Program Wars *v.*2.0 : Improving a Game-based Learning Approach for Teaching Fundamental Programming Concepts

Md. Hasan Tareque, Steven Deutekom, John Anvik, Maimoona Bashir *
[mdhasan.tareque,deutekom,john.anvik,maimoona.bashir]@uleth.ca
University of Lethbridge
Lethbridge, Alberta, CANADA

ABSTRACT

Game-based learning (GBL) provides an engaging way to introduce those with limited programming experience to fundamental programming concepts, Program Wars uses a GBL approach to teach fundamental programming concepts using cards that represent instructions, loops, variables and methods to create a programming language-independent program.

This paper introduces Program Wars *v.*2.0, which improves the prior implementation in several ways. These changes include the approach to teaching methods, introducing players to the concepts of searching and sorting algorithms, and revisions to the gameplay and UI to improve engagement.

A user study of Program Wars *v.*2.0 was conducted and shows that Program Wars *v.*2.0 is more effective than Program Wars *v.*1.0 in teaching the concepts of variables, loops and methods. Specifically, 60% of participants showed knowledge improvements of variables, 56% showed knowledge improvements for loops, and 44% showed knowledge improvements for methods. Qualitative results show that Program Wars 's game-based approach results in an engaging experience for learners.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**; • **Social and professional topics** → *Computational thinking*; *CS1*; *Computing literacy*.

KEYWORDS

Programming language education; Game-based Learning; Serious Game; Web application

ACM Reference Format:
Md. Hasan Tareque, Steven Deutekom, John Anvik, Maimoona Bashir . 2024. Program Wars *v.*2.0 : Improving a Game-based Learning Approach for Teaching Fundamental Programming Concepts. In *The 26th Western Canadian Conference on Computing Education (WCCCE '24), May 02–03, 2024, Kelowna, BC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3660650.3660671

*We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number RGPIN-2018-06004].

WCCCE '24, May 02–03, 2024, Kelowna, BC, Canada

ACM ISBN 979-8-4007-0997-5/24/05
https://doi.org/10.1145/3660650.3660671

1 INTRODUCTION

Experience-based education plays a vital role in the computer science (CS) curriculum, as learning through activities that combine social and educational aspects is very effective [20]. However, constraints such as time, proper tools and realistic environments introduce difficulties to such an approach.

Game-Based Learning (GBL) uses games as an educational tool to facilitate learning and engagement [23]. It is a learning approach that increases a student's interest in education [24] and their becoming more engaged [18, 19, 22] with better learning outcomes [12, 28, 32, 35] than through traditional methods. Finally, GBL can assist in bridging between theory and practice [14] and has been used to teach topics such as object-oriented programming [31], debugging [25] and software project management [16].

Most games and environments developed with the intention of teaching computer programming concepts do so by requiring the student to either learn an existing programming language (e.g. Java [2], Python [4, 6] or JavaScript [4, 5, 7]) or a language specific to the learning environment (e.g. Scratch [26] or Alice [17]). If the goal is to teach the fundamental concepts of computer programming, this requirement can result in a situation where the learner feels intimidated, confused or frustrated when their "program" does not work correctly. Also, as Bromwich et al. noted, many of these educational programming languages and environments are "often ambitious in what they are trying to teach beginning programmers. They even go as far as to try teaching concurrency, something with which even advanced programmers often have difficulty" [15].

Also, such learning games and environments commonly use either a puzzle-solving premise, such as navigating a maze, or a sandbox paradigm, where a learner can utilize the language in an open-ended manner. As puzzle-solving activities tend to favour a specific demographic [29], and the sandbox paradigm requires oversight by an outside influence (i.e. an instructor) to ensure learning progression [15], both approaches have significant drawbacks.

Introducing gameplay into a CS curriculum is not a new concept and is known to be an effective one [13]. Although learning through gameplay may have more sustainable effects [30], the main challenge is that of maintaining a balance between learning and engagement [27]. Finally, the gameplay needs to be presented in a structured way such that the user can relate the outcome to real-world programming.

To further explore the effectiveness of using GBL, we created Program Wars, a web-based card game for teaching programming language and cybersecurity concepts. Based on the results of a user study of Program Wars *v.*1.0 and informal user feedback, we created an updated version (Program Wars *v.*2.0) containing new cards and a revised user interface to make Program Wars a more

Table 1: Alignment of features with Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.

