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

Michael A. Miljanovic
Ontario Tech University
Oshawa, Ontario, Canada
michael.miljanovic@ontariotechu.ca

Jeremy S. Bradbury
Ontario Tech University
Oshawa, Ontario, Canada
jeremy.bradbury@ontariotechu.ca

ABSTRACT

Serious games have become a popular alternative learning tool for computer programming education. Research has shown that serious games provide benefits including the development of problem solving skills and increased engagement in the learning process. Despite the benefits, a major challenge of developing serious games is their ability to accommodate students with different educational backgrounds and levels of competency. Learners with a high-level of competence may find a serious games to be too easy or boring, while learners with low-level competence may be frequently frustrated or find it difficult to progress through the game. One solution to this challenge is to use automated adaptation that can alter game content and adjust game tasks to a level appropriate for the learner. The use of adaptation has been successfully utilized in educational domains outside of Software Engineering, but has not been applied to serious programming games. This paper presents GidgetML, an adaptive version of the Gidget programming game, that uses machine learning to modify game tasks based on assessing and predicting learners' competencies. To assess the benefits of adaptation, we have conducted a study involving 100 students in a first-year university programming course. Our study compared the use of Gidget (non-adaptive) with GidgetML (adaptive) and found that students who played Gidget during lab sessions varied significantly in their performance while this variance was significantly reduced for students who played GidgetML.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; • **Applied computing** → *Computer games*; • **Computing methodologies** → Machine learning.

KEYWORDS

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

ACM Reference Format:
Michael A. Miljanovic and Jeremy S. Bradbury. 2020. GidgetML: An Adaptive Serious Game for Enhancing First Year Programming Labs. In *Software Engineering Education and Training (ICSE-SEET'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377814.3381716

ICSE-SEET'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7124-7/20/05…$15.00
https://doi.org/10.1145/3377814.3381716

1 INTRODUCTION

Gonczi [5] 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." Although this varies from country to country, 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 when included as an activity in a structured course 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 [18]. Unfortunately, serious games have to deal with the challenge of player retention, especially when players come from a diverse set of backgrounds and skill levels. The concept of flow [3], 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 handle the challenge of dynamically accommodating players of different skill levels is the use of adaptation [6]. By creating a model of a player's experience, designers can use their game data to alter game content or change parameters, in order to better challenge skilled learners or reduce the difficulty for learners who are struggling. This is a form of stealth assessment [16], 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.

The wealth of existing games that help players to learn computer science conceptions provides a great opportunity to make adaptive modifications without having to make 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 [9]. 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. Thus, our first research question is:

- RQ1 - Does Gidget benefit from adaptation?

In order to evaluate the effectiveness of the adaptation, we need to look out how player performance changes over the course of gameplay. Specifically, we expect to see that variance among players should be more consistent, as the task content is set to a level of difficulty that is appropriate for them based on their competence. We created an adaptive version of Gidget called GidgetML, in the hopes of answering 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's results, and some discussion of our findings.

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. Vahldick et al. [17] categorized 40 serious programming games based on type, platform, educational content, and programming language. We previously [14] reviewed a different selection of 49 serious programming games as well as the research questions and instruments used to evaluate them. From both of these reviews, one of the games that stood out as potentially suitable for the current study was Gidget [9].

Gidget (see Figure 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 [8]. Gidget has been shown in several studies to improve learning [9] and engagement [11], 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 [4], which focuses on teaching geographical content and environmental issues, uses Confidence-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 designed 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 [1]. We chose an approach similar to that of the ELEKTRA project [7], 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 our previous work [13]. The methodology outlines 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 2). 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 are available in Section 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 [10], as well as the Game Experience Questionnaire [2]. The Game Experience Questionnaire has been analyzed for reliability, validity, and functionality using a combination of Rasch modeling [15] 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 intrusive physiological measures such as Galvanic Skin Response (GSR).

3 IMPLEMENTATION

3.1 Gidget

The original Gidget game[1] is a game 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 players must be careful to choose effective routes through

[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/.

