UPDATING A WEB-BASED CARD GAME TO TEACH PROGRAMMING, CYBERSECURITY AND SOFTWARE DEVELOPMENT LIFE CYCLE CONCEPTS

MD. HASAN TAREQUE
Bachelor of Science, Bangladesh University of Professionals (BUP), 2010

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

UPDATING A WEB-BASED CARD GAME TO TEACH PROGRAMMING, CYBERSECURITY AND SOFTWARE DEVELOPMENT LIFE CYCLE CONCEPTS

MD. HASAN TAREQUE

Date of Defence: August 12, 2021

Dedication

This thesis is dedicated to my family members

(My parents, Wife, Elder brother & sister)

Abstract

Game-Based Learning (GBL) has been shown to be effective in teaching software engineering practices and principles. This research updates Program Wars, a web-based card game, to improve the support for learning concepts of various programming structures and concepts (i.e. variables, loop, method). Additionally, the game's support for learning cybersecurity practices and concepts was refined. A user study evaluated this new version of Program Wars, and it was found that the latest version performs better in terms of learning various programming components along with cybersecurity concepts than the older version. Finally, a new gaming mode was introduced to the newest version of the game to teach the Software Development Life Cycle and the Iterative Software Development Methodology. A separate user study is also proposed in this research work to evaluate this version of the gameplay.

Acknowledgments

Firstly, I would like to thank Almighty ALLAH for giving me the opportunity and strength to complete this research and master's program.

I would like to express my deepest gratitude to my supervisor Dr. John Anvik. Without his guidance and encouragement, it would not be possible for me to finish the thesis work. Dr. Anvik offered his valuable suggestions and recommendations throughout my master's program. Thank you, Dr. Anvik, for your continued support and encouragement towards me.

I am very grateful to my M.Sc. supervisory committee members Dr. Wendy Osborn and Dr. Lorraine Beaudin, for their valuable feedback and time. I want to extend my thanks to my amazing 'Sibyl Lab' members for always being encouraging. Especially I would like to thank Steve Deutekom for his support.

I am extremely grateful to my parents for their love, prayers, and sacrifices. I would also like to thank my in-laws as well as my brother and sister for their continuous support.

I would like to express my love and gratitude to my wife (Neonta) for being with me and support me. Neonta, without you, it would not be possible for me to go through this journey.

I am grateful to the School of Graduate Studies for providing financial assistance for my graduate study and research work.

I would also like to thank my friends and well-wishers, especially Ahmed Shoeb Al Hasan, for the constant support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Successful software development projects depend a lot on software engineering (SE) education as the participants rely on different areas of software engineering knowledge. However, designing a curriculum covering all of the software engineering knowledge expected for industrial software development is cumbersome because it requires both knowledge and practical skills. For example, software developers need to learn about the project environment, demands by the stakeholders, and new technologies. Most importantly, successful software development depends on technical expertise along with applied conceptual SE knowledge. As a result, software engineering education needs to be designed in such a way that it will fulfill the changing requirements of the growing software development industry. By introducing different SE knowledge areas and industry requirements into SE education, students will be prepared for the future.

Experience-based software engineering education can play a vital role in terms of reaching this goal. Learning through different activities that combine social and educational aspects are more effective than standard curriculum-based education [1]. However, constraints such as time, proper tools and realistic environments introduce difficulties to a learning model. Game-Based Learning (GBL) has been used as an effective way of teaching concepts practically. The use of games for education can be divided into two techniques: gamification and serious games.

Games that focus on the properties of GBL should consider some basic elements like interaction. JP Gee mentioned in his research [2] that interaction is essential for a successful game design. The game should support the building of an interactive relationship between the player and the real world. Also, the game should have challenging contexts where a player can innovate various strategies to win against an opponent. J. Mcgonigal mentioned the idea of an ‘epic’ win while playing a game [3], where a gamer shows concern, optimism as well as surprise while competing against different opponents. N.Whitton [4] mentioned GBL should also focus on the learning context and outcomes; otherwise, there is no certainty that any engagement in the game will lead to subsequent engagement in learning.

Gamification, refers to the implementation and use of game design elements, usually in a non-game setting [5]. *Gamification* has been used for Knowledge-Based Learning (KBL), edutainment[1], corporate activities, administrative training, marketing research, and real-world project management skills [6]. Around 70% of the top global organizations use gamified applications for marketing and performance measures [7]. On the other hand, *Serious games* focus on the usefulness of the entire gameplay in the context of learning. For example, ProDec [8] uses the concept of the serious game, where participants can learn more about different phases of software project management.

Although there are many areas of software engineering research, learning the fundamental software engineering principles through the use of gameplay is an area that needs more attention. Introducing gameplay into a SE curriculum is not a new concept; rather, it is known to be a very effective one [9]. Nevertheless, this model has some restrictions like time constraints and lack of proper applications for demonstration.

This research work aims to explore further the effectiveness of teaching fundamental software engineering concepts through game-based learning. Specifically, Program Wars, a web-based card game for learning programming language and cybersecurity concepts, is extended in two ways. *First*, the teaching of basic programming and cybersecurity concepts are refined by introducing additional cards and modifying the User Interface (UI). *Second*,

[1]Learning through different activities and entertainment are known as “edutainment.”

basic software engineering concepts are introduced into the game by adding an alternative playing mode for the game.

