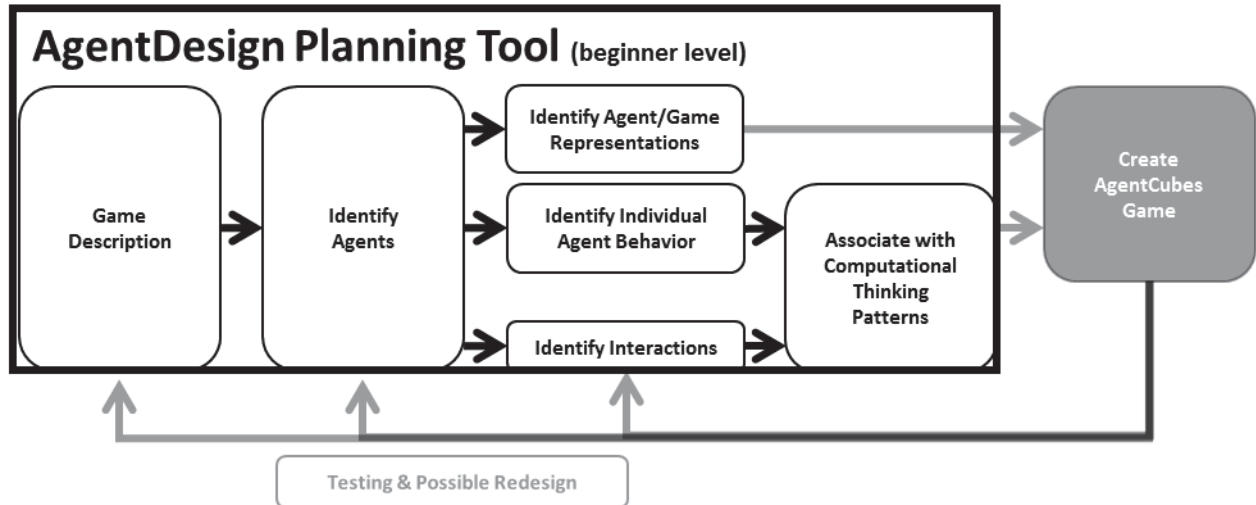


Figure 22. Focus of Pathway Beginner Level for Game Activity Process

The figure below provides a screenshot of what an elementary student would have seen as a prompt for the beginner level task for the iRemix Game activity pathway. The beginner level prompt for the pathway asked the students to design their game using the AgentDesign *planning tool*, but first encouraged them to look at games that had been made using AgentCubes. Asking the students to look at previously made games was meant to provide some context to the students about what could be created using the programming environment. Links were available to the students on this beginner level page for both AgentDesign and a collection of games called the Scalable Game Design Arcade, so the pathway, and iRemix, was the primary access point of the Game activity for all students.

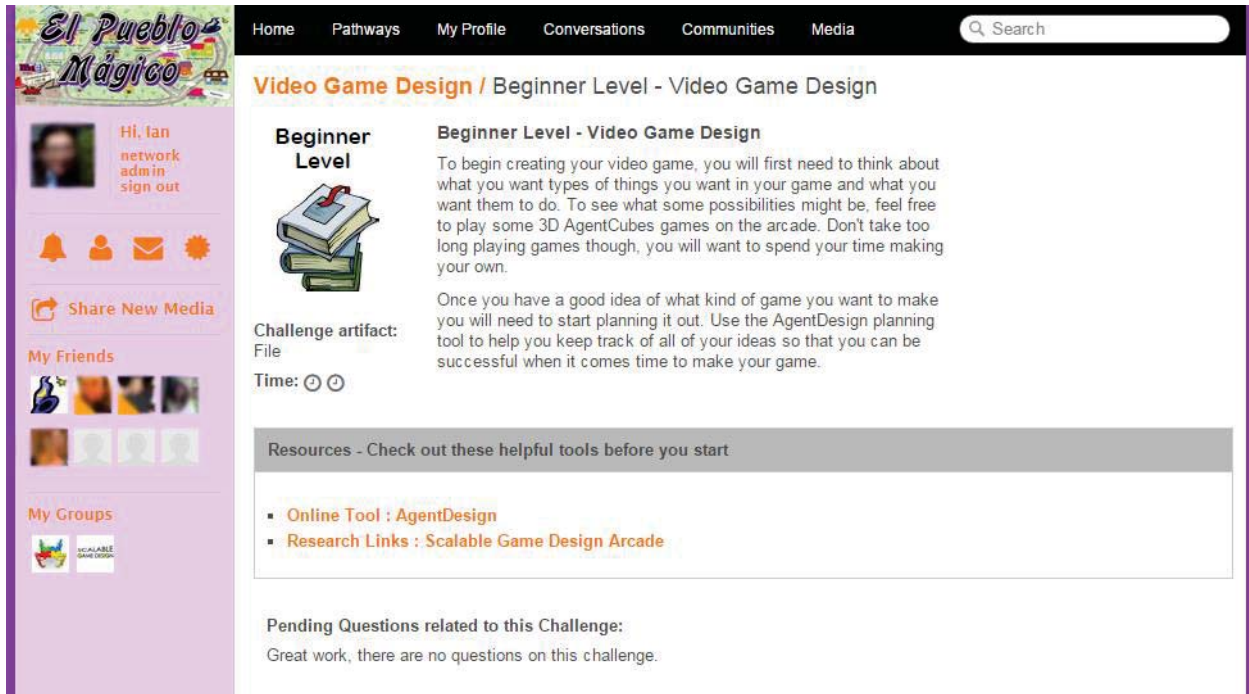


Figure 23. iRemix Pathway - Beginner Level

Creating the game was the focus of the second level of the Game activity pathway, as shown in the following figure.

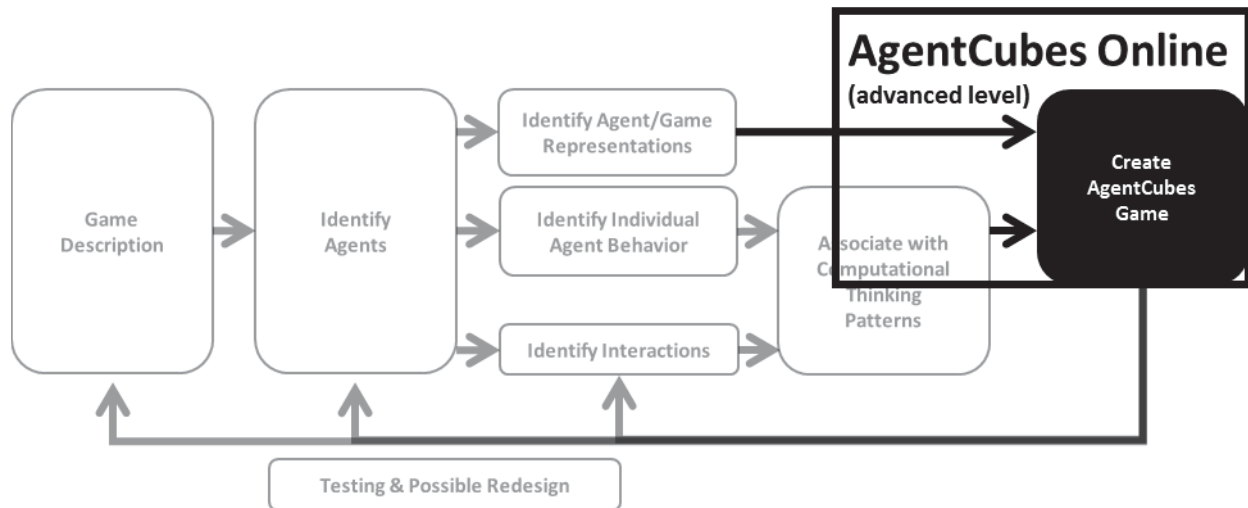


Figure 24. Focus of Pathway Advanced Level for Game Activity Process

Once students had completed the beginner level pathway and designed a game using my AgentDesign tool, they could then move on to creating their game using AgentCubes Online as part of the advanced level of the Game activity pathway (page shown below). The intention was that the

students would use a summary of their design, called the summary page and explained later, to create their games. Along with the prompt asking students to use their design summary page to assist them in creating a game, a short video explaining how to get started using AgentCubes Online was available for the students to watch. Once the students were ready to begin creating their game they could use the available link to access AgentCubes Online.

The screenshot displays the iRemix Pathway interface for an 'Advanced Level' challenge. The top navigation bar includes 'Home', 'Pathways', 'My Profile', 'Conversations', 'Communities', and 'Media', along with a search bar. The main content area is titled 'Video Game Design / Advanced Level - Video Game Design'. On the left, there is a sidebar with a user profile for 'Hi, Ian network admin sign out', a 'Share New Media' button, and sections for 'My Friends' and 'My Groups'. The main content area features a 'Challenge artifact' section with a flask icon, a video player titled 'AgentCubes - Getting Started', and a 'Resources' section with a link to 'AgentCubes Online'. Below the resources, there is a section for 'Pending Questions related to this Challenge' which states 'Great work, there are no questions on this challenge.'

Figure 25. iRemix Pathway - Advanced Level

Finally, the feedback loop from players of a student's game was used as part of the expert level of the Game activity pathway, shown below. Using feedback from other people playing the student's game, the student could choose to redesign different parts of their game, such as the overall idea, the agents, or how the agents would behave in the game.

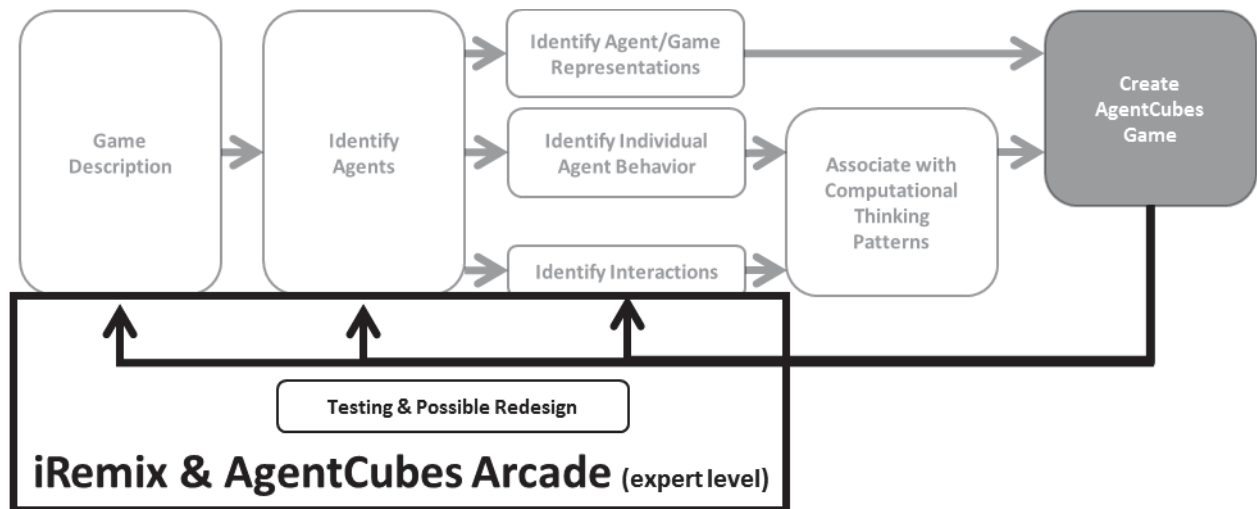


Figure 26. Focus of Pathway Expert Level for Game Activity Process

The task for the expert level of the Game activity pathway (shown below) was for students to have their games tested and then decide if they wanted to change any part of the game. Students reaching this expert level would need to have a working game and be given feedback about its playability. The reasoning for incorporating the social feedback aspect of the pathway was to emphasize the importance of the social aspect of any software development, which is especially true for the development of video games.

Home Pathways My Profile Conversations Communities Media Search

Video Game Design / Expert Level - Video Game Design

Expert Level **Expert Level - Video Game Design**

This final challenge is an important part of creating any game. You need to get some feedback from other people to see how fun it is to play.

Ask anyone to play your game and have them answer the following questions:

1. What was great about the game?
2. What wasn't so great about the game?
3. Were there any bugs (places where the game was broken)?
4. Were there any ways to cheat?

Challenge artifact:
Blog

Time: ☹

Once you get that feedback, fix what you feel needs to be fixed. You may want to keep something that a player didn't like, and that's ok, just try to make something that you are happy with.

Resources - Check out these helpful tools before you start

- [Online Tool : Get Feedback](#)

Pending Questions related to this Challenge:
Great work, there are no questions on this challenge.

Hi, Ian
network admin
sign out

Share New Media

My Friends

My Groups

Figure 27. iRemix Pathway - Expert Level

Using iRemix as the portal to access the activity, the students were given information and access to the necessary tools (AgentDesign and AgentCubes Online) as well as provided prompts to clarify each part of the activity. In the following section, the intervention for the second study (AgentDesign), and its theoretical basis is described through an explanation of the conceptual framework of the study and a description of the tool.

Conceptual Framework

The conceptual framework for study 2 focused on supporting the elementary students in thinking about, recording, and accessing CTPs so that they would be able to successfully build a game using AgentCubes Online. The five conjectures that were developed as part of this process are shown below in Table 11. Figure 28 shows the conjecture map (Sandoval, 2004, 2014) for these five conjectures and each are described in this section.

Table 11. Study 2 Conjectures

Associated Theory	Conjecture	Description
Concept Formation	1:	<i>Designing a game with the tool will help the students learn algorithms.</i>
Scaffolding Concept Formation	2:	<i>Students need to COMPLETE the design process to learn about the CTPs and to create their game.</i>
Concept Formation	3:	<i>Students will learn the PRINCIPLES of CS through the scaffolded process of matching everyday language with CTPs.</i>
Scaffolding Study 1 Results	4:	<i>Students will focus more on other aspects of the design process, and less on drawing pictures of aspects of a game while they design if they are not given the opportunity to draw pictures.</i>
Study 1 Results	5:	<i>The tool will help the students build the game.</i>

The Game activity was intended to provide students with the opportunity to develop a sense of individual CTPs that were free from any individual game context while they created their own video games. Each CTP (algorithm, which represents a CS principle) represents a different phenomenon that programmed agents can enact (e.g. collision, generation, transportation, etc.). By using each individual CTP multiple times in a particular game design it was expected that students would establish more general principles regarding those CTPs that could then be applied to any future game. For this to occur, the activity related the principle of CTPs to the students' own experiences and ideas and allowed for the students to develop the connection between the two. This was the role of the AgentDesign *planning tool*, especially in cases where there is no formal teacher of the activity like EPM.

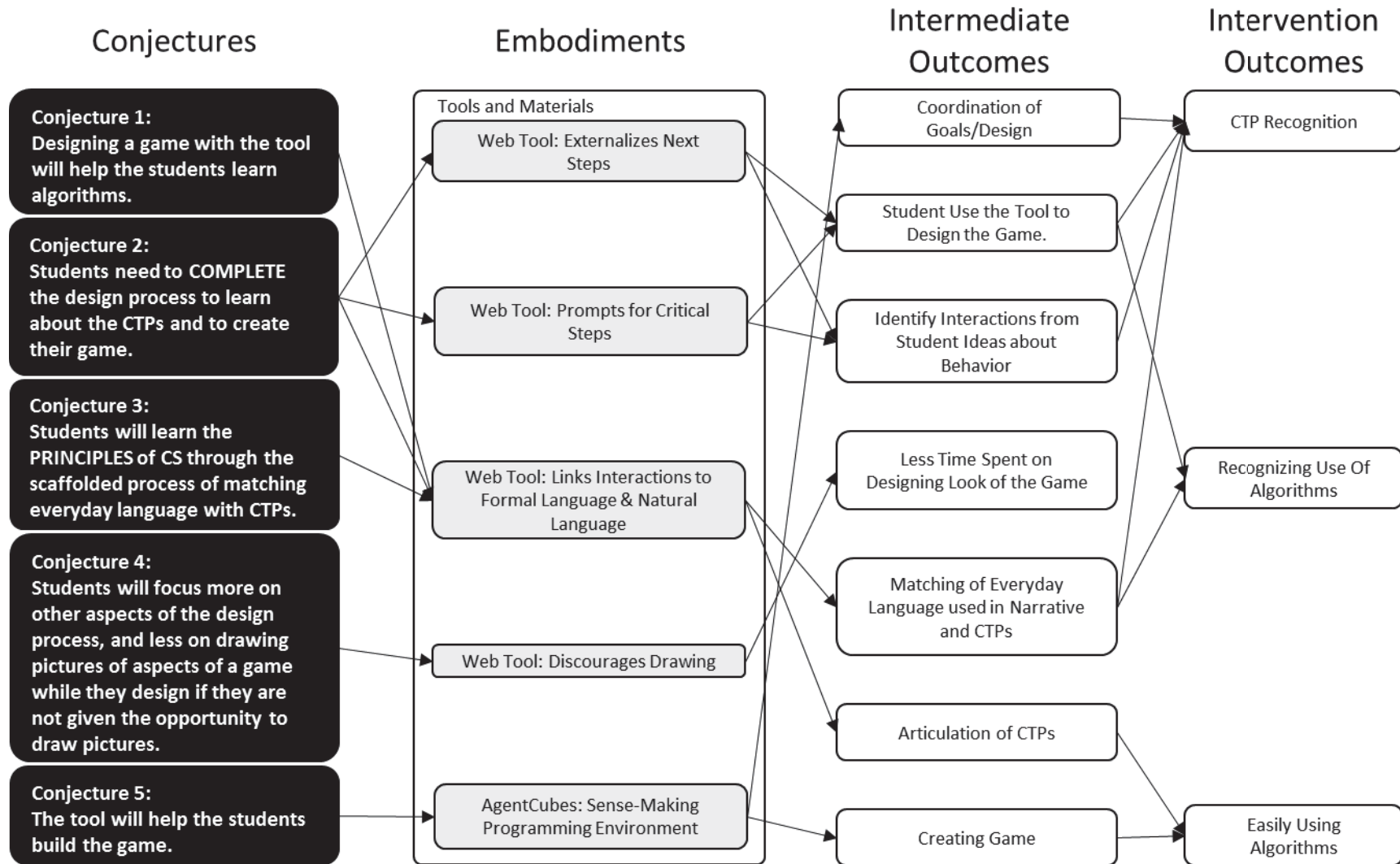


Figure 28. Study 2 Conjecture Map

Conjecture 1: *Designing a game with the tool will help the students learn algorithms.*

The intention of the AgentDesign *planning tool* was not only to help students make games from their own ideas, but also to help them learn CS principles. Algorithms are an important CS principle for computer scientists to be aware of because algorithms provide solutions to common problems regardless of the specific language the computer scientist is working within. Furthermore, learning about algorithms becomes an invitation to the CS discipline by giving students access to a more authentic CS experience than simply learning to program. For games created using agent-based programming environments an important set of algorithms are CTPs. The CS principle of “algorithms,” in this study express common ways to program game behavior like collisions, pushing, absorption, tracking, and many other phenomena that occur in visual games.

