ASIAN JOURNAL OF THE SCHOLARSHIP OF TEACHING AND LEARNING VOLUME 9 ■ NUMBER 1 ■ MAY 2019

ARTICLE

University Students' Perspectives On The Impact Of Games In Preparing For A Programming Course

Oka KURNIAWAN, CHEUNG Ngai-Man, NG Geok See

Information Systems Technology & Design,
Singapore University of Technology & Design (SUTD)

Correspondence:

Name: Dr. Oka KURNIAWAN
Address: Singapore University of Technology (SUTD), 8 Somapah Road, Singapore 487372
Email: oka_kurniawan@sutd.edu.sg

Recommended Citation:

Kurniawan, O., Cheung N. M., & Ng G. S. (2019). University students' perspectives on the impact of games in preparing for a programming course. *Asian Journal of the Scholarship of Teaching and Learning, 9*(1). 5-29.
https://doi.org/10.24112/ajsotl.93104

ABSTRACT

Programming in computing is often considered difficult and challenging by many students. This article discusses students' perspectives on the impact of board games and online games in helping them prepare for their programming course. We compare two kinds of games: a modified board game called "Robot Turtles" and an online game called CodeCombat. The results of our study show that students prefer online games in comparison to board games. Moreover, they were also able to learn computing concepts through these games. One surprising observation from the study was that students from the class which used these games performed just as well in the mid-term programming assessment as students that were taught computing concepts using traditional teaching methods. This indicated that games could be used to replace traditional lecture methods in preparing students for an undergraduate programming course. Students who attended the class that used games experienced higher levels of confidence in writing and debugging a computer programme as compared to their peers who attended the class that used only traditional lectures.

INTRODUCTION[1]

According to a study by Bransford, Brown, and Cocking (2000), students' learning tend to be affected by their prior knowledge. They assert that possessing misconceptions and the "wrong" prior knowledge can hinder the students' ability to grasp a concept correctly. In addition, lacking prior knowledge can make it difficult for students to understand new concepts. This means that people do not learn new things from scratch. If anything, when they encounter a new piece of information, they will try to connect and associate it with what they have learnt before. When they do not have enough prior knowledge, they will not be able to grasp or understand the lessons well (Vygotsky & Cole, 1978). Hence, an educator's role is to scaffold students' learning so that they have the needed prior knowledge to fully understand the concepts being taught during the lessons.

Scaffolding students' learning to ensure that they have the needed prior knowledge is one of the challenges we face as educators at the Singapore University of Technology and Design (SUTD), particularly for the course "Digital World". This is an introductory course in programming and computational thinking which uses the Python programming language. It is compulsory for all students at SUTD, which they have to take in Term 3 of the Freshmore[2] portion of the university's undergraduate curriculum. One of the main challenges of teaching this course is managing a class comprising students from a wide spectrum of backgrounds, levels of knowledge, and learning interests. The majority of students taking "Digital World" have no previous exposure to programming concepts or practice, and their learning curve for this course would be very steep. The question then is, how do we equip students taking "Digital World" with preliminary programming concepts in order to scaffold their learning for this course?

Another issue that we face in teaching this course is motivating students to learn programming. Since it is compulsory, all students have to take "Digital World" whether they like it or not. According to statistics collected from past iterations of the course, less than one-third of the student cohort go on to do a computing degree. Even for students who ultimately choose to do a computing major, their decisions may be motivated by other extrinsic factors (Jenkins, [2001]). Students intending to major in disciplines such as Engineering may still be able to see the benefits of learning programming. However, we observed that among students intending to study disciplines such as Architecture, the level of interest and motivation to learn programming tends to be low, even though SUTD's Architecture degree programme focuses a lot on computation. Recent studies have attempted to analyse the effectiveness of motivating students in computer science courses, especially non-computer science majors (Forte

& Guzdial, 2005; Kurkovsky, 2006). In order to motivate the latter group of students, Forte and Guzdial (2005) emphasised the importance of recognising their levels of interest in the subject and tailoring the curriculum accordingly. This approach is seconded in the work by Kurkovksy (2006). Meanwhile, there is evidence in the literature indicating that many students view programming courses as being difficult and demanding (Bennedsen & Caspersen, 2007; Teague, 2011); such perceptions could also be contributing factors for students deciding not to continue to learn programming. Therefore, another question that we raise in this study is: how should educators motivate these students to learn programming?

Besides tailoring the course to the interests and learning needs of a diverse student cohort, another possible way to engage and motivate them is by applying active learning strategies in the classroom (Cordes & Parrish, 2002). SUTD's undergraduate programme promotes active learning in all its courses, and this effort to increase student engagement led the authors of this paper to investigate games as a way to learn programming.

Games have been used by many educators to motivate students to learn programming in different ways. Feldgen and Clua (2004) showed that introducing problem-solving assignments which incorporate game-like elements made programming concepts more relevant and interesting, which led to an increase in student participation. Similarly, Jiau, Chen, and Ssu (2009) discussed the learning benefits of game-based assignments in computer programming courses, particularly how the competitive element enhanced students' motivation to complete the learning activities. In addition, Combéfis, Beresnevičius, and Dagienė (2016), as well as Vahldick, Mendes, and Marcelino (2014) provided comprehensive reviews of the various games for learning introductory programming available on the market, pointing to the rising popularity of using games in learning programming.

It has been shown that games can also be a motivation tool for students (Cliburn, 2006). In Cliburn's (2006) study, students were given a choice between completing a games-based or a non-games programming project. In almost 80% of the assignments submitted, students took the games-based option. Interestingly, it was found that the average grades of students who chose the non-games option were higher than those who chose the games-based option. Nevertheless, the satisfaction survey results in Cliburn's (2006) study indicated that games provided extra motivation for the students.

