Program Wars

A Card Game for Learning Programming and Cybersecurity Concepts

John Anvik
University of Lethbridge
john.anvik@uleth.ca

Vincent Cote
University of Lethbridge
vincent.cote@uleth.ca

Jace Reihl
University of Lethbridge
jace.riehl@uleth.ca

ABSTRACT

Although there are many computer science learning games with the goal of teaching programming, such games typically require the person to either learn an existing programming language or the game's own specialized language. This can be intimidating, confusing or frustrating for an individual when they cannot get their "program" to work correctly (e.g. syntax error, infinite loop). Additionally, such games commonly use a puzzle-solving approach that does not appeal to some demographics.

This paper presents a programming-language-independent approach to teaching fundamental programming and cybersecurity concepts using simple vocabulary. This approach also uses the familiar activity of playing cards against opponents to create a more dynamic and engaging learning experience. The approach is demonstrated by a web-based game called *Program Wars* . Results from a user study show that players are able to effectively connect game concepts to actual programming language structures; however, whether players' comprehension of computer programming is improved is unclear.

CCS CONCEPTS

• **Social and professional topics → Software engineering education**; • **Software and its engineering** → *Language features*;

KEYWORDS

Programming language education, web application, card game

ACM Reference Format:
John Anvik, Vincent Cote, and Jace Reihl. 2019. Program Wars: A Card Game for Learning Programming and Cybersecurity Concepts. In *Proceedings of SIGCSE Technical Symposium (SIGCSE'19).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Although there are many learning games and environments with the goal of teaching computer science concepts [4, 9, 13, 22] and software engineering [2, 7, 14, 17], those that focus on teaching programming typically fail in two respects.

First, most such games require the user to either learn an existing programming language (e.g. Javascript, Python, C) or their own specialized language (e.g. Scratch [21] or Alice [20]). For an individual not familiar with programming structures, it can be intimidating, confusing or frustrating when they cannot get their "program" to work correctly. For example, novices commonly create syntax errors when they do not understand the "programming language", or they create logical errors, such as an infinite loop. Bromwich et al. [5] observed that many educational programming languages and environments are often overly ambitious in what they teach.

Second, the gameplay generally falls into one of two different game styles: puzzle solving or sandbox. The premise of puzzle-solving style games tends to be a variation of getting a robot to navigate a maze, a problem domain that does not appeal to some demographics, such as females [18]. Sandbox games attempt to address this limitation by allowing the individual to create their own stories in an open-ended way, thereby connecting to an individual's interests. However, such flexibility requires the guidance of the individual by an outside influence, such as a teacher, to ensure the progression of learning [5].

Program Wars[1] is a web-based card game for teaching or reinforcing the fundamental concepts of programming and cybersecurity to those with limited or no programming experience. Being web-based, it is freely available for use by any institution wishing to adopt it.[2] In the game, players construct a computer program that meets or exceeds a goal number of statements. Players build their program using "instruction" cards and "repetition" cards. Some repetition cards allow the player to change the number of repetitions using "variable" cards. Players can also "group" instructions to protect portions of their program from being cyberattacked (i.e. "hacked").

Program Wars addresses both of the above limitations. First, Program Wars addresses the problem that "the bad usability of programming languages increases [the] difficulty [of learning programming]." [12]. It does this by using a programming-language-independent approach that focuses on the fundamental programming language concepts using a simple vocabulary (e.g. "instruction", "group", "repeat"). Second, Program Wars uses the familiar activity of playing cards, a game type that has been previously used for teaching such topics as algorithm analysis and design [8] and software engineering [1, 3, 19]. This game type has been found to have more general and long-term appeal [6, 11, 15, 16, 18] than solving a maze. In Program Wars the player faces one or more AI opponents for a dynamic and engaging learning experience.

Program Wars is similar in teaching scope to VISPROCON [5], which also focuses on only teaching the concepts of sequences, loops, and conditionals. Whereas VISPROCON uses the play model of navigating a robot through a maze, Program Wars uses the play model of a card game.

[1]https://github.com/johnanvik/program-wars
[2] https://programming-wars.firebaseapp.com/

A pilot user study with non-computer science undergraduates was conducted to answer three research questions (see Section 3). Participants completed a pre-game questionnaire, played several games of Program Wars, and completed a post-game questionnaire.