Additionally, from study 1, the evidence showed that the pencil-and-paper scaffolding tool did not support the students in a way that they could access important algorithms (CTPs), let alone learn about them. In Figure 29 the crossed out words represent the design practices and CS principles that students did not complete or attain, words that are not crossed out represent observed student practices from the data in study 1. Using the new scaffolding in the form of the AgentDesign *planning tool*, it is hypothesized that the students will now be able to both access and learn about algorithms using a centralized, scaffolding tool.

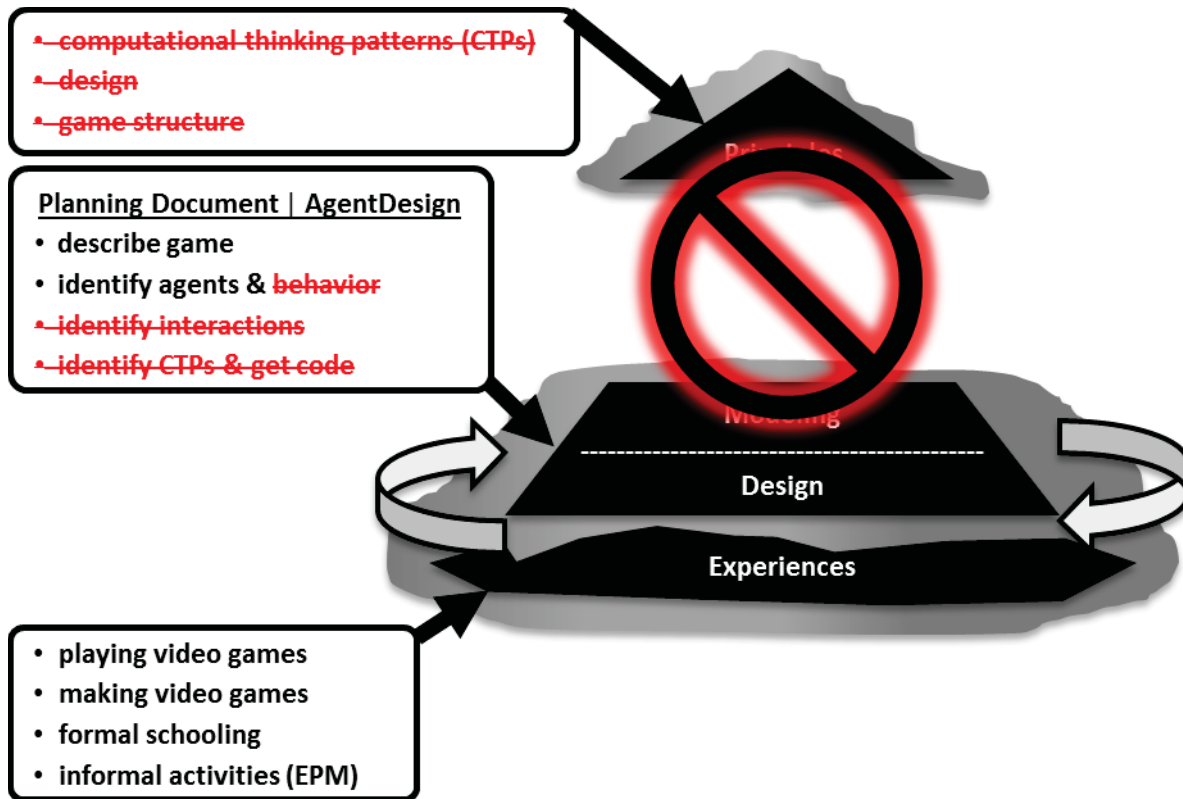


Figure 29. Disconnected Learning in Study 1

The theoretical basis for this conjecture is based on the formative assessment model (Black & Wiliam, 2009), Vygotsky’s theory of concept formation (Vygotsky, 1986), and the Freudenthal Institute’s iceberg model (Doorman & Gravemeijer, 2009; Webb et al., 2008).

AgentDesign was designed taking into account where the students would be coming from and where it wanted them to go. Students taking part in the Game activity will likely have played many video games before, know of many stories, and maybe even made a video game in the past. As part of the design, a goal of AgentDesign was to lead students to the CTPs that they would need to create their games using AgentCubes Online. Through this process, students would also be able to connect ideas that were theirs, and made sense to them, to formalized CS principles. This connection building was influenced by Vygotsky’s theory of concept formation and the iceberg model (see Figure 4). Concept formation is a cyclical process in which students make sense of formal knowledge through processing

their own knowledge and experiences and vice versa. For example, using AgentDesign a student can decide that something must happen if two agents touch one another (which she would have come up from experiencing other games), connect that idea to the formal CTP (algorithm) called collision, and then make sense of future collision scenarios using the formal algorithm she just learned about. The AgentDesign *planning tool* was created to mediate this process. The process of adding the computer-assisted scaffolding tool (AgentDesign) is similar to that used by Basawapatna's (2012). Basawapatna's (2012) developed a visual tool, called the Simulation Creation Toolkit, to assist middle school students in creating science simulations using animations of CTPs. He found that students using his toolkit were able to create simulations with heavy guidance. In comparison to my own work, the focus of his research was on the area of Figure 29 that is toward the bottom. Using his toolkit students would be assisted in building experiences with CTPs while making games and simulations. I am working to take that idea one step further by using a scaffolded tool to assist students in relating their current and past experiences to CTPs in order to develop conceptual understanding.

As just mentioned, the embodiment of this conjecture in the study was carried out through AgentDesign. Specifically, scaffolding student learning of algorithms is done when the student describes an interaction between agents, and then connects that description to a CTP. This is explained later in the description of the AgentDesign *planning tool*.

Conjecture 2: *Students need to COMPLETE the design process to learn about the CTPs and to create their game.*

The AgentDesign *planning tool* was developed to scaffold the process of designing a game, so that the design could be used to assist in creating that game in a programming environment. From the results of study 1, it was clear that additional scaffolding was needed to support descriptions of agent behavior, identifying interactions, and connecting those interactions to CTPs (see Figure 21). In study 1, none of the 4 groups that were examined connected their ideas to CTPs during the planning process (see

Table 7 - Table 10). According to Vygotsky's theory of concept formation (Vygotsky, 1986), students would not be able to learn about CTPs without connecting their own experiences and ideas to the formal knowledge that is provided within the AgentDesign *planning tool*. Since the groups in study 1 did not complete the *planning* document, I believe that they were not prepared to begin creating their games since they never accessed this formal knowledge. Using scaffolding, AgentDesign guides students through the design process so that they can see what the next steps are, as well as access the formal knowledge that would be needed to create their games. And by completing the design process, not only will students access this knowledge, they will be provided the opportunity to *USE* the terms associated with that new knowledge.

Transitioning the scaffolding to a more interactive medium, like a web application, has been shown to be useful in supporting student learning. Various affordances of the computer technology make it easier for students to manipulate their ideas and representations. White et al. (2002) found that using a software based tool to scaffold student activities helped them to learn about inquiry and be metacognitive. In other work that has used technology to support student learning, Wu et al. (2000) found that the software tool they used, called eChem, supported student conceptual development of chemical representations through building and viewing models and representations.

Within the AgentDesign *planning tool*, the embodiment of this conjecture was done through the explicit prompts explaining what the students needed to do. These prompts led students to important information about CTPs that would both assist the students in creating their games, as well as present the CTPs in a way that made sense in relation to their ideas.

Conjecture 3: *Students will learn the PRINCIPLES of CS through the scaffolded process of matching everyday language with CTPs.*

In order to mediate conceptual development, the design process prompted students to identify CTPs from their descriptions of behavior and interactions. Most students designing a game will likely

have played other games in the past and will be able to recognize CTPs that they would need for their game from an available list presented to them. Recall that CTPs are a set of algorithms that represent common phenomena in games. These can include, but aren't limited to, collision, absorption, tracking, pushing, etc.

Using the scaffolding tool, AgentDesign, the students were presented with a list of CTPs and were asked to draw upon both their prior experiences with games and their own language in the design to make an association of their everyday language descriptions to the CTPs (Figure 33). Developing this relationship between experiential knowledge and formal knowledge is an important aspect of concept development (Otero & Nathan, 2008; Vygotsky, 1986). This activity focused directly on the cyclical nature of concept formation to not only relate students' prior experiences and knowledge to certain CTPs, but also provide symbolic principles with which to interpret past and future experiences with gaming and programming CTPs.

In the embodiment of this conjecture (AgentDesign), students are provided opportunities to make sense of formal CTPs and the terms that describe them, like collision, in ways that make sense to them. They begin designing by only using their own thoughts and words, but then have opportunities to connect those everyday words to the formal terms and concepts presented to them using the AgentDesign *planning tool*.

Conjecture 4: *Students will focus more on other aspects of the design process, and less on drawing pictures of aspects of a game while they design if they are not given the opportunity to draw pictures.*

Analysis of student behavior from study 1 showed that discussing or drawing out visual aspects of the game during the design process was a common focal point for 3 of the 4 groups (see Table 7 - Table 10). The data from study 1 also found that the high focus on the look of the game (agent depictions and game board drawing) that students had was at the expense of designing what they wanted to happen within the game (agent behavior, interactions, CTPs). Given that finding, a design

choice was made to not provide an opportunity for students to draw during the scaffolded design process. This was intended to reduce the amount of time that students spent focusing on the look of their game.

The embodiment of this conjecture in the activity was within the *AgentDesign planning tool*, which is discussed in detail later. Within *AgentDesign*, students were not given the ability to draw anything. All descriptions of the visual aspects of the game were carried out through text.

Conjecture 5: *The tool will help the students build the game.*

Results from study 1 provided evidence that since students did not complete the pencil-and-paper *planning document* they had difficulty creating their game using *AgentSheets*. The *planning document* was intended to address many of the questions that the students commonly asked, such as how to make an agent move using the keyboard (the user control CTP) or how to make one agent die when it's touched by another agent (collision CTP). The scaffolding tool used in this iteration of the study was designed to provide clearer guidance of what needed to be done for the students to be successful.

The primary embodiment of this conjecture for the activity were the prompts that students were given while working through the design process. In study 1, the students generally skipped the entire page that asked them to identify interactions and CTPs. In contrast, the *AgentDesign planning tool* clearly required students to move on to the next necessary step, which addresses the previous problem of students skipping parts of the design process. The process of designing a game using *AgentDesign* for this study was sequential and forced, meaning that the students will complete their designs. And if the students complete their designs, they will have lots of information, in the form of CTPs and sample code, to build their games.

An additional embodiment of this conjecture is that the CTP information was located within the *planning tool*, instead of on a separate resource. The students were able to see their everyday language descriptions alongside the formal language descriptions of CTPs and sample code. Being able to see why

the sample code was relevant, and having easy access to it, would allow the students to use their design information to create their games without needing to ask an expert to explain every common phenomenon that CTP algorithms address.

The AgentDesign Planning Tool

The AgentDesign *planning tool* was created to address the difficulties students had transferring their own ideas to the programming environment. This was a new scaffolding tool for the Game activity and was developed by me. It was written using JavaScript, HTML5 and PHP. The intention of making the tool web-based was to provide cross-platform access, which includes not only Apple and Windows operating systems, but also tablets and smartphones. This was a necessary design choice as more schools are moving away from traditional computers and towards tablets.

The purpose of transitioning the scaffolding tool to the AgentDesign *planning tool*, from the previously used pencil-and-paper *planning document* from the first study, was to address the problems identified in study 1. AgentDesign guided students through a highly scaffolded, step-by-step process of describing the game, identifying agents, describing agent behaviors, identifying interactions between agents, and connecting interactions to CTPs and sample code. This was intended to assist students in developing the disciplinary principles of algorithms and design, and also identify key aspects of video game structure. There was also less emphasis on the look of the game and agents by not having a space for drawing to be done within the tool, since that may be a reason that students did not focus on agent behavior using the pencil-and-paper version. The design practices that AgentDesign focuses on are shown in Table 12.

Table 12. AgentDesign Planning Tool Design Practices

Design Topic	Design Practice Details
1. Initial Idea	Recording the initial game idea.
2. Agent Names	Identifying and recording agent names from the project description.
3. Agent States	Naming different representations an agent will have.
4. Agent Behavior	Describing individual agent behavior.
5. Identifying Interactions	Identifying which agents will interact.
6. Interaction Description	Describing the interaction between two agents.
7. CTP Association	Associating CTPs to an interaction that has been described.

AgentDesign provides four sections that guide students through the design practices listed in Table 12. The four sections are the (A) Project Description, (B) Agents, (C) Selecting Interactions, and (D) Building Interactions. There is also a fifth section of AgentDesign, the (E) Summary, that does not have students work through any of the design practices, but allows them to view their design as a whole. The association between the sections of AgentDesign and the design practices from Table 12 are shown in Table 13.

Table 13. AgentDesign Planning Tool Sections and Design Practices

AgentDesign Section	Design Practices
Project Description	1. Initial - Recording the initial game idea. 2. Agent Names – Identifying and recording agent names from the project description.
Agents	3. Agent States - Naming different representations an agent will have. 4. Agent Behavior - Describing individual agent behavior. 6. Interaction Description - Describing the interaction between two agents.
Selecting Interactions	5. Identifying Interactions - Identifying which agents will interact.
Building Interactions	6. Interaction Description - Describing the interaction between two agents. 7. CTP Association - Associating CTPs to an interaction that has been described.
Summary	None - Review of design as a whole.

The following sections elaborate on how each design practice is enacted within AgentDesign.

Project Description

The project description section of AgentDesign asks students to write their initial game idea (design practice 1) and identify agents (design practice 2). These tasks were purposefully placed within the same section so that a visual confirmation could be made for each agent identified. Figure 30 shows an example of what the project description section looks like when a game has been described and agents are identified.

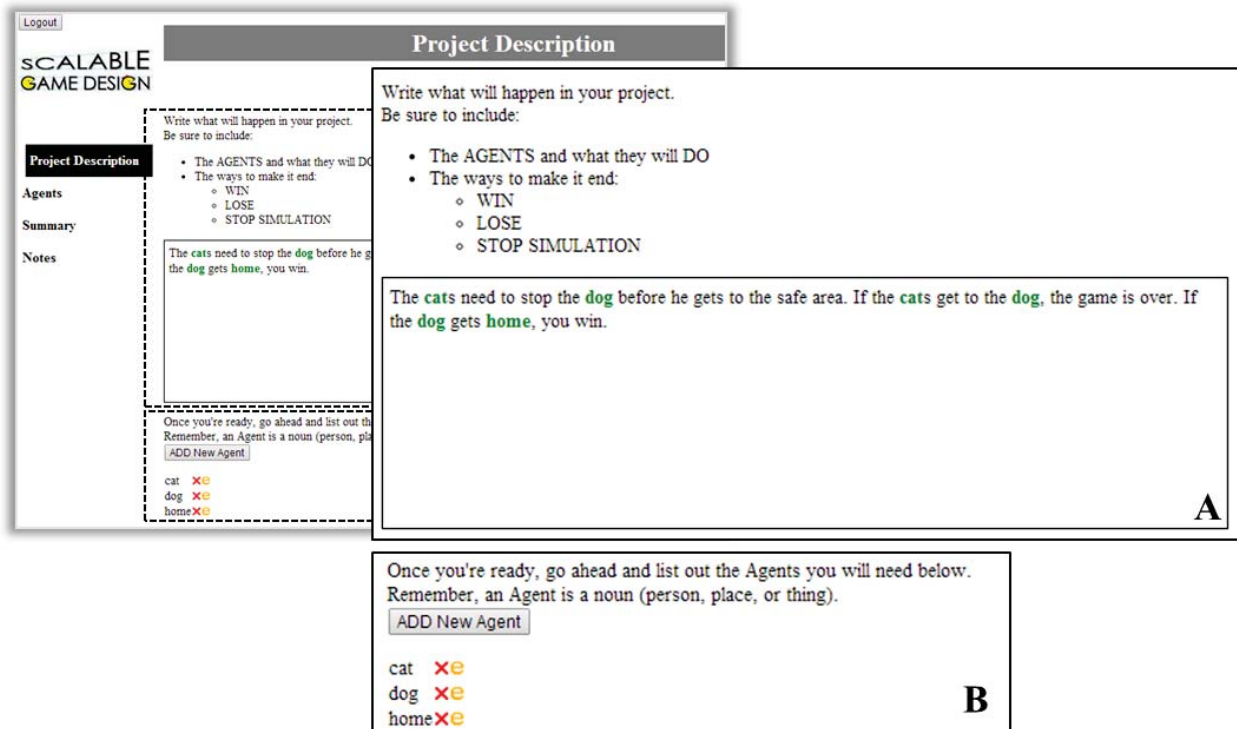


Figure 30. AgentDesign Planning Tool - Project Description

Initial Idea

The initial idea (Figure 30A) prompts a student to “Write what will happen in your project.” It encourages the naming of agents and descriptions of behavior. The prompt also asks students to elaborate how the game or simulation, depending on what the student is making, will end.