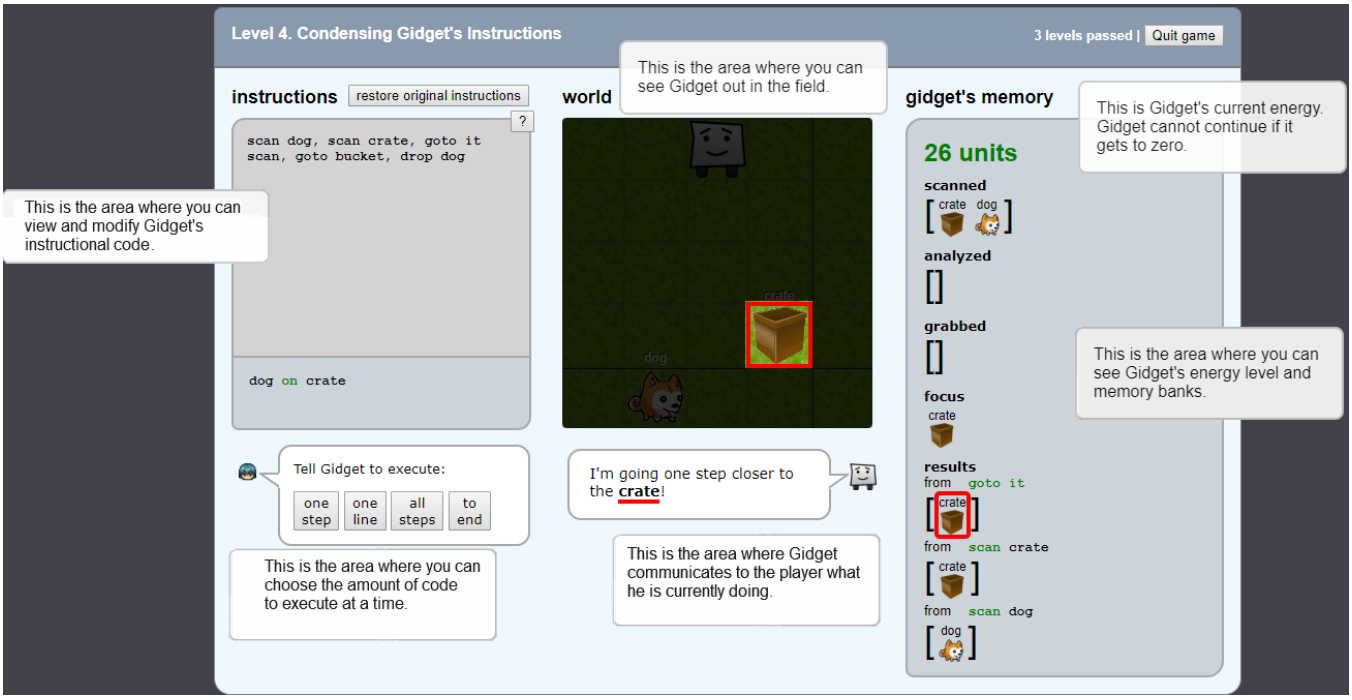


Figure 1: The Gidget game (with annotations), where learners help a damaged robot fix its programs by debugging its code [10].

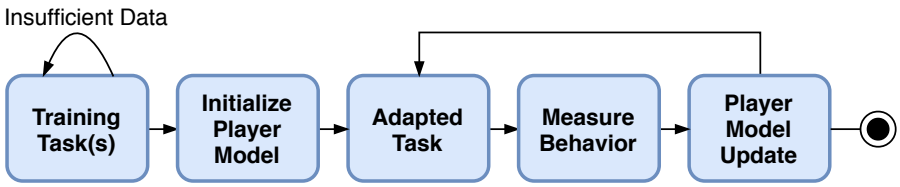


Figure 2: Adaptive gameplay sequence [13]

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 an older version that 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

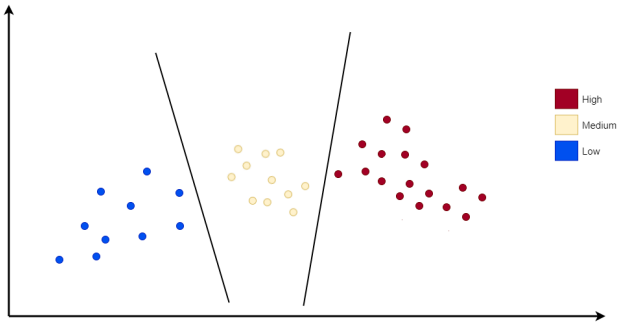


Figure 3: K-means categorization. Each player is categorized into one of three groups (high, medium, low) based on their gameplay data.