The questions to be answered by this research are:

R.Q. # 1: Do the refinements to the UI and gameplay of Program Wars improve a player's understanding of basic computer programming concepts?

This Research Question (R.Q.) is divided into three (3) sub-questions to evaluate the learning outcomes more precisely.

(a) Do the refinements to the UI and gameplay of Program Wars improve a player's knowledge of the Variable concept?

(b) Do the refinements to the UI and gameplay of Program Wars improve a player's knowledge of the Loop concept?

(c) Do the refinements to the UI and gameplay of Program Wars improve a player's knowledge of the Method concept?

R.Q. # 2: Does the refinement of the cybersecurity aspects of Program Wars lead a player to better understand real-life cybersecurity threats and how to combat them?

R.Q. # 3: How can Program Wars be modified to teach the basics of the Software Development Life Cycle (SDLC) and the Iterative Software Development Methodology (ISDM)?

The contributions of this thesis are:

i. A new version of Program Wars for teaching programming and cybersecurity concepts. The latest version of the game (Program Wars *v*.2.0) has significant changes to the User Interface (UI) and gameplay. In particular, the card and gameplay for the programming concept of procedure/function/method were revised. Also, the prior cybersecurity cards were refined to better teach about specific cyberattacks.

ii. A user study, of the latest version of the game. This user study corrected some concerns with the prior user study. Also, the study assesses Program Wars ability to teach cybersecurity concepts, which was not part of the previous study.

iii. The latest version (Program Wars *v.*2.0) was expanded for teaching the Software Development Life Cycle (SDLC). The expanded version of the game introduces players to the Iterative Software Development Methodology (ISDM) by adding dedicated sprints into the gameplay.

This thesis is organized in the following manner. The background and related works are described in Chapter 2. A detailed description of the version of Program Wars that was modified (v 1.0) is given in Chapter 3 before describing how the game was changed to refine the teaching of basic programming language and cybersecurity concepts in Chapter 4. The user study used to answer **R.Q.# 1** and **R.Q.# 2**, including the analysis of the results, is described in Chapter 5. How Program Wars *v.*2.0 was modified to teach the SDLC and ISDM (used to the answer **R.Q.# 3**) are described in Chapter 6. The thesis is then concluded with some future research directions in Chapter 7.

Chapter 2

Related Work

Several research works have been conducted regarding the use of games to teach computer programming and software engineering aspects.

This chapter begins with an overview of game-based learning research in software engineering before discussing specific examples of the board, card, and web-based games for teaching software engineering.

2.1 Game-Based Learning research in SE

The following is an overview of the previous work regarding Game-Based Learning in the field of software engineering.

Mauricio et al. [10] identified a methodology that can be applied in different interactive games for Software Engineering (SE) Game-Based Learning. Also, they explored various primary studies related to SE education and identified the learning outcomes, and mapped those outcomes to different stages of SE projects. They divided the SE knowledge into eight (8) areas. Among those areas, Software Process (PRO) has thirty-two (32) relevant research works that used the Game-Based Learning (GBL) process.

Pieper et al. [11] presented a case study of the Software Engineering Method and Theory (SEMAT) to identify the educational outcomes in Digital Game-Based Learning. SEMAT is a part of the emerging OMG[2] standard [12]. The case study shows that evaluating a software development integration scenario can provide an in-depth analysis of the result.

[2]The Object Management Group (OMG) is a computer industry standards consortium.

Nevertheless, the data was not sufficient to reach a conclusive decision regarding the pattern of learning. As a result, SEMAT was not recommended to be a standard. The researchers state that if the study could be conducted in a broader spectrum (i.e. larger data-set), the result of the case study might have a more conclusive outcome.

Tao et al. [13] emphasize learning software engineering through different gaming approaches. They created `Pex4Fun` to serve both the social aspects of Game-Based Learning and the presentation of software engineering content. The learning outcome from the research work is that gaming and entertainment can be a source of education and that interactive learning has excellent value. Another discovery is that learning while playing can be effective in the industrial field.

Swapneel et al. [14] discussed how highly addictive socially optimized (HALO[3]) provides concentration through an adaptive environment. The game environment is developed in such a way so that the player can enjoy their work in a gaming atmosphere. The game is designed as a simple plugin to integrate with an IDE[4], such as Eclipse or Microsoft Visual Studio. Making the user comfortable with the working environment is one of the significant contributions of HALO. HALO adopted a context-switching-free environment using this technique. HALO is designed for the software industry, where the different projects have different requirements. Along with that, it supports various social aspects like teamwork and project management. The initial stage of the game is known as 'Quest', which is a preliminary introduction of the system. Here someone senior must work voluntarily or can be assigned. If the quest is more challenging for a single player, it could be done in a team, and then it is called a party. A simple task like use-cases or bug fixing can be part of quests. Another essential aspect is context switching. If an employer works on different modules, then it will require more time, but HALO groups the same type of coding environment altogether so that less context switching is required. In this process, a balanced

[3]The author suggested that there were no connections between the proposal and the game name.

[4]Integrated Development Environment (IDE) is known as a software application that provides comprehensive facilities to computer programmers for software development.

environment has been created where players progress through multiple difficulties. There was no evaluation technique for the HALO, which is one of the drawbacks of this research. This research work is a game-based approach in software engineering that is beneficial for both academic and industrial fields.