A text box is available for the student to write out the project description. It is not required, or expected, that everything needed for the game will be recorded at such an early point, but it provides an area for the student to start brainstorming. As the student realizes that parts of the game description

have not been included, she is free to return to this section and add more information. Also, since the AgentDesign *planning tool* is designed for elementary students, the text for the project description is automatically saved once the student clicks out of the box.

Agent Names

Also in this section is a prompt to identify anything that could be an agent named in the project description, which would be any noun in the description (Figure 30A). An agent is identified and named using an “ADD New Agent,” button and a text input prompt within this section (Figure 30B). As the individual agents are identified they are listed below the naming prompt and their corresponding name changes to the color green in the project description (Figure 30A). The color change provides an indication to the student for which agents have been accounted for and which still need to be identified.

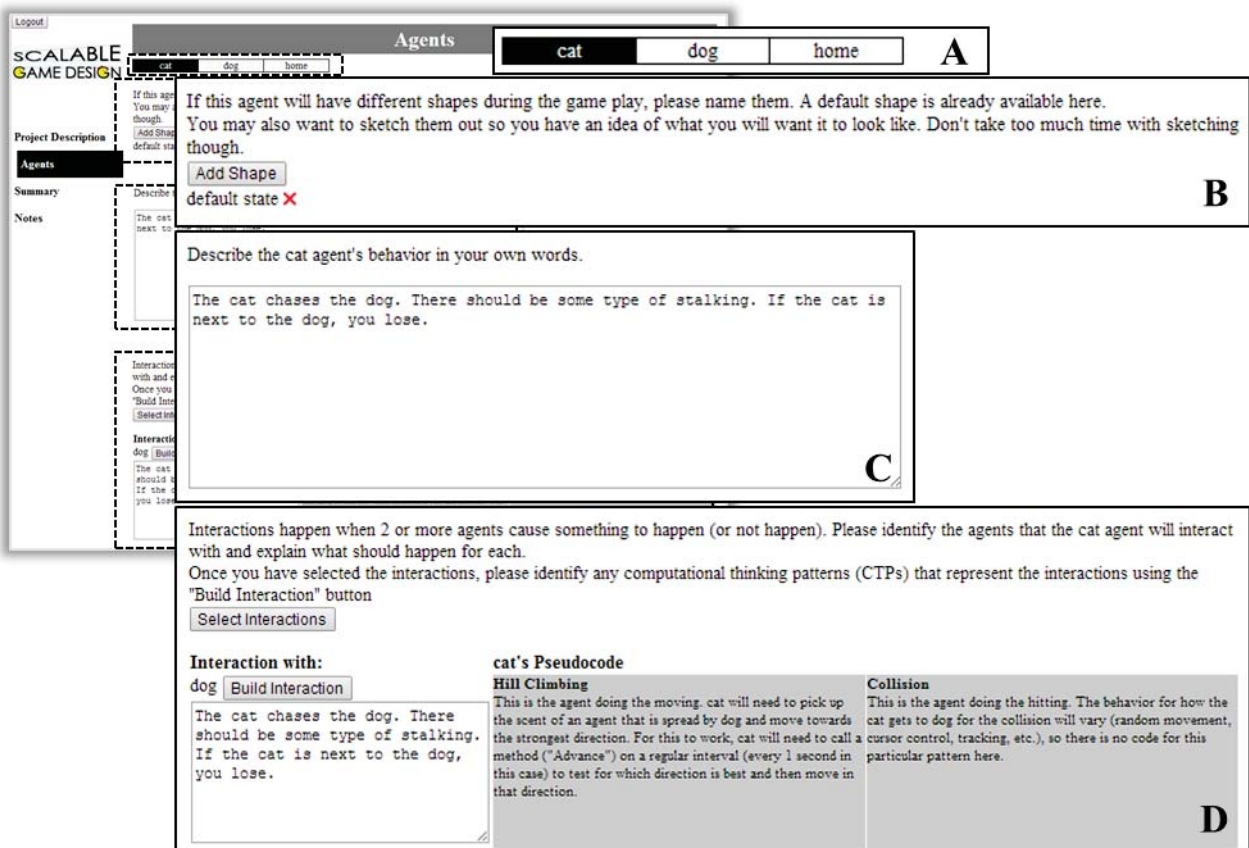


Figure 31. AgentDesign Planning Tool - Agents

Agents

For each agent identified in the project description section, a new space is created in the agents section (Figure 31) of the AgentDesign *planning tool* for agent states to be identified (design practice 3), behavior to be described (design practice 4), and interactions to be identified and described (design practices 5 & 6). Agents can be selected from a list at the top of the section, the “cat,” agent is currently selected in Figure 31A.

Agent States

In some cases an agent may have different states during the game play. The computer-assisted planning tool uses a button labeled “Add Shape,” and a prompt to ask whether or not an agent will look differently during the gameplay (Figure 31B). Only the naming is available here, and not an ability to draw out the agent’s representation. This was a purposeful choice due to the students focusing on drawing in prior research.

Agent Behavior

A text area is available to describe an agent’s behavior in everyday language (Figure 31C). This area is specific to each agent, and is intended to be a refinement of the general description written in the project description section. Similar to the functionality of the project description text box, the information is saved once the student clicks out of the box, so deliberate saving is not necessary.

Selecting Interactions Section

Within each agent’s section a button labeled “Select Interactions,” is available that opens a popup window (Figure 32) allowing for the selection of any interactions that the current agent will have with other agents, the user, or all agents (design practice 5). The window shows the current agent’s behavior description and checkboxes for all of the agents they had previously identified from the project description as well as the possible user of the game. For each agent they identify as having an interaction with the current agent, a new textbox (left side of Figure 31D) is made available in the agents section for a description of that interaction to be written. The selecting interactions window (Figure 32)

also highlights all previously identified agents as green as an indicator for the student to see what agents have already been described in the selected agent's behavior.

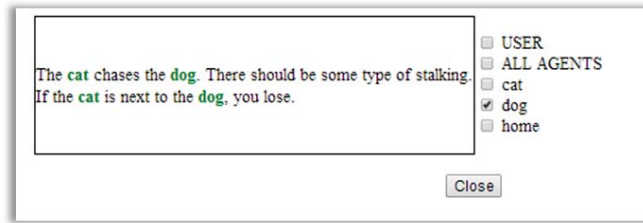


Figure 32. AgentDesign Planning Tool - Selecting Interactions Window

Building Interactions

Building interactions is accessed through a button labelled “Build Interactions,” in the agents section (Figure 31C). A “Build Interactions,” button is available for each interaction that was selected earlier. Clicking on the button opens a window (Figure 33) where the interaction description is available along with descriptions of CTPs. Describing interactions (design practice 6) and then choosing what CTP aligns with that description (design practice 7) has the intention of moving the student from everyday language to programming language and development of understanding algorithms through associating student ideas with CTPs.

Interaction Description

While it may be obvious to explain what an agent will do on its own, the students may not consider what will happen in cases where it interacts with another agent. A good example of this would be all of the things that happen when the frog in “Frogger,” is hit by a car; such as change the frog depiction, give a signal that the player lost, and stop/restart the game. This space allows for a refined written description of what will happen in interaction using everyday language. There are two locations where an interaction description may be written, one is in the Agents section (Figure 31C, left) and the other is in the Building Interactions window (Figure 33A).

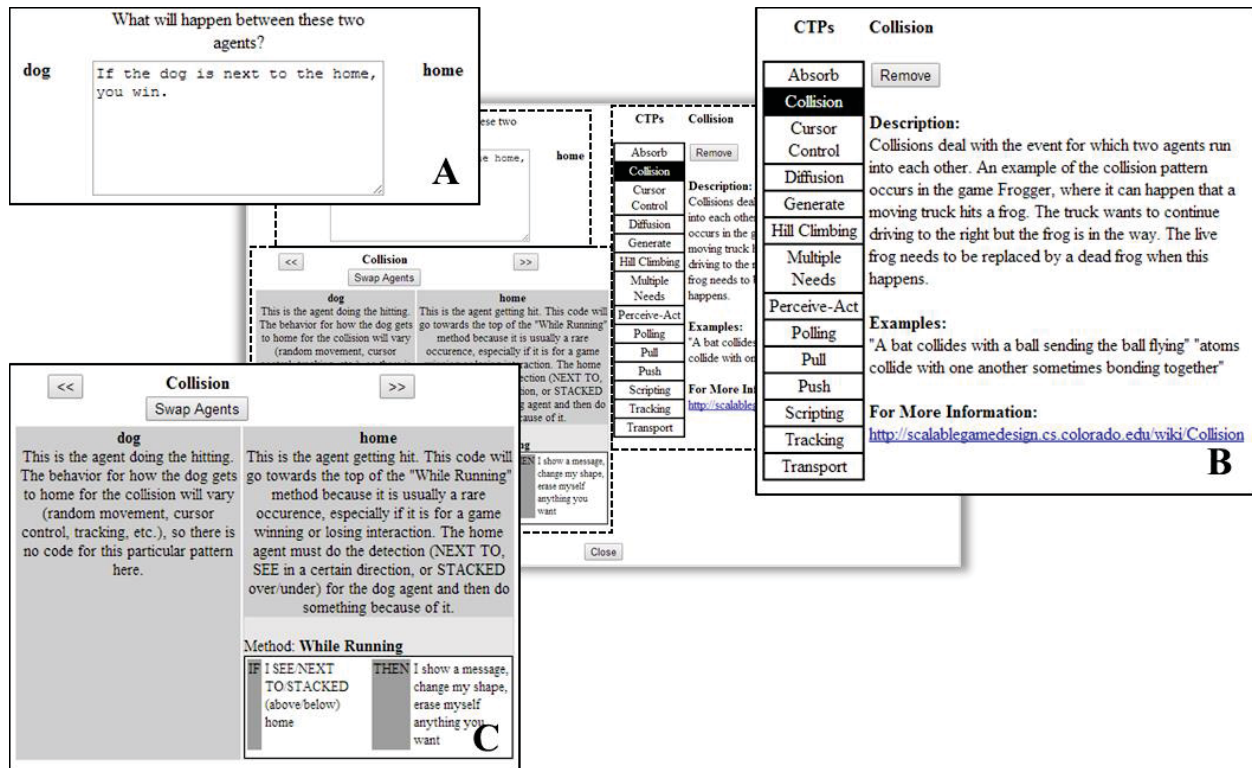


Figure 33. AgentDesign Planning Tool - Building Interactions Window

CTP Association

Using the interaction description as a guide, the CTPs that match the interaction can be identified. A list of CTPs is available in the build interactions window (Figure 33B). When a CTP is selected from the list, general information about the selected CTP is presented along with an option to add sample code to the design of both interacting agents through an “Add,” or “Remove,” button at the top. By connecting the interaction and CTP to both interacting agents, no work is duplicated as the student works through the design for each agent. For example, the collision CTP that was identified as needed when the “dog,” is next to “home,” is added to both the “dog,” and “home,” agent sections regardless of which agent is currently being designed.

CTP Sample Code

Figure 33C shows the sample code and everyday language descriptions for a CTP that has been associated to an interaction. In this case, the collision CTP was added from the list in Figure 33B. Once a CTP is added, an everyday language description of what occurs between the two agents is given (top of

the gray box), and sample code is shown (in white) when the mouse cursor hovers over the description. The descriptions and sample code are also shown in the respective agents' section (see Figure 31C).

The sample code shown can be modified from simple pseudocode to exact sample code using the arrow buttons at the top of the area (Figure 33C). The option to vary the sample code between easier to access pseudocode and exact sample code was meant to allow students to see the relationship between everyday language and academic programming language for that CTP.

A final feature of the CTP sample code area was the ability to swap which agents the descriptions and sample code are associated with. For the example shown in Figure 33C, if the "home," agent was doing the hitting, and not the "dog," agent, the description and code could be switched using the "Swap Agents," button located at the top of the area. This was a necessary feature due the application not being able to differentiate between which set of code should be associated to which agent by default.

Summary

The summary is accessed from the navigation menu on the far left. This section organizes all of the information that the student has entered and presents it on one page. In study 1, this organization was also supposed to have been present in the pencil-and-paper *planning document*. However, since the *planning document* was never completed in study 1, the information that the summary provides was never available for students. The students needed the organization scaffolded for them, and that is what the summary does.

At the top of the summary page is the overall description of the project. Next, a list of the agents is shown, along with their descriptions and any states that may have been identified. After the list of agents, the interactions and CTPs that were identified are grouped by interaction and provide the interaction description, agents involved, description of the CTP, and sample code. A student's summary

page is shown in Figure 34. It was intended that the students would print out the summary sheet and use it as a guide to make their game in AgentCubes Online.

Description	
There's a pet store, and all the pets have escaped. The main character has to run around and collect all the pets in 10 minutes.	
Agents	
Alisha:	
Alisha will run around to try to find the pets	
States;	
- default state	
pets:	
States;	
- default state	
Interactions & CTPs	
Interaction Description: Alisha will catch the pets.	
Transport	
Alisha This is the agent doing the transporting.	pets This is the agent being transported. It should not move off of <i>aiyana</i> , because that will stop the transporting from continuing.
Method: While Running	
IF pets is on top of me THEN Transport pets in a certain direction	
Interaction Description: User is going to move Aiyana around with the arrow keys.	
User Control	
Alisha This is the agent doing something (moving, jumping, etc.) when the user presses a keyboard key.	USER This is the user, who presses keys, but doesn't have any programmable behavior
Method: While running	
IF A key is pressed THEN I do something	

Figure 34. AgentDesign *Planning Tool* - Summary

The AgentDesign *planning tool* prompts students to go through a sequential design process. To complete the process the students must complete prior tasks, such as identifying interactions before being able to describe them and connect the interactions to CTPs. This sequential ordering of the process, as well as making all information centrally available, were central design choices meant to address the missed opportunities that students had in study 1 of not identifying interactions and connecting those interactions to CTPs.

This section discussed the AgentDesign *planning tool*, which juxtaposes students' everyday experiences and language with academic principles by relating the ideas of story to design and behavior to CTPs. This tool served the embodiment of my conjectures for study 2 (see Table 11 and Figure 28) and is consistent with my theoretical perspective discussed earlier.

Study Analysis

An analysis of the students' practices focused on the students' designs and resultant games. The amount of alignment between each student's ideas and designs, as well as between the designs and resultant games was examined. The degree of alignment was then compared to the results of study 1 to examine any changes in student use of design practices while using the AgentDesign *planning tool* as an intervention.

Data Sources

The two data sources used were the information entered into the AgentDesign *planning tool* by the student and the programming code and resultant games created using AgentCubes Online. The information students entered into AgentDesign provided the students ideas and their design of the game. Using AgentCubes Online, the students programming code and games could be analyzed and compared to the students' initial designs.

Student Designs

The AgentDesign *planning tool* was the primary space for students to design their games. As explained earlier, the designs that students would create were to be used to assist the students in creating their games using AgentCubes Online. Data collection from this source was primarily a printout of the summary section of each student's design. Recall that the summary section organized all of the information that the student had previously entered. A sample of a student's design summary that was used for analysis was shown earlier in Figure 34.

AgentCubes Online Programming Code and Games

Students created their games using a web-based programming environment called AgentCubes Online, which is explained in Appendix B. Each students' programming code and playable games was

accessed through the web-based interface available on AgentCubes Online. Analysis of the students' programming code and games was completed using this interface.

Analysis Methodology

The purpose of analyzing the student designs was to test the hypothesis that the new scaffolding tool, AgentDesign, assisted the students in enacting the design practices missed in study 1. Two types of analysis were carried out on the data. (1) The students' designs were analyzed for completion, if/when they enacted design practices that were skipped by groups in study 1, and the alignment of their CTP use with their descriptions of agent behavior. (2) The students' programming code and games were also compared to the designs to check for alignment between the two.

Student Designs

The analysis of the students' designs examined which design practices from Table 12 (reproduced here) were met by the student.

Reproduction of Table 12. AgentDesign Planning Tool Design Practices

Design Topic	Design Practice Details
1. Initial Idea	Recording the initial game idea.
2. Agent Names	Identifying and recording agent names from the project description.
3. Agent States	Naming different representations an agent will have.
4. Agent Behavior	Describing individual agent behavior.
5. Identifying Interactions	Identifying which agents will interact.
6. Interaction Description	Describing the interaction between two agents.
7. CTP Association	Associating CTPs to an interaction that has been described.

A simple coding scheme was used to rate the completeness of each design practice for the students' design summaries. The coding scheme is shown in Table 14. Please note that the number of sentences written for the *Initial Idea* was used to quantify the amount of writing each student did to initially describe her/his game.

Table 14. AgentDesign Design Practices Coding Scheme

Design Practice	Description of Codes
1. Initial Idea	# sentences - number of sentences used to describe the initial idea. non-descriptive - the initial idea was not descriptive.
2. Agent Names	yes - agents were identified. no - agents were not identified.
3. Agent States	default - no states were changed from default. some - less than half of the agents' states were changed from default. most - more than half of the agents' states were changed from default. all - all of the agents' states were changed from default.
4. Agent Behavior	none - none of the agents had descriptions of their behavior. some - less than half of the agents had descriptions of their behavior. most - more than half of the agents had descriptions of their behavior. all - all of the agents had descriptions of their behavior.
5. Identifying Interactions	yes - interactions were identified. no - interactions were not identified.
6. Interaction Description	none - none of the interactions were given descriptions. some - less than half of the interactions were given descriptions. most - more than half of the interactions were given descriptions. all - all of the interactions were given descriptions.
7. CTP Association	yes - CTPs were associated to interactions. no - CTPs were not associated to interactions.

