MSc thesis: Educational Games for Teaching Computer Science

Department of Computer Science and Software Engineering, University of Canterbury

Benjamin Ian Gibson

11/12/2013

CONTENTS

ABSTRACT

Much work has done on teaching Computer Science by having students program games, but little has been done on teaching Computer Science by having the students learn from playing educational games. The current work in this field does not seem to be particularly cohesive, so there is no clear idea of what has already been done, and what works. The focus of this thesis is to provide a clearer picture of the range of games available for teaching Computer Science, and to provide guidelines for designing and evaluating them.

The first and primary part of the thesis was to find and provide detailed information on as many of the existing educational games that teach Computer Science as possible. An extensive search was performed, and 41 games were found. From these it can be seen that while a few topics, mainly binary and introductory programming concepts, have sufficient coverage, most topics in Computer Science have barely been touched. Of the games for teaching Computer Science that were found, most were available online, at no cost, and only required a short time investment to play.

The second part of the thesis focuses on growing the number of games that could be used for teaching Computer Science. This is achieved by providing guidelines on producing new work, and an example game is produced to test the guidelines.

1. INTRODUCTION

1.1 Educational Games

Educational games have been proven to be effective learning tools. Research into the effectiveness of educational games is perhaps best summarized by the work of Traci Sitzmann who has produced a meta analytical review covering a variety of papers evaluating the effectiveness of educational games[34]. Sitzmann found that training using games resulted in 20% higher confidence, 11% higher knowledge of facts, 14% higher skill-based knowledge and 9% higher retention relative to a comparison group. Educational games have garnered a reputation for producing higher levels of motivation. One of the potential key benefits of this is that learners might want to use them in their spare time, and there may be some evidence for this in that Sitzmann found that games that allowed students unlimited access resulted in better learning than those that didn't. This suggests that learners are making use of this extra access outside of mandatory hours. It is for these reasons that teaching computer science using educational games is an interesting topic.

1.1.1 Blockly

Blockly Maze is a good example of an educational game that teaches computer science. *Blockly Maze*[1] is a demo/introduction for the visual programming language named Blockly, available online at `https://code.google.com/`

[1] `http://blockly-demo.appspot.com/static/apps/maze/index.html`

`p/blockly/`. In *Blockly Maze*, the player controls an avatar and must issue commands in advance to them in order to navigate a short maze 1.1. These commands behave and look like the components of the visual programming language itself. The game is short, with only ten levels, but still manages to introduce loops and conditionals. Most of the learning in *Blockly Maze* is achieved through playing the game itself rather than presenting the content through text. This means that the players can jump straight into the game and start learning how to program using the Blockly visual programming language. The game has good potential to help players to gain confidence in risk-free environment. This helps the game achieve a dual purpose of serving as an introduction to the Blockly visual programming language as well as encouraging a type of attitudinal learning. In which the player's confidence improves and consequently their attitude towards programming will probably also improve. Another strength of *Blockly Maze* is it is free and available on the web. *Blockly Maze* does not require any atypical software installation and it can probably be completed within half an hour. This means that with access to a fairly typical computing environment, students be given an encouraging introduction to programming within the scope of a single lesson, most likely with time to spare to discuss the game before or afterwards.

1.2 Thesis goals

Prior to this work we knew an unfortunately small amount about the existing educational games that teach computer science. In fact we didn't even have a clear idea of what ones exist. Research into teaching computer science using educational games is on-going and it is from this research that it is clear that there are educational games that teach computer science, as much of the research itself produces them. However what is not clear from this research is what

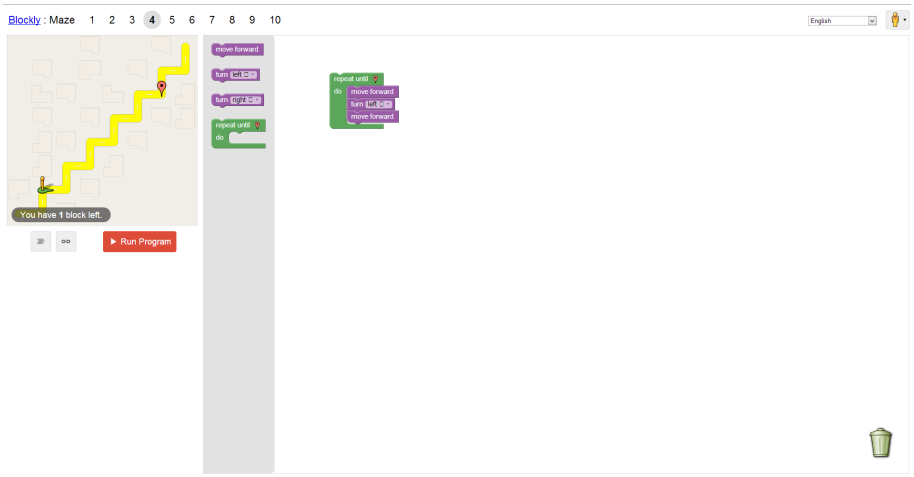

Fig. 1.1: Level 4 of *Blockly Maze*

games exist or where to look for them. The goal of this thesis is to remedy this problem. As many educational games that teach computer science as possible will be discovered and presented in this thesis. Traditionally these games will be examined based on how interesting they will be to researchers and how beneficial they will be to educators. Once this information has been gathered, it should be easier to extrapolate what might be done next in order to encourage using these educational games for teaching computer science.

1.3 Thesis structure

Chapter 2 (Significance) will answer in more depth the question of why using educational games to teach computer science is important as well as discuss in what research has already been done on using educational games to teach computer science. Chapter 3 (Identifying gaps) will briefly cover what questions have yet to be investigated and present the goal of the thesis as well as the process from which it will be achieved. Chapter 4 (Compiling a list of educational games that teach Computer Science) will cover how a list of potential games

will be populated as well as a definitions for both what a game is and what computer science is. These definitions will be such that when the list of potential games are compared to them, the games that do not meet these criteria, can be culled. This will help produce a subset of games that we can be more confident teaches computer science and produces the educational benefits associated with educational games. Nest each potential game that was found will be listed and compared to the definition of what computer science is, as well as the definition of what a game is. Chapter 5 (Developing criteria for reviewing educational games that teach Computer Science) will discuss what information might be important to gather from the games that have been identified. This chapter will then present a set of criteria that a game can be compared to. This criteria will not necessarily make value judgments about the game, rather the purpose of the criteria is to concisely describe the interesting properties of the game. Chapter 6 (Guidelines) will take the criteria presented in chapter 5 (Developing criteria for reviewing educational games that teach Computer Science) as well as the information found in the review summary and use these to discuss how we might produce future games that teach computer science. Chapter 7 (Case study) will discuss a game produced using the guidelines within the context of the guidelines. Chapter 8 (Discussion), will cover what was found throughout the thesis as well as what it might mean. Chapter 9 (Conclusion) will briefly discuss what was achieved by this thesis as well as what might be done next. Appendix A (Educational games that teach Computer Science) provides some brief information about each game that was reviewed. Appendix B (Review summary) will summarize some of the findings of the review. Some of the key results of this thesis have been published as a full paper at the WiPSCE 2013 conference [15].

1.4 Issues

While research into using educational games to teach has by and large produced positive results, this thesis has identified some potential issues. In particular it was found that some of the desirable properties of games discussed in chapter 6 (Guidelines) sometimes clash with each other and it is not always clear which properties should be prioritized. It seems that games that focus on skill-based learning are less likely to suffer from these guidelines clashes. Unfortunately for Computer Science, there are relatively few topics that benefit from such skill-based learning. A potential concern that arises from these guideline clashes is the games that encounter them may not be as effective learning tools as other educational games. This also means that these games may not be as effective learning tools as research suggests is typical for educational games. Additionally, the research for this thesis found that most games that teach computer science are quite short. This leads us to another potential issue—that of time efficiency. If the typical educational game lasts only a few minutes, then is it worth sitting a student in front of a computer to play the game? Games have been shown to be best used with a debriefing or discussion although it is unclear how promptly such a debriefing or discussion should be delivered. One way to achieve more time is if games are used as a form of homework where students would play at home then have the discussion the next day.

2. SIGNIFICANCE

This chapter will discuss why using educational games to teach computer science is a significant topic. First we cover the evolution of games as learning tools, secondly we provide evidence that educational games teach effectively, followed by a look at Computer Science within the context of educational games, and finally we discuss the practical implications of using educational games to teach.

2.1 Evolution of games as learning tools

Gee suggests that over time games have evolved into great learning tools [13, 14], although it is important to note that Gee is not specifically writing about educational games, but games whose sole purpose is entertainment. It seems Gee is making a rather bold claim. According to Gee, game designers, who likely have no training in education, are employing effective learning techniques that it would be best for schools to emulate. How is this possible? Gee's answer is that game development companies have had to learn how to implement good teaching techniques in order to survive. In other words games that did not employ good learning techniques were less successful than games that did, leading to an industry appreciation of such learning techniques.

Games that employed good learning techniques were successful primarily because players wanted games that were challenging and complex. Most modern games are proof of this; in fact, the level of challenge and complexity in some games can be off-putting to some new players. These games become highly

competitive, such as *Quake live*[1], *Starcraft 2*[2] and *League of Legends*[3]. Simple or easy games would not get the same type of attention from players, who would quickly become bored.

Such complicated games require that the user must learn how to play, and in some cases how to compete. This is where the learning techniques come in. Game developers had to develop effective learning techniques in order to teach people how to play their games. A good example of this is the game *Portal 2*[4], which consists of learning how to use fictitious technology to escape traps. The player advances from one technology to the next after a series of carefully design traps that teach the player how to utilize the technology. The game culminates in a battle in which the player must make use of everything they have learned, much like an exam. However, unlike a normal exam, this is an exam everyone is expected to pass. This is not because the exam is easy, but because the player has been given exactly the tools they need to succeed throughout the game. Such techniques include: Gee describes some of the learning techniques employed by games, such as suggesting that learning in games is "on demand" and "just in time", which is possible because the learning, typically, takes place within an appropriate context [14]. This is significant because people aren't very good at remembering information learned out of context or too long before they can make use of it (Barsalou, Brown; Glenberg and Robertson, as cited by Gee (2003) [13]). Another such technique Gee describes is that games strive to offer challenges to players that match the player's ability as closely as possible. This is in contrast to a normal class room environment where it is hard to teach to each individual's needs. Gee posits that games often allow players to be producers as well as consumers and that this does not happen often enough

[1] `http://www.quakelive.com/`
[2] `http://sea.battle.net/sc2/en/`
[3] `http://oce.leagueoflegends.com/`
[4] `http://www.thinkwithportals.com/`

in schools Brown (1994) (as cited in Gee (2003)[13]). Additionally Gee argues that games order challenges such that players form good generalizations early on, which is beneficial to learning, as shown in Elman 1993 (as cited in Gee (2003)[13]). Gee wrote that games utilize "a cycle of expertise" as described in Bereiter and Scardamalia (as cited in Gee (2003)[13]).

Gee backs up most of his claims with some encouraging evidence. However the evidence given doesn't fully back up the strength of his claims. To know that games have truly evolved the way Gee has claimed it would be necessary to compare older games to newer ones. Additionally it would be interesting to see if games including the properties Gee has listed are likely to be more successful than those that don't.

2.2 Evidence for the effectiveness of educational games

Much research has been done into using educational games to teach [11, 29, 21, 17]. The emphasis of this research has been on how effective using games to teach is compared to traditional teaching methods. This is perhaps because games are seen as primarily for entertainment, not more serious things such as education. This research has been ongoing for decades [37]. Yet despite the wealth of encouraging research on the subject, games haven't found their way into education as much as one might expect. Rather, some researchers, such as Gee [13], are instead focusing on bringing inspiration from educational games into how we teach in the classroom (such as quick feedback to students).

Much of the research into educational games has been on the effectiveness of educational games. Most of this research has found that educational games are effective teaching tools [17, 2, 27]. In order to give a clear idea of the findings of this research, Traci Sitzmann completed a comprehensive meta-analytic review [34]. Sitzmann reviewed more than 60 papers evaluating the effectiveness of

educational games, including unpublished papers. Unpublished papers were included to combat any potential publication bias, for which Sitzmanns research showed evidence. The broad base that of research that Sitzmann was working with allowed for comparing games to a variety of different instructional methods. Additionally, Sitzmann was able to compare properties, such as active or passive teaching to see which produced better results. Sitzmann statistically tested ten hypotheses about the effectiveness of educational games. Evidence was found to support all hypotheses, except one, the only unsupported hypothesis was that games with certain entertainment aspects would teach more effectively.

It has been suggested that there is a publication bias at play, in that it is claimed that research with results that show educational games more favorably are more likely to be published. Sitzmann factored for this in by including some work that failed to get published [34]. Sitzmann then compared the results of the unpublished work to the work that did get published. The unpublished work was found to be less favorable about the effectiveness of educational games, supporting the idea that there is a publication bias. These papers were included in Sitzmann's study in order to help reduce the effects of any potential publication bias.

Sitzmann found that educational games were more effective than lectures, reading, videos, assignments, discussion, the combination of computerized tutorial and assignment and the combination of lecture and instructional methods. These games were found to be less effective than hands on practice, computerized tutorials and the combination of group activities and discussion. However, when games and other instructional methods were compared with equal levels of active engagement there was no significant difference in effectiveness. This says that good educational games are equivalent to our best teaching techniques and more effective than most of our more popular teaching techniques, such as

lectures. In fact Sitzmann found that on average games resulted in 9% better retention, 11% better factual knowledge, 14% better skill based knowledge and 20% higher self-efficacy. However, it was also found that in order for games to be used most efficiently, they must be used alongside other instruction such as debriefings. Such debriefings consist of discussion, post game, about how the game relates to the topic.

2.3 Computer science

Work that has been done on teaching computer science using educational games tends to have a focus on making games rather than evaluating their educational effectiveness on a large scale [2, 8, 18, 16, 23, 26, 27]. This makes a certain amount of sense, if we assume that computer science is not fundamentally different than other subjects as far as teaching them in games is concerned. From this assumption it would make sense that games that teaching computer science would be just as effective as games that teach any other subject. Since there has already been a lot of research about the effectiveness of educational games, it would then be unnecessary to do a large amount of research about how effective teaching computer science with games is.

It appears there are only a few groups that have reported focused research on using educational games to teach computer science. These groups are the "Level up" group [8], the Game2Learn group [2] and the SimSE group [27]. "Level up" has students producing games that teach computer science and then employs them in other computer science classes. There are several published papers on one of the games produced by "Level up", Wu's castle [9, 8, 10, 7]. Wu's castle is in the style of older role playing games. The game is designed to teach players about arrays and loops, a topic that was chosen because it is a difficult subject in introductory courses [8].

The Game2Learn group gamified the virtual beadloom software, which simulated beadlooms of native American origin. The virtual beadloom software included more efficient ways of creating virtual bead art by using functions. However the beadloom group found that these functions were very rarely used, even though it would save time and effort. The beadloom group made the software into a game by adding a goal of using as few steps as possible. This successfully encouraged students to use the more efficient functions [2].

SimSE is a large software engineering simulation game. Rather than directly teaching software engineering principles, it aims to give the player experience of what working in a software engineering team is like. This is achieved by having the player give commands to a virtual software development team working on a project. There are several different versions of the game, each representing a different software development process. The ways the player can command their developers and how the development proceeds is dependent on which of these versions is installed.

Several more pieces of research have been done on using educational games to teach computer science. In each of these a game of some kind that teaches computer science is made. These games cover a wide variety of computer science topics: Komisarczuk and Welch developed an Internet engineering board game [23] that teaches internet peering; Soh used game days [35] to teach multi agent systems; Hill, Ray, Blair and Carver [20] utilized puzzles and games to teach operating systems; Shifroni and Ginat [33] developed a simulation game that teaches communication protocols; Brereton, Donovan and Viller [3] developed a video card game for teaching observational skills within the context of human computer interaction; and Wein, Kourtchikov, Cheng, Gutierez, Khmelichek, Topol and Sherman [36] used firefighting-like game scenarios to teach distributed systems.

Research has also been done on teaching computer science by having programming students make games [32]. While in research such as this games are being used to teach computer science it is important to note the students are not playing the games. The focus of this thesis is on the effectiveness of using educational games to teach, and so is specifically about having the students learn through playing the games,so the work on students developing games is not particularly relevant to this research, although we note that creating games is a strong motivator for some students.

2.4 Practical implications

Producing educational games can take a lot of time and be very expensive, sometimes millions of dollars [34]. Educators and researchers often lack the resources, time or know-how to produce such games. Each of these factors explain why such games are not more widely produced. Additionally, it could explain a strong focus on game concepts that are more likely to have good returns for a small investment—educational concepts that are easy to turn into games are the safest choice when investing large amounts of money and time. This leads to having a few topics covered particularly well, with very little, if any, content for the others.

Although the production of educational games is expensive and time consuming compared to more traditional lesson plans, this is counterbalanced by the long lifespan of a game. A game's lifespan is, in theory, potentially limitless, although realistically operating systems will be upgraded over time and eventually a game will no longer be supported. Fortunately this is unlikely to become an issue for any such game for years, more likely over a decade. For example, *the Oregon Trail* is an educational game developed in 1971 that is still popular

today. *The Oregon Trail* has even seen a remake for the Iphone[5].

In addition to being usable for years, video games can potentially be used over and over for free anywhere in the world, assuming that the producer of the game releases the game for free. Unfortunately this leaves the all the cost and time investment with the producer of the game. Perhaps web sites such as kickstarter[6] could be used to help solve this issue as with *Robot Turtles*[7]. This way the community could, at least, provide funding to the game producer.

Once produced, a game can be reused with much less financial or time investment. Additionally, video games consistently maintain the same level of quality. A single well-produced game can provide effectively limitless high quality learning experiences to people all over the world. These games can even, to a certain extent, be employed without instruction from an educator.

[5] https://itunes.apple.com/en/app/the-oregon-trail/id307519882?mt=8
[6] http://www.kickstarter.com/
[7] http://www.kickstarter.com/projects/danshapiro/robot-turtles-the-board-game-for-little-programmer

3. IDENTIFYING GAPS

As discussed in the previous chapter, the results of the research into using educational games to teach have been encouraging, such as Sitzmann finding using a combination of educational games and discussions or debriefings as one of the most efficient approaches tested [34]. However, only a small amount of research has been done on teaching computer science using educational games. Given the apparent effectiveness of educational games it seems that making use of them for teaching computer science would be an appealing idea.

However, SIGCSE[1], a research conference specifically focused on teaching computer science, had relatively few published papers about educational games. The reasons for this are unclear.

Most of the existing research about teaching computer science using educational games has focused on creating new games and testing their effectiveness [8, 9, 7, 10, 2, 27, 28, 35, 20, 23, 33, 3, 36]. The results of this research seemed to correspond with the findings of research into the effectiveness of educational games in general. However, the results only appear to correspond in that they were both positive results. It is unclear if educational games that teach computer science are any more or less effective than educational games that teach other subjects.

[1] `http://www.sigcse.org/`

3.1 Justification of topic

Each piece of research to do with games that teach computer science appears to have been done without much recognition of the other research done in the field. The coverage of computer science topics in the research is somewhat haphazard, and driven either by local needs or interests. Additionally, it is unclear what topics have already been covered by educational games. Some of the research justifies their chosen topics, such as *Wu's Castle* [8] picking arrays and loops, because they a difficult topic for beginning students. However, none of the researched reviewed made any mention of the existing games and the topics they covered. This is likely to be because it is difficult to identify what topics have already been covered by educational games, and this is a gap that this thesis aims to fill.

3.2 Justification for pedagogical approach

There appears to have been a number of different approaches to creating games that teach computer science put into practice, such as *the BeadLoom Game* 3.1 using gamification with their virtual bead loom software in order to motivate players into using more advanced approaches [2]. Gamification is the process of adding game-like elements such as scoring to an activity in an attempt to increase motivation. Another example is SimSE. which uses simulations of a software development team following various software development processes. In these two examples the first is using game elements to motivate learners to attempt more difficult tasks, while the other is using a simulation to provide learners with experience. Although games applied different approaches such as these, the merits of their approaches are rarely discussed, let alone tested. It is unclear what the advantages and disadvantages of the various approaches taken throughout the games are. We have evidence that suggests that most of

these approaches have good results, but are unfortunately left poorly equipped to properly discuss why.

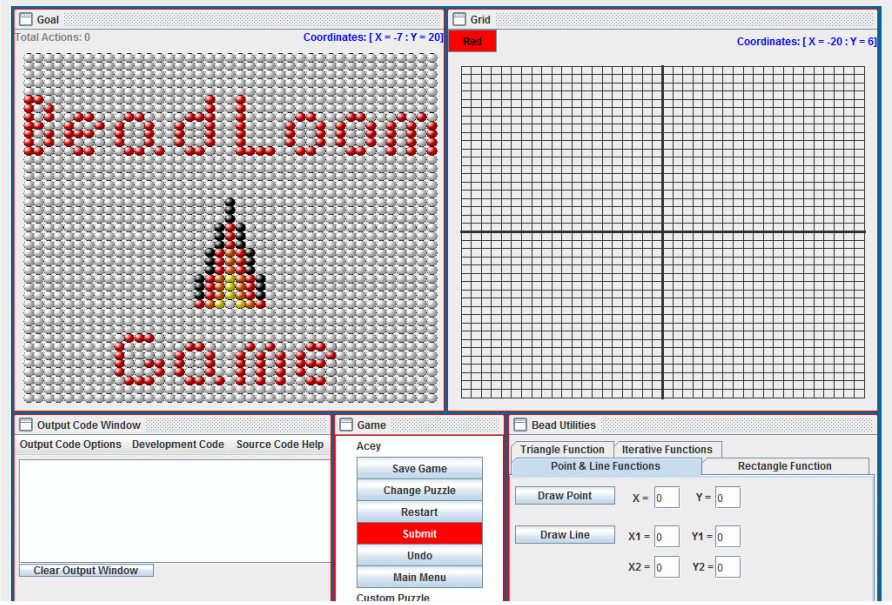

Fig. 3.1: The BeadLoom Game

3.3 Goal

The primary goal of this work is to get a clearer picture of the state of the field. As we have already found earlier in this chapter, the current research into using educational games to teach computer science has failed to identify the extent of the body of work. If we don't have a clear idea of what has already been done, any new pieces of research risk duplicating existing work. It is possible that this has already been happening. Games such as *Light-Bot* [2] 3.2 and *the Cisco Binary Game* [3] 3.3 teach computer science and yet seem to have been produced independently from research. This has a large impact on research into

[2] `http://armorgames.com/play/2205/`
[3] `http://forums.cisco.com/CertCom/game/binary_game_page.htm`

teaching computer using by using educational games. At least insofar as what educational games that teach computer science exist it is vital to consider all games, not just those described in published academic articles. The goal of this research is to gather information about as much of the body of work, including work independent of research such as Light Bot, as possible.

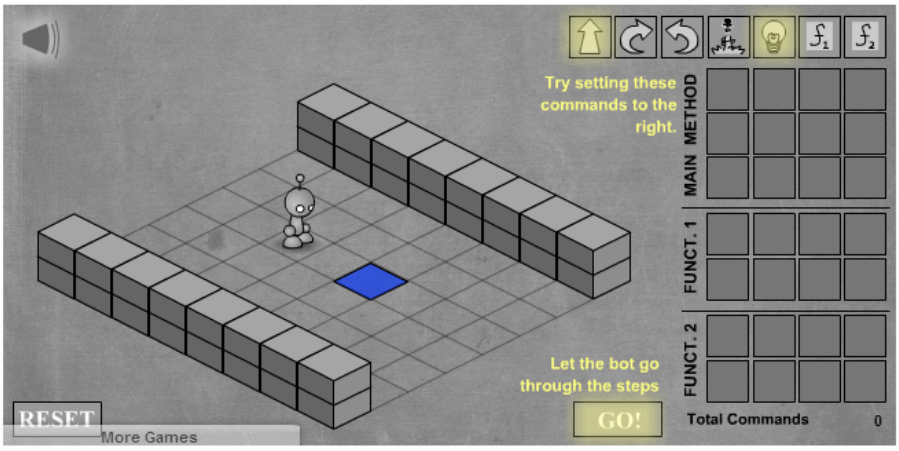

Fig. 3.2: Light-Bot

3.4 Research Method

This section will cover how the research will be conducted. First existing games that teach Computer Science will be identified, next valuable data about these games will be gathered and finally a set of guidelines for creating educational games that teach Computer Science will be produced.

3.4.1 Finding games

One of the key questions about the state of this area of research is simply what resources already exist. The benefits of teaching using educational games have been well established [34], but there only seems to have been a passing interest in putting them into practice. Even if an educator was interested in teaching

Fig. 3.3: The Cisco Binary Game

computer science using educational games it would currently be difficult for them to find games that suited their purposes, or even to find out if such a game exists. It is primarily for this reason that the key piece of information this research aims to gather about this body of work is what resources already exist. All other data gathered about the field will be based on this-once a list of existing games has been gathered it would be valuable to investigate and report on each one.

