Adaptive Serious Games for Computer Science Education

by

Michael A. Miljanovic

A thesis submitted to the School of
Graduate and Postdoctoral Studies in
partial fulfillment of the requirements for
the degree of

Doctor of Philosophy in Computer Science

Faculty of Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

October 2020

0.1 Thesis Examination Information

Submitted by: **Michael Miljanovic**

Doctor of Philosophy in **Computer Science**

Thesis Title: Adaptive Serious Games for Computer Science Education

An oral defense of this thesis took place on September 29, 2020 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Faisal Qureshi
Research Supervisor	Dr. Jeremy Bradbury
Examining Committee Member	Dr. Christopher Collins
Examining Committee Member	Dr. Roland van Oostveen
Examining Committee Member	Dr. Pejman Mirza Babaei
University Examiner	Dr. Bill Kapralos
External Examiner	Dr. Stefan Göebel, Technical University of Darmstadt

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

0.2 Abstract

Serious games have the potential to effectively engage students to learn, however, these games tend to struggle accommodating learners with diverse abilities and needs. Furthermore, customizing a serious game to the individual learner has historically required a great deal of effort on the part of subject matter experts, and is not always feasible for increasingly complex games. This thesis proposes the use of automatic methods to adapt serious programming games to learners' abilities. To understand the context of the problem, a survey was conducted of the serious programming game literature, which found that while many games exist, there has been very little consideration for the use of adaptation. Given the breadth of the existing serious programming game literature, a methodology was developed to support adaptation of existing games. To demonstrate the efficacy of this adaptive methodology in serious programming games, two case studies were conducted: 1) a study comparing adaptive and non-adaptive gameplay in the Gidget game, and 2) a study assessing non-adaptive gameplay, adaptive gameplay, and adaptive hints in the RoboBug game. The results from both case studies provide evidence to the need for adaptation in serious programming games, and illustrate how the adaptive methodology can be utilized to positively affect the engagement of learners and their ability to achieve learning outcomes.

Keywords: Adaptation; Programming; Education; Serious Games

0.3 Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

The research work in this thesis that was performed in compliance with the regulations of Research Ethics Board/Animal Care Committee under REB 14096 and REB 15074.

Michael Miljanovic

0.4 Statement of Contributions

Chapters 3 and 4, Section 2.3, and Appendix A.3 were published previously in papers co-authored with my supervisor, Jeremy S. Bradbury. Chapter 3 and Section 2.3 were published in the proceedings of the 2018 Joint International Conference on Serious Games [MB18a, MB18b]. Chapter 4 appeared in the proceedings of the 2nd International Conference on Software Engineering (ICSE 2020, SEET track), in July 2020 [MB20]. Appendix A.3 was published in the proceedings of the 2017 ACM Conference on International Computing Education Research (ICER 2017) [MB17]; although the RoboBUG game was developed as part of my master's thesis, the evaluation was conducted during my PhD research and is an important component of this thesis. For all papers I was the primary author and conducted the research under the supervision and in collaboration with Jeremy S. Bradbury.

0.5 Acknowledgements

I would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Graduate Scholarship (OGS) program for their generous financial support.

I would like to thank my supervisor, Jeremy Bradbury, for being an outstanding mentor and role model to me during my graduate studies. I am extraordinarily fortunate to have had the opportunity to learn from him and to work alongside him.

I would like to thank the examiners and members of my thesis committee: Christopher Collins, Stefan Göbel, Pejman Mirza-Babaei, Roland Van Oostveen, and Bill Kapralos, for their guidance and feedback.

I would like to thank Amy Ko and Michael Lee for their helpful advice and for creating the Gidget game.

I would like to thank my family and friends who have supported me and encouraged me every step of the way.

Finally, I would like to thank Veronica Palacios, for all of her love, patience, and understanding.

Contents

Abbreviations

ACM Association for Computing Machinery. 12, 23, 24, 26, 27, 28, 30

AI Artificial Intelligence. 9, 32, 33, 80, 115

CbKST Competence-based Knowledge Space Theory. 11, 35, 38, 39, 42, 45, 46

CS Computer Science. 1, 2, 4, 5, 11, 12, 15, 35, 41, 42, 43, 48, 90, 102, 103, 105, 113, 115

CT Challenge Tailoring. 34, 35

DDA Dynamic Difficulty Adjustment. 2, 34

DS Dynamic Scripting. 32

EA Evolutionary Algorithm. 32

EDS Evolutionary Dynamic Scripting. 32

GEQ Game Experience Questionnaire. xi, xiii, 18, 19, 53, 61, 68, 71, 89, 93, 104, 106, 107, 109, 112, 144

GSR Galvanic Skin Response. 53

IEEE Institute of Electrical and Electronics Engineers. 12, 23, 24

IT Information Technology. 5

ITS Intelligent Tutoring System. 9, 10, 80

KST Knowledge Space Theory. 10, 11

MKO More Knowledgeable Other. 37

ML Machine Learning. 4, 32, 112

OO Object Oriented. 27

PBL Problem-based Learning. 13

SAE Skill Assessment Engine. 39

SDF Software Development Fundamentals. 12, 23, 24, 28, 30

SDT Social Development Theory. 37

SE Software Engineering. 23

SOA Service-Oriented Architecture. 39

SRL Self-Regulated Learning. 11

ZPD Zone of Proximal Development. 37

List of Figures

List of Tables

Chapter 1

Introduction

Educators face many challenges in providing learners with meaningful and intrinsically motivating experiences [SC09]. This has prompted interest in alternative strategies to accommodate struggling learners, such as games, which have been shown to be effective in engaging student learning of a variety of skills [YW15]. Educational games have been applied in academia as well as professional contexts, including aviation and the military. One of the most heavily targeted fields of study for educational games is Computer Science (CS), where the value of understanding the fundamentals of programming has never been higher [CBD16]. This dissertation seeks to improve the development of educational games, specifically for computer programming.

A **serious game** is defined as "a game designed specifically for a purpose other than entertainment", such as education or training [DDKN11]. A major challenge for serious games is the need to accommodate learners with different abilities and skills. Some solutions to this problem include centralized approaches involving large databases of student data, or the use of customization by human experts, however, these are not always practical options for increasingly complex serious games or increasingly diverse groups of learners [WDD11]. In this thesis, I consider an alternative

automatic solution, namely adaptive serious games, which can customize specific game elements to directly influence learner performance - an approach that, to the best of our knowledge, has not been previously applied in CS [RIPB15]. This dissertation considers the following research question:

- **How can we apply adaptive approaches to existing serious games in the context of Computer Science education?**

As part of my review into the literature of serious games in CS education, it was clear that the vast majority of existing games focus on programming rather than more complex or abstract CS concepts. Thus, in order to answer this question, we consider the following open sub-problems:

1. What are effective adaptive approaches to use with serious programming games?
2. Do adaptive programming games provide a significant benefit for learners over non-adaptive games?
3. Can adaptive programming games find a balance between engaging game play and ability to achieve the learning outcomes of the computer science curriculum?

These sub-problems are not specific to CS, but are generic questions that have been considered by researchers who have previously applied adaptivity in non-CS contexts.

Games that incorporate **adaptivity** capture human-computer interactions, analyze them over time, and make automatic adjustments based on the data [IHK+12]. In non-educational contexts, this can take the form of Dynamic Difficulty Adjustment (DDA), where game play balance is adapted in order to ensure the player isn't bored from easy tasks, or too frustrated from difficult challenges. Adaptivity in serious

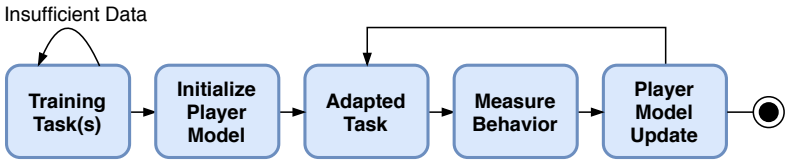


Figure 1.1: Adaptive Gameplay Sequence.

games can adjust both difficulty and learning content to facilitate the player's educational experience. Figure 1.1 illustrates how non-adaptive games can differ from adaptive games; rather than using a sequence of tasks or levels that are identical for all players, adaptive games can use a feedback loop to inform game content based on the behavior of players.

The process of learning relies on a learner's active **engagement**, which is based on their concentration and interest in an activity [RTC+07]. Research has shown that student engagement is positively correlated with **learning outcomes**, particularly critical thinking and problem solving skills [CKK06, BLL+07]. The demanding requirements of learning how to program for the first time suggest that engaging and motivating learners is essential to their development. In the context of computer-based learning environments for introductory programming, student attitude and expectations are highly correlated with efficacy, and a successful intervention can facilitate the encouragement and motivation of students to learn effectively [LLY10].

1.1 Thesis Statement and Scope of Research

Thesis Statement: *The use of an adaptive approach in serious games for computer programming can positively affect the achievement of learning outcomes and player engagement.*

In the previous section, I defined **serious games** as games with a primary purpose

other than entertainment. The serious games involved in my work are games that exclusively focus on programming activities at an introductory level. The goal of these games is to aid learners in developing skills for reading, writing, and understanding computer programs. Currently in the literature, serious programming games tend to be **non-adaptive**; that is, the game play content is identical for all players, regardless of skill level, and is not customized to the individual learner.

I introduce an **adaptive approach** to these games, that allows automated changes to be made to game play and game elements. I define an adaptive serious game as one that receives real-time in-game assessment feedback from players and uses it to update the game during, between, or after game play [Ray07]. My approach includes the development of a methodology for CS programming games that uses Machine Learning (ML) to assess players and modify aspects of the game, including game data as well as difficulty level. The goal of this work is to present an approach to helping players learn fundamental programming skills while maintaining learners' engagement, using in-game assessments as predictors of the learner's current status. In addition to using the data from a given learner, this approach also makes use of historical data to help categorize learners and predict their competences. Ultimately, this work seeks to show that the adaptive approach can improve a learner's game play experience in comparison with the original, non-adaptive version of a serious programming game.

1.2 Motivation

The major motivation behind the use of serious CS games is the need for more computer scientists working in the technology industry. Statistics from the U.S. Bureau of Labor predict that over the next 10 years, there will be 1 million more jobs for

computer scientists than students to take them [BoLS]. The need for technology professionals is simply growing faster than the number of people available. This is true in Canada as well; as of 2019 182,000 skilled workers are in demand, and our domestic supply will not be sufficient to fulfill this quota [II].

One reason for the lack of qualified Information Technology (IT) professionals is that historically, attrition rates for computer science courses in universities are high [BBY08]. In particular, introductory programming courses have a high rate of failure, as students struggle to understand syntax and semantics [RRR03], solve problems [MAD+01], and read code [Man06]. Instructors may find their courses have a bi-modal distribution, where students are either very successful because of past familiarity with programming, or struggle from lack of previous experience [CTT10]. Part of the problem is that most high school students are never introduced to computer science [Car06].

Modern video games show potential not just for engaging and entertaining users, but also for promoting learning [JVM05]. Many serious games already exist that aim to help students learn computer science concepts, such as Serious Cube [MSO12] and The IA Game [AVCW12]. Unfortunately, the efficacy of these games with respect to learning is unknown, as few studies in the literature "documented the empirical data on the effectiveness of instructional games" [Ke09]. More importantly, serious CS games that can be customized based on the abilities of the player are extremely uncommon; most games either present the same content to all players, or require players to make content choices prior to game play (e.g. difficulty settings). This means that current CS serious games for learning programming are unable to accommodate for players who demonstrate difficulty with achieving learning outcomes during gameplay. The goal of my research is to determine if a CS game that can adapt itself to the learner can improve the learning experience by helping to emphasize key ideas,

providing additional content for players who need assistance, and challenging those who are more competent with their abilities. The need for scientific and engineering methods to build games that are not only more realistic simulations of the physical world but also a means to provide effective learning experiences has been identified in the literature [GKH07].

1.3 Contributions

The specific contributions of this thesis include:

- The adaptive methodology that can be applied to existing serious programming games. This is the primary contribution of this work, which is intended to be widely applicable to a variety of programming games that use a task-based approach to game play.

- A survey of existing serious programming games in the literature and developed by industry. This list is presented as a reference for learners or instructors who are looking for examples of serious games, or seeking opportunities to take advantage of game-based learning.

- GidgetML, an adaptive version of the serious programming game Gidget, which introduces basic debugging concepts to learners without the need for prior programming experience. GidgetML was designed to assess learners by comparing their game performance with data gathered from previous players, and that assessment is used to adapt game content. An adaptive version of the serious programming game RoboBUG. Two variants of RoboBUG were created that adapt game content or provide adaptive hinting to aid struggling learners.

1.4 Thesis Organization

This chapter has presented the research questions and associated problems, the motivation behind the work, the thesis statement, and the contributions. The remaining chapters include the following:

1. Chapter 2: an overview of the background related to this work. This chapter includes discussion of literature from the domains of education, game-based learning, serious programming games, and adaptivity.

2. Chapter 3: the adaptive methodology and how it is applied to serious programming games. This methodology was used to develop the adaptive versions of games presented in Chapters 4 and 5.

3. Chapter 4: a case study on GidgetML, an adaptive version of the serious programming game Gidget. An experiment was conducted in order to compare GidgetML to the original non-adaptive game, and results from this study are presented.

4. Chapter 5: a case study on implementing adaptivity into RoboBUG, a serious programming game. Experiments were conducted to collect data for the adaptive version of RoboBUG, and comparisons were drawn between the non-adaptive versions as well as adaptive game play and adaptive hint variants of the RoboBUG game.

5. Chapter 6: a summary of the thesis, its contributions, limitations, future work, and conclusions.

Chapter 2

Background

This chapter provides an overview of the literature that is critical for the development of a methodology to adapt serious programming games. In the following sections, we will discuss the topics of education, game-based learning, serious programming games, and adaptivity.

2.1 Education

2.1.1 Self-Efficacy

The core principle behind this work lies in the importance of learner self-efficacy. According to Bandura, "perceived self efficacy refers to beliefs in one's capabilities to organize and execute the courses of action required to produce given attainments" [BFL99]. A learner's belief on their efficacy has a direct influence both on what they believe they can do, as well as what they actually try to do. Low self-efficacy can be a self-fulfilling prophecy - if someone believes they have no power to produce results, they will not bother to try. When success is not easily attained, those with high self-efficacy will persist, while those with low self-efficacy rapidly

quit [BS81].

Not only does self-efficacy reflect cognitive skills, but it also can contribute to one's intellectual performance. Collins conducted a study with children who identified themselves as either high or low efficacy at three topics of mathematics [Col82]. Children with a strong belief in their efficacy were quicker to discard faulty strategy, were able to solve more problems, chose to rework more problems that they had failed, and did so more accurately than students of equal ability who had a lower rating of self-efficacy.

Csikszentmihalyi claims that, with an appropriate learning experience, almost any sort of activity, no matter how frustrating or trifling it may be, can be imbued with meaningful personal significance [Csi75]. Making an activity interesting can be done by selecting challenges that match one's perceived capabilities, and by providing suitable feedback of one's progress.

However, educational technology can only do so much. As effective as a computer can be, learners, especially children, need human teachers to help build their sense of self-efficacy and to find meaning in their educational pursuits. The development of learner competencies requires them to find meaning and motivation to sustain their involvement in activities. This type of self-motivation requires the completion of personalized challenges that can create a sense of efficacy and self-satisfaction in one's performance [Ban91]. As Bandura notes,

> "To mountaineers, the toilsome activity of crawling over slippery rocks in foul weather is not inherently joyful. It is the self-satisfaction derived from personal triumphs over lofty peaks that provides the exhilaration. Remove the personal challenges, and crawling over rocks becomes quite boring."

2.1.2 Intelligent Tutoring Systems

An Intelligent Tutoring System (ITS) is a computer program that represents knowledge in the form of concepts, rules, and problem-solving strategies, and that can carry on an interaction with a student using Artificial Intelligence (AI) techniques [SB82]. These systems are a form of **'expert system'**, which is any kind of AI designed to provide advice about real-world problems, typically requiring specialized training to master [Cla84]. As the name implies, expert systems are typically built by interviewing experts and representing their knowledge using heuristics. A well designed ITS can deal with the challenge of managing arbitrary student behavior: no matter the type of activity, whether it is troubleshooting, or making moves in a game, the ITS can evaluate a partial solution and respond using its knowledge of teaching.

In order to operate, an ITS must contain a student model, which is how it can support adaptively assessing student mastery of material, as well as facilitating student feedback. This dynamic adaptation is intended to work based on a student's responses in interactions with the ITS, in combination with the past history of assessment data gathered from a prior population of students.

The design of expert systems is extremely time consuming, and these systems' estimates about a learner's knowledge at a given time are uncertain at best, due to factors such as careless errors by knowledgeable students, lucky guesses, changes in knowledge based on learning or forgetting, or patterns of responses that were not predicted by the expert designer [Vil92]. To address this challenge, Villano proposed the use of Bayesian Belief Networks. Villano combined this probabilistic model with a comprehensive theory of knowledge representation and assessment known as Knowledge Space Theory (KST), developed by Doignon, Falmagne, and associates [DF85]. In KST, each knowledge unit or item represents a single question, class of questions, or task that can be performed by a student. The knowledge state of a given student

is defined as the collection of items they are capable of answering. Villano's dynamic model updates with each new response from the student, in order to perform adaptive assessment and maintain an accurate estimate of the student's knowledge.

KST focuses specifically on observable solution behavior, but does not delve into the skills or competencies that are needed for these tasks. This inspired Heller to propose an extension to KST called Competence-based Knowledge Space Theory (CbKST) [HSHA06]. In this theory, each problem is assigned skills that characterize potential answer patterns. The goal of CbKST is to decide on which items to present to students based on their competence state, which is a probabilistic estimation of their set of skill competencies. CbKST has been used as the foundation of research by Albert and Kickmeier-Rust into applying micro-adaptivity in serious games outside of the CS context [KRA10].

CbKST has the benefit of relying on data that does not require self-assessment, as learners may tend to under or overestimate themselves on their own knowledge [Kay01]. This is more likely when the learner has poorly developed self-regulatory competence, in which case they may prefer to be guided through the learning process. However, CbKST also has the risk of overriding the will of the learner and making them feel less involved by removing their choices.

An alternative approach to CbKST is the use of Self-Regulated Learning (SRL), which describes how an individual regulates their cognitive processes in an educational setting and directs their own learning experiences [PP01]. Zimmerman argues that SRL fosters the self-satisfaction and motivation of learners to continue improving their learning methods [Zim02]. This boost in motivation is intended to help learners be more likely to succeed academically and to improve their viewpoints regarding their own future goals. SRL has already been taken up in the context of e-learning, where it has been argued that the best strategy for a learner is for them to take

responsibility of their own learning [SNA09]. SRL and CbKST are two approaches to learning that seem to compete with each other, and each has different benefits and drawbacks. While SRL may overload the learner with too many choices, CbKST risks making the learner feel uninvolved in their own experience.

2.1.3 Computing Curricula

The most recent version of the Association for Computing Machinery (ACM)/Institute of Electrical and Electronics Engineers (IEEE) Computing Curricula Guidelines (CS2013) was published at the end of 2013 [For13]. A list of 18 knowledge areas derived from previous curricula are divided into two subsets, Tier-1 and Tier-2, which respectively indicate whether all CS students must cover a topic, or if all students encounter the vast majority of the material. The knowledge area that has the most hours is Software Development Fundamentals (SDF) - Software Development Fundamentals, which is the primary topic for most introductory programming courses. Although other topics may also be covered in a first year course, most are not covered until later years of study.

According to Dziallas and Fincher, the ACM Curricula reports have become an institution in the CS field. With each new iteration of the curriculum, a process is undergone which requires new chairs, task forces, disciplinary groups, drafts, and the solicitation of community feedback [DF15]. Over time, these views on what is and what is not computer science have inherently shaped the academic discipline of CS.

The CS2013 report has been read by many CS educators, some of whom were responsible for contributing their knowledge to the document. Sekiya et al. conducted a study of the top 10 CS departments of universities in the United States, and found that CS2013 uniformly covered a sufficiently wide area of CS content [SMY15]. In terms of content, some of these universities emphasized human factors in their CS

programs, while others attached a greater importance to theoretical content. Based on their findings, Sekiya et al. suggest that CS2013 is a suitable set of guidelines for the foundation of universities to analyze their own curricula and design new ones.

However, the presence of topics in CS2013 does not necessarily imply that they will always be included into a university program to a sufficient extent. Debugging is the process of removing source code defects, and knowing how to effectively is a critical skill for novices and experts alike. Despite its integral role in development, CS2013 makes little reference to its importance and only mentions that courses should discuss debugging strategies, without providing guidelines or methods for instruction [CL04]. The absence of debugging instruction in many university courses may be why some instructors have turned to game-based learning as a way to introduce debugging and similar topics to their students.

2.2 Game-based Learning

2.2.1 Problem-based Learning

The traditional form of lecturing is not well suited for online delivery, where interaction between teacher and learner is at a minimum. This challenge can be addressed through the use of Problem-based Learning (PBL), whereby the learning process is taken from the teacher's control and placed on the shoulders of the learner [VOCFC14]. The effectiveness of PBL is due to the contexts from which the problems are drawn. In particular, contexts that are relevant to learners and seen as authentic help learners to connect to their own experiences and interests [HSDC07].

Watts divides problems into two different types: 'Given' and 'Own' [Wat91]. 'Given' problems are those where the contexts are constrained, and the learner is provided with both the strategy that should be used as well as the desired situation

after a solution is produced. The goal of these problems is to help learners understand 'how' to do something rather than constructing new understandings. 'Own' problems, also called 'blue-sky' problems, are more open ended. Learners are not provided with processes or desired goals, and instead given the choice of determining these aspects of their own volition.

An application of Problem-based Learning can be found in Game-based Learning. In the field of education, video games have been considered for usage within the existing education system, and research has focused on the inherent potential of games for producing learning [Gee03]. However, the goals of games and the goals of school-based learning do not match - schools are designed to efficiently produce learners according to a defined standard, while games are designed to allow players to think creatively with digital tools [SUFR06, Squ05]. This has led to attempts to integrate games into the curriculum which fall flat, despite the best interests of teachers, learners, and the gaming industry. These failures are attributed to two possibilities:

1. The games designed to educate do not engage their audience, or

2. The games that are engaging do not provide educational value.

Several authors have suggested that a successful educational game must be a game first and an educational tool second, otherwise the potential benefits of gaming are mitigated [VE07,Pre03]. However, this means there is some risk that a game designed this way may be entertaining, but lacking in learning opportunities.

Despite this, learning does occur during gameplay, and this form of learning shares many attributes with the pedagogy of problem-based learning [Roy08]. To progress in a game, a player must solve problems, and typically does so using tools and experience gained from previous levels in the game. To make a successful game for learning, a

pedagogy akin to problem-based learning must be applied. The use of structure and narrative in a game provides the context needed for a meaningful learning experience, and motivation for pursuing knowledge follows after. Although problems may be embedded in a game, the education that is experienced is no less real than a traditional lesson.

When it comes to learning programming, research has shown that students who learn within a context they are familiar with, such as media-computation or robotics, they are more motivated to learn and spend more time on task than required [FG04, FG05, Veg08]. Although it remains to be seen how beneficial the inclusion of gaming is in computing courses, the approach is being experimented with nonetheless [HLL+08, Wal03]. Wolz et al. categorize the uses of educational gaming to be the following [WBPW06]:

- Supporting foundational courses such as CS1[1]
- Providing specialized content in upper level courses
- Providing curriculum that encompasses a thematic approach
- Providing trans-disciplinary experiences

However, video gaming is a space where diversity and inclusivity have not always been fully supported. The Pew Internet and American Life Project surveyed 1,102 youths between the ages of 12 and 17, and found that 97% of them, including 99% of boys and 94% of girls, play some type of digital game [LKM+08]. Game designers must consider the importance of game play that appeals to more than one specific audience in order to support equity, diversity, and inclusion in their products. By

[1]CS1 denotes the name for the first course in introductory programming for a CS major at the university level.

including certain types of games with a broad appeal, such as puzzle and exploration based games, more underrepresented populations might be drawn into the CS discipline [MP09].

In a research report for the Bill and Melinda Gates foundation, high dropout rates in American high schools, especially among students from visible minorities, were described as "the silent epidemic" [BDJM06]. The report stated that nearly a third of public high school students drop out, and this rate is even higher for students from visible minorities. When 467 high school students were questioned as to why they dropped out, 47% responded "the classes were not interesting." This speaks to the need for ways to keep learners engaged, whether by well-designed educational games or other immersive environments, in order to support their learning and help them contribute to society.

2.2.2 Stealth Assessment

Interrupting a game play session to present an assessment to a learner is likely to break their immersion and increase frustration. This is why Shute proposes the use of 'stealth assessment', where data can be collected from students and their competences can be assessed in an invisible process [Shu11]. Stealth assessment represents a process whereby the performance data from a student is gathered continuously through a game play session, and a model is used to predict their various levels of competencies [SVBZR09]. These predictions are stored in a dynamic model, which is changed as additional performance data is collected.

The development of an appropriate dynamic competency model is a difficult task. If the granularity of the model is too large, then there is less specific evidence that can determine the competency of the learner. On the other hand, too fine a grain size means that the model itself is overly complex and requires additional resources to

spend on assessment. This is particularly problematic if assessments must be made in real-time, such as to provide the learner with a subsequent activity based on their performance.

In the face of complex problems, learners must be able to think critically, creatively, collaboratively, and systematically, and must be able to communicate their ideas effectively. These competencies, which have value in the real world and are desired by companies, also are the same ones that are needed to succeed in many games [GHL96]. However, these sorts of skills are not easily measured using a multiple-choice test. Several researchers have found that fixed response formats narrowed school curricula by emphasizing basic content knowledge, without assessing higher order thinking skills [KM91, She91].

To help learners succeed in a dynamic world, educators need to rethink their strategies of assessment, identify the skills that are essential for the current environment, and figure out best practices for assessment of student competencies. Game play has the benefit of producing rich sequences of actions while performing complex tasks, which can be used as a sample of a learner's set of skills and competencies that can be assessed [SV13]. Interactions within a game can be contrasted with the products of an activity, which is highly similar to the typical assessments used in normal education and training environments.

Stealth assessment is designed both to support learning and to maintain flow. Csikszentmihalyi describes flow as "a state of optimal experience, where a person is so engaged in the activity at hand that self-consciousness disappears, sense of time is lost, and the person engages in complex, goal-directed activity not for external rewards, but simply for the exhilaration of doing" [Csi96]. If the goal of game-based learning is to increase engagement with educational content, then it is critical that learners can maintain a state of flow during a game play session. The reason games can be so

engaging is accredited to their ability to foster feelings of control and mastery, as well as their ability to motivate players with social interactions, competition, knowledge, and escapism [HARVE98].

If educational games reach a point where they can be easily deployed and used for automated assessment of learners, teachers may be more inclined to incorporate them into their classes. Students would not only be able to learn in a fun and engaging manner, but they might be able to acquire educationally valuable skills that are not typically supported in school. Except in rare cases, the current education system does not teach or assess competencies such as persistence, creativity, self-efficacy, openness, and teamwork, all of which can substantially impact student academic achievement [NR07, OP07, Por09, Ste06, THHS07].

However, the challenge with using a performance-based measure to evaluate these competencies is the ability to create context-appropriate situations that elicit the desired competencies. Madaus and O'Dwyer argue that incorporation of performance assessments into testing programs is problematic due to their reduced efficiency, increased disruption, and more time consuming nature compared to multiple choice tests [MO99].

One approach that has been used for creating context-appropriate situations is to use digital learning environments that simulate problems [Ded05, DB12, QTB+12]. These environments can provide meaningful assessment opportunities by supplying students with scenarios that are designed with the particular competencies in mind.

2.2.3 Game Experience Questionnaire

Researchers have struggled with the creation of a reliable and valid indicator of participant experiences with gameplay, which led to the creation of the GEQ [PDKI07, IdKP13]. Not to be confused with the Game Engagement Questionnaire [BFC+09],

the Game Experience Questionnaire was created in line with theoretical constructs identified in the FUGA (The Fun of Gaming) Working Model of Digital Game Experience [PDKI07]. The questionnaire and its variants, which include versions that are to be administered before, after, or during game play, have seen use in a variety of studies of game genres, game environments, and other purposes [MBTO14, Nor13]. Its wide usage was the primary factor in its selection for use in this work, however there are some concerns with its validity and reliability.

The GEQ appears to be a useful tool, but recent research has discussed how its psychometric properties are yet to be established [BS15, JGP18, JNW15, Nor13]. The creators of the questionnaire claimed to have verified the various factors, but their work has yet to be reproduced. In particular, the challenge and negative affect components of the questionnaire have been pointed to by others as being problematic [LBM18]. Challenge is a particularly difficulty element of game experience to measure, as a user who is persistent and good at problem solving may have a better game experience with a challenging activity than someone with low tolerance and low self-efficacy [Nor13].

2.3 Serious Games

2.3.1 Introduction

The term **'serious game'** was first presented by Clark C. Abt in 1970 as a way to combine action and thought into an activity that simulates real life [Abt70]. Abt defines a game as "an activity among two or more independent decision-makers seeking to achieve their objectives in some limiting context." [Abt87]. Building on this definition, Djaouti et al. define serious games as ""games that do not have entertainment, enjoyment or fun as their primary purpose" [DAJ11]. Games are suitable as train-

ing devices because of their ability to engage learners of all ages, and because they can communicate and recreate relevant concepts and facts. Players have the ability to assume roles, solve problems, and be evaluated in the game, without risking the consequences that would be faced in a real-world situation. This ability of games to combine a playful experience with real challenges helps to prepare players for life while keeping them engaged and entertained.

Evaluation of games is a difficult task, and research evidence for the benefits of games is essential to persuading schools to adopt games for education [SJB07]. Unfortunately, empirical evidence of the impact of serious games and systematic knowledge of their effectiveness are lacking [MA12]. Part of the reason for this is that the standard metrics of software usability testing are not directly applicable [PKW+02]; instead, the game experience is based on three methodological categories: game quality, human-computer interaction, and player context experience [NDG10]. Game quality is the only category where traditional software testing, such as unit tests and bug tracking, is applied. Human-computer interaction between the player and the game can be modeled based on game play performance, as previously discussed, or measured using physiological tests such as eye tracking. Finally, the context and social impact of player experience can be judged with playability heuristics or qualitative interviews and questionnaires.

Within the field of game-based Computer Science learning, a large number of games have been developed that focus on computer programming [VMM14]. Unfortunately, serious programming games are often developed independently; existing work does not focus on methods that improve gameplay, and there is a need to analyze the use of games to support introductory programming [KKBM12].This means games may be created without learning from existing games, especially if games are not available via open-source licensing or other methods. Additionally, many serious games

that claim to have positive outcomes for players lack any scientific validation [PB10]. Without any supporting evidence, it is difficult to compare serious games and judge which are the most effective learning tools.

We have surveyed 49 serious programming games with respect to both game content and evaluation [MB17]. Specifically, we surveyed these games to answer the following research questions:

- *What programming knowledge is covered by existing serious games?*
- *How are serious programming games evaluated?*

The games included in our survey are exclusively games that involve reading and/or writing programs in order to help players develop computer programming skills. The goal of our review is to provide a comprehensive overview of the state-of-the-art research in these serious programming games while also identifying open research problems. The open problems we identified fall into two categories: new opportunities for serious games development and new opportunities for enhancing evaluation best practices. Our review methodology is described in Section 2.3.2 followed by our review results in Section 2.3.3. Related work is discussed in Section 2.3.4. Finally, we discuss our results and present open problems in Section 2.3.5.

2.3.2 Methodology

Identification and Selection Criteria

We used different selection criteria for our categories of research and commercial games. The search term we used for both was "introductory programming games". Research games were first gathered using the top 200 results from a Google Scholar web search in English, then pruned based on our exclusion criteria. We then selected

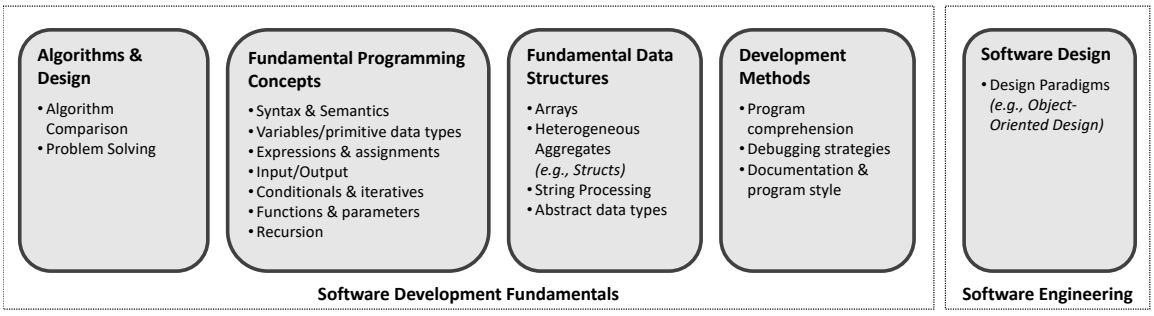


Figure 2.1: ACM Computer Science Curricula 2013 knowledge areas [For13]

games that were discussed in the related works sections of those papers that fit our criteria. For inclusion in the study, research games must have an associated peer-reviewed paper or be the subject of a thesis project. Games developed by researchers with no oversight were excluded. Unfortunately, not all research games were available to be played and we therefore had to evaluate some based only on their descriptions and not a first-hand evaluation. Commercial games were collected using a Google web search and through online game stores[2]. This includes games like Code Hunt [TB14], which is a game developed by Microsoft Research, but has been discussed in several peer-reviewed papers.

Classification Criteria

Audience. The targeted audiences for serious programming games is broken down into four separate categories:

- Children (age 5-13)
- High school students (age 14-17)

[2]Commercial sources such as the iOS game store, Google Play Store, `HourOfCode.com`, and `Tynker.com` offer over 100 different programming games, typically aimed at children. However, many of these games follow a similar approach or share visual programming environments. We chose to only include a small sample from these websites due to the high degree of game overlap and similarity.

- Undergraduate novices (age 18+, no programming experience)
- Undergraduate adepts (age 18+, programming experience)

Educational Content. The 2013 ACM/IEEE Curriculum Guidelines for Undergraduate Degree Programs in Computer Science [For13] include areas of knowledge for students learning programming (see Figure 2.1). The concepts we selected were from the Tier 1 list of knowledge areas for computer science, meaning that the topics are intended to be introduced to students in their first or second years of study at the undergraduate level.

We focused on the SDFs and Software Engineering (SE) knowledge areas. SDFs are critical for students to become both competent at programming and knowledgeable about designing and analyzing algorithms. The other software-oriented knowledge areas discussed in the ACM Curriculum require students to have strong foundations in SDFs. In addition, a number of games have been developed to help students understand the object-oriented paradigm; we chose to include this specific section of SE to acknowledge the multitude of games that included or focused on object-oriented development. Due to space limitations, we chose to exclude knowledge areas and concepts that were not included in more than two games.

We differentiate between the inclusion of educational content with a primary focus versus a secondary focus. Educational content that is classified as being a primary focus of a given serious game indicates that the game designers emphasize that their game is designed to teach that content. In cases where the primary focus is not explicitly stated, we make a determination based on play testing or on a description of the game play. Alternatively, if educational content is present in a serious game but not emphasized, we classify this as having a secondary focus. Content with a secondary focus may be introduced in the game or may need to be learned prior to

playing.