This paper proceeds as follows. First, a detailed description of Program Wars is presented. Next, a description of the pilot user study conducted to evaluate Program Wars is given followed by the results and a discussion of the study. Finally, improvements made to Program Wars subsequent to the pilot user study are described before concluding the paper.

2 PROGRAM WARS

This section presents a high-level view of the gameplay and a detailed description of the basic cards in Program Wars. Advanced cards are presented in Section 5.

2.1 Overview of Gameplay

At the start of Program Wars, the player chooses the target number of points (e.g. 35, 75, 105) and the number of opponents (1-3) against which they wish to play (the deck size is proportional to the number of players). Each player is dealt six cards and on each turn receives another card from the deck.

The gameplay of Program Wars is made up of a series of rounds in which each player has a turn. On a turn, a player can either play or discard a card from their hand. To win, a player needs to create a program that reaches a minimum number of points for the currently selected branch (see Section 2.2.1). The points are the total number of instructions that would be executed by the computer. Players use a variety of means to accumulate points, all the while fending off attacks from the other players. As the check for a winner happens at the end of each round, it is possible for more than one player to complete their program and win the game.

Figure 1 shows part of the way through a two-player game. Some of the elements shown in the figure are improvements made after the pilot user study (outlined by dotted red boxes and labelled "Version 2"; see Section 5 for details).

The top half of the screen (left to right) shows the current progress of the two players for each of their play area, the current player's hand, and the cyberattack and cybersecurity cards that are affecting the player.

The bottom half of the screen shows the current player's True and False play areas, with the False branch being the active play area for this round. In the True path, the player has one stack of cards totalling 15 instructions and another card stack totalling 18 instructions. Collectively, the True path contains 33 instructions. The False path contains two card stacks, each with one card, producing a total of 2 instructions for this path. However, due to the effect of a *Malware* card, the effective instruction total for the True path is reduced to 25 (see Section 5.1.1).

Three aids are provided to assist a player in learning the game. First, there is an 11-min video that shows a game being played. Second, players can access a page from the game that describes the rules of the game and all of the cards. Lastly, when a player clicks on a card, a description of the card is presented on the left side of the top area of the screen. Players can choose to disable this feature once they are familiar with the game.

2.2 Game Cards and Areas

The player builds their program using the basic building blocks of any program: instructions, groups of instructions, repetition, and branching. Also, the player can launch a cyberattack at an opponent, or prepare a defence against their opponents. This section describes each of the cards in the game and their relation to computer science concepts.

2.2.1 Branching. The programming concept of *branching* or *paths* is represented in the game by two playing areas, each representing the 'True' and 'False' path of the traditional *if-then* conditional statement. At the start of each round in the game, one of the two playing areas is randomly selected to be the 'active' branch and players build their program in that play area during their turn. [3]

2.2.2 Instructions. As in actual computer programs, instructions form the backbone of the player's created program. Each instruction card represents a fixed number of instructions (1, 2, or 3). These cards are the base of a card stack created in one of the two playing areas. As the player needs to play this type of card to start gaining points in the game, it is often the card played on their first turn.

Figure 1 shows examples of instruction cards. In the True play area, there is an example of an *Instruction-3* card, and in the False play area, there are two *Instruction-1* cards.

2.2.3 Repetition. Repeat cards allow the player to multiply the effect of an Instruction, Group, or another Repeat card. These cards represent the concept of a *loop*. There are three sizes of Repeat cards: 2, 3, and 4. By placing a Repeat card on another Repeat card, the player can form a nested loop. An example of Repeat cards forming a nested loop is shown in the True play area where a *Repeat-3* has been placed on a *Repeat-2*, which is on an *Instruction-3*, creating a card stack of 18 instructions.

In addition to the fixed-size Repeat cards, there is also a variable Repeat card (called *Repeat-X*). This card has no advantage by itself; it acts as a *Repeat-1* card. This card is used in conjunction with a Variable card. Figure 1 shows an example of its use, along with a Variable card, on the leftmost side of the True play area.

2.2.4 Variables. Variable cards represent the concept of a *variable*. Placing a Variable card on a *Repeat-X* increases its multiplicative power. The Variable cards have values of 3 through 6 inclusive. If the Variable card is on the top of the card stack (i.e. not buried by another Repeat card) it can be replaced by another Variable card on a player's turn.