3.4.2 Review

Once we have a clearer idea of what existing educational games that teach computer science exist it is useful to know which parts of computer science they teach. This will tell us which computer science subjects currently lack coverage and perhaps any that are sufficiently covered. This information could

be very useful for educators wishing to utilize the benefits of educational games, to find out which topics they could potentially expect to use educational games to teach. Additionally this information would be valuable for anyone looking to expand on the catalogue of existing games. Once this information is compiled it would become much easier to decide which topics future work should aim to teach by looking at what topics have not already been covered. Another benefit of compiling information on computer science topic coverage would be to see if some topics are covered more than others. We may find that some topics are more appealing than others, at least when it comes to creating educational games about them.

The information will be gathered with a broad audience in mind; those interested in educational games may have different motivations and need different information. People interested in expanding the existing catalogue of games might be interested in information such as what types of learning games tend to target, while people interested in using a game in a learning environment, such as a classroom, might be more interested in how easily a game can set up. This research will be aiming to provide useful information for anyone looking to make use of educational games that teach Computer Science.

Information about how engaging a game is likely to be will be gathered. One of the key strengths of educational games appears to be the level of engagement they realize with learners [29]. One may find it beneficial to use educational games as supplements to other forms of instruction if only to aim for a higher level of engagement and interest in the subject. For these reasons this research will aim to provide useful information about how engaging each game is.

Traci Sitzmann [34] has identified several key components from educational games that help them achieve engagement with learners. Each game will be investigated for each of these components. In some cases this will involve deci-

sions that amount to little more than anecdotal speculation, however even this kind of information can be useful if taken with the appropriate skepticism.

Each game will also be studied in terms of how deployable it is. An educational game is beneficial only if it can actually be used in practice. This is not as simple as having an educational game that 'works', for it to be put into practice it must also work in an environment that can be realistically be expected to be available to educators. Learning environments vary too much to prescribe a simple list of recommendations or requirements for the educational games being studied. As such this research will aim to gather data on the deployment requirements of each game without making specific claims about how deployable a game is. For example, many schools have computers locked down so that new installations are difficult, and for this reason a web-based game would be more desirable, since no installation is required. In another context Internet access might be poor, and an installed game would be more effective.

The deployment requirements of each game will be presented in terms of several categories, as well as any miscellaneous notes. Such categories will include the deployment platform, financial costs and time requirements. Deployment platforms could include Mobile platforms, Cloud based games (web-based), games that require installation on a computer (desktop) and games that can be played without a computer at all ("unplugged").

Games will be reviewed in terms of the types of learning encouraged, as it would be useful to know what type of knowledge a learner may receive from playing a game. For example, in many cases it is not particularly useful for computer science students to learn skill based knowledge when it comes to decimal to binary conversion. This because computer science students do not benefit much from being skilled at decimal to binary conversions; they primarily interested in understanding the underlying concepts. Additionally, researchers may

be interested to see what kinds of learning have been targeted by most of the existing games. It could be interesting to see if particular types of learning are more common in educational games than others. If one were looking to create new educational games it would be useful to know not only what topics have been covered, but how they were covered. For each of these reasons data will be gathered from each game about the types of learning they encouraged.

Ultimately the aim of this research is to gather information that is useful to anyone interested in educational games that teach computer science about what such games already exist. This will be achieved by compiling a list of existing computer science games and comparing each one to a list of criteria specifically targeted at providing useful information.

3.4.3 Guidelines

Once this information has been compiled it will be easier to expand the field. Future works of research will be able to look at what already exists before they contribute. When contributing in this way it would be useful to be able to find the best way to contribute in terms of each the criteria outlined above. It should prove relatively simple to adapt the criteria into a set of guidelines to provide a rough starting point for anyone contributing to the field in future. In this way this research will provide information on what needs doing and how to achieve it.

To test the guidelines developed, we also report on an example of the results of producing a game by following the guidelines and with the results of the review in mind. This should result in an educational game we can claim, with confidence, didn't exist before, and is useful for educational purposes.

4. COMPILING A LIST OF EDUCATIONAL GAMES THAT TEACH COMPUTER SCIENCE

4.1 Definitions

In order to properly compile a list of games that teach computer science a definition of both what a game is and what computer science is will be needed.

4.1.1 Definition of Computer Science

Defining the scope of "computer science" is a challenge; there are many definitions, and the term is sometimes used loosely. Furthermore, there are variants such as "computing" and "informatics" which can have different meanings in different countries. For this research the curriculum guidelines provided by the ACM have been chosen. ACM are described as "the world's largest educational and scientific computing society". The ACM/IEEE curricula are widely used educational benchmark; and many curricula around the world are based on them. The current active ACM/IEEE computer science curriculum was released in 2008, but we have also incorporated changes mooted in the 2013 revision [30][1], which is being finalized at the time of writing.

4.1.2 Definition of a Game

For the purposes of this list a modified version of Roger Caillois's [5] definition of a game, as described by Salen and Zimmerman [31]) will be used. The

[1] `http://ai.stanford.edu/users/sahami/CS2013`

modifications are the removal of the clause 'Unproductive', as educational games are necessarily productive. This clause is replaced with an educational clause saying that the game must teach a computer science subject, as described later in this chapter. Additionally the clause 'Free', which stipulated that a game is something that one plays at their own discretion, has been removed. This clause was removed because the intent of this list of criteria is to provide information for researchers and educators, it is their decision if they present games in such a way that they are free or not. However, by keeping the key criteria we can ensure that we are talking about the kind of game that has been reported as beneficial. Given these changes to meet this definition a game must have the following characteristics:

Separate: circumscribed in space and time, defined and fixed in advance;

Uncertain: the course cannot be determined, or the result obtained beforehand, with some latitude for innovations being left to the player's initiative;

Governed by rules: under conventions that suspend ordinary laws, and for the moment establish new legislation, which alone counts;

Make-believe: accompanied by a special awareness of a second reality or of a free unreality, as against real life;

Educational: the activity teaches a computer science topic; that is, the student understands a topic better or is more skilled at a task after having played the game.

4.2 Identifying potential games

The first step in identifying relevant games was compiling a list of anything that looked remotely like a game that taught computer science. To populate this list

we consulted both academic and non academic sources; academic sources were predominantly through the University of Canterbury's online library, subsidiary databases and Google scholar. From the University of Canterbury's online library 'SIGCSE' publications were given the most detailed focus. Every conference from the SIGCSE database was thoroughly scrutinized for relevancy. Contributions from the SIGCSE mailing listwere also called for. The list was cumulatively compiled and posted on Wikipedia[2] (at the time of writing it is still only as an article for creation, but it is still accessible to those provided a link) calling for members of the SIGCSE mailing list to contribute games that were not on the list. Numerous resources were found on the CS Unplugged website[3]. The CS unplugged website was a useful hub of information with access to information and games that taught computer science as well as referring to various other sites via hyper-links. Hundreds of Google searches were employed, including utilizing a python script to make this more efficient. A lot of terms that related to computer science, mostly taken from the ACM Curriculum guide, were used predominantly for these searches. Google searches did not prove to be very efficient or helpful for this research. Each game from this list was then compared to both the definitions presented earlier in this chapter, that of a game and that of computer science.

The following tables show all the activities found that looked like they may have been educational games that teach computer science. Each table represents activities of different types, specifically web based, mobile, unplugged or desktop. Some activities belong to more than one of these categories, in which case they are only described in one of the tables and which table they're described in is based off what seemed most appropriate, such as *RoboZZle* which

[2] `http://en.wikipedia.org/wiki/Wikipedia_talk:Articles_for_creation/List_of_educational_games_that_teach_computer_science`

[3] `http://csunplugged.org/`

was first released as web-based game[4] 4.1 but is now also available as a mobile game[5][6]. The first column indicates the activity's name. The remaining columns represent each of the criteria the activity must meet in order to be considered a game for the purposes of this research. An activity is only considered a game if it meets these 5 criteria, although an activity will also be excluded if it is not available.

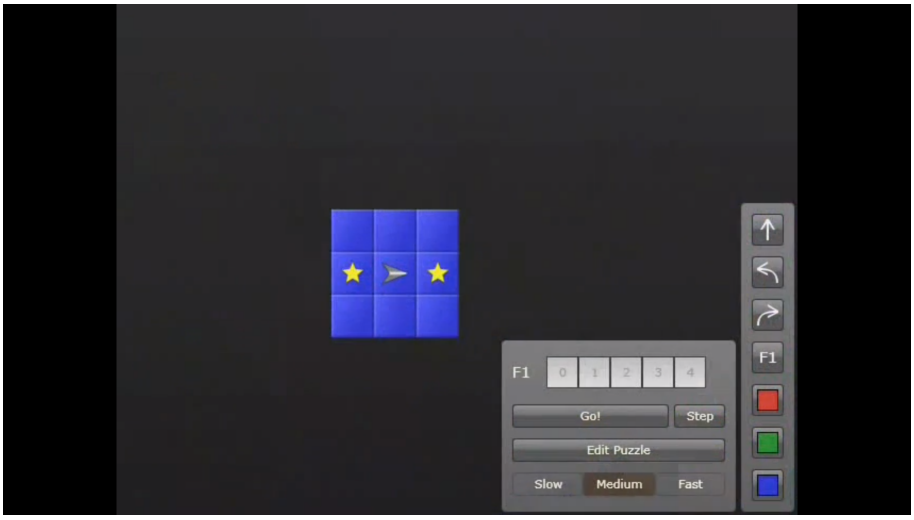

Fig. 4.1: RoboZZle as a web game

Several activities were found that were not available. This meant that either the activity could not be accessed or reviewed because of a language barrier, or that while information about the activity was available, the activity itself wasn't, sometimes because the activity has not yet been released, but in a few cases, because the activity used to be available, but can no longer be accessed. Activities that were not available could not be accurately compared to the rest of criteria to test if they are games or not, so in these cases the unknown information is marked with a question mark.

[4] `http://www.robozzle.com/`
[5] `https://play.google.com/store/apps/details?id=com.team242.robozzle&hl=en`
[6] `https://itunes.apple.com/en/app/robozzle/id350729261`

The criterion 'Separate' was not used to exclude any games. This could suggest that it was not a very useful part of the criteria, since it did not help at all in deciding if any activities where games or not. Alternatively it could suggest that the purpose of the criteria has been misunderstood and the criteria has not been applied properly. The understanding of this criteria that was used for this research was that the activity's boundaries were well defined in advance, such that arbitrary actions do not constitute a part of the activity. It is also possible that the activities that this criteria would have excluded were prematurely excluded for not 'looking' enough like a game, or that it did not exclude any activities through chance.

The criterion 'Uncertain' was used to exclude 19 activities. This criterion excluded more activities than any other. This could suggest that either this criterion was the most useful or overly restrictive. This criterion tended to exclude activities that had fixed outcomes, such as the *Treasure Hunt activity*[7] and activities where the players didn't have sufficient room for innovation. Lack of player innovation was apparent in activities where the participants only got to follow instructions, and the results of activity were left up to chance, such as *Tetradice*[8], or as soon as the player learned the trick behind the game there was no longer any innovation to be had, such as *the 21 card trick*[9].

The criterion 'Governed by rules' was used to exclude two of activities: one that was particularly vague: *"Google page ranking"*[10], and one that had no goal: *logic gate simulator*[11]. The *Google page ranking* activity asked the players to perform a task, but gave them no restrictions on how to perform it or any criteria for success. The *logic gate simulator* gave the player a set of tools to build logic gates with, but set no criteria for success, making it behave more

[7] `http://csunplugged.org/finite-state-automata`
[8] `http://www.tetrakys.es/en/tetradado`
[9] `http://www.cs4fn.org/mathemagic/magicshuffles/`
[10] `http://csi.dcs.gla.ac.uk/`
[11] `http://www.kolls.net/gatesim/`

like a toy than a game.

The criterion 'Make believe' was difficult to use as a deciding factor because it required distinguishing between make believe and abstraction. For example, crossword puzzles do not provide the player with a sense of another word, but they are so abstracted that they don't really have much to do with reality either. For this reason the deciding factor that was applied to these abstracted activities was what they were being used for — a problem as part of a puzzle has much less bearing on reality than a problem as part of an exam.

Some activities border on teaching Computer Science, such as *Cisco Edge Quest*[12] and *Cisco Edge Quest 2*[13] 4.3. These two games probably include some information that would be useful when teaching about networks, but the focus of the games appears to be on teaching about CISCO routers. Several other activities had similar issues, although there were also activities that didn't end up teaching computer science in any significant way, such as *SpaceChem*[14] 4.2. Another kind of game that bordered on teaching Computer Science was identified: games where the key game-play mechanic involved programming such as *Core Wars*[15]. However, these games were not included onto this list unless the game also taught programming, rather than just using it to play. This isn't to say that such games could not be used to teach programming, rather that they appear to be more competitive than educational. Some of these games, such as *Code Hero*[16], use programming as a key gameplay mechanic as well as teaching programming, these games were not excluded from this list.

[12] `https://learningnetwork.cisco.com/docs/DOC-7633`
[13] `https://learningnetwork.cisco.com/docs/DOC-7634`
[14] `http://www.spacechemthegame.com/`
[15] `http://www.corewars.org/`
[16] `http://primerlabs.com/codehero`

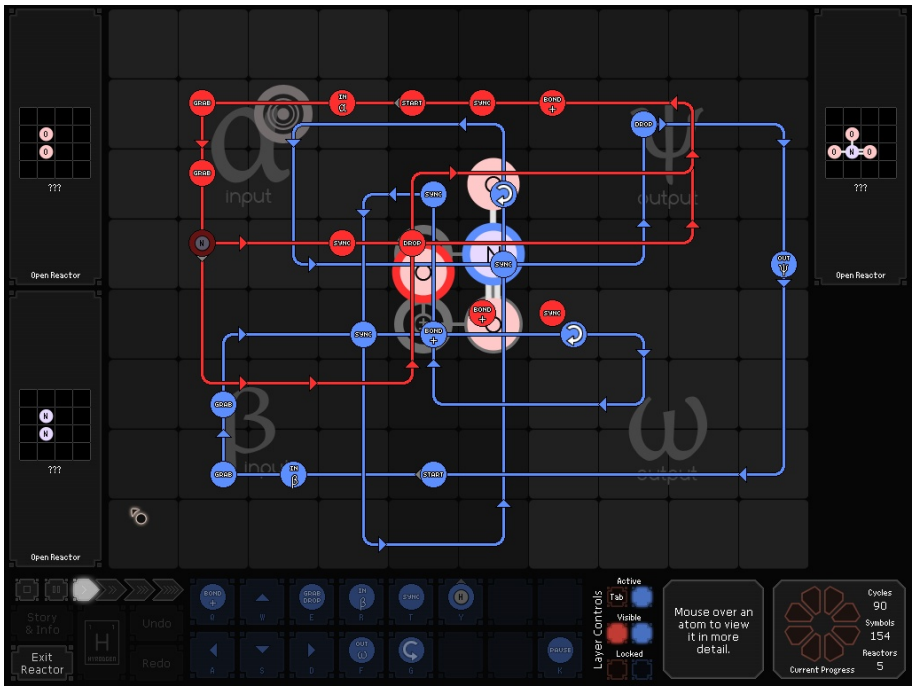

Fig. 4.2: SpaceChem

4.3 Evaluation of games

Tables 4.1 to 4.5 show an evaluation of more than 80 activities that were candidates for being games. The evaluation used the criteria discussed earlier in this chapter. The games that meet the criteria are shown in Tables A.1, A.2 and A.3 in Appendix A.

Tab. 4.1: Evaluation of potential web games

Name	Separate	Uncertain	Governed by rules	Make believe	Educational
0's and X's (cs4fn)	✓	✓	✓	✓	✗
21 Card Trick	✓	✗	✓	✓	✓
BeadLoom Game	✓	✓	✓	✓	✓
Binary Card Game	✓	✗	✓	✓	✓
Binary flash Cards	✓	✓	✓	✓	✓
Binary Gun	✓	✓	✓	✓	✓
Binary Number Quiz	✓	✓	✓	✓	✓
Blockly Maze	✓	✓	✓	✓	✓
Brando The Egg Hunter	✓	✓	✓	✓	✓
Cisco Binary Game	✓	✓	✓	✓	✓
Cisco Edge Quest 2	✓	✓	✓	✓	✗
Cisco Multiplayer Callenge	✓	✓	✓	✗	✓
Cisco Subnet game	✓	✓	✓	✓	✗
Cisco Unified Communication Simulation Challenge	✓	✓	✓	✓	✗
Cisco Wireless Explorer	✓	✓	✓	✓	✗
CS Unplugged Binary	✓	✓	✓	✓	✓
Dead Mans Chest	✓	✓	✓	✓	✓
Hamster Nim	✓	✓	✓	✓	✗
Light-Bot	✓	✓	✓	✓	✓
Light-Bot 2.0	✓	✓	✓	✓	✓
Manufactoria	✓	✓	✓	✓	✓
Orange Game Flash, the	✓	✓	✓	✓	✓
Picture Logic the Puzzle Game	✓	✓	✓	✓	✓
Pinky's Pipe Pickle	✓	✓	✓	✓	✓
RoboZZle	✓	✓	✓	✓	✓
Sorting Bricks	✓	✓	✓	✓	✓
Swap Puzzle	✓	✓	✓	✓	✓
Tour Finder	✓	✓	✓	✓	✓
Tour Finder, Two Player	✓	✓	✓	✓	✓
Treasure Hunter	✓	✓	✓	✓	✓
Turing Test Jokes	✓	✗	✓	✗	✓

Tab. 4.2: Evaluation of potential unplugged games

Name	Separate	Uncertain	Governed by rules	Make believe	Educational
Algorithms (cs inside)	✓	✗	✓	✓	✓
Battleships Unplugged	✓	✓	✓	✓	✓
Binary Crossnumber Puzzle	✓	✓	✓	✓	✓
Binary Puzzle	✓	✗	✓	✓	✓
Communications Protocols (cs inside)	✓	✓	✓	✓	✓
Crossbin Puzzles	✓	✓	✓	✓	✓
CS Unplugged Information Theory	✓	✗	✓	✗	✓
CTRL-ALT-HACK	✓	✓	✓	✓	✓
D0x3d!	✓	✓	✓	✓	✓
Distributed Computing (cs inside)	✓	✗	✓	✓	✓
Google and Page Ranking (cs inside)	✓	✗	✗	✓	✓
Graphs and Reasoning (cs inside)	✓	✓	✓	✓	✓
Image Identification (cs inside)	✓	✓	✓	✓	✗
Image Representation (cs inside)	✓	✗	✓	✓	✓
Information Retrieval Indexing (cs inside)	✓	✗	✓	✓	✓
Internet Peering Game	✓	✓	✓	✓	✓
Mystery of the Tower, the	✓	✓	✓	✓	✗
Number Bracelet Games	✓	✓	✓	✓	✗
Orange Game, the	✓	✓	✓	✓	✓
Predictive Text and Machine Learning (cs inside)	✓	✗	✓	✓	✓
Purse for the Tetradice, the	✓	✗	✓	✓	✓
Radix Sort Game	✓	✗	✓	✓	✓
Roborally	✓	✓	✓	✓	✓
Scheduling (cs inside)	✓	✗	✓	✓	✓
Science Night Activity with Constructed Islands	✓	✗	✓	✓	✓
Security Protocol Game, the	✓	✓	✓	✓	✓
Tetradice	✓	✗	✓	✓	✓
Treasure Hunt Unplugged	✓	✗	✓	✓	✓
Zero Knowledge Games	✓	✗	✓	✓	✓

Tab. 4.3: Evaluation of potential desktop games

Name	Separate	Uncertain	Governed by rules	Make believe	Educational
Cisco Edge Quest	✓	✓	✓	✓	✗
Cisco Mind Share Game	✓	✓	✓	✓	✓
Cisco myPlanNet	✓	✓	✓	✓	✓
CyberCIEGE	✓	✓	✓	✓	✓
Logic Gate Simulator	✓	✓	✗	✓	✓
SimSE	✓	✓	✓	✓	✓
SpaceChem	✓	✓	✓	✓	✗
ToonTalk 3	✓	✓	✓	✓	✓
Wolfram binary card game	✓	✗	✓	✓	✓

Tab. 4.4: Evaluation of potential mobile games

Name	Separate	Uncertain	Governed by rules	Make believe	Educational
Binary Game	✓	✓	✓	✓	✓
Binary Maglock	✓	✓	✓	✓	✓
Cargo Bot	✓	✓	✓	✓	✓
Cross Binary	✓	✓	✓	✓	✓

Tab. 4.5: Unavailable activities

Name	Separate	Uncertain	Governed by rules	Make believe	Educational
Age of Computers	?	?	?	?	?
Binary Madness	?	?	?	?	?
Code and Conquer	?	?	?	?	?
Code Hero	?	?	?	?	?
EleMental: The Recurrence	?	?	?	?	?
Gidget	?	?	?	?	?
Robot Turtles	?	?	?	?	?
Space Invaders (cs4fun)	?	?	?	?	?
Tsubaki	?	?	?	?	?
Wu's Castle	?	?	?	?	?
Zipping it up (cs inside)	?	?	?	?	?

Fig. 4.3: Cisco Edge Quest 2

5. DEVELOPING CRITERIA FOR REVIEWING EDUCATIONAL GAMES THAT TEACH COMPUTER SCIENCE

This chapter develops criteria that will be used to review each game. The purpose of the review is to describe in detail what resources are available for teaching computer science using educational games. This chapter will highlight what details are important and how they are going to be discussed in the following chapter.

When it comes to using an educational game in a learning environment such as a classroom, there are two important factors that are likely to be considered: availability and learning. These two factors are important simply because in order for learning to happen an activity must be accessible to students and the activity must encourage learning. Learning refers to what players learn from playing the game, while availability refers to how easily the game can be used to teach. Both learning and availability bring up several key questions that need to be addressed. Learning can be broken down into three key questions. What topic does the game teach? How well does the game teach its topic? How does the game teach its topic? Availability can also be broken down into three key questions. How easy is the game to run? How much does the game cost financially? How much time does the game require? The review will address each of these six questions for every game. This chapter will describe how each of these questions can be answered.

To ensure the six key questions will be addressed in a meaningful way a set of criteria for answering each question has been developed. Availability will be discussed without making any claims about how easy it is to deploy a game into a learning environment. This is because the practicality of using a game will depend on the situation in which it is being used. Some schools, for example, may have more money to spend. Learning will be discussed with respect to established sources. In the case of which topic is being taught this means referring to an accepted list of Computer Science topics. For the other two key learning questions this means referring to peer-reviewed research. Below is a discussion of how each question in the criteria will be answered.

5.1 Criteria for evaluation of educational games that teach Computer Science

5.1.1 What topic does it teach?

The ACM/IEEE curricula are widely used educational benchmark, and many curricula around the world are based on this one. The current active ACM/IEEE computer science curriculum was released in 2008, but the 2013 revision[30][1], which is being finalized at the time of writing, will be used to identify which topics belong to computer science. Each game will be compared to these guidelines to classify which topic(s) they teach.

5.1.2 How well does it teach its topic?

A review of education literature has found several key factors to consider when designing educational games that are effective for teaching. These key considerations can be used to give an indication of how effective a game maybe for teaching. These considerations are that:

[1] http://ai.stanford.edu/users/sahami/CS2013

- An educational game should teach actively rather than passively.
- Players learn best when they are given freedom to play as much as they need.
- Learning suffers from diminishing returns the longer the game is played.
- Learning is maximized when the game is followed with a debriefing
- Increasing motivation should be the key goal, as providing feedback has a positive impact on motivation.
- An educational game should aim to achieve 'flow'
- An educational game should allow the player to build confidence in a risk free environment.

An educational game should teach actively rather than passively. It has been proposed that for one to learn they must be actively engaged with the content [4] [22]. This proposal has been backed up by the work of Traci Sitzmann [34]. Sitzmann found in a meta-analytic review that games that teach their content actively instead of passively achieve better learning outcomes [34]. To teach actively through an educational game means to have the game-play teach the content. Teaching passively means having the learning in the game separate from the game-play, for example, instructing the content by explaining it in text before or during the level.

Players learn best when they are given freedom to play as much as they need. Sitzmann [34] cites Garris, Ahlers and Driskell [12] and Malone [25] that games are intrinsically motivating and that such activities are crucial for deep learning. Sitzmann [34] uses this as support for their hypothesis that people learn more from an educational game when they have unlimited access

to it. Sitzmann [34] goes on to support their hypothesis with empirical evidence from a meta-analytic review of related research.

Learning suffers from diminishing returns the longer the game is played. Dekkers and Donatti [6] (as cited in Garris, Ahlers and Driskell [12]) found that the more an educational game is used, the less time efficient the learning becomes. This seems to clash with the previous point that players learn best when they are given freedom to play as much as they need. It seems to follow that while players learn the most from being given unlimited time to play a game, learning takes disproportionately more time the longer the game is played. This suggests that having limited in-class time to play an educational game but supporting it by making the game available outside of class may be an efficient strategy.