Recently, there has been a trend of teaching students programming without using computers, as shown in online examples such as the "CS Unplugged" website and the "Robot Turtles" board games, which teaches preschoolers basic coding concepts (Robot Turtles, n.d.). Certain games are suitable to teach programming due to its discrete nature. In fact, the design of some of the computer games used to teach programming were inspired by board games because the latter tend to be inherently discrete (Bezáková, Heliotis, & Strout, 2013). At the same time, these board games were designed to be visual, allow for social interaction and users receive immediate feedback from other players (Eagle & Barnes, 2009). Meanwhile, the interaction within computer games tend to be just between the individual player and the computer system, and feedback is limited to perhaps the computer detecting some common mistake committed by the individual player. Board games allow more interaction and exchange of feedback among the users, and it has been shown that such games can help students develop a deeper and more robust understanding of computing concepts. Battistella and Gresse von Wangenheim (2016) have conducted a systematic review of various games (digital and non-digital) used for teaching computing, as have Vahldick, Mendes, and Marcelino (2014).

This current study describes an effort that was done in 2016 just prior to the start of "Digital World". In this study, we present students' perceptions of using games-based activities to learn programming concepts. These were some questions we tried to answer:

1. To what extent does the use of games impact students' levels of confidence in learning programming as compared to a traditional lecture-based lesson?
2. What are students' perceptions of the effectiveness of using games in acquiring the needed programming skills as compared to lecture-based lessons?
3. Do differences between games-based and lecture-based lessons cause any significant impact on students' subsequent academic performance for the course "Digital World"?

We first compared their perceptions of acquiring certain skills for two different kinds of classes. One of the classes employed games-based learning activities, while the other class used the traditional lecture-based approach. We also compared the performance between these two classes in their mid-term test. We then focused on the differences between the two kinds of games-based activities used in the class that adopted the games-based approach. Thus far, we have not found any study which compares the effectiveness of using board games to computer games in learning programming. Therefore, we are also interested in the following two questions:

4. What are students' perceptions of the effectiveness of using board games as compared to computer games in helping to scaffold their learning of basic programming concepts?
5. From the students' perspectives, which kind of games (board games or computer games) are more effective in motivating them to learn programming?

In this study, we compare students' perspectives on the impact of using Robot Turtles and CodeCombat to teach a preparatory class on programming. We focussed on whether using a particular kind of game was effective in enabling them to learn basic programming concepts. These basic programming concepts, such as instruction code, programme counter, step-wise execution, and the role of programming, are important in helping students scaffold their learning of more advanced and complex programming concepts.

METHODOLOGY

Educational context

SUTD offers four undergraduate degrees, or Pillars, in Engineering and Architecture. Out of the four degrees, the Information Systems Technology and Design (ISTD) Pillar can be considered a combination of the traditional Computer Science and Computer Engineering degree programmes. The Engineering Product Design (EPD) and Engineering System Design (ESD) Pillars are two other undergraduate engineering degree programmes offered by SUTD. The last one is the Architecture and Sustainability Design (ASD) Pillar. All SUTD students have to take common courses in their first three terms. The course "Digital World", which introduces students to programming and computational thinking, is offered in the third term.

In terms of pre-tertiary exposure to programming, Singapore's secondary education curriculum does not include a compulsory programming course, and only a few junior colleges offer Computing as an A-level subject. Students in the polytechnics who major in engineering fare better, as they normally take one or two subjects in programming. However, the majority of SUTD's undergraduates come from the junior colleges, which means that the majority have little to no programming background at all. Moreover, the majority of students taking "Digital World" can be considered non-computing majors.

Intervention

Our intervention was in the form of a five-day preparatory workshop with three hours of lessons for each day. The purpose of this workshop was to teach basic computational thinking to students who have no programming background at all, and it is hoped that this workshop could equip these students with some basic programming knowledge in preparation for "Digital World". The workshop topics would include introductory computational thinking, some computing concepts, and the Python programming language. The workshop would also introduce students to the three basic structures of computer programmes, i.e. sequential, branch, and iteration.

Students were recruited during the December school holidays in 2015. An email was sent to all Freshmore students inviting them to participate in the workshop. It also highlighted the conditions of eligibility for the workshop. The first condition was that only students with little or no programming background could participate in the workshop. The second condition was that participating students had to attend all sessions. Out of a total cohort of 358 students who took "Digital World" that academic year, 90 students registered for the workshop. After some drop-outs, 53 students completed the workshop.

We divided the workshop into two classes of about the same size. Both classes were taught by two Undergraduate Teaching Assistants (UTAs). Even though both classes were taught by different teaching assistants, they covered the same topics.

Traditional Class

The first class was taught using slides, live demonstrations, and exercises, that is, activities and tools used in a typical lecture. The lesson started with introducing workshop participants to how a computer works and what programming is. The class was then introduced to the Python programming language which would enable them to write programming codes for the computer. The advantage of this class was that it gave students ample time to work on many exercises. In this class, students could jump directly into learning Python syntax and immediately do coding. Each day, the students were presented with various programming problems which they discussed and solved individually or as a group.

Games Class

In the second class, we started the lesson using board games to introduce computational thinking concepts and how the computer works. We then moved on to using an online game where students had to write programming code to move an avatar. We introduced proper Python syntax during the last few lessons. A survey was conducted at the end of the workshop.

Figure 1. The setup for the board game Robot Turtles. Players had to use cards to give the Turtle Master instructions to move their turtle closer to the gem.

The board game that students used in the games-based class was Robot Turtles (Robot Turtles, n.d.), as shown in Figure 1. Though the game is designed for young children, it has some additional rules that can be used for older children and adults. In the workshop, to make the game more challenging, we modified these additional rules and set a time limit for the players. They had to play all the cards during their turn before a Turtle Master could move their turtle. The rules had been modified such that players no longer needed to go through the initial level, which has no obstacles. Instead, they had to tackle all the blocks and obstacles once they started playing the game.

The class played the game on the first day of the workshop and the UTAs discussed some of the computational concepts found in the game. These included the instruction code and its implication on a computer's functionality, as well as the roles of a programmer. The UTAs also discussed the sequential nature of executing the code and the role of a programme counter.