Figure 1 shows an example where a *Variable-5* card is used (leftmost side of the True play area). If the player chooses, they could replace this card with the *Variable-3* or *Variable-4* card in their hand.

2.2.5 Groups. A Group card represents the concept of a *function*, *method*, or *procedure* in the game. A player can 'group' either a single instruction card or a stack of cards (i.e. a collection of Instruction, Repetition and Variable cards) that totals the value of the Group card. Group cards have fixed values of 2 through 6, inclusive. Group cards protect a portion of the player's program from a *Hack* cyberattack (see Section 2.2.6). The use of Group cards also

[3] We plan to explore the use of Variable cards and a conditional expression to more accurately represent this concept in a future version of the game.

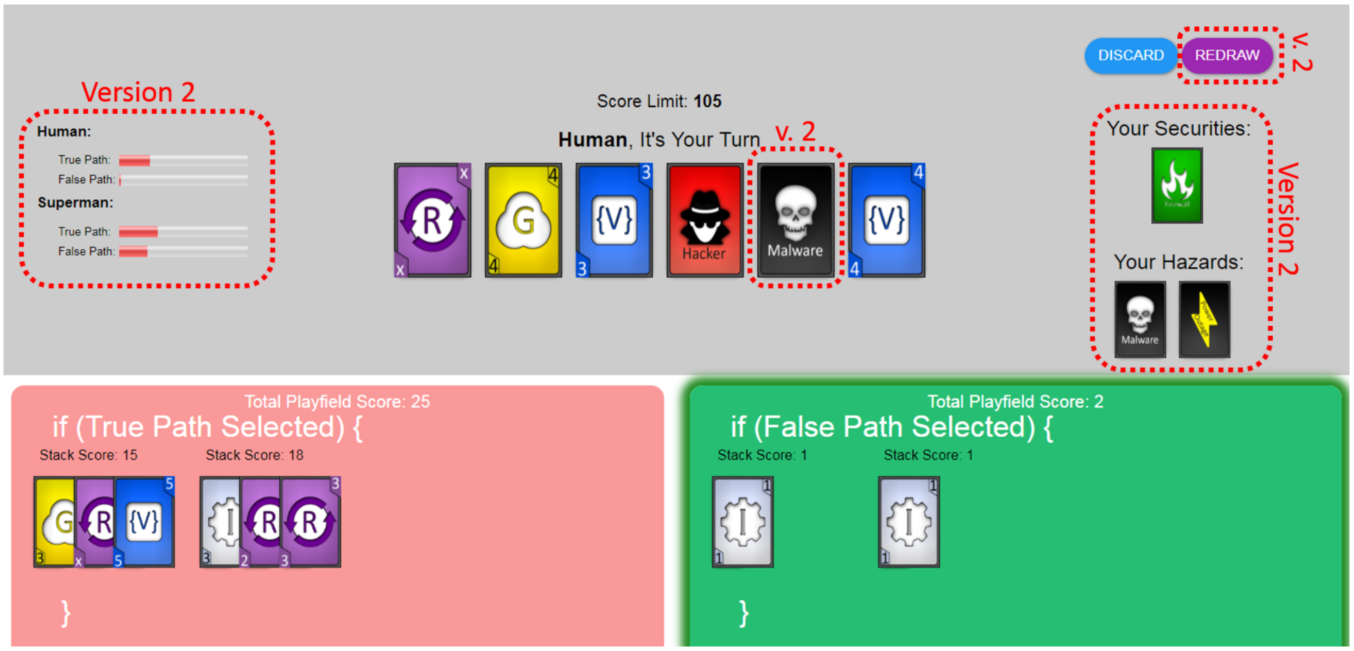


Figure 1: Partway through a game where the *False* path has been selected for the current round.

introduces players to the software engineering practice of source code refactoring [10].

For example, assume that in Figure 1, the False play area also contained an *Instruction-2* card. On their turn, the player could play the *Group-4* card from their hand to replace the two *Instruction-1* cards and the *Instruction-2* card, as the three card total is 4. We can see that a similar situation occurred previously on the True path, where a *Group-3* card is under a *Repeat-X* and *Variable-5*.