Concepts	Program Wars *v.*2.0	Curriculum Guidelines
Programming language basics	`Instruction` and `Repeat` cards	CMP.cf.8
Control Flow	Bonus goals	CMP.cf.1
Method/Function	`Method` card	CMP.cf.4
Algorithms	`Search` & `Sort` cards	CMP.cf.2

effective teaching tool. This paper focuses on the effectiveness of Program Wars *v.*2.0 in teaching the fundamental programming concepts; therefore, we omit the details regarding how Program Wars *v.*2.0 teaches cybersecurity concepts.[1] Specifically, we focus on how Program Wars *v.*2.0 improves the teaching of functional decomposition (i.e. methods), a key topic shown not to be effectively presented in the previous version [11]. Also, the concept of algorithms is introduced with the addition of cards that perform searching and sorting, two fundamental types of algorithms.

Table 1 provides a summary of the various game elements in Program Wars *v.*2.0 and how they align with the ACM's *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering* [21] in the knowledge area of `Computing Essentials (CMP)`.

The research question for this work is "*Do the refinements to the UI and gameplay of Program Wars* v.*2.0 improve a player's understanding of basic computer programming concepts?*" This research question is further divided into three sub-questions to evaluate the learning outcomes more precisely.

(1) Do the refinements to the UI and gameplay improve a player's knowledge of the *Variable* concept?
(2) Do the refinements to the UI and gameplay improve a player's knowledge of the *Loop* concept?
(3) Do the refinements to the UI and gameplay improve a player's knowledge of the *Method* concept?

A user study was conducted to answer our research questions. We found that after playing Program Wars *v.*2.0, most subjects' knowledge about variables, loops and methods improved and that this improvement was better than that found for Program Wars *v.*1.0.

This paper proceeds with a description of Program Wars *v.*2.0 before presenting the details and results of the user study. Finally, a review of the relevant related work regarding the use of GBL for teaching computer programming fundamentals is given before concluding the paper.

2 OVERVIEW OF PROGRAM WARS *V.*2.0

Program Wars is played over a series of rounds in which each player takes a turn building their program. Each player's hand consists of five cards, and a new card is drawn at the end of their turn.

There are two game modes: *Beginner* and *Standard* . To win, a player needs to create a program that reaches the game's goal number of points (100 for *Beginner* and 200 for *Standard*), where points represent the number of instructions that would be executed based on the cards played. Programs are built by creating stacks of `Instruction`, `Method`, `Repeat` and `Variable` cards (Section 2.1). Each stack of cards represents a portion of the player's program. The total score for a player is the sum of all their stack scores.

If a player reaches or exceeds the goal number of points, the game finishes at the end of the current round. In *Beginner* mode, the player's instruction score determines the winner, and players can tie. In *Standard* mode, bonus points are awarded for achieving specific objectives, such as the use of `Repeat` or `Variable` cards.

2.1 Game Cards

As in actual computer programs, instructions form the backbone of the Program Wars "program". The `Instruction` card represents a fixed number of instructions (1, 2, or 3) in a real-world program. These cards are the primary means of gaining points in the game.

Players can stack a `Repeat` card, which represents a loop, onto an `Instruction` card to increase the effective number of instructions in their program. These collections of cards are referred to as a "card stack" or simply "stack". There are three sizes of `Repeat` cards: 2, 3, and 4. The player can form a nested loop by placing a `Repeat` card on another `Repeat` card. Program Wars only allows nesting up to two levels to keep scores from growing too quickly, which would result in short games where players are not given an opportunity to practice and internalize the concepts.

In addition to the fixed-size `Repeat` cards, there is a variable `Repeat` card (called `Repeat-X`). This card does not by itself increase a stack's point value, acting like a `Repeat-1` card. However, the player can place a `Variable` card on a `Repeat-X` to increase its multiplicative power. `Variable` cards have values of 3 to 6, thereby creating equivalents of `Repeat-3` to `Repeat-6` cards. Like real-world variables, X can be changed by replacing a `Variable` card with another `Variable` card during the game.

The `Method` card acts as a proxy for the contents of the *Method Stack* area in the UI (see Section 2.2), with the player's total score being adjusted accordingly. If a new `Instruction` card is added to the *Method Stack* area, the player's score will be adjusted according to the number of `Method` cards in the *Main* area. As with `Instruction` cards, the player can use `Repeat` cards to increase the effect of a `Method` card.

The use of algorithms is an essential part of computer programming. Program Wars *v.*2.0 introduces players to two key categories of algorithms: sorting and searching. The first is the `Sort` card, which allows a player to rearrange the top five cards of the deck into whatever order they choose, thereby setting up which cards are drawn by each player for the next few turns. The `Search` card allows a player to search for a specific card within the top ten cards of the deck and put the card in their hand.