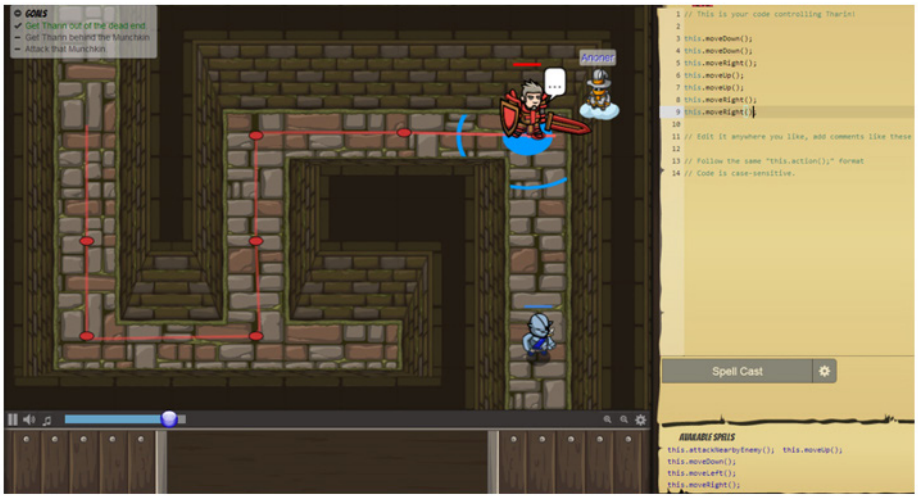

Figure 2. The interface of the online game CodeCombat. Players write Python codes to move the avatar in order to achieve specific tasks.

After introducing some basic computational thinking concepts, we asked students to play CodeCombat (Figure 2), which is an online strategy game (CodeCombat, n.d.). In CodeCombat, a player writes Python codes to move an avatar so that it performs various actions and tasks. Students wrote their Python codes in an online programming editor and saw how their codes affected the avatar when they ran the codes. CodeCombat is similar to Robot Turtles in that the Python codes are analogous to the instruction cards, and the avatar is analogous to the turtle.

Some of the notations in CodeCombat required students to have an understanding of object-oriented programming language, but this was not explained to them during the workshop. For example, students took for granted that they simply had to type the following code to move the avatar forward:

```
self.forward()
```

However, we did not explain the dot notation and object methods.

The basic levels of CodeCombat would introduce students to programming concepts such as sequential flow and conditional logics. Similar programming concepts can be found in Robot Turtles where the sequence of the cards affects the turtle's movements differently. Though there are no conditional statements in Robot Turtles, the game has a mechanism to introduce programming statements such as function calls and repetitions. The more advanced levels in CodeCombat also allow students to learn iteration in programming. After students have played with Robot Turtles and CodeCombat, they were introduced to the Python syntax. Students then had to practice using what they learnt of the Python programming language so far to solve common programming problems.

Measurement of intervention outcomes

At the beginning of the workshop, a survey on students' programming backgrounds was given to both classes. This was to help us get a sense of how proficient students were in their programming skills. The background survey asked students the following:

1. Which pillar/course will you choose after Term 3?
2. I have used the following programming language (tick all options that apply)
3. Rate your programming skills (Choose from "Zero", "Novice", "Intermediate", or "Advanced")
4. Number of lines of codes I have written before (Choose from "Zero", "1 to 10 lines", "10 to 50 lines", "50 to hundreds of lines", etc.)

At the end of the workshop, we administered another survey on students' perceptions. Students from both classes were asked to give scores ranging from 1 ("Strongly Disagree") to 5 ("Strongly Agree") for the following statements:

1. (Q1a) Given a problem, I can confidently identify the basic structures (sequential, iterative, branch) involved.
2. (Q1b) Given a problem, I can confidently state its input, output, and what needs to be computed.
3. (Q1c) Given a problem, I can confidently write its Python code.
4. (Q1d) Given an error in the code, I can confidently debug and solve the problem.
5. (Q2a) Learning programming in this workshop helps me to think algorithmically (i.e. step-by-step thinking approaching a solution).
6. (Q2b) Learning programming in this workshop helps me in my problem-solving skills.
7. (Q2c) I am more familiar with Python syntax after the workshop.

The class that used games had additional games-related questions:

1. (Q3a, Q4a) Playing X helps me to understand about what an instruction code is.
2. (Q3b, Q4b) Playing X helps me to understand about what the computer does in executing instructions.
3. (Q3c, Q4c) Playing X helps me to understand about what programming a computer is.
4. (Q3d, Q4d) Playing X helps to introduce algorithmic thinking (i.e. step-by-step thinking approaching a solution).

where X refers to:

- Robot Turtles (board game) for Q3
- CodeCombat (online game) for Q4

Lastly, we asked students if the games motivated them to learn computing:

1. (Q5a) Learning computing using a board game (like Robot Turtle) motivates me.
2. (Q5b) Learning computing using an online game (like CodeCombat) motivates me.

RESULTS

First, we present the survey results on workshop participants' backgrounds and their levels of familiarity with programming. We will then show the impact of the workshop and the differences between the traditional class and the class that used games. As mentioned earlier in the sub-section "Educational Context", there are four Pillars or degree programmes in SUTD, and all students have to choose their Pillar by the end of Term 3. Figure 3 shows the choice of Pillars of the workshop participants.

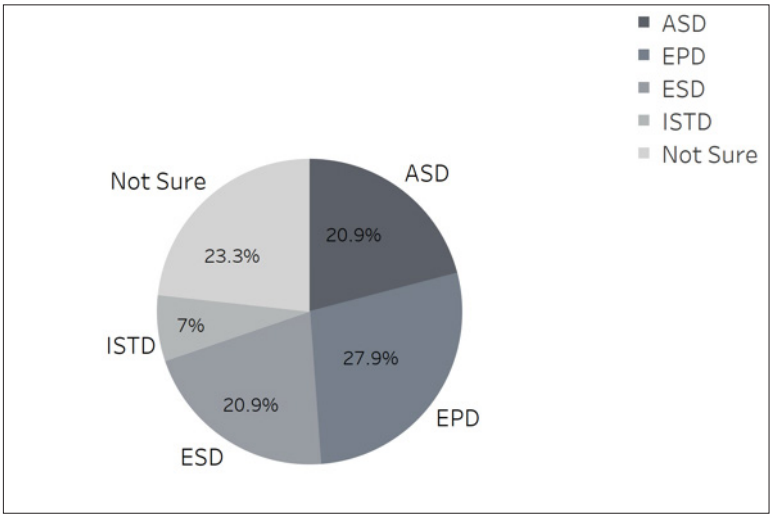

Figure 3. Survey results indicating the choice of Pillar or degree programme of workshop participants. The four Pillars offered at SUTD are: Architecture and Sustainability Design (ASD), Engineering Product Design (EPD), Engineering System Design (ESD), and Information Systems Technology and Design (ISTD).