2.2.6 Cyberattack. As cybersecurity is an important aspect of modern day software development, Program Wars includes one cybersecurity concept in addition to programming language concepts. This card also makes the game interactive between the players.

The *Hack* card represents an attack by a malicious programmer. It allows a player to remove one stack of instructions from one of an opponent's play areas. Card stacks that contain a *Group* card are protected from being 'hacked'.

3 PILOT USER STUDY

This section presents a pilot user study that collected both quantitative and qualitative data regarding the playing of Program Wars. The objective of the study was to answer three research questions:

RQ1 Does playing Program Wars lead to players having a better understanding of a computer program?
RQ2 Is a player of Program Wars able to connect game concepts to elements in actual computer programs?
RQ3 What are the players' impressions of Program Wars?

The remainder of this section presents the methodology used to answer these research questions.

3.1 Methodology

To find answers to our research questions, we conducted a user study.[4] Participants in the study were asked to do the following:

(1) **Navigate to the URL for the study.** The study was conducted entirely in a web browser. The intent was to allow participants to participate in the study at their own choice of time and at their own pace without influence from the researchers. It was also believed that doing so would lead to more participation in the study.[5]
(2) **Complete a demographics questionnaire.** This questionnaire collected information about the participant's background including previous experience with programming concepts, software for teaching programming, and card games.
(3) **Complete a pre-game survey.** Participants were given five multiple choice questions to test their pre-existing skill at understanding a computer program. For each question, the participant examined a simple arithmetic program containing no input/output statements[6] and responded with what they thought was the computed value.
The programs were presented in one of four programming languages (C, Python, Pascal or Fortran), as a goal of Program Wars is to teach the fundamental concepts of programming independent of a specific programming language. The program for each question was the same with the programming language being randomly chosen. For example, assuming

[4]The practices and procedures for the user study were reviewed by a Human Subject Research Committee to ensure that standards for ethical treatment of participants, such as informed consent and privacy, were addressed.
[5]Although participants may have received outside help from peers or online resources during the study, we believe this to be unlikely or to have a minimal effect given the questions asked in the questionnaires.
[6]This choice was made to limit the complexity of the programs and responses.

five different participants, for the first question Participant #1 was shown a C program, Participant #2 was shown a Fortran program, Participant #3 was shown a Pascal program, Participant #4 was shown a C program, and Participant #5 was shown a Fortran program. Figure 2 shows an example of one pre-game survey question using C.[7]

(4) **Play Program Wars.** Participants were asked to play three short games of Program Wars.

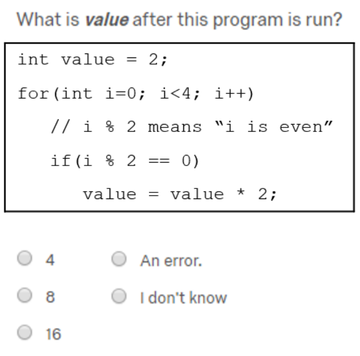
(5) **Complete a post-game questionnaire.** Participants were asked five multiple choice programming questions similar to those asked in the pre-game survey. Again, the programs presented were given in one of the four programming languages. The five questions in the post-game survey were paired with corresponding questions in the pre-game survey to require the same knowledge for understanding the program, but not necessarily presented using the same programming language. Figure 3 shows the corresponding post-game Python variant corresponding to the question given in Figure 2. Participants' post-game responses were compared with their pre-game responses to answer RQ1.
Participants were also given four multiple choice questions to assess if they could connect the Grouping, Instruction, Repetition and Variable cards used in Program Wars with elements of a computer program. Figure 4 shows an example of such a question. As with the programming questions, the connection questions were given using a randomly selected programming language from the four languages. The results of these questions were used to answer RQ2.
Finally, participants were asked for any qualitative feedback about Program Wars to answer RQ3.

Participants were expected to take 45 minutes to complete the study - 15 minutes for each of completing the pre-game questionnaire, playing three games, and completing the post-game questionnaire. In order to match pre- and post-game questionnaires, participants were asked to provide an email address in both questionnaires. This matching was done by only one of the authors to preserve privacy as much as possible. Once the matching between pre- and post-game questionnaires were completed, this personal identifying information was replaced by a unique study id. Qualtrics was used for presenting and collecting the questionnaire data.