Miljanovic et al. [15] discussed Robobug, a debugging technique through gaming. Debugging is an essential skill in software development. Many good programmers are struggling to find the bugs in a code segment. This skill requires practice and patience, which might be difficult for a new programmer to adopt. Many computer science students struggle with debugging and feel left behind. Robobug presents different debugging techniques for different levels. It also has hints, which are very helpful for learning.

Szabo [16] proposed GameDevTycoon for teaching software engineering. Based on their work, there are six (6) major Software Quality Factors where GameDevTycoon performance was measured. The researchers classified Software Engineering Games into thirteen segments, where GameDevTycoon covers most of the criteria. Gameplay analysis and software process models are simultaneously in this game. Three primary stages of software development are taught through the gameplay. The initial stage is known as the garage, the second stage is called team management, and the final stage is known as world domination. Each stage has separate responsibilities; for example, in the initial stage in software development, one should focus on the quality and latest research. As the project grows, team building is an additional responsibility that needs to be handled. When the project is in the saturation stage, new research works should be promoted to cultivate new technology. Again some small but significant issues regarding team bonding and employee workload also need to be adequately addressed.

Uskov et al. [6] discussed the different classifications of gameplay and their impact on several learning criteria. According to the researchers, gamification is a growing area in the business industry. The researchers emphasized the SWOT (Strengths, Weaknesses, Opportunities, Threats) framework [17] to find the learning criteria. However, they found that creating a software engineering game-based engine is a challenging task, which programmers and industry should prioritize.

2.2 Games for Learning Computer Programming

Several games have been created to teach the basics of programming. However, most of them are platform-oriented (i.e. only work on Windows) or target a specific programming language (e.g. C++ or Java). For learning the basics regarding computer programming, it is essential to understand the underlying logic and conceptual ideas. In this section, several types of games used to teach programming basics, especially board games, card games, desktop-based games and web-based games, are presented.

2.2.1 Board Games

Board games are a part of tabletop games where different components of the game can be moved on an adjacent surface or board according to some predefined rules [18]. The following are a couple of board games, which teach different aspects of computer programming.

Battle Bots v2 [19] was inspired by the Robo Rally [20] game. The required number of players for the game is between two and twenty. In the middle of the board, there is a repair center. Each player will start the game with twenty-two cards. Each move is divided into two groups: the programming round and the action round. In the programming round, players have to play the movement cards. In this stage, no action card can be played. There are five phases where different action cards can be played in action round, along with movement cards. The player who survives the match wins the game. The main advantage of the game is that the players have to plan about the moves; as in programming, it is essential to set the goal. A player can design their game plan depending on the other players' moves or personal requirements. As a result, a fair amount of calculation is needed, which is very important for the logical aspect of computer programming. The game has no actual programming interface, which is one of the drawbacks of the game.

Robot Turtles [21] is a game to teach kids to code by using simple direction cards to move a specific coloured turtle. The game's main focus is to get the coloured turtle to the same coloured jewel piece on the board. There are three instructions: forwards, rotate 90 degrees left and rotate 90 degrees right. There are also some obstacles where players might have to use different techniques to overcome the situation. Some unique cards are also introduced, like fire cards for shooting. The higher the level, the more complex cards need to be used. A game setup of Robot Turtles is presented in Figure 2.1.[5]

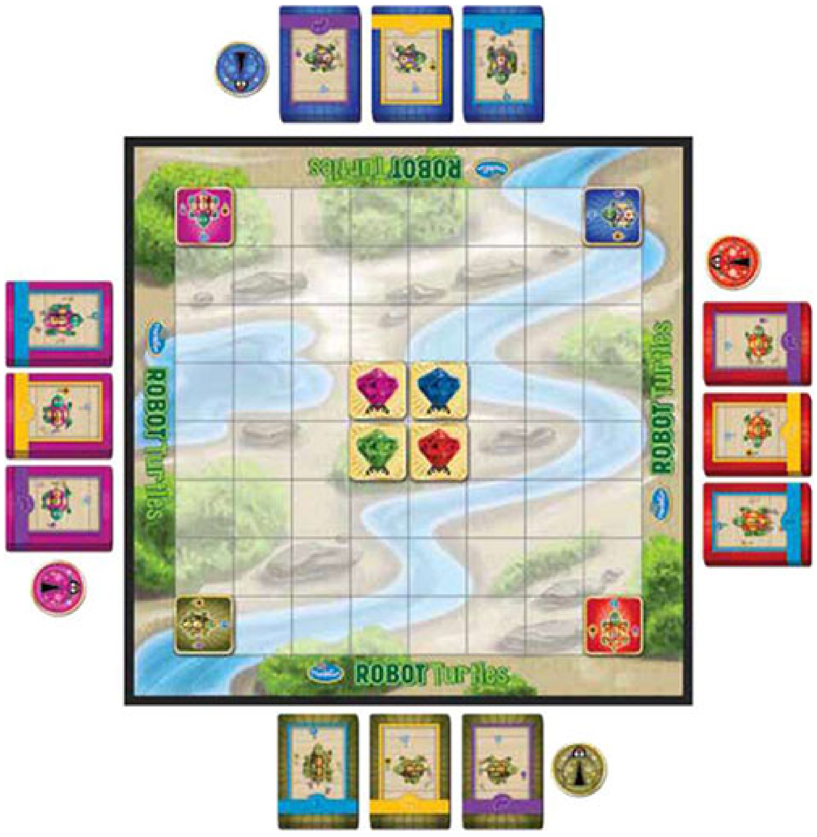


Figure 2.1: Robot Turtles gameplay

Regarding programming, the jump card is the most crucial one. The idea of jump cards is to replace a set of instruction cards that can then be used repeatedly. There is also a bug card that can undo the immediate move the player has made. Robot Turtles is designed for kids' initial steps in programming, like single statement compilation, which is well

[5]Image downloaded from: https://www.ultraboardgames.com/robot-turtles/game-rules.php in July 2021

executed through a single card movement. The introduction of a jump card is an excellent initial concept of function calls. Bug cards introduce the idea of mistakes in programs.