Interestingly, the survey results indicated that the majority (about 70%) of workshop participants chose to pursue a non-computing Pillar, i.e. ASD, EPD, or ESD. Only a few students (7%) indicated that they wanted to pursue a computing Pillar, i.e. ISTD. This survey result makes sense since those intending to take a computing degree would usually have some background and prior knowledge of programming. Furthermore, such students would not attend such a workshop since it is intended for those with little or no prior knowledge in programming.

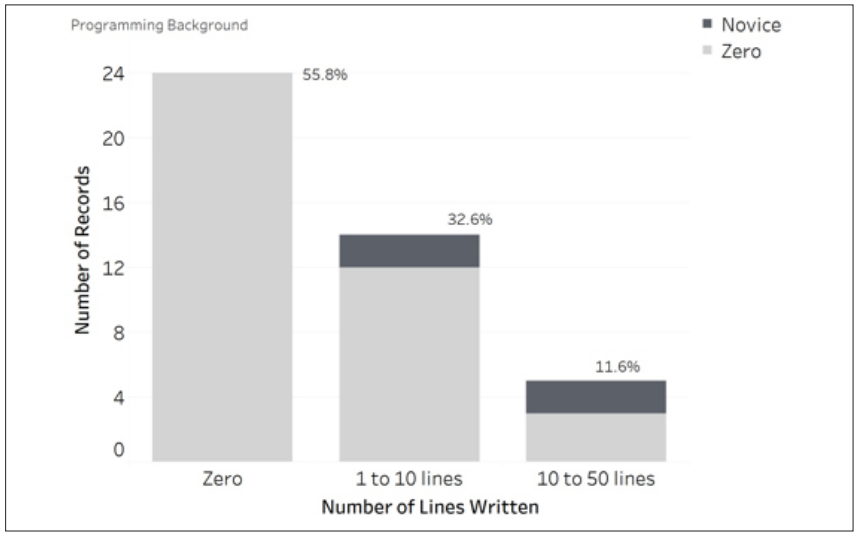

Figure 4. Survey results indicating the programming backgrounds of workshop participants. The horizontal axis shows the number of lines of code participants have written. On each bar, the different colour shows students' self-evaluation of their level of programming background. Most participants consider themselves programming novices with zero background and have never written any programming code. A few indicated they have written between 10 to 50 lines of codes.

Figure 4 indicates the workshop participants' programming backgrounds. The horizontal axis shows the number of lines of programming code participants had written prior to attending the course, and on each bar, the different colour indicates students' self-evaluation of their level of programming knowledge. Most students claimed that they were programming novices with zero programming background. The majority of participants also indicated that they had written "Zero" (55.8%) or only "1 to 10 lines" (32.6%) of programming code prior to attending the workshop, while a few (11.6%) claimed to have written between "10 to 50 lines" of code. None of the participants claimed to be experts in programming. This survey result shows that we managed to target students with little to no background in programming to attend the workshop.

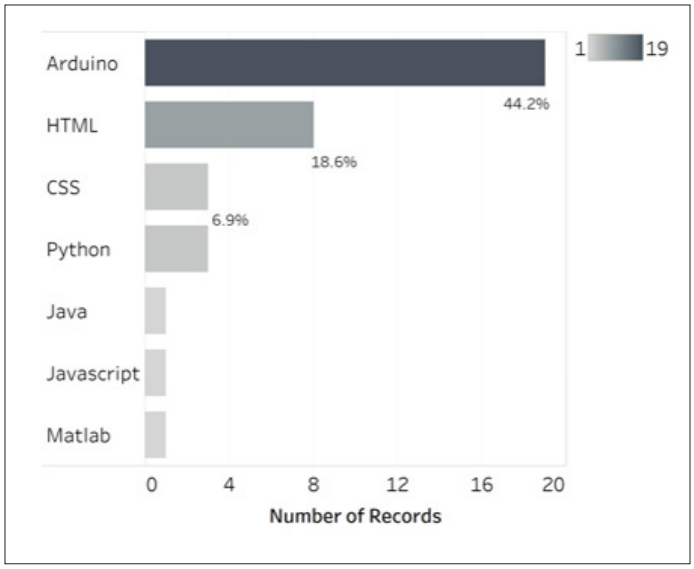

Figure 5. Survey results indicating workshop participants' level of familiarity with different programming languages prior to attending the workshop. Most participants have used Arduino (19 respondents) which is similar to C. The grey scale indicates the number of responses, which also corresponds to the width of the bar.

Figure 5 shows participants' level of familiarity with different programming languages. The survey results show that most students (about 44%) have used Arduino, which is similar in syntax to C. This result is also not surprising, as most participants would be familiar with Arduino, having been exposed to it when they attended the compulsory course "Introduction to Design" in Term 2. Some students (about 19%) also indicated that they have used HTML before, but it is more a markup language than a programming language. There were 18 students who indicated that they have used Arduino before, and it is possible that these could be the same group who indicated that they have written several lines of code (results reflected in Figure 4). Looking at both Figures 4 and 5, we can only deduce that some participants have written simple programming code using Arduino prior to attending the workshop, while the majority have not written any codes at all.