Evaluation. When available, we also review and classify the evaluation of a serious game[3]. First, we identify which games evaluate learning outcomes, player engagement, positive feedback, and accessibility. Second, we classify the evaluation methods used, ranging from informal player feedback and game results to full empirical studies about learning outcomes.

2.3.3 Results

We identified 49 serious programming games that met our selection criteria – 36 research games and 13 commercial games. Approximately half (23) of these games can be downloaded or played online.

Audience

The largest audience of the surveyed games was undergraduate novices with no programming experience (21 games) followed by undergraduates with some programming experience (17 games), children (seven games) and high school students (four games).

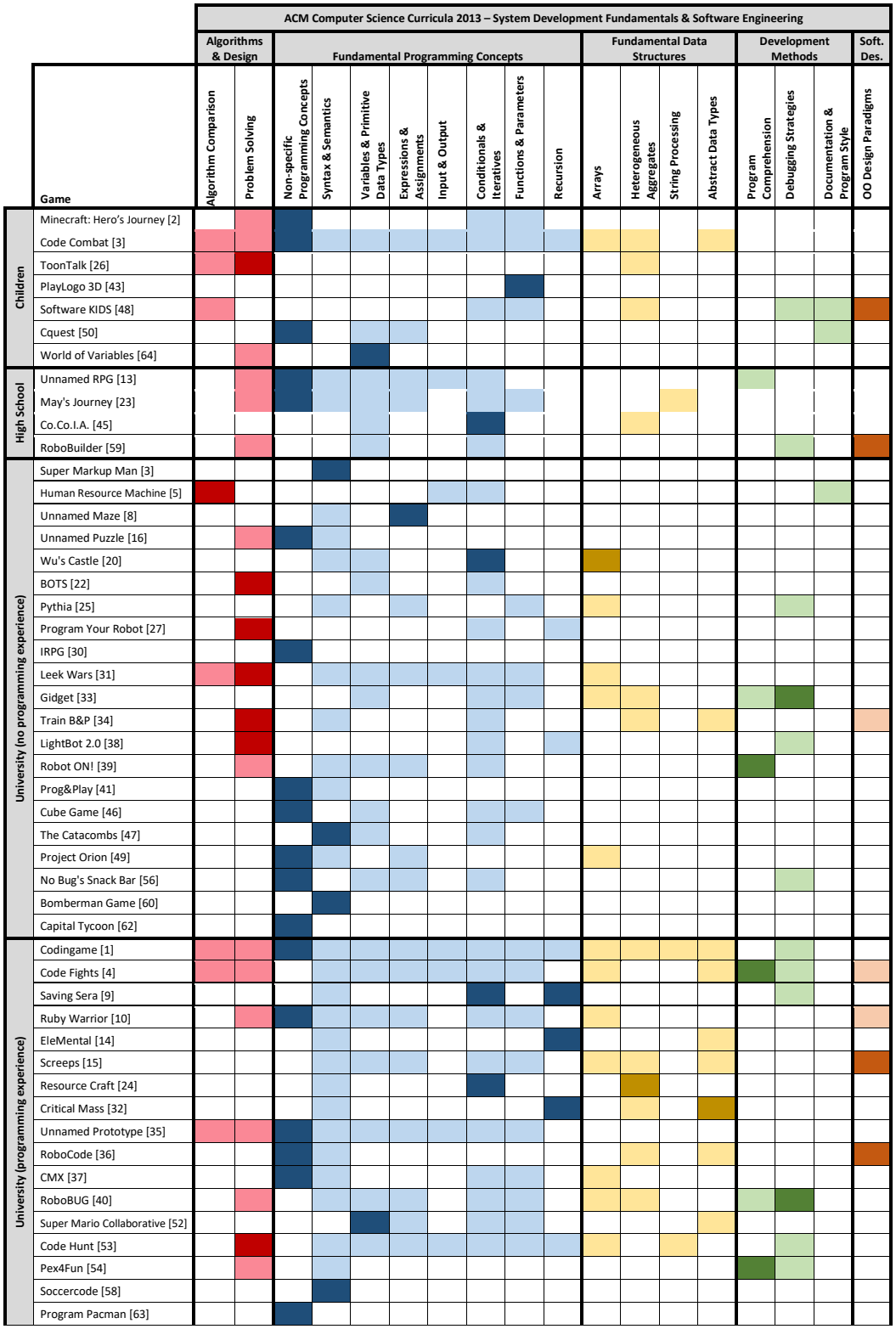
Educational Content

The educational content in each game was assessed based on the knowledge areas identified in the ACM/IEEE Computer Science Curricula 2013 (see Table 2.1):

- **Algorithms and Design:** this SDF unit covers the importance of algorithms in problem-solving, including mathematical functions and divide-and-conquer strategies. The comparison of algorithms in the surveyed serious games was not widely covered. An exception was the Human Resource Machine game in which

[3]Serious games with evaluations are primarily a subset of those that have accompanying research papers, technical reports or theses.

Table 2.1: Classification of serious programming games based on educational content



players are incentivized to minimize the number of instructions and steps taken to complete tasks. When algorithm comparison was included, it was often done informally and without any mention of algorithmic complexity. Interestingly, only 21 of the 49 games had an emphasis on problem solving. For example, Robocode [Lon07] is not problem solving-based but is instead competition-based with players completing programming challenges against opponents.

- **Fundamental Programming Concepts:** these are the most commonly targeted topics for serious programming games, which is consistent with the goal of introducing students to programming and helping them learn how to read and write code. 'Syntax and Semantics' was the most commonly covered concept with 30 games using some sort of written programming language; the remaining games used a drag-and-drop block interface for creating programs, or use a high-level language with little room for error (e.g. Gidget [LK12]). The next most widely covered concept was 'conditionals and iteratives' with 28 games, followed by 'variables and primitive data types' with 23 games. Not all games required fundamental concepts like variables. For example, PlayLogo 3D [PAM13] does not include variables as players need only submit individual commands with functions to play the game. While 'Recursion' was the primary focus of several games [BPCL08,CDHB09,Law04], it was one of the least covered concepts along with 'Input and Output.' This result was surprising given that the target audience for many of the games was university students with some programming experience.

- **Fundamental Data Structures:** the most common data structure concepts were 'arrays and lists' (13 games) and 'heterogeneous structures' (12 games). Few research papers explicitly stated a focus on data structures, and our iden-

tification was primarily the result of game testing and reading game play descriptions. 'Abstract data types' were not included in most games. Exceptions include Critical Mass [Law04], which required players to navigate a tree structure. Finally, 'string processing' was only a secondary focus of three games, and other ACM concepts including 'linked lists' and 'referencing' were not targeted by any of the games.

- **Development Methods:** 'debugging' was the most commonly targeted development concept, with 12 games featuring some focus on debugging code. However, this does not include all of the ACM's program correctness topics (e.g. test-case generation, unit testing). Even the games that choose to focus on debugging [MB17, LK12] are not comprehensive with respect to debugging topics. 'Program comprehension' was the focus of a few games, but the vast majority of games required players to write their own code. CodeFights is an example of a game where players must interpret code written by someone else and develop program comprehension skills through trying to understand foreign code. Although games with real programming languages allow for commenting, very few games focused on documentation and program style, and only did so as a secondary focus. Other development methods, including refactoring and the use of software libraries, were not covered by the games.

- **Software Design:** the majority of software design areas presented by the ACM are intended for learners above the beginner level. However, the curriculum indicates that software design should be covered at an early stage. IBM's Robocode [BW04, Har04, Lon07, OG06] has a strong focus on software design – specifically Object Oriented (OO) design, as players learn about abstraction through the use and modification of the game's robot objects.

Learning Focus

Identifying the primary focus of serious programming games was especially difficult when not explicitly stated by the game designers. When not stated, we based our identification of a primary focus from playing the available games and inferring based on the content of the research papers. In the end we found that 18 games focused primarily on general introductory programming, without a specific topic. Most of these 18 games included other fundamental programming concepts, but there were some research papers that introduced a game for learning introductory programming without detailing specific content. Problem solving was the second most common focus, with 7 different games. One example of this is Lightbot 2.0 [MCPS16], where players do not learn a programming language but do develop an understanding of sequencing and implementation of algorithms. The general trend of programming games that focus on problem solving is that they target simple problems and program-based solutions, with limited or no emphasis on formality.

Evaluation

A variety of evaluation methods were used in the surveyed games (see Table 2.2) – 23 surveys, 11 sets of game play statistics, 10 skill tests, four sets of interviews and one evaluation using expert feedback. 21 games used only one evaluation method while 14 used multiple methods. The most common evaluation subject matter was positive feedback. There were 21 cases of participants reporting that they liked a game, often through a survey. 16 games were evaluated for learning effects on the players, but unfortunately seven of these did not have a statistically significant learning effect. Although many of the papers cited engagement as a motivation for using serious games, only 11 were actually evaluated for player engagement. Finally, only eight of the games were tested for accessibility.

Table 2.2: Classification of serious programming games based on evaluations methods

		Research Questions				Method of Evaluation					
	Game	Did the users have positive feelings about the game?	Was the game accessible?	Were users engaged while playing the game?	Was there a learning effect from playing the game?	Informal Feedback	Survey/Questionnaire	Formal Interview	Skill Tests	Game Play Statistics	Expert Feedback
Children	ToonTalk [26]		✓							●	
	PlayLogo 3D [43]		✓								●
	Software KIDS [48]	✓					●				
	Cquest [50]	✓				●					
High School	Unnamed RPG[13]				✓		●		●		
	May's Journey [23]	✓					●			●	
	Co.Co.I.A. [45]			✓		●					
	RoboBuilder [59]			✓		●					
University (no programming experience)	Unnamed Maze [8]			✓			●			●	
	Unnamed Puzzle [16]			✓			●				
	Wu's Castle [20]				✓		●		●		
	BOTS [22]				✓				●		
	Pythia [25]		✓				●				
	Program Your Robot [27]	✓					●				
	IRPG [30]	✓				●					
	Gidget [33]			✓	✓				●	●	
	Train B&P [34]			✓	◆		●			●	
	LightBot 2.0 [38]	✓			✓		●				
	Robot ON! [39]	✓			✓		●	●	●		
	Prog&Play [41]	✓	✓		◆		●		●	●	
	The Catacombs [47]	✓	✓	✓	◆		●	●	●	●	
	Project Orion [49]	✓	✓		◆		●				
	No Bug's Snack Bar [56]	✓				●					
	Capital Tycoon [62]			✓	◆		●				
University (programming experience)	Saving Sera [9]	✓	✓	✓	◆		●	●	●	●	
	EleMental [14]	✓			✓		●		●	●	
	Resource Craft [24]				✓		●			●	
	Critical Mass [32]	✓					●			●	
	Unnamed Prototype [35]			✓				●			
	RoboCode [36]	✓		✓	✓		●				
	CMX [37]	✓					●				
	RoboBUG [40]	✓	✓		◆		●		●		
	Code Hunt [53]	✓				●					
	Pex4Fun [54]	✓				●					
	Soccercode [58]	✓				●					
	Program Pacman [63]	✓					●				

EMPIRICAL EVIDENCE FOR A GIVEN RESEARCH QUESTION WAS POSITIVE (✓) or INCONCLUSIVE (◆)

DATA WAS COLLECTED & ANALYZED USING A GIVEN RESEARCH METHOD (●)

2.3.4 Related Work

Although there are reviews that investigate the impact of serious games [CBM+12, Don07], there is very little research focusing on serious programming games. One exception is a review by Vahldick et al. [VMM14], that focuses specifically on games for improving introductory programming skills. The review categorizes 40 games by type (Logo-based, adventure, general), platform (Windows, iOS, Java, Web, Android, Linux), competency (writing, reading, debugging), topic (including some of the ACM 2013 CS curricula topics from SDFs), and language (Textual/visual block graphics, Java/Javascript, C/C++/C#, and others).

Our work has two similarities with the Vahldick et al. review – first, 15 games are included in both studies and second, both studies survey the learning topics or content of the games. With regard to this overlap, we have included 34 games in our study that were not included by Vahldick et al. Furthermore, 25 games included in their study were not included in ours. Reasons for exclusion include: 13 games were outside of our selection criteria (e.g., non-english, not focused on learning programming), four games were extremely similar to other games in the survey, and eight games were no longer available online and did not have published papers. Our initial intention was to include as many of the previously studied games as possible in order to reproduce and validate the learning portion of the Vahldick et al. results. However, this was not possible as many of the overlapping games have been updated in the three years since their study and we no longer have access to the versions of games surveyed. The main difference between our work and the Vahldick et al. review is that we have surveyed a wider selection of games with the intention of assessing the learning aspects of the games (learning content and learning evaluation) as opposed to the game characteristics (e.g., platform, language, genre).

2.3.5 Discussion & Conclusions from Survey

Our results show that the 49 serious programming games surveyed focus primarily on a subset of the ACM computing knowledge areas. Unfortunately, many of the games are not released publicly and we were unable to independently verify the learning content of these games through play testing. The lack of access is problematic for both researchers developing computer science educational games and instructors seeking to find effective learning tools. The surveyed serious games focus largely on the problem solving and fundamental programming concepts knowledge areas. There are a lack of games that focus on data structures, development methods and software design. Furthermore, while the primary learning focus of many games was introductory programming, few of the games appear to cover all of the ACM's SDF. This indicates **a need to determine if new serious programming games can bridge the curricula gaps**. It is possible that some SDFs are not well suited for game-based learning.

With respect to game design, we observed that the majority of the games were not multiplayer. There is **a need for further research on the learning benefits of competitive and collaborative serious games for programming**. We also observed that while a number of games were designed with accessibility and inclusivity in mind (e.g. Saving Sera [BPCL08], May's Journey [JZ16]), many did not include any detail on these important aspects of design. This maybe an indicator that **best practices for accessible and inclusive design of serious programming games need to be adopted.**

With respect to the evaluation of serious program games, we were unable to observe common methodological practices other than a tendency to assess if players liked a game. This indicates a **need for the establishment of best practices in evaluating serious programming games**. We believe that the use of alter-

native evaluation methods in addition to in-class studies would be beneficial. In particular, it may be helpful to consider controlled experiments and expert feedback (e.g. used only in PlayLOGO 3D [PAM13]) in combination with playability heuristics [DW09]. Finally, in addition to establishing best practices for evaluation, there is a **need for third-party evaluations**. Third-party evaluations do not suffer from self-confirmatory bias, provide valuable data that can independently validate a serious game's learning effects, and can lead to wider adoption of serious games in Computer Science education.

2.4 Adaptivity

2.4.1 Adaptive Games

An adaptive game system is a strategy used specifically to modify the game play experience. The level of AI in adaptive game systems can vary from simple rule-based systems, to more advanced ML algorithms. Non-adaptive games may also include AI, typically in the form of AI **agents**. These agents are artificial entities not intended to provide assistance or alter the game based on the player's performance, but rather to serve as an obstacle that reacts to the player's actions. Although these AI agents can provide elements of enjoyment and challenge to a non-adaptive serious game, they will behave similarly for different players regardless of player skill levels. On the other hand, adaptive game AI can rectify the loss of flow by making dynamic adjustments during game-play.

Games outside of the education domain use a variety of different techniques for adapting to users, including evolutionary learning and dynamic scripting [Pon04]. Evolutionary Algorithms (EAs) are metaheuristic search methods that generate solutions to a problem, then apply genetic operations to those solutions in an attempt

to optimize results. Although EAs have been used in the context of simple computer games, they are computationally expensive and are not guaranteed to find a good solution. By contrast, Dynamic Scripting (DS) is an AI learning technique commonly used in commercial serious games that allows difficulty scaling with respect to a human's skill level [SPSKP06]. DS is a probabilistic rules-based approach that assigns likelihoods to different AI agent actions depending on their fitness, which is evaluated after an action is taken. In competitive games, rules that lead to success are more likely to be maintained, as the goal of the AI is to use the least exploitable strategy. One drawback to this technique is the need for domain experts to create a complex model that will only represent a limited range of tasks. A recent strategy for using dynamic scripting involves combining it with the generative characteristics of evolutionary algorithms in an approach called Evolutionary Dynamic Scripting (EDS) [KTHR15]. EDS is able to generalize to new scenarios using a small set of general rules that reduce the workload of the expert while evolving rules using genetic programming.

Although AI has been present in computer games since their inception, adaptive systems that dynamically adjusted difficulty settings were not incorporated until many years later [Pon04]. A 2009 study by Hagelback et al. found that players prefer playing against AI opponents with adapting difficulty than AI opponents with non-adaptive performance [HJ09]. In the context of serious games, Thomas and Young's 'Annie' learning system was successfully able to diagnose students' knowledge acquisition in an exploratory game environment [TY10]. After observing participant game play, their system was able to predict participant test answers and results with higher accuracy than a trained human expert. Bellotti and colleagues' study included an adaptive experience engine that improved player satisfaction with serious games [BBDGP09]. This engine is designed to take information from content experts

and incorporate it into a game through different tasks that are chosen based on the player's performance. Most importantly, their engine is modular and can be added on top of existing serious games that make use of the learning task paradigm.

Non-adaptive games have struggled with their need to appeal to a particular audience, which limits their value to those outside the target demographic. Unlike traditional games, adaptive games can cater the gaming experience based on the individual user, and not just a specific group [GD04]. Adaptive game design requires an entertaining experience to be provided to users of all types, regardless of their motivation or level of skill. Historically, designers have used features such as difficulty settings (e.g. easy, medium, hard), but these settings are static and do not consider that players are dynamic entities who will change as they absorb the game experience.

When a game is too easy, it is boring, and when a game is too hard, it becomes frustrating. Games with static difficulty settings can lead to mismatches between player skill and the challenges presented by the game [Hun05]. This led to the development of dynamic difficulty adjustment (DDA), which offers a modulating system to respond to changes in a player's ability over the course of game play. As contemporary computer games are being played by increasingly diverse audiences, their levels of skill and interest in particular types of games vary significantly. Removing the option to set one's own difficulty setting can be particularly beneficial for novices, who assume a low success rate regardless of their actual performance.

An alternative to DDA is the idea of Challenge Tailoring (CT), which occurs when the difficulty of a skill-based event is altered in response to a player's ability [ZR12]. CT is similar to DDA, which applies online real-time changes to game mechanics. However, CT is a more generalized approach, which includes both online and offline optimization, as well as not being limited to adapting the difficulty of the game; CT adaptations might include changing the levels that are selected, or providing

additional feedback to help players. CT relies on having a dynamic player model as well as an algorithm that can adapt content based on that model.

Effective player modeling for CT relies on a data-driven approach to predict future player behavior. Not only must the model properly assess the player's current state, but it must also account for changes in player behavior due to learning effects. A successful model allows an adaptive game to effectively forecast future behavior, accommodate changes from learning, and better direct players towards content they will find meaningful and engaging.

2.4.2 Adaptive Serious Games

Adaptive training systems are "serious games whose goal it is to engender communication opportunities for players to learn about their strengths and weaknesses, receive real-time in-game performance feedback, and share diverse solutions and strategies during, between, and after game play in order to update, or adapt, player understanding" [Ray07]. These are distinct from what we might call **'non-adaptive'** serious games, which always provide the same game content to players regardless of their in-game performance. Approaches to understanding players in a virtual setting require either obtaining direct feedback from users, or observing user interactions with the system [DBTMGFM10], each of which presents its own challenges. Use of a probabilistic model of students based on player performance can provide insights about learning that occurs during game play [MC05], however, this has not been investigated in the context of computer science. The interactivity of games can help produce detailed information about users that can be used to automate their experiences in accordance with the student model. Such models can be based around **CbKST**, which represents the knowledge of learners in terms of their **competences** [AL99]. Each general competence statement should have specific learning outcomes, which can be

measured and assessed more directly. Outside of CS, CbKST has been shown to improve learning performance from adaptive interventions in serious games [KRMSA11].

Adaptive games can be fully autonomous, or allow for content experts to create and include game content after the game's release. An autonomous serious game can promote instructive game play, manage the level of challenge of the user experience, provide scaffolding selectively where needed, and support learners in their efforts to reflect on their play and improve their skills [JVM05]. Alternatively, content experts can manually modify a serious game with the assistance of authoring tools that allow them to generate modules for a particular game engine. Effective selections of additional content and task delivery strategies can be based on assessing each player's performance in a game. This process has been modeled in the context of 'sandbox' serious games - games that allow players to roam freely without a mandatory route [BBDGP09]. To maintain flow, an effective adaptive system needs to automatically balance the difficulty of game content with the developing skills of the learner. The goal of an adaptive game engine should be to keep players within the 'flow zone' as much as possible.

Research has only recently investigated player emotions as a source for adaptivity in serious games [SBF+17]. Although a variety of instruments can be used to assess a player's emotional state, visual observations of the player are the least intrusive. Whole-body and facial behavior have both been used to indicate engagement and boredom, based on the frequency with which a player shifts posture or makes certain facial expressions [BBCC+06, WRN+15]. In addition, the use of eye movement data has been shown to improve user experience and reduce frustration in the context of non-serious games [WSB14]. Access to a multitude of measurements about player emotions is critical to creating an accurate model of a player's engagement and level of frustration, but there are limits to what is logistically feasible for use in a serious

game. The requirement of specialized equipment in a serious game that measures physiological data is an obstacle to delivering the game to as many learners as possible.

In research with developing serious games, children were found to require additional time to learn how to play compared to adults, while both could reach the same time with help [IPB11]. The main reason that learners are more successful with human tutors than other tools is because of a human's ability to adapt to the learner [VLS+05, NW87, Blo84, ACKP95]. Historically, serious games, do not adapt to the learner, and there is yet to be a standardized approach that successfully adapts game environments based on learning content [IKH+12].

A static software system is limited in its ability to adept to users due to its design. When it comes to adaptive games, there are similar limitations as to what can actually adapt or be adapted [SS16]. Despite progress in Artificial Intelligence from projects such as the Human Brain Project [MML+11] and advances in deep learning with neural networks [KSH12], it still remains to be seen if fully personalized serious games are a real possibility. However, techniques such as machine learning and data mining techniques have begun to see usage in the development of adaptive serious games. Automated adaptivity has the potential to reduce negative gameplay experiences and personalize interactions with individuals of different skills and demographics.

Vygotsky's Social Development Theory (SDT) [Vyg80] focuses on the social context of the learner, and defines the maximum level of challenge for a specific topic and individual as the upper limit of the Zone of Proximal Development (ZPD) [Lan04]. The lower limit of the ZPD is the minimum level that can be learned by someone without any assistance, thus the ZPD is the difference between what can be achieved independently versus what is possible with full guidance.

This leads to a definition of adaptivity in serious games, provided by Ismailović et al. [IHK+12]:

> "ADAPTIVITY in serious games is an approach that enables a serious game to learn from learner's behavior by [1] intelligently monitoring and [2] interpreting learner's actions in the game's world and to intervene in the game by [3] automatically adjusting the learning content and [4] the game elements according to the student's individual ZPD as necessary and using the principles of More Knowledgeable Other (MKO), where ADAPTIVITY is a MKO for the learner according to the SDT."

Based on this definition, adaptivity can be separated into four different stages:

1. Monitoring Players, where observations are used to collect data,
2. Learner Characterization, where the observations are used to rate the learner's individual skills,
3. Assessment Generation, where learning content is adjusted based on 1 and 2, and
4. Adaptive Intervention, where game elements not pertaining to learning content are adjusted.

Some game elements may affect a learner's ability to solve a given task, such as the speed of a moving line of text which provides information relevant to the learning content.

One of the most well known projects for adaptive game-based learning is the European ELEKTRA project[4], which used micro-adaptivity to make non-invasive changes to gameplay without disrupting flow [KRHAA08]. A subsequent project, 80Days, used data from interactions and manipulations of game objects to make similar adaptive changes [SKRM+12]. The assessment behaviors from these games

[4](http://www.elektra-project.org/)

are based in CbKST, where procedures interpret a learner's competence level based on their in-game actions and store them in a dynamic user model. The model is then used to select appropriate stories which are provided to players continuously until the desired competence state is reached.

2.4.3 Adaptation Techniques

A variety of techniques used in artificial intelligence have been incorporated into adaptive gameplay literature. In one study, neural networks were successfully used to predict future marks of students using raw gameplay data [IVCFGD+13]. However, this approach may not be suitable for real-time adaptation due to the time complexity of using a neural network. Another study made use of an X-means clustering algorithm [PM+00], extending the K-means clustering algorithm [M+67], which partitioned observations of player strategies in a serious game after estimating the number of significantly different clusters.

Another technique for use in an adaptive serious game is the Skill Assessment Engine (SAE), which calculates probabilistic values for a learner based on their in-game actions [PCW08]. Similarly to ELEKTRA and 80Days, this engine implements Knowledge Space Theory, and an evaluation by the creators found the following was true for their group of participants with higher adaptivity:

1. Higher amount of invested effort and a higher degree of absorbedness
2. Higher relatedness with Non-Player Characters (NPCs)
3. Higher usefulness of the interactive environment
4. Easier manipulation of the interactive environment
5. Higher confidence in their own learning achievement

The success of the SAE led to the creation of a service-based adaptive game on probability called The Journey [CBB+14]. Using Service-Oriented Architecture (SOA) [BKBH07], the authors developed a method based on CbKST that was able to realize their in-game adaptation of The Journey.

2.5 Summary of Background

In this chapter, we have introduced the related areas of education, game-based learning, serious games, and adaptivity. The information from the education literature discussed in Section 2.1 is critical to understanding the needs of learners and how self-efficacy and engagement relate to the learning process. Game-based learning, discussed in Section 2.2, is a form of problem-based learning, is a way that learners can become engaged with educational content by relating learning outcomes to meaningful tasks. In Section 2.3, we presented a survey of 49 existing serious programming games that provide an illustration of the current state of the literature. Finally, Section 2.4 discusses the concepts of adaptivity and how they can be applied to serious games. These four topics are all relevant for the development of the adaptive methodology that is presented in the following chapter.

Chapter 3

A Methodology for Adapting Serious Programming Games

3.1 Introduction

The use of serious games is one approach that has shown effectiveness in engaging students to learn a variety of skills [YW15]. As discussed in Chapter 1, the potential for serious games to increase motivation and engagement among learners is particularly important for the field of CS, where low engagement levels give cause for concern [BSMK16]. Furthermore, the widespread interest in understanding the fundamentals of programming has led to CS being a heavily targeted field of study for serious games researchers [CBD16].

While serious games have considerable promise, challenges still exist with respect to their design and evaluation. One of the open challenges is customizing serious games to suit learners of different levels of ability and knowledge. Existing solutions to this challenge can require substantial human effort, such as the monitoring and customization of gameplay by human experts and the creation of large, diverse prob-

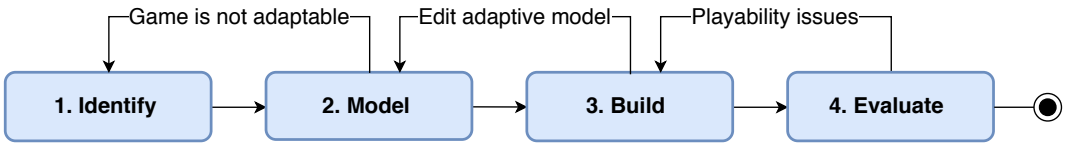


Figure 3.1: An overview of a methodology for making adaptive serious games [MB18a]

"Once a game has been identified for adaptation, a model and plan is developed that connects the game play tasks with a method for assessing learners. After the model has been created, the adaptation functionality is built into the existing code base. Finally, the new adaptive serious game should be evaluated to determine its efficacy (e.g., learning, engagement) and the evaluation results should be compared with the efficacy results of the original non-adaptive serious game." [MB18a]

lem sets. One drawback of these approaches is that they are not always practical when working with increasingly complex serious games [WDD11]. In this work, we consider the use of adaptive serious games to make serious games suitable for learners with different skills. Adaptive serious games do not have the same drawbacks as the above mentioned approaches as they can automatically modify game elements and content to directly impact learner performance [RIPB15].

The main contribution of this thesis is a new methodology for incorporating adaptive gameplay and content into existing non-adaptive serious games. We have chosen to focus our methodology on CS serious games because many of the existing serious games for learning programming have widespread adoption and have empirical research to support their educational value (e.g., Code Hunt [BHX+15]). We believe modifying existing serious games that have already been adopted and evaluated is a more desirable approach than building new games from scratch.

3.2 Methodology

Adaptive games can be fully or semi-autonomous, allowing for game content to be included after the game's release. An autonomous serious game can promote instructive gameplay, manage the challenge of the user experience, provide scaffolding where

needed, and support learners [JVM05]. As was discussed in Chapter 2, a common approach to adaptive serious games is to use CbKST in combination with a probabilistic approach [AL99]. Our methodology leverages CbKST for making adaptive serious programming games from non-adaptive games and includes four key phases: identification, modelling, building and evaluation (see Fig. 3.1). To assist in explaining our methodology, we use the example of creating an adaptive version of Gidget [LK12]. Gidget is a non-adaptive serious game where players complete missions by repairing faulty programs.

3.2.1 Identifying a Potential Adaptive Game

Both technical and learning factors should be considered when deciding if an existing serious game is an appropriate candidate for adaptive methods.

Technical Factors. In order to adapt a game, the source code will need to be publicly available and extendable. Thus, it is necessary to ensure that for third-party games, the software license for the chosen game allows for modification. The quality and robustness of the source code should be examined when identifying a serious game for adaptation as both of these factors can impact the modification of the source code. Also the playability of the game and the existence of playability studies should be considered. Gidget is an ideal choice technically because it is open source, includes documented source code, and has been evaluated indirectly with respect to playability.

Learning Factors. First, adapting the learning content of a game requires a clear understanding of the required knowledge, topics, and learning outcomes that are present in the original game [For13]. Second, making informed decisions about adapting a game requires detailed knowledge about the learners who will play the game. Learners of various demographics including age groups may respond differently

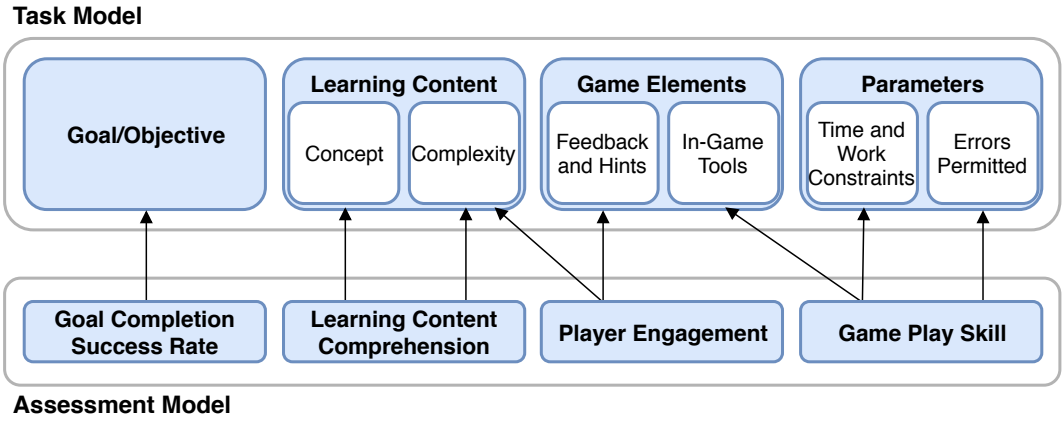


Figure 3.2: Task and Assessment Models for Adaptive Games.

to in-game adaptations. Additionally, knowledge about the level of programming experience of the game's intended audience is needed in order to make good decisions on how to adjust learning content. Special consideration should also be given to adapting for learners of diverse educational backgrounds outside of CS and choosing games that are inclusive. Third, in order to properly evaluate the new adaptive serious game at the end of the process, it is best to choose an existing game that has already been evaluated with respect to learning as the existing evaluation can serve as a baseline in assessing the adaptive version. Our example, Gidget, focuses on learning debugging and has a target audience of general learners with no previous programming experience. Gidget has also been previously evaluated with respect to learning [LK15].

3.2.2 Modelling the Gameplay Tasks and Learning Assessment

Before implementing adaption into a serious game it is necessary to understand and model the gameplay tasks as well as the learning assessment (see Fig. 3.2).

Task Model. A typical serious programming game includes a sequence of in-

creasingly difficult tasks that pertain to learning content. Often, serious games are designed such that a player's success in the game is dependent on the completed and failed task objectives. Although the criteria for determining whether an objective is failed varies from game to game, failure is often accompanied by feedback or hints, as well as a reset of parameters such as time or error limits. The existing tasks in the game can be modelled and used as a template for adaptation. The most important task properties that should be included in this model are objectives, learning content, game elements, and parameters. These properties can be extracted from documentation as well as the structure and content of the game's source code. In Gidget, each level is a task with one or more objectives, each of which is completed when a given physical object on a grid is moved to a specified location. The primary learning concept in Gidget is debugging, and each level presents increasingly complex objectives, with partially incorrect code for completing those objectives. In addition to the debugging levels, newer versions of Gidget include levels that introduce concepts such as conditionals, functions, and arrays. Gidget provides substantial feedback to the player by visualizing every step of the code on the grid, and allows players to choose the number of steps to process at a given time. In order to encourage efficient programs, Gidget has an 'energy' limit that restricts the number of moves that can be taken during a level, but does not limit the gameplay time or number of errors permitted.

Assessment Model. Our model of assessment is based on CbKST and a probabilistic evaluation of the learner's competence in the learning content. The use of CbKST necessitates the inclusion of goal completion success rate and learning content comprehension in our model as predictors of a learner's competence. Since Gidget allows players to repeat a task until it is correctly solved, players must be assessed based on the efficiency of their code solutions. This includes measuring the error

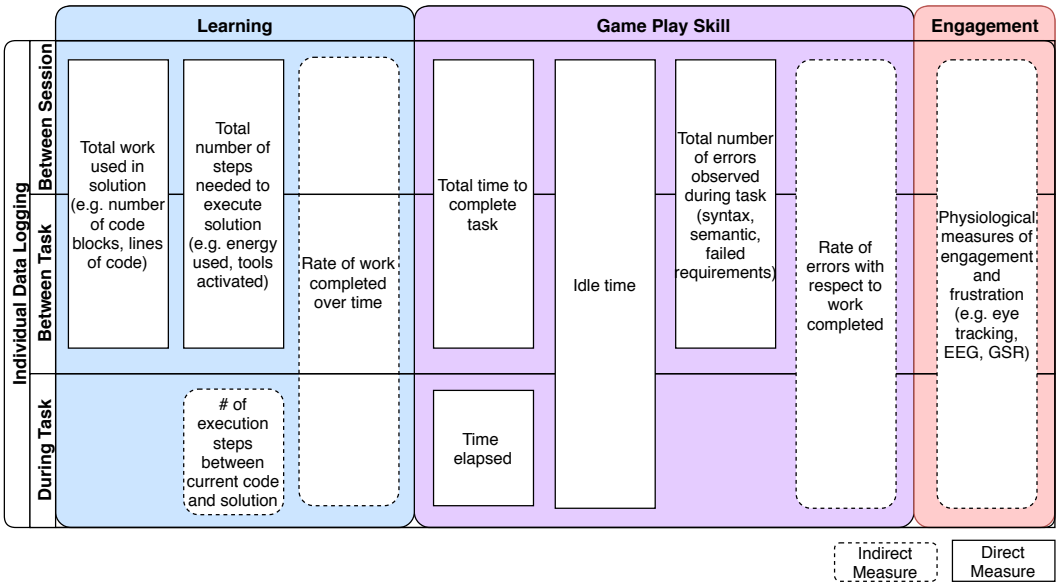


Figure 3.3: Data Logging for Adaptation.

rate in each level, the number of lines of code in each solution, and how much energy is expended per level. The model also needs to consider player engagement and how it is assessed. Maintaining player engagement in serious games is often achieved by varying the complexity of the learning content to challenge skilled learners or to aid learners who are frequently experiencing difficulty. Finally, we include gameplay skill assessment in our model as it is important to distinguish between skilled video game players and players with high competence of learning content. Gameplay skill assessment may be useful in determining if a player's in-game behavior is related to learning content competence, or due to issues with the game's mechanics. Gidget does not include many features related to gameplay skill assessment (e.g., time limit, score tabulation).