Code Master [22] is a single-person puzzle game that teaches logical problem-solving. There are sixty levels in the game with several challenging levels. The game consists of a map with six different levels ranging from easy (green) to expert (red). There is also a guide scroll that indicates the order in which the program statements should run. Also, the guide has places for conditional tokens. There are three colours for paths: green, blue and red. For each path, there is a specific token related to the colours. Numerical numbers are represented in the map nodes. The goal is to get to a portal and collect crystals using a limited path and number of moves. A game setup of Code Master is presented in Figure 2.2.[6]

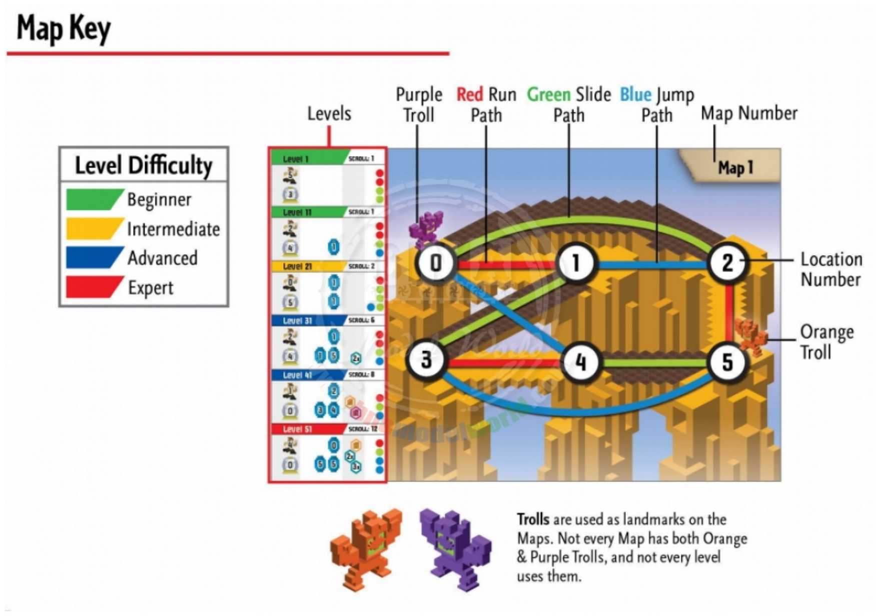


Figure 2.2: Code Master gameplay

[6]Image downloaded from: https://www.bnamodelworld.com/educational-toys-thinkfun-tn1950 in July 2021

The game focuses on the programming construct of conditional statements (i.e. if-else). Some nodes have a self-loop, which illustrates the concept of a repeat statement. Finally, the shortest path selection is also a focus point. The game requires critical thinking before each move, which is vital for developing logical thinking. Unidirectional and bi-directional edges in a graph can be learned by playing the game.

2.2.2 Card Games

A card game can be defined as a game that involves different playing cards as the main components with which the game can be played [23].

Potato Pirates [24] was the only physical card game found that intends to teach programming concepts. The game's main objective is to make the user familiar with different programming concepts through social interactions. The game's final goal is to collect seven specific cards, named Potato King, through different logical actions performed through different cards. The game's programming logic is presented in Figure 2.3.[7]

A player acquires all seven Potato Kings by drawing them from the deck or by eliminating other players' ships and seizing their cards to win the game. Players can power up their attacks with programming concepts cards such as loops and conditionals. The game covers the concepts of programmings such as variables, functions, while-loops, if-else conditionals and nested-loops. Some surprise cards serve the purpose of interrupts and control flow. The game illustrates the branching condition very nicely, which is equivalent to the if-else statement. The use of variables is well defined. Different repeat structures are also presented reasonably. The game covers the loop structure of programming languages. There is also an option for creating a loop inside a loop known as a nested loop. The game also covers the case structure (switch case or nested if-else).