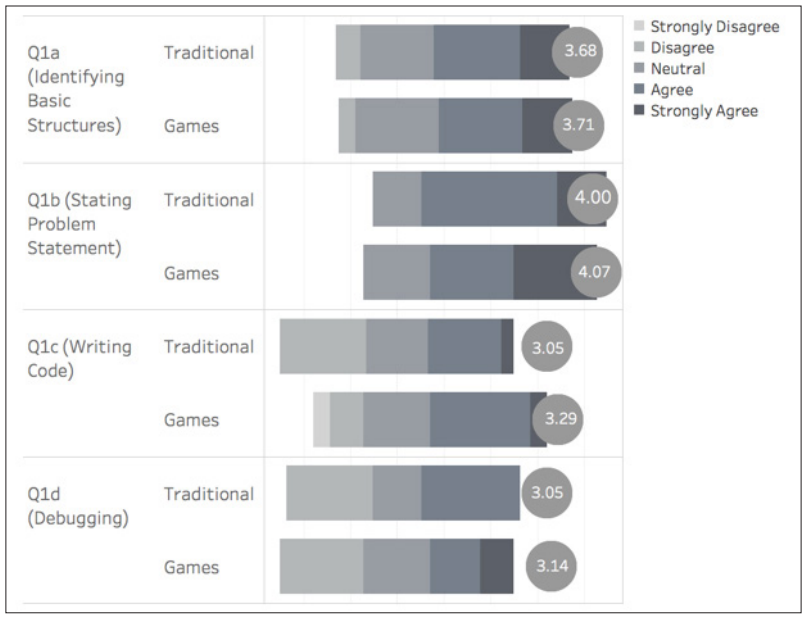

Figure 6. Students' perceptions of whether attending the workshop enhanced their levels of confidence to learn programming. The results indicated that students who attended the games-based class showed slightly higher levels of confidence compared to those that attended the lecture.

Figure 6 shows the workshop participants' responses on whether attending the preparatory workshop had impacted their levels of confidence to learn programming. It would address our first question on whether using games-based learning activities had any impact on their levels of confidence in learning programming. The results indicated that overall the games-based class showed slightly higher levels of confidence compared to the lecture-based class. Students who attended the games-based class showed slightly higher levels of confidence in debugging and identifying a computer programme's basic structure, as indicated in the higher scores for Statements Q1d (3.14 as compared to 3.05 for the traditional class) and Q1a (3.71 as compared to 3.68 for the traditional class) respectively. The former also showed significantly higher levels of confidence in writing Python codes, based on the scores for Statement Q1c (3.29 as compared to $^{3}.^{05}$ for the traditional class). It seems that the games-based pedagogy helped them identify the three basic structures as they kept repeating these tasks throughout the games. With regards to debugging, CodeCombat provided students with hints whenever they made mistakes while writing the code. This could be the reason behind students experiencing slightly higher levels of confidence when it came to debugging the code. We also found it interesting that the class that used the games-based

approach were more confident in writing Python code. One reason could be that while participating in CodeCombat, the game would immediately task students to write Python codes from the beginning. This was different from the lecture-based class, which began with an introduction to the syntax and getting students to discuss the problem and how to solve them. The writing of the code was shown as a live demo by the instructors. During the games-based class, however, CodeCombat forces students to immediately write Python code from the start. This immediate application and practice of what they learnt would explain their higher levels of confidence in writing Python code, as reflected in the scores for Statement Q1c.

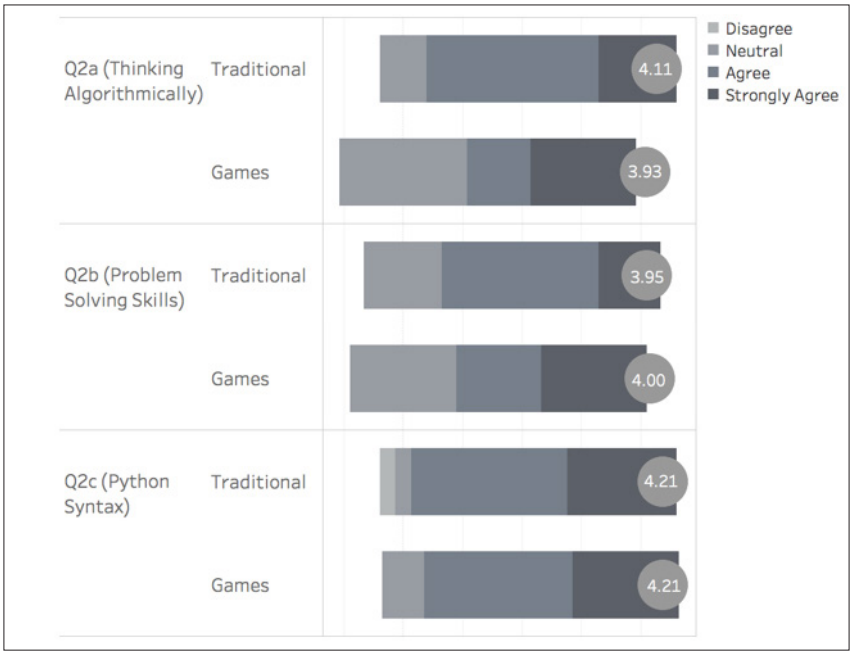

Figure 7. Students' perceptions of the effectiveness of games-based learning compared to traditional lecture-based learning to acquire programming skills. These include algorithmic thinking and problem solving, as well as Python syntax. The results for Q2a indicate that the traditional lecture-based class was more effective than games-based learning in developing algorithmic thinking to devise programming solutions.

In our second research question, we were interested to find out students' perceptions of the effectiveness of game-based learning compared to lecture-based learning in helping them acquire course-specific skills. These included algorithmic thinking, problem-solving skills, and mastering Python syntax. Figure 7 shows students' perceptions of their computational thinking skills and mastery of Python syntax. It was interesting to note that students from both classes gave about the same score for their problem-solving skills (Statement Q2b) and familiarity with Python syntax after attending the

workshop (Statement Q2c). On the other hand, the scores for Statement Q2a indicated that lecture-based learning was more effective than games-based learning in helping them develop algorithmic thinking to devise programming solutions (4.11 compared to 3.93 for the games-based class). This could be due to the different paradigms that games use. It was possible that the games they played did not offer a variety of problems to be solved, but rather repetitive tasks to complete. Most of these tasks were similar, and made increasingly complex as the levels increased. On the other hand, instructors from the traditional lecture-based class would provide ample examples of various problems for students to solve, which seemed to boost their levels of confidence in algorithmic thinking. However, it was also worth noting from the survey results that neither class gave a negative score (i.e. "Disagree" or "Strongly Disagree"). Thus, both approaches seemed to help students enhance their algorithmic thinking. It was noted, however, that being presented with a variety of problems (as done in the traditional class) seemed more effective in giving students' levels of confidence a boost.