Learning is maximized when the game is followed with a debriefing. Garris, Ahlers and Driskell (2002)[12], Lee and June (1999)[24], Hays and Robert (2005)[19] all recommend using educational games alongside of, or embedded in, other learning programmes. Sitzmann [34] supports these recommendations with empirical evidence from their meta-analytic review of related research.

Increasing motivation should be the key goal, as providing feedback has a positive impact on motivation. Garris, Ahlers and Driskell (2002) [12] claim that 'The "holy grail" for training professionals is to harness the motivational properties of computer games to enhance learning and accomplish instructional objectives'. This claim would imply that increasing motivational factors should be the key goal in educational game design. One such method for increasing the motivational factors of an educational game is providing feedback. Garris, Ahlers and Driskell (2002) [12] state that feedback has a positive impact on motivation and in the case of educational games has a positive impact on performance.

An educational game should aim to achieve 'flow'. Both Cordova and Lepper and Parker and Lepper (as cited in Garris, Ahlers and Driskell [12]) found that one of the desired motivational learner outcomes is engagement. Csikszentmihalyi (as cited in Garris, Ahlers and Driskell [12]) called the state of being fully engaged 'flow'. It seems to follow that an educational game should aim to achieve flow. Garris, Ahlers and Driskell (2002) [12] list a set of educational game design features that help achieve flow:

- Matching the level of skill required to the player's skill level
- Clear goals and feedback
- Encouraging intensely focused concentration
- Giving the player a high degree of control

An educational game should allow the player to build confidence in a risk free environment. Building confidence helps trainees cope with problems that arise in a real world environment [1]. This means that building confidence is useful, especially, as suggested in Driskell and Johnston (as cited in Garris, Ahlers and Driskell [12]), for complex, stressful or dangerous tasks. It seems to follow that an educational game should aim to allow players to build confidence in a risk free environment, this can be achieved by gradually increasing the difficulty level to match player's growing skill [12].

5.1.3 How does it teach its topic?

There are three types of learning outcomes a game could focus on: Cognitive, Skill-based and Affective [1]. Garris, Ahlers and Driskell [12] provide a simplified description of each of these types of learning. Skill-based learning is described as learning 'Performance of technical or motor skills'. Affective learning is described as learning 'Beliefs or attitudes regarding an object or

activity'. Cognitive learning is split into three subcategories: declarative, procedural and strategic. Declarative knowledge is described as 'Knowledge of the facts and data required for task performance'. Procedural knowledge is described as 'Knowledge about how to perform a task'. Strategic knowledge is described as the 'Ability to apply rules and strategies to general, distal, or novel cases'. Bandura and Wood [1] investigated cognitive, skill-based and affective learning. They also broke each type down into several categories. Cognitive learning was broken down the same categories as above. Skill-based learning was broken down into automaticity (automaticity is a point where a player has learned a skill well enough to simultaneously utilize it and perform other tasks) and compilation (compilation is a stage prior to automaticity where a learner can complete a task fast and with few errors). Affective learning was broken down into attitudinal and motivational. Bandura and Wood [1] also discussed a few ways in which each of these categories have been successfully tested for in studies. Verbal knowledge can be tested for using recognition and recall tests, power tests or speed tests. Knowledge organization can be tested for using free sorts and structural organization. Cognitive strategies can be tested for using probed protocol analysis, self-report measures or readiness for testing measures. Compilation can be tested for using targeted behavior observation, hands-on testing or structured situational interviews. Automaticity can be tested for using secondary task performance measures, interference problems or embedded measurement. Attitudinal can be tested for using self-report measures. Motivational can be tested for using self-report measures, free recall tests (a test that asks the participant to memorize and recall some information from memory) or free sort measures.

Educational games will often want to reward players for improving their understanding of its content. This can be done by employing similar, if not

the same, tests to find out if the player is learning the content in order to give rewards for successful learning. Such tests are likely to be found as part of the game-play of the game rather than explicit tests. For this reason each game will be investigated looking for similarities to these tests within the game-play of the game. If a enough elements of a test are present we can say the game employs the appropriate kind of learning. In addition if the game has the test as a part of it, but separate to the game-play we can make the same conclusion. However, we must first look at the tests and describe how they might exist in an educational game:

- Recognition and recall tests (including free recall): These should be fairly easy to find in games. It is likely that a game will simply present the recognition and recall tests as challenges in game. One should look for multi choice, true/false and free recall based challenges.

- Power tests: Again these should be fairly easy to find in games. One should look for challenges with no time limit.

- Speed tests: These are likely to be both easy to find and prevalent in games. One should look for challenges with a time limit.

- Free sort and structural organization: This is an unlikely game-play element, however it should be easy to spot if it exists. If the game is asking you to arrange elements by how they are related, then the game is probably making use of this test.

- Probed protocol analysis/Structured situational interviews: This is an unlikely game-play element. If it did exist in a game it would be likely the game-play would not revolve around the player directly controlling the outcome. Instead the player would teach something else how to do it, or describe the steps of how it is done in some way.

- Self-report: This is much the same as probed protocol analysis, and it is a very unlikely game-play element. However this is potentially even harder to translate into game-play. It's possible a game could do something such as asking the player to predict how well they will do before they play in order to get an indication of how well the player thinks they are doing. The player could gain some small bonus if their predictions are correct, motivating them to try and answer accurately.

- Readiness for testing: It's possible this could exist in the form of letting the player decide when they are ready for an important 'test', such as a 'boss' (a type of challenge often used in video games wherein the final obstacle of a stage is a powerful enemy that must be defeated) of a level. Alternatively it could work by getting the player to predict how well they would do against the boss.

- Targeted behavior observation: Look for a game that gives out rewards based on how much you have improved over time. Achievements are a common way in which this is used. Such games are likely to be using targeted behavior observation.

- Hands-on testing: This could be achieved through tracking what the player is doing in relation to a set of steps that lead to a solution. Players that have achieved compilation may skip steps that are typically taken towards the goal, yet still achieve a solution. A player could even be rewarded for skipping steps or penalized for taking unnecessary ones.

- Secondary task performance: Look for a game with multiple objectives-primary objectives that teach the subject and give the most rewards and secondary distraction objectives that give fewer rewards that need to be completed simultaneously. Alternatively the reward sizes could be re-

versed so that the player is primarily rewarded for succeeding at secondary task performance. This approach would require that the main objective has some other way to ensure it really is the main objective. For example, one could have the player win only if they complete the main objective.

- Interference problems: The idea is to produce a different type of problem that looks like a normal problem. The goal is to see if the new type of problem affects the player's performance. A game implementing this approach may use such interference problems with harsh penalties for failure. This would mean the player is only likely to succeed once they can solve interferences problems just as well as normal problems.

- Embedded measurement: A game would need to challenge the player in a variety of ways and would probably reward them only for succeedingin most or all of the challenges.

Games that do not have similarities to these tests can still be said to employ particular types of learning. If a game seems to target a particular type of learning without using similarities to one of these kinds of tests, the type of learning will be stated and evidence will be provided supporting the claim.

5.1.4 Availability

There are several factors that affect availability: **How easy is it to run?** The ease of 'running' a game could refer to a range of things from installing software to setting up a game in a classroom. Each game will be placed into one of four categories: web games, game software, mobile games or unplugged games. These categories describe what is required to use the resource. However, the categories make no claim on the practicality of it as a resource, as this will vary depending on the situation in which they are being used. Web games are game software that can be played through a web browser on a typical desktop or

laptop computer without requiring any up front software installation. A good example of a web game would be a Flash based game. Web games require very little set up time; typically all that is needed is a computer, Internet connection, web browser and URL. Desktop games are, like web games, game software that can be run on a typical desktop or laptop computer. However, unlike web games, desktop games do require up-front software installation. For example, a desktop game could be a downloadable executable or a browser based game that requires a non-standard plugin. Desktop games require more set up time than a Web game, but can be larger in size since they no longer have to be downloaded in their entirety each time the game is loaded. Mobile games are game software that can be run on a mobile operating system such as an iPhone, iPad or Android device. Mobile games can potentially avoid a different kind of set up time, that of getting students on a computer. Mobile games can also be played (i.e. learning can happen) at otherwise unusable times, such as while traveling. However, mobile games do require the students having access to mobile devices. Installing applications on mobile devices is typically fairly fast and easy. Unplugged games are games that take place away from a computer. Typically with a group of people performing a physical activity together, or playing a board game. Unplugged games may not require any sort of computer hardware and may often be able to be used in a class room rather than having to go to a computer lab. Additionally, unplugged games often gives students a chance to move around or interact with each other. However, unplugged games often require physical components and depending on which components are required these games could range from the least complex to set up to the most. Any additional information on what's required to run the game will also be provided, especially in the case of unplugged games. This additional information is likely to include things like how many people the activity is

aimed at or how much space it requires.

How much does it cost? Financial cost could potentially be a significant issue. It is for this reason each game's financial cost will be described in several ways. However, this description makes no claim on the practicality of using the resources, such as how much cost is too much, as this will vary depending on the situation in which they are being used. The cost of each game will be described in terms of upfront cost, ongoing costs, micro-transactions and donations. An upfront cost is a necessary cost required to gain access to the software initially. Ongoing costs are necessary costs required to keep access to the software. Micro-transactions are optional costs required to unlock part of the game. Micro-transactions are only optional in the sense that one may choose not to unlock parts of the game. Donations are an entirely optional cost that has no impact on the game.

How much time investment does it require? Time investment will be described by putting the game into one of five estimated time frames:

- Less than 10 minutes
- Less than 30 minutes
- Less than an hour
- Less than two hours
- More than two hours

These time frames are not intended to suggest when the players would have 'finished' or become bored of the games, but they represent where the best value is obtained. Players should be allowed access outside of class and be encouraged to play the games more to learn everything they have to offer. These times are estimates, decided using the author's own experience playing the games. They are not thoroughly tested.

5.2 Criteria format

The final part of this section will describe the format that will be used to describe each game in the review. The format is derived from the discussion of criteria in the earlier part of this chapter and will be used as a template for the review of each game.

What topic does it teach? This question will be answered by categorizing any topics the game covers using the ACM curriculum guidelines [30][2].

How well does it teach its topic? Each of the following points about educational game design will be addressed in the review by the question that follows the point.

- An educational game should teach actively rather than passively: Does the game teach actively or passively?
- Players learn best when they are given freedom to play as much as they need: Does the game itself limit how much it can be played?
- The rate of learning decreases the longer the game is played: As with the previous point, does the game itself limit how much it can be played? Additionally, how closely is the game-play tied to the content? Will learning stop much before the game is finished?
- Learning is maximized when the game is followed with a debriefing: Does the game include some sort of debriefing or discussion that covers how the game relates to subject matter?
- Increasing motivation should be the key goal and providing feedback has a positive impact on motivation: How much, if any, feedback is provided?
- An educational game should aim to achieve 'flow':

[2] `http://ai.stanford.edu/users/sahami/CS2013`

  - Matching the level of skill required to the player's skill level: Does the game itself match difficulty to level of skill? Does the game offer difficulty settings so the player can do this themselves?
  - Clear goals and feedback: Are the goals clear? Feedback has already been addressed (see increasing motivation above).
  - Encouraging intensely focused concentration: How much does the game encourage focused concentration?
  - Giving the player a high degree of control: How much control does the player have in the game?

- An educational game should allow the player to build confidence in a risk-free environment: Does the game's difficulty level gradually increase?

Note that there are two points about how a game's difficulty increases. The first aims to address if the game's difficulty matches the player's skill throughout the game while the second aims to address if the difficulty level is likely to help the player build confidence. The level of difficulty in a game and how it increases mean slightly different things for each of these questions. A game that starts with a low difficulty level and increases it slowly might not ever challenge players with higher skill levels, but will help encourage confidence in players with lower skill levels.

How does it teach its topic? Which types of learning does the game target:

- Cognitive:
  - Verbal knowledge: Does the game include recognition and recall tests, power tests or speed tests?
  - Knowledge organization: Does the game include free sorts or structural organization?

  - Cognitive strategies: Does the game include probed protocol analysis, self-report measures or readiness for testing measures?

- Skill-based:

  - Compilation: Does the game include targeted behavior observation, hands-on testing or structured situational interviews?
  - Automaticity: Does the game include secondary task performance measures, interference problems or embedded measurement?

- Affective:

  - Attitudinal: Does the game include self-report measures?
  - Motivational: Does the game include self-report measures, free recall tests or free sorts?

In the case of self-report and other similar measures, the measures must be related to the type of learning. For example, a game including self-report measures is not necessarily testing for cognitive strategies, attitudinal changes and motivational changes. Such a game would also to be using self-report measures to ask the player about their cognitive strategies, attitudes or motivation levels in order to fulfill the criteria.

What each of these tests may look like in a game is described earlier in this section. However a game may not use any of the above tests but still appears to be using a particular type of learning. In such cases evidence will be provided for the game using that type of learning.

How easy is it to run? Which category does the game fall under: web games, desktop games, mobile games or unplugged games? Additional information will also be provided as necessary.

How much does it cost:

- What is the game's up front cost?
- What are the game's on going costs?
- What micro-transactions does the game support?
- Does the game suggest a donation?

How much time investment does it require? What time category does the game fall under: less than 10 minutes, less than 30 minutes, less than an hour, less than two hours or more than two hours?

Now that we have a criteria for reviewing games, each game can be investigated in detail, this investigation appears in appendix B providing a summary of it's findings and the investigation itself is available online[3]. From the next chapter the thesis will move on to future work and later discuss findings.

[3] `http://tinyurl.com/coscgames`

6. GUIDELINES

This chapter provides a set of guidelines based on the criteria developed in chapter 5, and in the next chapter the guidelines are applied to create a new game. The guidelines are presented as a kind of checklist to consider in the design of a game.

6.1 Guideline 1: What do you want the game to teach?

Will the concept(s) you want to teach have a significant amount of player innovation? If the only innovation is learning the concept, the game is likely to have limited replayability/longevity. This will clash with some of the guidelines below. This can be mitigated to a certain extent by introducing game elements unrelated to the concepts being taught. However this also clashes with other guidelines, such as keeping the educational content relevant.

6.2 Guideline 2: What type of learning does the game target?

There are three primary types of learning a game could target. Before designing your game you should decide which of the following types of learning you are trying to teach.

- Skill based: improves performance/teaches technical or motor skills.
- Affective: learning beliefs or attitudes towards an object or activity.
- Cognitive: learning declarative, procedural or strategic knowledge.

  - Declarative knowledge: Knowledge of the facts and data required for task performance
  - Procedural knowledge: Knowledge about how to perform a task
  - Strategic knowledge: Ability to apply rules and strategies to general, distal, or novel cases

6.3 Guideline 3: How does the game measure successful learning?

Depending on the type(s) of learning the game is targeting there are several tests that measure if the player is actually learning the content. These tests can be included as gameplay mechanisms for testing player success. In some cases these same mechanisms can be used to teach the concept, although this is not always practical. Some of these tests are not particularly practical for use in educational games however. For this reason, only those that are practical have been listed below.

Cognitive:

- Verbal knowledge:
  - Recognition and recall tests: Present the player with multi-choice, true/false or free recall based challenges.
  - Power tests: Present the player with questions or question-like challenges with no time limit.
  - Speed tests: Present the player with questions or question-like challenges with a time limit.
- Knowledge organization:
  - Free sorts or structural organization: Present the player with a challenge that asks them to arrange elements by how they are related.

- Cognitive strategies:
  - Probed protocol analysis: Do not do a direct simulation of what the player is being trained for. Instead have the player teach something else how to do it, or describe the steps of how it is done in some way as a challenge in the game.
  - Self report: Ask the player how well they think they will do on a given part of the game before they play it. Give them some small bonus if their predictions are correct. This will motivate them to predict how well they are doing in such a way that the game can measure it.
  - Readiness for testing: Let the player choose when to take some important in-game 'test'. A good example would be something like increasing the difficulty level or choosing to battle a 'boss'. Alternatively you could ask the player how well they think they would do against a 'boss', much like with self report.

Skill based:

- Compilation:
  - Targeted behavior observation: Give out rewards based on how much the player has improved over time. For example, one could use something like achievements.
  - Hands-on testing: Tracking what the player is doing in relation to a set of steps that lead to a solution. Players that have achieved compilation may skip certain steps and still achieve a solution. A player could be rewarded for skipping steps or punished for taking unnecessary ones.
  - Structured situational interviews: As explained earlier.

- Automaticity:
  - Secondary task performance: As with hands-on testing, use primary and secondary objectives. However, with secondary task performance the key thing is that both objectives must be completed simultaneously. The primary objective should be necessary to survive/complete the level. However, the secondary objective should grant the most points. The primary objective should be the main focus of the learning and the secondary objective a distraction. In this way the player must master the focus of the learning so they can get the most points.
  - Interference problems: Provide the player with a set of challenges and amongst the normal challenges hide interference challenges. These are challenges that look like normal challenges, but are different in some way. The goal is to see if the player acknowledges the interference problem and handles it easily. As such, failing an interference challenge should come with a harsh penalty. This will mean the most successful players are the ones that deal with interference problems the best.
  - Embedded measurement: Challenge the player in a variety of ways and reward them for uniform success.

Affective:

- Attitudinal:
  - Self-report: As explained earlier.
- Motivational:
  - Self-report: As explained earlier.

  - Free recall: As explained earlier.
  - Free sorts: As explained earlier.

6.4 Guideline 4: How does the game teach its topic well?

- An educational game should teach actively rather than passively.
- Players learn best when they are given freedom to play as much as they need. The game itself should not limit how much it can be played. Some games will have difficulty achieving the required replayability without making other sacrifices, as noted earlier.
- Learning decreases the longer the game is played. The educational content should be tied closely to the gameplay. An educational game should be relatively brief and to the point. The more content made for a game the less effective that content will be for teaching, unless the content being introduced is new educational content.
- Learning is maximized when the game is followed with a debriefing. This does not and probably shouldn't mean that the game itself includes the debriefing. It is best to keep the game as active as possible and leave the debriefing for the instructor or at least keep it separate.
- Increasing motivation should be the key goal, and providing feedback has a positive impact on motivation. Provide feedback with the goal of having a positive impact on motivation.
- An educational game should aim to achieve 'flow':
  - Matching the level of skill required to the players skill level: Either have the game itself match difficulty to level of skill (this first option

is preferable) or offer difficulty settings so the player can do this themselves.

- Clear goals and feedback: Provide clear goals. Feedback has already been addressed.
- Encouraging intensely focused concentration: It could be a good idea to reward players for such concentration, or perhaps penalize them for the lack of it.
- Giving the player a high degree of control. However, this should not trivialize the game.

- An educational game should allow the player to build confidence in a risk-free environment. This should be achieved by gradually increasing the game's difficulty level.

6.5 Guideline 5: Availability

6.5.1 Platform

Consider the audience of your game when approaching availability. When designing a game for a particular platform you must consider if your audience will have easy access to the platform and how that could be achieved when designing for a particular platform. It may be no coincidence that most computer science games are web games—web games are likely to be playable on most computers without requiring an initial set up. Unplugged games may also be desirable because they do not require access to a computer at all, although they do often require other game components that may not necessarily be available to the audience. Mobile games require access to an appropriate mobile device, which are less common than computers and desktop games, and typically involve setup times, which can stack up when a teacher has to install the game on multiple

devices.

6.5.2 Cost

Financial costs should be considered when designing your game. As discussed later in chapter 8 (discussion), most games reviewed had little to no cost. This enables these games to reach a wider audience. However sometimes this may not be a priority as some audiences are more willing to spend money than others. Additionally, some particularly sophisticated games might require higher levels of funding, which means they are more likely to benefit from charging a higher amount, in order to cover costs, or so they can promise profits in order to encourage funding from investors or publishers. Other sources of funding such as kickstarter could be a good idea, as we have seen with *Code Hero* [1] 6.1.

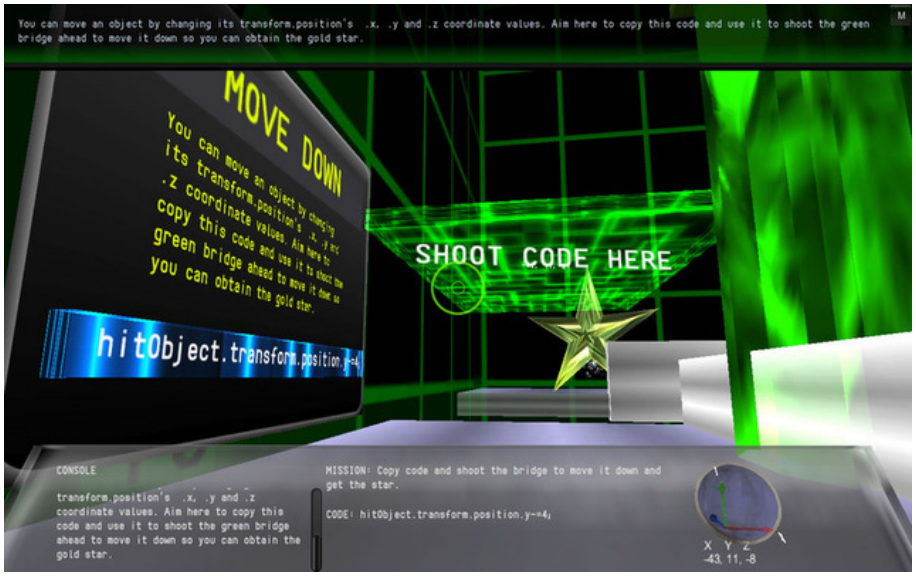

Fig. 6.1: Code Hero

Time is a very important consideration when designing an educational game. One must consider how long the audience is likely to be willing to dedicate

[1] `http://primerlabs.com/codehero`

to learn the content at hand. This may be why shorter games tend to be produced more than longer games, although the possibility that shorter games are simply easier to produce should not be ignored. These guidelines have already suggested providing a high amount of content for the player, however this point is independent from how much time invest the player must give in an important way. The player should be given as much access to the game as they desire, but this can be achieved by making the game available to the player whenever they want it, again making web based games desirable. If the player is given sufficient access to the game then the main consideration when it comes to time investment is how long it takes for the player to benefit from learning the main topic, any future playing is likely to be reinforcement.

7. CASE STUDY

In this chapter we use the guidelines presented in the previous chapter to develop a game to teach the concept of error correction by teaching parity on a RAID 5 system called *the RAID Arrays game*7.1 7.2 7.3 7.4.

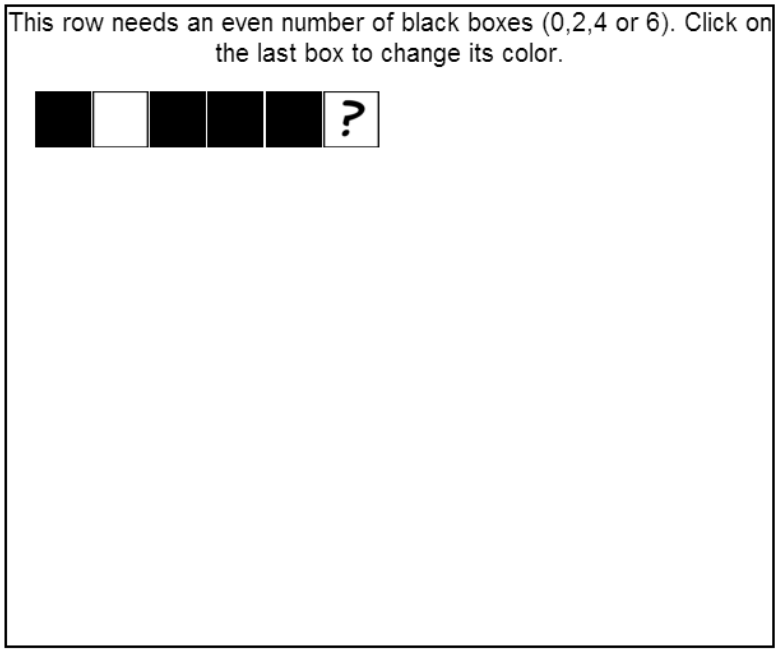

Fig. 7.1: First introductory stage of *the RAID Arrays game*

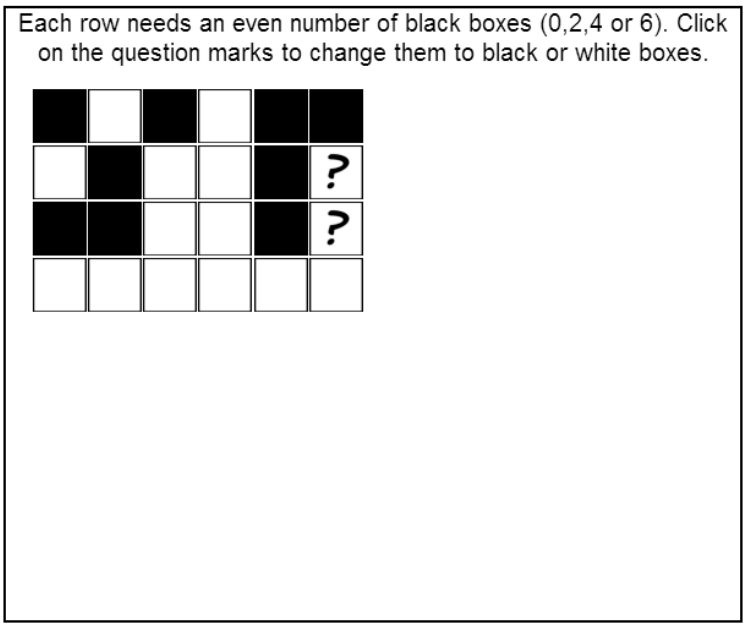

Fig. 7.2: Second introductory stage of *the RAID Arrays game*

7.1 Guideline 1: What do you want the game to teach?

The key idea behind the game is as long as a set of hard disk drives (HDD) keeps their data even among rows (even number of 0s and 1s) you can restore the data if one of the HDDs is lost. This is the kind of concept that doesn't require a large time investment to learn, which means the player will hopefully learn this concept quickly. After this point the game could have either ended, giving the game a limited amount of longevity, or continue without teaching any more on this topic. There is a clash in the guidelines here, and for this game I have chosen to provide artificial' longevity. This is because teachers can limit in-class time spent on the game, removing the downside of the artificial longevity; however the reverse can not be easily achieved.