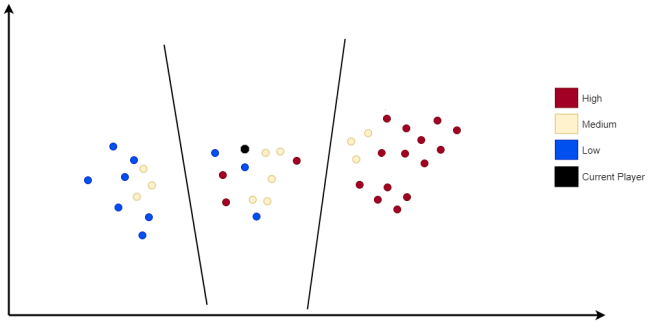


Figure 4: K-means categorization during gameplay.
Among those in the same category as the current player, the plurality were originally categorized as in the medium competence group, so the current player is also categorized as medium.

18th level, when they manage to close the leak in the 'goop' factory and rescue all of the animals in danger.

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 [12]. During gameplay, a k-means clustering is performed using the available data from the current 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 3 and 4.

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 to the project during the implementation 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 5.

4 METHODOLOGY

4.1 Adaptive Methodology

The adaptive methodology we devised [13] involves a four phase process (see Figure 6):

(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 [17] and our own review [14]. 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)

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

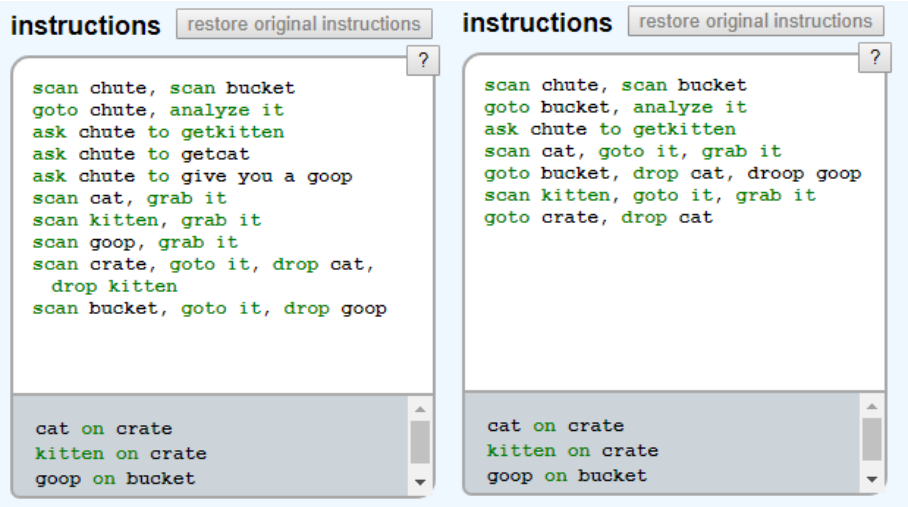


Figure 5: Adaptivity in the Gidget game.

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".

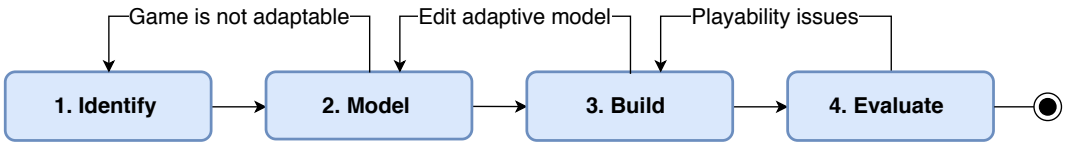


Figure 6: An overview of a methodology for making adaptive serious games [13]

"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." [13]

- Energy Units (number of steps permissible before failure)
- Order (levels were numbered 1 to 18)

In their survey, we [14] noted that debugging is the primary concept covered in Gidget. 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 to compare it with the original version is outlined in the following section.