3.2.3 Building Adaptation into the Existing Code Base

This phase includes using the models to plan the adaption approach, logging player behavior, initializing the gameplay, and applying the adaptation strategy.

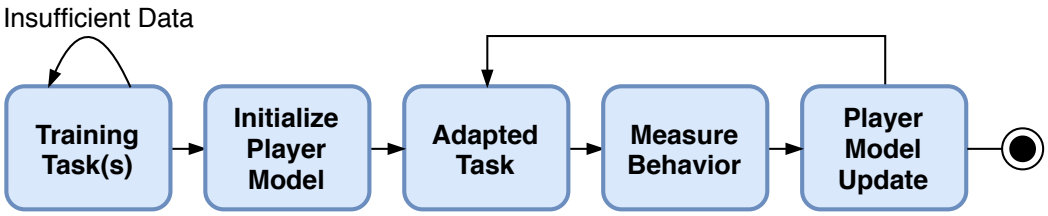


Figure 3.4: Adaptive Gameplay Sequence.

Plan Adaptation. The task and assessment models should be used to determine which game features to adapt. Once these features are chosen, an adaptive algorithm is chosen to determine when, what, and how the tasks are adapted. Example algorithms may be rules-based approach or use machine learning, but should ultimately be probabilistic and follow the principles of CbKST. In Gidget, features for adaptation include the starting code errors, the gameplay obstacles and the energy limits. The adaptive algorithm in Gidget could involve the creation of a set of rules that use the past performance of the player to determine whether or not to adjust the features.

Data Logging. Learner-specific adaptation requires constant logging and measurement of learning data, game skill data, and engagement data (see Fig. 3.3). Depending on the adaptation strategy, data may be gathered for assessment during a task, between a task, or between gameplay sessions. In addition to adaptation, the data logged can be used for evaluating the gameplay experience.

Initialize Gameplay. An initial sequence of the game's tasks should be designated as non-adaptive 'training tasks' in order to initially assess the learner. Following training, an adaptive game should customize each task in accordance with the individual learner's data (see Fig. 3.4). There are several different options that a developer might consider for initializing the training portion of the game: predefined common initialization for all players, self assessment of programming skill (e.g., expert, skilled, unskilled), or game difficulty (hard, medium, easy). As Gidget is targeted towards players with no programming experience, we chose to use a common initialization of

gameplay for all players.

3.2.4 Evaluating the New Adaptive Game

One of the challenges with serious game development in general is the need for accurate and reliable evaluation. One benefit to evaluating adaptive versions of existing serious games is that many of the games have existing evaluation studies that can be replicated and reproduced for the adaptive versions. This allows us to evaluate the benefits of the adaptation by comparing the study results for the original and adaptive versions. If the original serious game did not have a previous evaluation, we recommend following best practices, which may include questionnaires, skill tests, interviews, and controlled experiments.

3.3 Summary of Methodology

There has already been considerable investment in the development and adoption of CS serious games. As best practices for the development of new serious games evolve, it is important that we establish practices to evolve legacy serious games to leverage new ideas and methods. With this goal in mind, we have proposed a methodology for making serious games incorporate learner-based adaptation[1]. Our approach is based on the premise that an adaptive serious game will provide a better experience for learners, and improve their achievement of learning outcomes by directly adapting to their needs. The use of automatic adaptation within a serious game can provide benefits for engagement by adjusting gameplay difficulty to the learner's abilities. This methodology has been applied to two games: Gidget, which will be discussed in Chapter 4, and RoboBUG, which will be discussed in Chapter 5.

[1]This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Chapter 4

GidgetML - An Adaptive Serious Game for Enhancing First Year Programming Labs

4.1 Introduction

Gonczi describes the development of 'competency-based' learning to be where "courses are defined in terms of outcomes to be achieved by students, and assessment of learners is based on the criteria expressed in competency standards. [Gon99]" Although this varies based on country or region, a competency-based approach to learning has many benefits over traditional approaches, particularly for the purposes of linking practice to theory and enhancing student adaptivity. An example of an activity that can give learners practical experience with educational content is a serious game.

The use of serious games has been shown to support student-centered learning, independent learning, actively engage students in their learning process, improve students' self-learner skill, and develop problem solving skills [ZCMM18]. Unfortunately,

serious games have to manage the challenge of player retention, especially when players come from a diverse set of backgrounds and skill levels. The concept of flow [Csi97], which is the suitable increase of challenge with respect to player skill, is extremely important to ensuring that players remain engaged with the game experience.

One way to address player retention is to dynamically accommodate players of different skill levels using adaptation [HBWM+18]. By creating a model of a player's experience, designers can use their game data to alter game content or change game parameters, in order to better challenge skilled learners or accommodate learners who are struggling. This is a form of stealth assessment, which allows the evaluation of learners to occur behind the scenes in a way that does not disrupt a player's flow in the game [SKW17].

The wealth of existing games that help players learn computer science concepts provides a great opportunity to make existing games adaptive rather than having to develop new games from scratch. Unfortunately, not all existing games are necessarily suitable for adaptation. Many serious games published in the literature are not open sourced or available to play, and others simply do not feature a style of gameplay that can be adapted without significant modifications to the source code.

One of the most well known games in the literature for learning computer programming is Gidget, developed by Michael Lee and Amy Ko [LK11]. The Gidget game follows a task-based sequence of gameplay that is uniform for all players, which means it is potentially a good candidate for automated adaptation. Our first research question is:

- RQ1 - Does Gidget benefit from adaptation?

In order to evaluate the effectiveness of the adaptation, we need to look at how player performance changes over the course of gameplay. Specifically, we expect to see that variance among players should be reduced, as the task content is adapted

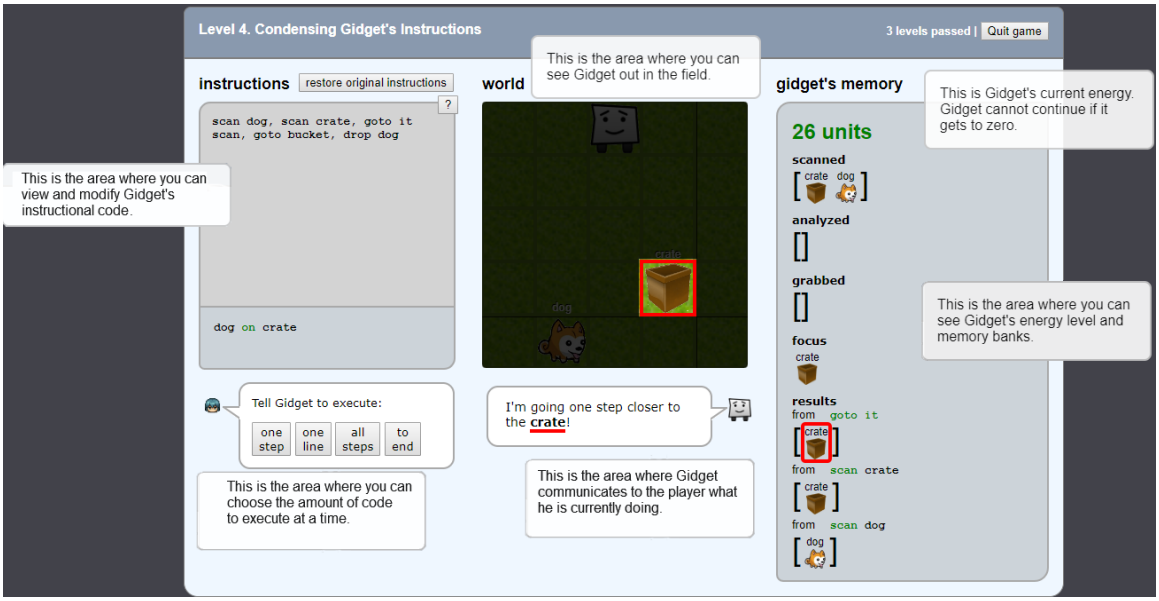


Figure 4.1: The Gidget game (with annotations), where learners help a damaged robot fix its programs by debugging its code [LK12].

to a level of difficulty that is appropriate for each individual learner's competence. In our adaptive version of Gidget, called GidgetML, we hope to answer the second research question:

- RQ2 - Is GidgetML effective at adapting to a learner's level of competency?

In the subsequent sections of this paper, we will discuss some background of serious game and adaptivity literature, how we implemented adaptivity into Gidget, the methodology behind our experimental design, our study results, and a discussion of our findings.

4.2 Background

The use of educational games for programming has grown immensely over time, as designers have been able to increasingly leverage the power of game development

practices. In the survey previously discussed in Section 2.3, we reviewed a different selection of 49 serious programming games as well as the research questions and instruments used to evaluate them [MB18b]. We also considered the survey by Vahldick et al., which categorized 40 serious programming games based on type, platform, educational content, and programming language [VMM14]. From both of these reviews, one of the games that stood out as potentially suitable for the current study was Gidget [LK11].

Gidget (see Figure 4.1) is a 2D puzzle game based around helping students learn to problem solve and fix bugs based on faulty starter code. The goal of the game is to help the titular character to save animals and clean toxic waste through a set of instructions that are followed in sequence. As players proceed through the 18 levels of the game, they learn new commands for Gidget and practice interacting with the environment of the game. Each level requires the player to complete a number of specific tasks efficiently before Gidget runs out of energy. The creators of Gidget found that the game was successful at teaching programming to learners who did not necessarily want to learn programming, and that the debugging-based approach was helpful for avoiding the problem of needing programming knowledge before playing the game [LBK+14]. Gidget has been shown in several studies to improve learning [LK11] and engagement [LKK13], which is why it was chosen as a potentially suitable game for adaptation.

Although programming games have yet to fully leverage the power of automatic adaptation, there are existing games from other educational domains that have incorporated different forms of adaptivity. The 80Days project [GMRS09], which focuses on teaching geographical content and environmental issues, uses Competency-based Knowledge Space Theory and a rule building strategy to choose a path through the game that is appropriate for a given player. SeaGame, a multiplayer game de-

signed to promote best practices in sea-related activities, was used to show how to assign tasks to players using an adaptive experience engine based on computational intelligence [BBDGP09]. We chose an approach similar to that of the ELEKTRA project [KRA10], as Gidget did not lend itself well to the creation of a large number of additional task content, and the use of micro-adaptations was something that could be efficiently implemented in Gidget's source code.

The current paper follows the adaptive game methodology outlined in Chapter 3, as well as our previous work [MB18a]. Recall that the methodology includes a general strategy for how to identify an existing adaptable game, create models of student and task behaviour, build an implementation of an adaptive algorithm into the game, and evaluate the results (see Figure 3.1). The proposed gameplay sequence for players is to begin with a number of training tasks to collect enough player data to begin adaptation. After this, the data is used to initialize a model of the player's behavior, which is used to repeatedly adapt tasks. Changes in player behavior are measured during each task, and the model is updated to include this new information. Subsequent tasks are then adapted according to the player model, for the rest of the gameplay sequence. More details on the adaptations used in GidgetML are available in Section 4.4.1.

In order to evaluate students in our study, we chose to use a combination of game play data and questionnaires that would assess learning and engagement. The questionnaires in our study include a replication of a questionnaire from a previous study on Gidget [LK12], as well as the GEQ [BFC$^+$09]. The GEQ has been analyzed for reliability, validity, and functionality using a combination of Rasch modeling [Ras60] as well as behavioural and questionnaire data. The strength of the questionnaire as a measurement tool was valuable to us to help examine player engagement without the use of physiological measures such as Galvanic Skin Response (GSR) that would

necessitate the use of special equipment.

4.3 Implementation

4.3.1 Gidget

The original Gidget game[1] is intended to help students learn debugging with the help of a personified robot named Gidget. Gidget tells the story of a factory malfunction that has resulted in toxic 'goop' being released and threatening pets and other animals. The player is given control over Gidget's programming, and must write programs in an imperative language that help Gidget to complete the goals in a given level. However, Gidget only has a limited amount of energy, and if the player's program is not able to complete the level goals before Gidget runs out of energy, Gidget will fail and the player will have to try again.

The game includes a total of 18 levels, the first half of which are tutorials for the various commands needed to make Gidget complete the level's goals. This includes the following commands:

- *scan* - the command used for Gidget to load an object into memory in order to interact with it using other commands. Scanning an object costs 1 energy unit.
- *goto* - the command used for Gidget to move to a scanned object. Each step Gidget takes costs 1 energy unit.
- *grab* - the command used for Gidget to pick up an object in the current space. Grabbing an object costs 1 energy. In addition, Gidget's movement using goto commands costs an additional energy unit for each object carried. This means

[1]The source code for the original Gidget game is available at https://github.com/amyjko/Gidget and the latest version of Gidget can be played online at http://www.helpgidget.org/.

players must be careful to choose effective routes through each level, and prevents them from simply picking up all the objects in the level and putting them together.

- *drop* - the command used for Gidget to drop an object in the current space. Many levels require Gidget to transport goop or animals into buckets or crates.

- *analyze* - the command used for Gidget to determine the functions and properties of an object in the current space. Some levels require Gidget to only interact with objects that share a certain trait. Other levels require Gidget to activate the functions of an object, but the name of those functions is not known until the analyze command is used.

- *ask* - the command used for Gidget to call a function from another object. For example, one of the early tutorial levels requires Gidget to 'ask battery to energize gidget', where energize is a function of the battery object. This command cannot be used until the object being referenced has been analyzed.

- *avoid* - a command that can only be combined with the goto command. This allows Gidget to take a path to the destination while attempting not to encounter a specific object. For example, 'goto bucket avoid crack' makes Gidget walk to the bucket object while taking a path that does not include cracks in the ground.

In addition to these commands, Gidget includes conditional statements using the 'if' keyword; for example, 'goto goop, if it isn't glowing, grab it'. Although newer versions of Gidget include functions, iteratives, and other introductory programming concepts, the version that was accessible to us on Github was older and did not have these features.

Players are introduced to each command in a tutorial level that includes incorrect starter code. The learning in Gidget comes from players executing the incorrect code, observing the behaviour, and determining how to modify the code to create a correct solution. Later levels of the game require a great deal of testing, especially when some of the functions and properties of objects needed to complete a level are not available until the analyze command has been used in several instances. Players complete the game after the 18th level, when they manage to close the leak in the 'goop' factory and rescue all of the animals in danger.

4.3.2 GidgetML

The first steps taken to create GidgetML[2]. were to model the player as well as the levels in Gidget. In the process of completing a level, a player will likely attempt multiple solutions at the puzzle, and will likely fail repeatedly until coming upon a working solution. Using this data, we chose to use the number of failures as well as the energy (i.e. program steps) used in successful solutions as our way to classify student competence. We chose not to consider time as a factor due to a lack of incentives in the game for learners to play at a quick pace.

In order to classify students, we chose a k-means clustering approach based on the failures and energy expenditures of each player on each level. We chose to limit the number of clusters to 3 in order to have a sufficient number of candidates in each cluster during the training phase. The clusters were ranked based on their normalized feature vectors, giving us a "low", "medium", and "high" categorization for each player. We only used complete data sets from our training data in the clustering algorithm, however there are ways that missing data can be handled [LDSS04]. During gameplay, a k-means clustering is performed using the available data from the current

[2]GidgetML is open-sourced and available on Github at https://github.com/sqrlab/GidgetML

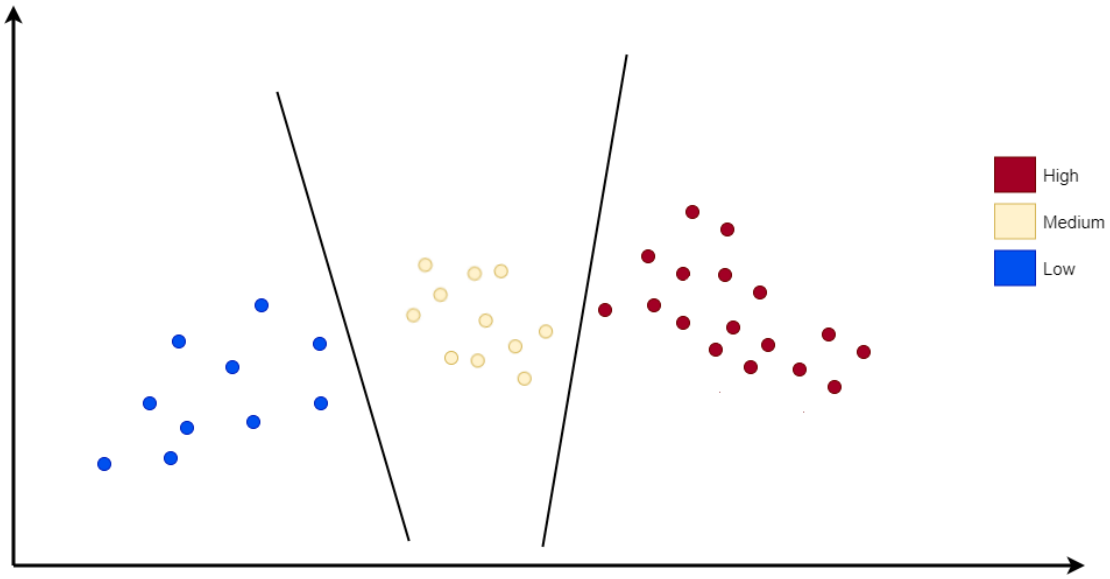


Figure 4.2: K-means categorization. Each player is categorized into one of three groups (high, medium, low) based on their gameplay data.

player in comparison to the previous players who have been already categorized. The other players in the cluster containing the current player are then examined for their previous categorization; the current player is given the same categorization as the largest group from their current cluster. An example of this can be seen in Figures 4.2 and 4.3.

A level is comprised of an environment of game objects, a list of goals to be completed, incorrect starter code to be modified, and an initial energy limit that dictates how many commands can be taken before Gidget fails to complete the level. Although all of these level parameters could be altered automatically, we chose to focus on changing energy limits and starter code, as these adaptations would allow optimal solutions to be accepted regardless of adaptation status. Alterations to the environment in particular would have required a time-intensive amount of manual work, and we wanted to avoid adapting features that might not be easily applicable to other similar games. Overall, the total number of lines of code added or removed

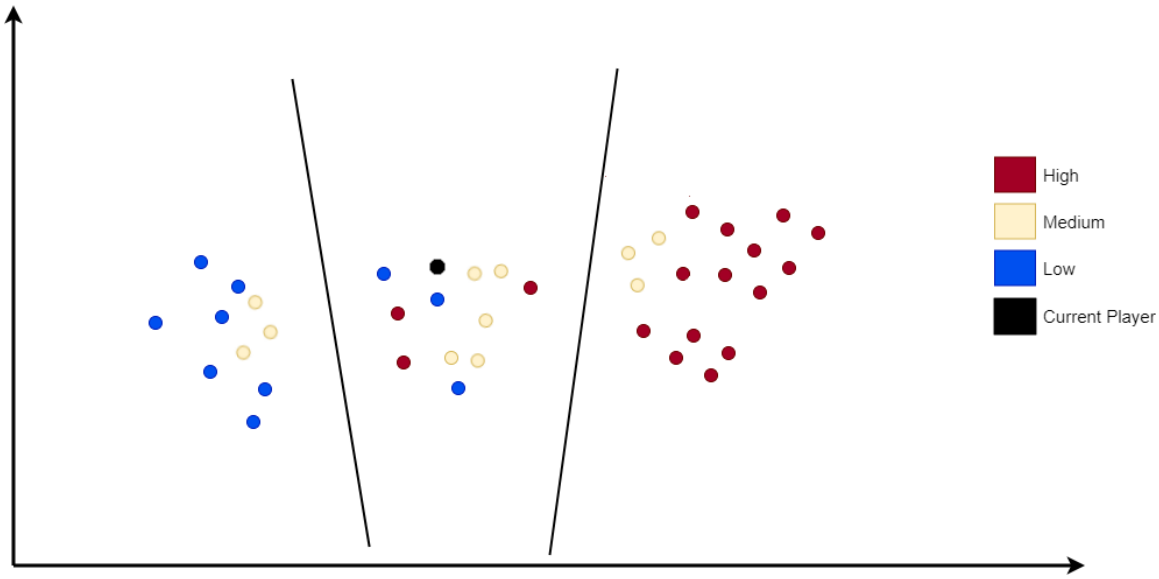


Figure 4.3: K-means categorization during gameplay.
Among those in the same category as the current player, the plurality were originally categorized as in the medium competence group, so the current player is also categorized as medium.

in the Gidget code base during the modification was 1017 out of the original 11800 total lines of code, meaning that less than 10% of the code needed to be modified.

To match the number of clusters from our k-means approach, we developed three variations on each level in the game after the first three tutorial levels, which are used as a starting point of player data. When a player is categorized as "low", "medium", or "high", the player's next selected level is set to the variation that matches their categorization; however, if a player's categorization changes from one extreme to the other (e.g. High to low, or low to high), they are instead set to the "middle" categorization. The difference between low and high adaptations can be observed in Figure 4.4.

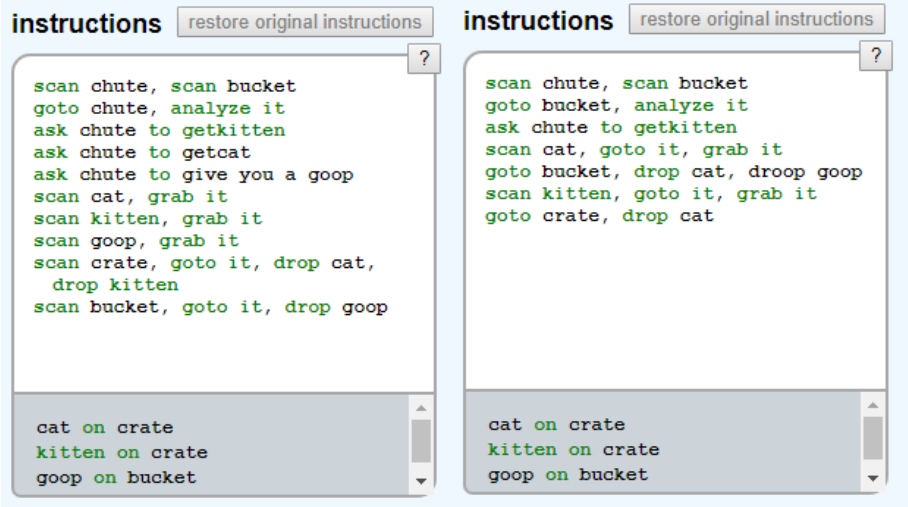


Figure 4.4: Adaptivity in the Gidget game.

The starter code on the left side of the screen is presented to students who are categorized in the low competence group, while the code on the right is for those in the higher competence group. A correct solution to this level is to change the line of code on the left from "ask chute to give you a goop" to "ask chute to getgoop".

4.4 Methodology

4.4.1 Adaptive Methodology

Recall that the adaptive methodology we presented in Chapter 3 has a four phase process:

1. Identifying a serious programming game for adaptation
2. Modelling and connecting tasks with in-game assessments
3. Implementing adaptation into the existing code
4. Evaluating the adaptive game in comparison to the original version

In the identification phase, we selected from among those games listed in the review by Vahldick [VMM14] and our own review [MB18b]. Since our target audience

were university students with no programming experience, we only considered games that were appropriate for that group. Many of the games listed in the reviews did not have available source code, which further narrowed our search. Finally, among the remaining games, Gidget was selected as the one that had the most prior research and evidence supporting its efficacy as a serious programming game.

During the modeling phase, we identified game tasks as having the following features:

- Instructions (starter code)
- Goals (requirements for task completion)
- World (environment conditions)
- Energy Units (number of steps permissible before failure)
- Order (levels were numbered 1 to 18)

In our survey, we noted that debugging is the primary concept covered in Gidget [MB18b] . In order to facilitate the activity of debugging, we chose starter code as one of the features that would be adapted during game play. Energy units were also chosen as a way to adapt the game, as this adaptation would allow low competence players additional flexibility in their solutions while restricting the solution set for high competence players.

For player behaviour, we identified failed attempts, energy used in solutions, and time elapsed as potential options for data analysis. However, since players were not incentivized to play the game as quickly as possible, we chose to only examine failures and energy usage for adaptation. We chose to weigh these features equally in terms of assessing player competence.

For implementing the adaptation, we created three versions of each level associated with low, medium, or high competency. We determined the amount of energy required to complete each level with an optimal solution, and reduced the energy limitations for less competent groups. The starter code for the lowest competency group was set to be within 1-2 statements away from a valid solution, while the highest competency group would receive code that often required each line to be modified at least once.

Our evaluation of GidgetML in comparison with the original version of Gidget is described in the following section.

4.4.2 Experimental Design

With permission from the course instructor, we designed a study for using Gidget and GidgetML in a first year programming course at Ontario Tech University, before students began to complete any of their lab assignments. An overview of the experimental design is shown in Figure 4.5. Course labs took place on multiple days of the week, so we chose to divide the class by having the three labs on the first day (54 students) complete the original non-adaptive version of Gidget, and the remaining labs (93 students) complete GidgetML.

After each lab completed their play sessions, the categorizations of past player data were updated using k-means clustering. This means that players from the final lab had their data compared to players from all previous labs, including those in the adaptive sessions. During these updates, each of the three categories were ranked based on their normalized failure rates and energy amounts used per level.

After completing the game, players were directed to a questionnaire with the same questions from Lee & Ko's study on engagement in Gidget [LK12]. Participants were asked for their age, gender, and how much past experience they had with programming. We added the GEQ to this questionnaire to see if we would find any

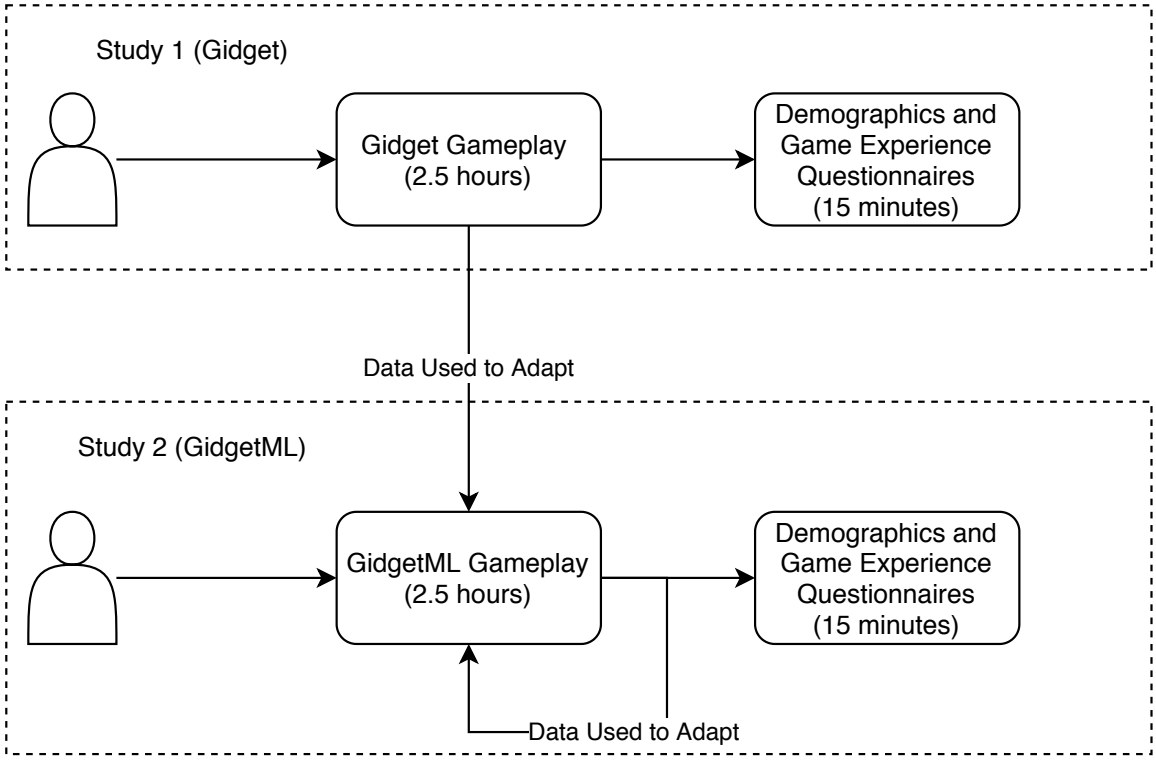


Figure 4.5: Evaluation methodology for both Gidget studies.
Study 1 used the original version of Gidget, and data was collected from the questionnaires provided to participants as well as game play data logged during the 2.5 hour gameplay session. Study 2 used the logged data from Study 1 to assess participants in the adaptive GidgetML game, and the data collected from gameplay was used to continuously adapt GidgetML for future participants.

significant affective differences between the Gidget and GidgetML groups [BFC+09]. The questionnaire included the following questions which students answered with a Likert Scale (Strongly Disagree to Strongly Agree):

- "I enjoyed playing the game"
- "I would recommend this game to a friend wanting to learn programming"
- "I wanted to help Gidget succeed"

The questionnaire also included the following open-ended questions, which were independently coded as 1 (positive), 0 (neutral), or -1 (negative):

- " Describe your feelings about the dialogue between yourself and Gidget."
- "Describe your feelings towards Gidget's avatar (image)."
- "How was this learning experience different, if at all, from any previous experience you have had dealing with programming?"

A student's t-test was used to examine significance between the Gidget group and GidgetML group on each feature of our data.

4.5 Results

Out of 147 students registered in a first year class, 100 took part in the study. The group of participants from the first day of laboratories was labelled the Gidget group, totalling 32 students, and those from the remaining labs were labelled the GidgetML group, totalling 68 students. The average age of participants was 18.49. 81 students identified as male and 12 identified as female; details on the genders (e.g. transgendered) of the remaining 7 students have been kept hidden to protect anonymity.

77 students had taken a computer science course before (84% of Gidget, 74% of GidgetML), 16 students had experience making a website from scratch (19% of Gidget, 15% of GidgetML), 70 had written a computer program before (75% of Gidget, 78% of GidgetML), and 10 students had written or contributed to developing software (13% of Gidget, 9% of GidgetML).

We did a keyword search through participants' responses to our questionnaires, searching for the words 'hard', 'difficult', 'challenge', and 'frustrate', and found the following results:

Interesting quotes from Gidget Group:

- *"Really great and difficulty as levels progressed was challenging."*

- *"The challenge of the game kept me interested in it but not being compensated for my efforts left me unwilling to continue further."*
- *"There was no clear dialogue on how to obtain the battery to complete the level."*
- *"The game was getting difficult to understand in what I had to accomplish and what commands I had to use in order to finish certain steps which was frustrating."*
- *"Frustration and not knowing why the code wasn't working. Maybe after a few tries a hint could be provided."*
- *"Either the level was too hard or there was too much dialogue."*
- *"Difficult to understand at first made sense after a few tries."*
- *"I got frustrated."*
- *"It seemed frustrating sometimes because I had difficulty comprehending what Gidget was trying to say."*
- *"Got too hard!!"*
- *"It felt like manual debugging and I definitely felt the same kind of frustration as when a program doesn't work."*
- *"Despite the frustration I enjoyed playing the game with my friends, we got some laughs out of it."*

Quotes from Adaptive Group:

- *"...increasing difficulty throughout the levels of the game."*
- *"I ran out of time and there was a sharp difficulty spike on level 18."*

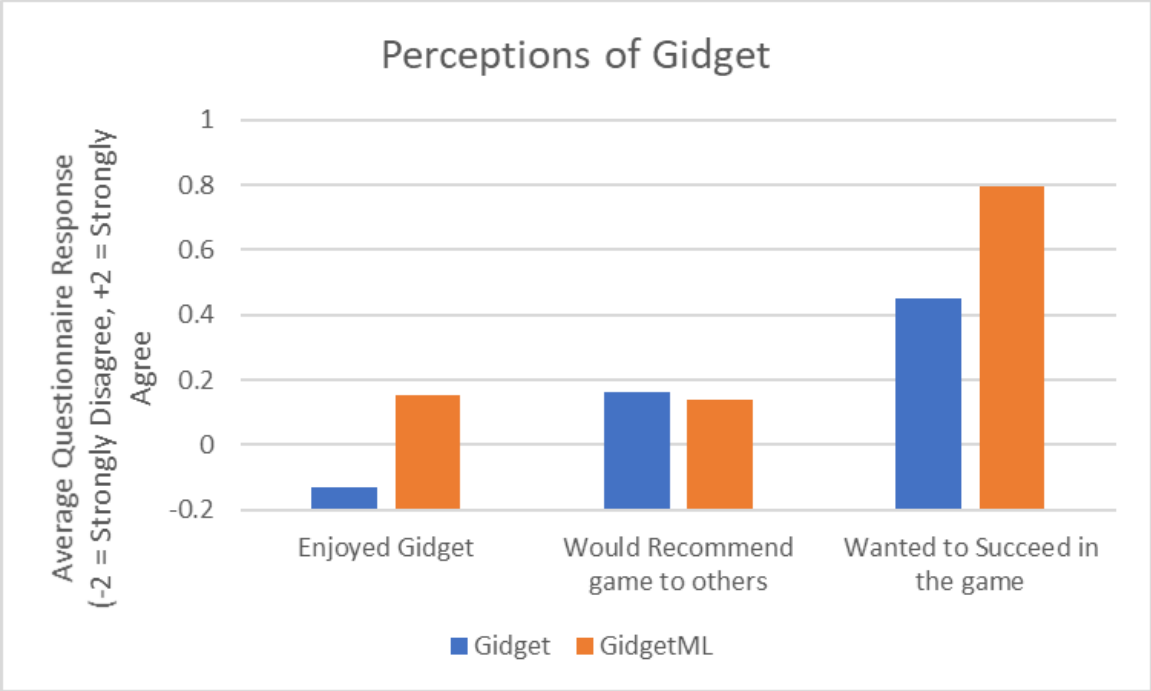


Figure 4.6: Perceptions of Gidget

- *"It was more difficult [than real programming] because I found the commands confusing."*
- *"I like the challenges. They're small and cute so I would love to play more."*
- *"Gidget was very friendly and I felt bad when I could not complete particular levels."*
- *"The game became challenging and I felt like I was confused on what the level was asking me."*
- *"...it was frustrating at times but it was still a good experience."*

Results from student perceptions of Gidget are shown in Figure 4.6. Students' expressed enjoyment of Gidget was relatively neutral (mean=0.063, std=1.044), but students in the GidgetML group (mean=0.154, std=1.034) enjoyed the game more

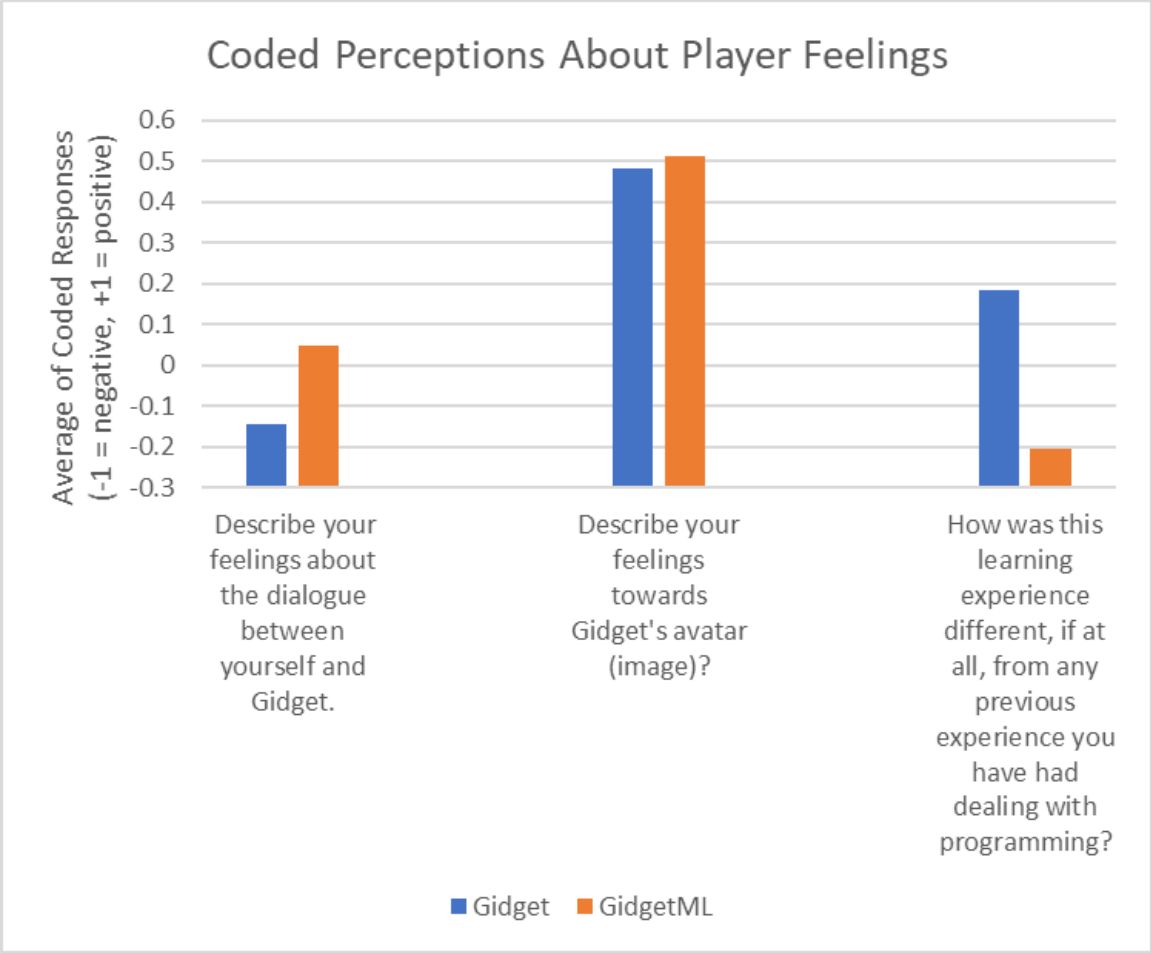


Figure 4.7: Coded perceptions of Gidget

than those in the Gidget group (mean=-0.133, std=1.074). This result was not significant (t=-1.242, p=.109). Students overall would recommend the Gidget game (mean=0.147, std=1.076), regardless of whether they were in the GidgetML (mean=0.141, std=1.067) or Gidget group (mean=0.161, std=1.128). This result was not significant (t=0.087, p=.465). Students generally wanted Gidget to succeed (mean=0.681, std=1.013), although this effect was stronger in the GidgetML Group (mean=0.794, std=0.986) than the Gidget group (mean=0.452, std=1.060). This result was almost significant (t=-1.542, p=.063).