Figure 1 shows these cards as they appear in the game, with the exception of the `Search` and `Sort` cards as these two cards are not part of the player's constructed program. The bottom right side of the figure shows a set of `Instruction` cards (grey cards) being used to "create a method" and the player's "program" consisting of method calls (yellow `Method` cards). The bottom left side shows the other player's "program," which includes the use of

[1] These details can be found in [33]

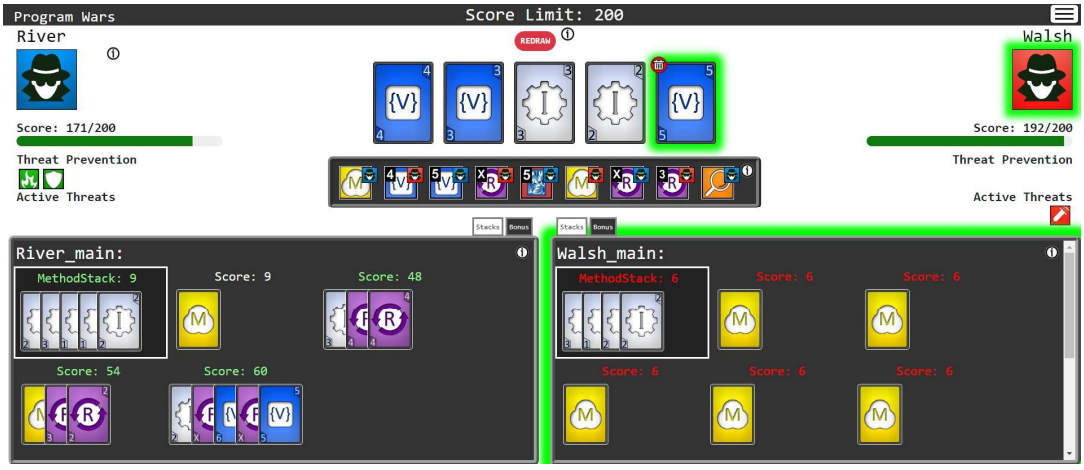

Figure 1: Program Wars *v.*2.0 near the end of River and Walsh's game.

`Repeat` cards (purple cards) and `Variable` cards (blue cards), as well as `Instruction` and `Method` cards.

2.2 User Interface

Figure 1 shows the user interface for playing Program Wars. The player information is shown in the top left and right corners. This information includes the player's progress towards the score goal. Also, there are indicators related to the cybersecurity aspects of Program Wars (`Threat Prevention` and `Active Threats`).[2].

In the top middle of the interface is the current player's hand. The green glow around their image (e.g. Walsh) indicates the current player. Above the player's hand is a button that lets the player redraw their hand. Under the player's hand is a recent history of cards played, ordered from left to right.

The lower half of the screen contains the play areas for the players. The `Program Editor` area (i.e. `Stacks` tab) is where cards are played. The `(PLAYER_NAME)_main` area represents the player's program. A section of this area, called the *Method Stack*, represents a method that can be called from the "main program". `Instruction` cards placed in the *Method Stack*, to a maximum sum of 9, are used as the value for the `Method` card for that player. Figure 1, shows that the players' "main program" comprises a mixture of `Instruction` cards and calls to their "method" which are repeated. Figure 2 shows what River's program might be if written in C.

The `Bonus` tab in the `Program Editor` shows the player's bonus points they have earned for meeting certain conditions during the game, including using `Repeat` cards, `Variable` cards, nested loops, and objectives related to the cybersecurity aspects of Program Wars. These conditions are presented as a series of *if-then* statements. The number of points for meeting a bonus objective is shown in green if the player has met it; otherwise, it is shown in red. This is how conditional statements are presented in Program Wars *v.*2.0.

```c
int main() {
    unsigned int instructions = 0;

    // Method Card (Total 9)
    instructions += method();

    // Instruction Stack (Total 48)
    for(int i=0; i < 4; i++) {
        for(int j=0; j < 4; j++) {
            instructions += 3;
        }
    }

    // Method Stack (Total 54)
    for(int i=0; i < 2; i++) {
        for(int j=0; j < 3; j++) {
            instructions += method();
        }
    }

    // Instruction Stack (Total 60)
    unsigned int var1 = 5;
    for(int i=0; i < var1; i++) {
        unsigned int var2 = 6;
        for(int j=0; j < var2; j++) {
            instructions += 2;
        }
    }

    // Outputs total instruction count of 171
    std::cout << "Total Instructions: " << instructions << std::endl;
}
```

```c
int method() {
    unsigned int total = 0;

    // Instruction-2
    total += 2;

    // Instruction-3
    total += 3;

    // Instruction-1
    total += 1;

    // Instruction-1
    total += 1;

    // Instruction-2
    total += 2;

    return total;
}
```

Figure 2: River's program in C++.

3 USER STUDY OF PROGRAM WARS *V.*2.0

To assess changes in knowledge of computer programming concepts from playing Program Wars *v.*2.0, we conducted a 'within-subjects' study. In other words, subjects were asked questions about computer programming concepts both before and after playing Program Wars *v.*2.0, and their responses were compared.