3.2 Participants

The target population for Program Wars is those with little or no programming experience. For the study, participants were recruited from an undergraduate introductory course for non-computer science majors. The course provided a survey of computer science topics, including a one-week introduction to computer programming using Python and Scratch. It was believed that participants from this course would have a high probability of meeting the study criteria (although this was not guaranteed). The course had an enrollment of 80 students, of which over half (46 students) participated in the study.

[7]The corresponding Pascal, Python and Fortran programs have been omitted from the paper for reasons of space.

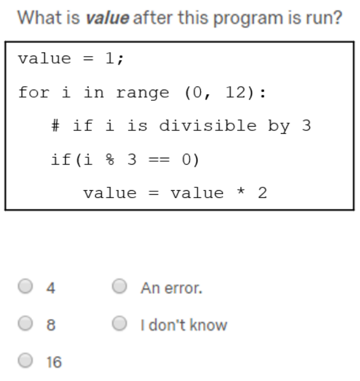


What is ***value*** after this program is run?

```
int value = 2;
for(int i=0; i<4; i++)
   // i % 2 means "i is even"
   if(i % 2 == 0)
      value = value * 2;
```

4
8
16
An error.
I don't know

Figure 2: A pre-game programming question (C).



What is ***value*** after this program is run?

```
value = 1;
for i in range (0, 12):
   # if i is divisible by 3
   if(i % 3 == 0)
      value = value * 2
```

4
8
16
An error.
I don't know

Figure 3: A post-game programming question (Python).

4 RESULTS

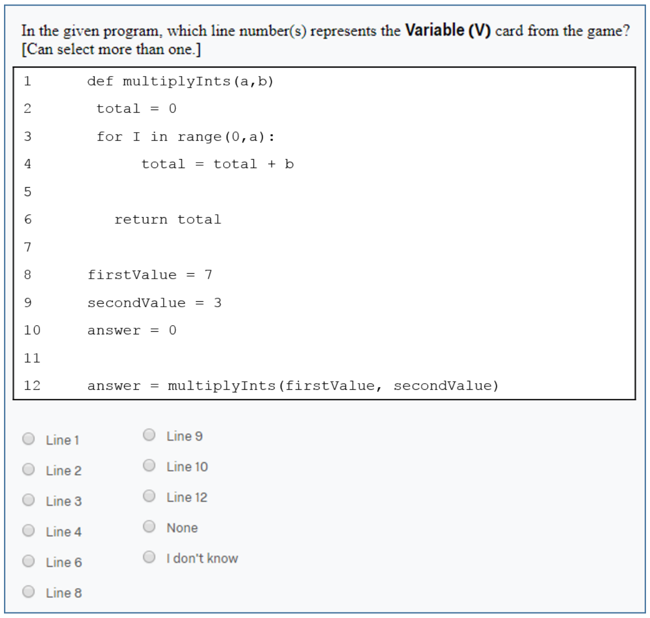
This section presents the results of our pilot study and discusses what we discovered in our evaluation.

4.1 Demographics

All forty-six participants were in the age range of 18-25 years, with 70% reporting as male and 30% as female. The participants were from a wide range of disciplines as shown in Table 1. Roughly half of the participants (56%) had previously used programming learning sites such as Scratch, Code.org, or Khan Academy. Figure 5 shows the participants' reported level of experience with counting card games (e.g. Poker or Bridge) and video games. Nearly all participants were familiar with such card games and play video games. As may be expected, participants played video games more frequently.

Table 1: Areas of study of the participants.

Arts	9	Humanities	1
Business / Management	9	Social Science	7
Education	4	Science	14
Health	1		



In the given program, which line number(s) represents the **Variable (V)** card from the game? [Can select more than one.]

```
1       def multiplyInts(a,b)
2        total = 0
3        for I in range(0,a):
4             total = total + b
5
6          return total
7
8       firstValue = 7
9       secondValue = 3
10      answer = 0
11
12      answer = multiplyInts(firstValue, secondValue)
```

- Line 1
- Line 2
- Line 3
- Line 4
- Line 6
- Line 8
- Line 9
- Line 10
- Line 12
- None
- I don't know

Figure 4: A post-game question connecting a game card to a programming concept (Python variant).