Results from coded student perceptions of Gidget are shown in Figure 4.7. Re-

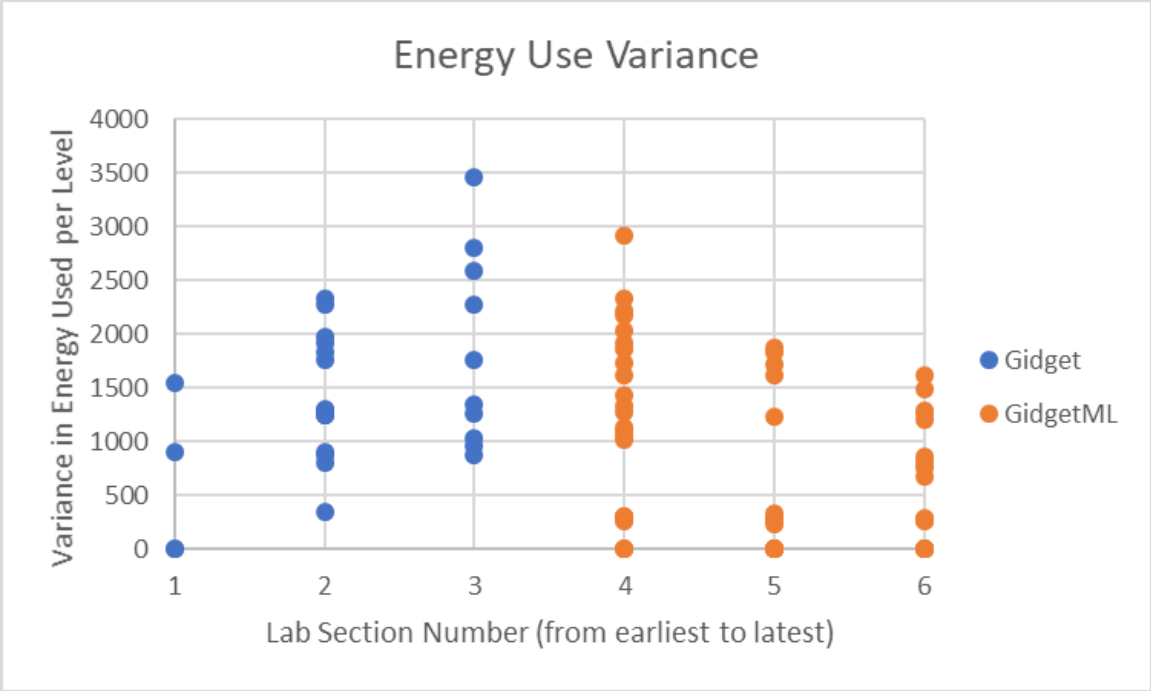


Figure 4.8: Variance of Energy used in Solutions

sponses were coded as 1 for positive responses, 0 for neutral responses, and -1 for negative responses. Students overall had a neutral impression of Gidget's dialogue (mean=-0.0294, std=0.840), with students in the Gidget group having a slightly more negative opinion (mean=-0.143, std=0.756) towards it than those in the GidgetML group (mean=0.050, std=0.904). The difference between groups was not significant (t=-0.924, p=.179). There was generally a positive opinion of Gidget's avatar, regardless of group (mean=0.500, std=0.738). The difference between groups was not significant (t=-0.164, p=.435).When comparing the game to real coding activities, the game was overall viewed neutrally (mean=-0.045, std=0.824), but those in the Gidget group (mean=0.185, std=0.879) significantly (t=1.915, p=.030) preferred the game over past experiences with programming than those in the GidgetML group (mean=-0.205, std=0.767) .

Student data on the variance of how much energy (steps) was needed to complete

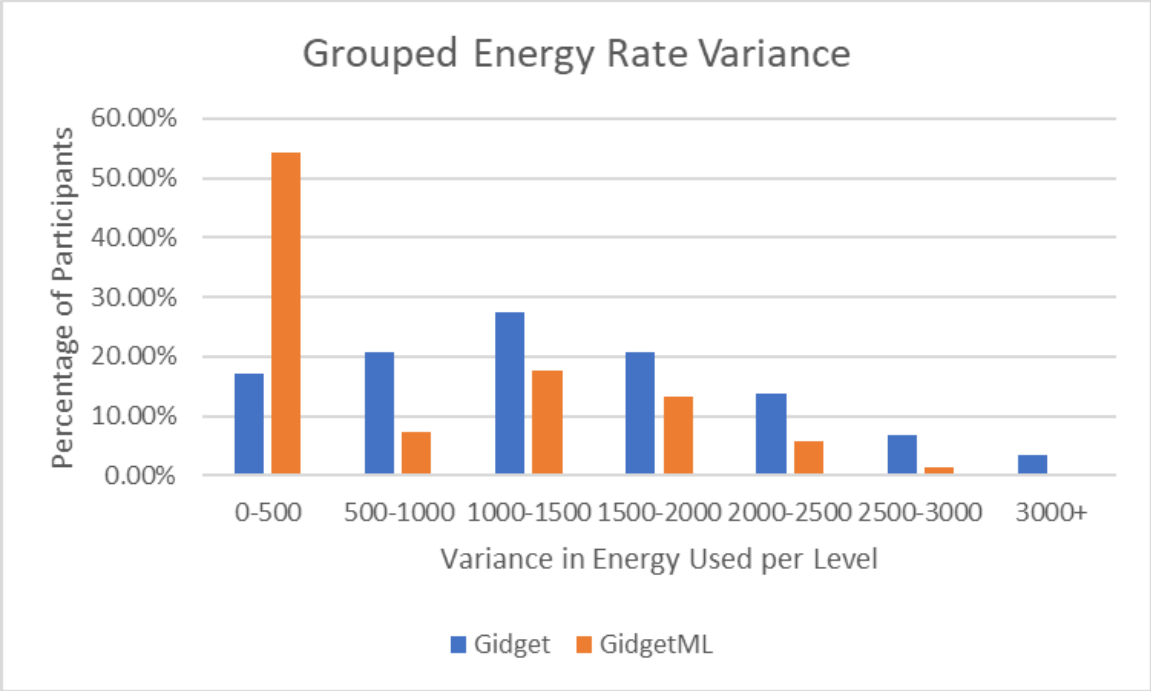


Figure 4.9: Grouped Variance of Energy used in Solutions

levels is shown in Figures 4.8 and 4.9. Figure 4.8 shows how each lab differs on the variance of energy used in their program solutions. The Gidget group, from labs 1-3, had a significantly higher average variance (mean=101, std=103) than the GidgetML group of labs 4-6 (mean=32.7, std=34.5). Figure 4.9 illustrates the difference between the two groups. The difference between groups was statistically significant (t=3.842, p=.000108).

Student data on the variance of number of failures per level is shown in Figures 4.10 and 4.11. Figure 4.10 shows how each lab differs on the variance of energy used in their program solutions. The Gidget group, from labs 1-3, had a significantly higher average variance (mean=1411, std=827) than the GidgetML group (mean=765, std=763) of labs 4-6. Figure 4.11 illustrates the difference between the two groups. The difference between groups was statistically significant (t=4.949, $p < 0.00001$).

Figure 4.12 shows the responses of students to the GEQ. On average, students

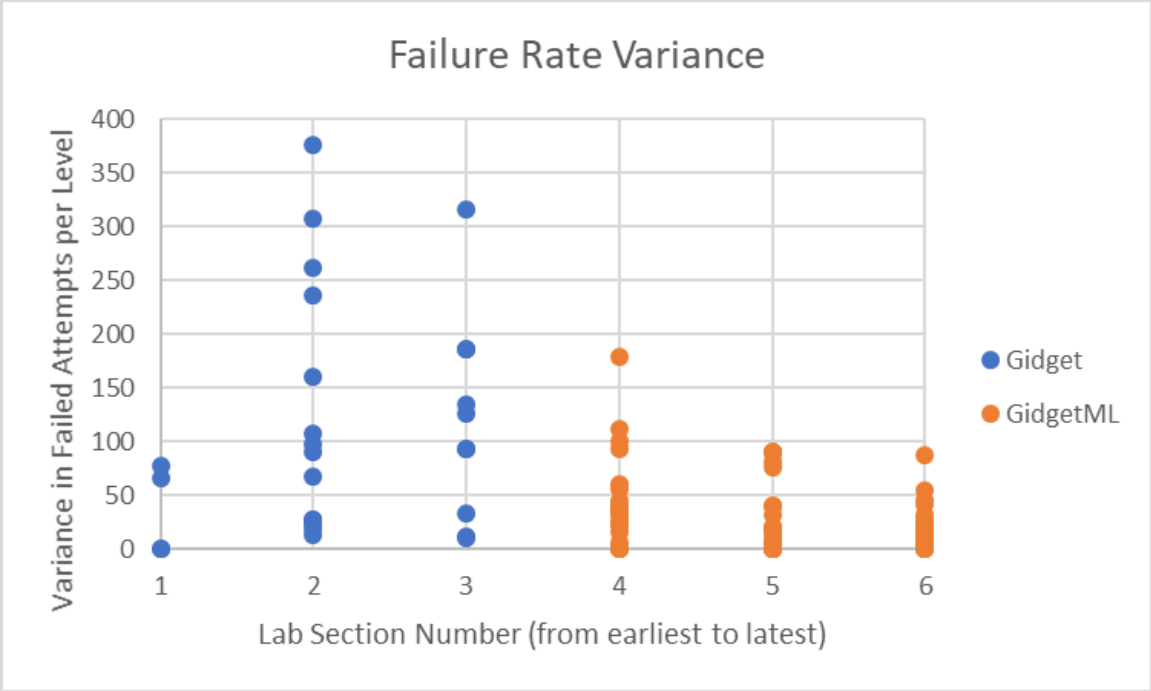


Figure 4.10: Variance of failures per level

scored low (below 2 out of 4) on all of the GEQ items. Interestingly, this means students felt neither a significantly negative or positive experience. There was no significant difference found between the two groups.

4.6 Discussion

- RQ1 - Does Gidget benefit from adaptation?

To answer this question, we looked for evidence that would inform us about the need for Gidget to be adaptive. Although we had the option of having students self-report whether Gidget was appropriate for their competency level, we instead opted for an analysis of the logged data available from game play. In Figures 4.8 and 4.10, we show how the Gidget group's variance was both high in value as well as highly varied between participants. This high variance illustrates that the performance of

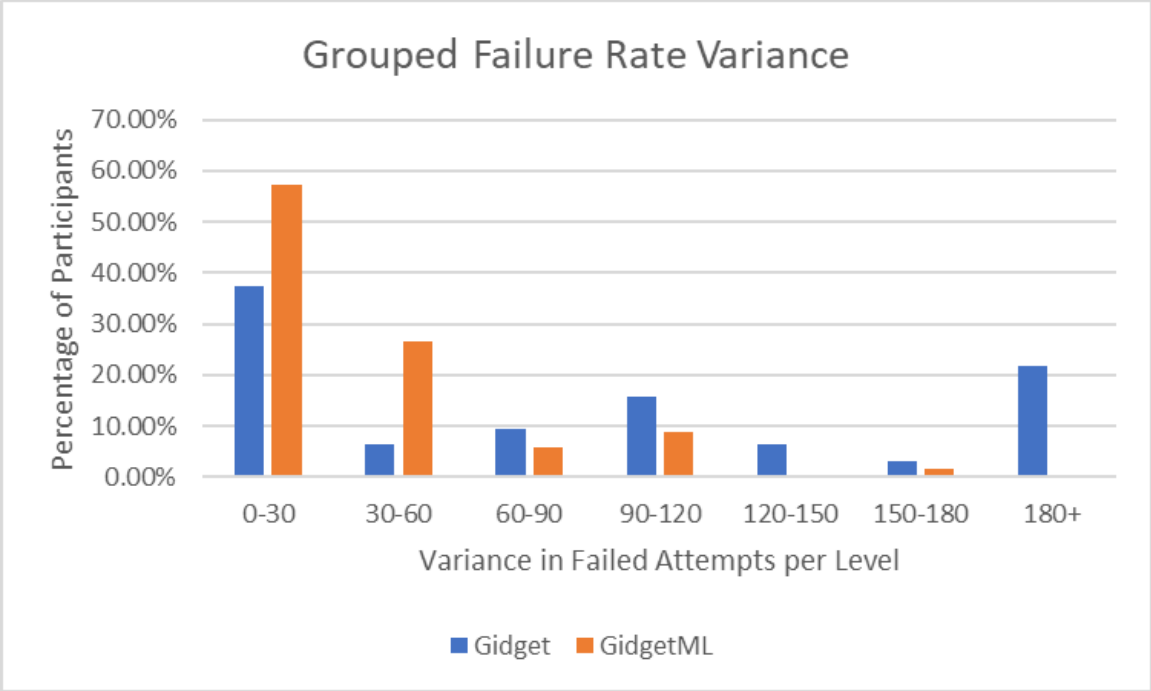


Figure 4.11: Grouped variance of failures per level

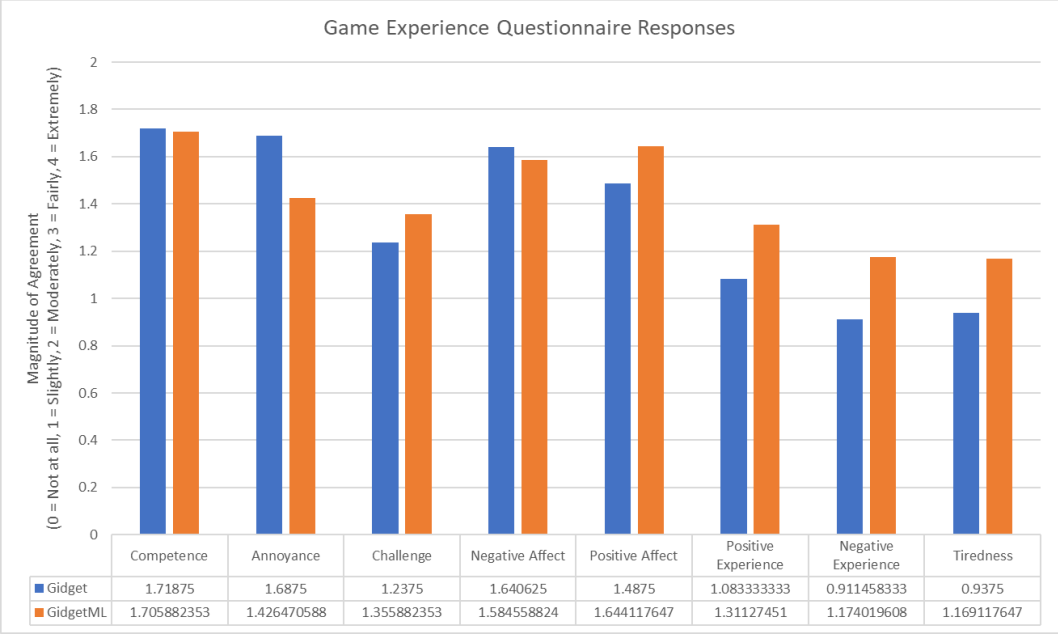


Figure 4.12: Responses to GEQ [BFC+09]

students differed greatly from level to level, suggesting that Gidget could benefit from adaptation.

- RQ2 - Is GidgetML effective at adapting to a learner's level of competency?

Using our methodology from Chapter 3, we sought to answer this question by determining if an adaptive version of Gidget could reduce the variance in participant failure rates and energy usage. A reduced variance would indicate that the difficulty of the game was more consistent with respect to the competence of the participant. From Figures 4.9 and 4.11, the change observed in variance between the Gidget and GidgetML groups suggests that our adaptation had a significant effect on students, and the different coded responses support the idea that the Gidget group had more negative experience with the game than the GidgetML group. Although we did not see significant differences on answers from the GEQ, this may simply be due to players not having strong emotional feelings about the game one way or another. In Figures 4.8 and 4.10, we can also see that the variance of the GidgetML group seemed to decrease over time, as we had access to more data for use in adaptation. This is a promising observation that with additional data, GidgetML could be even better at adapting to learner competences.

4.7 Threats to Validity

We encountered several challenges while conducting this study. First, the length of the original Gidget game was a large hurdle for many students in the classroom environment, as many were not able to complete all of the game's levels. Second, the k-means clustering algorithm used in GidgetML was a challenge, especially when extreme outliers were introduced into the data set, which caused issues for low amounts of student data. Third, our work could have benefited by gathering a larger, more varied population of participants from different schools and countries rather than a population in a single class at a single university. Although most students reported

similar backgrounds in computer programming, there were most likely differences in specific content that they learned from their prior education at different high schools.

We chose to adapt Gidget because it was a third-party developed game. This was a deliberate choice as adapting a game that we created ourselves would potentially introduce bias into our research. However, a side effect of this choice was that some of the feedback from participants was related to issues that arose from the original game independently of our adaptive implementation. Furthermore, while Gidget is similar to other existing serious programming games in its delivery of tasks and imperative coding style, there are many other games that have a significantly different style of gameplay that would require alternative adaptive strategies.

Although we followed our general adaptive methodology for serious programming games [MB18a], some of our implementation choices were specific to Gidget; in particular, the starter code and concept of energy units were unique to the game and would not be present in other serious games. However, the principles of modifying starter code or putting restrictions on the number of permitted steps in a solution are generalizable to other serious programming games. Any game that requires players to program a sequence of steps to solve a puzzle can have these elements, and thus might be adapted by a similar implementation to the one we used for GidgetML.

4.8 Summary & Future Work

We found that Gidget had the potential to benefit from adaptation, as observed through the high levels of variance among learners who played the game. GidgetML, our adaptive version of Gidget, significantly decreased the observed variance, without extensively changing the content in the original game. In the short-term, our upcoming research will investigate correlations between the use of adaptation and student

grades. Although grades on lab scores do not necessarily indicate learning, it would be useful to know if there are correlations between student academic evaluations and the adaptive assessments, especially in a longitudinal approach. We also hope to conduct another study with a more balanced population, to determine if there are differences in adaptation benefits when considering demographics such as gender.

In the future, it would also be interesting to explore variation on GidgetML by looking at the use of different adaptable game features such as time and different adaptation strategies (i.e. machine learning algorithms). In particular, it would be valuable to explore adaptation strategies that can learn from incomplete data sets of student gameplay. Another avenue of future work is to explore GidgetML in different contexts including outside the classroom and in scenarios where students replay the game. Finally, we plan to continue exploring the benefits of our adaptive methodology with other games that help students learn programming.

Chapter 5

Adapting Game Play and Hints in RoboBUG

5.1 Introduction

The first version of RoboBUG was developed during my master's thesis with the aim of addressing student challenges with debugging through an exploratory setting that does not require students to write code [Mil15] . The inspiration for RoboBUG came about while reviewing the serious game literature - it was evident that most games in the literature required students to write source code to solve problems, and few games involved students comprehending and analyzing code written by someone else. In an industry setting, most source code is written by teams of developers rather than a single individual. Furthermore, a significant amount of debugging in industry requires an understanding of another developer's source code in order to locate errors and fix bugs. Expert debuggers have an advantage over novices both in terms of familiarity with bugs as well as proficiency in **debugging techniques** [GO86]. In addition to a lack of emphasis on debugging in serious games, it was also observed

that in general, debugging and its relevance to programming is not always reflected in university computer programming curricula. Although debugging is included as a topic in the ACM/IEEE-CS curriculum [For13], it is rare to see a university course focus extensively on debugging.

RoboBUG is designed for students who have already learned the fundamental concepts of programming but are not experts in debugging. These students often find debugging to be frustrating and difficult, especially when they also are not the creators of buggy code [CL04]. RoboBUG introduces four different debugging techniques with real world applications, and aims to improve novices' competence with debugging by requiring them to apply these techniques to progress through the game.

As part of my PhD research, I conducted a previous evaluation of RoboBUG which found that it helped students to achieve learning outcomes, particularly for those who were not initially skilled at debugging (See Appendix A.3) [MB17]. Although the evaluation is not a primary contribution of this thesis, it was essential for RoboBUG to meet the methodological requirements discussed in Chapter 3. The game itself is relatively short and was designed to fit in the time frame of an undergraduate computer science lab (approximately 60 minutes). The game is divided into four different 'chapters', each based on a different debugging technique. Players are permitted to choose which levels they would play, and are are not required to start at the first level and proceed through a pre-determined path from beginning to end. My previous evaluation of RoboBUG, combined with its flexible customizability, make it a suitable game for adaptation.

As with the previous work on GidgetML, two research questions needed to be answered:

- RQ1 - Does RoboBUG benefit from adaptation?

- RQ2 - Is the adaptive version of RoboBUG effective at adapting to a learner's level of competency?

In addressing these research questions, we chose to adapt RoboBUG in two different ways:

1. A variant of RoboBUG that is similar to the adaptation used in GidgetML where game play content and source code provided to students was adapted,

2. A variant that provides adaptive hints to players at different frequencies based on their level of competence.

The purpose of these two variants was to answer a third research question:

- RQ3 - Do adaptive hints better support learning and engagement for players than adapting game content?

In the subsequent sections of this chapter, I will discuss the background behind RoboBUG, the implementations of the two variants, the methodology behind the experimental study, the results from the study, and a discussion of the findings on making RoboBUG adaptive.

5.2 Background

5.2.1 Serious Debugging Games

The survey of serious games presented in Section 2.1 found that very few games directly address the challenge of debugging. Most programming games, such as Lightbot [GBW13], only incorporate the activity of debugging implicitly, where players can debug their code if their first attempt at a solution is incorrect. However, there is

nothing to prevent players who make mistakes from starting a new solution from scratch and avoiding the debugging process entirely.

There are some games where this behavior is not a viable option for players. One example is Code Hunt [BHX+15], where players rely on a table of input and output values to generate a correct source code solution. Although the code is never intended to be syntactically, semantically, or logically incorrect, players must still use problem solving skills akin to those used for debugging to determine how to change their code into one that matches the desired inputs and outputs. However, Code Hunt is still not a debugging-focused serious game, in that the application of debugging and the methods used are ad-hoc.

Perhaps the most well known game that was designed to help with learning debugging is Gidget [LBK+14], discussed previously in Chapter 4. Recall that in Gidget, players are presented with incorrect code at the start of each level, and the intent of the game is for students to observe the incorrect behavior of the code, and use that knowledge to create a correct solution. However, even Gidget has some limitations when it comes to learning debugging. Firstly, some players may decide to delete all of the starter code and create a solution from scratch, which removes the debugging activity entirely. We observed this particular behaviour in our previous study on GidgetML. Secondly, the strategies needed to debug levels in Gidget do not encompass many of the debugging techniques used in real world debugging. These limitations were part of the inspiration for the creation of the original RoboBUG game, which addressed the limitations by removing the code modification component of the debugging task, and by incorporating explicit debugging strategies into the game.

5.2.2 Debugging Techniques

An empirical study of novice debugging patterns discovered that "many students with a good understanding of programming do not acquire the skills to debug programs effectively, and this is a major impediment to their producing working code of any complexity" [AEH05]. This is explained partially by inconsistent knowledge in novices with weak debugging skills and a lack of familiarity with debugging techniques. One of the key factors that determine the ability of a good programmer to also be a good debugger is knowledge of the actual program implementation [AEH05]. This is particularly relevant when novices are debugging code that was written by someone else, as they are far less likely to fully understand the intention and behavior of that code.

An exploration of how novice programmers pick debugging tactics [Lee19] considered the following debugging techniques:

- generating program output (e.g. print statements)
- tracing by hand (i.e. code tracing)
- asking other programmers for help
- searching for external resources (e.g. documentation)
- rewriting code
- testing code with sample input
- adding comments
- applying undo button in a Software Development Environment (SDE)
- searching for code with SDE

- writing a to-do list on paper

These tactics vary in their efficacy from situation to situation, and it is unclear how novices decide which strategy to employ. Among the tactics listed here, RoboBUG helps novices to learn how and when to use code tracing, print statements, divide-and-conquer strategies using selective commenting, and also breakpoints which are not as widely used by novice programmers.

5.2.3 Use of Hints in Serious Games

Hattie and Timperley state that "Feedback is one of the most powerful influences on learning and achievement, but this impact can be either positive or negative" [HT07]. It is most useful in situations where the feedback addresses faulty interpretations, and less so when there is a complete lack of understanding by learners, or when learners become overly reliant on the feedback to solve a problem. Hattie and Timperley distinguish between four different levels of feedback [HT07]:

- **Feedback about a task** (FT), such as whether the work is correct or incorrect
- **Feedback about processes** (FP), such as suggesting a specific strategy to complete a task
- **Self Regulation** (FR), such as reminding a learner that they know a certain concept, and asking them to verify the concept was used in a solution
- **Feedback about the self** (FS), such as praise and encouragement

Ideally, feedback should follow a path from FT to FP to FR, with minimal use of FS. However, Simmons and Cope found that when FT was too specific, learners were not able to develop their understanding [SC93]. If feedback was immediate and overly

specific, learners instead would focus on the current goal moreso than the strategies required to attain the goal, leading to trial and error strategies with less effort on the intended learning.

In the context of a serious game, which can be considered a form of ITS, human intervention to aid learners can be substituted by AI agents that facilitate knowledge development. Goldberg and Cannon-Bowers found that "a visible agent in a learning environment significantly impacted motivation outcomes when compared to voice and text only feedback conditions" [GCB15]. This feedback was only provided to learners when their actions met specific critical thresholds, such as when a learner was identified as missing a step in a procedure. In this case, a pre-defined rule set was used to determine if intervention is required in real time. Serious programming games such as BOTS have incorporated this sort of intelligent feedback in the form of personalized hints, which were provided automatically based on the sequence of game actions taken by a player [HPB14]. However, the intelligent feedback of BOTS has not been evaluated for its efficacy with respect to learning or engagement.

5.3 Implementation

5.3.1 RoboBUG

RoboBUG (see Figure 5.1) is a serious game based on the activity of source code debugging. The original version of RoboBUG can be seen in Figure 5.2. This older version of RoboBUG featured a plotline where players controlled a robotic avatar that tried to exterminate invading bugs, but this was updated with the current, more inclusive theme. The physical bugs that had 'infested' the code were changed into game obstacles that would damage the player if they collided, and a 'hacking' activity was added to allow the player to reveal hidden code. As a reward for completing

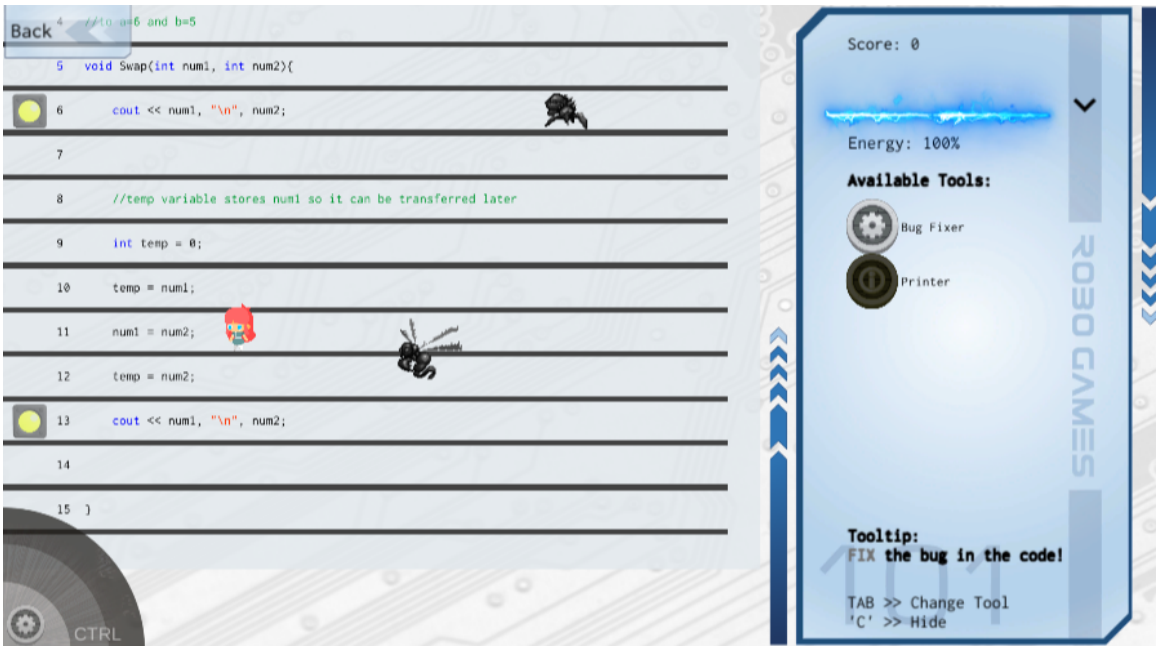


Figure 5.1: The latest version of the RoboBUG game used in this chapter.

levels quickly and with fewer mistakes, players are awarded points that can be spent on various upgrades, including increased speed or resistance against obstacle damage. Aside from these small changes, the core gameplay of RoboBUG has remained the same since conception.

In the current version of RoboBUG, players take the role of a boy named Guy, a girl named Ivy, or a robot named V.I. (short for Virtual Intelligence), and attempt to thwart a programmer villain named Black Hat, who is creating malicious bugs to attack the nearby university. The characters in the game are shown in Figure 5.3. The game is divided into four 'chapters', each of which addresses a different debugging technique. Each chapter can be completed independently of the others, and are not designed to be increasingly difficult, although some chapters may be more challenging than others. The chapters are divided into smaller 'levels', which are designed to build upon each other and provide the opportunity to apply the chapter's debugging technique in different ways. To complete a level, a player must apply the technique

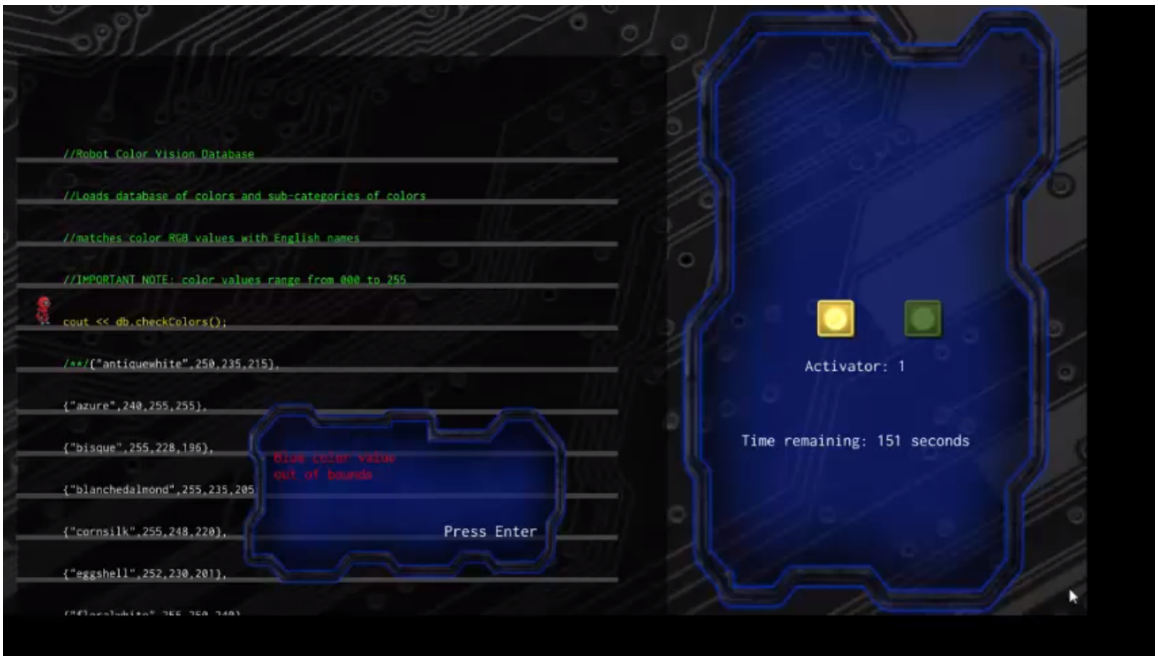


Figure 5.2: Original version of RoboBUG.

to locate a single faulty line of code in a short program, and indicate the location using a 'bugfinder' tool. Correctly identifying the location of the bug ends the level successfully, while falsely flagging a location with a bug results in the player failing and needing to repeat the level. As the player progresses through the game, they are introduced to additional tools that allow them to trigger print statements, 'warp' from one program to another, comment out specific blocks of code, or activate breakpoints for observing run-time variables. Importantly, none of the bugs in the game are syntax errors, but instead the game focuses on logical and semantic errors, which are more difficult to detect.