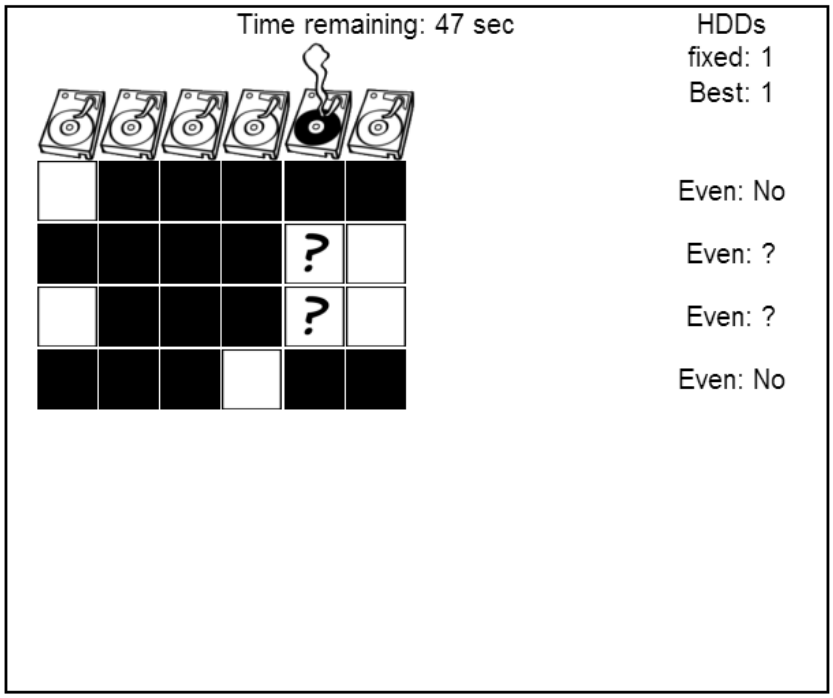

Fig. 7.3: Third introductory stage of *the RAID Arrays game*

7.2 Guideline 2: What type of learning does the game target?

There are three primary types of learning a game could target. Before designing your game you should decide which types of learning you are trying to teach. The topic of *the RAID Arrays game* is declarative knowledge, specifically that as long as a set of HDDs keeps the number of bits even among rows you can restore the data if one of the HDDs is lost. The game will also teach procedural knowledge to a certain extent, by teaching the player how this is achieved, although this is not a key learning objective of the game, merely a necessary step along the way. *The RAID Arrays game* also employs skill based learning, as a means of improving the longevity of the game. This is achieved by presenting the player with randomized puzzles and asking them to solve them within a time limit.

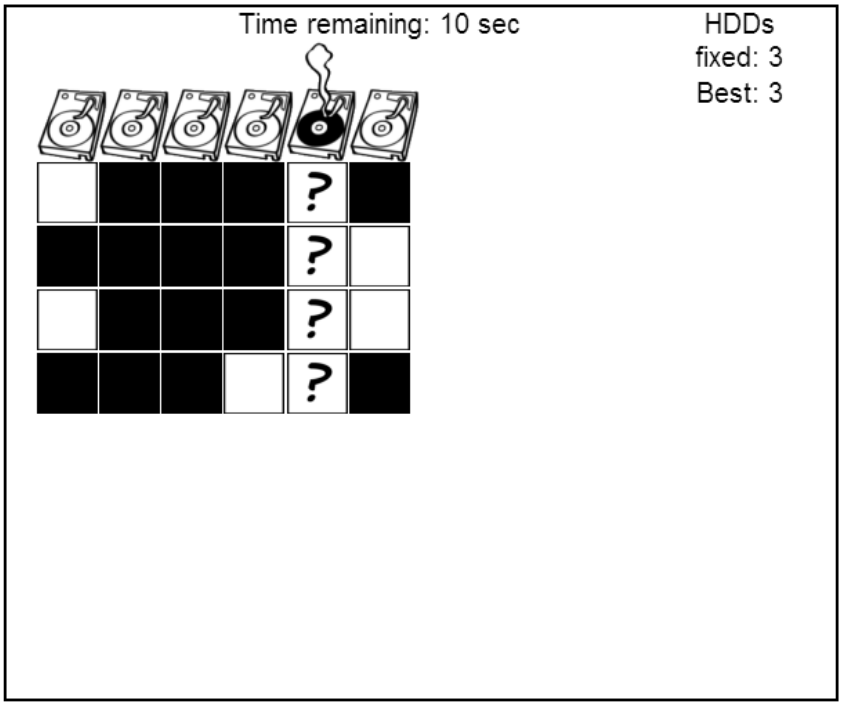

Fig. 7.4: The RAID Arrays game proper

Essentially the player is developing the skill of seeing if a row of squares has an even number of a given color in it.

7.3 Guideline 3: How does the game measure successful learning?

The RAID Arrays game uses both power tests and speed tests by asking the player to solve puzzles both under time limits and unrestrained. However, the primary thing being tested is their ability to solve the puzzles, which indirectly teaches the player about the key topic. Essentially the speed tests do the teaching of the topics over time as the player does several of these tests, although the player should fairly quickly be able to see that they will be able to solve any of these tests (which is the point).

7.4 Guideline 4: How does the game teach its topic well?

- *An educational game should teach actively rather than passively.*
  - *The RAID Arrays game* teaches actively by presenting the player with a series of puzzles designed to show the player that given a grid (representing a RAID Array) with an even number of black squares and white squares (an abstraction from 0s and 1s) they can always reconstruct the pattern if one of the columns (HDDs) is lost. The only passive teaching presented is that hard drives are mention in text by name as the player leaves the tutorial.
- *Players learn best when they are given freedom to play as much as they need. The game itself should not limit how much it can be played. Some games will have difficulty achieving the required replayability without making other sacrifices, as noted earlier.*
  - *The RAID Arrays game* achieves this by giving the player arbitrary puzzles to solve in a time limit. The player can continue to play the game after they have learned the key idea by continuing to practise their skill at identifying (and correcting) odd rows in a grid.
- *Learning decreases the longer the game is played. The educational content should be tied closely to the gameplay. An educational game should be relatively brief and to the point.*
  - *The RAID Arrays game* keeps the gameplay closely tied to the content for the initial few levels. However, by the time the player has learned how to play the game properly there is little or no educational value left for them in the game. It is advised that if this game is used in class it should be used only be used very briefly before a

debriefing to discuss how it relates to RAID Arrays.

- *Learning is maximized when the game is followed with a debriefing. This does not and probably shouldn't mean that the game itself includes the debriefing. It is best to keep the game as active as possible and leave the debriefing for the instructor or at least keep it separate.*
  - *the RAID Arrays game* includes small snippets of text that explain that the game has an educational context, without actually fulfilling the role of debriefing themselves.
- *Increasing motivation should be the key goal, and providing feedback has a positive impact on motivation. Provide feedback with the goal of having a positive impact on motivation.*
  - The player gets visual feedback in the form of seeing squares they click on alternate between states (black and white, equivalent to 1 and 0). Additionally the player is provided some visual feedback when they solve a hard drive as the one they are working on changes its icon to a normal hard drive and another one breaks, indicated by a discolored disk and smoke rising from the HDD as well as all the 'bits' for the column changing to the unknown. A record of how many HDDs the player has fixed is kept as well; the intent of this is act as a scoring system, motivating the player to play again to better their score.
- *An educational game should aim to achieve 'flow':*
  - *Matching the level of skill required to the players skill level: Either have the game itself match difficulty to level of skill (this first option is preferable) or offer difficulty settings so the player can do this themselves.*

    * The game's difficulty rests on a bar the players set for themselves. The only kind of progress the player can make in the game, beyond leaving the tutorial, is beating their own high score. Each time the player improves enough at the game to raise their high score they will set the bar higher for next time.

- *Clear goals and feedback: Provide clear goals. Feedback has already been addressed.*

    * The goal of the game is made clear through a minimal tutorial that appears more like levels in the game, although it is not impossible to make it throughout the tutorial without having learned how to play. Once through the tutorial, scaffolding is provided to further help the player realize the goal of the game. The goal of the game remains consistent all throughout.

- *Encouraging intensely focused concentration: It could be a good idea to reward players for such concentration, or perhaps penalize them for the lack of it.*

    * It is possible to play the game without focused concentration, however the player is very likely to get a much lower score if they play this way as it there are a prohibitively large number of possible combinations that could be correct.

- *Giving the player a high degree of control. However, this should not trivialize the game.*

    * The player is afforded the ability to change the state of any of the unknown 'bits' at their leisure. However any further control (such as allowing them to change the state of other 'bits' was not allowed so that the goal of the game was not obfuscated.

- *An educational game should allow the player to build confidence in a risk-free environment. This should be achieved by gradually increasing the game's difficulty level.*
  - The two introductory levels to help achieve this for *the RAID Arrays game*, each of which provides an unlimited amount of time to solve the problem, as well as simplified problems (in the case of the first introductory level). Additionally, the game provides scaffolding for the player when they leave the tutorial by telling the player which rows are currently even. Further still, the game's difficulty is generally well matched to the player's skill level as the player is always playing against their own score (as described earlier).

7.5 Guideline 5: Availability

7.5.1 Platform

Consider the audience of your game when approaching availability. When designing a game for a particular platform you must consider if your audience will have easy access to the platform and how that could be achieved when designing for a particular platform.

The RAID Arrays game has been designed for use as a web game, although it could be easily be adapted to be a mobile or desktop game. A web game was picked so that the game can be played quickly and easily by anyone at a computer. Further *the RAID Arrays game* was developed using Crafty[1] a Javascript engine that is compatible with modern browsers (including tablets) as well as some of the older versions of Internet Explorer. Using Crafty broadens the reach of the game and ensures it is as easy as possible for the game to be put to use in schools or at home.

[1] `http://craftyjs.com/`

7.5.2 Cost

Financial costs should be considered when designing your game.

The RAID Arrays game was not an expensive game to develop—at the time of writing the game has not required any funding. This has allowed the freedom to release *the RAID Arrays game* for free so it will be accessible to anyone with an appropriate platform and Internet access. *The RAID Arrays game* is being included as part of the open source "CS field guide"[2].

Time is a very important consideration when designing an educational game.

The RAID Arrays game only requires a short amount of time to establish its main point, however the game was designed such that it can be played much longer than the required time, in order to reinforce the learning. In order to lengthen *the RAID Arrays game* in any meaningful way it would be best to introduce new educational content. There is more that could be taught on the topic of RAID Arrays, although teaching them within the context of the same game might be challenging. One potential solution could be to produce multiple short games each covering a similar amount to *the RAID Arrays game*, allowing for the games to be used together or separately depending on which suits the user best.

7.6 Teacher's feedback

The RAID Arrays game was made available in the error correction section of the "CS field guide" and shown to a small class for a high level education course about teaching computer science at the University of Canterbury. A short survey of five questions was also provided:

1. Do you think your students could play this game without prior instructions?

[2] `csfieldguide.org.nz`

2. Do you think your students could learn from this game, if accompanied by an appropriate discussion?

3. Do you think your students would find this game motivating?

4. Do you (did you?) think this game is easily accessible, or would you prefer a different format, such as an unplugged game?

5. This game teaches a bit about RAID Arrays, however the gameplay may outlive the educational content. Do you think the game's replayability should be artificially shortened?

There were only four responses, some of which were incomplete. As such this survey has not yielded any statistically significant results, but the responses garnered were favorable towards the design, and provided useful specific feedback for improving the game.

8. DISCUSSION

In this section the main lessons learned are drawn together. We look at the existing games that were identified, and patterns in their suitability for use in Computer Science education.

8.1 Games currently available

More than 80 potential games were found, and less than half of them met the criteria for being an educational game that teaches computer science from chapter 4 (Compiling a list of educational games that teach Computer Science). While this might only be a modest number of games, it shows that there is at least some interest in teaching computer science using educational games. Information was found about 11 games that are not currently available, but because these games were not available, they could not be compared to the criteria or reviewed.

8.2 Topics covered

Only 7 of the 18 knowledge areas from the ACM curriculum guide were covered by educational games. Additionally, only a few topics from each knowledge area were covered. Most of these topics covered had some similarities in that they were all fairly easy to represent with abstractions. Only a few of the more involved games relied on simulations to teach. This suggests perhaps unsurprisingly that it is easier to create a game based on abstractions than to

create a simulation. However, it is possible that these types of topics are more popular not because they are easier to create games for, but because of some other reason. For example, educators may be more interested in educational games that cover subjects that students tend to have more difficulty learning which could motivate people to create games to help teach these topics.

Two topics in particular were well covered, put in their simplest form these were introductory programming concepts and binary to decimal conversion. The majority of games taught either programming concepts games and binary games, from this we can conclude one of two things—either that these topics are in some way easier to develop for, or that these topics are in some way more popular to develop for. Generally, games that taught these topics had more in common with each other than just the topic they taught. Most of the binary games presented the player with a random decimal number and a list of bits that the player could click on in order to change them between 0 and 1 (see 3.3). The player would often be shown the resulting decimal number so that they could figure out how binary decimal conversion works. In some cases the place value of each bit was also shown. The games that taught introductory programming concepts tended to give the player indirect control of some sort of avatar such as a robot (see B.1). The player would then issue commands to the avatar in order to have it complete some sort of puzzle—perhaps escaping a maze. Often the player was restricted in the number of commands they could give the robot and the type. Sometimes the player would even be allowed to use functions in order to give commands and allowing for the teaching of recursion. The similarities between these games were strong and relatively few varied from this approach. This suggests that it is less likely that the popularity of the topic is what lead to high numbers of games like these, and more likely that these types of games are more interesting to develop. However, it may just be the case that this type

of game is more appealing to the target audience.

8.3 Availability

Most of the games were short, free, and available online using an Internet browser with fairly standard add-ons such as Flash. This may be so that they can reach a wide audience or this maybe one of the easiest kinds of game to develop. Games that are short are likely to require less time to develop, and games that are free are likely to reach a wider audience, as is likely the case with games available through a browser. Games available through a browser are also easy to set up, which can be quite an issue in schools. The second most popular format for a game is an unplugged game—that is, a game that is available away from a computer such as a game using dice or a board game. Unplugged games could be popular for a number of reasons. For some they could be easier to develop since they typically don't require any programming and in some cases they could be easier to deploy because they don't require access to a computer. However, for unplugged games there is also the Computer Science Unplugged project which could be responsible for the high number of games in this category. Computer Science Unplugged is a project that compiled and produced a lot of activities that teach computer science away from a computer. Most of Computer Science Unplugged activities were targeted at a younger audience but still managed to cover an impressive depth of content. It is possible that most games were short not because it is easier to develop shorter games, but rather that it may be in some way preferable to just quickly teach a small subset of topic in a single game. It seems only beneficial that most games were available for free.

8.4 Educational properties

Sitzmann found that games that focused on active teaching were more effective than games that concentrated on passive teaching [34]. Of the 41 games reviewed, 33 utilized mostly active teaching (roughly 80%). This is an encouragingly high number. It seems that it is either natural to develop games in this way, or that most developers already know that active teaching produces better results.

It was found that in some games the amount of educational value decreased over time. Often this was because a game aimed to teach only a single concept. Once the student had learned this concept there was no longer any more educational value to be found from the game. There seemed to be only two ways to get around this problem. One is to teach more than one concept within the game, and the other is to teach a skill that can be practised. Garris, Ahlers and Driskell [12] and Dekkers and Donatti [6] found that the more an educational game is used, the less effective the learning was per unit of time. This is perhaps explained by the results of this review. It was found that 43.9% of games reviewed lost a significant amount of their educational value over time. In fact, only 26.8% of games kept a high level of educational value throughout the entire game. The remaining 29.3% of games were mixed in that they covered more than one idea. In each of these cases one of these ideas was strongly skill-based. The skill-based learning would continue throughout the whole game but any other concepts would not. It is interesting to note that in each of these games the skill-based learning was of less value to computer science than the other concepts the game taught. For example, many of the games that taught binary would teach converting between binary and decimal as a skill—specifically, in the speed of translation or the accuracy of translation. Initially this is beneficial learning for computer science. Computer science students may be required to

learn how to convert binary to decimal or vice versa, but it is not necessary for a computer science student to be able to do this quickly or specifically highly accurately. As such, once the basic concept has been learned, any additional refinement of the skill is unnecessary, rather learning about the patterns found in counting in binary is more valuable.

Sitzmann found that learning is maximized when accompanied by a debriefing or discussion about how the games relate to the topic at hand. 22 of the games reviewed (53.7%) included their own debriefing or discussion. However, such a debriefing or discussion is by necessity passive teaching and therefore perhaps better done outside of the game.

Garris, Ahlers and Driskell [12] state that feedback has a positive impact on motivation, and in the case of educational games, has a positive impact on performance. Encouragingly, only 9 out of 41 (22%) games reviewed in this research gave low amounts of feedback. This suggests that game developers are actively trying to provide high amounts of feedback for players. However, it is unclear if the goal of this feedback is to increase learning or enjoyment.

An educational game should aim to achieve flow. Flow has been described as the sensation of being so intensely focused on something that you lose track of time and other things going on around you. Flow seems like not only a desirable feature of a game, but something an educator would desire to have happen in the classroom with their lesson. It is unclear which of the two potential motivators will be leading towards targeting flow in games but they seem to go in hand in hand fairly well. Games can achieve flow by matching the level of difficulty to that of the player's skill, as well as giving clear goals and feedback, encouraging intensely focused concentration, and giving the player a high degree of control. Only 13 games (31.7%) provided no way for the game's difficulty to be adjusted, whereas 13 games automatically adjusted their difficulty, 12 games

let the players pick the difficulty, and in 3 games the difficulty was dependent on the players themselves because these games were multiplayer games. It was found that 75.6% of games had clear goals, and 85.4% of games had simple goals, and 68.3% had both clear and simple goals, and only 7.3% did not have simple or clear goals. Only 19.5% of games required only a low amount of concentration to play successfully, and only 14.6% of games allowed the player a low amount of control. Collectively we can see that games seem likely to be aiming to achieve flow with roughly 70% or more of games succeeding in each category. However, it is important to note that this does not mean many games succeeded in every category at once, and in fact no games meet all the criteria.

Sitzmann reports that players learn best when they are given freedom to play as much as they need. In most cases this will be dependent on restrictions in the educational environment. However, some 19.5% of games did restrict the amount that they could be played themselves. This was typically done through having only a small amount of content. Some games went so far in the other direction as to offer an unlimited amount of re-playability in that their content was randomly generated. However, realistically players will eventually tire from these games too. Once again, it is possible that the game developers knew that providing re-playability would increase learning benefits. However, it is also possible that re-playability was provided for financial motivations. If a game earned money through advertising, for example, it may earn more if it encourages players to come back and play again.

Our criteria asked for games that used active teaching, retained their educational value throughout the game, provided high amounts of feedback, encouraged flow, and allowed the player to play as much as they needed. For all but one of these categories roughly 70% or more of games complied. It seems that games had trouble keeping their educational content tied to the game play more

than anything else. This could suggest that this is a particularly hard thing to do, or that the motivation of some of these games is less to teach and more for enjoyment. If this were the case the developer may not be so concerned if the educational value was not as long lived. As described in the chapter 7 (Case Study), when developing *the RAID Arrays game* there was a clash between two of the guidelines. These guidelines relate to two of the criteria from the review: longevity and keeping educational content tied to game play. With *the RAID Arrays game* the author found it difficult to have the game teach the concept desired while still having enough gameplay content. The solution I ended up using was to introduce an arbitrary skill (correcting bits within a time limit) that would lengthen the game play but diminish the amount of rate of learning over time. This could to some extent explain what is happening in other games, since a full 10 games (24.4%) appear to have the same diminishing returns of educational value. While using a skill to teach, they start off teaching a computer science concept, but after a while all that is left is a skill that is not particularly useful, as described earlier with the binary games. If we factor this into our consideration of games that are tied to educational content we find that the remaining games that were not strongly tied to the educational content over time make up 64.5% of games. It is important to note that this is 64.5% of the 31 games that remain if we discount all games that were both skill based and had diminishing educational value, otherwise this value becomes 48.8%. Even after factoring this in we are left with a large number of games that do not stay tied to the educational content over time.

8.5 Learning targeted

Games could target one of three different types of knowledge: cognitive, skill-based, or affective. However, affective learning doesn't end up being of much

interest to us. This is because most topics from the ACM curriculum guidelines do not cover affective knowledge. There was a large overlap between games that covered cognitive knowledge and skill-based knowledge. Only 19.5% of the games did not teach at least some fact-based knowledge, that is, cognitive knowledge, whereas 41.5% of games did not cover skill-based knowledge i.e. games are far more likely to cover cognitive knowledge than skill-based knowledge. However, the extent to which the focus is on these kinds of knowledge is not clear. As we saw above, many games that focused on skill-based knowledge which also taught cognitive knowledge, but the learning of the cognitive knowledge diminished over time whereas the skill-based learning remained. Of the games that covered skill-based learning, 83.3% targeted automaticity and 41.7% targeted compilation. This may suggest that the pursuit of automaticity may be more rewarding for players or make for more interesting game play elements. For cognitive knowledge, 63.6% of games taught verbal knowledge, whereas 69.7% of games taught using cognitive strategies, while only one game taught using knowledge organization. This shows—perhaps unsurprisingly—that both cognitive strategies and verbal knowledge were very popular choices for developing games, and that knowledge organization was very rare. It is likely that knowledge organization appeared so rarely because it is difficult to include as a game play mechanic. It is likely that verbal knowledge was common because it is easy to convey in educational games as it is often possible to include verbal knowledge without significant support from game play mechanics. Such verbal knowledge can be implicit in the game through the state of the game, or mentioned in-game via text or through a character. Sometimes a naming convention for a game mechanic or item in game is sufficient. For example labeling the numbers in a binary game as bits teaches the player what they are called. Cognitive strategies, however, require at least some game play

mechanics dedicated towards teaching them. From this it seems that we will see educational games with game mechanics targeting cognitive strategies and automaticity. This could be quite useful information. It may be showing us that targeting cognitive strategies or automaticity is a good technique for making a game more enjoyable or appealing to players, or that such games are easier to make. Another way to look at it is that there are fewer games that don't target automaticity or cognitive strategies. So even though particular topics may have already been covered, they are unlikely to have been covered without teaching automaticity or cognitive strategies. We may find that we want to target compilation or knowledge organization for a particular topic instead.

9. CONCLUSION

9.1 Aim

Prior to this work little research has been done into using educational games to teach computer science. Most of this research has been focused on producing games that teach computer science and evaluating their effectiveness. Relatively few topics are covered a, and while in some cases justification was given for the choice of topic, in none of the investigated research was any attempt made to establish that the game produced was different to any other. In fact, other such educational games were rarely mentioned. It appeared that while research into this field was being done and games that teach computer science certainly existed, there was a need for an overview of the field. Vital information such as, how many games that teach computer science exist, what topics they teach, and what do they have in common appeared to have been left completely unanswered. The key focus of this research has been to fill this gap.