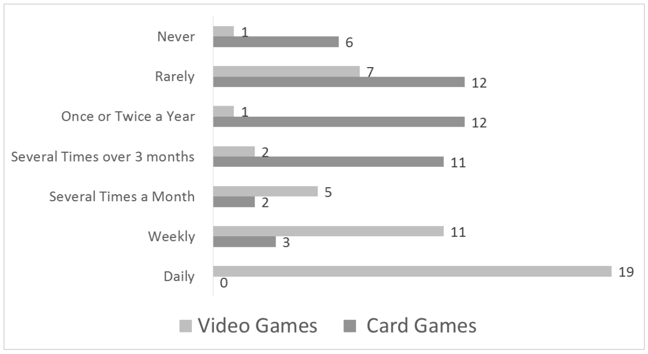


Figure 5: Participants’ experience with card/video games.

4.2 Analysis

This section presents the results of the analysis regarding any improvements in the comprehension of programs written in a programming language (RQ1), how well the participants were able to connect the concepts in Program Wars to elements in an actual program (RQ2), and their impressions of the game (RQ3).

4.2.1 Comprehension of Computer Programs. Trends regarding participants’ improvement in comprehension of a computer program were assessed by comparing whether or not a participant correctly answered similar questions before and after playing the game. If a participant was found to have answered more program comprehension questions correctly after playing the game, an improvement was considered to have occurred. If a participant was found to answer more questions incorrectly after playing the game, program comprehension was considered to have declined. The results show that, when using this approach to evaluate comprehension, 19 participants (56%) had a decline in program comprehension, with 7 (21%) showing improvement and 8 (24%) showing no change. Participants were found to have the most improvement regarding a question containing a simple calculation using variables.

We believe that two factors resulted in the observed decline in program comprehension:

Limited number of questions. The pre- and post-game questionnaires were set up so that the knowledge tested by a question was paired (i.e. Question #1 of the pre-game questionnaire tested similar knowledge as Question #1 of the post-game questionnaire). As study participants tend to favour answering fewer questions and to make the pairing manageable, only five questions were asked. Consequently, an incorrect answer in the post-game questionnaire has a significant impact on the results. We plan to have a wider variety and quantity of program comprehension questions in future studies.

Use of multiple programming languages To reinforce the program-language independent nature of Program Wars, the computer programs for each question were presented in one of four programming languages. However, it appears that this choice resulted in an unintended amount of confusion among the participants.

Although all programming comprehension questions presented “I don’t know” as a response, it is possible that some participants just guessed at the answer.

Another threat to the study is the use of Python as one of the programming languages. Originally, the user study was to be conducted early in the semester, before students in the course were exposed to Python. However, due to unanticipated delays, the study was conducted later in the semester around the time when students were being introduced to Python and Scratch. It is unclear how much this timing affected the results of the study.

We plan to restrict the questions to using pseudocode in future studies to reduce this confusion and to remove the risk of participants having a knowledge of the programming language used in the questions.

Table 2: Participants who correctly connected card types to programming language elements in an actual program.

Card Type	Participants
Grouping	11 (31%)
Variable	24 (67%)
Instructions	19 (53%)
Repetition	17 (47%)

4.2.2 Connection to Programming Concepts. Table 2 presents the results regarding how well the participants were able to connect the card types in Program Wars to elements of a program expressed in a programming language. As shown by the table, participants most easily made a connection with the concept of a variable but had a harder time identifying the concept of grouping. Participants were also able to easily match *Instruction* and *Repetition* cards to elements in a program. It was found that 3 participants (8%) were able to correctly connect all card types, 10 participants (28%) were able to connect three card types, 10 participants (28%) were able to connect two card types, 9 participants (25%) were able to connect one card type, and 4 participants (11%) were not able to make any connection between card types and program elements. In other words, nearly two-thirds of participants were able to make a connection between two or more of the programming concepts represented in the game and elements of an actual program.

However, as the Variable card is only used with the Repetition card in the game, it is possible that some participants may not have understood that these two concepts are independent of each other. Also, as different programming languages use different constructs and syntax for representing these concepts, presenting programs in a more programming-language-neutral (e.g. pseudocode) may better assess the participant's improvement or decline in comprehension. We plan to address these two items in our next study.

4.2.3 Player Feedback. Twenty-two participants (61%) provided feedback on Program Wars. Twelve participants commented that once they understood the game's mechanics (either by watching the video, reading the rules or playing their first game), the game was fun, engaging and easy to play.