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, before students began to complete any of their lab assignments. 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 [10]. Participants were asked for their age, gender, and how much past experience they had with programming. We added the GEQ [2] to this questionnaire to see if we would find any significant affective differences between the Gidget and GidgetML groups. 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 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.

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 participant's responses to our questionnaires, searching for the words 'hard', 'difficult', 'challenge', and 'frustrate', and found the following results:

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."*

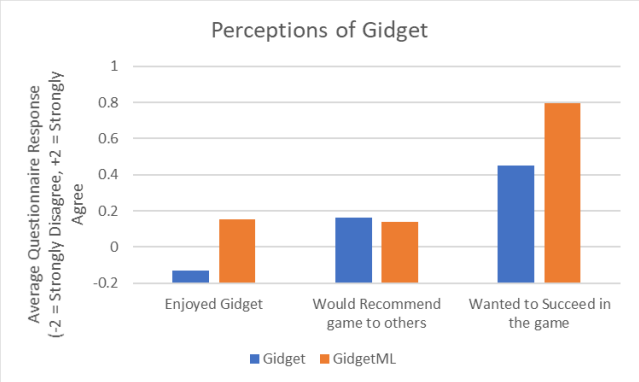


Figure 7: Perceptions of Gidget

- *"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."*
- *"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 7. 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 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 8. Responses were coded as 1 for positive responses, 0 for

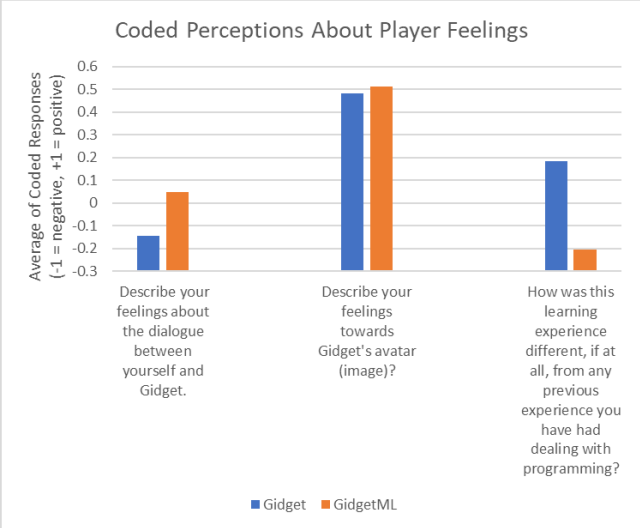


Figure 8: Coded perceptions of Gidget

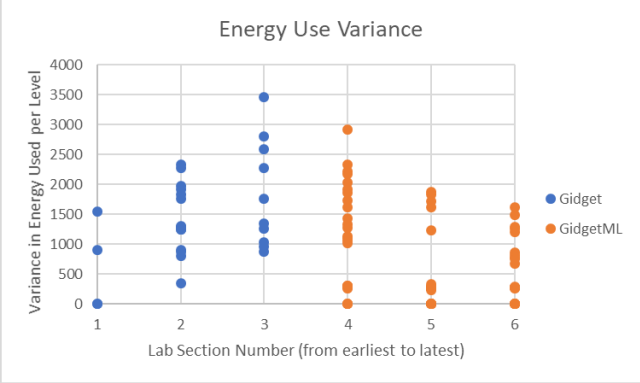


Figure 9: Variance of Energy used in Solutions

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 levels is shown in Figures 9 and 10. Figure 9 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=101.382, std=103.124)