The user study was conducted online and without any supervision due to COVID protocols at the time of the study.[3] The study had three phases. First, the subject completed a pre-game survey consisting of demographic questions and multiple-choice questions to assess the subject's prior knowledge and experience with computer programming. Next, they were asked to play Program Wars *v.*2.0 until they felt they understood the game's concepts (a minimum of three times, a maximum of ten times). Finally, they completed a

[2]This paper does not discuss the cybersecurity aspects of Program Wars *v.*2.0, but details can be found in [33]

[3]The study procedure was reviewed and approved by the university's Human Subject Research Committee. Further specific details of the study procedure can be found in [33].

post-game survey with three types of questions: multiple-choice knowledge-testing questions similar to those of the pre-game survey, multiple-choice questions to assess if they could map the game mechanics to real-world scenarios, and feedback questions.

To participate, a subject must have been 18 years of age or older with little to no programming knowledge (i.e., non-CS majors). Subjects were recruited from within the university. Participation in the study was expected to take 60 ∼ 90 minutes, depending on the number of times the subject played the game. Unique study ids were used to link the pre-and post-game responses of a subject.

In the pre-game survey, subjects were asked two (2) demographic questions related to their age range and level of education. Also, subjects were asked an experience question related to their prior programming knowledge. In both the pre-and post-game surveys, the subjects were given computer programming questions where they were presented with some pseudocode and asked about the outcomes of the program. Each question was designed to test one or two knowledge areas of either variables, loops, or methods.

The subjects were asked questions in the post-game survey related to Program Wars *v.*2.0 gameplay. These questions were to assess if subjects could make connections between the game and real-world programming. For these questions, the pseudocode was used again, with different colour rectangles used to highlight different areas of the program. Subjects were asked to identify the appropriate concept (i.e. loop, variable or method).

3.1 Results

Twenty-five (25) subjects met the participation criteria and completed all of the stages of the study, with 22 subjects being in the 17 to 24 age group, and 3 in the 25 to 34 age group. Regarding their education, 21 reported completing their high school degree, 2 had their bachelor's degree, 1 had a graduate level of education, and 1 had an associate (2-years) degree. Regarding computer programming experience, 17 reported no experience with computer programming and 8 reported little experience. All subjects reported playing the game at least three times in the post-game survey.

3.1.1 Changes in Programming Comprehension. In the pre-game survey, 9 subjects were able to correctly answer questions about variables, 4 subjects correctly answered the loop question, and no one was able to correctly answer the question regarding methods. In the post-game survey, 15 subjects correctly answered questions about variables, 5 subjects correctly answered about loops, and 15 subjects were able to correctly answer the question regarding methods. In other words, 6 subjects showed improved knowledge of variables, only 1 subject showed improved knowledge regarding loops, and 15 subjects showed knowledge gains regarding methods. After re-examining the questions regarding loops, these questions may have been too challenging to understand for those not having seen a programmatic loop structure before.

3.1.2 Programming Concept Mapping: Regarding connecting game elements to elements of a pseudocode program, 15 subjects were able to correctly connect the `Variable` card, 14 subjects were able to correctly connect the `Repeat` card, and 11 subjects were able to correctly connect the `Method` card. From the 25 subjects, 8 were able to relate all three concepts, 6 subjects were able to relate two concepts, 4 subjects were able to relate one concept, and there were 7 subjects who could not relate any concepts with real-world programming elements.

Among the three concepts, most subjects (n=15) were not able to map the `Method` card to a method in the pseudocode. Also, for the subjects who were able to make only two connections, 4 out of 6 were those who missed the method concept mapping. Of the 4 subjects who could only connect one concept successfully, 2 were able to identify the variable structure. Further examination of the post-game programming questions indicated that some of the pseudocode might have been too challenging for the subjects, considering that 31% of the subjects could not connect a programming structure correctly with the gameplay.

3.1.3 Subject Feedback. In the post-game survey, almost all subjects (n=23) reported that they liked playing Program Wars *v.*2.0, and 19 agreed that they would suggest playing Program Wars *v.*2.0 to their friends. Regarding self-assessment of learning, a large portion (n=13) were not sure that playing Program Wars *v.*2.0 improved their understanding of programming knowledge. Perhaps this is a result of not providing feedback to subjects on if they answered questions correctly. Although there were help and tutorials resources provided, a few participants mentioned that initially, the gameplay was tricky. As the study was unsupervised, it cannot be known whether participants made use of those help and tutorial resources. When asked whether they will play Program Wars in the future, 11 subjects said that they would play Program Wars *v.*2.0 to further improve their knowledge of computer programming.