After examining the completeness of each design, I refined my analysis by doing a count of the categories shown in Table 15 for each. This process was carried out on paper printouts of the design summaries taken from the AgentDesign *planning tool*. I decided whether or not a design was missing an interaction or CTP based solely on the students' project descriptions, descriptions of agent behavior, or descriptions of interactions. My primary interest for pursuing this refined analysis was to see how often the students were able to identify interactions between agents that they had described in their design, and then how often they successfully associated the identified interactions to the correct CTPs.

For these counting categories I focused on three overall groups of counts: agents, interactions, and CTPs. These three groups were chosen to be more closely examined because they required the students to interact with AgentDesign based on what they had written. For example, agents would need

to be identified after being described in the game description. Interactions would also need to be identified after being described in an agent’s behavior, and CTPs would need to be identified from behavior or interaction descriptions.

In Table 15, the term “correct,” (such as “interactions correct”) indicates an alignment between a student’s descriptions, written in their own words, and the interactions that were identified by that student using AgentDesign. Similarly, interactions or CTPs were counted as “needed,” if I believed a student did not identify an interaction or use a CTP that was clearly described in the game description or agent behavior.

Table 15. Counting Categories for Student Design Summaries

	Category	Description
AGENTS	# agents named in description	The number of possible agents (nouns) named in the description.
	# agents named outside description	The number of agents named outside of the description, such as in other agent’s behavior descriptions.
	# identified agents not named anywhere	The number of agents identified in AgentDesign, but NOT named in the description or elsewhere.
	# identified agents named anywhere	The number of agents identified from the description in AgentDesign.
	# non-identified agents	The number of agents named in the description or elsewhere, but not identified in AgentDesign.
INTERACTIONS	# interactions correct	The number of interactions correctly selected based on the description or agent behavior.
	# interactions needed	The number of interactions perceived by me as needed based on the description or agent behavior.
	# interactions unclear	The number of interactions that were selected but were not described in either the description or agent behavior.
CTPS	# CTPs correct	The number of CTPs correctly selected based on the description, agent behavior, or interaction descriptions. NOTE: In the event where the CTP used was “user control,” but the USER agent was not selected, the CTP selection was counted as correct if the agent being controlled was correctly identified. In these cases, the interaction would have been coded as “unclear.”
	# CTPs not correct	The number of CTPs incorrectly selected based on the description, agent behavior, or interaction descriptions.

# CTPs needed	The number of CTPs perceived by me as needed based on the description, agent behavior, or interaction descriptions.
# CTPs unclear	The number of CTPs selected but not described in the description, agent behavior, or interaction descriptions. For this case, it would be unclear as to why the student chose the CTP.

Below is a sample of the raw coding that was carried out using these counting categories for the interactions group. Notice that in the second row Amelia correctly identified six interactions in her design but also needed to identify three other interactions based on what she wrote in her game description or descriptions of behavior. She also did not identify any additional interactions that were unclear based on what she wrote.

Table 16. Sample of Raw Interaction Counting Categories of Students' Design Summaries

<i>Name</i>	<i># interactions correct</i>	<i># interactions needed</i>	<i># interactions unclear</i>
Alisha	2	0	0
Amelia	6	3	0
Carl**	0	unknown	0
Clint	5	0	0
Paul	2	1	1
Adam	5	2	0

The purpose of breaking down the identification of agents into five different counting categories was to explore the effectiveness of the identification functionality in helping the students identify all of their agents. The counts kept track of how many agents were named in the game description, how many were named in the agent behavior descriptions, and how many named agents were identified using AgentDesign. The count of agents named in the behavior descriptions was kept to examine how often the students did not go back and modify their description as their design was developed. A count was also kept of which agents were not identified at all, regardless of where they were named (description or agent behavior), and which agents were not named anywhere but were still identified.

Three different categories of counting were used for the interactions of each design. The intention of looking at these numbers was to understand how well the students carried out interaction

identification. I wanted to see how often correct interactions were identified, how often they missed identifying interactions and needed to add them, and how often they added interactions that were not previously described. All descriptions of the project and agent behavior were used to judge if any interactions were correct, needed, or unclear, which gave me an indication of how well AgentDesign was working. In some cases, the number of needed interactions was not something I could make a judgement about based on the lack of information elsewhere in the design.

The four categories used to organize CTP identification and use were intended to show how often the students correctly associated descriptions of agent behavior or interactions to CTPs, how often they incorrectly associated a CTP to an interaction, how often they did not associate a needed CTP to an interaction, and how often it was unclear to me why a CTP was chosen. I relied only on the information written by the student to make these judgements. A coded sample of a design summary is available in Appendix A.

Inter-rater Reliability of Correctness of Designs Coding

Inter-rater reliability was carried out on the correctness coding scheme for a sample of five student design summaries. Overall, the outside coder and I were mostly in agreement on the number agents named and identified using the tool. However, there were a few exceptions in interpreting whether some of the named objects by students should have been different depictions of the same agent, or separate agents. In regards students **correctly** identifying and using interactions and CTPs, we were in nearly full agreement for four of the five summaries. For the fifth summary, we found disagreements due to our different interpretations of grouping named objects or keeping them as separate agents. A major set of disagreements between my coding and that of the outside rater was on the **needed** interactions and CTPs. The outside rater believed that there were many more needed interactions and CTPs than I found. This was appeared to be due to the rater inferring what the student may have intended, and not coding only from the students' written descriptions.

Transfer of Designs to Games

For each game, four categories were used to organize what was done in the games compared to the designs. The first category was a comparison of the agents that were named in the design and the agents that were created in AgentCubes Online, which gave an indication of how well the students followed their initial design. Next, the number of agents that were created, but not named in the design, was counted as a second category. The third category examined the alignment of CTP usage between the design and the resultant game. The final category listed any of the CTPs that the student used, regardless of whether or not that CTP was included in the design.

Findings

Student Designs

The intervention for study 2, AgentDesign, helped the students get further through the design process than when they used the pencil-and-paper *planning document* in the first study (see Figure 35). From study 1 I found that the students reached the point of discussing an interaction only once, and never got to the point of associating interactions to CTPs. With the introduction of the AgentDesign *planning tool*, many students identified interactions and then associated those interactions to CTPs. Of the 18 students that used AgentDesign to design a game, 11 of them reached the point of associating CTPs to interactions.

	Recording Game Idea	Recording Agents	Recording Agent States	Recording Agent Behavior	Identifying Interactions	Describing Interactions	Associating CTPs
Study 1 (Groups)	4/4	4/4	4/4	3/4	0/4	0/4	0/4
Study 2 (Students)	16/18	17/18	2/18	14/18	12/18	8/18	11/18

Figure 35. Comparison of Design Practices for Studies 1 & 2

AgentDesign supported the students through the design process better than the pencil-and-paper *planning document*. From the figure above, none of the groups in study 1 identified interactions,

described the interactions, or associated those interactions to CTPs. In the second study, many of the students were able to accomplish these design practices and create more complete designs.

Table 17 presents which design practices from Table 12 were completed within the AgentDesign *planning tool* for all students that designed a game in the second study. I should also mention that Earl created two different designs, and so he is in this table twice.

Table 17. Student Design Practices using AgentDesign

Name	Initial Idea	Agent Names	Agent States	Agent Behavior	Identified Interactions	Interaction Descriptions	CTP Association
Alisha	2 sentences	yes	defaults	some	yes	all	yes
Alan	3 sentences	yes	defaults	all	yes	all	yes
Amelia	6 sentences	yes	defaults	most	yes	all	yes
Arthur	4 sentences	yes	defaults	all	yes	none	yes
William	non-descriptive	yes	defaults	all	yes	none	yes
Gary	4 sentences	yes	defaults	all	yes	none	yes
Carl	9 sentences	yes	defaults	none	no	none	no
Clint	6 sentences	yes	defaults	all	yes	all	yes
Earl	4 sentences	yes	defaults	all	yes	none	no
Earl (2)	3 sentences	yes	defaults	some	no	none	no
Gill	3 sentences	yes	defaults	all	no	none	no
Ken	non-descriptive	yes	all	all	yes	some	yes
Mary	4 sentences	yes	defaults	none	no	none	no
Paul	6 sentences	yes	most	most	yes	all	yes
Adam	6 sentences	yes	defaults	most	yes	all	yes
Sonya	3 sentences	yes	defaults	none	no	none	no
Sally	4 sentences	yes	defaults	all	no	none	no
Sarah	1 sentence	no	defaults	none	no	none	no
Jorge	5 sentences	yes	defaults	some	yes	all	yes

Looking closer at how robust the students' designs were I used a counting coding scheme (Table 15) to examine how correct each student's design was, and what they may have missed or used incorrectly, based on the descriptions they wrote in AgentDesign. A table of these overall counts is shown below.

Table 18. Correctness of all Students' Designs using AgentDesign

Category	Count
# agents named in description	92
# agents named outside description	5
# identified agents not named anywhere	24
# identified agents named anywhere	71
# non-identified agents	26
# interactions correct	30
# interactions needed	34**
# interactions unclear	15
# CTPs correct	25**
# CTPs not correct	0**
# CTPs needed	35**
# CTPs unclear	18**

** Value may be larger due to the lack of information provided in the students' descriptions.

Identifying Agents

In both study 1 and 2, the students were able to identify and record most of the agents that they wanted in their games during the design process. Looking at the data for study 2 I found that the students identified most of the agents (71 of 92) they had named in their project descriptions, descriptions of agent behavior, or descriptions of interactions. Some reasons for the missing identifications may have been that the students made a mistake and did not identify them or they chose to not identify those agents since they only played a story role or were not important for the game functionality. Figure 36 shows the breakdown of the number of times that students identified agents that were named (71), didn't identify named agents (26), or identified agents that weren't named in any description (24). I should note that 16 of the 24 agents that were identified, but not named, were from one student's design in which he wrote very little in his game description or in the agent behavior descriptions.

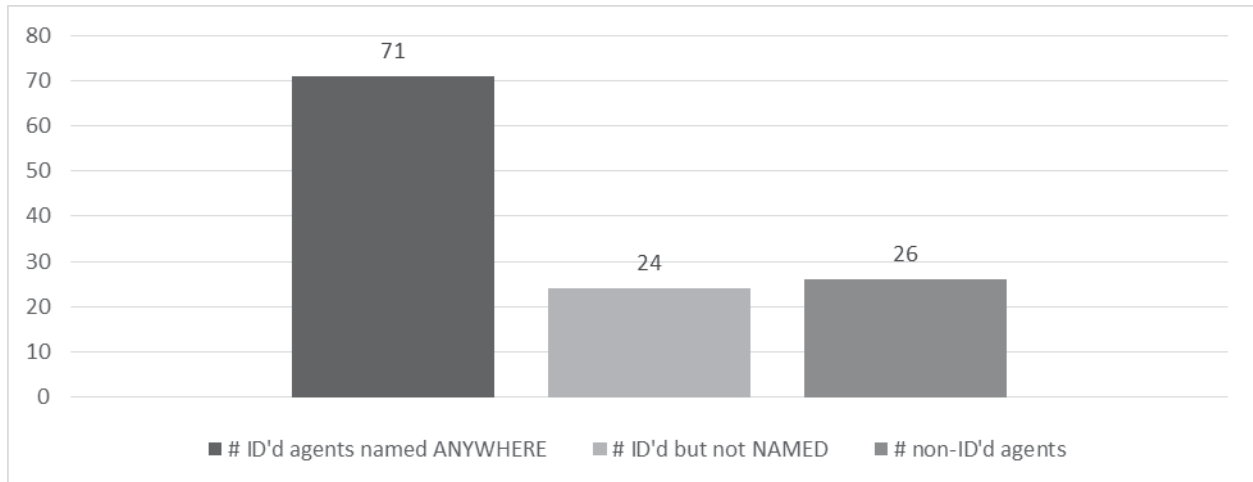


Figure 36. Counts of Identified and Non-Identified Agents in AgentDesign Summaries for All Students

Figure 36 provides a representation of the number of identified agents that were named in AgentDesign, identified agents that were not named, and agents that weren't identified using AgentDesign's functionality but were named somewhere for all students. This figure shows that when students named an agent in their game description or descriptions of agent behavior, most of the students *used the tool* to identify those agents so that the design could be further refined. In study 1, students were also able to identify agents, but the explicit link between the written descriptions and identifying agents was not present in the pencil-and-paper *planning document*.

Counts for individual students are shown in Figure 37 and Table 19. It is important to note that the number of identified agents that were not mentioned in the students' designs was 24, but that number is skewed by data from one student. William identified 16 agents without naming them anywhere in his design, so for the other 17 students there were only 8 instances of an agent being identified in AgentDesign that was not named somewhere in the descriptions.

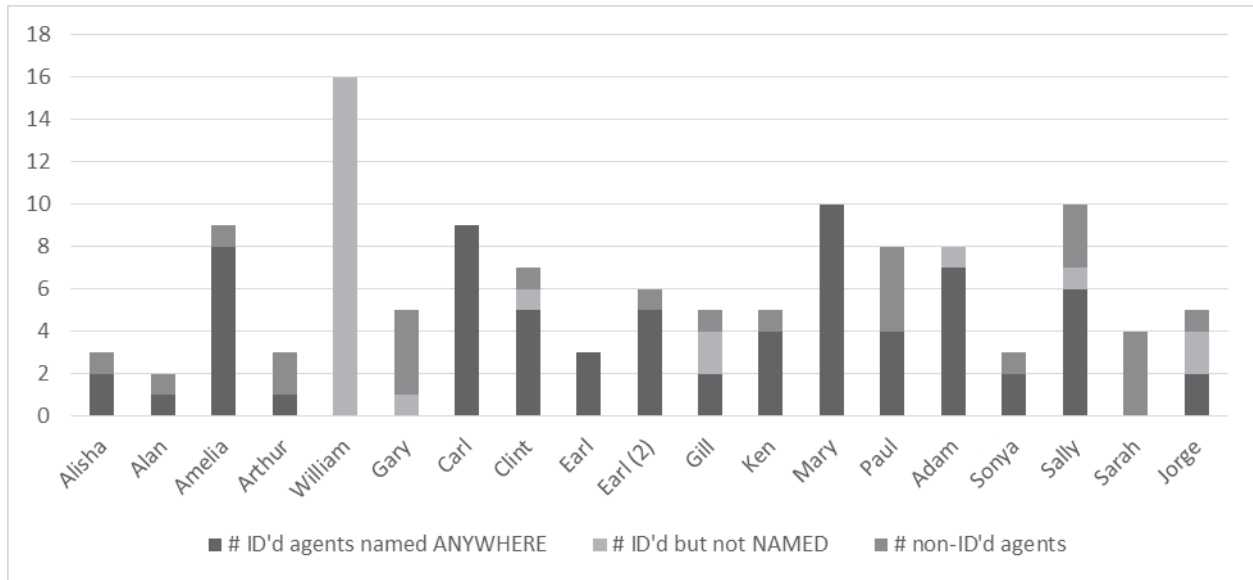


Figure 37. Students' Identification of Agents using AgentDesign

Table 19 provides the same information as Figure 37, only in table form. Also shown in the figure and table is that most of the students were able to identify at least some of the agents that were described. Many of the students identified most of the agents they named in descriptions, and only three students (William, Gary, and Sarah) did not identify any agents that were named in their designs.

Table 19. Individual Counts of the Students Identification of Agents using AgentDesign

Name	# ID'd agents named ANYWHERE	# ID'd but not NAMED	# non-ID'd agents
Alisha	2	0	1
Alan	1	0	1
Amelia	8	0	1
Arthur	1	0	2
William	0	16	0
Gary	0	1	4
Carl	9	0	0
Clint	5	1	1
Earl	3	0	0
Earl (2)	5	0	1
Gill	2	2	1
Ken	4	0	1
Mary	10	0	0

Paul	4	0	4
Adam	7	1	0
Sonya	2	0	1
Sally	6	1	3
Sarah	0	0	4
Jorge	2	2	1
Totals	71	24	26

Interactions

As shown in Figure 35, in study 1 only one group discussed interactions between agents, and no groups recorded interactions between agents. In contrast, 12 of the 18 students using AgentDesign in study 2 identified interactions, and 8 of those 12 described the interactions using the tool. Looking at the individual students' designs from AgentDesign showed that thirty of the interactions identified by the students were correctly aligned with descriptions that the students wrote using AgentDesign. However, from my analysis I also found that at least 34 interactions were needed, based on what the students wrote in the project descriptions or descriptions of agent behavior, but were not identified. I would also like to point out that the number of needed interactions may be higher, as indicated by the (**) in Table 18 since I could not determine if the students' descriptions were just incomplete or they made an incorrect interaction identification.