Chapter 1 - Code Tracing

The initial chapter focuses on code tracing, which is the technique that requires the fewest in-game tools. The game's story involves the main characters overheating, as the university's temperature controls were bugged and not correctly calculating a

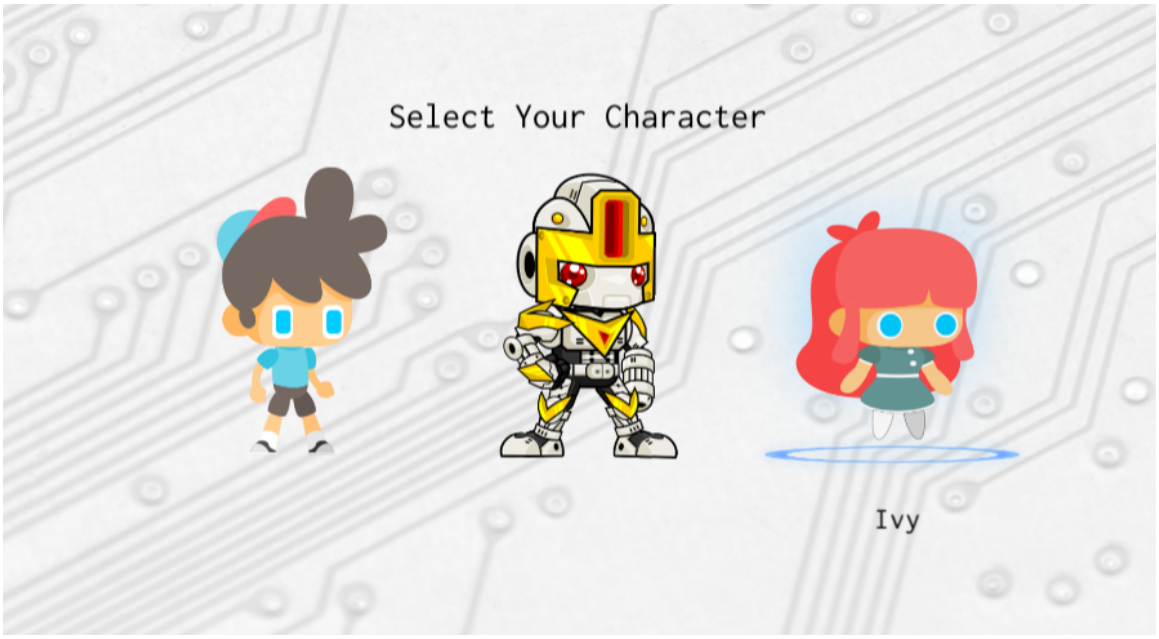


Figure 5.3: Characters in the latest version of RoboBUG. Players may choose from a boy (Guy), girl (Ivy), or robot avatar (V.I).

room's average temperature. The main function in this chapter calculates the average of a set of numbers, and players are expected to locate the bug by reading through the code and identifying the line where a bug has been placed. Some of the bugs in the levels from this chapter include incorrectly initialized variables, incorrect arithmetic calculations, and extraneous code.

Chapter 2 - Print Statements

The second chapter introduces print statements, which are used to output individual variable values when a program is running. The game's story involves a mixup with the characters' file systems, as their files have been unsorted and placed out of order. As the player progresses through the levels in this chapter, they must debug functions for swapping elements in a list, adding elements to a sorted list, and finally sorting an unsorted list. The bugs in these levels include mostly algorithmic and out of bounds errors.

Chapter 3 - Divide and Conquer

The third chapter introduces the strategy of divide and conquer, whereby a bug is located by using the process of elimination to determine which sections of the code the bug could or could not be located within. The game's story involves the characters' computers no longer displaying colors, due to numerical errors in a 'color database.' To find the bugs in this chapter, the player is given access to a 'comment' tool that can comment out a block of code, then execute the program and display whether the bug persists (i.e. was not commented out). All bugs in these levels are simply changes to numerical values, but the extensive length of the code makes these bugs extremely time consuming to locate unless a divide and conquer strategy is used.

Chapter 4 - Breakpoints

The final chapter introduces the concept of breakpoints, whereby the run-time execution is paused at the given breakpoint and the value of multiple variables can be observed. In the game, the characters are having issues with their wireless connections due to computer miscalculations in a three dimensional space. An unintuitive function for calculating distance between two points has bugs introduced that cause incorrect behavior, which can be observed using different breakpoints. The design intention of this chapter was for the bugs themselves to be very difficult to spot, but for their location to become evident when the data from the breakpoints is assessed. The bugs in this chapter include formulas with extra negations or changed operators.

5.3.2 RoboBUG Adaptations

To aid in comparison across games it would be preferable to keep the implementation of adaptivity in RoboBUGas similar as possible to the adaptation used in GidgetML.

However, the nature of RoboBUG as an exploratory game which does not include a program writing component entails some differences in how the game must be adapted in comparison to GidgetML. For example, RoboBUG does not have the same 'energy' system as Gidget, which was needed to measure the efficiency of a solution. Instead, the only metric that might indicate efficiency in discovering a bug is time, as an expert debugger would likely find bugs more quickly than a novice. Fortunately, RoboBUG still provides the opportunity for students to fail, and thus learn from their mistakes, which occurs any time a student incorrectly indicates the presence of a bug. The only way to successfully complete a level is for students to apply the relevant debugging technique in order to find the bug, and use their 'tools' to specify where they believe the bug is located. The task of figuring out how to fix the bug is not included in the game. We focus on finding and not fixing bugs because, as in real debugging cases, finding a bug takes up significantly more effort than fixing it [FLM+08].

The two variables used in the assessment of RoboBUG players were time and number of failures per level. Once a player completes a level, they are categorized in comparison to the non-adaptive RoboBUG players using a K-means classifier with 3 centroids. The result of this calculation is a set of three different groups, one of which contains the current player. The average number of failures per group is then used to sort the three groups into high, medium, and low competence ratings, which then denotes the competence of the current player. This competence rating is used to determine the degree of adaptation for the next level that the player chooses to complete.

Unlike GidgetML, two different adaptations were created based on the original RoboBUG game. The first of these adaptations focused exclusively on providing **hints** to players when they failed a level or used one of their in-game tools. Most of these hints were customized for each level, but the first hint provided when a

player failed was a general reminder of the relevant debugging technique and how it should be used. The second of these adaptations followed a similar strategy to the one used in Gidget, where the provided code was altered to make the bugs more obvious, and penalties were reduced on players who made mistakes or collided with the bug obstacles. Both variants varied the degree to which they adapt based on the level of competence of the player, where those assessed at a 'high' level of competence did not have the game altered at all, while those in the 'low' level of competence received the maximum level of adaptive changes. Competence levels for players were recalculated after each task was completed. The work needed to implement the adaptations constituted an addition of 7317 lines of code and the deletion of 1606 lines of code, resulting in a net 5711 lines of code being added, or about 5% of the total project size.

Hint Adaptation

The adaptive variant of RoboBUG with hints supports players using three different types of hints, which show up at different frequencies based on the adaptive setting. For players classified as high competence, no hints are provided. Hints for medium and low competence players vary as follows:

- When a player fails a level for the first time, they are provided a reminder of how to apply the relevant debugging technique for that level. For example, players attempting to complete levels in the divide-and-conquer chapter would be shown the message: "The bug is somewhere in this code, but it could take a long time to do code tracing. Use your tools to check different sections of code. If the bug persists when you comment something out, it must be somewhere else."

- If the player is classified as low competence and fail a second time, or classified as medium competence and fail a third time, a customized hint for that level is provided.

- If the player is classified as low competence, any failures after the second will result in players being presented with a hint that specifies exactly what the player should think about when debugging the code. A medium competence player will be shown these same hints every other time they fail a level after their third failure.

- Whenever a player of low competence uses a tool in specified locations, and every other time a player of medium competence does the same, they are shown a customized hint that elaborates on the information that the tool provides. For example, a print statement that outputs a list of numbers would only show the numbers themselves to a player of high competence, while a low competence player would see both those numbers as well as an English description of what the values represent in context. These descriptions are presented to the player by one of the characters in the game other than the player's chosen avatar, to create the impression of a pair programmer helping to debug the code.

Gameplay Adaptation

The adaptive variant of RoboBUG that does not include hints attempts to recreate some of the adaptations that were used in the study of GidgetML. Like GidgetML, the code that the players are initially provided with is altered based on the competence of the player, and like the hint variant, no adaptations occur for high competence players. The adaptations for medium and low competence players are as follows:

- The lines of code that contain the bugs were customized to be more noticeable

for medium competence players, and even more noticeable for low competence players. For example, in a level where the correct line of code is "avgT = 0.0", the high competence player would see the bug as "avgT = false", the medium competence player would see "avgT = 'hello world'", and the low competence player would see "Average = 10 billion".

- Low competence players only take half as much damage when they falsely identify a bug or collide with an obstacle, while medium competence players take 75% as much damage. This means that without acquiring in-game upgrades, medium and low competence players are permitted at least one mistake in identifying a bug while high competence players only get one attempt before failing the level.

- The moving obstacles in the game travel at half speed for low competence players, and 75% speed for medium competence players.

5.4 Methodology

The evaluation methodology for RoboBUG is comprised of two separate experimental stages, conducted in two different semesters with two different groups of first year students. The first study was specifically designed to be a non-adaptive evaluation of the RoboBUG game, while the second study used data from the first to adapt the game for each student. The studies took place in the students' regularly scheduled lab setting, and were supervised by teaching assistants without the presence of the researchers.

The methodology followed in the experimental studies is shown in Figure 5.4. In both studies, participants first completed a skill testing questionnaire of ten mul-

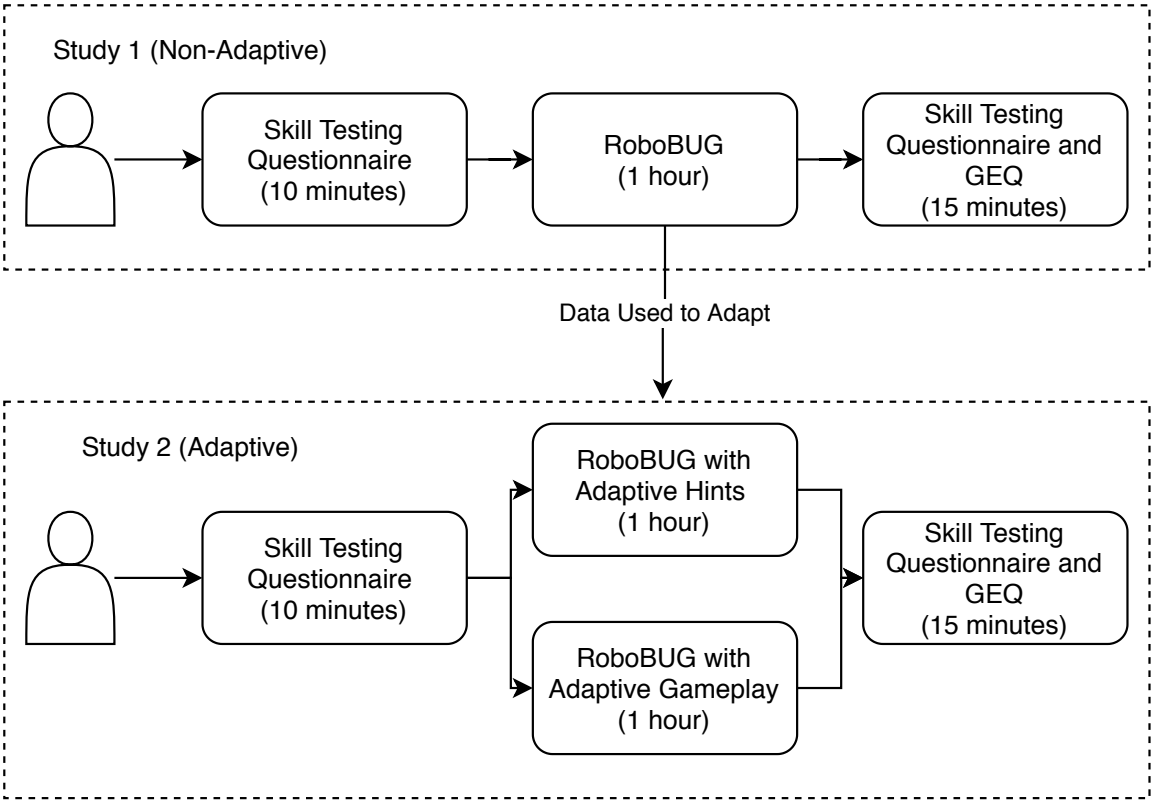


Figure 5.4: Evaluation methodology for both RoboBUG studies.
Study 1 used the Non-Adaptive version of RoboBUG, and data was collected from the questionnaires provided to participants as well as game play data logged during the 1 hour gameplay session. Study 2 randomly assigned participants to either the Adaptive Hints or Adaptive Gameplay variant of RoboBUG, using the logged data from Study 1 to assess participants.

tiple choice questions that evaluated their knowledge of debugging and debugging techniques. Students were not informed as to the results of the skill testing questionnaire. Once this questionnaire was completed, which took approximately 5-10 minutes participants were instructed to play RoboBUG for approximately one hour, before completing the exit questionnaire using the last 10 minutes of the lab time.

The exit questionnaire repeated the same skill testing questions from the initial test, in addition to questions from the GEQ [BFC+09]. The GEQ is a questionnaire designed to assess the thoughts and feelings of players immediately after gameplay on the components of Immersion, Flow, Competence, Positive & Negative Affect,

Tension, and Challenge. The full lists of questions used can be found in the Appendix.

5.5 Results

5.5.1 Overview

Out of 147 students in a first year programming course with students registered in a CS program, 107 students took part in the first experiment (non-adaptive). Of these 107 students, 68 both played the game and completed both questionnaires.

Out of 49 students in a first year programming course with non-CS students, 29 students took part in the second experiment (adaptive). Of these 29 students, 23 both played the game and completed both questionnaires. This group of 23 was divided randomly into two smaller groups: one which played the adaptive variant with hints (11 students), and one which played the adaptive variant which altered game content (12 students).

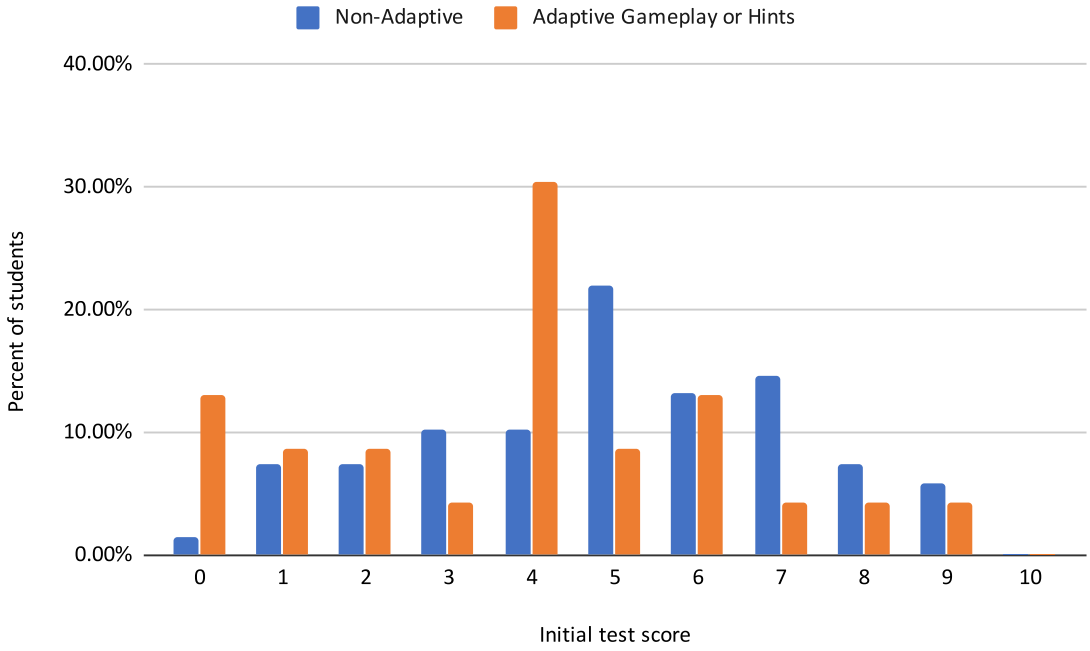


Figure 5.5: Initial Debugging Skill Test Scores

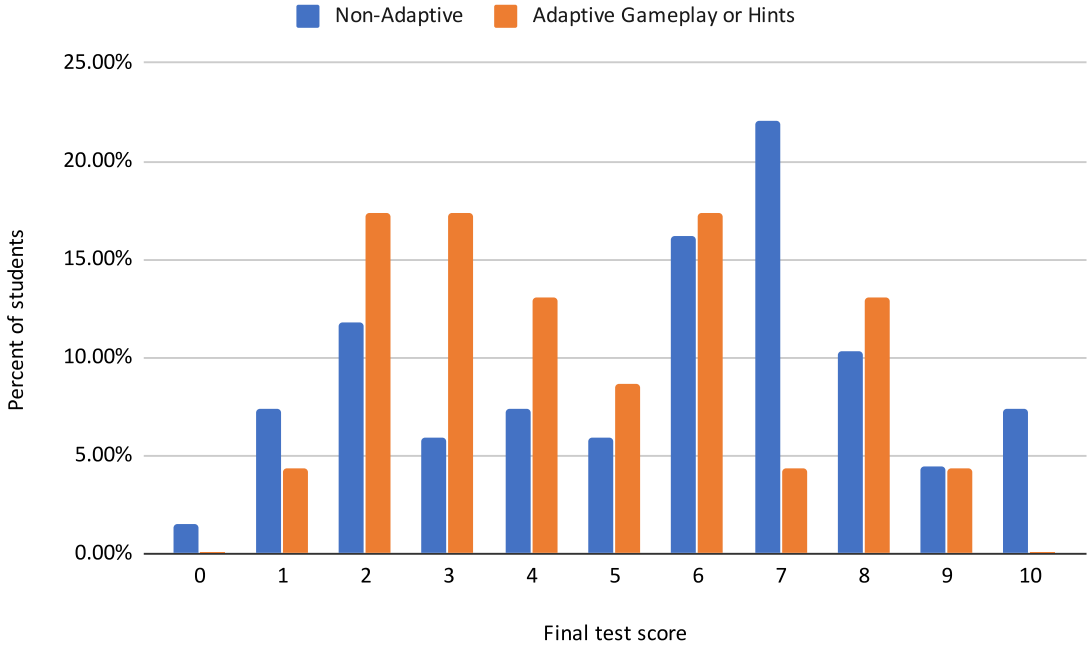


Figure 5.6: Final Debugging Skill Test Scores

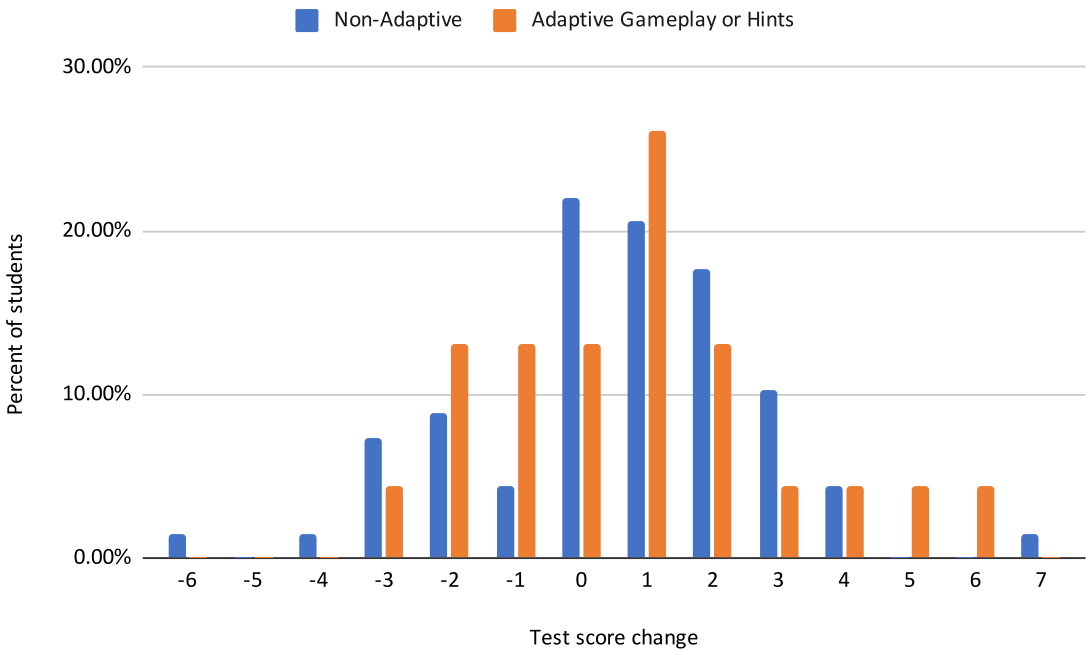


Figure 5.7: Changes in Debugging Skill Test Scores after playing RoboBUG

5.5.2 Test Scores

When comparing test scores between the non-adaptive and adaptive groups, the initial test results were significantly higher for the non-adaptive group (M=4.99, SD=2.26) than the adaptive group (M=3.87,SD=2.51), t(89)=1.9884, p=0.0498. This effect is shown in Figure 5.5. Although the mean remained higher for the non-adaptive group (M=5.54, SD=2.69) than the adaptive group (M=4.65, SD=2.33) for their final test results, this difference was not significant, t(89)=1.4193, p=0.1593. This is shown in Figure 5.6. The adaptive group had a higher improvement in test scores (M=0.78, SD=2.28) than the non-adaptive group (M=0.56, SD=2.20), as shown in Figure 5.7, but this was not significant. There was no significant difference between test scores or test score improvements between the hint group and no-hint adaptive group. The average adaptive setting, which was nearly identical for both groups, both did not seem to correlate with test scores, except in the case of the hint group, where the frequency of hints correlated with higher results on the final test (R=0.536, P=0.110).

	Adaptive	Non-Adaptive
Competence	**M=1.750, SD=1.121**	M=1.848, SD=1.172
Immersion	**M=2.00, SD=1.09**	M=1.72, SD=1.22
Flow	**M=1.261, SD=0.890**	M=1.110, SD=1.112
Challenge	**M=2.348, SD=0.970**	M=1.860, SD=1.178
Tension	M=2.152, SD=1.318	**M=2.235, SD=1.131**
Negative Affect	**M=2.239, SD=1.054**	M=2.206, SD=1.144
Positive Affect	**M=1.826, SD=1.202**	M=1.419, SD=1.128
	Adaptive Hints	Adaptive Gameplay
Competence	**M=2.045, SD=1.313**	M=1.667, SD=1.052
Immersion	**M=2.27, SD=1.10**	M=1.75, SD=1.06
Flow	**M=1.409, SD=0.917**	M=1.125, SD=0.882
Challenge	**M=2.682, SD=0.751**	M=2.042, SD=1.076
Tension	M=1.818, SD=1.347	**M=2.458, SD=1.270**
Negative Affect	M=1.955, SD=1.083	**M=2.500, SD=1.000**
Positive Affect	**M=2.045, SD=1.254**	M=1.625, SD=1.170

Figure 5.8: Mean Results from GEQ. The higher mean in each row is in bold.