9.2 The review

This research used a systematic approach to identify as many games that teach computer science as possible, as well as gathering key information about each of these games. This search was continued throughout the entirety of the research resulting in, at the time of writing, eighty five potential games. This list was culled by comparing each game to a definition of what a game is, and a definition

of computer science. For the definition of computer science, the ACM curriculum guide 2013 Iron Man draft[1] was used. For the definition of a game, a modified version of Roger Caillois's definition was used; the modification was to remove the non-productive criterion since educational games are necessarily productive. The definition of a game was then that it was fun, free, separate in space and time, uncertain, with some room for innovation on the players behalf, governed by rules and make believe. At times judgment calls had to be made, chapter 4 (Compiling a list of educational games that teach Computer Science) addresses some of these. After this process, we were left with just 41 games that met the criteria. In order to most efficiently gather important data from each game a review criteria was constructed. Each game was then reviewed based upon this criteria.

The goal was gathering as much useful information as possible rather than making value judgments. The review targeted four key pieces of information:

- What topic does the game teach?
- How likely is the game to successfully teach its topic?
- What type of learning the game targets?
- How accessible is the game?

The next part of the criteria did not actually make any claim about how likely a game was to succeed at teaching its topic rather, it lists a series of properties that have been found to encourage learning in educational games. Such properties include teaching actively and aiming to achieve flow. The next section of the criteria looks for certain types of games play mechanics that may suggest the game was targeting a certain type of learning; if a game is targeting a certain

[1] `http://ai.stanford.edu/users/sahami/CS2013/`

type of learning it may use game-play mechanics as tests for this type of learning. Accessibility of the game was described by:

- What was required to run the game;
- How much the game cost to run;
- How much time it took to play the game or at least to get some beneficial learning out of it.

Each of the 41 games was then compared to this list of criteria.

9.2.1 Accessibility

In terms of accessibility, four general types of games were identified. These were desktop games, mobile games, unplugged games and web games. Desktop games run on a computer, but require installation before they can be used. Mobile games run on a mobile device such as an iPhone, Android device or an iPad—in general, a smart phone or a tablet. Web based games run on a standard browser set up. This allows for plugins like Flash to have been installed. Browser based games could potentially be played on a mobile or a desktop device. Unplugged games were games that took players away from the computer, typically a board game or other sort of physical activity. Of the 41 games reviewed, five were desktop games, four were mobile games, 21 were browser-based games and a eleven were unplugged games. Most games were found to be available at no cost. Only six games had a purchase cost, three of which were two US dollars or less. Most games also required only a small time investment—half hour or less. There were only ten games that required more time than this, of which only half required a particularly large amount of time, that is, two or more hours. It is important to note that several of the games that required less than half an hour had more than half an hour's worth of content, but substantial learning of the

core concepts of the game required less than half an hour. From this we start to get an idea of what the typical game that teaches computer science looks like: a typical educational game that teaches computer science is a web-based game that requires only half an hour or less for meaningful learning and is available for free.

9.2.2 Topic coverage

Despite identifying 41 games that teach computer science, only a few topics were covered by these games. In particular it was found that games covered two main topics, and a third topic with moderate coverage. The two main topics were binary and decimal conversion and introductory programming concepts, and the third topic was networking concepts. While networking had a moderate coverage it was spread over several different subtopics of networking, whereas with binary and decimal conversion and introductory programming concepts, the games all covered very similar subject matter. Of the eighteen knowledge areas given in the ACM/IEEE curriculum, only seven were covered by games, and even these usually only covered a narrow subtopic. In particular, the following topics had at least one game for them: The traveling salesman problem, sorting and searching algorithms, run length encoding, Turing machines, software engineering models, cyber security and graphs. The games that taught introductory programming concepts and binary decimal conversion were largely web-based and mobile games. In fact, mobile games were exclusively of used for binary or introductory programming concepts.

The binary games typically consisted of a game that presented the player with an arbitrary number of randomly generated decimal or binary values and required the player to enter the appropriate converted decimal or binary number. Typically this came with a time limit in order to put more pressure on the player. Most of these games included some sort of scaffolding, such as decimal numbers

under each bit to represent their place value. Additionally, some games had metaphors for what each bit was, such as switches, where as other games just used zeros and ones that could be changed with a click or a tap.

Most of the introductory programming concepts also had several factors in common. These games typically involved moving some sort of agent such as a robot through a maze performing certain tasks. The player would have to issue the commands in advance and was given a limited scope in which to do this. Often the player would be allowed constructs such as functions, which if used correctly could increase the number of commands the player could issue. Players where sometimes encouraged to use techniques such as recursion to produce potentially limitless number of commands. Players could also issue conditional commands in some of these games.

9.2.3 Learning quality

The next section of the review looked at how a game might encourage learning using techniques with strong theoretical underpinning, as mentioned earlier. Specifically, the following properties were looked for:

- Active teaching
- Length of the game
- How closely gameplay was tied to the educational content and for how long
- Level of feedback provided
- Did the game have clear and simple goals?
- Did the game encourage intensely focused concentration?
- Level of control the player had in the game

- Was the level of difficulty of the game kept well matched to the level of the players skill? Additionally, was the game's level of difficulty such that it helped the player gain confidence?

All of these properties, aside from how close educational content stayed tied to the game play, were found, when viewed independently of one another, to be present in roughly 70%, or more, of games. It was found that just over a quarter of games kept a high level of educational value throughout the entire game. It was found that no games had all of these properties and in fact only one game (*Roborally*[2] 9.1) had all of these properties except keeping the educational content tied to the game play. It was found, on average, that each game had just over half of these properties.

9.2.4 Types of learning

The final part of the review investigated in what way each game taught its content. In particular, learning was categorized into three different types of knowledge: cognitive, skill-based and affective. While it was interesting to see which games taught affectively, and such learning did not typically fit in with the ACM curriculum guidelines. That is not to say that the ACM curriculum guidelines are in any way incompatible with affective learning or they do not encourage it, but simply that the games that did encourage affective learning did so in such a way that the learning was not covered by the ACM curriculum guidelines. It was found that over 80% of games covered some fact-based knowledge as cognitive knowledge, and over 50% covered skill-based knowledge, so it seems that both fact-based and skill-based knowledges are popular for educational games that teach computer science. However, fact-based knowledge is significantly more popular than skill-based knowledge. It is perhaps surprising

[2] `http://boardgamegeek.com/boardgame/18/roborally`

Fig. 9.1: Roborally

that more games focus on skill-based knowledge considering that computer science topics do not typically involve much focus on skill-based learning. Although this surprise is perhaps alleviated when we considered that almost half of the games that utilized skill-based learning had difficulty keeping the development of the skill relevant to the learning of the computer science topic.

9.2.5 Guidelines

Compiling this information allows for better informed decisions on how to continue work in this field. One important conclusion to make from this data is that we could use more games that teach computer science, particularly since many

topics do not appear to have been touched. Attempting to provide some insight into a better approach for the future of this field was the main purpose of this work. The next step was to provide some guidelines that might help indicate how the corpus of existing games teach computer science might be expanded. Such a set of guidelines were established based on both the review criteria and the data gathered from the review. In short, these guidelines suggest that one should aim to cover a topic that has not already been covered, or aim cover it in a different way than has already been done. To achieve this the developer must acknowledge the type of learning they are targeting, for which it may be best to test for learning using game mechanics, ensuring that the player cannot complete the game unless they are learning in the desired way. The teaching in the game should be as active as possible. The game should provide sufficient content and/or keep the educational content tied to the game play for as long as possible, and it should aim to achieve flow by matching the challenge to the level of the players skill, keeping clear and simple goals, providing plenty of feedback, encouraging intensely focused concentration and giving the player a high degree of control. Additionally, the game should aim to help the player build confidence. One should also consider carefully how to make their game available to the target audience. In some cases the best choice may be to release it as an Unplugged game for those who do not have much access to computers, or as a webgame in order to reduce set up time.

Most existing games have opted to teach a fairly small part of a topic quickly. Since games teach best when embedded into a discussion, keeping them short makes them easier to embed into a lesson. However, if a game is too short it may not be worth the time taken to put students on a computer just to teach a small part of a topic. It is worth carefully considering and perhaps investigating the target audience before deciding how much content a game should have. Finally,

financial costs should also be considered when designing a game. It is likely best to assume that a game with a lower cost will reach a wider audience. However, universally encouraging educational games to be released at low cost maybe a bad idea, as it may become difficult to procure the funding required to produce more expensive games which maybe worth more investment on the consumers end.

9.3 Future work

The field of using educational games to teach computer science could benefit from future work targeting areas that haven't been covered by existing games. Endeavoring to have new games to cover new topics may lead us to one day having covered most, or even all, of the curriculum, providing teachers with a very useful resources for most scenarios. Additionally, this could be one way of finding, albeit by brute force, if certain concepts are harder to teach than others using educational games. Another very important area for future work is investigating why certain areas of computer science had already been covered by educational games and others haven't. This may be a more direct way of finding out if certain topics are harder to create educational games for than others. Establishing whether or not particular topics can be covered more effectively by games than others is a very important research area. This is because it is quite possible that our research based on the existing educational games, which has established them as effective learning tools, may only be relevant for topics of a certain type. Another area for future work could be testing the guidelines presented in this research. While most of the guidelines have a strong theoretical background, some notable areas such as targeting certain types of learning have a much less solid foundation. In particular, the review suggests what certain tests for different types of learning might look like as game mechanics. While

these are based on the research of Bandura and Wood [1], the translation is not as strong as it could be. Additionally, it is not necessarily the case that testing for a type of learning in game will encourage that kind of learning. These ideas are not necessary parts of the guidelines, so their potential fallibility does not undermine the usefulness of the guidelines, but it would still be interesting to see them tested. Additionally, it will be beneficial to see how the guidelines worked together as a whole. As found in chapter 7 (case study) with *the RAID Arrays game*, there can sometimes be trade-offs between suggestions from the guidelines.

Appendix A

EDUCATIONAL GAMES THAT TEACH COMPUTER SCIENCE

Name of game	Topic	Link
Desktop		
CyberCIEGE	Cyber security	`http://cisr.nps.edu/cyberciege/`
Cisco myPlan-Net	History of networking	`https://learningnetwork.cisco.com/docs/DOC-7635`
Cisco Mind Share Game	Networking and binary	`https://learningnetwork.cisco.com/docs/DOC-3820`
ToonTalk 3	Programming concepts	`http://www.toontalk.com/English/free.htm`
SimSE	Software engineering models	`http://www.ics.uci.edu/~emilyo/SimSE/`
Mobile		
Binary Game	Binary	`https://itunes.apple.com/app/binary-game/id304957372`
Binary maglock	Binary	`https://play.google.com/store/apps/details?id=air.com.Machiavelli.BinaryMaglock`
Cross Binary	Binary	`https://itunes.apple.com/us/app/cross-binary/id433859279`
Cargo Bot	Programming concepts	`https://itunes.apple.com/us/app/cargo-bot/id519690804`

Tab. A.1: List of computer science educational games identified (continued in Table A.2)

Name of game	Topic	Link
Unplugged		
Binary Crossnumber Puzzle	Binary	`http://puzzlepage.blogspot.co.nz/2008/01/binary-crossnumber-puzzle.html`
Crossbin Puzzles	Binary	`http://cse4k12.org/crossbin/index.html`
Internet Peering Game	Internet peering	`http://drpeering.net/white-papers/Peering-Simulation-Game.html`
The Security Protocol Game	Networking security protocols	`http://web.science.mq.edu.au/~len/secgame/index.html`
D0x3d!	Networking security terminology	`http://d0x3d.com/`
CTRL-ALT-HACK	Networking security terminology	`http://www.controlalthack.com/`
Tablets of Stone	Network protocols	`http://csi.dcs.gla.ac.uk/`
Roborally	Programming concepts	`http://www.wizards.com/default.asp?x=ah/prod/roborally`
The Orange Game	Routing	`http://csunplugged.org/routing-and-deadlock`
Battleships Unplugged	Searching (linear, binary, hash)	`http://csunplugged.org/searching-algorithms`
Map coloring	Graphs and graph algorithms	`http://csi.dcs.gla.ac.uk/`

Tab. A.2: List of computer science educational games identified (continued from Table A.1 and continued in A.3)

Name of game	Topic	Link
Web		
Binary Flash-cards	Binary	`http://scratch.mit.edu/projects/1134080/`
Binary Fun	Binary	`http://britton.disted.camosun.bc.ca/binary.swf`
Binary Number Quiz	Binary	`http://scratch.mit.edu/projects/723745/`
Cisco Binary Game	Binary	`http://forums.cisco.com/CertCom/game/binary_game_page.htm`
CS Unplugged Binary	Binary	`http://sucs.org/~tobeon/project/binary.html`
Beadloom Game	Programming concepts	`http://www.game2learn.com/?page_id=92`
Blockly	Programming concepts	`http://blockly-demo.appspot.com/static/apps/maze/index.html`
Brando the Egg Hunter	Programming concepts	`https://www.nonamesite.com/web/cs-stem/home`
Light-Bot	Programming concepts	`http://armorgames.com/play/2205/light-bot`
Light-Bot 2.0	Programming concepts	`http://armorgames.com/play/6061/light-bot-20`
RoboZZle	Programming concepts	`http://robozzle.com/`
Swap Puzzle	Programming concepts	`http://www.cs4fn.org/algorithms/swappuzzle/`
Treasure Hunter	Programming concepts	`https://www.nonamesite.com/web/cs-stem/home`
The Orange Game Flash	Routing	`http://www.info-study.net/unplugged/activity10-j.html`
Picture Logic the Puzzle Game	Run-length encoding (compression)	`http://tonakai.aki.gs/picturelogic/play/index_e.php?PNum=1`
Sorting Bricks	Sorting algorithms	`http://mathsite.math.berkeley.edu/sorting/brick.html`
Tour Finder	Traveling salesman problem	`http://www.tsp.gatech.edu/games/tspOnePlayer.html`
Tour Finder, Two Player	Traveling salesman problem	`http://www.tsp.gatech.edu/games/tspTwoPlayers.html`
Pinky's Pipes Pickle	Graph algorithms	`http://www.cs4fn.org/algorithms/maxflow/`
Manufactoria	Turing machines	`http://pleasingfungus.com/Manufactoria/?Manufactoria`
Dead man's chest	Gray codes	`http://www.cs4fn.org/binary/lock/`

Tab. A.3: List of computer science educational games identified (continued from Table A.2)

Appendix B

REVIEW SUMMARY

An extensive review of all 41 games is available online[1]. The review applies the criteria to each game, and will be useful for those considering a particular game. This chapter will summarize some of the more interesting and important findings of the review. The summary will discuss topic coverage, which game design elements were recurrent, and availability. The summary will not discuss what types of learning where targeted, although this will be picked up in chapter 8 (discussion).

B.1 Topics covered

Only a few topics have much content available, and a couple of these have a relatively large amount of coverage. Specifically, two types of game were quite prevalent, those teaching introductory programming concepts, and games teaching decimal to binary conversion.

The games that taught introductory programming concepts were *RoboZZle*, *Light-Bot*, *Light-Bot 2.0* B.1, *Brando the Egg Hunter*, *Treasure Hunter* and the *BeadLoom Game*. In general these games give a familiarity with high level concepts covered in introductory programming courses. It seems the primary goal of these games is a type of affective learning, as the games introduce players

[1] `http://tinyurl.com/coscgames`

into some of the challenges of programming in a fun and rewarding way. This is generally achieved by having the player issue commands to a character, often a robot, towards solving a puzzle. Commands are simple, such as move forward or move left, and must be issued prior to execution. The goal of the puzzles is typically fairly straightforward, often as simple as get from point A to point B, collecting something along the way.

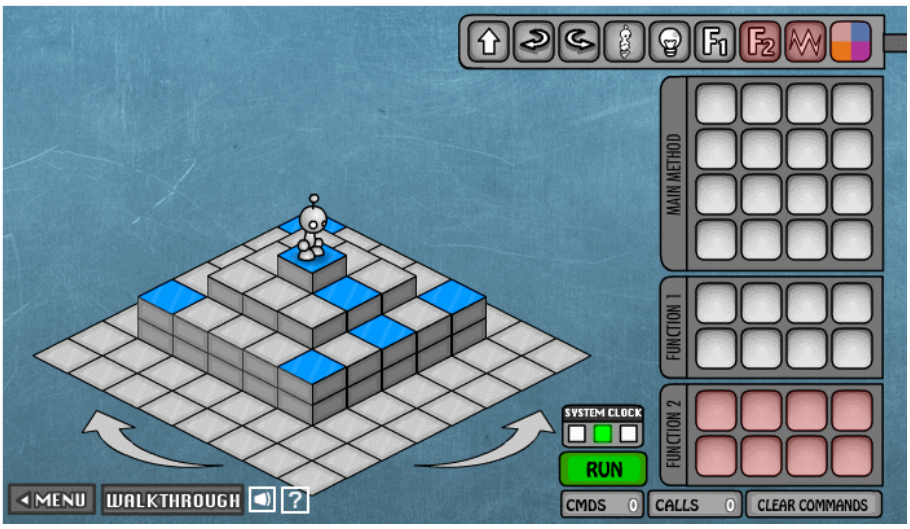

Fig. B.1: Light-Bot 2.0

Additionally, these games teach some programming concepts from the SDF (Software Development Fundamentals) category from the ACM curriculum 2013 (iron man, draft). Two topics in particular were covered in most of these games, SDF/Fundamental Programming Concepts and SDF/Algorithms and Design Solving. Of these topics only functions, conditionals, recursion and problem-solving strategies were taught.

Each of these games teach the same problem solving strategies that are taught in early programming courses. This is achieved by presenting the player with puzzles for which these strategies are needed.

Functions are often represented directly or indirectly as custom (but limited)

sets of commands that can be called with a single command. The use of functions is typically encouraged either by restricting the number of commands a player can make or rewarding them for using as few commands as possible.

Conditionals appear in fewer games. They are typically represented fairly directly by special commands that only function when the character is in a certain circumstance. For example, in *Light-Bot 2.0* the robot can activate colored squares, changing their own color as well. The player can issue commands that only activate if the robot is a particular color.

RoboZZle and *Light-Bot 2.0* also taught the concept of recursion. Recursion was taught by presenting the player with a puzzle that could only be solved by using a function that called itself. This was either achieved by restricting the number of commands a player could make or by inflating the number of commands it required if recursion was not used.

The games that taught decimal to binary conversion were *Cisco Binary Game*, *Binary Fun*, *Binary Flashcards*, *CS Unplugged Binary Game*, *Binary Number Puzzle*, *Crossbin Puzzles*, *Binary Crossnumber Puzzle*, *Cross-Binary* and *Binary Game*. Most of these games had strong similarities, featuring a time limit in which a player must convert a given decimal number to binary. Binary to decimal conversions were also covered with games like the *Cisco Binary Game*[2] which covered both directions. Binary comes under the topic AR/Machine-level representation of data from the ACM curriculum guide 2013 (iron man draft).

The feedback that binary games provided was rich enough that players can learn binary simply from playing them—the player could flip a bit and see in real time the changes to the equivalent decimal. Additionally, players were shown the place value for each bit, for up to eight bits. Numbers were presented randomly, meaning that players were not given any direct experience with counting and may not see some of the more prevalent patterns that occur

[2] `http://forums.cisco.com/CertCom/game/binary_game_page.htm`

with binary numbers. Most games represented binary numbers directly using 1 and 0s. However, metaphors were also used, such as on/off switches and even watermelons with a number of seeds equal to their place value.

No other topics have received the same extent of treatment as introductory programming concepts or basic binary concepts have. However, networking does have some extent of coverage, although it is more diverse. *CyberCIEGE*, *"Picture Logic the Puzzle Game"*, *the Orange Game* and *the Orange Game Flash* all teach networking topics. *CyberCIEGE* covers cyber security from IAS/Network Security, specifically attack types and defense mechanisms and countermeasures. *"Picture Logic the Puzzle Game"* covers Run-length encoding, which finds its best fit under AR/Memory system organization and architecture, although could be covered by several other topics dealing with data compression. *The Orange Game* and its Flash based counterpart cover routing from NC/Routing And Forwarding.

Other topics covered included Turing machines, sorting algorithms, the traveling salesman problem, software engineering models and the history of networking. *Manufactoria* covers Turing machines from AL/Basic Automata Computability and Complexity. *Sorting Bricks* covers sorting algorithms from AL/Fundamental Algorithms. Tour finder covers the traveling salesman problem from AL/Basic Automata Computability and Complexity, specifically P versus NP problem. *SimSE* covers software engineering models from SE/Software Processes, specifically process assessment models. *Cisco myPlanNet* covers the history of networking from SP/History—history of the Internet and History of networking.

B.2 Availability

41 games were reviewed, of which 21 were browser based games, 4 were mobile games, 11 unplugged games and 5 game software. Almost all of these games

are free, with notable exceptions being a few cheap mobile games and *Robo-rally* (US$49.99 list price on amazon.com, September 2013), Cisco Mind Share (US$42.95 from the CISCO learning network store, September 2013), *CTRL-ALT-HACK* (US$25.00 list price on amazon.com, September 2013) and d0x3d!. The d0x3d! game is open source, so is essentially free, although a version with cards and counters can be purchased for US$25. In general, web and mobile games were short, normally requiring up to 30 mins of play time. Some games such as Lite-bot 2.0 and *RoboZZle* have a reasonable amount of content and support a much higher degree of longevity. However, the amount of time investment for valuable learning is still relatively small. All of the Game software required much larger amounts of time. *SimSE*, *ToonTalk 3*, *CyberCIEGE*, *Cisco Mind Share* and *Cisco myPlanNet* all required more than two hours of play time.

B.3 Educational properties

B.3.1 Active teaching

Binary games

Of the games that taught binary number conversion, most of them are presented in a way that students who know nothing about binary will be able to learn it from the game as they play it, without prior instruction. This active teaching is done by giving the player a decimal number and requiring them to enter the binary equivalent by flipping bits. The player is shown how much each bit is worth and the decimal value of the binary number they have entered (except in the hard mode of the *CS Unplugged Binary Game*), so they can initially try to solve the conversion by trial and error, and eventually develop strategies. In the *Cisco Binary Game* they also present the player with a binary number and must enter its decimal value using a pop-up number pad. *Binary Flashcards* has an example button that shows a diagram that could help people figure out

how to play the game. Another option is in *Binary Number Puzzle* where in easy mode the player is shown how much each bit is worth, in the form of seeds on water melons. Each watermelon is a bit and the place value of the bit is the number of seeds in the watermelon. A commonality between *Cross-Binary* and *Binary Crossnumber Puzzle*s was that in order to solve a given puzzle the player must convert hints from decimal into binary. In addition, *Cross-Binary* has a small bar up the top that can do the conversion for the player. *Crossbin Puzzles* uses a different approach where in order to solve a given puzzle the player must convert hints from hexadecimal into decimal and enter the binary representation in the puzzle. Alternatively, if the player is able they could convert straight from hexadecimal into binary.

Only three of the binary games utilised any passive teaching: *CS Unplugged Binary Game* teaches passively in the tutorial by explaining conversion between decimal and binary with text, and *Binary Crossnumber Puzzle*s teach passively with short instructional text above the puzzle. *Binary Maglock* teaches predominantly passively through a tutorial that explains how to play the game. While the game itself may teach binary actively, it seems unlikely a player unfamiliar with binary would learn it solely from playing this game. A more active approach provides scaffolding to help students learn the concept; for example, in the easy mode of the *Binary number quiz* the student is shown how much each bit is worth, in the form of seeds on water melons. This scaffolding is removed in the hard mode.

Logic games

The games that taught programming concepts primarily taught actively, with some utilising supporting passive teaching. In the case of *ToonTalk 3*, tools were slowly introduced that, when combined, could be used to build instructions in a way similar to programming. These tools are introduced throughout the

game as the main focus of the gameplay. The tools functions are described passively by an in-game character but are quickly put into use by the player. The rest of the programming games taught actively by presenting the player with a series of specifically designed puzzles designed to familiarise the player with the subject matter, such as *Brando the Egg Hunter* and *Treasure Hunter*. The majority of *RoboZZle*, *Light-Bot Light-Bot 2.0*, *Cargo Bot* and *Blockly Maze* are taught actively. This active teaching is achieved by presenting the player with a series of puzzles that are designed to familiarise the player with functions and problem solving strategies, except for *Blockly Maze* which has the same effect but achieves it through teaching players how to use Blockly, a visual programming language. *Blockly Maze* initially provides simple paths to follow and turtle commands (such as move forward and turn left), and gradually provides more commands (such as loops and conditionals), and more complex paths that end up as arbitrary mazes. *RoboZZle* has a small amount of passive teaching in the tutorial. *Light-Bot*, *Light-Bot 2.0* and *Blockly Maze* have a small amount of passive teaching throughout the game. The passive instruction in the form of text is used to explain how to play the game and some of the concepts involved. In order to succeed at the game *Roborally*, players must plan their robot's movements in advance, although the movements can be interfered with by other robots and certain board elements. The result of this is that players must develop problem solving strategie, somewhat similar to those used when programming, in order to succeed at the game. The *BeadLoom Game* teaches actively by presenting the player with a series of puzzles. The puzzles are simply images the player must replicate using as few as possible operations. These puzzles are designed to familiarise the player with problem solving strategies. As far as teaching Computer Science is concerned, the main strategy that is taught is using iteration to solve problems.