When asked for any additional comments, a couple of subjects indicated it was a 'good game,' and they had fun playing the game. A couple of subjects found the UI and game rules to be complex. However, this may be an expected reaction, as, for any new card game, the initial couple of rounds are typically a challenge as players learn the rules of the game. This is part of the reason that subjects were asked to play Program Wars at least three times. One of the subjects remarked that the game has *"A very good concept to teach programming [. . .] basics and ideas, The game is really enjoyable, and it enhanced my knowledge regarding basic programming [. . .]."*

3.2 Analysis

To understand the the effect of playing Program Wars *v.*2.0, we chose to complete a 'within-subject' analysis whereby we categorize a subject's knowledge improvement into one of four categories. If a subject answered both the pre-and post-game questions for a concept correctly, they were considered to have had *Prior Knowledge* of that concept before playing the game. If a subject answered a concept's question(s) correctly in the pre-game survey but incorrectly in the post-game survey, then their knowledge improvement was considered as *Unclear* as they seemed to have lost knowledge. If a subject answered a concept's question(s) incorrectly in the pre-game survey and correctly in the post-game survey, we considered this a demonstration that *Knowledge Improved* for that concept. Finally, if a subject answered a concept's questions incorrectly in both the pre-and post-game survey, this was considered as *No Change in Knowledge*. The results for the knowledge areas of *Variable*, *Loop*, and *Method* are shown in the last three columns of Table 2.

Table 2: Knowledge changes from playing Program Wars *v.*2.0.

Knowledge Improvement	Pre-Game	Post-game	Variable	Loop	Method
Had Prior Knowledge	Correct	Correct	9	2	0
Unclear Result	Correct	Incorrect	0	2	0
Knowledge Improved	Incorrect	Correct	7	3	16
No Change in Knowledge	Incorrect	Incorrect	9	18	9

Table 3: Analysis and results for understanding if Program Wars *v.*2.0 helped in learning programming concepts.

Conclusion	Knowledge Gain	Conceptual Mapping	Variable Concept	Loop Concept	Method Concept
Playing Program Wars *v.*2.0 helped	Improved or Previous	Did Connect	10	2	7
Playing Program Wars *v.*2.0 may have helped	Unclear or No Change	Did Connect	5	12	4
Contradictory evidence	Improved or Previous	Did Not Connect	5	9	6
Playing Program Wars *v.*2.0 did not help	Unclear or No Change	Did Not Connect	6	3	9

3.2.1 Does Playing Program Wars v.2.0 Help In Understanding Programming Concepts? From the study results, we found clear improvement in their knowledge was shown by 16 subjects for methods, 7 subjects for variables, and 3 for loops (i.e. the *Knowledge Improved* row). This would indicate that by playing Program Wars *v.*2.0, most subjects gained knowledge about the method concept, some gained knowledge about the variable concept, and not many gained new knowledge about the loop structure.

A paired samples t-test was performed to compare the improvement between pre- and post-game scores for each knowledge area. There was a significant difference in scores between pre- ($M = 0.36, SD = 0.24$) and post-game ($M = 0.64, SD = 0.24$) for variable; $t(24) = -3.055, p = 0.003$. Similarly, there was a significant difference in pre-game ($M = 0, SD = 0$) and post-game ($M = 0.64, SD = 0.24$) scores for method; $t(24) = -6.532, p = 4.68x10^{-7}$. However, a significant difference was not found between the pre- ($M = 0.16, SD = 0.14$) and post-game ($M = 0.2, SD = 0.17$) scores for loops; $t(24) = -0.44, p = 0.332$.

To answer our research question, we examined the results in Table 2 along with the results of the programming concept mapping. Table 3 shows the results of this analysis. Depending on a subject's observed knowledge improvement and if they were also able to correctly map Program Wars *v.*2.0 elements to real-world programming elements (columns 2 & 3), this was taken as evidence that either playing Program Wars *v.*2.0 (1) helped the subject improve their knowledge, (2) may have helped as subjects correctly mapped concepts but answered questions incorrectly, (3) that the evidence is contradictory as subjects showed prior knowledge or gains but did not demonstrate they could map concepts, or (4) that the evidence shows that playing Program Wars *v.*2.0 did not help improve their knowledge. The last three columns of Table 3 show the results of this analysis for the three programming concepts studied.

The results of (1) and (2) are taken to support saying that playing Program Wars *v.*2.0 does help, and the results of (3) and (4) are taken as evidence that playing Program Wars *v.*2.0 does not help. Based on the analysis, Program Wars *v.*2.0 helped 60% of subjects to understand the *Variable* concept, helped 56% to understand the *Loop* concept and helped 44% to understand the *Method* concept. Therefore, we conclude that **playing Program Wars *v.*2.0 does improve a player's knowledge of basic computer programming concepts**.