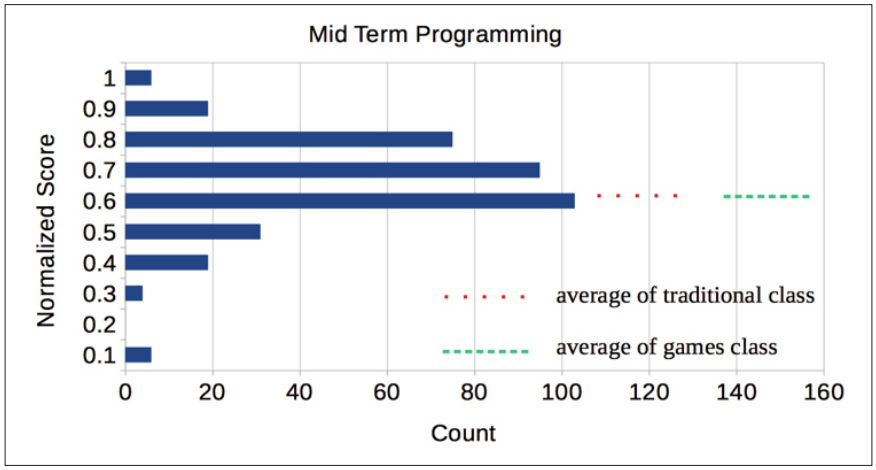

Figure 8. Distribution of mid-term results of the entire cohort that took "Digital World". The average scores of students who participated in the workshop are indicated in the two dotted lines. Their average scores were about the same as the rest of the cohort; there is no significant difference between the traditional lecture-based class and the class that used games.

To answer our third research question, Figure 8 shows the distribution of the mid-term results of the entire cohort that took "Digital World" in Term 3. For the workshop participants, we wanted to find out if the different approaches, one using games and the other one using a lecture, made an impact on their subsequent academic performance within the course. This mid-term result assessed students' programming skills and level of algorithmic thinking after students had completed "Digital World". It was interesting to note that the average scores of the workshop participants were around the same as the

average of the rest of the cohort. It was also noteworthy that the distribution of the mid-term scores seemed to skew towards the higher figures, with the majority scoring around 0.6 or higher. Another interesting point was that the average scores of students who participated in the workshop were the same for both classes, whether it was games-based or lecture-based. This means that a class that adopts a games-based approach can be as effective in providing students with needed prior knowledge as a lecture-based class.

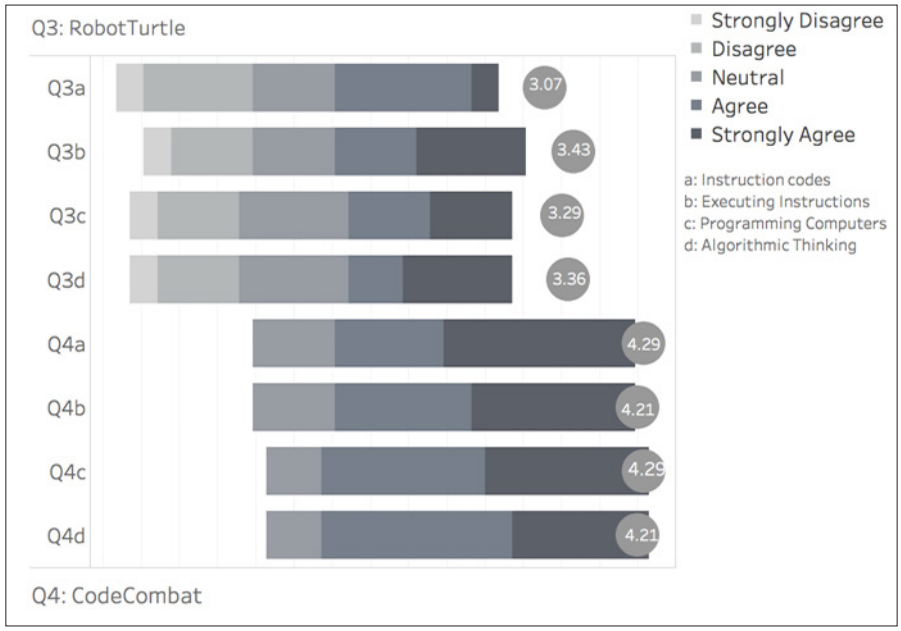

Figure 9. Students' perceptions of the effectiveness of using board games (e.g. Robot Turtles) as compared to computer games (e.g. CodeCombat) in helping them understand basic programming concepts. The results indicate that students found CodeCombat (Q4) more effective than Robot Turtles (Q3) in helping them understand computing and algorithmic thinking.

In our fourth research question, we wanted to find out students' perceptions of the effectiveness of board games, in this case Robot Turtles, as compared to computer games, such as CodeCombat, in helping them scaffold their learning of basic programming concepts. The comparison between Robot Turtles and CodeCombat in achieving its learning objectives is shown in Figure 9. The higher scores for Statements Q4a-Q4d indicate that students found CodeCombat more effective in helping them understand programming and algorithmic thinking, as compared to Robot Turtles. They found CodeCombat effective in helping them understand what computer instructions were, how the computer would execute these instructions, and what programming a computer means.

Students' seeming preference for CodeCombat over Robot Turtles could be due to a few reasons. For one thing, Robot Turtles employed cards while CodeCombat used Python instructions to move the objects. In this way, CodeCombat was similar to subsequent programming-related learning activities students would encounter, and this helped them to associate the concepts better. CodeCombat was also preferred in introducing them to algorithmic thinking as compared to Robot Turtles. This could be due to the limited ways the cards can be used and the limited challenges one can play in Robot Turtles. In Robot Turtles, while the players can change the maze and the number of obstacles and challenges would increase as the player advances further, the algorithmic thinking needed to solve the challenges remained the same. On the other hand, in CodeCombat the player would encounter more complex challenges as they advanced further, which would require them to apply algorithmic thinking.