Networking games

"Picture Logic the Puzzle Game" teaches actively. This active teaching is achieved by presenting the player with a series of nonograms (two dimensional grids where run-length representations are given for each row and column). To solve a nonogram the player must become familiar with the concept of run-length encoding, although the term isn't mentioned explicitly. The game also relates to computer tomography, a part of computer vision, where a two- or three-dimensional image is reconstructed from one-dimensional scans.

The *Orange Game* and the *Orange Game Flash* teach actively. The goal of the game is move fruit from hand to hand until all of the fruit is in the right hands. There are restrictions on how the player can move the fruit so that the game emulates, to a certain extent, a network. The intent is that the player will gain an appreciation of congestion and deadlocks through playing this game.

Dead Man's Chest teaches actively. This active teaching is achieved by presenting a puzzle for which the player must employ an algorithm that follows a pattern used in Gray codes to complete.

Pinky's Pipes Pickle teaches using active teaching. This active teaching is achieved by presenting the player with a network of pipes which they must channel water through in the most efficient way possible. Each pipe can only handle a set amount of water. This gives the player familiarity with network congestion, which can be discussed within the context of Computer Science.

Tablets of Stone simulates a hands on experience with the execution of communication protocols that the players are required to improvise. In this way the game provides an entirely active experience.

Most of the *CyberCIEGE* content is conveyed both actively and passively. The game offers passive teaching in the form of text and video tutorials. Additionally, the game teaches actively by putting the player in charge of situations

where employees deal directly with the subject matter. The *Cisco Mind Share* teaches passively. The game presents the player with a problem to solve and the appropriate text that teaches them how to solve it. *CTRL-ALT-HACK* B.2 teaches passively. This passive teaching is achieved by presenting the player with information about attack types on mission cards in game. The *Internet Peering Game* simulates a hands on experience for the players with running an Internet service provider company, including experience with peering negotiation. In this way the game provides an entirely active experience. However, teaching how the game is played maybe where the main part of learning comes in, at least as far as viewing it under its topic from the ACM curriculum guidelines. How to teach the players to play the game is not clearly defined, but it seems unlikely to be presented actively as a part of the game. The *Security Protocol Game* simulates a hands-on experience for the players with the execution of security protocol's and their vulnerabilities. In this way the game provides an entirely active experience. What the protocols are and how they are executed is not taught explicitly by the game itself, meaning there is no guarantee that it will be taught actively. However, the players will still get experience with the protocols vulnerabilities actively through playing the game. *Dox3d!* teaches actively through constructing a fictitious scenario with some similarities to real world networking issues. This is furthered by making in-game terms relevant to real world terminology.

Other games

Manufactoria has a small amount of passive teaching throughout the game in the form of short videos explaining your next "job". However, the rest of the game is taught actively. This active teaching is achieved by presenting the player with a series of puzzles that are designed to familiarise the player with finite state machines and Turing machines.

Fig. B.2: Cards from CTRL-ALT-HACK

The *Tour finder* game and its multi-player counterpart *Tour Finder Game, Two Player* teach actively. The game teaches what the travelling salesman problem is and the difficulty of finding optimal solutions in large problems. This is achieved by simply asking the player to find the best route they can for a given set of cities. The *SimSE* game teaches actively by having the player control a software development team using either an agile or waterfall process. The player must complete each step of the software engineering process in order to develop some fictitious software project, which gives the player the opportunity to try their own approaches, giving them first-hand experience with each process.

Cisco myPlanNet and *Sorting Bricks* were the only games that had a lot of passive teaching. In *Cisco myPlanNet* all the information about the history of the Internet is presented as blocks of text. *Sorting Bricks* describes at each step how to apply a sorting algorithm with text on the left hand side of the screen, although the player must actively make decisions on how to achieve the requirements of the instruction i.e. decide which bricks to compare. Additionally, the game offers animations that show how each algorithm works on larger data sets. The player does get to put the algorithm into practice by following each step of the algorithm themselves. In this way the game offers a small amount of active learning. In *Battleships Unplugged* a decent amount of the content is

covered through active teaching. Students will realise from playing through the three types of searching which is more efficient and possibly why. The activity suggests teaching outside of the game in the form of discussions. Rather than acting as passive teaching in the game this fills the role of discussion/debriefing.

Map Coloring is intended to be used to illustrate the complexity of the problem that is addressed by an algorithm. This is taught actively by having players attempt to solve the problem optimally before representing it in graph format.

Swap Puzzle teaches actively and passively. This active teaching is achieved by presenting the player with a series of puzzles and recording the actions the player takes to solve the puzzles. The player can then see their actions followed as an algorithm that solves the puzzle. Additionally, the game starts with a page of text teaching the player how to play the game, but also explaining a bit about algorithms.

B.3.2 Keeping the game content tied to the educational content

Binary games

Cross-Binary, *Binary Crossnumber Puzzle* and *Crossbin Puzzles* assumes that the player knows how to do the convert between binary and decimal. The other binary games did not make this assumption and could be played without any prior experience with binary. *Cross-Binary* has a built in tool that does the conversions for the player, meaning players are likely to be learning how to convert between binary and decimal. In order for the player to improve at the following games (*Binary Game*, *Cisco Binary Game*, *Binary Fun*, *Binary Flashcards*, *Binary Number Puzzle*, *CS Unplugged Binary Game*, *Binary Maglock*) the player must almost certainly also improve at translating between decimal and binary. The *Binary Game*, *Cisco Binary Game* and *Binary Fun* become exclusively about speed of translation after a while. In the games *Binary Flashcards*,

CS Unplugged Binary Game, and *Binary Maglock* eventually the only challenge left in the game is being consistently accurate.

Logic games

There are a range of levels in which the gameplay is related to the educational context of teaching introductory programming concepts. In the games *Blockly Maze* and *ToonTalk 3*, gameplay is strongly tied to the educational context. In *Blockly Maze* the player is taught the basics of using the 'Blockly' visual programming language with very little focus on anything else. Since 'Blockly' is a visual programming language, this naturally lends itself to teach some basic programming concepts. In *ToonTalk 3* the gameplay revolves around the content in 'puzzle mode'. The puzzle mode's gameplay consists of introducing each of the games tools, which are effectively used to program. Each tool performs a specific function, such as performing simple arithmetic on numbers. The games *Light-Bot*, *Light-Bot 2.0*, *Cargo Bot*, *RoboZZle* and *Roborally* are moderately tied to the educational context. The player must develop problem solving strategies and use functions to be able to solve the puzzles. However, the problem solving strategies required are fairly different to those used when programming. The longer these games go, the amount Computer Science being learned diminishes and the games become more about finding patterns. In the case of *RoboZZle*, this is mitigated to a certain extent by the campaign, which has levels specifically designed to highlight important Computer Science concepts.

In contrast *Brando the Egg Hunter*, *Treasure Hunter* and *BeadLoom Game* are only loosely tied to the educational context of teaching introductory programming concepts. In *Brando the Egg Hunter* the game-play quickly becomes more about geometry than Computer Science In *Treasure Hunter* the game-play becomes trivial fairly quickly with regard to teaching elements of Computer Sci-

ence. However, throughout the levels the game cycles through a few optional trivia questions about computing in general.

Much of the *BeadLoom Game* gameplay has little to do with Computer Science at face value, but it does teach some problem solving strategies somewhat similar to those utilised in Computer Science. Additionally, a lot of the gameplay could be used as an example of Computer Science concepts. For example, one could use the this game as an example of how complex pictures can be drawn using just triangles for graphics. As a bonus, the game implements the painter's algorithm.

Networking games

The game-play of *Security Protocol Game* and *Cisco Mind Share* is strongly tied to teaching the vulnerabilities of the security protocols at hand. Additionally, knowledge of how the protocol works could be reinforced by seeing it used in practice. However, if players start making up their own security protocols the game is perhaps less relevant to Computer Science. *CTRL-ALT-HACK* keeps the game-play tied fairly closely to the content. The player will learn about 'white hat hackers' and what they do as they play through the game. However, the more the game is played the less new things there are to be learned by the player.

Throughout the *Cisco Mind Share*, game-play is strongly tied to the developers intended educational outcomes. The game essentially goes from one problem to the next, teaching you about the problems as you solve them with the learning continuing throughout the game, but not all of the game's content is related to Computer Science specifically. *Dead Man's Chest* keeps the gameplay tied fairly closely to the content. The educational content is the pattern itself through which the player will get a greater understanding of the more they play the game. However, after a while the player might realise the pattern

and gain no further educational benefit from playing the game. *The Orange Game*, *the Orange Game Flash* and *CyberCIEGE* were all moderately tied to the educational context of teaching networking. In *the Orange Game Flash* it is likely that the player only needs to complete one level of each layout in order to have learned the optimal amount from this game, after which the game becomes more about figuring out algorithms that are only useful to this game. This is because the only challenge that would be left is to solve the levels more efficiently. It is likely that the player in *the Orange Game* only needs to complete a game once in a few different layouts in order to have learned the optimal amount from this game. In *CyberCIEGE* each scenario revolves, specifically addressing a topic. Only some of the content is Computer Science, although learning continues throughout the game. *Map Coloring* keeps the game-play tied somewhat closely to the educational content. Since the educational focus is an appreciation of the complexity of the problem, once this is learned the game no longer has much educational value.

The game-play of *Tablets of Stone* is strongly tied to teaching the complications of safely delivering data over a network. However, once players are familiar with these complications, *Tablets of Stone* is unlikely to teach anything particularly valuable.

Internet Peering Game, *Dox3d!* and *"Picture Logic the Puzzle Game"* were only loosely tied to the educational context of teaching networking. In order to play the *Internet Peering Game* the learner needs to have received most of the Computer Science related learning prior to playing the game, so the game is predominately reinforcing prior learning.

Since the content in *Dox3d!* is relevant to Computer Science, but does't fall directly under the ACM curriculum, this has an impact on how closely it can be tied to the educational context of teaching networking within the ACM

curriculum. Given this impact, the game is about as closely tied to the context as it can be. However, after playing the game a couple of times the learner will have exhausted the educational content available.

In *"Picture Logic the Puzzle Game"* once the player has learnt how run-length encoding works the game no longer teaches Computer Science.

Pinky's Pipes Pickle keeps the game-play content tied fairly closely to the educational content, although it is likely that the player only needs to complete a few levels in order to have learned the optimal amount from this game, after which the game becomes more about figuring out algorithms that are primarily specific to this game.

Other games

Battleships Unplugged, *Tour Finder Game* and *Tour Finder, Two Player* were strongly tied to the educational content. However, in each of these games valuable learning was fairly short-lived. *Battleships Unplugged* ties the educational content very closely to the game-play, at least for the first play through, but playing the game more than once would probably have only small learning benefits. *Tour Finder Game* and *Tour Finder, Two Player* are almost exactly tied to the Travelling Salesman Problem. The player does nothing but try and find the best route for a given problem. However, after a short while the player will have learnt what the travelling salesman problem is. Afterwards the player can only learn that it gets increasingly difficult the larger the number of cities, something the player will only find if they increase the number of cities themselves. *Manufactoria*, *Sorting Bricks* and *SimSE* were moderately closely tied to their educational content. In *Manufactoria* the player must figure out how finite state automata work in order to solve the puzzles. The educational content is spread throughout the puzzles. This means that players will continue learning until near the end.

In *Sorting Bricks*, game-play is fairly tightly tied to some of the content. The game-play has the player try out an algorithm with the goal of making as few comparisons as possible. This will allow the player to gain first hand experience with the efficiency of the algorithms. Learning continued through most of *SimSE* B.3 as the players were required to simulate a software development process from beginning to end.

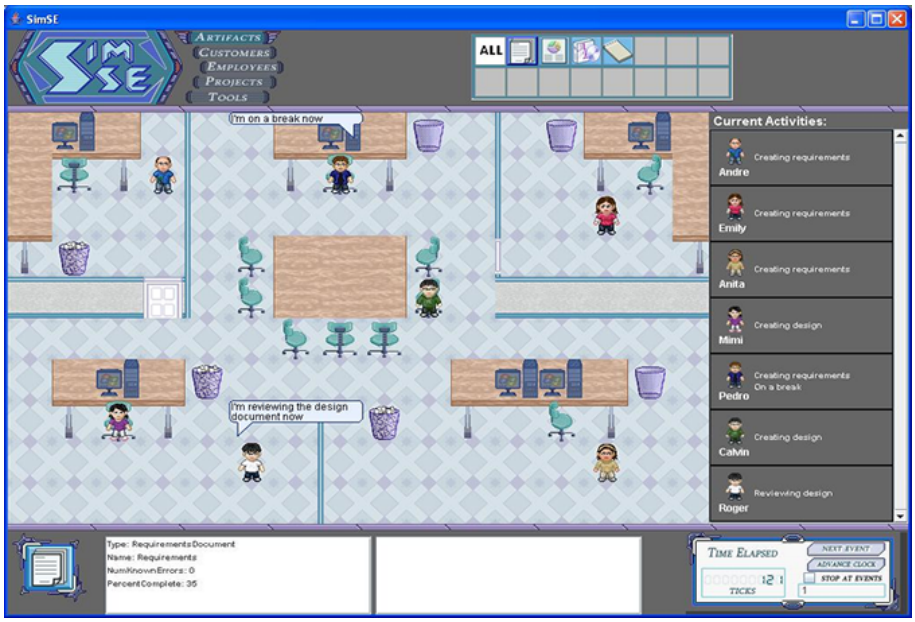

Fig. B.3: SimSE

Cisco myPlanNet was loosely tied to its educational content with the Computer Science content spread fairly evenly throughout the game right from the start to the end.

The game-play of *Swap Puzzle* is tied fairly loosely to the content. The player must develop problem solving strategies to be able to solve the puzzles. However, the problem solving strategies required are fairly different to those used when programming. The player learns most of the learning related to Computer Science through passive teaching at the early stages of the game,

although the game doesn't really last long enough for this to cause problems, with the gameplay staying tied to the content overall.

B.3.3 Debriefing/discussion included

Binary games

Of the games that taught binary, the only ones that included debriefings or discussions were *Cross-Binary*, *Binary Game*, *Cisco Binary Game*, *CS Unplugged Binary Game*, *Binary Maglock* and *Binary Fun*. In *Cross-Binary*, pressing 'how to play' in the menu produced a description of what binary is and how it relates to this game. *Binary Game*, *Cisco Binary Game* and *Binary Maglock* briefly discuss binary and the game's relation to it in their respective instruction section. *Binary Crossnumber Puzzle* provides a very brief discussion on the web page the puzzle is hosted on. *Binary Fun* has an 'about' section which talks a bit about how the game teaches binary and what inspired the making of the game. The tutorial in *CS Unplugged Binary Game* describes the process of converting binary to decimal as well as why it is important.

Logic games

Only two of the games that taught introductory programming concepts included debriefings or discussions. These games were *Light-Bot* and *ToonTalk 3*. At the end of *Light-Bot* the Game it explains that the player has been using the kind of logic a programmer uses regularly and suggests the player learn to program. This is a form of very short debriefing. *ToonTalk 3* gives programming context and mentions how it relates to Computer Science.

Networking games

Only four of the games that taught networking included debriefings or discussions. They were the Orange Game Flash, *CTRL-ALT-HACK*, *Dead Man's Chest*, *CyberCIEGE* and *Cisco Mind Share*. Extra information is given on the web page the *the Orange Game Flash* is hosted on, although it is in Japanese.

Additionally, the game gives some idea of context by describing each of the level layouts as network patterns. *CyberCIEGE* includes a set of video tutorials that discuss in depth the games relation to the subject matter. As *Cisco Mind Share* is almost entirely passively taught, ongoing discussions are a fundamental part of the game. A major part of *CTRL-ALT-HACK* seems to fulfill the role of a debriefing or discussion. The website, for *Dead Man's Chest* and *Pinky's Pipes Pickle* goes into some detail about how each of the games relate to their subject matter. The *Tablets of Stone* format fundamentally encourages this kind of discussion and the activity comes packaged with resources for teachers that fulfill this role.

Other games

Only five of the games that were Computer Science games that did not specifically teach networking, binary or introductory programming concepts included debriefings or discussions. They were *Sorting Bricks*, *Cisco myPlanNet*, *SimSE*, *Map Coloring* and *Battleships Unplugged*. At the start of the game of *Sorting Bricks*, a short piece of text describes what the game is about. Additionally, the game has several animations showing larger sets of data being sorted with the various algorithms. This helps gives context on the usefulness of each algorithm. Most of content in the game *Cisco myPlanNet* is embedded into discussions. In *SimSE* a brief description is given when the player first starts the game. In *Battleships Unplugged* the activity itself describes where and what to discuss in between games. *Map Coloring* comes packaged with an educational module that fulfills this role. The instructions at the start of *Swap Puzzle* also fulfill this role.

B.3.4 Feedback provided

Binary games

Of the games that taught binary *Binary Game* and *Cisco Binary Game* pro-

vided the learner with a high amount of feedback. Rich feedback is provided in *Binary Game*. If the player is successful in solving a challenge it disappears, the player's score is increased, and a new challenge appears. Additionally, the player is told the decimal equivalent of the binary number they have entered. *Cisco Binary Game* also provides rich feedback. If the player is successful in solving a challenge it disappears, the player's score is increased and a new challenge appears.

Cross-Binary, *Binary Fun* and *Binary Maglock* provided a moderate amount of feedback. The predominant source of feedback in *Cross-Binary* is that the player is shown how long they have been playing for, and which numbers have been found. Feedback is provided in *Binary Fun* by the player being shown the decimal value of the binary number they have entered as well as the goal in decimal. However, when the player reaches the goal they do not automatically win, instead they must declare they have won, to win the round. If the player declares they have the right number, but it is incorrect, or if they run out of time, then the player loses the game. *Binary Maglock* provides feedback to the player through providing a new challenge if they are successful in solving the previous one, but if the player is incorrect a sound is played to indicate this and the challenge remains.

Crossbin Puzzles, *Binary Crossnumber Puzzle*, *Binary Flashcards*, *CS Unplugged Binary Game* and *Binary Number Puzzle* provided the learner with a low amount of feedback. Answer sheets were provided for *Crossbin Puzzles*, so that the player can confirm their answers. Additionally, as the player gets close to completing the puzzle they will get some indication on their accuracy, if they find numbers don't match up where they overlap, this suggests the player has made an error somewhere. The only feedback provided in *Binary Crossnumber Puzzle* is that the puzzle will not work if any part is filled in incorrectly. As

soon as the player finds a contradiction they will know they have done something wrong. Limited feedback is provided in *Binary Flashcards* and *Binary Number Puzzle*. The player is told if their solution is correct as well as how many correct and incorrect solutions they have had so far. In *CS Unplugged Binary Game* the player is told if their solution is correct and when the player finishes ten problems they are shown how many were correct.

Logic games

Of the games that taught introductory programming concepts *Treasure Hunter*, *Brando the Egg Hunter*, *RoboZZle*, *Cargo Bot*, *ToonTalk 3*, *Roborally*, *Blockly Maze*, *Light-Bot 2.0* and *Light-Bot* provided the learner with a high amount of feedback. In *Treasure Hunter*, *Brando the Egg Hunter*, Blockley Maze, *RoboZZle*, *Cargo Bot* B.4, *Light-Bot 2.0* and *Light-Bot* the player is able to watch their 'program' execute, step by step, these games highlight which step is currently executing, although it is not very obvious in *Treasure Hunter*. The program executes in a simple, but highly visual way. This makes the feedback easy to understand and accessible. The player is able to choose when they want feedback by pressing a key word, such as "execute" (in the case of *Treasure Hunter*), which runs their 'program'. Additionally, in *Light-Bot 2.0*, *Cargo Bot* and *RoboZZle* the player can choose to increase or decrease the speed of execution. This allows players to investigate the parts they are having trouble with closely, without wasting too much time. The program executes in a simple, but highly visual way. This makes the feedback easy to understand and accessible. The player is able to choose when they want feedback by pressing go, which executes their 'program'. In *ToonTalk 3* The player constructs objects step by step, seeing the product as they go. Additionally, the player is instructed, to some extent, on how they are doing In *Roborally* plenty of feedback is provided. The player physically executes their program one command at a time.

This allows the player to see if their robot ends up where they want it to be. Additionally, players record errors by accumulating damage tokens in certain undesirable situations.

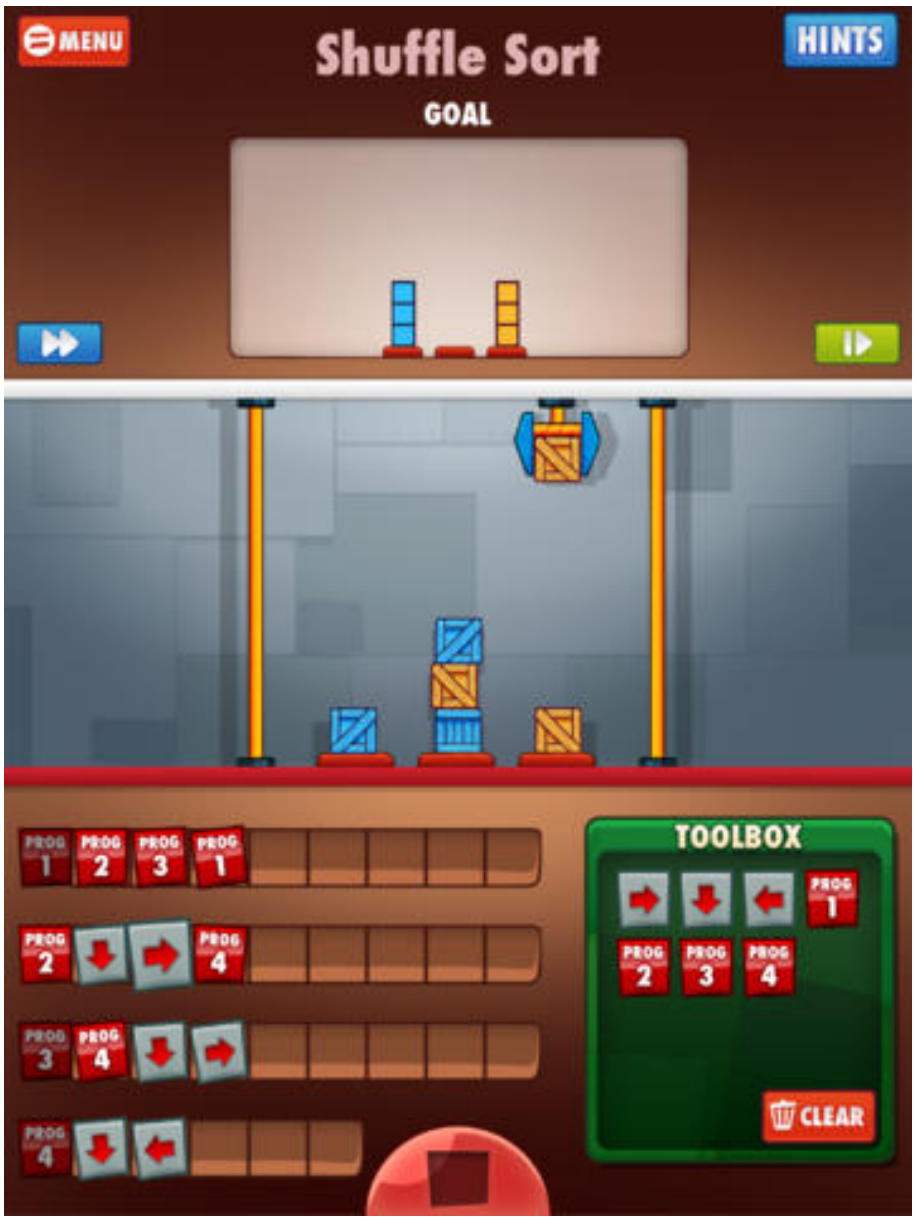

Fig. B.4: Cargo Bot

The only introductory programming concepts game that provided only a moderate level of feedback was the *BeadLoom Game*. In this game the player is told how many operations they have performed and are shown a visual representation of their input. Additionally, the player is told how optimal their solution is once they finish the puzzle in the form of a medal. However, the game doesn't actually tell the player which medal they achieved, instead it just puts their highest achievement next to the puzzles name.

Networking games

Of the games that taught networking, *Dox3d!* was the only game that provided the learner with a high amount of feedback. In *Dox3d!* player positioning and tiles with color coding provided most of the feedback making it simple and easy to understand. Orange tiles have been compromised and players vulnerability is determined by their positioning on the board. *The Orange Game Flash*, *Pinky's Pipes Pickle*, *CyberCIEGE* and the *Internet Peering Game* all provided the learner with a moderate amount of feedback. In *the Orange Game Flash* the player is told when they have completed a level, which level layouts they have completed and how many moves they have made. In *Pinky's Pipes Pickle* feedback is primarily provided in terms of numbers representing how much water is flowing through a pipe, but there is also a visual representation of the water flowing through the pipe, which increases in thickness the more water is flowing. Textual feedback is provided frequently in a number of ways in *CyberCIEGE*. The player is told what employees are thinking and planning on doing. Additionally, the player is given details on how successful their company is financially. Text bubbles appear above employees heads showing what the employee is saying, this along with some body language gives feedback on what the employee is doing and how happy they are. In the *Internet Peering Game* feedback is provided in the form of the players current balance and how many

territories on the grid they have access to.