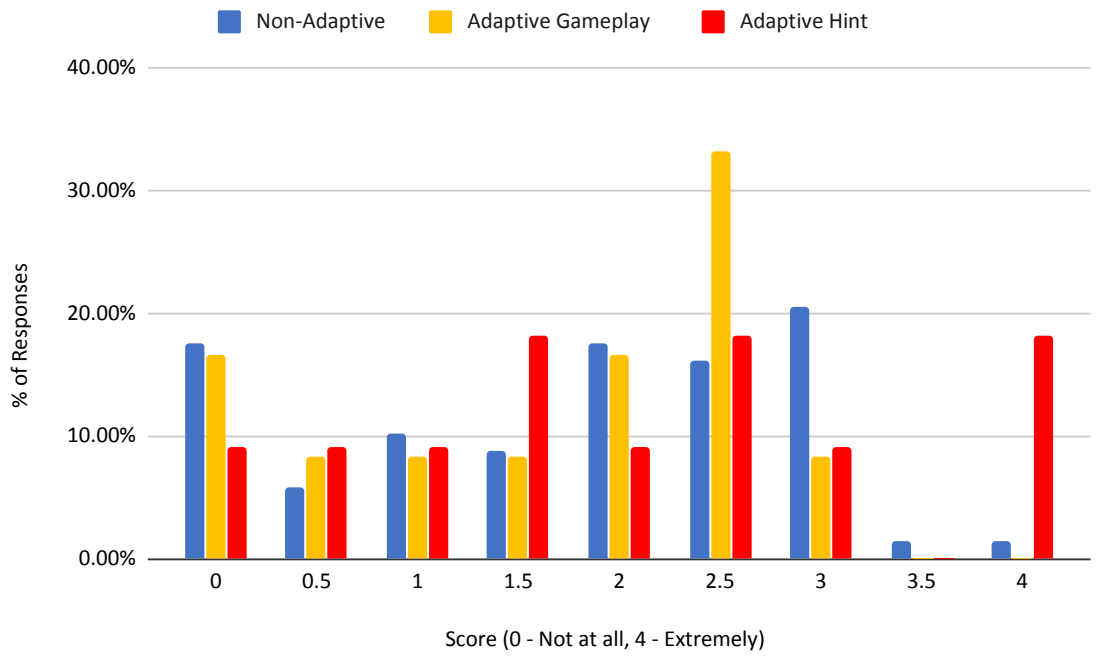


Figure 5.9: Self-Assessment of Student Competence

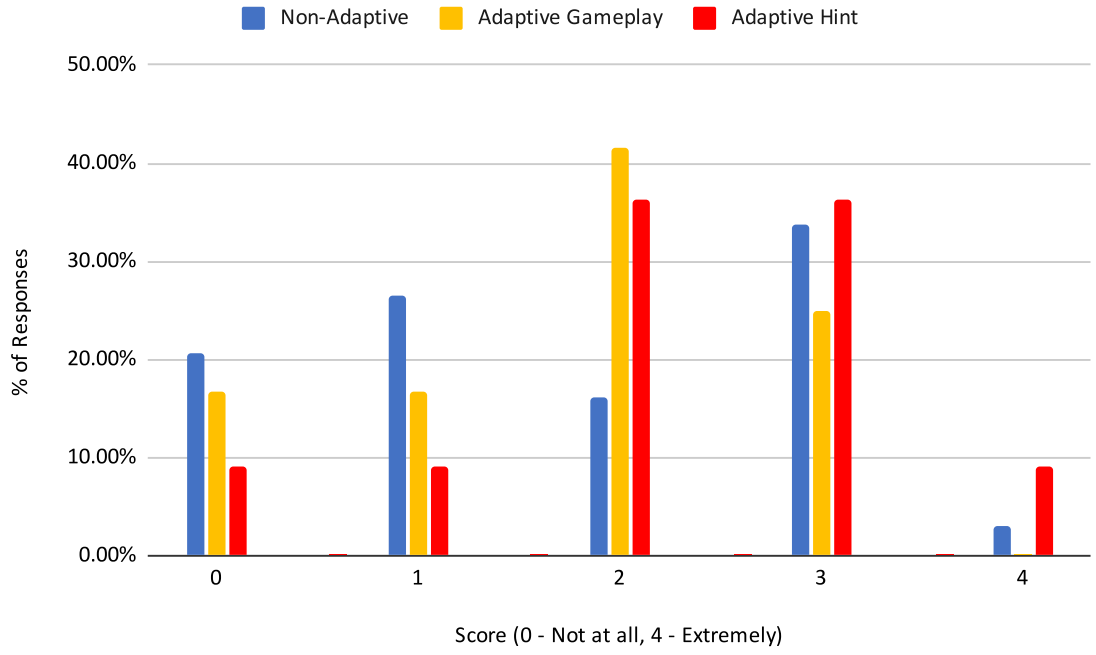


Figure 5.10: Self-Assessment of Student Immersion

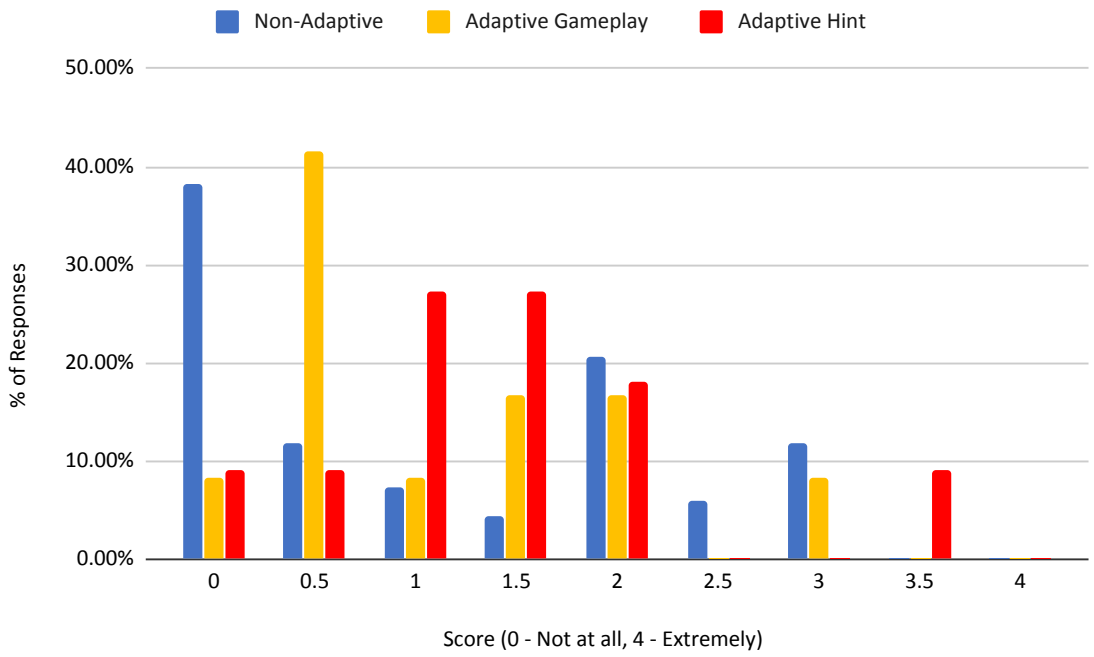


Figure 5.11: Self-Assessment of Student Flow

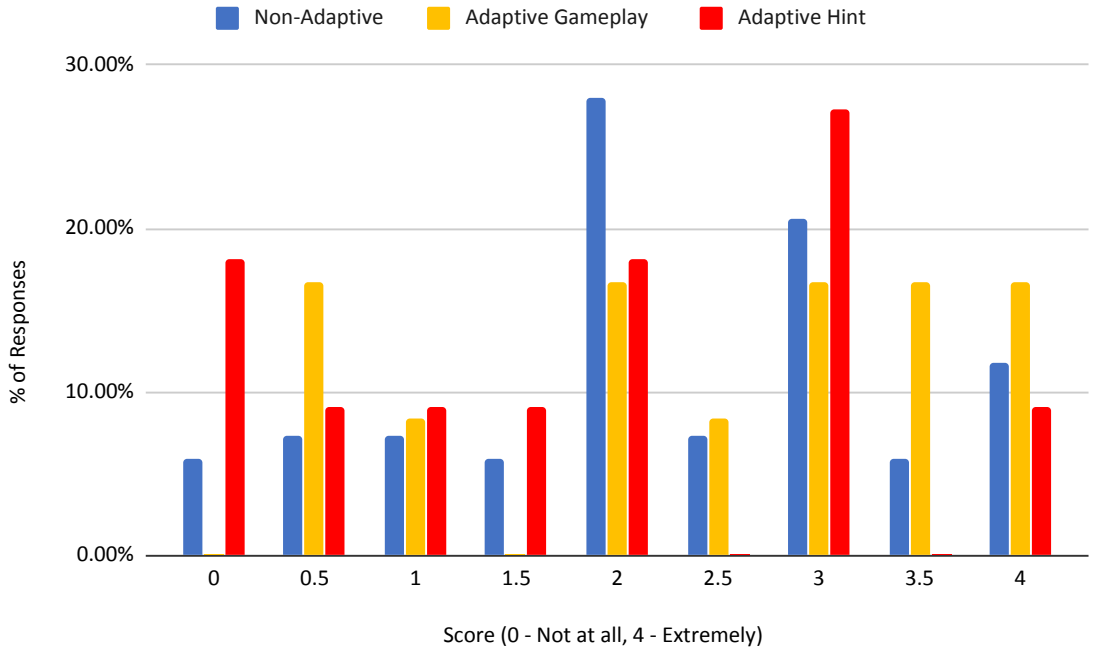


Figure 5.12: Self-Assessment of Student Tension

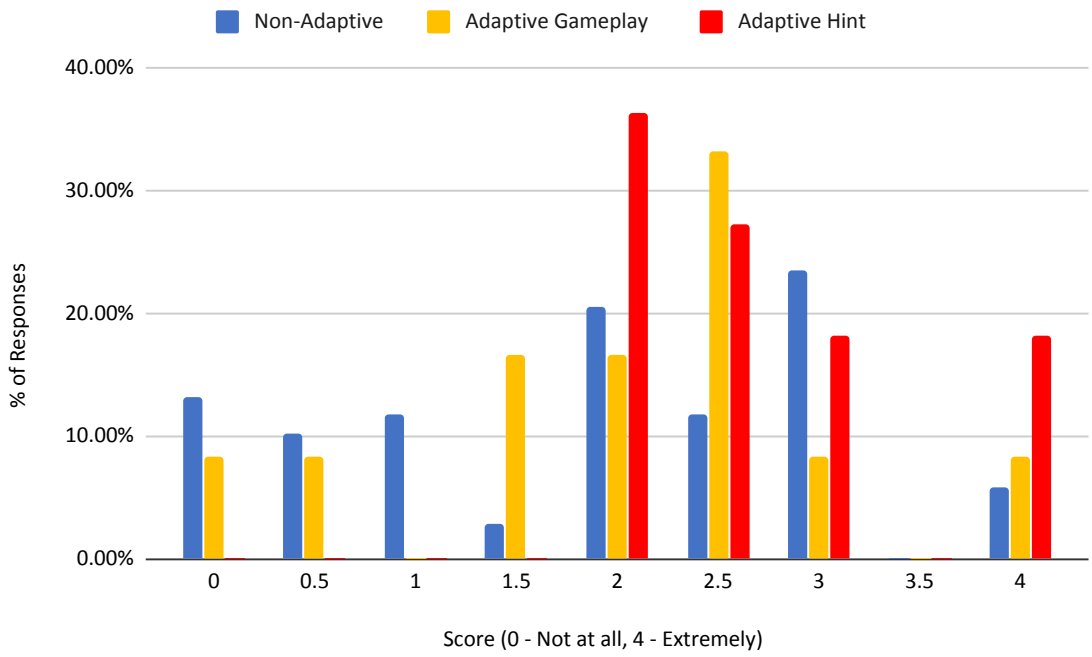


Figure 5.13: Self-Assessment of Student Challenge

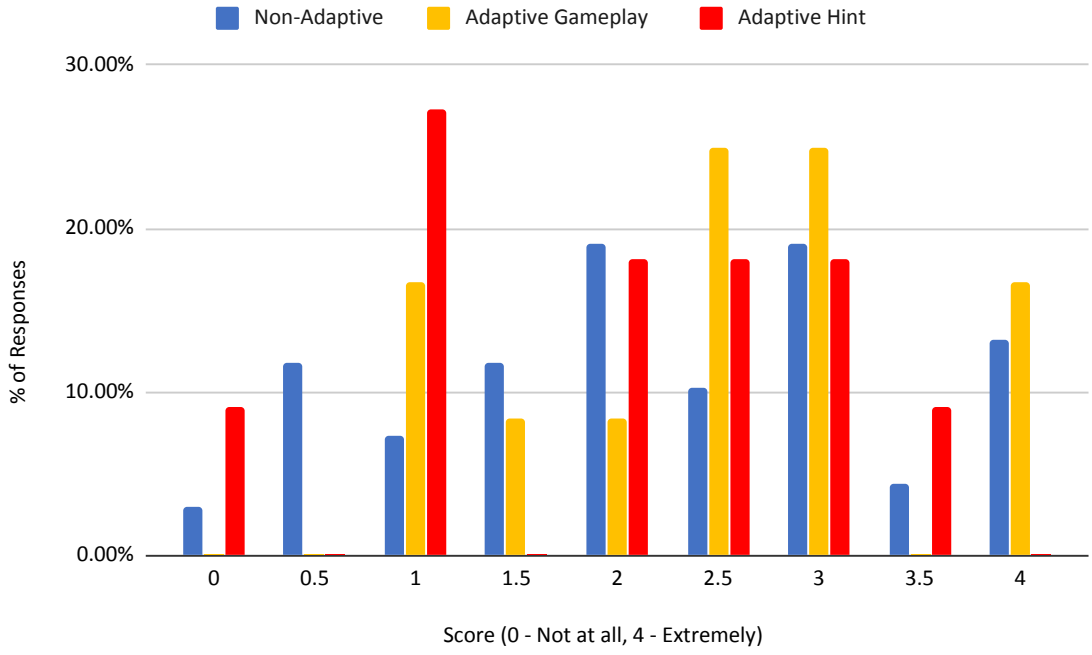


Figure 5.14: Self-Assessment of Student Negative Affect

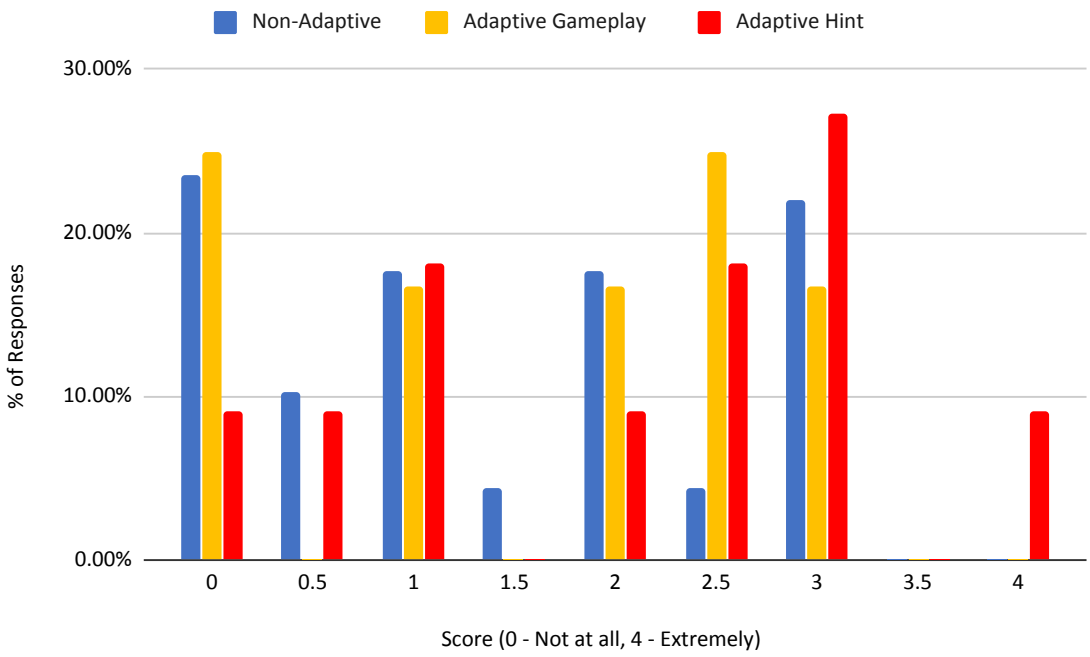


Figure 5.15: Self-Assessment of Student Positive Affect

5.5.3 Game Experience Questionnaire

A comparison of mean results from the GEQ can be found in Figure 5.8.

For challenge, the adaptive group (M=2.348, SD=0.970) had a higher mean than the non-adaptive group (M=1.860, SD=1.178), which was not quite statistically significant, t(89)=1.7887, p=0.771. This is shown in Figure 5.13. Meanwhile, the adaptive hint group experienced a statistically significantly higher level of challenge (M=2.682, SD=0.751) than the non-adaptive group (M=1.860, SD=1.178), t(77)=2.234, p=0.0284.

For positive affect, the adaptive group (M=1.826, SD=1.202) had a higher mean than the non-adaptive group (M=1.419, SD=1.128), and this more significant when specifically comparing the hint group (M=2.045, SD=1.254) to the non-adaptive group (M=1.419,SD=1.128), t(77)=1.683, p=0.0965. This is shown in Figure 5.15.

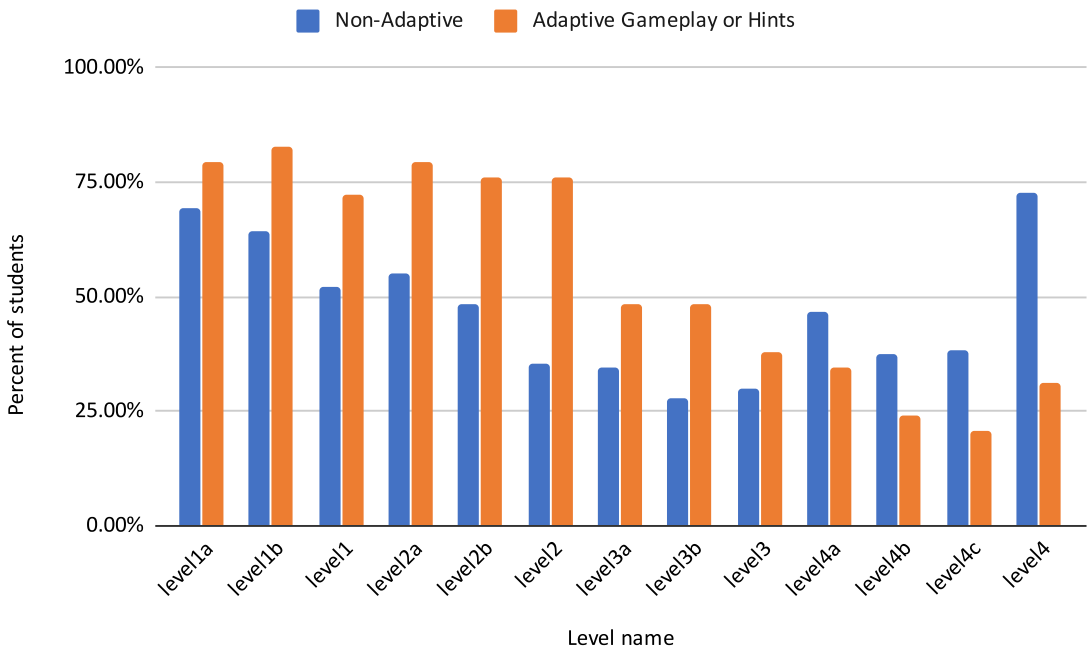


Figure 5.16: Percent of Students who completed a given level

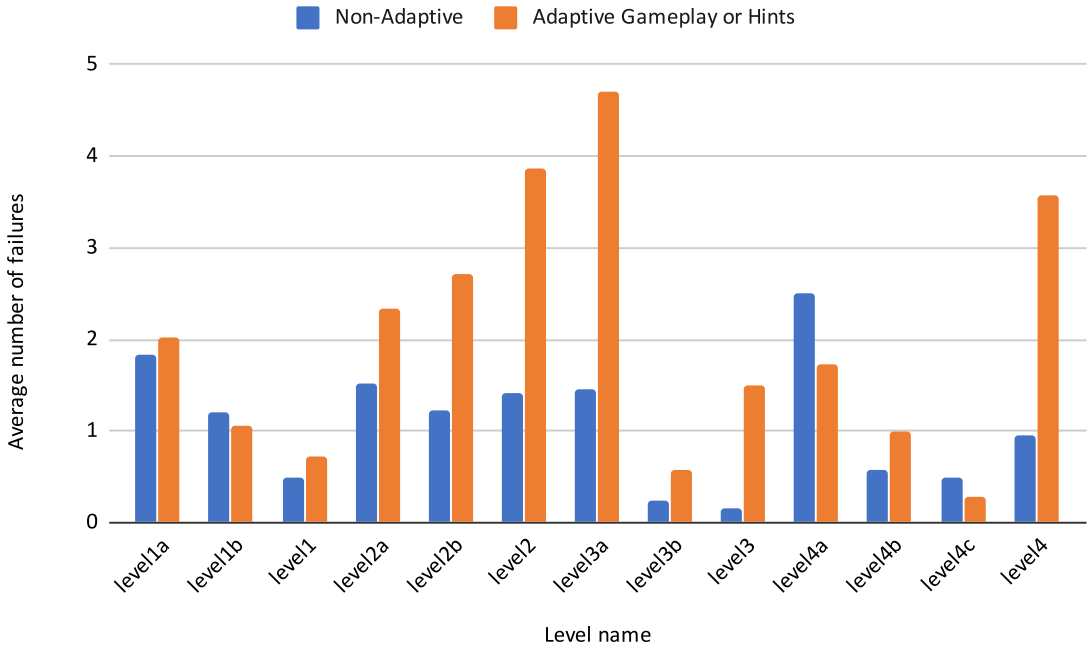


Figure 5.17: Average Failures per Level (Non-Adaptive vs. Adaptive)

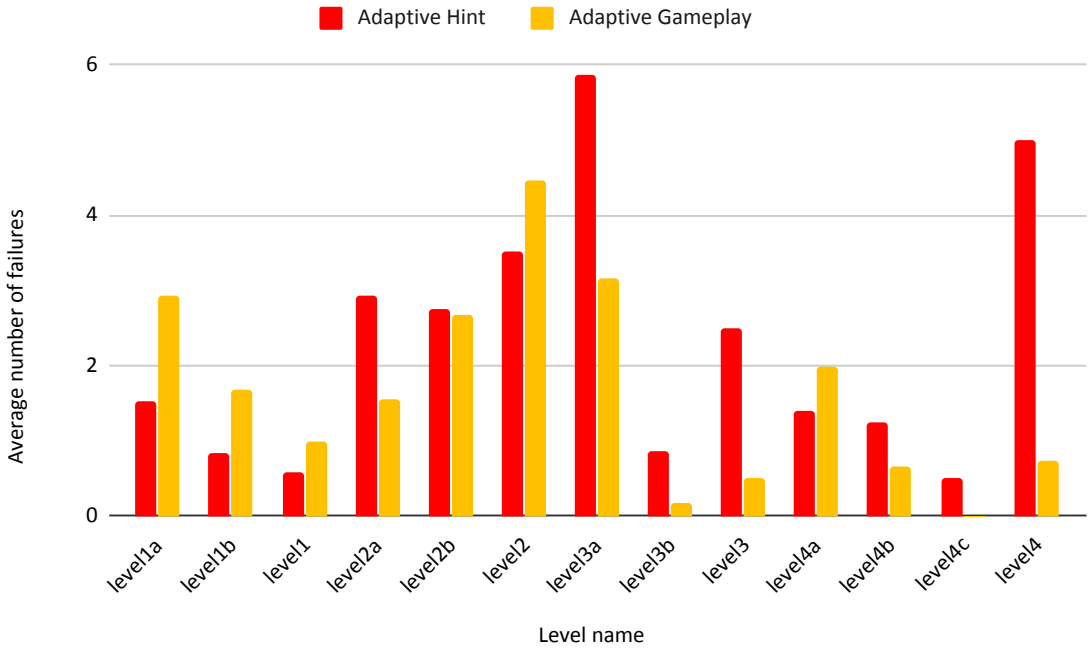


Figure 5.18: Average Failures per Level (Hints vs. No Hints)

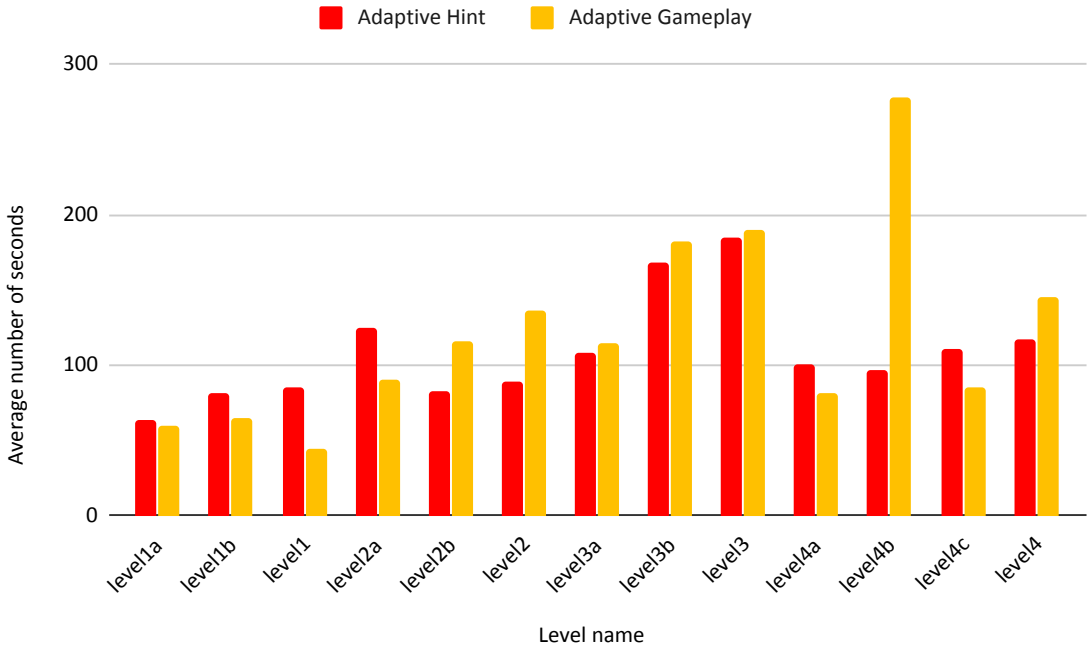


Figure 5.19: Average Time per Level (Hints vs No Hints)

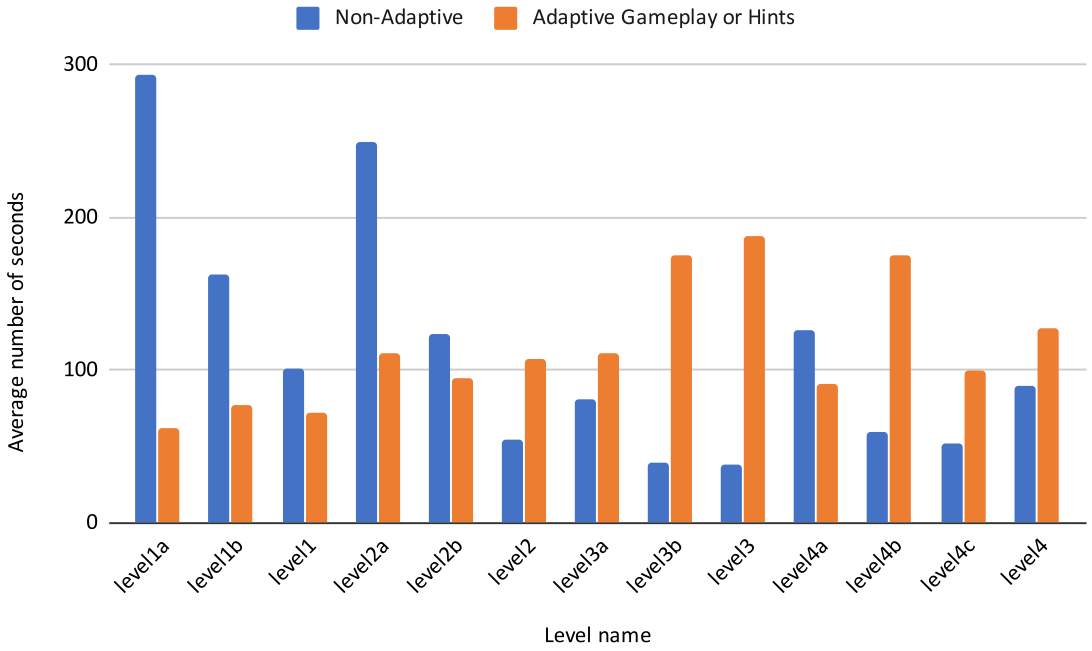


Figure 5.20: Average Time per Level (Non-Adaptive vs. Adaptive)

5.5.4 Game Play

Comparing the adaptive levels assigned to students, the average level for the hint group (M=1.690, SD=0.3708) was very similar to that of the no-hint adaptive group (M=1.682, SD=0.2995).

When observing which game levels were completed by students, as shown in Figure 5.16, the players in the adaptive group (M=0.6667, SD=0.1689) were significantly more likely to play the first three sections of the game than the non-adaptive group (M=0.4641, SD=0.1512), t(16)=2.680, p=0.0164. The non-adaptive group was more likely to skip directly to the fourth and final section of the game (M=0.4883, SD=0.1658) than the adaptive group (M=0.2759, SD=0.06295), although this result was not quite statistically significant, t(6)=2.395, p=0.0536. Recall that chapters are independent of each other, meaning that students did not need to complete the earlier chapters in order to finish the later ones.

For average failures per level, as shown in Figures 5.17 and 5.18, the adaptive group made significantly more errors (M=2.009, SD=1.378) than the non-adaptive group (M=1.083, SD=0.6841), t(24)=2.170, p=0.0401. The no-hint adaptive group had fewer failures per level (M=1.657, SD=1.336) than the hint group (M=2.274, SD=1.710), but this was not a significant result.

Comparing average time spent on each level, both the adaptive (M=114.7, SD=40.83) and non-adaptive groups (M=113.2, SD=79.70) had very similar means. However, the non-adaptive group spent a noticeably larger amount of time on early levels of the game, while the adaptive group spent more time on the later levels. This result is shown in Figures 5.19 and 5.20. For the adaptive groups, the hint group spent less time on average per level (M=109.1, SD=34.59) than the no-hint adaptive group (M=122.5, SD=64.98).

5.6 Discussion

The most significant result from the study was related to players' choices of which levels to play through in the RoboBUG game. The adaptive group, which was comprised of non-CS students who may be less familiar with debugging than their CS counterparts, were far more likely to play through the game 'normally' (i.e. from the beginning to the end) than the non-adaptive group, who often would skip directly to the final chapter of the game after playing through the game's tutorials. This suggests that some of the non-adaptive group felt confident in their abilities without needing the game's help to learn, and wanted only to complete the game as quickly as possible. Meanwhile, the non-adaptive group was less likely to make it to the final chapter of the game, opting instead to play sequentially from the beginning until they gave up on playing.

The most significant result from the test score observations was the higher initial mean for the non-adaptive group compared to the adaptive group. This could be explained by the difference in participants; the non-adaptive group was comprised of CS students who may have had more exposure to the topic of debugging compared to the non-CS students in the adaptive group. However, it is noteworthy that the adaptive group had greater improvements in their test scores after playing the game, and more importantly, the post game test scores were not statistically significantly different. This suggests that the game helped to bridge the gap in competence between the non-CS students and the CS students. Although both the hint and no-hint adaptive groups had similar test results, the presence of more frequent hints predicted better results on the final test, independently of the player's initial test score. This could be because the hints themselves included general information about debugging (such as explaining when it is appropriate to use a given technique) that were directly

relevant to the skill test.

The number of errors made per level was significantly higher for the adaptive group than the non-adaptive group, which is supported by the difference in initial test scores. It is also important to note that the majority of hints provided to those in the hint group were only made visible after the player failed a level at least once. However, the hint group did not fail significantly more often than the non-adaptive hint group, which suggests there was not a reliance by the hint group on failing levels on purpose just to see hints.

When examining the time spent on each level, it seemed that those in the non-adaptive group spent more time on the earlier levels of the game, while the adaptive group spent more time on the later levels (which the non-adaptive group often skipped to directly). One explanation for this is that those in the non-adaptive group who opted not to skip to the end of the game were the least competent and thus less likely to feel confident enough to skip to the end of the game. Another possibility is that the first levels of the game were more difficult without some form of adaptation, and thus participants found themselves struggling to understand how the game worked without any hints or obvious bugs to help them figure things out.

Unfortunately, no significant results were found from the GEQ. However, when looking at the average results, it seems that the adaptive group scored higher on competence, immersion, flow, challenge, and positive affect compared to the non-adaptive group. The hint group similarly scored higher on those categories compared to the no-hint adaptive group. For the categories of tension and negative affect, the adaptive group scored lower than the non-adaptive group, and the hint group scored lower than the no-hint adaptive group. As RoboBUG is not an action game intended to inspire feelings of tension, these results could mean that the adaptive group had a better game experience than the non-adaptive group, and the hint group had a better

game experience than the no-hint adaptive group.

5.7 Threats to Validity

Unfortunately, the adaptive portion of this study took place during the COVID-19 crisis in March of 2020. Unlike the non-adaptive portion of the study, it was not possible for any teaching assistants to be physically present to verify that all students in their labs were taking part in the experiments. This meant that the data collected was only from a portion of the class, and it is possible that the students who did not participate were also the ones who might perform significantly better or worse than their peers. The reduction of participants from this event compounded with the smaller size of the class that was selected for the study (49 potential students total for the adaptive study vs. 147 potential students for the non-adaptive study) is the most substantial threat to validity for this work.

In addition to issues of sample size, there is also the concern with regards to the competence of the students who participated in the two studies. Specifically, the students from the larger, non-adaptive study were students enrolled in CS programs, while those in the adaptive study were exclusively non-CS students taking a course in computer science. Although the games were presented to students after a similar number of weeks of the course had transpired, and some CS students had no experience with programming prior to taking the course, there was still a significant effect of CS students scoring higher on the skill testing questionnaire, and the discrepancy in competence between these two groups of students is worth consideration.

Finally, unlike GidgetML, the ability of players to choose the levels they wished to complete meant that it was not possible to compare variance of failures or time, as students did not necessarily complete the same levels as each other. It is quite likely

that students' test scores were impacted by the levels which they chose to complete. If students were forced to complete all levels of the game in sequential order, they may have scored higher on the tests, but this option was not available due to the limited amount of time students were present in the lab setting.

5.8 Summary

The goal of this work was to answer three research questions about the use of automated adaptivity in RoboBUG. The first question was:

- RQ1 - Does RoboBUG benefit from adaptation?

Although there are many variables involved in this study, one of the most significant results found was that students from the non-adaptive group, which were observed to have higher competence than those in the adaptive group, were significantly more likely to skip directly to the final chapter of the game, presumably to complete it as quickly as possible. Many of these students still completed the first few levels of the game, but there came a point where they opted to skip directly to the fourth and final chapter. This suggests that many of these students believed that RoboBUG was not sufficiently challenging (at least, at the start), and thus was not appropriate for their level of competence. The affective response of these students to the game was overall more negative than positive, and the improvement in test scores after playing RoboBUG was not significant. These observations support the idea that RoboBUG could benefit from adaptation.

The second research question was:

- RQ2 - Is the adaptive version of RoboBUG effective at adapting to a learner's level of competency?

When comparing the results between the non-adaptive and adaptive groups, the greater improvements in test scores for the adaptive groups could indicate that the adaptive RoboBUG game was more appropriately suited for them. This is also supported by the result that the players of the adaptive RoboBUG game chose to voluntarily complete the game in order rather than to skip levels to complete the game as quickly as possible. Even though the adaptive group failed levels at twice the rate as the non-adaptive group, their post test scores were still comparable. Finally, the adaptive group had higher means for the positive variables on the GEQ, as well as lower means for the negative variables. These observations suggest that RoboBUG was at least somewhat effective at adapting to a learner's level of competency.

The final research question was:

- RQ3 - Do adaptive hints better support learning and engagement for players than adapting game content?

On average, those in the hint group failed any given level approximately two times, which is the primary method through which hints are administered, thus it is certain that hints were presented to players throughout the game. With regards specifically to the GEQ, those in the hint group had the highest average scores for the positive feelings of competence, immersion, flow, challenge, and positive affect, while scoring the lowest on tension and negative affect. Players who received a larger amount of hints also tended to score higher on the post test questionnaire. These observations support the claim that the adaptive hints supported learning and engagement more than adapting game content.

Although the second study had some challenges with regards to its execution, the results of this work provide valuable insight into the role that adaptation can play in games like RoboBUG. The hint group seemed to have the most positive

game experience, however the adaptive group had results that improved over the non-adaptive group regardless of adaptation strategy. It is possible that the best adaptation for RoboBUG could be one that includes both hints as well as changes to game content, but this would require further investigation.

Chapter 6

Summary and Conclusions

6.1 Summary

This thesis presents a novel methodology to the use of adaptive techniques for serious computer science. The methodology was demonstrated using two case studies that incorporated adaptivity into Gidget and RoboBUG, which were originally designed with no options for changing player difficulty or content. After implementing data collection, machine learning assessments, and automated adaptation, evaluations of both games supported our hypothesis regarding the benefit of adaptivity in CS serious games.

In our evaluation, players of the original Gidget found that the game was either far too easy or far too difficult, and this variance was reduced significantly by the use of automated adaptation. This reduction in variance indicates that the game was able to provide students of varying levels of competence with a game play experience that was compatible with their level of skill. Although results from the GEQ were not significant, it was found that the GidgetML did no worse than Gidget version in terms of engaging students, and in fact the non-adaptive group had a (non-significantly)

more negative experience than the adaptive group. In addition, student perceptions of GidgetML were generally more preferable to Gidget, as they showed a higher tendency to enjoy the game and want to succeed.

In our evaluation, players of RoboBUG's non-adaptive version were observed frequently starting the game, then immediately jumping to the final chapter, suggesting a mismatch between the game's challenge and the players' skill. This effective was significantly reduced for those playing the adaptive version of RoboBUG, regardless of whether they were playing the adaptive hint or adaptive gameplay variant. When comparing the two adaptive variants of RoboBUG, there was a non-signficiant improvement in terms of gameplay experience for those given adaptive hints compared to adaptive gameplay. A small sample size of 23 students in the adaptive group made it difficult to generalize the results from the case study, however it is promising that the significant difference in results on the skill testing questionnaire diminished after the non-adaptive and adaptive groups played the different versions of RoboBUG.

6.2 Contributions

The main contributions of this work include:

1. The systematic review of Serious Programming Games, including educational content and methods of evaluation. This review illustrates the existing state of serious programming games and what is available for instructors or learners who are interested in game-based learning. It provides a foundational understanding of the CS serious games literature upon which this thesis is based.

2. The adaptive methodology of modifying existing CS serious games to incorporate automated adaptivity. This methodology includes how to identify suitable

games for adaptation, create models of learner behavior and task properties, implement automated adaptations, and evaluate the results of adaptations.

3. The implementation and evaluation of GidgetML, an adaptive version of Gidget developed using the aforementioned methodology. Results from the case study showed how Gidget was suitable as a candidate for adaptation, and that GidgetML helped to reduce variance in the performance of players with regards to solution efficiency and failure rates.

4. The implementation and evaluation of two adaptive variants of RoboBUG, developed using the same methodology as GidgetML. Results from the case study showed how RoboBUG was suitable as a candidate for adaptation, and that the adaptive version of RoboBUG addressed challenges with player engagement found from players of the original non-adaptive game.

The adaptive versions of both Gidget and RoboBUG have been publicly released as open source projects, to allow for their use by students, educators, and researchers[1]. The games also allow for a degree of customization that can be helpful to instructors who have an interest in including new content into the games.

6.3 Limitations

Our adaptive methodology could be used in the development of new serious programming games, but was designed specifically for improving existing ones and was not evaluated with new games. This was an explicit choice for evaluation purposes but also for adoption purposes, as existing games may already have users who will also be interested in the adaptive variance.

[1]GidgetML can be accessed at `https://github.com/sqrlab/GidgetML`. The adaptive variants of RoboBUG can be accessed at `https://github.com/sqrlab/RobotON`

The methodology can not be used with all existing CS games as not all programming games will be suitable for adaptation. It is potentially the case that games which are initially ineffective at helping players achieve learning outcomes will remain ineffective, even when adaptivity is introduced. A task-based game style is best suited to match the Competence-based Knowledge Space Theory that is the foundation of this work.

The generalizability of the adaptive methodology outside of introductory programming games has not been evaluated. More advanced topics may require tweaking the methodology, or finding an alternative approach altogether. For example, my methodology is intended for individual unassisted players; there may be better alternatives for learners in environments where instructor aid is readily available, or where collaborative work is involved.

A K-means machine learning algorithm with three centroids was used in both studies, and alternative ML techniques were not explored. K-means was selected due to the uncertainty with student categorizations, however the algorithm may struggle when presented with outlier data points. Although this did not seem to be a problem for the two case studies, future work with using the K-means algorithm for serious games should consider alternative strategies that can handle the possibility of outlying data from players.

Improvements on this research could be made in terms of the evaluation of students after gameplay. As previously mentioned in Chapter 2, there have been concerns raised about the validity and reliability of the GEQ. The results from the GEQ used in the two case studies appeared to be consistent with my hypothesis, but the data gained from gameplay data was more useful in characterizing the behavior of the participants. Small sample sizes, especially for the second case study, present additional barriers to generalizing the results of this work. In practice, implementing

adaptive serious games at a larger scale would require data collected from multiple years worth of student data, which was not feasible within the scope of this thesis. The evaluation would also benefit significantly from being able to compare game play data to student grades. This would improve confidence in the machine learning algorithm's ability to correctly assess students, rather than relying solely on game performance and making probabilistic estimations about student competences.

The RoboBUG case study was significantly impacted by the onset of the COVID-19 crisis. Originally, the study had been planned to take place during course lab sessions, but the closure of Ontario Tech University's campus led to the questionnaires and activities being distributed online to students who were not being monitored directly by their teaching assistants. This likely had an impact on the number of participants who took part in the study, as well as how long participants chose to play the game and the completion rate of all associated questionnaires. A noticeable percentage of students played the game but did not complete all questionnaires in the study, which led to a lower sample size than anticipated. Furthermore, the students involved in the adaptive RoboBUG study were registered in a computer science course, but were not registered in a CS program, and thus had a significantly different set of competences in comparison to those in the non-adaptive RoboBUG study. This effect could be seen in the difference in results on the pre-game test. While it is good to see that the difference in results between groups diminished after game play, it remains difficult to compare the two groups due to their different CS backgrounds.

Finally, the adaptive hint and adaptive gameplay variants of the RoboBUG game were not found to be significantly different in terms of students' game play experience. It is possible that an optimal adaptive approach would incorporate both of these adaptations, but this was not considered in this thesis due to requiring substantially more development effort and study participants.

6.4 Future Work

There are three main areas of future work:

1. Further evaluation of the adaptive methodology with other CS serious games,
2. Applying adaptation within tasks in addition to between tasks,
3. Considering the use of biometrics and eyetracking data to adapt serious games.

6.4.1 Extending the Approach to Other CS Serious Games

Gidget and RoboBUG are both task-based games with a focus on debugging, but vary significantly in their game play. Gidget has more similarities to other games in the serious programming game literature, as it involves players using code solutions to solve in-game problems. RoboBUG is more exploratory, relying on code comprehension rather than the ability of learners to write correct code. There are a number of other games in the literature that include different content or different forms of game play which may or may not be suitable for the adaptive approach from this thesis. Assessing the generalizability of the adaptive methodology requires examining its efficacy with these other game styles including cooperative or competitive games, which include social elements that are not found in Gidget or RoboBUG. Future work in this area could consider these types of games, or games that include more advanced programming concepts, which differ significantly from the content in Gidget and RoboBUG.

6.4.2 Modifying the Adaptation Strategy Timing

Gidget and RoboBUG were adapted using a between-task approach, where the data used for the adaptation was collected after task completion, and content was only

changed at the start of a task. However, this strategy may not be as effective as adapting content during a task, as soon as a learner is observed to be frustrated or bored with the game. It is also possible that the process for data collection could be improved by including additional data points, including course performance evaluations or data from other games and activities. The proposed approach is not designed to accommodate games without a task-based sequence of game content, which may require an alternative strategy to be used.

6.4.3 Assessment and Adaptation Variables

The primary variables used for adaptation in the case studies were failure rates and code efficiency, but these may not be the best variables that predict student competence. Biometrics or eye tracking data may provide better predictions of player performance, or may be used to supplement existing data sources [BLZ14]. It is not yet clear exactly which variables are the most accurate at learner assessment or which should be used for AI decision making.

6.5 Conclusions

The need for engaging educational tools to support online learning in CS can be addressed by the use of serious games - many of which have been identified by the review in Chapter 2. However, it is the responsibility of game developers to ensure that these games are designed to provide the best learning experience possible. This requires special consideration to be given to the wide variety of individuals who may wish to benefit from these games, as serious games have historically struggled to address the needs of learners from different backgrounds. In the pursuit of an optimal learning and gaming experience, developers must take into account the diversity of

their audiences and consider how strategies like automated adaptation can address the challenges faced by learners of all backgrounds.

Thesis Statement: *The use of an adaptive approach in serious games for computer programming can positively affect the achievement of learning outcomes and player engagement.*

To support this statement, the adaptive methodology presented in Chapter 3 can provide potential benefits to learners by adapting games based on learner performance, without obtrusively interrupting their game play experience and disrupting their sense of immersion. The application of the adaptive approach has been demonstrated in case studies on Gidget and RoboBUG, where the adaptive versions of these games made improvements on the original non-adaptive versions. These are some of the first games in the serious programming game literature that take advantage of machine learning for assessing and adapting to learners, but they will hopefully not be the last. Machine learning is a powerful tool that has not been fully utilized in the context of serious programming games, and there are still many opportunities for the power of adaptation to be leveraged to develop better serious games.

Bibliography

[Abt70] Clark C Abt. Serious games: The art and science of games that simulate life. *Viking Compass Book, USA*, 1970.

[Abt87] Clark C Abt. *Serious Games*. University Press of America, 1987.