[7]Image downloaded from: https://www.toytag.com/products/potato-pirates in July 2021

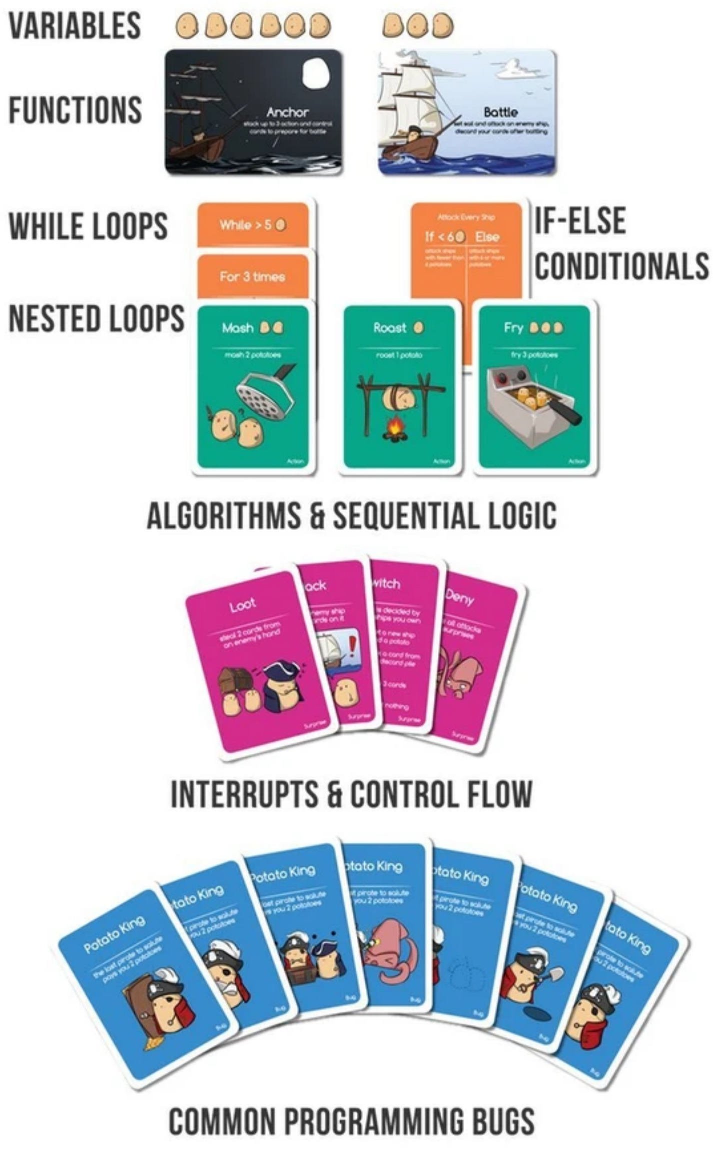
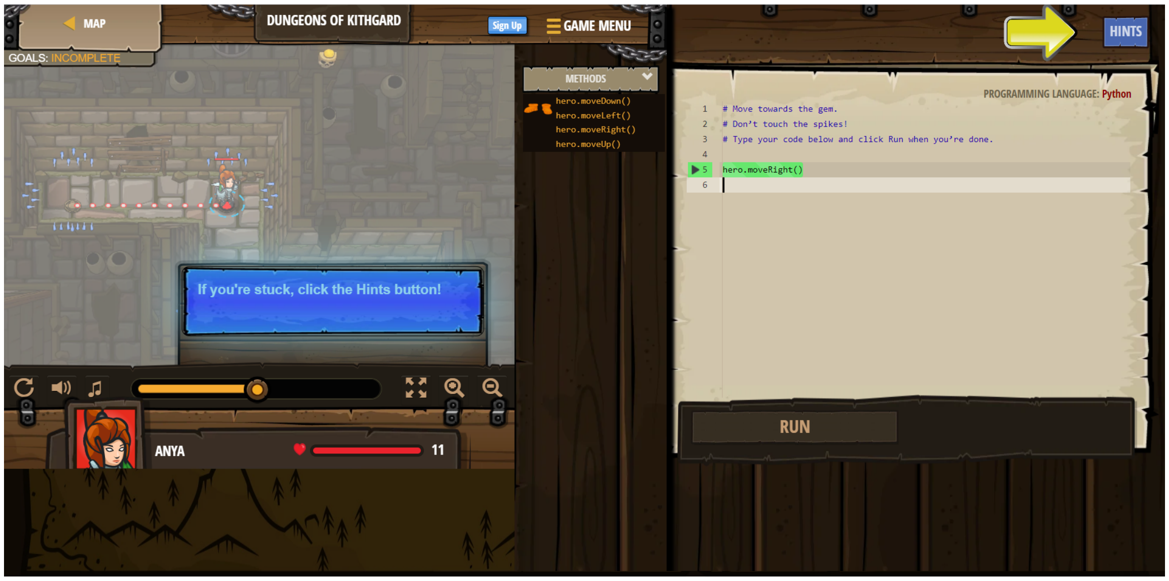


Figure 2.3: Potato Pirates programming logic.

2.2.3 Web-based Games

Games that can be played on the World Wide Web are known as web-based games. These games use standard web technologies or browser plugins [25].

CodeCombat [26] is a web-based game focusing on learning JavaScript, Python, and other programming languages. The game is built upon concepts of swords-and-sorcery, where the player has to role-play as a warrior. The game's main objective is to learn necessary programming skills by overcoming different challenges like clearing the maze, picking up gems, and avoiding any spikes or attacking ogres. The gameplay is split between a code editor on the right and a simulation effect on the left. Figure 2.4[8] shows the interface of the game. The avatar is controlled by using a set of commands. At each level, the player has to accomplish a set of tasks. As the game progresses, the player is gradually introduced to new concepts like loops, conditionals, and variables. If a person does not have programming experience, CodeCombat is very suitable. As the game progresses, the tasks involve more complex programming concepts. Most importantly, the levels themselves become more complicated due to possible interactions with the objects in the game world.



```
hero.moveDown()
hero.moveLeft()
hero.moveRight()
hero.moveUp()
```

```
# Move towards the gem.
# Don't touch the spikes!
# Type your code below and click Run when you're done.

hero.moveRight()
```

Figure 2.4: CodeCombat gameplay.