Three participants expressed frustration with the random number generator when either they were dealt a hand without instruction cards and it took several turns to make progress, or they completed their program on one path and the other path was repeatedly selected, leading to the AI winning instead. One participant expressed frustration at how often the AI hacked their program. Finally, responses from five participants indicated misunderstandings with how a game ends.

5 IMPROVEMENTS TO PROGRAM WARS

Following the user study, we made two significant improvements to Program Wars, as well as a number of improvements to the user interface (e.g. player's progress) and usability (e.g. redraw hand) (labelled as "Version 2" in Figure 1). This section provides details for these two improvements.

5.1 Cyberattacks and Cybersecurity

A participant of the study commented that "*... once I got the hang of the basics, there wasn't much room to improve.*", a sentiment echoed by two others. Therefore, we added two new cyberattack cards and corresponding counters for these cards to provide more complex player interactions and gameplay strategies.

5.1.1 Cyberattacks. The two new cyberattack cards are *Malware* and *Power Outage*.

The *Malware* card represents the concept of a computer virus, Trojan horse, worm, ransomware, spyware, scareware or adware. The intent of the *Malware* card is to slow down an opponent's computer, hence the card's effect is to reduce an opponent's instruction total by 25% (i.e. the player's program is only 75% effective).

The *Power Outage* card represents the occasion when electricity stops flowing to a computer, such as someone tripping on the power cord or hitting a power switch, or a tree branch knocking down a power line. The effect of the *Power Outage* card is to prevent a player from playing any further cards on their playing areas.

Figure 1 shows an example of both of these cards. Note that the player's True portion of their program has been reduced from 33 to 25 ($\lceil 33 * 0.75 \rceil$). The player's False portion is also affected, however as the number of instructions is small there is no practical effect ($\lceil 2 * 0.75 \rceil = 2$). The player is currently also under the effect of a *Power Outage* and can only play either the *Hack* or *Malware* card from their hand, or redraw their entire hand.

5.1.2 Cybersecurity. To counter the cyberattacks, two types of cybersecurity cards are provided: safeties and remedies. The safety cards are *Firewall*, *Generator*, and *AntiVirus*. The remedy cards are *Battery Backup* and *Overclock*.

Safety cards protect a player from specific cyberattacks for the remainder of the game. The *Firewall* card protects a player's program from *Hack* cards, the *Generator* card protects the player from *Power Outage* cards, and the *AntiVirus* card protects the player from *Malware* cards. There are only one of each of these cards in the deck no matter the size of the deck. Figure 1 shows the player with a *Firewall* safety card.

Remedy cards either prevent or counter one cyberattack, depending on whether the card is played before or after the player is attacked. The *Battery Backup* card counters a *Power Outage*, and *Overclock* counters a *Malware* attack. *Overclock* also 'speeds up' a player's program by 25% (i.e. the player's total number of instructions is 125% of the total of their cards). Once a remedy card is used to counter an attack, it and the cyberattack card are discarded.

5.1.3 Expanded Scoring. The winning condition for a game is reaching a target number of instructions on the currently selected True or False path. We expanded the scoring for the game to reward players for practising what may be considered good programming practices (e.g. the use of *Group* cards to promote the concept of "refactoring" [10]).

The scoring for the game now also considers the size of the player's program (i.e. the number of instructions in both play areas) and the use of 'good programming practices'. Players are rewarded for their use of modular programming (i.e. *Group* cards), repetition and variables. The player is also rewarded for completing their program, good cybersecurity practices (i.e. the use of Safety cards and having a malware-free system), and not overheating their system (i.e. no overclocking). These bonuses reinforce the need for a player to make strategic decisions during the game.

6 CONCLUSION

This paper presented the web-based card game, Program Wars. Program Wars uses simple vocabulary for teaching the fundamental programming language concepts in a programming-language-independent manner. A pilot user study conducted on Program Wars showed that connecting game concepts to actual programming language elements were possible for most participants. However, due to the choice of having multiple programming languages and other confounding factors, it was unclear if the participants' ability to understand a computer program was improved.