[ACKP95] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4(2):167–207, 1995.

[AEH05] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. An analysis of patterns of debugging among novice computer science students. In *Proceedings of the 10th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 84–88, 2005.

[AL99] Dietrich Albert and Josef Lukas. *Knowledge Spaces: Theories, Empirical Research, and Applications*. Psychology Press, 1999.

[AVCW12] Nicoletta Adamo-Villani, Steve Cooper, and David Whittinghill. Building a serious game for teaching secure coding in introductory programming courses. In *Proc. of Eurographics 2012 - Educators*. May, 2012.

[Ban91] Albert Bandura. Self-regulation of motivation through anticipatory and self-regulatory mechanisms. In *Perspectives on Motivation: Nebraska Symposium on Motivation*, volume 38, pages 69–164, 1991.

[BBCC+06] Nadia Bianchi-Berthouze, Paul Cairns, Anna Cox, Charlene Jennett, and Whan Woong Kim. On posture as a modality for expressing and recognizing emotions. In *Emotion and HCI workshop at BCS HCI London*, 2006.

[BBDGP09] Francesco Bellotti, Riccardo Berta, Alessandro De Gloria, and Ludovica Primavera. Adaptive experience engine for serious games. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):264–280, 2009.

[BBY08] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. In *ACM SIGCSE Bulletin*, volume 40, pages 402–406. ACM, 2008.

[BDJM06] John M Bridgeland, John J DiIulio Jr, and Karen Burke Morison. The silent epidemic: Perspectives of high school dropouts. *Civic Enterprises*, 2006.

[BFC+09] Jeanne H Brockmyer, Christine M Fox, Kathleen A Curtiss, Evan McBroom, Kimberly M Burkhart, and Jacquelyn N Pidruzny. The development of the game engagement questionnaire: A measure of engagement in video game-playing. *Journal of Experimental Social Psychology*, 45(4):624–634, 2009.

[BFL99] Albert Bandura, WH Freeman, and Richard Lightsey. Self-efficacy: The exercise of control, 1999.

[BHX+15] Judith Bishop, R Nigel Horspool, Tao Xie, Nikolai Tillmann, and Jonathan de Halleux. Code hunt: Experience with coding contests at scale. In *Proc. of the 37th International Conference on Software Engineering-Volume 2*, pages 398–407. IEEE Press, 2015.

[BKBH07] Robin Bloor, Marcia Kaufman, Carol Baroudi, and Judith Hurwitz. *Service Oriented Architecture for Dummies®*. John Wiley & Sons, 2007.

[BLL+07] Chris Berka, Daniel J Levendowski, Michelle N Lumicao, Alan Yau, Gene Davis, Vladimir T Zivkovic, Richard E Olmstead, Patrice D Tremoulet, and Patrick L Craven. Eeg correlates of task engagement and mental workload in vigilance, learning, and memory tasks. *Aviation, Space, and Environmental Medicine*, 78(5):B231–B244, 2007.

[Blo84] Benjamin S Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.

[BLZ14] JH Byun, CS Loh, and T Zhou. Assessing play-learners' performance in serious game environments by using in situ data: Using eye tracking for serious game analytics. In *Annual Conference of the Association for Educational Communications and Technology (AECT), Jacksonville, FL*, 2014.

[BoLS] U.S. Department of Labor Bureau of Labor Statistics. Occupational outlook handbook, 2016-2017 edition. `https:`

//www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm. (accessed: March 27, 2017).

[BPCL08] Tiffany Barnes, Eve Powell, Amanda Chaffin, and Heather Lipford. Game2Learn : Improving the motivation of CS1 students. *GDCSE '08 Proceedings of the 3rd International Conference on Game Development in Computer Science Education*, pages 1–5, 2008.

[BS81] A Bandura and M Shunk. Cultivating confidence, self-efficacy and interest through proximal motivation. *Journal of Personality and Social Psychology*, 35:125–139, 1981.

[BS15] Florian Brühlmann and Gian-Marco Schmid. How to measure the game experience? analysis of the factor structure of two questionnaires. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, pages 1181–1186, New York, NY, USA, 2015. Association for Computing Machinery.

[BSMK16] Matthew Butler, Jane Sinclair, Michael Morgan, and Sara Kalvala. Comparing international indicators of student engagement for computer science. In *Proceedings of the Australasian Computer Science Week Multiconference*, page 6. ACM, 2016.

[BW04] Esmail Bonakdarian and Laurie White. Robocode throughout the curriculum. *Journal of Computing Sciences in Colleges*, 19(3):311–313, 2004.

[Car06] Lori Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. *ACM SIGCSE Bulletin*, 38(1):27–31, 2006.

[CBB+14] Maira B Carvalho, Francesco Bellotti, Riccardo Berta, Francesco Curatelli, Alessandro De Gloria, Giorgia Gazzarata, Jun Hu, Michael Kickmeier-Rust, and Chiara Martinengo. The journey: a service-based adaptive serious game on probability. In *International Conference on Games and Learning Alliance*, pages 97–106. Springer, 2014.

[CBD16] Sébastien Combéfis, Gytautas Beresnevičius, and Valentina Dagiene. Learning Programming through Games and Contests: Overview, Characterisation and Discussion. *Olympiads in Informatics*, 10(1):39–60, 2016.

[CBM+12] Thomas M Connolly, Elizabeth A Boyle, Ewan MacArthur, Thomas Hainey, and James M Boyle. A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, 59(2):661–686, 2012.

[CDHB09] Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes. Experimental evaluation of teaching recursion in a video game. *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, 1(212):79–86, 2009.

[CKK06] Robert M Carini, George D Kuh, and Stephen P Klein. Student engagement and student learning: Testing the linkages. *Research in Higher Education*, 47(1):1–32, 2006.

[CL04] Ryan Chmiel and Michael C Loui. Debugging: from novice to expert. *ACM SIGCSE Bulletin*, 36(1):17–21, 2004.

[Cla84] William J Clancey. Methodology for building an intelligent tutoring system. *Methods and Tactics in Cognitive Science*, pages 51–84, 1984.

[Col82] JL Collins. Self-efficacy and ability in achievement behavior, paper presented at the annual meeting of the american educational research association. *New York*, 1982.

[Csi75] Mihaly Csikszentmihalyi. Beyond boredom and anxiety. san francisco. *CA, US: Jossey-Bass*, 1975.

[Csi96] Mihaly Csikszentmihalyi. Flow and the psychology of discovery and invention. *New York: Harper Collins*, 1996.

[Csi97] Mihaly Csikszentmihalyi. *Finding Flow: The Psychology of Engagement with Everyday Life.* Basic Books, 1997.

[CTT10] Malcolm Corney, Donna Teague, and Richard N Thomas. Engaging students in programming. In *Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103*, pages 63–72. Australian Computer Society, Inc., 2010.

[DAJ11] Damien Djaouti, Julian Alvarez, and Jean-Pierre Jessel. Classifying serious games: the g/p/s model. In *Handbook of research on improving learning and motivation through educational games: Multidisciplinary approaches*, pages 118–136. IGI Global, 2011.

[DB12] KE DiCerbo and JT Behrens. Implications of the digital ocean on current and future assessment. *Computers and their Impact on State Assessment: Recent History and Predictions for the Future*, pages 273–306, 2012.

[DBTMGFM10] Ángel Del Blanco, Javier Torrente, Pablo Moreno-Ger, and Baltasar Fernández-Manjón. Integrating adaptive games in student-centered virtual learning environments. *International Journal of Distance Education Technologies (IJDET)*, 8(3):1–15, 2010.

[DDKN11] Sebastian Deterding, Dan Dixon, R Khaled, and L Nacke. From game design elements to gamefulness: defining gamification. In *Proc. of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 9–15, 2011.

[Ded05] Chris Dede. Planning for neomillennial learning styles. *Educause Quarterly*, 28(1):7–12, 2005.

[DF85] Jean-Paul Doignon and Jean-Claude Falmagne. Spaces for the assessment of knowledge. *International Journal of Man-machine Studies*, 23(2):175–196, 1985.

[DF15] Sebastian Dziallas and Sally Fincher. Acm curriculum reports: A pedagogic perspective. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, pages 81–89, 2015.

[Don07] Mary Jo Dondlinger. Educational video game design: A review of the literature. *Journal of Applied Educational Technology*, 4(1):21–31, 2007.

[DW09] Heather Desurvire and Charlotte Wiberg. Game usability heuristics (play) for evaluating and designing better games: The next iteration. In *Int. Conf. on Online Communities and Social Computing*, pages 557–566. Springer, 2009.

[FG04] Andrea Forte and Mark Guzdial. Computers for communication, not calculation: Media as a motivation and context for learning. In *Proc. of the 37th Annual Hawaii International Conference on System Sciences, 2004.*, pages 10–pp. IEEE, 2004.

[FG05] Andrea Forte and Mark Guzdial. Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, 48(2):248–253, 2005.

[FLM+08] Sue Fitzgerald, Gary Lewandowski, Renee McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18(2):93–116, 2008.

[For13] ACM Joint Task Force. Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Technical report, Association for Computing Machinery (ACM) IEEE Computer Society, 2013.

[GBW13] Lindsey Ann Gouws, Karen Bradshaw, and Peter Wentworth. Computational thinking in educational activities: an evaluation of the educational game light-bot. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, pages 10–15, 2013.

[GCB15] Benjamin Goldberg and Janis Cannon-Bowers. Feedback source modality effects on training outcomes in a serious game: Pedagogical agents make a difference. *Computers in Human Behavior*, 52:1–11, 2015.

[GD04] Kiel M Gilleade and Alan Dix. Using frustration in the design of adaptive videogames. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 228–232, 2004.

[Gee03] James Paul Gee. What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)*, 1(1):20–20, 2003.

[GHL96] James Gee, Glynda Hull, and Colin Lankshear. The new work order: behind the language of the new capitalism. boulder, 1996.

[GKH07] Frank L Greitzer, Olga Anna Kuchar, and Kristy Huston. Cognitive science implications for enhancing training effectiveness in a serious gaming context. *Journal on Educational Resources in Computing (JERIC)*, 7(3):2, 2007.

[GMRS09] Stefan Göbel, Florian Mehm, Sabrina Radke, and Ralf Steinmetz. 80days: Adaptive digital storytelling for digital educational games. In *Proceedings of the Second International Workshop on Story-Telling and Educational Games (STEG'09)*, volume 498, 2009.

[GO86] Leo Gugerty and Gary M Olson. Comprehension differences in debugging by skilled and novice programmers. In *Proc. of the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, pages 13–27, 1986.

[Gon99] Andrew Gonczi. Competency-based learning. *Understanding Learning at Work*, pages 180–195, 1999.

[Har04] Ken Hartness. Robocode: using games to teach artificial intelligence. *Journal of Computing Sciences in Colleges*, 19(4):287–291, 2004.

[HARVE98] A Hirumi, R Appelman, L Rieber, and R Van Eck. Four perspectives on preparing instructional designers to optimize game-based learning. *Tech Trends, AECT*, 1998.

[HBWM+18] Maurice Hendrix, Tyrone Bellamy-Wood, Sam McKay, Victoria Bloom, and Ian Dunwell. Implementing adaptive game difficulty balancing in serious games. *IEEE Transactions on Games*, 2018.

[HJ09] Johan Hagelback and Stefan J Johansson. Measuring player experience on runtime dynamic difficulty scaling in an rts game. In *Proc. of the 2009 IEEE Symposium on Computational Intelligence and Games*, pages 46–52. IEEE, 2009.

[HLL+08] Susan Haller, Brian Ladd, Scott Leutenegger, John Nordlinger, Jody Paul, Henry Walker, and Carol Zander. Games: good/evil. *ACM SIGCSE Bulletin*, 40(1):219–220, 2008.

[HPB14] Andrew Hicks, Barry Peddycord, and Tiffany Barnes. Building games to learn from their players: Generating hints in a serious game. In *International Conference on Intelligent Tutoring Systems*, pages 312–317. Springer, 2014.

[HSDC07] Cindy E Hmelo-Silver, Ravit Golan Duncan, and Clark A Chinn. Scaffolding and achievement in problem-based and inquiry learn-

ing: a response to kirschner, sweller, and. *Educational Psychologist*, 42(2):99–107, 2007.

[HSHA06] Jürgen Heller, Christina Steiner, Cord Hockemeyer, and Dietrich Albert. Competence-based knowledge structures for personalised learning. *International Journal on E-learning*, 5(1):75–88, 2006.

[HT07] John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007.

[Hun05] Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 429–433. ACM, 2005.

[IdKP13] Wijnand A IJsselsteijn, Yvonne AW de Kort, and Karolien Poels. The game experience questionnaire. *Eindhoven: Technische Universiteit Eindhoven*, pages 3–9, 2013.

[IHK+12] Damir Ismailović, Juan Haladjian, Barbara Köhler, Dennis Pagano, and Bernd Brügge. Adaptive serious game development. In *Games and Software Engineering (GAS), 2012 2nd International Workshop on*, pages 23–26. IEEE, 2012.

[II] Information and Communications Technology Council (ICTC). Digital talent: Road to 2020 and beyond, a national strategy to develop canada's talent in a global digital economy. `http://www.ictc-ctic.ca/wp-content/uploads/2016/03/ICTC_DigitalTalent2020_ENGLISH_FINAL_March2016.pdf`. (accessed: March 27, 2017).

[IKH+12] Damir Ismailović, Barbara Köhler, Juan Haladjian, Dennis Pagano, and Bernd Brügge. Towards a conceptual model for adaptivity in serious games. In *IADIS International Conference-Game and Entertainment, 2012*, 2012.

[IPB11] Damir Ismailović, Dennis Pagano, and Bernd Brügge. Wemakewords-an adaptive and collaborative serious game for literacy acquisition. In *IADIS International Conference-Game and Entertainment*, 2011.

[IVCFGD+13] Ana Illanas Vila, José Ramón Calvo-Ferrer, Francisco J Gallego-Durán, Faraón Llorens Largo, et al. Predicting student performance in foreign languages with a serious game. 2013.

[JGP18] Daniel Johnson, M John Gardner, and Ryan Perry. Validation of two game experience scales: the player experience of need satisfaction (pens) and game experience questionnaire (geq). *International Journal of Human-Computer Studies*, 118:38–46, 2018.

[JNW15] Daniel Johnson, Lennart E Nacke, and Peta Wyeth. All about that base: differing player experiences in video game genres and the unique case of moba games. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2265–2274, 2015.

[JVM05] W Lewis Johnson, Hannes Högni Vilhjálmsson, and Stacy Marsella. Serious games for language learning: How much game, how much AI? In *Artificial Intelligence in Education*, volume 125, pages 306–313, 2005.

[JZ16] Chaima; Jemmali and Yang; Zijian. A serious game to teach middle and high school girls programming. (April), 2016.

[Kay01] Judy Kay. Learner control. *User modeling and user-adapted interaction*, 11(1-2):111–127, 2001.

[Ke09] Fengfeng Ke. A qualitative meta-analysis of computer games as learning tools. *Handbook of Research on Effective Electronic Gaming in Education*, pages 1–32, 2009.

[KKBM12] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences*, 47:1991–1999, 2012.

[KM91] Thomas Kellaghan and George F Madaus. National testing: Lessons for america from europe. *Educational Leadership*, 49(3):87–93, 1991.

[KRA10] Michael D Kickmeier-Rust and Dietrich Albert. Micro-adaptivity: Protecting immersion in didactically adaptive digital educational games. *Journal of Computer Assisted Learning*, 26(2):95–105, 2010.

[KRHAA08] Michael D Kickmeier-Rust, Cord Hockemeyer, Dietrich Albert, and Thomas Augustin. Micro adaptive, non-invasive knowledge assessment in educational games. In *2008 Second IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, pages 135–137. IEEE, 2008.

[KRMSA11] Michael D Kickmeier-Rust, Elke Mattheiss, Christina M Steiner, and Dietrich Albert. A psycho-pedagogical framework for multi-adaptive educational games. *IGI Global*, 1:45–58, 2011.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[KTHR15] Reinier Kop, Armon Toubman, Mark Hoogendoorn, and Jan Joris Roessingh. Evolutionary dynamic scripting: Adaptation of expert rule bases for serious games. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 53–62. Springer, 2015.

[Lan04] Peter E Langford. *Vygotsky's Developmental and Educational Psychology*. Psychology Press, 2004.

[Law04] R. Lawrence. Teaching Data Structures Using Competitive Games. *IEEE Transactions on Education*, 47(4):459–466, 2004.

[LBK+14] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. Principles of a debugging-first puzzle game for computing education. In *Proceedings of the 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 57–64. IEEE, 2014.

[LBM18] Effie L-C Law, Florian Brühlmann, and Elisa D Mekler. Systematic review and validation of the game experience questionnaire (geq)-implications for citation and reporting practice. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pages 257–270, 2018.

[LDSS04] Dan Li, Jitender Deogun, William Spaulding, and Bill Shuart. Towards missing data imputation: a study of fuzzy k-means clustering method. In *Proceedings of the International Conference on Rough Sets and Current Trends in Computing*, pages 573–579. Springer, 2004.

[Lee19] Felix Lee. *Exploring How Novice Programmers Pick Debugging Tactics When Debugging: A Student's Perspective*. PhD thesis, UC Irvine, 2019.

[LK11] Michael J Lee and Amy J Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the Seventh International Workshop on Computing Education Research*, pages 109–116. ACM, 2011.

[LK12] Michael J. Lee and Andrew J. Ko. Investigating the role of purposeful goals on novices' engagement in a programming game. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pages 163–166, 2012.

[LK15] Michael J. Lee and Andrew J. Ko. Comparing the effectiveness of online learning approaches on cs1 learning outcomes. In *Proc. of the 11th Int. Conf. on Computing Education Research (ICER'15)*, pages 237–246, 2015.

[LKK13] Michael J. Lee, Amy J. Ko, and Irwin Kwan. In-game assessments increase novice programmers' engagement and level completion speed. In *Proceedings of the Ninth Annual International ACM*

Conference on International Computing Education Research, ICER '13, pages 153–160, New York, NY, USA, 2013. ACM.

[LKM+08] Amanda Lenhart, Joseph Kahne, Ellen Middaugh, Alexandra Rankin Macgill, Chris Evans, and Jessica Vitak. Teens, video games, and civics: Teens' gaming experiences are diverse and include significant social interaction and civic engagement. *Pew Internet & American Life Project*, 2008.

[LLY10] Kris MY Law, Victor CS Lee, and Yuen-Tak Yu. Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, 55(1):218–228, 2010.

[Lon07] Ju Long. Just For Fun: Using Programming Games in Software Programming Training and Education-A Field Study of IBM Robocode Community. *Journal of Information Technology Education*, 6:279–290, 2007.

[M+67] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[MA12] Konstantin Mitgutsch and Narda Alvarado. Purposeful by design?: a serious game design assessment framework. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 121–128. ACM, 2012.

[MAD+01] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda

Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *ACM SIGCSE Bulletin*, 33(4):125–180, 2001.

[Man06] Linda Mannila. Progress reports and novices' understanding of program code. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, pages 27–31. ACM, 2006.

[MB17] Michael A. Miljanovic and Jeremy S. Bradbury. RoboBUG: A serious game for learning debugging techniques. In *Proc. of the 2017 ACM Conference on International Computing Education Research (ICER '17)*, pages 93–100, 2017.

[MB18a] Michael A Miljanovic and Jeremy S Bradbury. Making serious programming games adaptive. In *Proceedings of the Joint International Conference on Serious Games*, pages 253–259. Springer, 2018.

[MB18b] Michael A Miljanovic and Jeremy S Bradbury. A review of serious games for programming. In *Proceedings of the Joint International Conference on Serious Games*, pages 204–216. Springer, 2018.

[MB20] Michael A Miljanovic and Jeremy S Bradbury. GidgetML: an adaptive serious game for enhancing first year programming labs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, pages 184–192, 2020.

[MBTO14] Elisa D Mekler, Julia Ayumi Bopp, Alexandre N Tuch, and Klaus Opwis. A systematic review of quantitative studies on the enjoyment

of digital entertainment games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 927–936, 2014.

[MC05] Micheline Manske and Cristina Conati. Modelling learning in an educational game. In *AIED*, volume 2005, pages 411–418, 2005.

[MCPS16] Anuradha Mathrani, Shelly Christian, and Agate Ponder-Sutton. PlayIT: Game Based Learning Approach for Teaching Programming Concepts. *Educational Technology & Society*, 19(5):5–17, 2016.

[Mil15] Michael A Miljanovic. Robobug: a game-based approach to learning debugging techniques. Master's thesis, 2015.

[MML+11] Henry Markram, Karlheinz Meier, Thomas Lippert, Sten Grillner, Richard Frackowiak, Stanislas Dehaene, Alois Knoll, Haim Sompolinsky, Kris Verstreken, Javier DeFelipe, et al. Introducing the human brain project. *Procedia Computer Science*, 7:39–42, 2011.

[MO99] George F Madaus and Laura M O'Dwyer. A short history of performance assessment: Lessons learned. *Phi Delta Kappan*, 80(9):688, 1999.

[MP09] Briana B Morrison and Jon A Preston. Engagement: gaming throughout the curriculum. *ACM SIGCSE Bulletin*, 41(1):342–346, 2009.

[MSO12] Tamotsu Mitamura, Y Suzuki, and T Oohori. Serious games for learning programming languages. In *Proc. of Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference*, pages 1812–1817, 2012.

[NDG10] LE Nacke, Anders Drachen, and Stefan Göbel. Methods for evaluating gameplay experience in a serious gaming context. *International Journal of Computer Science in Sport*, 9(2):1–12, 2010.

[Nor13] Kent L Norman. Geq (game engagement/experience questionnaire): a review of two papers. *Interacting with Computers*, 25(4):278–283, 2013.

[NR07] Erik E Noftle and Richard W Robins. Personality predictors of academic outcomes: big five correlates of gpa and sat scores. *Journal of Personality and Social Psychology*, 93(1):116, 2007.

[NW87] Richard Niemiec and Herbert J Walberg. Comparative effects of computer-assisted instruction: A synthesis of reviews. *Journal of Educational Computing Research*, 3(1):19–37, 1987.

[OG06] Jackie O'Kelly and J. Paul Gibson. RoboCode & Problem-Based Learning : A non-prescriptive approach to teaching programming. In *Innovation and Technology in Computer Science Education (ITiCSE '06)*, number June, pages 26–28, 2006.

[OP07] Melissa C O'Connor and Sampo V Paunonen. Big five personality predictors of post-secondary academic performance. *Personality and Individual differences*, 43(5):971–990, 2007.

[PAM13] Ioannis Paliokas, Chistos Arapidis, and Michail Mpimpitsos. Game based early programming education: The more you play, the more you learn. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7544:115–131, 2013.

[PB10] Velian T Pandeliev and Ronald M Baecker. A framework for the online evaluation of serious games. In *Proc. of the Int. Academic Conf. on the Future of Game Design and Technology*, pages 239–242. ACM, 2010.

[PCW08] Neil Peirce, Owen Conlan, and Vincent Wade. Adaptive educational games: Providing non-invasive personalised learning experiences. In *2008 Second IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, pages 28–35. IEEE, 2008.

[PDKI07] Karolien Poels, YAW De Kort, and WA IJsselsteijn. *Game Experience Questionnaire: Development of a Self-Report Measure to Assess the Psychological Impact of Digital Games*. Technische Universiteit Eindhoven, 2007.

[PKW+02] Randy J Pagulayan, Kevin Keeker, Dennis Wixon, Ramon L Romero, and Thomas Fuller. *User-centered Design in Games*. CRC Press Boca Raton, FL, 2002.

[PM+00] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *Icml*, volume 1, pages 727–734, 2000.

[Pon04] Marc Ponsen. *Improving Adaptive Game AI with Evolutionary Learning*. PhD thesis, Delft University of Technology, 2004.

[Por09] Arthur E Poropat. A meta-analysis of the five-factor model of personality and academic performance. *Psychological bulletin*, 135(2):322, 2009.

[PP01] Minna Puustinen and Lea Pulkkinen. Models of self-regulated learning: A review. *Scandinavian Journal of Educational Research*, 45(3):269–286, 2001.

[Pre03] Marc Prensky. Digital game-based learning. *Computers in Entertainment (CIE)*, 1(1):21–21, 2003.

[QTB+12] ES Quellmalz, MJ Timms, BC Buckley, M Silberglitt, and D Brenner. Simscientists: Measurement in simulation-based science assessments. *Unpublished manuscript*, 2012.

[Ras60] Georg Rasch. Studies in mathematical psychology: I. probabilistic models for some intelligence and attainment tests. 1960.

[Ray07] Elaine M Raybourn. Applying simulation experience design methods to creating serious game-based adaptive training systems. *Interacting with Computers*, 19(2):206–214, 2007.

[RIPB15] Barbara Reichart, Damir Ismailovic, Dennis Pagano, and Bernd Brügge. Adaptive serious games. *Computer Games and Software Engineering*, 9:133, 2015.

[Roy08] Karl Royle. Game-based learning: A different perspective. *Innovate: Journal of Online Education*, 4(4), 2008.

[RRR03] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[RTC+07] Jeremy Roschelle, Deborah Tatar, S Raj Chaudhury, Yannis Dimitriadis, Charles Patton, and Chris DiGiano. Ink, improvisation, and

interactive engagement: Learning with tablets. *Computer*, 40(9), 2007.

[SB82] Derek Sleeman and John Seely Brown. *Intelligent tutoring systems*. 1982.

[SBF+17] Claudia Schrader, Julia Brich, Julian Frommel, Valentin Riemer, and Katja Rogers. Rising to the challenge: An emotion-driven approach toward adaptive serious games. In *Serious Games and Edutainment Applications*, pages 3–28. Springer, 2017.

[SC93] Malcolm Simmons and Peter Cope. Angle and rotation: Effects of different types of feedback on the quality of response. *Educational Studies in Mathematics*, 24(2):163–176, 1993.

[SC09] David J Shernoff and Mihaly Csikszentmihalyi. Cultivating engaged learners and optimal learning environments. *Handbook of Positive Psychology in Schools*, pages 131–145, 2009.

[She91] Lorrie A Shepard. Will national tests improve student learning? *The Phi Delta Kappan*, 73(3):232–238, 1991.

[Shu11] Valerie J Shute. Stealth assessment in computer-based games to support learning. *Computer Games and Instruction*, 55(2):503–524, 2011.

[SJB07] Tarja Susi, Mikael Johannesson, and Per Backlund. Serious games: An overview, 2007.

[SKRM+12] Christina M Steiner, Michael D Kickmeier-Rust, Elke Mattheiss, Stefan Göbel, and Dietrich Albert. Balancing on a high wire: adap-

tivity key factor of future learning games. *An Alien's Guide to Multi-adaptive Educational Computer Games*, 43, 2012.

[SKW17] Valerie Shute, Fengfeng Ke, and Lubin Wang. Assessment and adaptation in games. In *Instructional Techniques to Facilitate Learning and Motivation of Serious Games*, pages 59–78. Springer, 2017.

[SMY15] Takayuki Sekiya, Yoshitatsu Matsuda, and Kazunori Yamaguchi. Curriculum analysis of cs departments based on cs2013 by simplified, supervised lda. In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*, pages 330–339, 2015.

[SNA09] Christina M Steiner, Alexander Nussbaumer, and Dietrich Albert. Supporting self-regulated personalised learning through competence-based knowledge space theory. *Policy Futures in Education*, 7(6):645–661, 2009.

[SPSKP06] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game ai with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[Squ05] Kurt Squire. Changing the game: What happens when video games enter the classroom? *Innovate: Journal of Online Education*, 1(6), 2005.

[SS16] Alexander Streicher and Jan D Smeddinck. Personalized and adaptive serious games. In *Entertainment Computing and Serious Games*, pages 332–377. Springer, 2016.

[Ste06] Robert J Sternberg. The nature of creativity. *Creativity research journal*, 18(1):87, 2006.

[SUFR06] Richard Sandford, Mary Ulicsak, Keri Facer, and Tim Rudd. Teaching with games - using commercial off-the-shelf computer games in formal education. *Futurelab*, 112:12, 2006.

[SV13] Valerie Jean Shute and Matthew Ventura. *Stealth assessment: Measuring and Supporting Learning in Video Games*. MIT Press, 2013.

[SVBZR09] Valerie J Shute, Matthew Ventura, Malcolm Bauer, and Diego Zapata-Rivera. Melding the power of serious games and embedded assessment to monitor and foster learning. *Serious games: Mechanisms and effects*, 2:295–321, 2009.

[TB14] Nikolai Tillmann and Judith Bishop. Code hunt: Searching for Secret Code for Fun. In *Proceedings of the 7th International Workshop on Search-Based Software Testing*, pages 23–26, 2014.

[THHS07] Sabrina Trapmann, Benedikt Hell, Jan-Oliver W Hirn, and Heinz Schuler. Meta-analysis of the relationship between the big five and academic success at university. *Zeitschrift für Psychologie/Journal of Psychology*, 215(2):132–151, 2007.

[TY10] James M Thomas and R Michael Young. Annie: Automated generation of adaptive learner guidance for fun serious games. *IEEE Transactions on Learning Technologies*, 4(3):329–343, 2010.

[VE07] Richard Van Eck. Building artificially intelligent learning games. In *Games and simulations in Online Learning: Research and Development Frameworks*, pages 271–307. IGI Global, 2007.

[Veg08] Jay Vegso. Enrollments and degree production at us cs departments drop further in 2006/2007. *Computing Research News*, 20(2):4, 2008.

[Vil92] Michael Villano. Probabilistic student models: Bayesian belief networks and knowledge space theory. In *International Conference on Intelligent Tutoring Systems*, pages 491–498. Springer, 1992.

[VLS+05] Kurt VanLehn, Collin Lnch, Kay Schulze, Joel A Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. The andes physics tutoring system: Five years of evaluations. Technical report, Naval Academy Annapolis MD, 2005.

[VMM14] Adilson Vahldick, Antonio Jose Mendes, and Maria Jose Marcelino. A review of games designed to improve introductory computer programming competencies. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 2015-Febru(February):1–7, 2014.

[VOCFC14] Roland Van Oostveen, Elizabeth Childs, Kathleen Flynn, and Jessica Clarkson. *Integration of PBL Methodologies into Online Learning Courses and Programs.* ERIC, 2014.

[Vyg80] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes.* Harvard university press, 1980.

[Wal03] Henry M Walker. Do computer games have a role in the computing classroom? *ACM SIGCSE Bulletin*, 35(4):18–20, 2003.

[Wat91] Mike Watts. *The science of problem solving: A practical guide for science teachers.* Burns & Oates, 1991.

[WBPW06] Ursula Wolz, Tiffany Barnes, Ian Parberry, and Michael Wick. Digital gaming as a vehicle for learning. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 394–395, 2006.

[WDD11] Joost Westra, Frank Dignum, and Virginia Dignum. Scalable adaptive serious games using agent organizations. In *Proc. of the 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1291–1292. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[WRN+15] Mats Wiklund, William Rudenmalm, Lena Norberg, Thomas Westin, and Peter Mozelius. Evaluating educational games using facial expression recognition software: measurement of gaming emotion. In *European Conference on Games Based Learning*, page 605. Academic Conferences International Limited, 2015.

[WSB14] Stefanie Wetzel, Katharina Spiel, and Sven Bertel. Dynamically adapting an ai game engine based on players' eye movements and strategies. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 3–12. ACM, 2014.

[YW15] Wong Seng Yue and Wong Lai Wan. The effectiveness of digital game for introductory programming concepts. In *Proc. of the 10th Int. Conf. for Internet Technology and Secured Transactions (ICITST 2015)*, pages 421–425, 2015.

[ZCMM18] D Zhao, AE Chis, GM Muntean, and CH Muntean. A large-scale pilot study on game-based learning and blended learning methodologies in undergraduate programming courses. In *Proceedings of the EDULEARN Conference, Palma de Mallorca, Spain*, 2018.

[Zim02] Barry J Zimmerman. Becoming a self-regulated learner: An overview. *Theory into Practice*, 41(2):64–70, 2002.

[ZR12] Alexander E Zook and Mark O Riedl. A temporal data-driven player model for dynamic difficulty adjustment. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

Appendix A

Appendix

A.1 Game Experience Questionnaire

Please indicate how you felt while playing the game for each of the items, on the following scale:

1 - not at all

2 - slightly

3 - moderately

4 - fairly

5 - extremely

Table A.1: GEQ Questions

Statement shown to participants	**Variable measured**
I felt successful	Competence
I felt bored	Negative Affect
I found it impressive	Immersion
I forgot everything around me	Flow
I felt frustrated	Tension
I found it tiresome	Negative Affect
I felt irritable	Tension
I felt skilful	Competence
I felt completely absorbed	Flow
I felt content	Positive Affect
I felt challenged	Challenge
I had to put a lot of effort into it	Challenge
I felt good	Positive Affect

A.2 RoboBUG Skill Test

Multiple Choice

Choose the best answer for each question.

1) What is **code tracing**?
    a. **Reading through code to make sure it is behaving properly**
    b. Separating code by section to isolate a bug
    c. Observing code during run-time and checking the internal value of variables
    d. Running code with inputs and ensuring the output gives the desired result
2) What is **testing**?
    a. Reading through code to make sure it is behaving properly
    b. Separating code by section to isolate a bug
    c. Observing code during run-time and checking the internal value of variables
    d. **Running code with inputs and ensuring the output gives the desired result**
3) What is a **divide-and-conquer** approach?
    a. Reading through code to make sure it is behaving properly
    b. **Separating code by section to isolate a bug**
    c. Observing code during run-time and checking the internal value of variables
    d. Running code with inputs and ensuring the output gives the desired result
4) What are **breakpoints** used to do?
    a. Reading through code to make sure it is behaving properly
    b. Separating code by section to isolate a bug
    c. **Observing code during run-time and checking the internal value of variables**
    d. Running code with inputs and ensuring the output gives the desired result
5) What is **black box** testing?
    a. Testing code while looking at it during run-time
    b. **Testing code without seeing its internal behavior**
    c. Testing code that requires a very large number of test cases
    d. Testing code that has unknown behavior
6) Suppose you are debugging code where you need to know the values of variables during run-time. Which of these methods is appropriate? **(Choose up to 3)**
    a. **Adding Print statements**
    b. **Using Breakpoints**
    c. Black box testing
7) When is a divide-and-conquer approach useful? **(Choose up to 3)**
    a. **Some of the code is faulty, but most of it is functional**
    b. **Code that is known to be functional does not need to be run, or can be easily replaced**
    c. **There is a large amount of code that makes code tracing infeasible.**
8) Which of the following techniques do not require you to actually run any code?
    a. **Code tracing**
    b. Black box testing
    c. Debugging with breakpoints
    d. Divide-and-conquer approach

9) Given the code below, which of these test cases is most appropriate for determining if there is an error in the maximum function?

```
1.  int Foo (int a, int b, int c)
2.  {
3.      if(a>b){
4.          return minimum(a,b,c);
5.      }
6.      else if(c>a){
7.          if(c>b){
8.              return c;
9.          }
10.         else {
11.             return maximum(a,b,c);
12.         }
13.     }
14.     return a;
15. }
```

a. Foo(3,2,1)
b. Foo(0,0,0)
c. Foo(3,1,0)
d. Foo(0,0,1)
e. Foo(0,2,1)

10) In the code below, where is the best place for a breakpoint if you want to understand more about the array values during each iteration of the sort?

a. Line 11
b. Line 14
c. Line 15
d. Line 17

```
1.  //Sorts a list of numbers
2.  //Input : List of numbers
3.  //Output : Sorted list
4.
5.  void BubbleSort (int array[],int size)
6.  {
7.      int i = 0;
8.      int temp;
9.      bool swapped = true;
10.     while(swapped){
11.         swapped = false;
12.        while(i<size-1){
13.            if(array[i]<array[i+1]){
14.                temp = array[i];
15.                array[i] = array[i+1];
16.                array[i+1] = temp;
17.                swapped = true;
18.            }
19.            i++;
20.        }
21.    }
22. }
```

A.3 Original RoboBUG Study

RoboBUG: A Serious Game for Learning Debugging Techniques

Michael A. Miljanovic
University of Ontario Institute of Technology
2000 Simcoe Street North
Oshawa, Ontario, Canada
michael.miljanovic@uoit.ca

Jeremy S. Bradbury
University of Ontario Institute of Technology
2000 Simcoe Street North
Oshawa, Ontario, Canada
jeremy.bradbury@uoit.ca

ABSTRACT

Debugging is an essential but challenging task that can present a great deal of confusion and frustration to novice programmers. It can be argued that Computer Science education does not sufficiently address the challenges that students face when identifying bugs in their programs. To help students learn effective debugging techniques and to provide students a more enjoyable and motivating experience, we have designed the RoboBUG game. RoboBUG is a serious game that can be customized with respect to different programming languages and game levels.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; • **Software and its engineering** → *Software testing and debugging*; • **Applied computing** → Computer games;

KEYWORDS

debugging, programming, software engineering, computer science, education, serious games, game-based learning

1 INTRODUCTION

Research related to programming and software development often focuses on advancing the state-of-the-art in software developer practices, techniques and tools. Research into programming and software development learning tools is equally important as software developers must first learn best practices before they can effectively perform them.