3.2.2 Comparison between v.1.0 and v.2.0. A similar user study assessed the effectiveness of Program Wars *v.*1.0 [11]. Table 4 presents a comparison between the two versions of Program Wars in terms of the three programming concepts (i.e. variable, loop, method). From the comparison, we found that Program Wars *v.*2.0 performs better for learning about loops and methods than Program Wars *v.*1.0, and Program Wars *v.*1.0 performs better for teaching the variable concept. However, after examining demographic information of the study for Program Wars *v.*1.0, those subjects were recruited from "an undergraduate introductory course for non-computer science majors ... [which included] a one-week introduction to computer programming using Python and Scratch." This indicates that the subjects in the prior study likely had some recent experience with programming concepts. Therefore, the subjects' prior knowledge about variables was likely less in this study, and the improvement shown for Program Wars *v.*2.0 is likely to be more meaningful.

3.3 Threats to Validity

As the study was conducted asynchronously online due to COVID restrictions, this situation may have affected the validity of the results as subjects were not able to ask clarifying questions, and researchers were not able to make direct observations. This is likely

Table 4: Results comparison between Program Wars *v.*1.0 and Program Wars *v.*2.0.

Programming Concept	v 1.0 (%)	v 2.0 (%)
Variable	67	60
Loop	47	56
Method	31	44

most evident from the subjects' comments regarding the complexity of the game. Had the subjects been observed while completing the study, clarifications could have been provided, and further insights regarding Program Wars *v.*2.0 may have been possible. Although in-game access to web pages describing the gameplay and cards[4] is provided, it is unknown if subjects used this information.

To judge changes in a subject's knowledge, we compared a subject's answers in their pre- and post-game surveys. Therefore, it is possible that some subjects guessed answers. For example, the two subjects who were considered to have *Unclear* results for the *Loop* concept may have been the result of them correctly guessing the answer(s) in the pre-game survey and incorrectly in the post-game survey. Similarly, those who were considered to have *Had Prior Knowledge* may contain some subjects that correctly guessed answers in both surveys. As subjects were not observed when completing the study, we cannot determine if this behaviour existed and, if so, how much it impacted the results.

In answering our research question, the "Playing may have helped" and "Contradictory Evidence" conclusions could be considered as 'weak support' for answering the question either way. If we consider that 16%, 32%, and 20% of subjects fall into the "weak support" category for the variable, loop, and method concepts respectively, this means that Program Wars *v.*2.0 may be more effective at teaching the programming concepts than the analysis shows. However, the converse may also be true.

Finally, as in any such study, we cannot eliminate the possibility of social bias (i.e. "please the researcher" bias) in the feedback responses. However, as some subjects did express their frustrations when playing Program Wars in their comments, we believe that this bias was minimal.

4 RELATED WORK

Although several research works propose methods for teaching computer programming, we restrict ourselves to card games, board games, or web-based games as these are the most comparable to Program Wars.

Potato Pirates [9] is a physical card game with the objective of teaching programming concepts. The game covers a variety of computer programming concepts such as variables, functions, while-loops, if-else conditionals and nested loops. Also, some cards represent interrupts and control flow.

Several board games have been developed for teaching programming concepts. Battle Bots v2 [1] was inspired by the Robo Rally [34] game. The game has similar programming and action-performing aspects as Program Wars. Robot Turtles [10] was designed for children and uses simple direction cards to move a coloured turtle toward a goal square. The game includes a *Jump* card that can replace a set of instruction cards, which inspired the *Method* card in Program Wars *v.*2.0.

There are many web-based games for learning programming. In CodeCombat [5], the player role-plays as a warrior in a swords-and-sorcery setting and creates programs to overcome challenges like clearing a maze and avoiding hazards. Blockly Maze [3] introduces the concept of programming loops and conditions using a graphical programming language which can compile to other languages. The game's main objective is to take the avatar from a starting point to an endpoint. Similarly, Kodable [8] uses a graphical programming language to introduce children to the concepts of logic, loops, conditional statements, and functions.

Codin [7] supports the learning of many programming languages, such as Python, Ruby, Java, and Scala. The main objective of the game is to improve player's existing coding skills by solving different problems, applying new strategies, and getting inspired by other opponents as they compete on a leaderboard.

5 CONCLUSION

The focus of this work was to further investigate the use of GBL in teaching the core principles of computer programming to those with little to no experience in this area. Using the results from a prior study of Program Wars, we examined the effects of changing several aspects of the game to better support learning and engagement. These changes included adding the `Method` card, introducing the concept of algorithms in the form of the `Search` and `Sort` cards, and changes to the UI. Also, the conditional statement concept was made more explicit in this version using bonus objectives. Regarding improvements to engagement, changes were made to the game such as two modes of play were introduced to allow players a basic and more advanced playing experience.