REFERENCES

[1] J. H. Andrews. 2013. Killer App: A Eurogame about software quality. In *2013 26th International Conference on Software Engineering Education and Training (CSEE T)*. 319–323. https://doi.org/10.1109/CSEET.2013.6595269

[2] Ufuk Aydan, Murat Yilmaz, Paul M. Clarke, and Rory V. OâĂŽConnor. 2017. Teaching ISO/IEC 12207 software lifecycle processes: A serious game approach. *Computer Standards & Interfaces* 54 (2017), 129 – 138. https://doi.org/10.1016/j.csi.2016.11.014 Standards in Software Process Improvement and Capability Determination.

[3] Alex Baker, Emily Oh Navarro, and AndrÃľ 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.

[4] Timothy Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. Computer Science Unplugged: school students doing real computing without computers. 13 (01 2009).

[5] 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 (CHINZ '12)*. ACM, New York, NY, USA, 56–63. https://doi.org/10.1145/2379256.2379266

[6] Peter Drake and Kelvin Sung. 2011. Teaching Introductory Programming with Popular Board Games. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 619–624. https://doi.org/10.1145/1953163.1953338

[7] Anke Drappa and Jochen Ludewig. 2000. Simulation in Software Engineering Training. In *Proceedings of the 22Nd International Conference on Software Engineering (ICSE '00)*. ACM, New York, NY, USA, 199–208. https://doi.org/10.1145/337180.337203

[8] Joao F. Ferreira and Alexandra Mendes. 2014. The Magic of Algorithm Design and Analysis: Teaching Algorithmic Skills Using Magic Card Tricks. In *Proceedings of the 2014 Conference on Innovation &#38; Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, NY, USA, 75–80. https://doi.org/10.1145/2591708.2591745

[9] Charles Wesley Ford, Jr. and Steven Minsker. 2003. TREEZ - An Educational Data Structures Game. *J. Comput. Sci. Coll.* 18, 6 (June 2003), 180–185. http://dl.acm.org/citation.cfm?id=770818.770848

[10] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[11] Bradley S. Greenberg, John Sherry, Kenneth Lachlan, Kristen Lucas, and Amanda Holmstrom. 2010. Orientations to Video Games Among Gender and Age Groups. *Simulation & Gaming* 41, 2 (2010), 238–259. https://doi.org/10.1177/1046878108319930 arXiv:https://doi.org/10.1177/1046878108319930

[12] Nicolas Guibert, Patrick Girard, and Laurent Guittet. 2004. Example-based Programming: A Pertinent Visual Approach for Learning to Program. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*. ACM, New York, NY, USA, 358–361. https://doi.org/10.1145/989863.989924

[13] Lasse Hakulinen. 2011. Using Serious Games in Computer Science Education. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11)*. ACM, New York, NY, USA, 83–88. https://doi.org/10.1145/2094131.2094147

[14] Apurva Jain and B. Boehm. 2006. SimVBSE: Developing a Game for Value-Based Software Engineering. In *19th Conference on Software Engineering Education Training (CSEET'06)*. 103–114. https://doi.org/10.1109/CSEET.2006.31

[15] Jennifer Jenson, Suzanne de Castell, and Stephanie Fisher. 2007. Girls Playing Games: Rethinking Stereotypes. In *Proceedings of the 2007 Conference on Future Play (Future Play '07)*. ACM, New York, NY, USA, 9–16. https://doi.org/10.1145/1328202.1328205

[16] Kristen Lucas and John L. Sherry. 2004. Sex Differences in Video Game Play:: A Communication-Based Explanation. *Communication Research* 31, 5 (2004), 499–523. https://doi.org/10.1177/0093650204267930 arXiv:https://doi.org/10.1177/0093650204267930

[17] E. O. Navarro and A. v. d. Hoek. 2005. Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model. In *18th Conference on Software Engineering Education Training (CSEET'05)*. 25–32. https://doi.org/10.1109/CSEET.2005.16

[18] 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

[19] Elizabeth Suescun Monsalve, A.X. Pereira, and Vera Werneck. 2017. Teaching software engineering through a collaborative game. (01 2017), 874–895.

[20] Carnegie Mellon University. 2018. Alice.org. Online. http://www.alice.org

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

[22] Atif Waraich. 2004. Using Narrative As a Motivating Device to Teach Binary Arithmetic and Logic Gates. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '04)*. ACM, New York, NY, USA, 97–101. https://doi.org/10.1145/1007996.1008024