Feedback is fairly limited in *Tablets of Stone*, but what feedback there is maybe fairly interesting. Messages the players receive tell them a fair amount about the security protocols.

Networking games *"Picture Logic the Puzzle Game"*, *Dead Man's Chest*, *Security Protocol Game* and *Cisco Mind Share* all offered the learner only low amounts of feedback. In *"Picture Logic the Puzzle Game"* the player is only given feedback on the correctness of their solution once they have actually completed the puzzle. However, the player is also given some information on how well they performed compared to other players. In *CTRL-ALT-HACK* the only feedback provided is 'hacker cred', a type of scoring system. Whereas in *Cisco Mind Share* feedback is provided only at the end of each stage telling the player how well they have done. In *Dead Man's Chest* the player can only see through a window a couple of Ts or discs. If the player clicks show cracker they get more feedback by being able to see all the current Ts and discs. Additionally, the game shows how many moves the player has made and what the binary equivalent of where they are up to.

Security Protocol Game feedback is fairly limited in the game, but what feedback there is may be valuable to the learner. Messages the players received tell them about the security protocols.

The level of feedback given in *the Orange Game* is unknown as this is left up to the person employing the activity.

Other games

Six of the games that were Computer Science games that did not specifically teach networking, binary or introductory programming concepts provided a high level of feedback to the learner, *Manufactoria*, *Sorting Bricks*, *Tour Finder Game*, *Tour Finder Game* Two Player, *Cisco myPlanNet*, *SimSE*. *Man-*

ufactoria provides rich feedback via the player being able to watch as their 'machine' is tested. Additionally, the player can choose to increase or decrease the speed of execution. This allows players to investigate the parts they are having trouble with closely, without wasting too much time. The machine is tested in a simple, but highly visual way. This makes the feedback easy to understand and accessible. The player is able to choose when they want feedback by pressing the play button, which tests their 'machine'. The player is also able to test their machine out on whatever input they choose instead of having the game show them ways in which it fails. The player in *Sorting Bricks* is shown which brick is 'heavier' once they click on a pair by a line between the two with an arrow pointing to the 'heavier' brick. As the player makes such a comparison a counter in the top right hand of the screen is incremented, so the player can see how many comparisons sorting the bricks took. Additionally, redundant lines are removed once the player has found the place for the brick. Finally when the player has sorted the bricks an animation plays to show they have solved the problem and the order of the bricks. *Tour Finder Game* and *Tour Finder Game, Two Player* provide rich feedback to the learner via the player being shown which city they have currently selected, the path they have made, the length of the path, how many cities in the path and, upon completion, how close they are to the optimal solution. Additionally, once the player has a solution they are able to view the optimal solution. The optimal solution also shows the differences between itself and the players solution. In *Cisco myPlanNet* the player is told how much money they are making, what they have researched and can research next, how happy their customers are and how many customers they have. Additionally, the customers send messages to the player telling them about their experiences with the services the player is offering. The player of *SimSE* is provided with in depth details about how the software is developing

such as how many bugs have been found. The player is also given information on each of their employees, some of the assets of the company and how much time has passed all of which is valuable feedback for the learner to process.

Some feedback is provided in *Swap Puzzle* as the player is able to watch their 'program' execute, step by step. Additionally, the player is presented with not only a visual representation of their moves, but a textual record describing exactly which moves they have made and alerting the player if they attempt to make an illegal move. The player is able to choose when they want feedback by pressing 'run your program'.

Battleships Unplugged has had less feedback for the learner but what it did provide was Valuable. This feedback came in the form of how many guesses it takes to complete each scenario.

Little feedback is provided by *Map Coloring*, the player is required to ensure they are following the rules themselves, although a teacher could assist in this role as well.

B.3.5 Difficulty adjustment provided

Binary games

Of the games that taught binary only two games *Binary Game* and *Binary Fun*, adjusted their level of difficulty automatically.

In *Binary Game* the game matches the difficulty level to the player's skill level. As the player progresses further into the game the difficulty of each challenge increases and, in the challenge mode, more challenges are presented before the time runs out, increasing the pace of the game. For the first few puzzles in *Binary Fun* the player is given some extra time before the countdown starts. However, this is the only way in which the game scales its difficulty level. In effect this means the game has only two difficulty settings.

Of the games that taught binary only five games, *Cross-Binary*, *Crossbin*

Puzzles, *CS Unplugged Binary Game*, *Binary Number Puzzle* and *Cisco Binary Game*, allowed the player to pick the level of difficulty. In the game *Cross-Binary* B.5, while it is not explicitly stated anywhere, the puzzles appear to have been put in order of difficulty. *Binary Number Puzzle* offers two difficulty settings, the hardest setting removes some scaffolding by no longer showing the place value of the bits. *CS Unplugged Binary Game* offers three difficulty settings. It appears that the higher difficulty settings increases the difficulty of the puzzles by a small margin. Additionally, the highest setting removes some scaffolding by no longer showing the place value of the bits. In *Crossbin Puzzles* there is a demo puzzle that is easier than the other two puzzles. Additionally, one could make more difficult puzzles by simply increasing their size. *Cisco Binary Game* matches the difficulty level to the player's skill level. As the player progresses further into the game the difficulty of each challenge increases and more challenges are presented, increasing the pace of the game.

No difficulty adjustment was provided in the games *Binary Maglock*, *Binary Crossnumber Puzzle* and *Binary Flashcards*.

Logic games

Of the games that taught introductory programming concepts four adjusted the difficulty level automatically: *ToonTalk 3*, *Light-Bot*, *Light-Bot 2.0*. *Blockly Maze*. As the player advances through puzzle mode in *ToonTalk 3* they are given more complex puzzles requiring new and sometimes more complicated tools. In *Light-Bot* the game starts with lower difficulty puzzles and increases them as the player continues. The player is not allowed to pick what order they solve the puzzles in, it is predetermined. This roughly matches difficulty to the players level of skill. However, the difficulty increase does not appear to be linear. Near the end of the game there is a significant increase in difficulty. *Light-Bot 2.0*'s puzzles are organized by the game into five categories. These categories are

basics, recursion, conditionals, expert and user created. It seems the game was intended to be played in that order as well. If the game is played in that order it starts with lower difficulty and increases as the player progresses, aside from the user created puzzles. The player is allowed to pick which category they play in, but the order of the puzzles within those categories is predetermined, aside from the user created puzzles. This roughly matches difficulty to the players level of skill. However, the difficulty increase does not appear to be linear. Near the end of the game there is a significant increase in difficulty. *Blockly Maze* puzzles are in increasing order of difficulty, introducing new more challenges concepts as you go.

Roborally, *RoboZZle*, *BeadLoom Game* and *Cargo Bot* allow the player to pick the level of difficulty. *Roborally* does this by offering multiple missions with different levels of difficulty. Additionally, the game is a multiplayer game so difficulty somewhat depend on the skill level of the people playing the game. Each puzzle in *RoboZZle* has a helpful difficulty gauge on right hand side. The difficulty is measured on a scale of one to five. The player may pick which puzzle they want to solve as they feel they are ready. *RoboZZle* even allows the player to sort the puzzles in order of difficulty. *BeadLoom Game* offers three difficulty settings, as well as a tutorial and games that don't have a given difficulty. The three difficulty levels are easy, medium and hard. *Cargo Bot* organizes its puzzles by difficulty, between easy, medium, hard, crazy and impossible, as well as providing a tutorial. The player may pick which puzzle they want to solve as they feel they are ready.

No difficulty adjustment is provided in *Brando the Egg Hunter* or *Treasure Hunter*

Networking games

Three of the games that taught networking, (*Pinky's Pipes Pickle*, *Cisco*

Mind Share, and *"Picture Logic the Puzzle Game"*), adjusted the difficulty level automatically. In *Pinky's Pipes Pickle* the puzzles are provided in order of complexity, which increases their difficulty a bit through out the puzzles. In *Cisco Mind Share* the player progresses through stages that sometimes rely on information learned in earlier stages. In this way the game increases its difficulty level as it progresses. *"Picture Logic the Puzzle Game"* offers increasingly difficult puzzles as the player progresses. Additionally, the player is able to pick which puzzle they play. In the description of each puzzle it tells the player how many other players successfully completed the puzzle, giving an indication of how difficult the puzzle is.

In *Security Protocol Game* and *"Picture Logic the Puzzle Game"* (as mentioned above) the player can pick difficulty level. *Security Protocol Game* provides the opportunity for self selection depending on which security protocol is picked, the game will change the likelihood of various parties succeeding dramatically.

No adjustment for difficulty is provided in *CyberCIEGE*, *the Orange Game* or *the Orange Game Flash*. In *the Orange Game* and *the Orange Game Flash* the closest thing to difficulty settings offered are different layouts. However, it is not clear that any layout is more difficult than any other.

Dead Man's Chest's difficulty only changes slightly, making progress gets a bit more difficult after the first few steps.

Tablets of Stone doesn't appear to vary in difficulty much and if it does vary it will most likely depend on the players involved.

Internet Peering Game, *Dox3d!* and *CTRL-ALT-HACK* are all fundamentally multi player games where the difficulty level is based off other players. This means that the level of challenge in the game is determined by who you are playing with or against, which is to a certain extent under your control.

Other games

The Computer Science games that adjusted their difficulty level automatically but did not teach networking, binary or introductory programming concepts were *Manufactoria* and *Sorting Bricks*. In *Manufactoria* the player may only play more difficult puzzles once they have solved the simpler ones. Similarly in *Sorting Bricks* the game offers the problems in order of their complexity.

Swap Puzzle organizes its puzzles by difficulty, with each puzzle being more difficult than the last. The player may pick which puzzle they want to solve as they feel they are ready.

Players can pick the level of difficulty in *Cisco myPlanNet*, *Tour Finder Game* and *Tour Finder Game, Two Player*.

The game *Cisco myPlanNet* increases in difficulty as it goes on, mostly at the player's discretion. This in effect allows the player to match the difficulty of the game to their own skill level. However, it is also possible for the player to play the game such that it is too hard or easy for them. Additionally, the game offers easy, medium and hard difficulty settings. *Tour Finder Game* and *Tour Finder Game, Two Player* enable the player to increase or decrease the number of cities, and thus the difficulty, but doing so will start a new problem. The ability to increase the size of the problem is limited.

No difficulty adjustment was provided in *SimSE*, *Map Coloring* or *Battleships Unplugged*. Although for *Map Coloring* more difficult maps could be generated.

B.3.6 Clear goals

Binary games

Of the games that taught binary, *Cross-Binary*, *Binary Game*, *Crossbin Puzzles*, *Binary Crossnumber Puzzle*, *Cisco Binary Game*, *Binary Fun* and *CS Unplugged Binary Game* kept the goals clear and simple. From reading the

instructions and meeting the suggested prerequisites in *Cross-Binary* the goals are very clear, simple and consistent. The player must fill in the puzzle like they would a crossword. However, instead of entering words they must enter binary representations of the decimal numbers given. In *Binary Game* the player must solve as many challenges as possible as fast as possible.

From reading the instructions in *Crossbin Puzzles* and meeting the suggested prerequisites, the goals are very clear, simple and consistent. The player must fill in the puzzle like they would a crossword. However, instead of entering words they must enter binary representations of the hexadecimal numbers given. *Binary Crossnumber Puzzle* requires the player to figure out the hints and fill all the squares with the corresponding binary numbers.

The player of *Cisco Binary Game* must solve as many challenges as possible without letting the screen fill up with unsolved challenges. Similarly in *Binary Fun* the player must finish as many rounds as possible in a row.

CS Unplugged Binary Game explains what the player needs to do for every puzzle. Additionally, the goal is the same in every puzzle, enter the decimal value of the given binary number.

However, in *Binary Maglock* the tutorial is confusing and it is not clear what some actions taken are leading towards. *Binary Flashcards*, *Binary Number Puzzle* require the player to have prior knowledge and understanding of binary numbers (in *Binary Number Puzzle*) or base ten conversion to binary (in *Binary Flashcards*) in order for the goals to be clear. perhaps the intention was reinforcement of this concept rather than teaching new skills and knowledge in binary.

Logic games

Of the games that taught introductory programming concepts *Roborally*, *RoboZZle*, *Light-Bot*, *Light-Bot 2.0*, *Brando the Egg Hunter*, *Cargo Bot*, *Trea-*

sure Hunter and *Blockly Maze* the goals of the game were clear and simple. In *Roborally* players need to reach each flag in the game in the right order to win. In *RoboZZle* the goal of each puzzle is clear and simple: collect all the stars. In every *Light-Bot* and *Light-Bot 2.0* puzzle the players only goal is to light up every blue square keeping things simple and consistent for the player. This is also the case in *Brando the Egg Hunter* where in every puzzle the players only goal is to collect all the eggs. In *Blockly Maze* the goal is again simple, to reach the flag at the end of the maze. Similarly the goal is to collect all of the treasure in *Treasure Hunter*. The only room for confusion in *Treasure Hunter* is the introduction of the padlocks. The padlocks are optional trivia questions, but they look like they could be treasure. The goal in *Cargo Bot* is always to replicate the diagram shown at the top of the screen by moving cargo with a claw.

In the puzzle mode of *ToonTalk 3* an in-game character describes the goals of each puzzle when the puzzle begins, making the goals fairly clear. However, the way to achieve these goals is not always made so clear. *BeadLoom Game* has goals that are not so transparent. The player must make a picture that matches the provided one, that much is clear. It is a little bit less clear that the player will also be rewarded for solving the puzzle using as few operations as possible. The player could quite easily think the rewards are being given based off speed of completion or not notice the rewards whatsoever.

Networking games

The following networking games had a high level of clarity of game goals or objectives: *the Orange Game*, *Tablets of Stone*, *Internet Peering Game*, *Security Protocol Game*, *Dox3d!*, *CTRL-ALT-HACK* and Picture logic. In *the Orange Game* the goals are clear simple and consistent, provided they were adequately explained prior to engagement. The goal in *Internet Peering Game* is easily

relatable, simply to end up with the most money at the end of the game. *"Picture Logic the Puzzle Game"*, as its name suggests, requires the player to solve each puzzle by drawing the required picture. *Security Protocol Game* has clear, simple goals for each group involved. In *Dox3d!* the goal is that the players have to work together to find specific assets inside a network without being detected. In *CTRL-ALT-HACK* the goal of the game is clear and simple, maximize your 'hacker cred'. In *Tablets of Stone* the goal is clear and simple, transmit your message securely within the restrictions given.

CyberCIEGE was also clear, but not necessarily simple to use. For each scenario objectives are given at the start in a text box. Additionally, there is always an objectives tab offering information on the players current objectives. Whereas the networking games *Cisco Mind Share* and *the Orange Game Flash* were simple, but clarity could have been improved upon. At each stage of *Cisco Mind Share* the player must simply move the right answer into the right spot. However, how to figure out the right answer is not always clear. Similarly in *the Orange Game Flash* the goals were again simple and consistent, but the game itself didn't explain them much. The game does give some indication of what the player must do, by labeling where each piece of fruit must be in order for the player to win.

The goal of *Dead Man's Chest* was not simple, even after reading the instructions it was still a little unclear. For *Pinky's Pipes Pickle* the goals are not particularly clear. The only way to know what the goal is is to read the textual description of how to play. Additionally, the player needs to find the optimal solution, which is perhaps not obvious when the problem is that the player has redundant water flowing.

Other games

Of the Computer Science games that did not specifically teach networking,

binary or introductory programming concepts *Battleships Unplugged*, *Manufactoria*, *Map Coloring*, *Tour Finder Game* and *Tour Finder Game, Two Player* all had clear and simple goals for the player to achieve in the game. The goal in *Battleships Unplugged* is clear and simple, as per traditional battleships, find your partners battleship In every puzzle in *Manufactoria* the goal is to accept only certain input patterns. Each puzzle states the types of patterns that should be accepted in English. This is normally unambiguous, but sometimes leads to confusion. In *Tour Finder Game* and *Tour Finder Game, Two Player* the player must simply try and find the shortest path, with the appropriate constraints

The main goal in *Cisco myPlanNet* is clear, research as many technologies as possible before the time runs out. However, to achieve this is somewhat complex and requires completing many subgoals which are a bit less clear. The game suggests certain subgoals to the player. Each of these suggestions tells the player how to achieve the subgoal, however, perhaps through a bug, the game would not tell me how far I had already gone towards achieving most subgoals. The goal in *Map Coloring* is simply to color in each region of the map, with only a restricted set of colors to choose from, without having any two adjacent regions share a color.

Sorting Bricks has a general goal of solving each problem using the least amount of comparisons. However, the main goal of each problem is clearly to learn and put into practice the algorithm, which will normally result in a lower amount of comparisons. If one were to only focus on the first goal, cheating the game would be easy. The game allows the player to undo each comparison with no disadvantage. This means the player can learn the order, undo all their comparisons and simply sort the bricks from memory. Conversely *SimSE* is often ambiguous it is often unsure what the player should be doing and what the primary goals are.

The goal of each puzzle from *Swap Puzzle* is simple, but not clear. The player is expected to swap the blue balls with the red balls opposite them. However, this is only described in the introductory text, which is reasonably long meaning players may be inclined to ignore it.

B.3.7 Encouraging intensely focused concentration

Binary games

Of the games that taught binary *Binary Game*, *Binary Maglock*, *Binary Crossnumber Puzzle*, *Cisco Binary Game* and *Binary Fun* encouraged a high level of concentration from the student. Progressing through the game in *Binary Maglock* is not practical without focused concentration, but the player is unlikely to understand what to do without reading the tutorial, which teaches passively and as such is less likely to encourage focused concentration. In *Binary Crossnumber Puzzle* it is not practical to solve the puzzle without focused concentration. In the *Cisco Binary Game* the amount of focused concentration required starts off fairly low, but it increases drastically the closer the player gets to losing. This is because the game presents more complicated challenges the longer the game continues. Additionally, the player will need to concentrate on solving problems faster if they are close to losing. In contrast in the game *Binary Fun* the concentration level required is high throughout. If the player is not concentrating the small amount of time allowed, eight seconds, is likely to be insufficient to get very far into the game at all.

A moderate level of concentration was encouraged in *Cross-Binary*, *Crossbin Puzzles*, *CS Unplugged Binary Game* and *Binary Number Puzzle*. The amount of focused concentration required in *Binary Game* starts off as moderate, but it increases the closer the player gets to losing. This is because the game presents more complicated challenges the longer the game continues. Additionally, the player often needs to concentrate on solving problems faster to achieve a bet-

ter score. In *Cross-Binary* some concentration is encouraged. The player will find which binary number goes where much quicker if they focus. Whereas in *Crossbin Puzzles*, with enough practice the conversion may no longer require concentration. However, for most players the game will require a reasonable amount of concentration in order to complete a puzzle. *CS Unplugged Binary Game* requires a moderate amount of concentration. The player is given unlimited time to solve problems of varying difficulty. At the highest difficulty level most players will need to concentrate to some degree to solve each puzzle. *Binary Number Puzzle* also requires a moderate amount of concentration by the player. The player is given unlimited time to solve randomised problems. At the highest difficulty level most players will need to concentrate to some degree to solve each puzzle. Additionally, the game has a time limit in which the player must solve as many problems as possible. In order to solve a greater amount of problems within the time limit focused concentration will be required. Conversely *Binary Flashcards* encouraged players to contribute only a low level of concentration. In fact, nothing is done to encourage the player to concentrate on the problem. The player is given an unlimited amount of time to solve relatively simple problems. In fact the game actually distracts the player, the background of the game is a series of distracting images and the game plays an unrelated and even more distracting song. It can be difficult to focus on the game without turning your sound off.

Logic games

Of the Logic games investigated a high level of concentration was encouraged in *ToonTalk 3*, *Roborally*, *RoboZZle*, *Cargo Bot*, *Light-Bot*, *Light-Bot 2.0* and *Blockly Maze*. Progress throughout puzzle mode in *ToonTalk 3* would be difficult without some degree of concentration. In *RoboZZle*, *Light-Bot*, *Light-Bot 2.0* and *Cargo Bot* most of the puzzles would most probably be unsolvable without

focused concentration. Likewise in *Roborally* players that do not focus on being successful during the game are extremely unlikely to succeed.

A moderate level of concentration is encouraged in *Brando the Egg Hunter* and *BeadLoom Game*. In *Brando the Egg Hunter* a player could guess their way through several of the puzzles, but some require a bit of concentration to complete. However, in *BeadLoom Game* some concentration is required for achieving higher, but not to simply solve the puzzle. This is because most puzzles have simple solutions as well as complex ones. Low levels of concentration are likely to lead to simple solutions, while higher levels of concentration are likely to lead to more complex solutions. The player is normally rewarded for finding more complex solutions.

In *Blockly Maze* and *Treasure Hunter* very little concentration is required. A player could guess their way through most, if not all of the puzzles.

Networking games

The Networking games that encouraged a high level of concentration were *Internet Peering Game*, *Dead Man's Chest* and *"Picture Logic the Puzzle Game"*. Proceeding in the *Internet Peering Game* may be difficult without focused concentration as negotiation is involved with other players. *"Picture Logic the Puzzle Game"* also requires a high level of concentration, it would be almost impossible to get very far in this game without it. *Dead Man's Chest* is designed such that trying to complete it without focused concentration would be likely to take a long time.

CyberCIEGE, *Cisco Mind Share*, *Tablets of Stone*, *Pinky's Pipes Pickle*, *the Orange Game*, *Security Protocol Game* and *Dox3d!* encourage a moderate level of concentration from the player. To solve most problems in *CyberCIEGE* the player must simply read some text and apply the appropriate solution. While in some cases this will mean the player cannot advance without focused concentra-

tion it is also unlikely that the player to want to read a lot of text. It is possible to complete the game by guessing in *Cisco Mind Share* but it would take a lot longer than a player who focused. Similarly *the Orange Game* would be very time consuming to complete the game without a plan. The game *Dox3d!* requires a moderate amount of concentration to play, but a higher level of concentration is encouraged by providing the player with a more enjoyable experience for it. In Security Protocol and *Tablets of Stone* if a player doesn't focus, the games are likely to fall to bits pretty quickly. However, this may not be particularly obvious to the players until it happens. *Pinky's Pipes Pickle* is easy enough to play without concentration, but finding the optimal solution as such is unlikely.

The Orange Game Flash does not require much in the way of focus. The player will most likely stop playing before any real concentration would be required to improve their performance. *CTRL-ALT-HACK* could be completed without much concentration once the rules have been learned, as all that is required is figuring out what you need to roll against and then rolling some dice.

Other games

Of the Computer Science games that did not specifically teach networking, binary or introductory programming concepts a high level of concentration was encouraged in both *SimSE* and *Manufactoria*. Where the game *Manufactoria* is difficult and most puzzles could not be solved without some periods of focused concentration, *SimSE* required focused concentration in order to achieve better results. A moderate level of concentration encouraged in both *Cisco myPlanNet* B.6 and *Battleships Unplugged*. The game *Cisco myPlanNet* encourages concentration for the most part. However, unfortunately, it actually discourages concentration on the educational parts of the game. The game is time

critical and reading the text that teaches you about the history of the Internet is optional and time consuming. *Battleships Unplugged* could be played simply by guessing, but in order to do well in the second or third scenario at least some focused concentration is required.

Map Coloring may be tempting to tackle without much concentration, but it is likely to take too long to win with this approach, in which case it will quickly become apparent that some concentration is required.

Solving the larger puzzles from *Swap Puzzle* without focused concentration could take a very long time, but this may not be obvious from the outset and as such players may be inclined, at least initially, to play lazily.

Sorting Bricks, *Tour Finder Game* and *Tour Finder Game, Two Player* only encouraged a low level of concentration, the player will be able to find good solutions, and sometimes optimal ones, with very little effort or focus. Likewise in the game *Sorting Bricks*, the player could play through the game without focusing. It would take them longer and they would be unlikely to learn anything.

B.3.8 Level of player control

Binary games

Cross-Binary, *Crossbin Puzzles* and *Binary Fun* all allowed the player a high level of control. The rules of *Cross-Binary* and *Crossbin Puzzles* state that the player must simply convert the numbers into binary and enter them into boxes. This would suggest the player has little control. However, the player is able to pick their puzzle, the order they solve the problems in, and how they attempt to solve the puzzle. The player even has access to the answers, so they can 'cheat' if they wish. In *Binary Fun* the player has just enough control to solve the challenges within the time limit.