The results of a user study of Program Wars *v.*2.0 showed that the changes resulted in improvements in learning. For example, the use of the `Method` card was much more effective in helping players to learn this concept, with 44% of players showing an improvement in knowledge versus 31% in the previous study. Similarly, the number of players showing a better understanding of the loop concept improved from 47% for Program Wars *v.*1.0 to 56% for Program Wars *v.*2.0. Although these improvements appear to be only modest, when considering that the participants in the Program Wars *v.*1.0 study were taken from an introductory CS course and had recently been exposed to these programming concepts, whereas participants of Program Wars *v.*2.0 study were taken from a non-CS background population, then these differences become more meaningful. Overall, the results indicate that if subjects continued to play Program Wars, they would see further knowledge improvements.

We plan to continue investigating how playing Program Wars *v.*2.0 affects players' knowledge about programming with a wider study involving other institutions and/or a study at the K-12 level. Also, we are exploring the extension of Program Wars *v.*2.0 to teach the basics of the software development life-cycle and iterative development.

[4] https://program-wars.firebaseapp.com/help

REFERENCES

[1] 2013. Battle Bots v2. https://www.curufea.com/doku.php?id=games:board:battlebots. [Online; accessed 24-Jan-2024].

[2] 2023. Robocode. https://robocode.sourceforge.io/. [Online; accessed January 25, 2024].

[3] 2024. Blockly Maze. https://www.brainpop.com/games/blocklymaze. [Online; accessed 24-Jan-2024].

[4] 2024. CheckIO. https://py.checkio.org/. [Online; accessed 24-Jan-2024].

[5] 2024. CodeCombat. https://codecombat.com. [Online; accessed 24-Jan-2024].

[6] 2024. CodeWars. https://www.codewars.com/. [Online; accessed 24-Jan-2024].

[7] 2024. CodinGame. https://www.codingame.com. [Online; accessed 24-Jan-2024].

[8] 2024. Kodable. https://www.kodable.com/. [Online; accessed 24-Jan-2024].

[9] 2024. Potato Pirates. https://potatopirates.game/. [Online; accessed 24-Jan-2024].

[10] 2024. Robot Turtles. http://www.robotturtles.com. [Online; accessed January 25, 2024].

[11] John Anvik, Vincent Cote, and Jace Riehl. 2019. Program Wars: A Card Game for Learning Programming and Cybersecurity Concepts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 393–399. https://doi.org/10.1145/3287324.3287496

[12] Serdar Arcagok. 2021. The impact of game-based teaching practices in different curricula on academic achievement. *International Online Journal of Education and Teaching (IOJET)* 8 (2021), 778–796. Issue 2.

[13] Alex Baker, Emily Oh Navarro, and André van der Hoek. 2005. An experimental card game for teaching software engineering processes. *Journal of Systems and Software* 75, 1 (2005), 3–16. https://doi.org/10.1016/j.jss.2004.02.033 Software Engineering Education and Training.

[14] Nathalie Barz, Manuela Benick, Laura Dörrenbächer-Ulrich, and Franziska Perels. 2023. The Effect of Digital Game-Based Learning Interventions on Cognitive, Metacognitive, and Affective-Motivational Learning Outcomes in School: A Meta-Analysis. *Review of Educational Research* 0, 0 (2023), 00346543231167795. https://doi.org/10.3102/00346543231167795 arXiv:https://doi.org/10.3102/00346543231167795

[15] Kohl Bromwich, Masood Masoodian, and Bill Rogers. 2012. Crossing the Game Threshold: A System for Teaching Basic Programming Constructs. In *Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction* (Dunedin, New Zealand) *(CHINZ '12)*. ACM, New York, NY, USA, 56–63. https://doi.org/10.1145/2379256.2379266

[16] Alejandro Calderón, Mercedes Ruiz, and Elena Orta. 2017. Integrating Serious Games as Learning Resources in a Software Project Management Course: The Case of ProDec. In *2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)*. 21–27. https://doi.org/10.1109/SECM.2017.3

[17] Carnegie Mellon University. 2024. Alice. Online. http://www.alice.org

[18] Juho Hamari, David J. Shernoff, Elizabeth Rowe, Brianno Coller, Jodi Asbell-Clarke, and Teon Edwards. 2016. Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning. *Computers in Human Behavior* 54 (2016), 170–179. https://doi.org/10.1016/j.chb.2015.07.045

[19] J. Huizenga, W. Admiraal, S. Akkerman, and G. ten Dam. 2009. Mobile game-based learning in secondary education: engagement, motivation and learning in a mobile city game. *Journal of Computer Assisted Learning* 25, 4 (2009), 332–344. https://doi.org/10.1111/j.1365-2729.2009.00316.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2729.2009.00316.x

[20] Molly Stewart LawlorKimberly A. Schonert-ReichlJenna Whitehead Jacqueline E. Maloney. 2016. *A Mindfulness-Based Social and Emotional Learning Curriculum for School-Aged Children: The MindUP Program.* Springer, New York, NY.