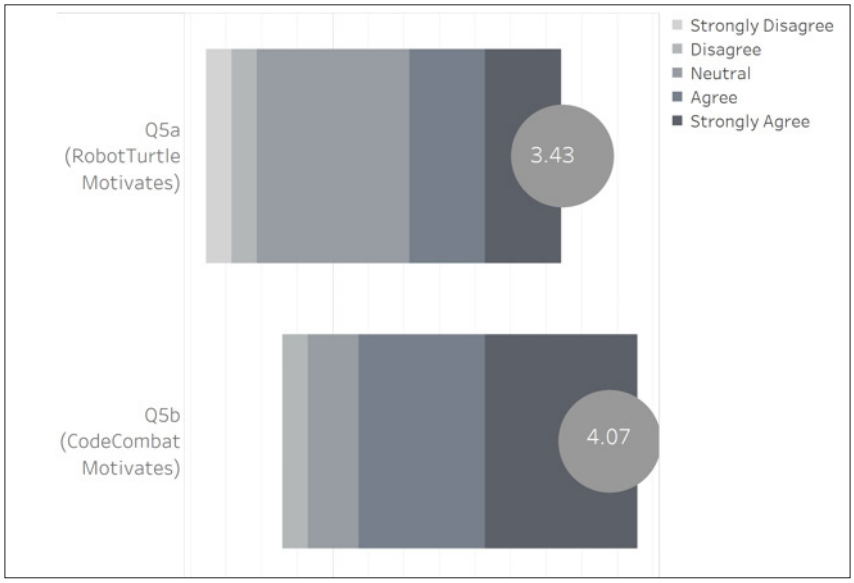

Figure 10. Students' perceptions of the effectiveness of using board games (e.g. Robot Turtles) as compared to computer games (e.g. CodeCombat) in motivating them to learn programming. The results indicate that students were more motivated to learn programming using online games such as CodeCombat (Q5b) than board games like Robot Turtles (Q5a).

In our last question, we wanted to find out students' perceptions of whether using board games, such as Robot Turtles, was effective in motivating them to learn programming, as compared to using computer games, such as CodeCombat. The results shown in Figure 10 indicate that students were more motivated by online games such as CodeCombat (Statement Q5b) than by board games such as Robot Turtles (Statement Q5a). The results were not surprising, considering that these students were more familiar with digital games than board games.

DISCUSSIONS AND LESSONS LEARNT

Observations made and future directions of the study

This was our first study to assess the impact of board games and online games in preparing students for the programming course "Digital World". This study was limited only to students' perspectives and was done as an action research to address a specific problem we observed, that is, how do we address the wide range of backgrounds and learning needs of students taking "Digital World". With this in mind, we will discuss the results obtained.

From the previous section, we noticed that the participants of this preparatory workshop were generally novices with little or no background in programming. Those who had written computer programming codes before have only used Arduino, but even then it was minimal. One thing that was obvious was that most participant had either never written any programming code or if they did, had very little experience doing so.

With this in mind, we conducted the workshop with the aim of helping these students learn some aspects of computational thinking before they attend "Digital World". They became familiar with algorithmic thinking and were given some overview of the problem-solving framework. In fact, their average scores for the mid-term assessment was about the same as the average scores of the entire cohort. However, they still had difficulty in the actual writing and testing of the programming code. This could be seen in Figure 6 where the levels of confidence in debugging and writing Python codes (Statements Q1c and Q1d) were generally lower than the other two tasks. This was to be expected since it was only a week-long workshop and mastering programming skills required time and practice.

In terms of academic performance, we could see that adopting the games-based approach to teach programming was comparable to the traditional lecture-based approach. A comparison of the average scores of the programming assessment and conceptual assessment, in which students had to answer conceptual questions, showed that neither gave a significant advantage. The pedagogy of using games, however, gave an advantage over the traditional lecture-based approach in motivating students and boosting their confidence levels. This requires further investigation. An important question to address is whether the usage of the games is sustained during the course "Digital World" and whether it increases their practice time. If it does, it may lead to better academic performance. This was not observed in this study because participants only used the games during the workshop before the formal start of "Digital World". This is something that we may explore in a future study.

Another result observed from this study was that students were more motivated to use online games compared to board games. This, however, is understandable since these students are born in a digital age and generally have more exposure to online games. At the same time, online games like CodeCombat are more interactive and visually engaging. Such games also provide opportunities for students to write code in an editor and receive immediate feedback. It is interesting to explore what are the aspects of online games that actually motivate students to learn. We are also aware that our study did not investigate the effects of social interaction and feedback from peers in engaging learners. This could be a very interesting topic to study in the future. At the same time, this result seems to support the findings by Battistella and Gresse von Wangenheim (2016), where out of the 107 games reviewed by those authors, the bulk of these were digital games, i.e. up to 68 digital games.

Limitations of the study

However, this study also had its limitations. For one thing, it only used the measurement of students' perceptions. This was true when assessing their prior knowledge as well as levels of engagement. We should have assessed prior knowledge in a more objective way, such as using a quiz to test their levels of knowledge of computing or programming concepts and abilities. However, administering such a test is not simple. If we wanted to test programming skills, this would involve the use of some programming language, and the participants may have little to no knowledge of the one particular programming language used in the test. On the other hand, if such a test was administered without using any programming language, the test would only assess the thinking process and not their actual programming skills.

In addition, levels of engagement could have been assessed not just from students' perceptions, but also through observations. In this study, we did not record any peer feedback or made any observations of the social interaction when students played the board game. Such data would have been interesting, since one advantage of using the board game over the computer game to teach programming would be to observe students' face-to-face interaction with their peers during the game. We also realised that observations may introduce some subjectivity if it was not done by the same person, and in similar circumstances. However, due to manpower limitations, we were not able to provide data from observations for this study and relied solely on students' perceptions.

Another limitation of our study was that we only used one particular game for the board game and one particular game for the computer game. Therefore, some of the students' perceptions may be affected by the features of that game rather than the general differentiation between board games and computer

games. It would be preferable to expose students to several board games and computer games, but there were not many options of such games that we could use which aligned to the workshop's learning objectives.

In this study, we chose natural settings in allowing students to use both types of games. This may limit any conclusions we could derive from survey statements Q3 and Q4, where students claimed that CodeCombat was more effective in helping them understand computing and algorithmic thinking than Robot Turtles. The reason was that students had gone through learning programming through Robot Turtles, which may have affected their understanding as they learnt to use CodeCombat.