Binary Game, *Binary Maglock*, *Binary Crossnumber Puzzle*, *Cisco Binary*

Game, *Binary Flashcards*, *CS Unplugged Binary Game* and *Binary Number Puzzle* allow the player a moderate level of control. In *Binary Crossnumber Puzzle* the player is limited to entering the solutions to each hint into the appropriate boxes, suggesting that the player has very limited control. However, the order in which the player attempts to solve each hint is only partially restricted, as is the process in which the player solves the problem, which gives the player the impression of control. In *Binary Game* and *Cisco Binary Game*, the player can control the game just enough to solve the challenges in a timely manner. Probably the biggest issue with controlling the *Cisco Binary Game* is entering decimal numbers on the number pad, which is slow and awkward. In *Binary Maglock* the player can enter 0s and 1s to make a binary number, starting from right and going to the left. Additionally, the player can delete their mistakes. In *Binary Flashcards* the player is given enough control to solve the puzzles, but no control over the puzzles or their difficulty. In *CS Unplugged Binary Game* the player is given enough control to solve the puzzles in the form of entering text into a box as well as some control over the game's difficulty level. In *Binary Number Puzzle* the player is given enough control to solve the puzzles in the form of entering text into a box. Additionally, the player has some control over the game's difficulty level.

Logic games

Brando the Egg Hunter, *BeadLoom Game*, *ToonTalk 3* and *Treasure Hunter* all give the player a high level of control. *ToonTalk 3* achieves this level of control by letting the players, to a certain extent, reprogram the world around them. In *Brando the Egg Hunter* and *Treasure Hunter* the player has perhaps even too much control, making them less challenging. Other games of a similar sort use restrictions on how much control the player has to increase the difficulty level. In *BeadLoom Game* the player has a great deal of control in the form of several

functions that draw a variety of shapes. The player inputs the parameters of these functions and can preview the output before drawing it. Some of these functions, such as linear and triangular iteration, are relatively complex. This allows the player a great amount of flexibility and control. Additionally, the game allows the player to play any puzzle at any point.

RoboZZle, *Light-Bot*, *Light-Bot 2.0*, *Cargo Bot* and *Blockly Maze* each give the player a moderate level of control. In *RoboZZle*, *Lightbot* and *Lightbot 2.0* the player controls their robot by programming it with commands. The player is given a variable amount of slots to put commands in. However, the player is sometimes also allowed to use functions, also with a limited amount of slots, to increase the number of commands they can execute, using methods such as recursion the player create infinite loops with functions. In effect this means the player can have a lot of control, if they are clever and given enough slots. However, the challenge of most puzzles is that they restrict the amount of commands the player can give the robot, which reduces the player's control. In *Cargo Bot* the player controls their robot claw by programming it with commands. The player is allowed to use a limited number of functions, each with a limited amount of slots. Using methods such as recursion the player can create infinite loops with their commands. In effect this means the player can have a lot of control, if they are clever. In *Roborally* the player issues five commands to their robot at the start of each turn. An undamaged robot gets nine possible commands to pick from at random. However, this amount decreases as the robot accumulates damage. However, even when the player gets less choice over their commands they still have the option to power down, removing all their damage. In *Blockly Maze* the player controls their avatar by programming it with commands. The player is given a set amount of commands to use. However, the player is sometimes also allowed to use loops, to increase the number

of commands they can execute. In effect this means the player can have a lot of control, if they are clever and given enough commands. However, the challenge of most puzzles is that they restrict the amount of commands the player can give the avatar, which reduces the player's control.

Networking games

CyberCIEGE, *"Picture Logic the Puzzle Game"*, *CTRL-ALT-HACK* and *Dox3d!* all afford the player a high level of control. *CyberCIEGE* has a lot of options and a lot of ways in which the player can control the environment. However, this is somewhat to the detriment of the game as the interface has a tendency to be very confusing and easy to get lost in. In *"Picture Logic the Puzzle Game"* the player able to make incorrect moves, mark spaces as not feasible and can pick more or less difficult puzzles. In *CTRL-ALT-HACK* the player is able to trade missions, skip debriefings (offering some benefit, but preventing trade) and play cards that influence the game. In *Dox3d!* the player has a lot of control of their actions each turn and is given a high amount of decisions to make.

Pinky's Pipes Pickle, *Internet Peering Game*, *Security Protocol Game* and *Tablets of Stone* each provide the player a moderate level of control. In *Pinky's Pipes Pickle* the player is simply given the ability to increase or decrease the flow of water through a pipe. In *Internet Peering Game* while the players options for expansion are limited to a certain extent, they have a lot of freedom in their negotiations with other players. If in *Security Protocol Game* the security protocol is prescribed to the players then only one party gets any control at all (the party trying to break the protocol). If the players are allowed to design their own protocol then they get at least that much control. In *Tablets of Stone* most players are given a reasonable amount of freedom, given only the restrictions on how they can pass messages, but there are a few players who are not given any

control at all.

In *Cisco Mind Share*, *the Orange Game*, *the Orange Game Flash* and *Dead Man's Chest* the player has a low level of control. In *Cisco Mind Share* the player simply picks answers from a list and drags them into the appropriate spots. In *the Orange Game* and *the Orange Game Flash* the challenge of the game revolves around limiting the player's control, meaning the player has a fairly limited amount of control. However, the player(s) is still able to control things like level layout freely. In *Dead Man's Chest* B.7 the player is able to move the slider left and right, so long as a T is not stopping it and swap discs for Ts or Ts for discs.

Other games

Cisco myPlanNet, *SimSE*, *Sorting Bricks*, *Tour Finder Game* and *Tour Finder Game, Two Player* all allow the player a high level of control. In *Cisco myPlanNet* the player is able to advance technology in a different way than what happened historically, the player can choose what services to provide, how much to charge for them and the player is also in complete control of purchasing and using the hardware required to deploy these services. In *SimSE* the main focus of the game is the player giving tasks to employees for them to complete. The player is given complete freedom in which tasks they give to which employee. Additionally, the player is given information about each employee's experience in each area, which helps the player make informed decisions. In *Sorting Bricks* the player able to make incorrect moves, move bricks as they please, compare bricks in more than one way and can pick more or less complex problems. In *Tour Finder Game* and *Tour Finder Game*, two player the player can solve the problem by linking cities, undo every step one by one, start problems of the same, greater or lesser difficulty and view the optimal solution.

Battleships Unplugged, *Map Coloring*, *Swap Puzzle* and *Manufactoria* each

provide the player a Moderate level of Control. In *Battleships Unplugged* the player is free to guess whichever number they'd like, as well as pick what number their is battleship. However, this is the full extent of their control. In *Map Coloring* the player is given the ability to choose what color to put where completely freely at first, with the only restrictions relating directly to their initial choices. In *Swap Puzzle* the player can move balls into empty slots left or right. Additionally, the player may have a ball 'jump' over a single other ball. The player is also given the freedom to have their game be replayed after they have solved the puzzle. In *Manufactoria* the player builds their machine in a fixed amount of space. One of the main challenges of each puzzle is building the machine within the confined amount of space. In effect this means the player's level of control is related to how cleverly they can compress their machine.

B.3.9 Allows the player to build confidence over time

Binary games

Binary Game, and *Cisco Binary Game* both allowed the player to build confidence over time by starting with a lower level of difficulty and increasing it over time. The difficulty levels of the *Binary Game* and the *Cisco Binary Game* are moderate at first, allowing most players with no experience with binary to jump right in. People with more experience with binary will also find a challenge once they have played through the first, slower part.

Cross-Binary, *Crossbin Puzzles*, *Binary Fun*, *CS Unplugged Binary Game* and *Binary Number Puzzle* each provided an increase of difficulty, to a limited extent, meaning the game may have some success at helping the player build confidence over time. For *Cross-Binary* the difficulty level appears to increase throughout the provided puzzles. For *Crossbin Puzzles* the provided puzzles difficulty only increases between the demo puzzle and the others. In *Binary Fun* the difficulty level only increases throughout the first few puzzles, after which

the difficulty level is fixed. In *Binary Number Puzzle* and *CS Unplugged Binary Game* the difficulty level does not automatically increase. However, difficulty does increase between settings gradually. However, in *Binary Number Puzzle* the increase is not particularly large. In the easier difficulty mode counting the higher number of dots in impractical given the time limit. If the place value of the bits was represented numerically the difference between difficulty levels may have been larger.

For each of *Binary Crossnumber Puzzle*, *Binary Flashcards* and *Binary Maglock* there does not appear to be any increase in difficulty, meaning the games are unlikely to help the player build confidence over time.

Logic games

RoboZZle, *Light-Bot*, *Light-Bot 2.0*, *Cargo Bot*, *BeadLoom Game* and *Blockly Maze* each allowed the player to build confidence over time by starting with a lower level of difficulty and increasing it over time. In the tutorial of *RoboZZle* the level of difficulty is such that it encourages player confidence. Additionally, *RoboZZle* offers a campaign that gradually increases difficulty over 143 puzzles. However, the player may at any point choose to play any puzzle they want, meaning the gradual increase of difficulty is not forced on the player. *Light-Bot*'s difficulty increases throughout the game at a rate that helps the player gain confidence, except near the end where the difficulty spikes. If the player plays the levels in order then *Light-Bot 2.0*'s difficulty increases throughout the game at a rate that helps the player gain confidence. Except near the end where the difficulty spikes, but these puzzles are labeled as expert puzzles, so this is not surprising. *Cargo Bot* and *BeadLoom Game* offer levels in an order that gradually increase in difficulty. However, the player may at any point choose to play any puzzle they want, meaning the gradual increase of difficulty is not forced on the player.

While *Blockly Maze* is a short game it seems to encourage player confidence naturally by gradually increasing the difficulty level. Although the last puzzle seems to be an intentionally larger step up in difficulty

ToonTalk 3 and *Roborally* each provided an increase of difficulty, to a limited extent, meaning the game may have some success at helping the player build confidence over time. In the puzzle mode of *ToonTalk 3* the difficulty level does increase, but is kept relatively low. However, it is likely that this was intentional as the game appears to be targeted at a younger audience. In *Roborally* the difficulty level is decided largely by the mission chosen by the players. However, the missions are clearly marked by difficulty.

For both of *Brando the Egg Hunter* and *Treasure Hunter* there does not appear to be any increase in difficulty, meaning the games are unlikely to help the player build confidence over time. The difficulty level of *Brando the Egg Hunter* did not increase in any way related to Computer Science. However, the geometry of the shapes does get a bit more complicated.

Networking games

"Picture Logic the Puzzle Game" allowed the player to build confidence over time by starting with a lower level of difficulty and increasing it over time. However, the puzzles quickly become large which could put some players off.

Cisco Mind Share, *CTRL-ALT-HACK*, *Internet Peering Game*, *Dox3d!* and *Pinky's Pipes Pickle* each provided an increase of difficulty, to a limited extent, meaning the game may have some success at helping the player build confidence over time. In *Cisco Mind Share* the player progresses through stages that sometimes rely on information learned in earlier stages. In this way the game increases its difficulty level as it progresses. *CTRL-ALT-HACK* is a competitive multiplayer game, meaning the difficulty is largely based off who you play with. In *Internet Peering Game* the difficulty level is almost entirely dependent on

the players themselves. The difficulty of *Dox3d!* will vary based off the people played with. However, the player is likely to be able to survive long enough to gain confidence in the game. In *Pinky's Pipes Pickle* the puzzles are provided in order of complexity, which increases their difficulty a bit through out the puzzles. The level of difficulty begins low and seems to stay relatively low.

For each of *CyberCIEGE*, *Dead Man's Chest*, *the Orange Game*, *Security Protocol Game*, *Security Protocol Game*, *Tablets of Stone* and *the Orange Game Flash* there does not appear to be any increase in difficulty, meaning the games are unlikely to help the player build confidence over time. For *Dead Man's Chest* the difficulty only increases a bit near the start. Additionally, the size of the increase near the start is probably inappropriately large to help the player gain confidence.

Other games

Cisco myPlanNet, *Sorting Bricks*, *Swap Puzzle*, *Tour Finder Game* and *Tour Finder Game, Two Player* each allowed the player to build confidence over time by starting with a lower level of difficulty and increasing it over time.

Cisco myPlanNet's difficulty level is determined by the difficulty setting the player chooses and the way the game is played. This makes it natural for the player to play the game in such a way that the difficulty level makes them feel confident. However, it is possible the player will make the game too hard for themselves and lose or give up. In *Sorting Bricks* the difficulty of each problem increases slowly, but the player still ends up using quick sort. *Swap Puzzle* offers four puzzles in order of increasing difficulty, but the player is free to play whichever one they choose. *Tour Finder Game* and *Tour Finder Game, Two Player* do not automatically increase in difficulty. However, the player is allowed a large amount of control over the difficulty of the game. This allows the player to gradually increase the difficulty at a rate that helps them gain confidence if

they choose too.

Manufactoria provides an increase of difficulty, to a limited extent, meaning the game may have some success at helping the player build confidence over time. While the game's difficulty level does gradually increase, it quickly gets difficult. This reduces the games ability to help to player gain confidence. This could result in some players giving up. This is made even worse because many of the educational concepts aren't covered until at least mid way through the game.

For each of *SimSE*, *Battleships Unplugged* and *Map Coloring* there does not appear to be any increase in difficulty, meaning the games are unlikely to help the player build confidence over time.

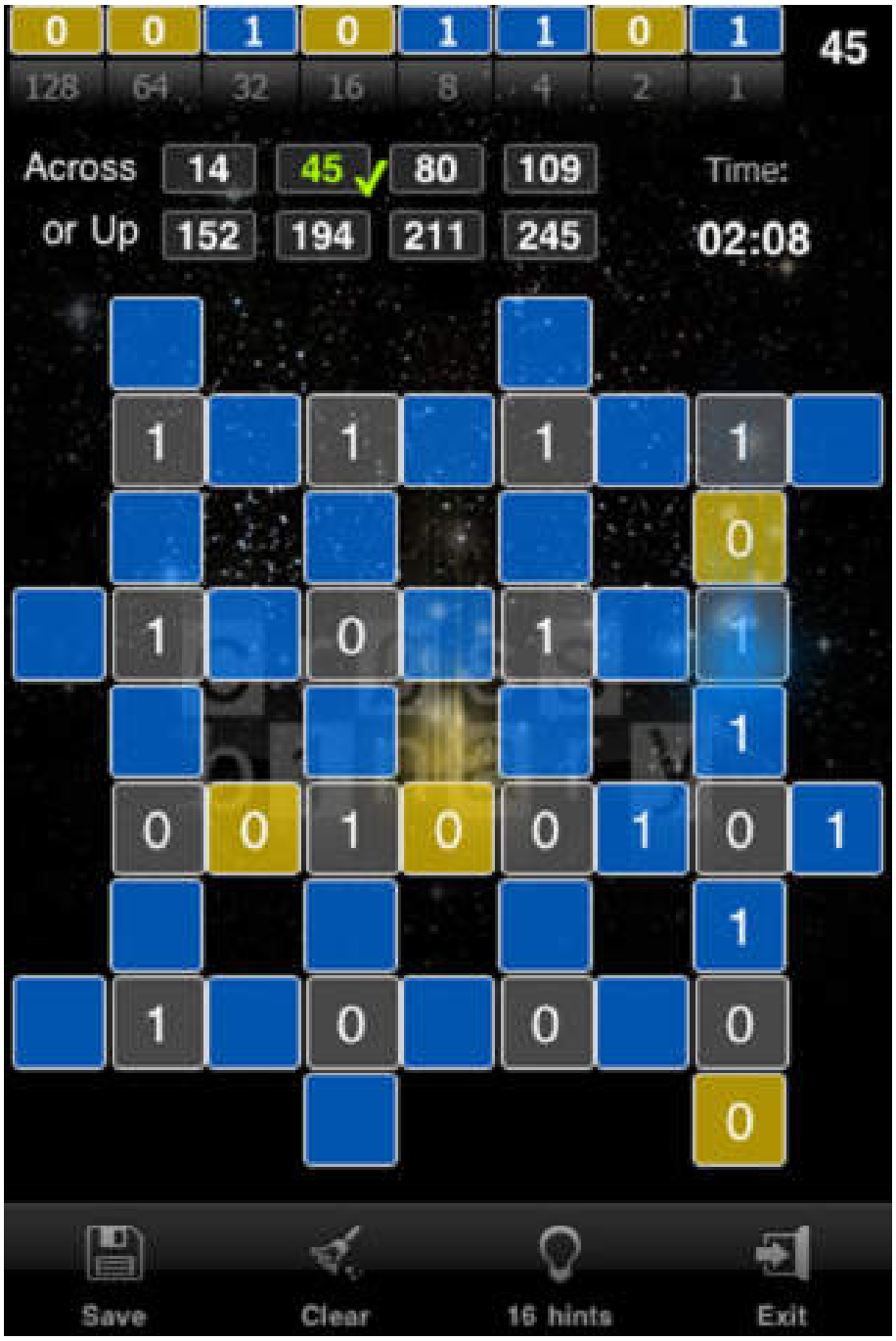

Fig. B.5: Cross-Binary

Fig. B.6: Cisco myPlanNet

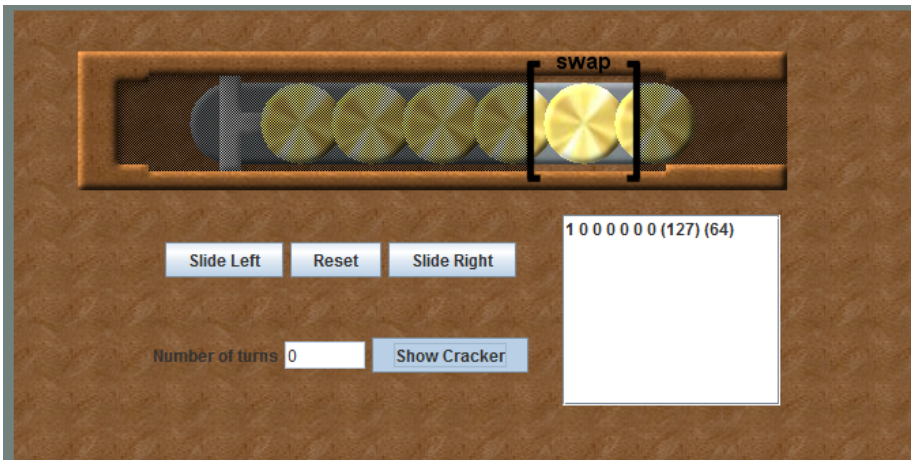

Fig. B.7: Dead Man's Chest

REFERENCES

[1] Albert Bandura and Robert Wood. Effect of perceived controllability and performance standards on self-regulation of complex decision making. *Journal of Personality and Social Psychology*, 56(5):805–814, 1989.

[2] Acey Boyce and Tiffany Barnes. BeadLoom Game: using game elements to increase motivation and learning. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 25–31, New York, NY, USA, 2010. ACM.

[3] Margot Brereton, Jared Donovan, and Stephen Viller. Talking about watching: using the video card game and wiki-web technology to engage IT students in developing observational skills. In *Proceedings of the fifth Australasian conference on Computing education - Volume 20*, ACE '03, pages 197–205, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[4] K G Brown and J K Ford. Using computer technology in training: Building an infrastructure for active learning. In K Kraiger, editor, *Creating, implementing, and maintaining effective training and development: State-of-the-art lessons for practice*, pages 192–233. Jossey-Bass, San Francisco, 2002.

[5] Roger Caillois. *Les jeux et les hommes: le masque et le vertige*, volume 184. Editions Gallimard, 1991.

[6] John Dekkers and Stephen Donatti. The Integration of Research Studies on the Use of Simulation as an Instructional Strategy. *The Journal of Educational Research*, 74(6):pp. 424–427, 1981.

[7] Michael Eagle. Level up: a frame work for the design and evaluation of educational games. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 339–341, New York, NY, USA, 2009. ACM.

[8] Michael Eagle and Tiffany Barnes. Wu's castle: teaching arrays and loops in a game. *SIGCSE Bull*, 40:245–249, 2008.

[9] Michael Eagle and Tiffany Barnes. Evaluation of a game-based lab assignment. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 64–70, New York, NY, USA, 2009. ACM.

[10] Michael Eagle and Tiffany Barnes. Experimental evaluation of an educational game for improved learning in introductory computing. *SIGCSE Bull.*, 41(1):321–325, March 2009.

[11] Simon Egenfeldt-Nielsen. Third Generation Educational Use of Computer Games. *Journal of Educational Multimedia and Hypermedia*, 16:263–281, 2007.

[12] Rosemary Garris, Robert Ahlers, and James E Driskell. Games, motivation, and learning: A research and practice model. *Simulation & gaming*, 33(4):441–467, 2002.

[13] James Paul Gee. What video games have to teach us about learning and literacy. *Comput. Entertain.*, 1(1):20, October 2003.

[14] James Paul Gee. Learning by design: Games as learning machines. *Interactive Educational Multimedia*, 8:15–23, 2004.

[15] Benjamin Gibson and Tim Bell. Evaluation of games for teaching Computer Science. In *8th Workshop in Primary and Secondary Computing Education*, Aarhus, Denmark, 2013. WiPSCE.

[16] Ray Giguette. Pre-games: games designed to introduce CS1 and CS2 programming assignments. *SIGCSE Bull.*, 35(1):288–292, January 2003.

[17] Victoria Guillén-Nieto and Marian Aleson-Carbonell. Serious games and learning effectiveness: The case of Its a Deal!. *Computers & Education*, 58(1):435–448, 2012.

[18] Mario Guimaraes, Huwida Said, and Richard Austin. Using video games to teach security. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, ITiCSE '11, page 346, New York, NY, USA, 2011. ACM.

[19] Robert T Hays. The effectiveness of instructional games: A literature review and discussion. Technical report, DTIC Document No. NAWCTSD-TR-2005-004, Naval air warfare center training systems div., Orlando Fl., 2005.

[20] John M D Hill, Clark K Ray, Jean R S Blair, and Curtis A Carver Jr. Puzzles and games: addressing different learning styles in teaching operating systems concepts. *SIGCSE Bull.*, 35(1):182–186, January 2003.

[21] J-C. Hong, C-L. Cheng, M-Y. Hwang, C-K. Lee, and H-Y. Chang. Assessing the educational values of digital games. *Journal of Computer Assisted Learning*, 25:423–437, 2009.

[22] D H Jonassen. Integration of problem solving into instructional design. In R Reiser and J Dempsey, editors, *Trends and issues in instructional design*

and technology, pages 107–120. Merrill/Prentice Hall,Upper Saddle River, 2002.

[23] Peter Komisarczuk and Ian Welch. A board game for teaching internet engineering. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 117–123, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[24] June Lee. Effectiveness of computer-based instructional simulation: A meta analysis. *International Journal of Instructional Media*, 26(1):71–85, 1999.

[25] Thomas W Malone. Toward a theory of intrinsically motivating instruction. *Cognitive science*, 5(4):333–369, 1981.

[26] Lasse Natvig and Steinar Line. Age of computers: game-based teaching of computer fundamentals. *SIGCSE Bull.*, 36(3):107–111, June 2004.

[27] Emily Oh Navarro and André van der Hoek. Sim{SE}: an educational simulation game for teaching the Software engineering process. *SIGCSE Bull.*, 36(3):233, June 2004.

[28] Emily Oh Navarro and André van der Hoek. Multi-site evaluation of {S}im{SE}. *SIGCSE Bull.*, 41(1):326–330, March 2009.

[29] Diana G. Oblinger. The Next Generation of Educational Engagement. *Journal of Interactive Media in Education*, 2004:1–18, 2004.

[30] Mehran Sahami, Steve Roach, Ernesto Cuadros-Vargas, and Richard LeBlanc. {ACM/IEEE-CS} computer science curriculum 2013: reviewing the {I}ronman report. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 13–14, New York, NY, USA, 2013. ACM.

[31] Katie Salen and Eric Zimmerman. *The game design reader*. MIT press, Cambridge, 2006.

[32] Scott Schaefer and Joe Warren. Teaching computer game design and construction. *Computer-Aided Design*, 36:1501–1510, 2004.

[33] Eyal Shifroni and David Ginat. Simulation game for teaching communications protocols. *SIGCSE Bull.*, 29(1):184–188, March 1997.

[34] Traci Sitzmann. A META-ANALYTIC EXAMINATION OF THE INSTRUCTIONAL EFFECTIVENESS OF COMPUTER-BASED SIMULATION GAMES. *Personnel Psychology*, 64(2):489–528, 2011.

[35] Leen-Kiat Soh. Using game days to teach a multiagent system class. *SIGCSE Bull.*, 36(1):219–223, March 2004.

[36] Joel Wein, Kirill Kourtchikov, Yan Cheng, Ron Gutierez, Roman Khmelichek, Matthew Topol, and Chris Sherman. Virtualized games for teaching about distributed systems. *SIGCSE Bull.*, 41(1):246–250, March 2009.

[37] Barbara Y White. Designing Computer Games to Help Physics Students Understand Newton's Laws of Motion. *Cognition and Instruction*, 1(1):pp. 69–108, 1984.