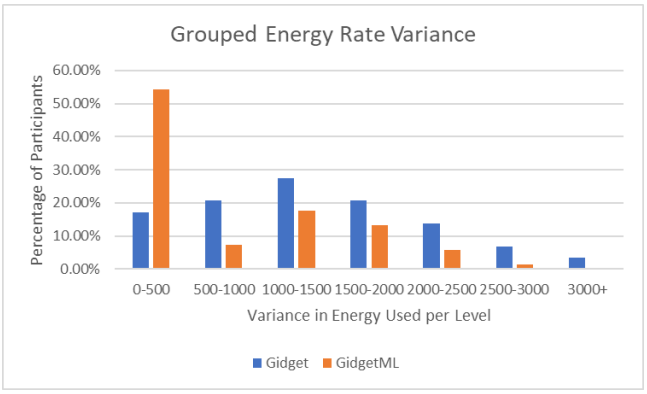


Figure 10: Grouped Variance of Energy used in Solutions

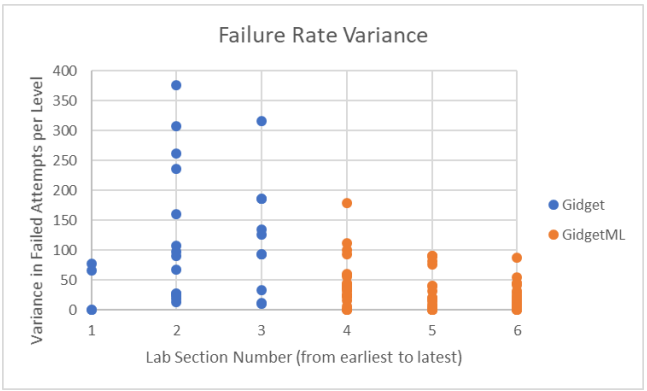


Figure 11: Variance of failures per level

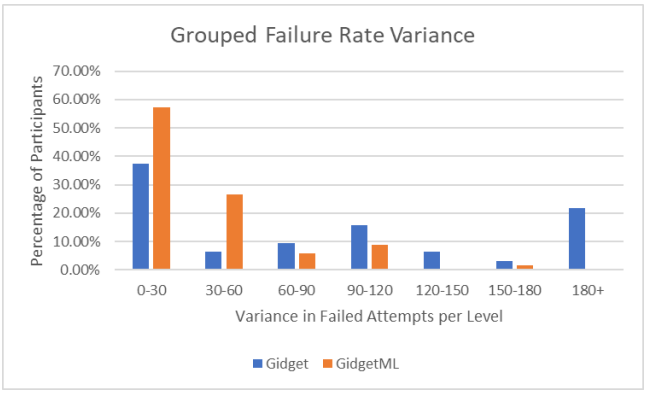
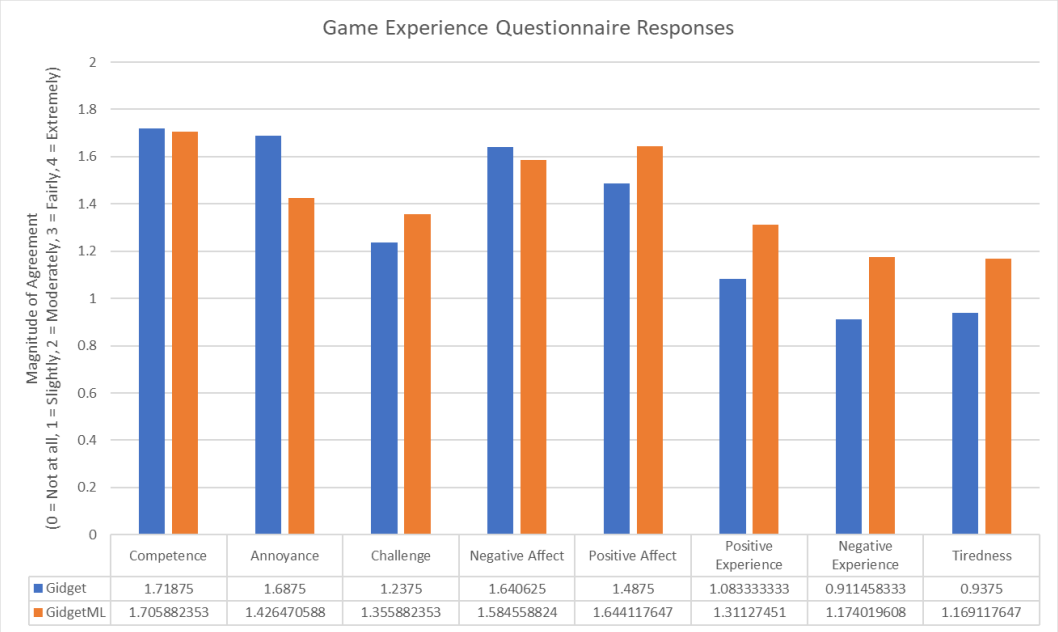