The other limitation of our study may have come from the way the survey questions were phrased. For example, the question on whether such games motivated them would not fully capture the concept of motivation. From this question, we were not able to determine whether the motivation was intrinsic or extrinsic. In this study, what we seemed to capture was more their levels of engagement. In this case, it might have been better suited to use the word "engagement" instead.

Lastly, it would be interesting to compare workshop participants' scores with those who had no prior programming knowledge and did not attend the workshop. The current study, unfortunately, did not do so because there were logistical challenges in capturing such data. This might be something to consider for the future study.

CONCLUSION

This paper presents the results of our study in which we adopted a games-based approach to equip students with prior programming knowledge before they commenced the compulsory programming course "Digital World". Students indicated that games helped them to grasp some computational concepts and prepared them for "Digital World". The class that used games performed as well as the class that was taught using the traditional lecture style. In this way, games can be used to replace the traditional lecture-based method, at least in such preparatory lessons. At the same time, we tested board games and online games, and students found themselves to be more motivated to use online games like CodeCombat rather than board games like Robot Turtles. The games helped them to learn some computing concepts and introduced them to programming. These games, especially CodeCombat, helped to boost their levels of confidence and motivation to write computer codes. The authors may consider exploring these issues further in a future study.

ACKNOWLEDGEMENTS

The authors would like to thank the Education Department of SUTD for the funding provided to do this research. We would also like to thank Dr. Nachamma Sockalingam from SUTD's Learning Sciences Lab for her valuable feedback in improving this manuscript.

ENDNOTE

1. This research was conducted with approval from the SUTD-IRB (Reference Code: 16-125).
2. The Freshmore term is the first three terms of SUTD's undergraduate common curriculum, where students receive a solid foundation in mathematics and sciences. Details of SUTD's unique academic structure can be found at https://www.sutd.edu.sg/Education/Unique-Academic-Structure.

ABOUT THE CORRESPONDING AUTHOR

Oka KURNIAWAN is a Senior Lecturer at the Information Systems Technology and Design (ISTD) Pillar of the Singapore University of Technology and Design (SUTD). He teaches an introductory programming course and conducted workshops on computational thinking. His research interest is in the use of tangible objects and immersive technology to teach programming.

REFERENCES

Battistella, P. E., & Gresse von Wangenheim, C. (2016). Games for teaching computing in higher education - A systematic review. I*EEE Technology and Engineering Education (ITEE), 1*(3), 8–30. https://doi.org/10.1109/ICSE-SEET.2017.18

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2), 32. http://doi.org/10.1145/1272848.1272879

Bezáková, I., Heliotis, J. E., & Strout, S. P. (2013). Board game strategies in introductory computer science. In *SIGCSE '13 Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 17–22). New York. Retrieved from http://dl.acm.org/citation.cfm?id=2445210

Bransford, J., Brown, A. L., & Cocking, R. R. (2000). *How people learn: Brain, mind, experience, and school*. Washington, D.C.: National Academies Press. http://doi.org/10.17226/9853

Cliburn, D. C. (2006). The effectiveness of games as assignments in an introductory programming course. *The 36th Annual Frontiers in Education Conference*, 6–10. http://doi.org/10.1109/FIE.2006.322314

CodeCombat (n.d.). *CodeCombat: Learn how to code by playing a game*. Retrieved from https://codecombat.com/

Combéfis, S., Beresnevičius, G., & Dagienė, V. (2016). Learning programming through games and contests: Overview, characterisation and discussion. *Olympiads in Informatics, 10*(1), 39–60. http://doi.org/10.15388/ioi.2016.03

Cordes, D., & Parrish, A. (2002). Active learning in computer science: impacting student behavior. In *32nd Annual Frontiers in Education* (Vol. 1, p. T2A–1–T2A–5). IEEE. http://doi.org/10.1109/FIE.2002.1157919

Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. In *SIGCSE '09 Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Vol. 41, pp. 321–325). http://doi.org/10.1145/1539024.1508980

Feldgen, M., & Clua, O. (2004). Games as a motivation for freshman students to learn programming. In *34th Annual Frontiers in Education* (pp. 1079–1084). IEEE. http://doi.org/10.1109/FIE.2004.1408712

Forte, A., & Guzdial, M. (2005). Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, *48*(2), 248–253. http://doi.org/10.1109/TE.2004.842924

Jenkins, T. (2001). The motivation of students of programming. *ITiCSE*, 53–56. http://doi.org/10.1145/507758.377472

Jiau, H. C., Chen, J. C., & Ssu, K.-F. (2009). Enhancing self-motivation in learning programming using game-based simulation and metrics. *IEEE Transactions on Education, 52*(4), 555–562. http://doi.org/10.1109/TE.2008.2010983

Kurkovsky, S. (2006). Improving student motivation in a computing course for non-majors. In *Proceedings of the 2006 International Conference on Frontiers in Education: Computer Science & Computer Engineering* (pp. 82–88). Las Vegas, Nevada. Retrieved from https://www.semanticscholar.org/paper/Improving-Student-Motivation-in-a-Computing-Course-Kurkovsky/6893b3fa4f8ec4ab67ec077a3ba65f65d4c8b073.

Robot Turtles (n.d.). *Robot Turtles: The board game that teaches programming to kids*. Retrieved from http://www.robotturtles.com/.

Teague, D. (2011). *Pedagogy of Introductory Computer Programming: A People-First Approach* (Master's thesis). Retrieved from https://eprints.qut.edu.au/46255/.

Vahldick, A., Mendes, A. J., & Marcelino, M. J. (2014). A review of games designed to improve introductory computer programming competencies. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings* (pp. 1–7). IEEE. http://doi.org/10.1109/FIE.2014.7044114

Vygotsky, L. S., & Cole, M. (1978). *Mind in society : The development of higher psychological processes*. Cambridge, Mass.; London: Harvard University Press.

APPENDIX. SURVEY QUESTIONS

Survey questions can be obtained from the following websites:

- Background Survey: https://goo.gl/forms/z1y2ApRZklBL5biF2
- Post-Workshop Survey Traditional class: https://goo.gl/forms/rMAFchrPP8ANjt022
- Post-Workshop Survey Game class: https://goo.gl/forms/JZuQjoMaAcD7cfbr2 ■