Blocky maze [27] is a game that introduces the concept of programming loops and conditions in Javascript without writing any Javascript code. The game is a combination of levels that teach programming. It is designed for children who have not had prior experience with computer programming. The game uses a graphical programming language

[8]Image downloaded from: https://www.pcmag.com/reviews/codecombat in July 2021

implemented in JavaScript, which can compile to JavaScript, Dart or Python. In the game, programming is done by dragging and dropping code blocks onto a design surface. Figure 2.5[9] shows the interface of the game. The main objective of the game is to take the avatar from a starting point to the endpoint. There are a total of ten levels, and the complexity increases as the game progress. Some goals are set to solve the maze in a particular number of steps, which is also challenging. The game interface is similar to a Google Map, which might help users adopt the game.

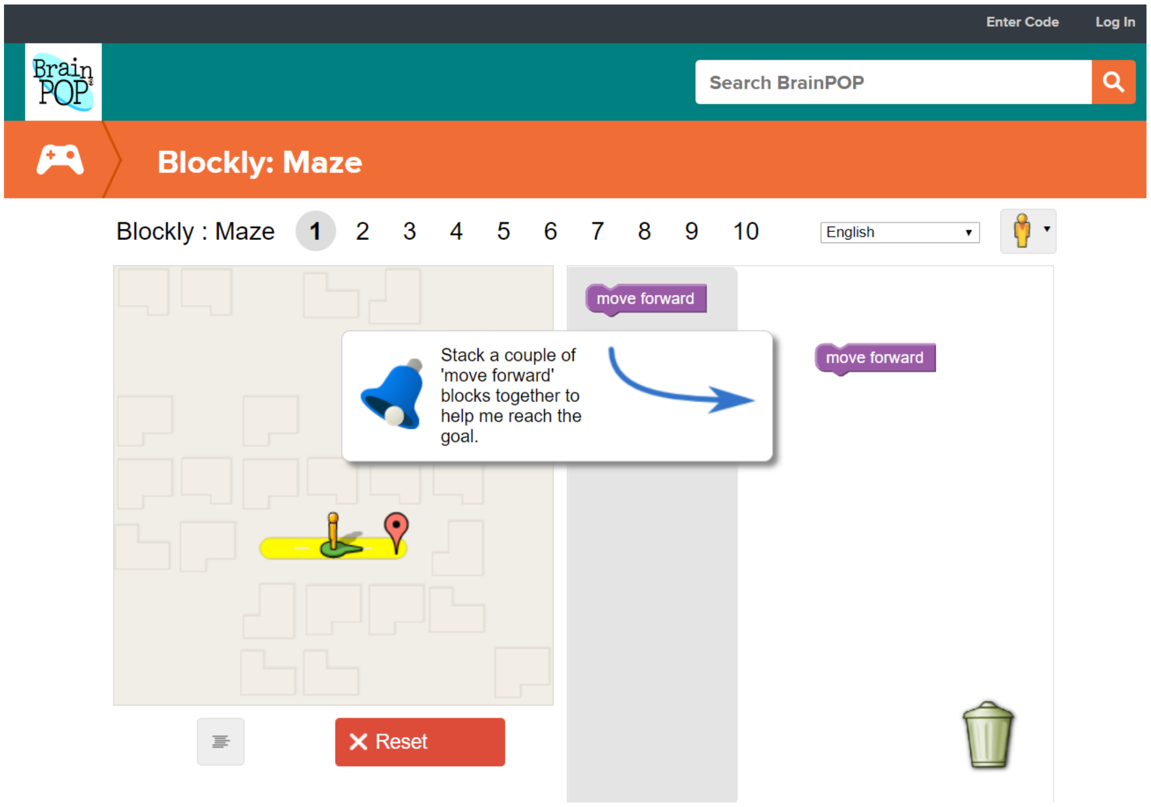


Figure 2.5: Blockly Maze gameplay.

Kodable [28] is a game developed for kids that focuses on introducing logic and the decisions (i.e. if-then-else) in computer programming. Kodable's programming language introduces players to step-by-step statements with different instructions used in the programming language. The primary programming concepts of conditional statements and loop structure are well defined in the game. The game's social aspect is presented in a par-

[9]Image downloaded from: https://s4scoding.com/images/google-blockly.jpg in July 2021

ents section with written teaching instructions that assist the parent in unlocking different levels for kids and extending logic skills into real life. The game is comprised of many features that are appropriate for children. The game's activities help the player think like a programmer, solve different problems, and eventually write real code using the game's custom coding interface. Figure 2.6[10] shows the interface of the game. For logic development, the game focus on learning the importance of statement sequence. The game introduces kids to different data types, such as integers and strings, and data structures, such as arrays. Finally, object-oriented programming concepts are also introduced by the game.