It is essential that programmers who seek to write reliable, high quality source code be able to efficiently identify and repair bugs in program code [24]. The process of fixing these bugs, *debugging*, has been shown to consume up to 50% of a programmer time in large software projects [5]. Furthermore, the ability to debug code is not easily acquired, and experts have a significant advantage over novices [3]. In addition to experience, experts have knowledge of a variety of debugging techniques including code tracing, instrumentation and the use of breakpoints in debuggers.

Game-based learning has already been implemented and proven effective for computer programming education [10, 16] which suggests that it may also prove useful in debugging education. Serious games should not only help players achieve learning outcomes but should also provide a fun and positive experience. Previous studies have shown that motivated and engaged learners will perform more effectively [8, 18]. Finally, the benefits of serious games suggest they may be an effective way to counter the frustration typically associated with debugging.

First year courses often do not teach debugging explicitly, and expect students to learn it for themselves. This lack of preparedness is compounded by the fact that there is no established set of best practices for teaching debugging [7]. This is compounded by a lack of online resources dedicated to helping novices learn to debug [3]. Even students with a good understanding of how to write programs still struggle with debugging [1]. These students may have the ability to fix bugs in their programs, but only after accomplishing the difficult task of finding the bugs first.

In general, the lack of accessibility to debugging techniques leads students to have a primarily negative experience, even when given the opportunity to learn debugging [20]. This fact is particularly problematic when students conclude that debugging skills are based on aptitude and are unable to be learned [4]. We hope to address the problem of accessibility and frustration with debugging education by creating RoboBUG, a puzzle-based serious game, that is designed to help students achieve debugging learning outcomes in an enjoyable rather than tedious way.

The creation of RoboBUG required us to address a number of challenges, including:

(1) **Game design:** How can debugging activities be represented as game tasks/actions? How can these tasks be connected to produce enjoyable and cohesive gameplay?
(2) **Game learning data:** What debugging topics and learning materials should be used in the game to achieve the desired learning outcomes while minimizing frustration?
(3) **Game evaluation:** How do we design our study to effectively assess debugging learning and level of enjoyment?

In addition, we needed to consider if the combination of game design and learning data challenges will allow a player to retain debugging technique knowledge after the game's completion.

In the remaining sections of our paper we present background on debugging and game-based learning (Section 2), an overview of our RoboBUG serious game (Section 3), the results of a pilot study (Section 4.1) as well as the results of two full studies (Section 4.2 and Section 4.3). Our studies were conducted with undergraduates at the University of Ontario Institute of Technology (UOIT).

2 BACKGROUND

2.1 Debugging

As mentioned in the previous section, debugging is the processing of finding and fixing problems (bugs) in a program. Debugging can involve the use of dedicated debugger tools and can use a combination of both static static and dynamic debugging techniques. Static techniques are those that do not require execution of the program

Conference'17, Washington, DC, USA
2020. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

Figure 1: A screenshot of the RoboBUG game

The RoboBUG game interface is divided into two regions: (1) The code region (left) in which the user controls an avatar to navigate the source code while using different debugging tools to eventually find bugs, (2) The sidebar region (right) in which the user can view information about available debugging tools, the currently active tool and the time remaining in the level. In addition to these two regions, the RoboBUG game also utilizes dialogs (bottom) to provide contextual information to the user including feedback from debugging activities.

while dynamic techniques rely on run-time information. Common debugging techniques include code tracing, print statements, divide-and-conquer and breakpoints. **Code tracing** is a static technique in which the programmer reads through code to make sure it is behaving properly. **Print statements** are a code injection approach to inserting output statements into the program that display internal program status information as output. **Divide-and-conquer** is a debugging strategy that allows a programmer to systematically separate source code into sections in an effort to isolate a bug. **Breakpoints** are a common feature in modern debuggers that allows the execution of a program to be paused in order to allow the programmer to view the internal value of variables at specific execution points.

After conducting a review of debugging techniques we decided to select the above four techniques for inclusion in the RoboBUG game. Other debugging techniques, such as black-box testing, were excluded in order to constrain the duration of the game. Although the chosen techniques are only applied in specific levels of the game, there is an overarching theme of program comprehension. Experts are faster and more effective debuggers than novices due to superior program comprehension strategies [14], which is why we chose to emphasize the importance of comprehension by having players debug code they did not write themselves.

In addition, we selected types of bugs that are common in student-written code. As syntax errors are often identified by compilers, they tend to be less problematic for students than logic or data errors [9]. Thus, we chose to include only logic and data errors in our tool, and because we believe the techniques we selected are best explained through their abilities to find these types of errors.

2.2 Game-based Learning

Serious games for Computer Science is an active research area [17, 23], especially with respect to learning how to write computer programs. Games such as Code Hunt [21] help learners develop their skills through puzzle tasks that require players to write programs in order to solve a specific problem. Serious programming games usually focus on the act of creating programs by writing source code, or alternatively by using a drag-and-drop interface, as seen with Program Your Robot [11]. A nonstandard example of a puzzle-based programming game is Robot ON! [15], that does not require players to write any programs. Instead, Robot ON! focuses on program understanding and comprehension by requiring players to read source code written by someone else.

Some games such as Gidget [13] have been designed with an emphasis on helping players learn about debugging. However, Gidget is meant to introduce general debugging, and uses unique pseudo-code instead of a common language such as C++ or Java. In addition,

Figure 2: A RoboBUG comic storyboard used to advance the game plot and setup a new game level

The RoboBUG game utilizes a plot to engage the user in the debugging learning tasks. Each level begins with a four panel comic that provides plot details and connects each level with an overall story. For example, in the above comic, alien bugs have infected the player's mech suit and caused the vision system to malfunction. In the corresponding level the user is tasked with detecting bugs in the mech suit's vision system source code.

we did not find any serious games in the literature that specifically focus on learning debugging techniques.

3 THE ROBOBUG GAME

RoboBUG[1] (see Figure 1) is a serious game intended to be played by first-year computer science students who are learning to debug for the first time. We chose to design RoboBUG as a puzzle-type game, as puzzles have been shown to be effective for both helping to learn material as well as demonstrating higher level concepts such as critical thinking and problem-solving [19].

RoboBUG was implemented in C# using the Unity game engine[2] and open source media elements. Although the standard version of RoboBUG is based on debugging in C++, it also includes a framework that allows instructors to create their own levels using other programming languages. New levels are specified using XML and can be customized with respect to different aspects of a level, including time limit, available tools, output text, and source code. These new levels can be inserted into the game with minimal effort.

In the RoboBUG game a player takes the role of a scientist whose world is under attack from an alien bug world. The alien world has sent an advanced army of tiny bugs that infect technology in the scientist's world – including the scientist's 'Mech Suit' (a robotic suit of armour). In order to help save the world from the alien bugs, the scientist must first purge bugs from the infected 'Mech Suit'. Bugs are purged by the scientist by virtually entering the infected source code to find all of the alien bugs. In each level the player (taking the role of the scientist) must fix a particular part of the Mech Suit (e.g. the vision system) by figuring out where the bug is hiding (see Figure 2). the game is finished once the player has completed all levels, found all of the bugs, and has a working Mech Suit. The actions required by different debugging techniques are represented as tools that the player can aim at lines of code. For example, a Breakpointer tool can be aimed at a line of code to insert a break point and a Warper tool can be aimed at a function call to jump to another part of the code (the function definition).

The default version of RoboBUG includes four levels that teach different debugging techniques in C++ (see Table 1). Each of these levels includes: a tutorial that introduces new debugging tools, 2-3 subproblems that contain small debugging tasks and partial source code and a final challenge that combines the tools introduced in the tutorials and the knowledge gained from the subproblems. The final challenge will typically involve detecting a bug in the full program.

A player's progress through the game is recorded in a set of log files that allow gameplay to stopped and resumed. Prototype testing has shown that the game with the default levels takes approximately 30 minutes to complete. RoboBUG has been used with the four default levels during an introductory programming course at UOIT.

3.1 Game Levels

In developing the default version of RoboBUG, we thought about the order and content that should be included in the game levels. We chose to first introduce code tracing as it can be used in combination with other techniques and it is not too time-consuming due to the short length of the example programs. Next, we selected the use of print statements in order to help novices learn to identify program behavior at run-time. This was followed by the strategy of divide-and-conquer, where novices can reduce the search space for bugs by commenting out code that is guaranteed to contain no bugs. Finally, we adapted some features of a debugger so that novices can learn the concepts of breakpoints and the value of observing a program's execution state during run-time.

The following subsections provide a brief walkthrough of the game levels in RoboBUG. Each level includes several parts that build upon each other with the final part requiring the player to debugging a subsystem of the 'Mech Suit'.

3.1.1 *Level 1: Code Tracing.*

- *Subproblem A:* A mathematical function definition is provided to the player that includes input, output, and behavior. The function is intended to return an average of a set of values (a floating point number), and the player's goal is to trace through the code and identify that the 'avgf' variable (responsible for storing the average) has a type (boolean) that doesn't match its intended use.
- *Subproblem B:* The source code from Subproblem A is expanded to calculate the average of a list of numbers using a loop; however, the player needs to identify that the loop adds the 'avgf' variable to a running total sum rather than properly averaging the numbers.
- *Subproblem C:* The source code is expanded again and now contains the full function for calculating the average of

[1] https://github.com/sqrlab/robobug
[2] https://unity3d.com/

Table 1: An overview of the levels and tools in RoboBUG

Level	Tools	Description
Level 1	Bugcatcher	The Mech Suit is unable to stand because it cannot correctly calculate the physical forces acting upon it. The player must practice **code tracing** by identifying bugs while reading through source code that calculates the average value of a set of physical forces.
Level 2	Bugcatcher, Activator	The Mech Suit is failing to correctly identify the most dangerous creatures that appear in its viewing area. The player needs to use **print statements** to identify program bugs in an algorithm that sorts the externally viewable bugs from most to least dangerous (threat assessment).
Level 3	Bugcatcher, Activator, Commenter, Warper	The Mech Suit vision system has been infected and no longer functions at all. To fix it, the player must search for a bug in the robot's visual color database. Since the database is large, the player will need to employ a **divide-and-conquer** strategy and comment out different blocks of source code.
Level 4	Bugcatcher, Activator, Breakpointer, Warper	The Mech Suit is not able to correctly calculate which creatures are closest in proximity. This is the most challenging level, requiring the player to use several debugging tools to locate bugs across multiple functions. This includes the use of **breakpoints** to display variable values and program state at run-time while locating the bug in a distance calculation function.

physical forces acting upon the 'Mech Suit'. However, an extraneous line of source code is present that increments the average after it has been calculated correctly.

3.1.2 *Level 2: Print Statements.*

- *Subproblem A:* The player is presented with a function that should swap two numbers; however, the final line of code performs the operation in reverse, and assigns the values incorrectly. This behavior can be identified when the player enables the appropriate print statements which show that the final result only includes one correctly swapped value.
- *Subproblem B:* The swap function from the Subproblem A has been incorporated into a function that sort a list of numbers that unfortunately includes an out-of-bounds indexing error. Using print statements the player can discover this area and identify that the bug is in the loop condition (a '>=' sign is used instead of a '>').
- *Subproblem C:* The final expanded source code is part of the threat assessment component in the 'Mech Suit' and contains a function that sorts a list of threat rankings. The print statements in this level show the state of the list at different stages of sorting. Observing each print statement should help the player realize that the first element of the list is accidentally not sorted and leads to an incorrect list.

3.1.3 *Level 3: Divide and Conquer.*

- *Subproblem A:* The player is presented with a large list of integer red-green-blue (RGB) color tuples, which each range between 0 and 255; these represent different colors in the 'Mech Suit' vision subsystem. A print statement at the beginning of the function indicates that one of the green values is out of bounds. By using a divide-and-conquer approach to commenting out the different code sections for different colors, the player can identify that one of the colors has an out of range green value.
- *Subproblem B:* The code in this level consists of multiple large lists of RGB colors, and a print statement at the beginning of the code indicates that there is an invalid blue color value. The player must use the commenter tool to comment out each list of colors until the error is located.
- *Subproblem C:* This level is similar to Subproblem B, except the source code color lists are divided across multiple files that must be checked separately. The player uses the commenting tool to comment out each file until they discover which file contains the error. Using their 'warper' tool, they can then warp to that file and isolate the invalid value.

3.1.4 *Level 4: Breakpoints.*

- *Subproblem A:* The source code in this part takes two pairs of numbers representing a location (x and y coordinates), and indicates which pair has a lower magnitude. The player uses code tracing to discover that there is a '=' instead of a '==' in a comparison statement. While this part of the level does not use breakpoints it is included to show the benefits of breakpoints in the subsequent parts of the level.
- *Subproblem B:* This part of the level requires the player to use breakpoints to check the values of coordinates used in each function, and identify a small logic error.
- *Subproblem C:* This part is similar to Subproblem B, but contains an error where a variable is unintentionally reassigned instead of used in a calculation.
- *Subproblem D:* The source code in the final subproblem of this level contains a small logic error in a purposefully obfuscated calculation. The error is obfuscated to enhance the benefits of breakpoints in finding the bug. This source code is from the 'Mech Suit' subsystem that calculates locations to target.

3.2 Game Mechanics

In order to complete a level, the player must navigate the avatar in the code region of the game interface (see Figure 1) using the arrow keys. Once a bug has been found, the player uses the *bugcatcher* tool to "capture" the bug at a specific line of code. If the location is incorrect, the player fails the level and must start it again. The player can also fail if he or she expends all of the available tools, or does not complete the level within the time allotted. As the game progresses,

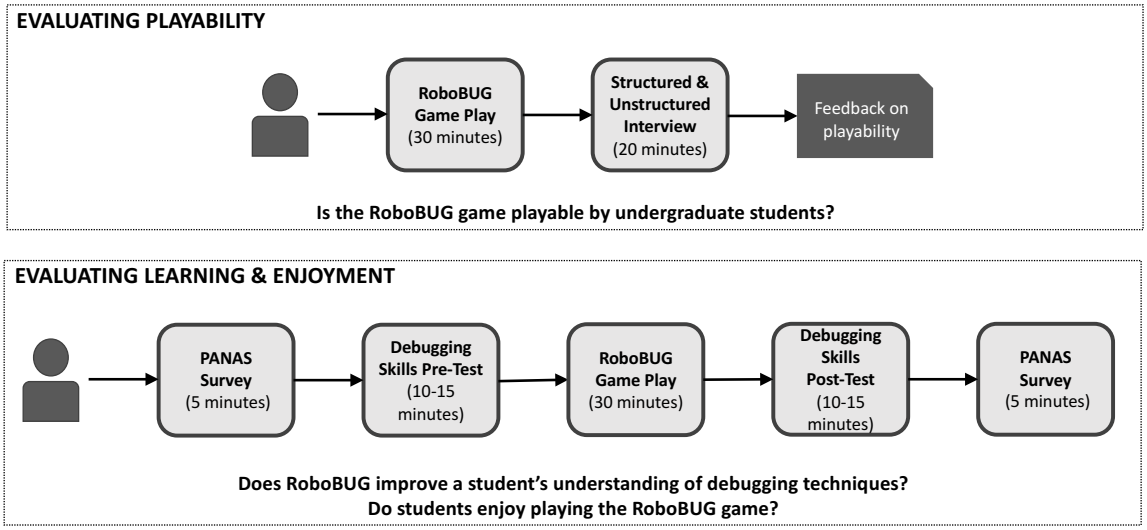


Figure 3: The RoboBUG evaluation methodology

the player is given access to additional kinds of tools that can activate print statements (*activator* tool), comment out source code (*commenter* tool), set and trigger breakpoints (*breakpointer* tool) and jump to different regions of the program (*warper* tool). Completion of each level requires the player to use the tools available before using the 'bugcatcher' tool to complete the level.

4 ROBOBUG EVALUATION

Serious games, including those in the Computer Science education literature, tend to be published without a proper evaluation [12] making it difficult to assess their impact on learning. The most effective type of evaluation to determine the efficacy of a game for learning is a user study [6]. In our evaluation of RoboBUG we have conducted three separate user studies with 23, 5 and 14 participants respectfully. Our first study (Section 4.1) was a pilot study of a first prototype of RoboBUG, our second study (Section 4.2) assessed the playability of the refined version of RoboBUG and our third study (Section 4.3) assessed the enjoyability and the achievement of learning outcomes in the refined version of RoboBUG.

4.1 Pilot Study

A pilot study of an early RoboBUG prototype assessed the value of the game in comparison with traditional assignment-based learning. All participants were undergraduate students at UOIT with knowledge of the C++ programming language. Participants were split into two random groups: a control group of 12 participants who completed a short assignment, and an experimental group of 11 participants who played the RoboBUG game. Both the assignment and the game were based on the same debugging techniques (see Section 2.1) and included similar source code – each level in the RoboBUG game had a corresponding assignment question.

The evaluation found no significant difference with regards to achieving learning outcomes between the assignment-based learning activity and the RoboBUG prototype. We believe this result

1 Very Slightly / Not at all	2 A Little	3 Moderately	4 Quite a bit	5 Extremely

1. Interested	_____	5. Strong	_____
2. Distressed	_____	6. Guilty	_____
3. Excited	_____	7. Scared	_____
4. Upset	_____	8. Enthusiastic	_____

Figure 4: Positive-Negative Affect Scale (PANAS) [22]

The Positive-Negative Affect Scale is a self-evaluation assessment of affect based on words that associate with positive or negative emotions. Total affect is calculated by adding all of the positive or negative item ratings, with higher scores representing higher affect levels.

was impacted by the fact that study participants who played the RoboBUG prototype found it complicated, and some participants were not able to complete the game without hints. Despite this, participants who played RoboBUG tended to find it to be more 'fun'. After the results of the pilot study RoboBUG was updated to address these issues, by subdividing levels, reducing complexity, and providing opportunities for players to fail and replay the levels.

4.2 Evaluating Playability

To evaluate the current version of RoboBUG, we first conducted a user study to address the following research question:

- *Is the RoboBUG game playable by undergraduate students?*

This was an important research question to answer first because not identifying and addressing issues with playability could serious impact our ability to assess the learnability and enjoyment of RoboBUG. In other words, we don't want design and technical issues to confound our evaluation of RoboBUG's potential as a learning tool for debugging.

Our evaluation involved the participation 5 first year Computer Science students at UOIT who were familiar with C++. Participants were between the ages of 18 and 25, with mixed demographics. Participants individually took part in a 1 hour session during which they were observed playing the RoboBUG game for at least 30 minutes (see Figure 3). Following the game play, the participants completed a 20 minute interview where they answered both structured and unstructured questions about their experience, including:

- What did you learn about debugging that you didn't know before?
- What aspect/part of the game was most enjoyable?
- What aspect/part of the game was the most frustrating?
- What aspect/part of the game was most innovative?
- What aspect/part of the game would you like to see improved?

During the interview, participants provided feedback about parts of the game where they became stuck or frustrated. The goal was for RoboBUG to be playable before measuring its efficacy as a game.

Overall, the game was viewed positively by the participants, who particularly enjoyed the game elements that differentiated RoboBUG gameplay from real debugging tasks. During the interviews, the participants gave the following opinions:

- *"[I enjoyed] trying to test my skills with how good I am with debugging."*
- *"It's a great tool, that's what I can say."*
- *"The way that the divide and conquer was set up was pretty cool."*
- *"The inclusion of breakpoints was kind of innovative."*
- *"[The warper tool] was interesting because I thought all of the code would be in one class."*
- *"I think that the warper/commenting, being able to zip between different segments of code was really good."*

While participants enjoyed playing RoboBUG, our study did identify some important playability issues with the game, including control problems and concerns with some of the game's levels. In particular, participants in all our evaluations had significant challenges trying to debug a level containing an off-by-one index bug. This bug was cited by participants to be especially frustrating, due to players having trouble identifying print statements that would help them find the bug. There was also some confusion with the way that the game handled commenting out source code, as players did not realize that a persisting error meant the bug was **not** commented out. A frequently requested change to the game was the idea of a 'hint' system that would provide better feedback to players who become frustrated or fail to complete levels.

RECALL – WHAT?

What can **print statements** be used for in debugging?

a. Outputting the value of a particular variable
b. Indicating the code that is not run during execution
c. Printing a fixed version of buggy code
d. Separating buggy code from bug-free code using text

UNDERSTANDING – WHEN?

Suppose you are debugging code where you need to know the values of variables during run-time. Which methods are appropriate?

e. Print statements or breakpoints
f. Breakpoints or divide-and-conquer
g. Divide-and-conquer or print statements
h. Print statements, breakpoints or divide-and-conquer

APPLICATION – HOW?

In the code below, where is the best place for a breakpoint if you want to find out the array values during each iteration of the sort?

i. Line 5
j. Line 7
k. Line 9
l. Line 11

```
1.          //Sorts a list of numbers
2.  //Input : List of numbers
3.  //Output : Sorted list
4.
5.  void BubbleSort (int array[],int
    size)
6.  {
7.      int i = 0;
8.      int temp;
9.      bool swapped = true;
10.     while(swapped){
11.         swapped = false;
12.         while(i<size-1){
13.             if(array[i]<array[i+1]){
14.                 temp = array[i];
15.                 array[i] = array[i+1];
16.                 array[i+1] = temp;
17.                 swapped = true;
18.             }
19.             i++;
20.         }
21.     }
22. }
```

Figure 5: Sample Skill-Testing Questions

*Listed in this figure are three of the ten questions included in the skill test given to participants before and after gameplay. The test was divided into three categories of questions: **recall** about **what** the techniques were, **understanding when** to use each technique, and **application** of **how** to use debugging techniques.*

Positive Keyword	Average Change	Negative Keyword	Average Change
Interested	-0.64	Anxious	-0.21
Enthusiastic	-0.64	Nervous	-0.14
Alert	-0.50	Guilty	-0.07
Excited	-0.36	Stressed	-0.07
Determined	-0.36	Depressed	-0.07
Attentive	-0.36	Scared	0.00
Proud	-0.14	Distressed	0.07
Inspired	-0.14	Hostile	0.07
Happy	-0.07	Jittery	0.29
Confident	0.00	Afraid	0.29
Active	0.07	Irritable	0.36
Strong	0.14	Upset	0.43
		Ashamed	0.43

Figure 6: Positive-Negative Affect Scores

*This graph shows the average change of affect for all participants based on each question on the PANAS. Positive scores indicate that participants associated **more** with that emotion after playing RoboBUG, and negative scores indicate that players associate **less** with that emotion after playing RoboBUG.*

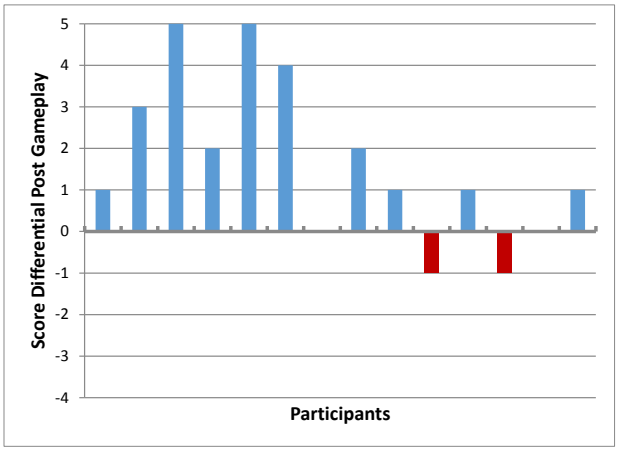


Figure 7: Change in Skill Test Scores by Participant after playing RoboBUG

Each bar represents a single participant's score differential on the Skill Test after playing RoboBUG. The participants are ordered from left to right based on increasing initial Skill Test scores (before playing RoboBUG). An interesting observation is that the largest positive score differentials were achieved by students with the lowest initial Skill Test scores.

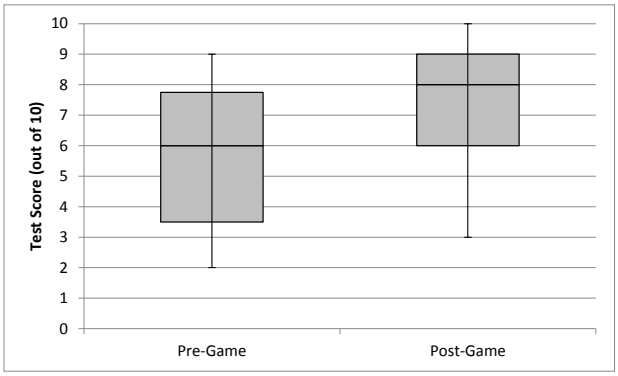


Figure 8: Box plot of the Skill Test Scores before and after playing RoboBUG

4.3 Evaluating Learning and Enjoyment

To further evaluate the current version of RoboBUG, we conducted a second user study to address the following research questions:

- *Does RoboBUG improve a student's understanding of debugging techniques (i.e., achieve learning outcomes)?*
- *Do students enjoy playing the RoboBUG game?*

Our evaluation again involved the participation of first year Computer Science students at UOIT between the ages of 18 and 25, with mixed demographics, gender, and race.

In this study we evaluated the game's ability to help students achieve learning outcomes as well as the user experience. This study involved a larger sample size than the accessibility study (14 students) and took approximately 1 hour to complete. Participants in this study first completed the Positive and Negative Affect Scale (PANAS) [22], which assesses the user's feelings (see Figure 4). The PANAS scale in our study contained 12 positive words and 13 negative words. After completing PANAS, participants completed a debugging skills pre-test. This pre-test included ten multiple choice questions about the four debugging techniques used in the game (see Section 2.1). These questions tested participant abilities to recall, understand, and apply the debugging techniques. The first four questions tested knowledge about the usage of each debugging technique. The next four questions tested understand of when the techniques should be used. Finally, the last two questions involved the application of the techniques themselves. Examples of these questions can be observed in Figure 5. Once the pre-test was complete, participants played the RoboBUG game for approximately 30 minutes. After the game, participants completed the debugging skills test and the PANAS questionnaire again.

The results of both the pre- and post-game PANAS data and skill test data were analyzed to identify any changes in positive and negative effect as well as debugging skills. The analysis was conducted using a paired t-test (see Figure 9). Our results indicate that RoboBUG helps students to achieve the debugging learning outcomes (see Figure 7 and Figure 8). Players became familiar with the nature of the debugging techniques, and were able to practice debugging and solve problems in a satisfying manner. In addition, the largest improvements in test scores were observed for participants with low initial test scores, suggesting that the game is most helpful for participants who are in the greatest need of assistance. Unfortunately, there was a non-statistically significant decrease in positive affect and a non-statistically significant increase in negative affect, indicating that the game still led to some user frustrations. It is possible that the game remains less frustrating than real debugging tasks, but our observations of participants suggest that the lack of

Paired-samples t-test				
	Mean	Std. Dev.	t(13)	p value
Test Scores			3.0970	0.0085
Pre-Game	5.71	2.46		
Post-Game	7.36	2.10		
Positive Affect			2.1272	0.0531
Pre-Game	45.93	5.40		
Post-Game	42.93	8.92		
Negative Affect			0.7555	0.4634
Pre-Game	18.86	5.67		
Post-Game	20.21	10.64		

Figure 9: Paired-samples t-test

There was a significant increase in debugging test scores after the game was played. No significant changes in positive or negative affect scores were observed.

a hint system and the difficulty of the tasks were major challenges. Ultimately, our game still requires participants to debug code and completely removing frustration related to debugging remains an open problem.

5 SUMMARY & CONCLUSIONS

We have presented the RoboBUG game as a serious game solution to the challenge of learning debugging in first year Computer Science courses. RoboBUG was evaluated for playability, learning benefits and enjoyment. Our evaluation of RoboBUG showed that the game helps students to achieve learning outcomes, but has a non-statistically significant impact on enjoyment (positive and negative affect). In addition, the game seemed to be most effective at aiding students who were not initially skilled at debugging. The RoboBUG game and source code are available online at https://github.com/sqrlab/robobug.

The short length of the game, chosen to fit within the experiment time frame, meant that we had to limit the amount of content we could include. It is possible that different effects on learning benefits and enjoyment might be observed with an extended play session, or with added new content. RoboBUG is designed to make the addition of levels accessible for instructors, but introducing new game mechanics requires further work from the game developers.

Since the completion of our study, we have continued to improve RoboBUG by updating the interface design elements, implementing a hint system to reduce frustration and enhancing the ability to extend RoboBUG with new levels. In addition, we have also developed a prequel game that will allow RoboBUG to be played by users who have limited programming experience [15]. Future areas of work include the addition of a points system for competitive play, adding a cooperative multi-player mode, and improving the replay-ability by using program mutation [2] to generate random bugs each time a level is played.

In addition to enhancing the RoboBUG game we are also focusing on additional evaluation. Specifically, we are in the process of conducting a longitudinal study of RoboBUG in a first year programming course at UOIT and believe this larger in-class study will complement the information from our controlled experiments.

6 ACKNOWLEDGMENTS

This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). We thank the reviewers for their thoughtful comments and suggestions.

REFERENCES

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An analysis of patterns of debugging among novice computer science students. In *Proc. of 10th SIGCSE Conf. on Innovation and Technology in Comp. Sci. Education (ITICSE '05)*. 84–88.

[2] James H Andrews, Lionel C Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proc. of International Conference on Software Engineering 2005 (ICSE '05)*. 402–411.

[3] Elizabeth Carter and G.D. Blank. 2014. Debugging Tutor: preliminary evaluation. *J. of Computing Sciences in Colleges* (2014), 58–64.

[4] Mei-Wen Chen, Cheng-Chih Wu, and Yu-Tzu Lin. 2013. Novices' debugging behaviors in VB programming. In *Proc. of Learning and Teaching in Comp. and Eng. (LaTiCE 2013)*. 25–30.

[5] Du Chuntao. 2009. Empirical study on college students' debugging abilities in computer programming. In *Proc. of 1st Int. Conf. on Info. Sci. and Eng. (ICISE 2009)*. 3319–3322.

[6] Heather Desurvire, Martin Caplan, and Jozsef A. Toth. 2004. Using heuristics to evaluate the playability of games. In *Proc. of 2004 Conference on Human Factors in Computing Systems (CHI '04) - Extended Abstracts*. 1509–1512.

[7] Sue Fitzgerald, Renée McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander. 2010. Debugging from the student perspective. *IEEE Trans. on Education* 53, 3 (2010), 390–396.

[8] R. Garris, R. Ahlers, and J. E. Driskell. 2002. Games, motivation, and learning: a research and practice model. *Simulation & Gaming* 33, 4 (2002), 441–467.

[9] Morgan Hall, Keri Laughter, and Jessica Brown. 2012. An empirical study of programming bugs in CS1, CS2, and CS3 homework submissions. *J. of Comp. Sci. in Colleges* 28, 2 (2012), 87–94.

[10] Roslina Ibrahim, Rasimah CM Yusoff, Hasiah M Omar, and Azizah. Jaafar. 2010. Students perceptions of using educational games to learn introductory programming. *Comp. and Info. Sci.* 4, 1 (2010), 205–216.

[11] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. 2012. A serious game for developing computational thinking and learning introductory computer programming. *Procedia - Social and Behavioral Sciences* 47 (2012), 1991–1999.

[12] Fengfeng Ke. 2009. A qualitative meta-analysis of computer games as learning tools. *Handbook of Research on Effective Electronic Gaming in Education* (2009).

[13] Michael J Lee and Andrew J Ko. 2014. A demonstration of gidget, a debugging game for computing education. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*. IEEE, 211–212.

[14] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Computer Science Education* 18, 2 (2008), 67–92.

[15] Michael A Miljanovic and Jeremy S Bradbury. 2016. Robot ON!: a serious game for improving programming comprehension. In *Proc. of the 5th International Workshop on Games and Software Engineering*. ACM, 33–36.

[16] Mathieu Muratet, Patrice Torguet, Jean-Pierre Jessel, and Fabienne Viallet. 2009. Towards a serious game to help students learn computer programming. *Int. J. of Comp. Games Tech.*, 1–12.

[17] Jackie O'Kelly and J. Paul Gibson. 2006. RoboCode & problem-based learning : A non-prescriptive approach to teaching programming. In *Proc. of 11th SIGCSE Conf. on Innovation and Technology in Comp. Sci. Education (ITICSE '06)*. 217–221.

[18] Valerie J Shute. 2011. Stealth assessment in computer-based games to support learning. In *Computer Games and Instruction*, Vol. 55. 503–524.

[19] A.C. Siang. 2003. Theories of learning: a computer game perspective. In *Proc. of 5th Int. Symp. on Multimedia Soft. Eng. (ISMSE 2003)*. 239–245.

[20] Beth Simon, Sue Fitzgerald, Renée McCauley, Susan Haller, John Hamer, Brian Hanks, Michael T Helmick, Jan Erik Moström, Judy Sheard, and Lynda Thomas. 2007. Debugging assistance for novices. In *Working Group Reports on Innovation and Tech. in Comp. Sci. Education (ITiCSE-WGR '07)*. 137–151.

[21] Nikolai Tillmann and Judith Bishop. 2014. Code Hunt: searching for secret code for fun. In *Proc. of 7th Int. Work. on Search-Based Soft. Testing (SBST 2014)*. 23–26.

[22] David Watson, Lee a. Clark, and Auke Tellegen. 1988. Development and validation of brief measures of positive and negative affect: The PANAS scales. *J. of Personality and Social Psychology* 54, 6 (1988), 1063–1070.

[23] Wai-Tak Wong and Yu-Min Chou. 2007. An interactive Bomberman game-based teaching/learning tool for introductory C programming. In *Proc. of 2nd Int. Conf. on Edutainment*. 433–444.

[24] Andreas Zeller. 2009. *Why programs fail: a guide to systematic debugging*. Elsevier.