[21] IEEE Computer Society & Association for Computing Machinery Joint Task Force on Computing Curricula. 2015. Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. https://www.acm.org/binaries/content/assets/education/se2014.pdf. [Online; accessed 16-Oct-2020].

[22] Omar Karram. 2021. The Role of Computer Games in Teaching Object-Oriented Programming in High Schools - Code Combat as a Game Approach. *WSEAS TRANSACTIONS ON ADVANCES in ENGINEERING EDUCATION* 18 (04 2021), 37–46. https://doi.org/10.37394/232010.2021.18.4

[23] Feng-Ying Li, Gwo-Jen Hwang, Pei-Ying Chen, and Yu-Jung Lin. 2021. Effects of a concept mapping-based two-tier test strategy on students' digital game-based learning performances and behavioral patterns. *Computers & Education* 173 (2021), 104293. https://doi.org/10.1016/j.compedu.2021.104293

[24] Priyaadharshini M, Natha Mayil N, R Dakshina, Sandhya S., and Bettina Shirley R. 2020. Learning Analytics: Game-based Learning for Programming Course in Higher Education. *Procedia Computer Science* 172 (2020), 468–472. https://doi.org/10.1016/j.procs.2020.05.143 9th World Engineering Education Forum (WEEF 2019) Proceedings : Disruptive Engineering Education for Sustainable Development.

[25] Michael A. Miljanovic and Jeremy S. Bradbury. 2017. RoboBUG: A Serious Game for Learning Debugging Techniques. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 93–100. https://doi.org/10.1145/3105726.3106173

[26] MIT University. 2018. Scratch. Online. https://scratch.mit.edu

[27] Pablo Moreno-Ger, Daniel Burgos, Iván Martínez-Ortiz, José Luis Sierra, and Baltasar Fernández-Manjón. 2008. Educational game design for online education. *Computers in Human Behavior* 24, 6 (2008), 2530 – 2540. https://doi.org/10.1016/j.chb.2008.03.012 Including the Special Issue: Electronic Games and Personalized eLearning Processes.

[28] Tahereh Partovi and Majid Reza Razavi. 2019. The effect of game-based learning on academic achievement motivation of elementary school students. *Learning and Motivation* 68 (2019), 101592. https://doi.org/10.1016/j.lmot.2019.101592

[29] Mikki H. Phan, Jo R. Jardina, Sloane Hoyle, and Barbara S. Chaparro. 2012. Examining the Role of Gender in Video Game Usage, Preference, and Behavior. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 56, 1 (2012), 1496–1500. https://doi.org/10.1177/1071181312561297 arXiv:https://doi.org/10.1177/1071181312561297

[30] Josephine M. Randel, Barbara A. Morris, C. Douglas Wetzel, and Betty V. Whitehill. 1992. The Effectiveness of Games for Educational Purposes: A Review of Recent Research. *Simulation & Gaming* 23, 3 (1992), 261–276. https://doi.org/10.1177/1046878192233001 arXiv:https://doi.org/10.1177/1046878192233001

[31] José María Rodríguez Corral, Antón Civit Balcells, Arturo Morgado Estévez, Gabriel Jiménez Moreno, and María José Ferreiro Ramos. 2014. A game-based approach to the teaching of object-oriented programming languages. *Computers & Education* 73 (2014), 83–92. https://doi.org/10.1016/j.compedu.2013.12.013

[32] Sumarie Roodt and Yusuf Ryklief. 2022. *Using Digital Game-Based Learning to Improve the Academic Efficiency of Vocational Education students.* 643–671. https://doi.org/10.4018/978-1-6684-5696-5.ch037

[33] Md. Hasan Tareque. 2021. *Updating a Web-based Card Game to Teach Programming, Cybersecurity and Software Development Life Cycle Concepts.* Master's thesis. University of Lethbridge.

[34] Ingo J. Timm, Tjorben Bogon, Andreas D. Lattner, and René Schumann. 2008. Teaching Distributed Artificial Intelligence with RoboRally. In *Multiagent System Technologies*, Ralph Bergmann, Gabriela Lindemann, Stefan Kirn, and Michal Pěchouček (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 171–182.

[35] Liang-Hui Wang, Bing Chen, Gwo-Jen Hwang, Jue-Qi Guan, and Yun-Qing Wang. 2022. Effects of digital game-based STEM education on students' learning achievement: a meta-analysis. *International Journal of STEM Education* 9, 1 (17 Mar 2022), 26. https://doi.org/10.1186/s40594-022-00344-0