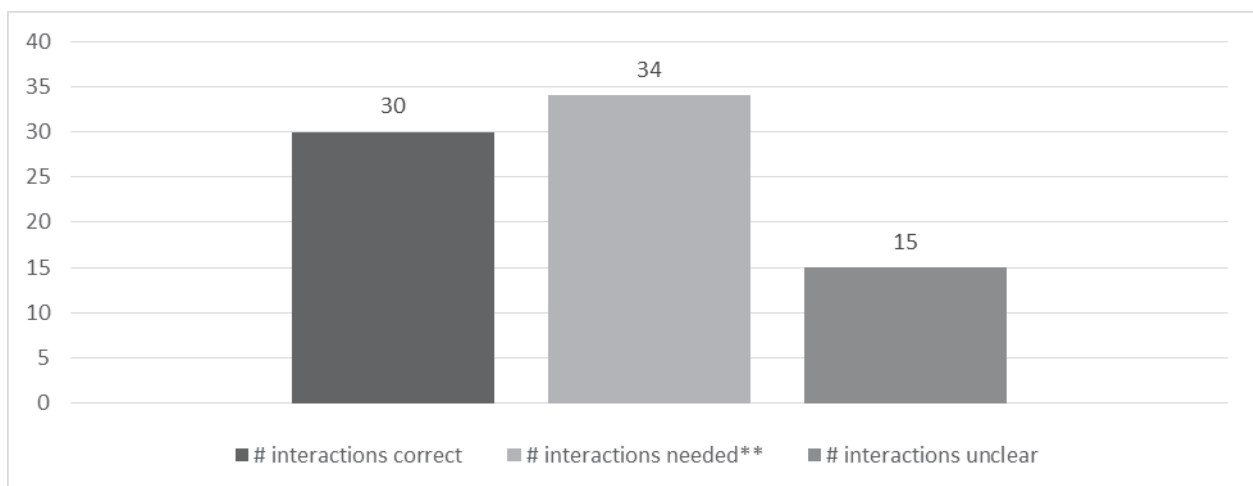


Figure 38. Counts of Alignment between Interactions and Descriptions in AgentDesign Summaries

Figure 38 provides a representation of the number of correctly identified interactions, needed interactions, and unclear identified interactions based on the students' descriptions. The figure shows that 30 interactions were explicitly identified using AgentDesign among 18 students, compared to 0 identified interactions among the 4 groups (13 students) in study 1.

As just mentioned, some of the designs did not have enough information for me to clearly assess what interactions were needed, so I could not provide a valid number for those designs. The students that did not have alignment between their descriptions and identified interactions are indicated by the (**) in Figure 39 and Table 20. On the other end of the spectrum, there were also 3 outlying students (Alisha, Clint, & Ken) who correctly identified all of the necessary interactions for their design and I could find no reason that they would need other interactions, however Amelia and Ken did select other interactions that were not described anywhere in their designs. The figure shows that a subset of students were successful in correctly identifying some interactions, indicated by the arrows, and most students identified interactions, although not all of them were correct.

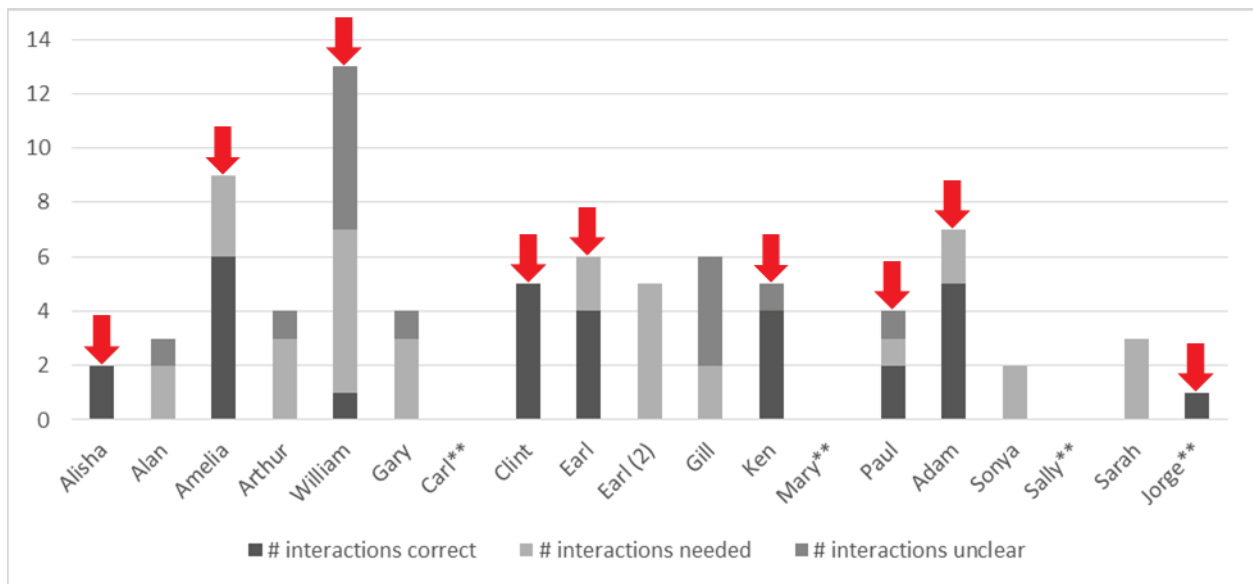


Figure 39. Students' Interaction Correctness using AgentDesign

Table 20 provides the same information as Figure 39, only in table form.

Table 20. Individual Counts of each Student's Correct Identification of Interactions using AgentDesign

Name	# interactions correct	# interactions needed	# interactions unclear
Alisha	2	0	0
Alan	0	2	1
Amelia	6	3	0
Arthur	0	3	1
William	1	6	6
Gary	0	3	1
Carl**	0	unknown	0
Clint	5	0	0
Earl	4	2	0
Earl (2)	0	5	0
Gill	0	2	4
Ken	4	0	1
Mary**	0	unknown	0
Paul	2	1	1
Adam	5	2	0
Sonya	0	2	0
Sally**	0	unknown	0
Sarah	0	3	0
Jorge**	1	unknown	0
Totals	30	34	15

CTPs

A goal of the Game activity was for the students to learn about CTPs, and in study 1 the students never reached a point during the design process where they accessed the CTP information. Using AgentDesign, 11 of the 18 students did access the CTP information and incorporated them into their designs. Table 18 and the figure below show that 25 CTPs that the students used in their designs aligned with interactions that the students had previously identified and described. However, I also found that there were many CTPs (35) that the students did not associate to interactions, but would need for a complete design.

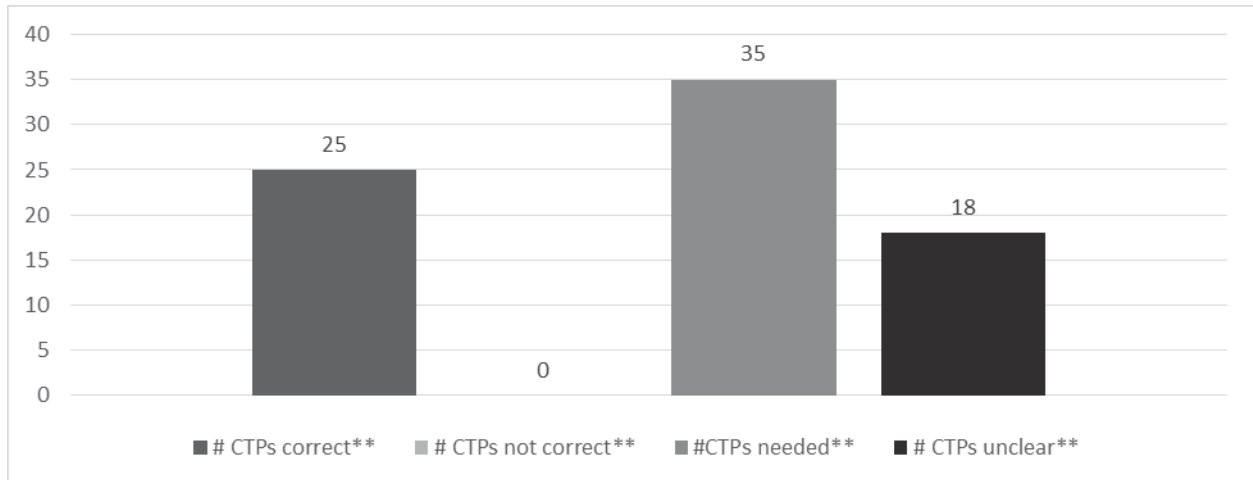


Figure 40. Counts of CTP Alignment with Descriptions in AgentDesign Summaries

Figure 40 provides the counts across all students of the correctly used CTPs, incorrectly used CTPs, needed CTPs based on student descriptions, and the CTPs that were unclear to me why they were used by the students. Similar to what was discussed for the use of the tool to identify interactions, the students correctly associated 25 CTPs to interactions for their designs, as opposed to 0 CTPs being correctly used in study 1.

Individual student use of CTPs is shown in Figure 41. Most of the students were able to correctly incorporate CTPs into their designs using AgentDesign, however, all but one student needed additional CTPs based on what they wrote in their descriptions. For some of the designs it was unclear as to what CTPs would be needed, so I was unable to make a judgement on that value for certain students. These students are indicated by the (**) in Figure 41. Also, two students, Gill and Paul, used CTPs in their designs that I could not understand why they were chosen. These were the only two students that had unclear CTP usage.

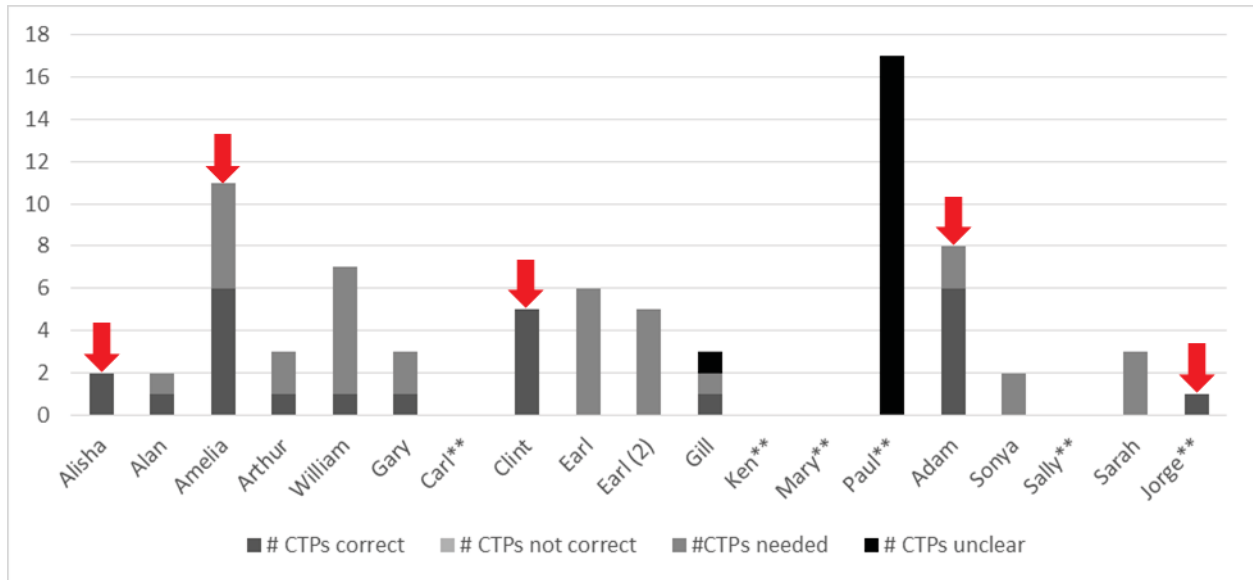


Figure 41. Students' CTP Alignment with Interactions and Descriptions using AgentDesign

Table 21 provides the same information as Figure 41, only in table form. The figure shows that, of the students that identified interactions, most of them correctly associated at least one CTP to an interaction, and several students correctly used CTPs (indicated by the arrows).

Table 21. Individual Counts of the Students Correct Use of CTPs using AgentDesign

Name	# CTPs correct	# CTPs not correct	#CTPs needed	# CTPs unclear
Alisha	2	0	0	0
Alan	1	0	1	0
Amelia	6	0	5	0
Arthur	1	0	2	0
William	1	0	6	0
Gary	1	0	2	0
Carl	0	0	unknown	0
Clint	5	0	0	0
Earl	0	0	6	0
Earl (2)	0	0	5	0
Gill	1	0	1	1
Ken	unknown	unknown	unknown	unknown
Mary	0	0	unknown	0
Paul	unknown	unknown	unknown	lots
Adam	6	0	2	0
Sonya	0	0	2	0
Sally	0	0	unknown	0
Sarah	0	0	3	0

Jorge	1	0	unknown	0
Totals	25	0	35	1

Transfer of Designs to Games

Table 22 shows that not all of the students that designed a game went on to create it. And those students that did choose to create a game minimally used CTPs, even if they identified the CTPs using AgentDesign and had access to sample code. The only CTP used by students was “User Control,” which would have made an agent do something when a key is pressed. In many cases, the agents created in the game did not correspond to the list of agents from the design. Only one student, Amelia, created all of her identified agents from her design in the programming environment. Amelia was also one of the few students to not create new agents that were not discussed in the design. Most other students created other agents that were not the agents they had already identified using AgentDesign.

Table 22. Transfer of the Students’ Design to the Programming Environment

Name	Game Name	Game?	Agent Match?	Other Agents?	CTPs Match?	CTPs used
Alisha	Pet shop	YES	0/2	6	1/2	user control
Alan	flappy bird	YES	1/1	0	0/1	none
Amelia	the crown rescue	YES	8/8	0	1/6	user control
Arthur		NO				
William	looney people	YES	2/16	0	0/1	none
Gary		NO				
Carl	brincobowling	YES	4/9	5	0/0	user control
Clint	underwaterbadguys	YES	0/5	8	1/5	user control
Earl	Mine Crab	YES	0/3	4	0/0	none
Earl (2)		NO				
Gill		NO				
Ken	awsomness	YES	3/4	4	0/8	none
Mary		NO				
Paul	you can run but you can't hide	YES	3/4	5	1/17	user control
Adam	zombie spider	YES	1/8	1	0/6	none
Sonya		NO				
Sally	girlcraft	YES	4/7	4	0/0	none
Sarah		NO				
Jorge	hrebrian123	YES	1/4	0	0/0	none

CTP Use and Conflicting Concepts/Principles

Another interesting finding was a phenomenon where students' use of CTPs were reasonable in an everyday language sense, but not in a programming sense. The students that exhibited this phenomenon seemed to choose CTPs based on how the CTP name fit the interaction descriptions and did not consider if those CTPs were the best programmatic fit for their design. This is discussed further in this section.

Within the data, there were 4 students that did not select the CTPs that best fit their descriptions of agent behavior or interactions. However, I was not comfortable saying that their choices were incorrect, although they would not have worked out well for the students when programming. In the following sections 4 cases are discussed in which students did not select the "most correct," CTP and appeared to be guided by their own understandings of how the interactions they described would occur in the real-world as well as their own understanding of the CTP terms.

Alisha

Alisha wanted to make a game called "Pet Store," where the player had to go around and catch escaped pets. Her game description is below.

"There's a pet store, and all the pets have escaped. The main character has to run around and collect all the pets in 10 minutes."

The interaction that stands out as confusing a real-world action with the CTP term is how the main character carries out the act of catching the pets. She described this interaction as "[main character] will catch the pets," and the CTP she chose was the transport CTP. Transport was defined in AgentDesign as, "the transport pattern occurs when one agent (AGENT-X) carries another agent (AGENT-Y). Transport will also incorporate a move, so it will move and carry whatever is above it on the worksheet." The more correct CTP she should have used was absorb, which would have made each pet disappear as her main character got next to one. Using the transport CTP, her game would have had all

of the caught pets being stacked on top of her main character agent and moving with that agent. Her choice just doesn't make sense from a programming perspective. However, from a real-world perspective, when a person catches an animal from the pet store of course they are going to carry it back, or *transport* it.

Alisha's use of the transport CTP was not correct in my view. But I also don't think it was wrong. Her own experiences and understanding of the act of catching pets and the term transport likely influenced her choice and probably made complete sense to her.

Amelia

Amelia's game focused on a princess needing to get to her crown. The description of her game, as she wrote it, is below.

"princess she is looking for her crown. As she is looking for her crown she is collecting flowers andbracelets. you win when you find the crown.You lose when the witch find you.If you find the fairy she will help you get to the crown.she need to pass through the dark forest to get to her crown in the tallest tower."

For her game, Amelia had written an interaction description between her princess and witch agents as, "they fight." From that description it is unclear what exactly she wanted to happen in the game and what CTP would have been appropriate. The CTP that she chose to associate with that interaction was the push CTP. The push CTP is generally used when one agent needs to move a different, stationary, agent and was defined in AgentDesign as, "this CTP occurs when an agent wants to be able to move around certain other agents through the act of pushing them." Because the push CTP did not seem to fit, I initially marked this association as incorrect, but upon going through the data again I realized that "pushing," is a typical act of fighting. It appeared as though Amelia was using her understanding of, and experiences with, the concept of fighting to influence her decision of which CTP to use. Pushing does occur in fighting, and from the list of CTPs, it would make the most sense.

I do not think the push CTP would have worked for Amelia for her game. The more correct CTP would have been collision, but with a name (push) that fits a real-world activity like fighting I do understand why she would have chosen it.

Ken

Ken's game was an outlier because all he did was name his four main agents in the description area. So it was unclear at first what he wanted his game to be. His game description is shown below.

"savior,civilian,weapons,giant monsters."

While Ken didn't provide much information in his game description for me to use for an analysis of correctness, it was clear that Ken wanted the savior to use the weapons to save the civilians from the monsters. The interaction, and corresponding CTP use, that was interesting was between the savior and the weapons. Using AgentDesign, he chose three CTPs to be associated with this interaction that would conflict if they were all programmed into a game, and yet none of them are necessarily wrong. The three CTPs he chose were transport, absorb, and push. In the real-world sense fighting a monster with a weapon would mean that the weapon would be transported, because it would have to be carried. In another sense, I could also think of the weapon as being pushed as I moved it around. And finally, in a game sense, I could pick up the weapon and it would disappear, this is a common phenomenon in most games. For these three CTPs, only one could work at a time within a game. An agent can't be transported and be pushed and be absorbed, and the same is true for the other two CTPs. My hunch is that Ken chose all of the CTPs that worked for his explanation without taking into account the context (game, and not real-life), or how the three would operate together. His understanding of the terms transport, absorb, and push, and the associated activity with the three, were in conflict with how they are defined as CTPs and function within a game.

Paul

Paul's game focused on a main character, Mark, teaming up with a group called the Blackeyes to fight off assassins. His game description is below.