Figure 12: Grouped variance of failures per level

than the GidgetML group of labs 4-6 (mean=32.777, std=34.579). Figure 10 illustrates the difference between the two groups. The difference between groups was highly statistically significant (t=3.842, p=.000108).

Student data on the variance of number of failures per level is shown in Figures 11 and 12. Figure 11 shows how each lab differs on the variance of energy used in their program solutions. The



	Competence	Annoyance	Challenge	Negative Affect	Positive Affect	Positive Experience	Negative Experience	Tiredness
Gidget	1.71875	1.6875	1.2375	1.640625	1.4875	1.083333333	0.911458333	0.9375
GidgetML	1.705882353	1.426470588	1.355882353	1.584558824	1.644117647	1.31127451	1.174019608	1.169117647

Figure 13: Responses to Gaming Experience Questionnaire (GEQ) [2]

Gidget group, from labs 1-3, had a significantly higher average variance (mean=1411.171, std=827.187) than the GidgetML group (mean=765.557, std=763.000) of labs 4-6. Figure 12 illustrates the difference between the two groups. The difference between groups was extremely statistically significant (t=4.949, p<.00001).

Figure 13 shows the responses of students to the GEQ. On average, students scored low 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.

6 DISCUSSION

Our first research question was "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 9 and 11, 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 students differed greatly from level to level, suggesting that Gidget could benefit from adaptation.

Our second research question was "Is GidgetML effective at adapting to a learner's level of competency?" Using our methodology from previous work [13], 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 10 and 12, 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 9 and 11, 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.

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 [13], 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.

8 CONCLUSION & FUTURE WORK

In our study, 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.

ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number 2018-06588].

Special thanks to Amy Ko and Michael Lee for their work on the original non-adaptive version of Gidget.

REFERENCES

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

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

[3] Mihaly Csikszentmihalyi. 1997. *Finding flow: The psychology of engagement with everyday life*. Basic Books.

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

[5] Andrew Gonczi. 1999. Competency-based learning. *Understanding learning at work* (1999), 180–195.

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

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

[8] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. 2014. 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)*. IEEE, 57–64.

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

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

[11] Michael J. Lee, Amy J. Ko, and Irwin Kwan. 2013. 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)*. ACM, New York, NY, USA, 153–160. https://doi.org/10.1145/2493394.2493410

[12] Dan Li, Jitender Deogun, William Spaulding, and Bill Shuart. 2004. 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*. Springer, 573–579.

[13] Michael A Miljanovic and Jeremy S Bradbury. 2018. Making Serious Programming Games Adaptive. In *Proceedings of the Joint International Conference on Serious Games*. Springer, 253–259.

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

[15] Georg Rasch. 1960. Studies in mathematical psychology: I. Probabilistic models for some intelligence and attainment tests. (1960).

[16] Valerie Shute, Fengfeng Ke, and Lubin Wang. 2017. Assessment and adaptation in games. In *Instructional techniques to facilitate learning and motivation of serious games*. Springer, 59–78.

[17] Adilson Vahldick, António José Mendes, and Maria José Marcelino. 2014. A review of games designed to improve introductory computer programming competencies. In *2014 IEEE Frontiers in Education Conference (FIE) proceedings*. IEEE.

[18] D Zhao, AE Chis, GM Muntean, and CH Muntean. 2018. 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*.