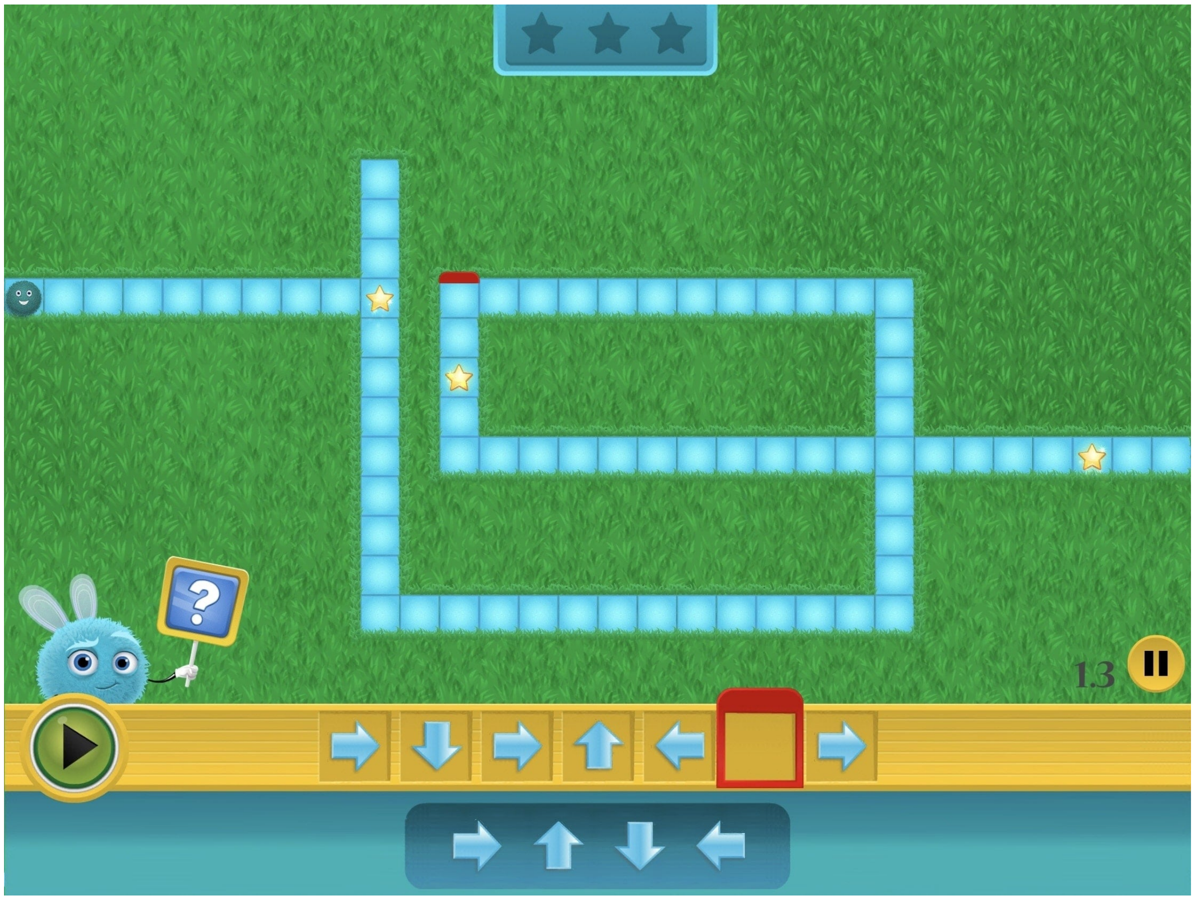
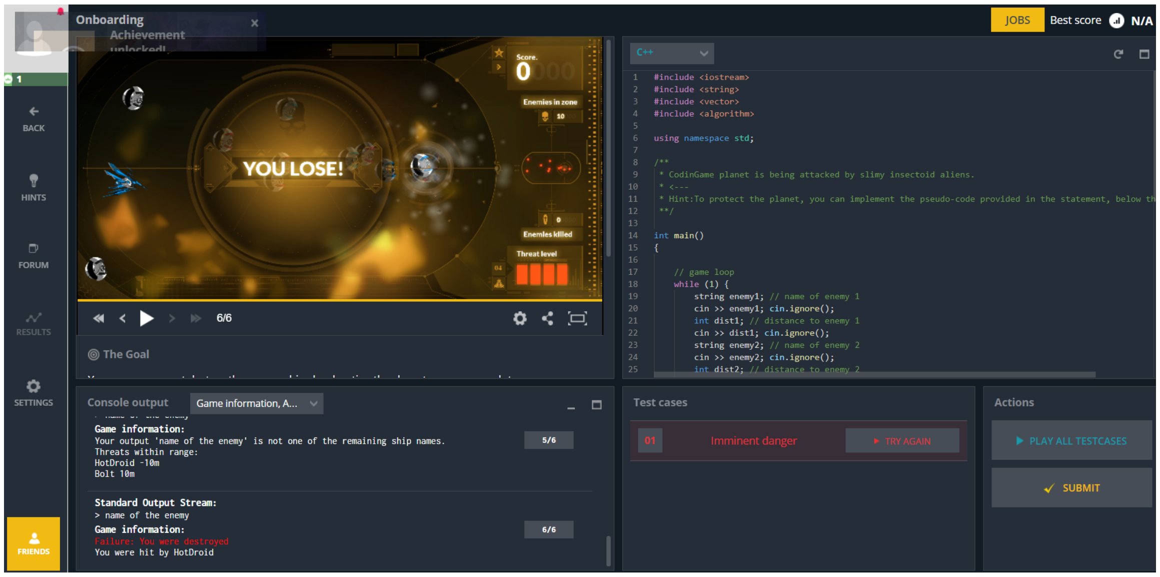


Figure 2.6: Kodable gameplay.