“Everything is quite in the city, New York City, but then an army attacks. everyone starts running. So the army tries to attack the people, but the people try to fight for their lives. There are other people in the city, the Blackeyes, minding their own business, but then they join Mark and fight the Assassins. At least 2 people need to survive on one side in order to win. To lose, the whole team dies.”

Looking at his design there were many interesting uses of CTPs. Like Ken, Paul seemed to associate interactions between agents to all of the CTPs that would work in both a real-world and game context. The interesting CTPs that he chose to associate with the interaction between his primary agent, Mark, and the agents chasing Mark, the assassins, included push, tracking, pull, and perceive-act. He described this interaction as, “mark will run away from the Assassins. Mark will attack the Assassins at the same time too.” Like Amelia, I suspect that he used the push CTP to represent fighting, especially since he did not use the collision CTP for this interaction. The tracking and pull CTPs could represent the assassins chasing mark, where tracking is a real-world concept, and pull is more of a game concept. The most surprising CTP used was the perceive-act CTP, which was defined in AgentDesign as, “this CTP uses information that an agent knows about itself and the agents around it to make a decision on what it should do. It basically looks at itself or around it, and depending on the state (depiction) or values that it finds, the agent will change itself or stay the same.” The term “perceive-act,” fit his story, but the CTP would not have been as useful as other CTPs to create functionality of the game he wanted.

Like the other students, Paul chose CTPs that would have had a conflicted meanings from what he knew of those terms and how they were defined in AgentDesign. I can completely understand his use of push for fighting, and perceive-act for how his agent would behave while being chased. Unfortunately, this conflict made his design more complicated and less useful than it could have been.

From the data there was an observable improvement of students enacting the intended design practices when using the AgentDesign *planning tool*, particularly for identifying interactions and using CTPs. In study 1, the students did not record any interactions and never accessed CTPs. In the follow-up

study, the students were able to successfully identify interactions, describe those interactions, and then correctly associate those interactions to CTPs. In most of the cases, the students did not correctly incorporate all of the interactions or CTPs that they would have needed (based on the descriptions that each student wrote), but given the results of study 1 any use of CTPs is a step forward. Possible improvements for the Game activity process and AgentDesign *planning tool* are discussed in the following section.

Study 2 Closing

There is still a lot of work that could be done in both exploring how best to expose students to design, but also how to get elementary students involved in computer science. The findings from study 2 show that students still had difficulty creating the games that they initially wanted to make. However, unlike study 1, the students in study 2 were able to access the CS principles and enact design practices that were skipped over in study 1. The following discussion section comments on the successes and issues that students had and how I may move forward with this work and improve AgentDesign or the activity.

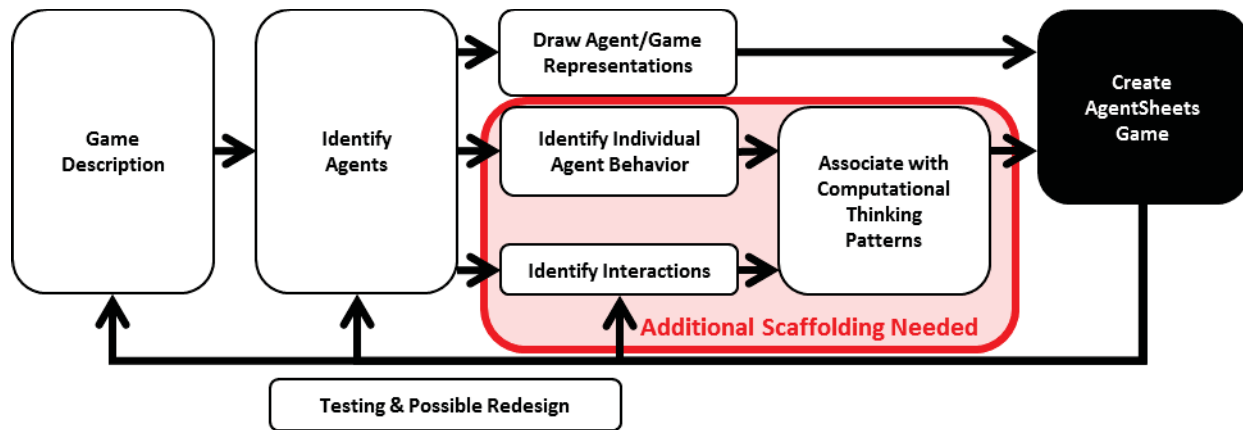
Discussion

There are four points of discussion that I would like to pursue in this section.

1. AgentDesign supported the students through the design process better than the pencil-and-paper *planning document* used in study 1.
2. For both studies, neither approach fully supported the students' transition from designing a game to creating it in a programming environment.
3. The scaffolding tool I have developed so far can be improved upon for a setting such as EPM, but also for everyday classroom use.
4. Some of the issues I encountered stemmed from the informal nature of the learning environment, so it is likely that what I was hoping to accomplish simply isn't possible without making the environment more formal.
5. More research needs to be done in CS education to develop useful approaches to teaching elementary students the discipline, as well as preparing elementary teachers to take part in the process.

Transfer from Design to Programming Environment

Unfortunately, neither study observed students successfully create their designs in a programming environment. The students in study 1 needed a lot of support from me to create their games after going through the design process, primarily due the fact that they did not complete the design process. The reproduction of Figure 21, shown here, emphasizes the focus of the intervention of study 2 being on supporting students in identifying agent behavior, interactions and CTPs.



Reproduction of Figure 21. Areas needing improved scaffolding for the Game activity

In study 2 the students didn't make the games they designed, if they made a game at all. Figure 42 highlights the area of the game activity that will be the focus of future work.

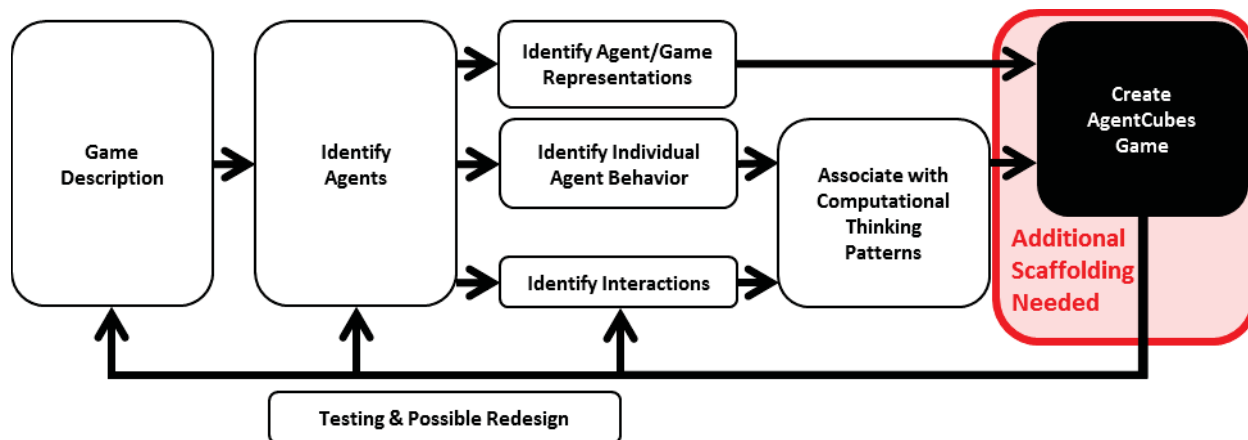


Figure 42. Areas needing improved scaffolding after study 2

In the following sections I provide some explanations as to why I think students were not successful in creating their games, or even starting to make a game to begin with.

One possibility that could have made moving from the design to creating a game problematic for students was in the support mediating the transfer. The design summary may not have been structured in a way that was helpful to the students to move from the *AgentDesign planning tool* to the programming environment (AgentCubes Online). There could have been too much, or even too little, information available to the students when they moved to the programming environment. Another possibility is that the programming environment, AgentCubes Online, was not as easy to access for

elementary students as I had expected. The short video I had provided on iRemix as an introduction may not have been enough, and the students may not have been comfortable asking for help when needed, or even understood that using AgentCubes Online was the next step.

Another explanation is that the students got bored with the activity or were distracted by the games available on the Scalable Game Design Arcade or elsewhere at EPM. I was positioned next to the Minecraft station, and many of the students that participated in the Game activity went back and forth between the two stations. Similarly, many students got distracted playing games in the Scalable Game Design Arcade, and so never moved on to creating their own games.

A final explanation is that the informal learning setting established at EPM, especially during study 2, was simply too free for the students to successfully complete the Game activity. Informal learning settings can be great in many ways, for example, I was able to leverage the students' ideas to have them design and create their own games. However, by their nature, informal learning settings are difficult to implement certain types of structured support. For study 2, students were not only learning about design, and how to work with the AgentDesign *planning tool*, but were also learning how to program. It is apparent that some type of support was missing, but it is worth mentioning that the needed support may not have been feasibly implemented in EPM at that time.

Unlike study 1, the students in study 2 did not seem as invested in making their games work and just stopped at a certain point. Some stopped after finishing the design, some after making some agents, and some after getting an agent to move. In a follow-up study I will need to more closely examine the reasons for students abandoning the game-making process.

Implications for CS and Informal Learning

The benefits that were being afforded through the informal learning setup of EPM also had a cost in that there was simply no easy way to step students through an introduction to AgentCubes Online without making the activity much more like "school," and therefore, formal. Would students

have been more successful in making a game, and still felt ownership and pride, if someone made them finish it? In study 1, the undergraduates were more likely to play the role of someone making the students finish, and they did finish many games in that study. But through the undergraduates playing that role a hierarchy was created that was counter to the setup that EPM intended. I do think that the students needed more support, but they also needed a type of structure that does not fit an informal learning environment.

A purpose of this study was to see if elementary students could design and make a game on their own using AgentDesign and AgentCubes, and I am not as sure as I used to be about that possibility. There were just too many new things for the students to learn, and too many institutionalized distractions in the learning environment, such as moving from station to station on a whim, for them to really be successful. And for the students to learn these new things, like working with AgentCubes, the activity would likely need to contain fewer distractions, be more structured, and provide limitations & scaffolding necessary to enhance the students' success. It appears that in attempts to empower the students, the informal environment of EPM, as far as CS learning is concerned, does the opposite. Framing this shortfall in terms of Vygotsky's zone of proximal development (ZPD), the activity certainly met the students at a place that they could participate, by using their own ideas to design a game, but the informal structure didn't encourage the students to reach the upper limit of the ZPD even though there were many supports in place (CS experts, videos, etc.). The amount of support that the students needed didn't fit an informal learning environment paradigm. In order to support the process and expectation that students create a game in AgentCubes online, I feel that much more structure needed to have been in place. This structure would have removed aspects of the informal learning environment that EPM sought to establish during study 2.

These two studies demonstrate the potential value of structure toward student empowerment and expectations and the trade-offs that occur for learning environments that may be more or less

formal. The informal nature of EPM encouraged the students to play, and have a low entry point for accessing the activity, but it also allowed for free movement to another activity if making a game got too hard or was not interesting anymore, or simply if another activity looked more fun (like playing minecraft). Making the activity more structured in future studies may take away from the informal nature of the current environment at EPM, but they may be necessary for supporting elementary students learning of CS, design, and game structure.

Implications for the “Make Your Own Game” Activity

As I mentioned earlier, the students did not successfully create their games from the designs that they developed in study 2. A possible solution to this issue would be to expedite or simplify the process of the “Make Your Own Game” activity. Essentially, the students need to get to the fun stuff and create their final product quicker. Some options to accomplish this goal include:

- shortening the design process, or reduce the number of steps,
- making the creation/programming process simpler and/or shorter,
- building in more opportunities to see the relationship between the design and programming.

Limiting the number agents allowed within the design will force students to create smaller games. This, in turn, will get the students through the design process quicker, as well as make the programming process less complex. While this option does shorten the design time by adjusting the activity, another possibility is to examine how AgentDesign may be modified to make the design process more compact. Currently, students start from a project description, name agents, describe agent behavior, identify interactions, describe interactions, and then link interactions to CTPs. It could be possible to combine some of these steps within the web tool, and that may alter how much time students need to complete the design process and start creating their game. A third option is to only allow students to design for one aspect of a game. Essentially, the students’ would “modify”, as in “use-

modify-create” (Lee et al., 2011), a part of a previously developed game design and then create the entire game using information already entered in the design.

Outside of modifying the design aspect of the activity, there are other possibilities to make the design to programming process simpler or more streamlined. One possibility is to have AgentDesign generate a project file for students to download and use as a template within the programming environment. For example, AgentDesign could create a programming project file that would include all of the agents created within the design (but with only a default representation) as well each agents’ behavior descriptions written in the comment sections. Providing students this opportunity would allow them to immediately start programming each agent’s behavior. This option also would allow the students to see the connection between components of their design and the game they would be creating.

A final option is to address the disconnectedness between the design and programming environment. In the current activity, there is a physical disconnect between the design and the programming environment because the design either needs to be printed out or be on a separate window. Directly inserting components of the students’ designs into a project file is one option for improving the understanding of the relationship between the design process and the creation process. However, another option is to set up AgentDesign in a way that AgentCubes Online could be shown as a panel within the web tool. This way, students would be able to go through step-by-step and see their design and resultant code side-by-side. Allowing students to observe the relationship between the design and the programming code will not only help build understanding of CTPs and design, but will also make the transition between the two much easier.

There are many options for expediting and simplifying the transition from students designing to creating their games in a programming environment. All of these will be taken into account in future iterations of the study.

Implications for Programming Environments

The two studies carried out for this dissertation utilized two different programming environments, which students used to create their games. These were AgentSheets, a 2D environment, and AgentCubes Online, a 3D environment. In regards to the students' ability to work within a 2D or 3D environment, I don't think there were any issues or advantages. In some sense the students may have felt that the 3D environment was cooler, but they were also very creative when working within the 2D environment in study 1.

The differences in students' abilities to work with either programming environment stemmed from the interfaces of each environment, and not so much the dimensional affordances or restrictions. For example, AgentSheets had an interface where almost all of the windows of the programming environment could be moved around or minimized. In contrast, AgentCubes Online had almost everything visible at all times. With windows and important components of the environment able to be minimized, it was very easy for students to lose windows and get lost using AgentSheets. However, the interface in AgentCubes Online always showing most aspects of the programming environment created a very full screen with lots of options. This could have been overwhelming to the students. It will be important to consider how well students can use the interface, and not just understanding the programming or dimensional affordances, for future iterations of this study.

Implications for the Design Scaffolding Tool

In study 1, the students never accessed the CS principles of interactions and CTPs and did not complete the design process. In the follow-up study, which used a revised scaffolding tool called AgentDesign, many of students were able to access CS principles and incorporate interactions and CTPs into their designs.

In many ways AgentDesign appears to have assisted the students in completing the design process, but not the game-making process. This section discusses aspects of AgentDesign that went well for the second study, ways it could be improved, and how it may be used in a classroom setting.

Successes

A success of the AgentDesign *planning tool* was that the students' design practices were more closely aligned with the intentions of the activity than the practices they exhibited when using the pencil-and-paper *planning document*. Students in study 1 were engaged in the activity, but almost never reached a point where they would identify interactions during the design process, and none of the groups in study 1 made a connection between agent behavior and CTPs. In study 2, many of the students identified interactions and also connected their own descriptions of agent behavior and interactions to CTPs. Connecting the student ideas to CTPs was an important step for the students to reach because it assists student development of CS principles. Two influencing factors for why the students got further in the design process may have been that the guided structure that AgentDesign provided was designed to lead students completely through the process and that the students were not encouraged to do any drawing within the tool.

From study 1 I learned that assisting students through this process would require that the tool be highly structured, but also that any new knowledge should be relatable to the students' original ideas. Guided by theory on conceptual development and formative assessment I developed a tool that relied on student ideas and experiences to contextualize CTPs and video game structure. Utilizing prompts between refinement stages is likely what got so many more students through the design phase of the activity in study 2 than in study 1.

Also from study 1 I learned that having a primary activity of the game design process be focused on drawing can distract the students from the overall goal. In study 1 the students were encouraged to draw out their agents and game boards within the pencil-and-paper *planning document*, and it was observed that these students spent a majority of their time focusing on the look of their games and not on what their agents would do in the game. By removing drawing as a requirement of the scaffolding

tool for study 2, the students actually reached the point of, and carried out, identifying interactions and linking those interactions to CTPs in the scaffolding tool.

Missed Opportunities

A continual missed opportunity that was present in both studies was providing clear, age appropriate language for the students. In study 1 the language was too open and did not provide enough clear direction for the students. In study 2, the language used in AgentDesign was likely at too high of a level for elementary students. In both cases, better clarity could have been provided to the students as they worked through the activity.

An example of where this occurred in study 2 is the prompt for directing students to identify different shapes for their agents. The intention of the prompt was to get students to decide if they would need to have their agent look differently during the game and read, “If this agent will have different shapes during the game play, please name them. A default shape is already available here. You may also want to sketch them out so you have an idea of what you will want it to look like. Don't take too much time with sketching though.” Only two students actually named alternate shapes (Ken & Paul, Table 17), and so it is likely that the students had no idea what this prompt was asking them to do.

Another example of unclear language is the fact that students chose CTPs that fit their planned out behavior from an everyday language perspective, but not from a programming language perspective, such as using the CTP “push,” to enact fighting within a game. The description of the “push,” CTP must have been unclear for the students (Paul & Amelia) to have used it to represent fighting between two agents. Another possibility is that the description wasn't necessarily unclear, but the students may have felt that they just didn't need to read the descriptions or that the description was too long for them to care about reading. For whatever reason, something about the CTP descriptions was just not working for some of the students and warrants some further research.

A third missed opportunity was providing a more organized, or straightforward, way for students to use their designs to create their games. The intention of the study 2 activity was for students to use the summary page from AgentDesign to guide them in creating their games, and the summary page is very text heavy and presents most of the information all at once. In retrospect, it may have been better to provide the needed design information in a way that students could step through on an agent-by-agent basis, as they would have needed to do in the programming environment. It is possible that the students with the most detailed plans simply got overwhelmed by the length of and amount of information provided in their summary pages.

Design Implications: Improving the Student Experience

Given that there were, and always will be, things that the scaffolding tool could have done better, in this section I will outline my planned changes for the AgentDesign *planning tool*. First, I would like to make the relationships between the components of a student's design more apparent on the summary page. Second, I would like to improve the wording that is used for the prompts so that they are clear and concise for an elementary student. Third, I would like to incorporate video and other graphics to improve the descriptions of individual CTPs.

As I mentioned earlier, AgentDesign was very text heavy and sequential, which was a purposeful design choice. A consequence of that choice may have been that the design summary was difficult for the students to grasp, and the relationships between agents may have been lost when the students used only the summary page to assist them with making their game. Future iterations of AgentDesign should revise how the whole of the design is presented to students on the summary page. In its current form, interactions between two agents and the corresponding CTPs are displayed in the summary page separate from any individual agent's description of behavior (see Figure 34). The intention of keeping all the CTP information in one place was to isolate the section of the design that offered sample code for the students, but I fear that it may have isolated the code to a point that it was no longer easily

understood in relation to the big picture that the student outlined in the game description and individual agents' descriptions of behavior.

Improved wording in prompts will also assist the students in better understanding what they need to do in the design process. The prompts I used likely made more sense to the undergraduates and CS Experts that were around than to the students. I would like to work with elementary literacy experts and elementary students to improve the language use of AgentDesign for any future use of the tool.

An alternative to using written language is to incorporate more graphics into AgentDesign, such as was done by Basawapatna (2012) in his Simulation Creation Toolkit. Since AgentDesign is web-based, it is possible to incorporate images, GIFs, or video into the explanatory aspects of the tool. I believe that had I provided multiple sources, and types, of information into the CTP descriptions I would not have seen students selecting "push," to represent fighting between two agents within a design.

I believe that modifying the way that prompts, explanations, and design summaries were communicated within AgentDesign would have improved the students' ability to successfully design their games. All of these proposed modifications are attainable given the web-based platform that AgentSheets was created in.

Design Implications: Modifications for Classroom Use

In addition to the modifications discussed above, an important consideration for future iterations of AgentDesign is how it may need to be altered to be best used in classrooms. A simple requirement would be that I would need to build in teacher accounts and the functionality to create and modify student accounts. But upon going through my own analysis of student data, there will also need to be a robust set of analysis tools added.

The first analysis tool will need to be a counting of agents, agent behavior described, interactions, and CTPs that the students have utilized within their design. Teachers may ask students to design their own games, or they may ask them to use a standard game description. Regardless of the

choice, knowing which students are not identifying interactions and incorporating CTPs can be a powerful tool for classroom teachers. If a student has not identified any interactions, that can be an indicator that the student may need help in understanding game design. Or if a student has not incorporated any CTPs into her design, she may need some assistance in understanding what CTPs, and algorithms, are used for in computer science.

A second analysis tool will need to allow teachers to move past the quantitative aspect of student designs and allow them to easily view the qualitative aspects. This may involve allowing each teacher to examine the design summaries for each student, but could also go as far as incorporating an automated assessment of needed agents and interactions based on language use. Giving teachers the opportunity to give students feedback about what agents they have identified or what CTPs they have chosen to use without having to look closely at every design can be extremely powerful within any classroom.

A final tool that will be useful for teachers in a classroom setting is the ability to compare a game design to the resultant programming for individual students, such as was done in Table 22. Many agent-based game repository sites, such as the Scalable Game Design Arcade or Scratch may be open to allowing a third party site like AgentDesign to do a comparison of designs to games since design is not a feature that is currently incorporated in their repositories.

Providing access to tools that analyze their students' designs, and possibly games, that are similar to what I have done for the analysis of these studies will help teachers to better incorporate design into their CS activities. The modifications I have suggested in this section may not be necessarily easy to carry out, but I do believe they will assist in the learning of core CS disciplinary practices in everyday classrooms.

Preparing Computer Science Elementary Teachers

I believe that elementary students are capable of working through an authentic CS activity of designing and creating their own games. From the data, I showed that the students were engaged in designing their own games, but seemed to have lost interest during the transition to creating the games. It may be that the setting and activity were too informal for the students to really be successful. If a more formal activity is to be used, one that is more like school, it is important to consider how to prepare teachers in formal school settings to be able to carry out an activity like this. So I would like to end this section by discussing how elementary teachers can be prepared to teach computer science in everyday classrooms.

Preparing elementary teachers to incorporate CS into their future classrooms will need to be carried out in three ways. The first is that teachers will need to develop an understanding of what it is computer scientists **DO**. The second is that future teachers will need to develop their own **content knowledge** of the CS discipline. The third is that future teachers will need to develop **pedagogical content knowledge** of the CS discipline so that they can teach students in an appropriate manner.

In many cases, future elementary teachers do not have as much experience or training as computer scientists. Additionally, there are stereotypes and stigmas for the types of people that become computer scientists. It is important that future teachers develop a realistic understanding of what computer scientists do, and that any student is capable of learning CS, so that they can then start to believe that their students can do those things as well.

A significant disadvantage that the CS discipline has when asking teachers to incorporate CS into their classrooms is that the teachers generally have no prior history with the discipline. Unlike other STEM disciplines like math or science, any future elementary teacher will likely not have experienced CS at any point in their formal education career. Since future teachers will not have this experience with the CS discipline, they must be provided with experiences to develop the needed knowledge.

The most challenging aspect of preparing future elementary teachers to teach CS in their classrooms is to help them develop pedagogical content knowledge. This is challenging because I don't believe that CS education has reached a point to have a cohesive understanding of what the best practices of teaching CS in an elementary classroom are. Some organizations will suggest that elementary students should not work with computers yet and only learn about logic and thinking through problems (<http://csunplugged.org/>), but others, including myself, would disagree that elementary students can't handle more authentic experiences of CS. Although I do concede that I may have asked students to do too much for the studies presented here. More research needs to be pursued that can contribute to answering the question of what are the best ways of exposing elementary students to learning CS.

References

- ACM Education Policy Committee. (2014). *Rebooting the Pathway to Success Preparing Students for Computing Workforce Needs in the United States*.
- Ames, C. (1992). Classrooms: Goals, Structures, and Student Motivation. *Journal of Educational Psychology*, 84(3), 261–271.
- Aspray, W., & Bernat, A. (2000). *Recruitment and Retention of Underrepresented Minority Graduate Students in Computer Science*. Washington, DC, USA.
- Atman, C. J., Adams, R. S., Cardella, M. E., Turns, J., Mosborg, S., & Saleem, J. (2007). Engineering Design Processes: A Comparison of Students and Expert Practitioners. *Journal of Engineering Education*, 96(4), 359–379.
- Basawapatna, A. (2012). *Creating Science Simulations Through Computational Thinking Patterns*.
- Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing Computational Thinking Patterns. In *SIGCSE '11* (pp. 245–250). New York, NY, USA: ACM Press. <http://doi.org/10.1145/1953163.1953241>
- Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '10*, 224. <http://doi.org/10.1145/1822090.1822154>
- Binkerd, C. L., & Moore, M. D. (2002). Women/Minorities in Computer Science: Where Are They? No Attention No Retention. *Journal of Computing Sciences in Colleges*, 17(5), 8–12.
- Black, P., & Wiliam, D. (2009). Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability*, 21(1), 5–31. <http://doi.org/10.1007/s11092-008-9068-5>
- Brown, A. L. (1992). Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings. *Journal of the Learning Sciences*, 2(2), 141–178. http://doi.org/10.1207/s15327809jls0202_2
- Cole, M., & Engestrom, Y. (2007). Cultural-Historical Approaches to Designing for Development. In J. Valsiner & A. Rosa (Eds.), *The Cambridge Handbook of Sociocultural Psychology* (pp. 484–507). New York, NY, US: Cambridge University Press. <http://doi.org/10.1017/CBO9780511611162.026>
- College Board. (2014). *AP Computer Science Principles Curriculum Framework*. New York, NY. Retrieved from <https://advancesinap.collegeboard.org/stem/computer-science-principles>
- Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design Research: Theoretical and Methodological Issues. *The Journal of the Learning Sciences*, 13(1), 15–42.
- Committee on Conceptual Framework for the New K-12 Science Education Standards, & National Research Council. (2011). *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. Washington, DC, USA.
- Cross, N. (2004). Expertise in Design: an overview. *Design Studies*, 25(5), 427–441.

<http://doi.org/10.1016/j.destud.2004.06.002>

- Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, 58(1), 240–249. <http://doi.org/10.1016/j.compedu.2011.08.006>
- Doorman, L. M., & Gravemeijer, K. P. E. (2009). Emergent modeling: discrete graphs to support the understanding of change and velocity. *Mathematics Education*, 41, 199–211. <http://doi.org/10.1007/s11858-008-0130-z>
- Dweck, C. S. (1986). Motivational Processes Affecting Learning. *American Psychologist*, 41(10), 1040–1048.
- Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014). Identifying Elementary Students' Pre-Instructional Ability to Develop Algorithms and Step-By-Step Instructions. In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14* (pp. 511–516). New York, New York, USA: ACM Press. <http://doi.org/10.1145/2538862.2538905>
- Fischer, G., & Scharff, E. (2000). Meta-Design: Design for Designers. In *DIS '00* (pp. 396–405). Brooklyn, NY: ACM.
- Fletcher, G. H. L., & Lu, J. J. (2009). Education Human Computing Skills: Rethinking the K-12 Experience. *Communications of the ACM*, 52(2), 23–25. <http://doi.org/10.1145/1461928.1461938>
- Frenkel, K. A. (1990). Women & Computing. *Communications of the ACM*, 33(11), 34–46.
- Goldberg, D. S., Grunwald, D., Lewis, C., Feld, J. A., & Hug, S. (2012). Engaging Computer Science in Traditional Education : The ECSITE Project. In *ITiCSE' 12* (pp. 351–356). Haifa, Israel: ACM.
- Ho, C.-H. (2001). Some phenomena of problem decomposition strategy for design thinking: differences between novices and experts. *Design Studies*, 22(1), 27–45.
- Kafai, Y. B. (1996). Learning Design by Making Games Children's Development of Design Strategies in the Creation of a Complex Computational Artifact. In Y. B. Kafai & M. Resnick (Eds.), *Constructionism in Practice: Designing, Thinking, and Learning in A Digital World* (pp. 71–96). Mahwah, NJ: Routledge.
- Kelleher, C., & Pausch, R. (2006). Lessons Learned from Designing a Programming System to Support Middle School Girls Creating Animated Stories. *Visual Languages and Human-Centric Computing (VL/HCC'06)*, 165–172. <http://doi.org/10.1109/VLHCC.2006.30>
- LeCompte, M. D., & Schensul, J. J. (1999). *Analyzing & Interpreting Ethnographic Data*. AltaMira Press.
- Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... Werner, L. (2011). Computational Thinking for Youth in Practice. *ACM Inroads*, 2(1), 32–37.
- Louca, L. (2005). The Syntax or the Story Behind it? A Usability Study of Student Work With Computer-Based Programming Environments in Elementary Science. In *CHI 2005* (pp. 849–858). Portland, OR.
- Lu, J. J., & Fletcher, G. H. L. (2009). Thinking About Computational Thinking. In *SIGCSE '09* (pp. 260–264). Chattanooga, TN, USA.

- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4).
<http://doi.org/10.1145/1868358.1868363>.http
- McNeill, K. L., Lizotte, D. J., Krajcik, J., & Marx, R. W. (2006). Supporting Students' Construction of Scientific Explanations by Fading Scaffolds in Instructional Materials. *Journal of the Learning Sciences*, 15(2), 153–191.
- Montanelli Jr, R. G., & Mamrak, S. A. (1976). The Status of Women and Minorities in Academic Computer Science. *Communications of the ACM*, 19(10), 578–581.
- Nasir, N. S., Hand, V., & Taylor, E. V. (2008). Culture and Mathematics in School: Boundaries Between “Cultural” and “Domain” Knowledge in the Mathematics Classroom and Beyond. *Review of Research in Education*, 32, 187–240. <http://doi.org/10.3102/0091732X07308962>
- NGSS Lead States. (2013). *Next Generation Science Standards: For States, By States*. Washington, DC, USA: The National Academies Press.
- Otero, V. K., & Nathan, M. J. (2008). Preservice Elementary Teachers' Views of Their Students' Prior Knowledge of Science. *Journal of Research in Science Teaching*, 45, 497–523.
<http://doi.org/10.1002/tea>
- Pane, J. F., Ratanamahatana, C. “Ann,” & Myers, B. A. (2001). Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54, 237–264. <http://doi.org/10.1006/ijhc.2000.0410>
- Papert, S. (1980). *Mindstorms*. New York, New York, USA: Basic Books.
- Parnafes, O., & DiSessa, A. (2004). Relations Between Types of Reasoning and Computational Representations. *International Journal of Computers for Mathematical Learning*, 9(3), 251–280.
<http://doi.org/10.1007/s10758-004-3794-7>
- Pea, R. D., Kurland, D. M., & Hawkins, J. (1985). LOGO and the Development of Thinking Skills. In M. Chen & W. Paisley (Eds.), *Children and Microcomputers: Research on the Newest Medium* (pp. 193–212). Sage.
- Pearl, A., Pollack, M. E., Riskin, E., Thomas, B., Wolf, E., & Wu, A. (1990). Becoming A Computer Scientist: A Report by the ACM Committee on The Status of Women in Computing Science. *Communications of the ACM*, 33(11), 47–57.
- Repenning, A., Webb, D. C., Brand, C., Gluck, F., Grover, R., Miller, S., ... Song, M. (2014). Beyond Minecraft Facilitating Computational Thinking through Modeling and Programming in 3D. *IEEE Computer Graphics and Applications*, 34(May-June 2014), 68–71.
- Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools. In *SIGCSE '10* (pp. 265–269). New York, NY, USA: ACM Press. <http://doi.org/10.1145/1734263.1734357>
- Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., ... Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, 52(11), 60–67.

- Rist, R. S. (1991). Knowledge Creation and Retrieval in Program Design: A Comparison of Novice and Intermediate Student Programmers. *Human-Computer Interaction*, 6, 1–46.
http://doi.org/10.1207/s15327051hci0601_1
- Robertson, J., & Nicholson, K. (2007). Adventure Author: a learning environment to support creative design. In *IDC '07* (pp. 37–44). Aalborg, Denmark.
- Sánchez-Ruiz, A. J., & Jamba, L. A. (2008). FunFonts: Introducing 4th and 5th Graders to Programming Using Squeak. In *ACM-SE '08* (pp. 24–29). Auburn, AL.
- Sandoval, W. (2004). Developing Learning Theory by Refining Conjectures Embodied in Educational Designs. *Educational Psychologist*, 39(4), 213–223. http://doi.org/10.1207/s15326985ep3904_3
- Sandoval, W. (2014). Conjecture Mapping: An Approach to Systematic Educational Design Research. *Journal of the Learning Sciences*, 23, 18–36. <http://doi.org/10.1080/10508406.2013.778204>
- Sengupta, P., & Farris, A. V. (2012). Learning Kinematics in Elementary Grades Using Agent-based Computational Modeling: A Visual Programming-based Approach. In *IDC 2012* (pp. 78–87). Bremen, Germany.
- Simon, B., Chen, T.-Y., Lewandowski, G., McCartney, R., & Sanders, K. (2006). Commonsense Computing: What students know before we teach (Episode 1: sorting). In *ICER '06* (pp. 29–40). Canterbury, United Kingdom. <http://doi.org/10.1145/1151588.1151594>
- Sivilotti, P. A. G., & Laugel, S. A. (2008). Scratching the Surface of Advanced Topics in Software Engineering: A Workshop Module for Middle School Students. In *SIGCSE '08* (pp. 291–295). Portland, OR, USA.
- Solomon, J. (2007). Putting the Science into Computer Science : Treating Introductory Computer Science as the Study of Algorithms. *Inroads - SIGCSE Bulletin*, 39(2), 46–49.
- Stipek, D. J. (1996). Motivation and instruction. In D. Berliner & R. Calfee (Eds.), *Handbook of Educational Psychology* (pp. 85–113). New York, NY: MacMillan. Retrieved from http://www.unco.edu/cebs/psychology/kevinpugh/motivation_project/resources/stipek96.pdf
- Stockard, R., Klassen, M., & Akbari, A. (2004). Computer Science Higher Education Pipeline. In *Consortium for Computing Sciences in Colleges: Eastern Conference* (pp. 102–113).
- Teague, J. (2002). Women in Computing: What brings them to it, what keeps them in it? *Inroads - SIGCSE Bulletin*, 34(2), 147–158.
- The Computer Science Teachers Association. (2012). *Computer Science K-8: Building a Strong Foundation*.
- The International Society for Technology in Education, & The Computer Science Teachers Association. (2011). Operational Definition of Computational Thinking for K-12 Education.
- Todd, K., Mardis, L., & Wyatt, P. (2005). We've Come a Long Way, Baby! But Where Women and Technology are Concerned, Have We Really? In *SIGUCCS '05* (pp. 380–387). Monterey, CA, USA.

- Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2006). *A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee*. New York, NY.
- Vygotsky, L. (1978). Interaction between Learning and Development. In M. Cole, V. John-Steiner, S. Scribner, & E. Souberman (Eds.), *Mind in Society The Development of Higher Psychological Processes* (pp. 79–91). Cambridge, MA: Harvard University Press.
- Vygotsky, L. (1986). *Thought & Language*. (A. Kozulin, Ed.) *Thought & Language*. Cambridge, MA: MIT Press.
- Webb, D. C., Boswinkel, N., & Dekker, T. (2008). Beneath the Tip of the Iceberg: Using Representations to Support Student Understanding. *Mathematics Teaching in the Middle School*, 14(2), 110–113.
- White, B., Frederiksen, J., Frederiksen, T., Eslinger, E., Loper, S., & Collins, A. (2002). Inquiry Island: Affordances of a Multi-Agent Environment for Scientific Inquiry and Reflective Learning. In P. Bell & R. Stevens (Eds.), *Proceedings of the Fifth International Conference of the Learning Sciences (ICLS)*. (pp. 1–12). Mahwah, NJ: Erlbaum.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <http://doi.org/10.1145/1118178.1118215>
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society*, 366, 3717–3725. <http://doi.org/10.1098/rsta.2008.0118>
- Wu, H., Krajcik, J., & Soloway, E. (2000). Using Technology to Support the Development of Conceptual Understanding of Chemical Representations. In *Fourth International Conference of the Learning Sciences* (pp. 121–128). Mahwah, NJ. Retrieved from <http://books.google.com/books?hl=en&lr=&id=OJM5N9PUZM8C&oi=fnd&pg=PA121&dq=Using+Technology+to+Support+the+Development+of+Conceptual+Understanding+of+Chemical+Representations&ots=dDmXc2Hwh&sig=xoI5mU4yk9Re-0a4Jsgq6V49FoQ>

Appendix A

Coded Samples of Student Design Summaries

Game: Pet Store

Description *named: 3 not: 0*

There's a pet store, and all the pets have escaped. The main character has to run around and collect all the pets in 10 minutes.

not: *ID: 2 NOT ID: 1 (store)*

Agents

aiyana:

aiyana will run around to try to find the pets

States:

- default state

pets:

States:

- default state

Interactions & CTPs *correct: 2 missing: 0 non-described: 0*

Interaction Description: aiyana will catch the pets.	
Transport <i>correct</i>	
aiyana This is the agent doing the transporting.	pets This is the agent being transported. It should not move off of aiyana, because that will stop the transporting from continuing.
Method: While Running	
IF pets is on top of me THEN Transport pets in a certain direction <i>correct</i>	

Interaction Description: User is going to move Aiyana around with the arrow keys.	
User Control <i>correct</i>	
aiyana This is the agent doing something (moving, jumping, etc.) when the user presses a keyboard key.	USER This is the user, who presses keys, but doesn't have any programmable behavior
Method: While running	
IF A key is pressed THEN I do something <i>correct</i>	

Game: Floppy Bird

Description *named: 2 not named: 0*

Bird will try to fly you have to press (spacebar) to make it fly. There is no winning! you lose by hitting a tube or letting the Bird drop.

ID'd: 2 NOT ID'd: 1 (tube)

Agents *FDZ*

Bird:

Bird tries to fly. You make Bird fly by pressing (spacebar). If you hit a tube or the bird drops you lose.

States:

- default state

correct: 0 needed: 2 (bird & user) (bird & tube) unclear: \$ non-due: 0

Interactions & CTPs *correct: 2 missing: 1 (collision) non-due: 0*

Interaction Description: Bird tries to fly. You make Bird fly by pressing (spacebar). If you hit a tube or the bird drops you lose.	
User Control <i>correct</i>	
Bird This is the agent doing something (moving, jumping, etc.) when the user presses a keyboard key.	Bird This is the user, who presses keys, but doesn't have any programmable behavior <i>incorrect</i>
Method: While running	
IF A key is pressed THEN I do something	

Appendix B

The AgentCubes Online Programming Environment

The AgentCubes Online programming environment (Repenning et al., 2014) is a web-based, 3-dimensional environment that is based on AgentSheets (described earlier). Like AgentSheets, the AgentCubes Online programming environment has visual representations of agents, in this case 3-dimensional shapes that enact programmed behavior within a virtual world, which is also 3-dimensional. Figure 43 shows what the programming environment looks like without any agents created or programmed.

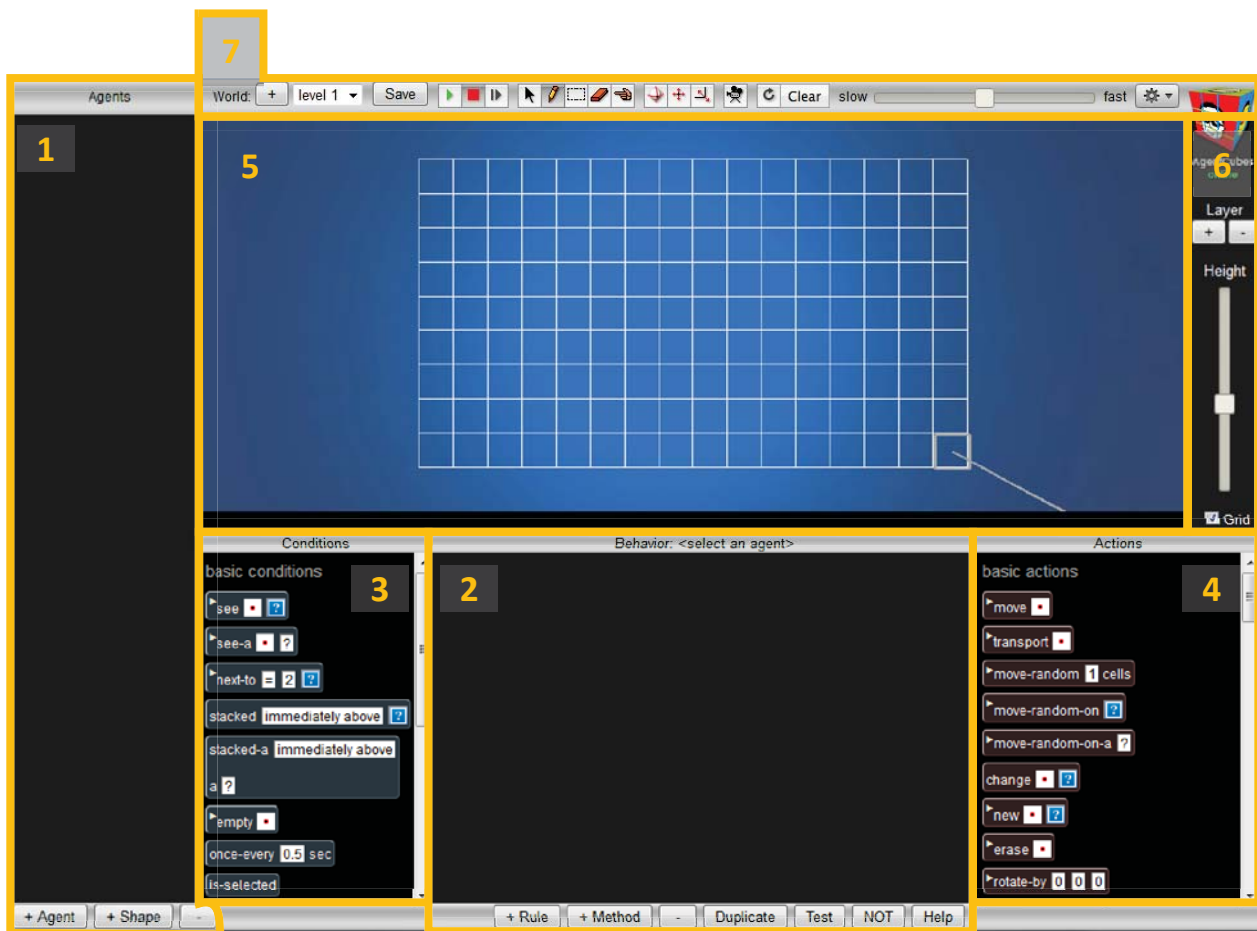
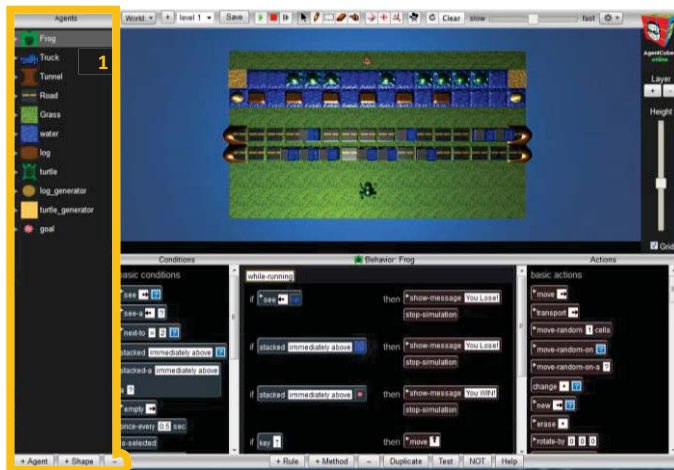
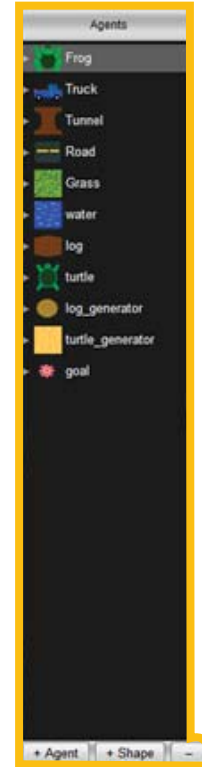


Figure 43. AgentCubes Online programming environment

Each component of the programming environment is elaborated on in the following sections, which include (1) Agents, (2) Behavior, (3) Conditions, (4) Actions, (5) Worlds, (6) Layers, and (7) Navigation. The example figures are below.

Agents (Figure 43-1)

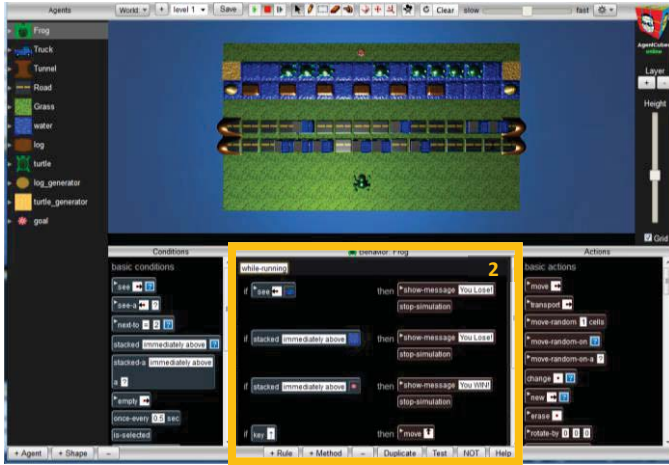
Agents are created, deleted, and accessed through area 1 shown in Figure 43. Agents can be created using the “+Agent,” button at the bottom of Figure 43-1. Different shapes can also be added to a specific agent through the “+Shape,” button. Adding a shape is useful when an agent needs to have the same behavior, but a different visual representation during the gameplay. When an agent or shape is added, it shows up in a list within area 1.



Behavior (Figure 43-2)

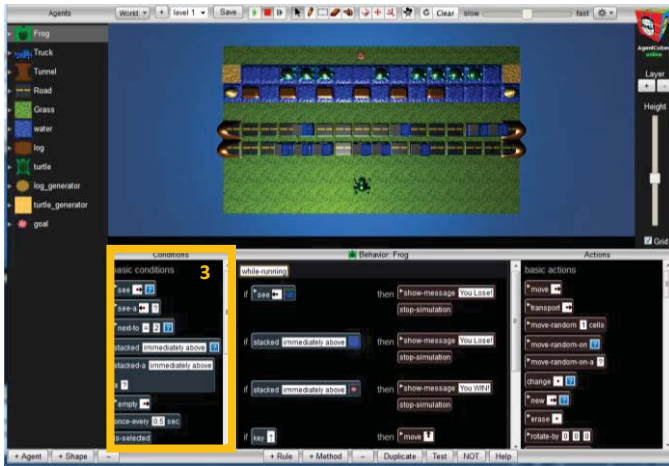
Like AgentSheets, behavior is controlled using methods and rules. An agent’s behavior can be accessed by clicking on the agent in the Agents window (Figure 43-1), and then edited by dragging and dropping conditions and actions into the rules (Figure 43-2). Rules are evaluated from top to bottom and the first rule that is true is run while the following rules are skipped.





Conditions (Figure 43-3)

See earlier AgentSheets description of conditions.



Actions (Figure 43-4)

See earlier AgentSheets description of actions.

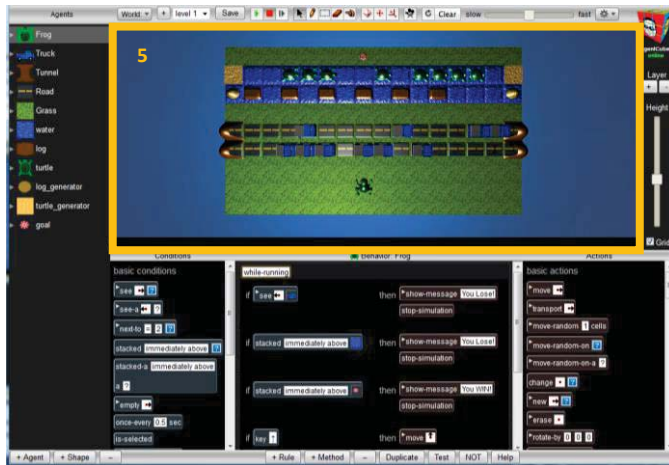


Worlds (Figure 43-5)

Worlds are the 3-dimensional environment in which the game is played. Agents can be placed in a layer in the world using the tools within the navigation toolbar (Figure 43-7). When the game is played, an

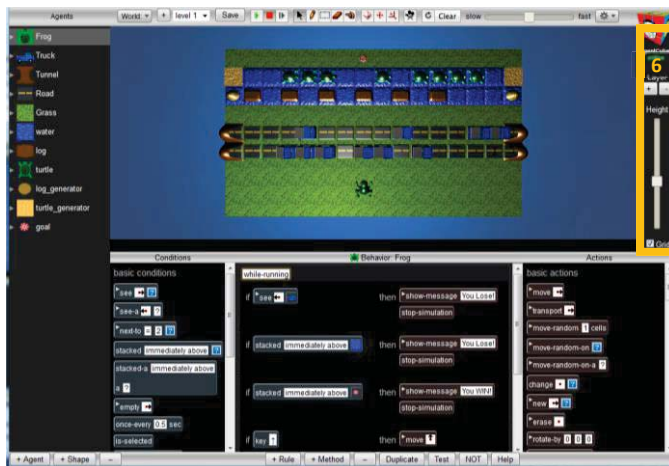


agent's programmed behavior is the only thing controlling what that agent will do within the world.



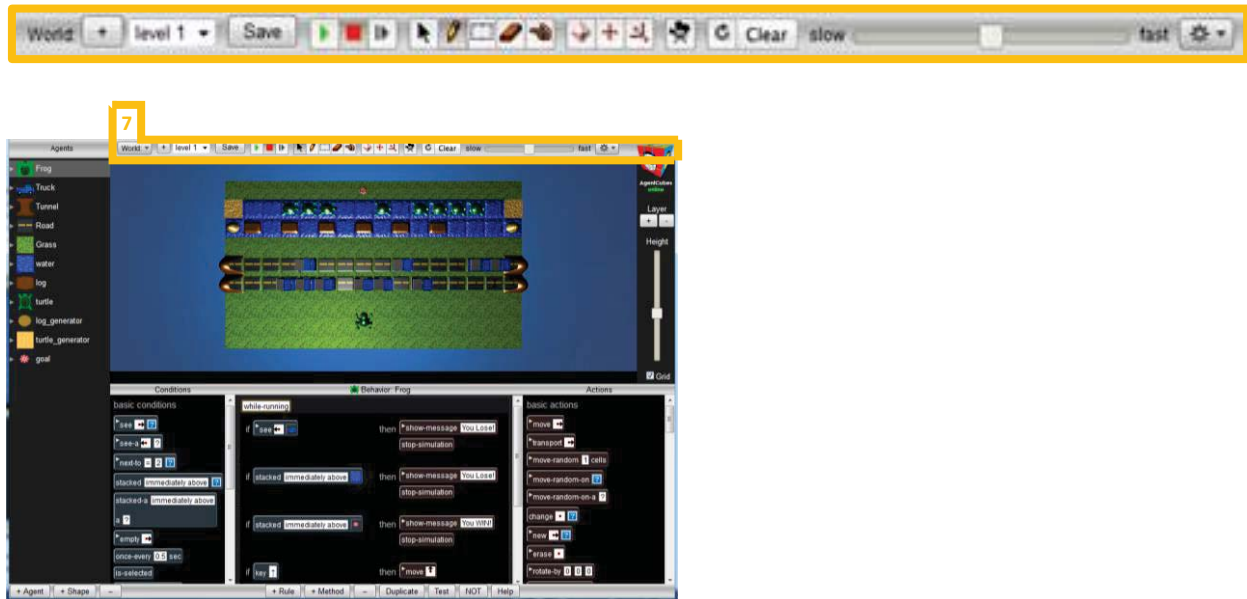
Layers (Figure 43-6)

Area 6 of the AgentCubes Online programming environment (Figure 43) provides the option to add multiple layers to the world. Many layers may be added for a world and the ability to modify the distance between those layers is also available.



Navigation (Figure 43-7)

The navigation toolbar in area 7 of Figure 43 provides the ability to add and switch between worlds, save worlds, control the running of the game, place or remove agents, and modify the view of the world.



i <http://sgd.cs.colorado.edu/>

ii <http://csunplugged.org/>

iii http://sgd.cs.colorado.edu/wiki/Category:Computational_Thinking_Patterns

iv <http://www.cde.state.co.us/cdereval/rv2011pmlinks.htm>

v <http://remixlearning.com/>