CodinGame [29] supports many programming languages. The game's main objective is to improve players' coding skills by solving different problems, applying new strategies, and getting inspired by other strong opponents. The game can be played in single and

[10]Image downloaded from: https://www.kodable.com/ in July 2021

multiplayer mode, which gives it the feeling of fun rather than learning. A player can choose any programming language among more than twenty, such as Python, Ruby, Java, and Scala. The targeted group for the game is the people who have basic programming knowledge and expert developers. In the game, users create a profile, which is used for challenges and contests. There are opportunities for players to make their profile public so that employers can find them to offer them a job. Also, the game has a forum for members to chat about languages, questions and share information. The game is not for beginners because it requires some basic knowledge of programming. Figure 2.7[11] shows the interface of the game.



```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

/**
 * CodinGame planet is being attacked by slimy insectoid aliens.
 * <---
 * Hint:To protect the planet, you can implement the pseudo-code provided in the statement, below th
 **/

int main()
{

    // game loop
    while (1) {
        string enemy1; // name of enemy 1
        cin >> enemy1; cin.ignore();
        int dist1; // distance to enemy 1
        cin >> dist1; cin.ignore();
        string enemy2; // name of enemy 2
        cin >> enemy2; cin.ignore();
        int dist2; // distance to enemy 2
```

```
Game information:
Your output 'name of the enemy' is not one of the remaining ship names.
Threats within range:
HotDroid -10m
Bolt 10m

Standard Output Stream:
> name of the enemy
Game information:
Failure: You were destroyed
You were hit by HotDroid
```

Figure 2.7: CodinGame gameplay.

Program Wars is a web-based card game. The game focuses on learning programming and cybersecurity concepts. The player's objective is to achieve a specific number of points through different cards. The game is programming language-independent, which means that it is not focused on any single programming language. Instead, the game helps to develop an understanding of the underlying logic of programming. Here the players do not have to be concerned about syntax errors, and programming terms are described using an elementary vocabulary. In Chapter 3, a more detailed analysis regarding Program Wars is

[11]Image downloaded from: https://www.codingame.com/start in July 2021

presented.

2.3 Games for Learning Cybersecurity

There are a couple of games that teach the basics of cybersecurity concepts. In this section, such games are described.

Cyber Threat Defender [30] is a collectable cybersecurity card game. The game's goal is to build a network as quickly as possible so that it can do more business and gain more points. While doing so, a player has to remember to defend their network because the opponent is going to try and disrupt other player's systems and networks. Figure 2.8[12] shows an example card from the game.

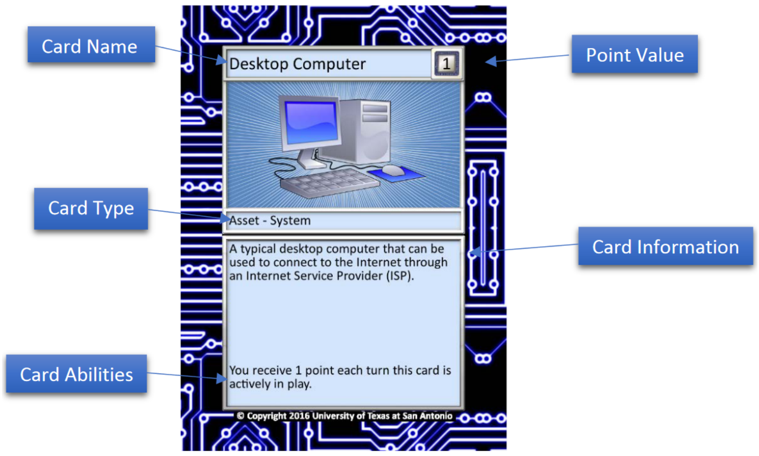

Card Name
Point Value
Desktop Computer
1
Card Type
Asset - System
Card Information
A typical desktop computer that can be used to connect to the Internet through an Internet Service Provider (ISP).
Card Abilities
You receive 1 point each turn this card is actively in play.
© Copyright 2016 University of Texas at San Antonio

Figure 2.8: Cyber Threat Defender card details

There is a defence for every attack, and for every defence, there is a corresponding attack to get around it. The player with the complete set of security defences will be the one who is able to protect the critical systems and emerge victoriously. For example, adding a wireless router without encryption will make the network vulnerable to attackers in the gameplay. For a player to successfully defeat their opponent, they must develop and implement a strategy for expanding and protecting their network. The main objective of the game is to make players familiar with basic and complex cybersecurity concepts.

[12]Image downloaded from: https://cias.utsa.edu/ctd_card_list.php in August 2021

Potato Pirates 2: Enter The Spudnet [31] is a serious game that various concepts of cybersecurity. In the game, a player needs to play across a network of shipping ports. Players must fulfill their five potato orders while playing ability cards to benefit themselves or harm others' shipments or structures. The board provides an analogy of a network map. The map is organized into interconnected, coloured networks (shipping lanes) of nodes (ports), each with its own IP address. The map is immediately recognizable as a network diagram with a wink to its pirating theme. Players can place firewalls, blocking travel to others, and play cards like Trojans, Ransomware, and so many more. There are 40 of these ability cards, with a cyber explanation for each in the clear and concise manual. Figure 2.9[13] shows the basic networking concepts into the game.

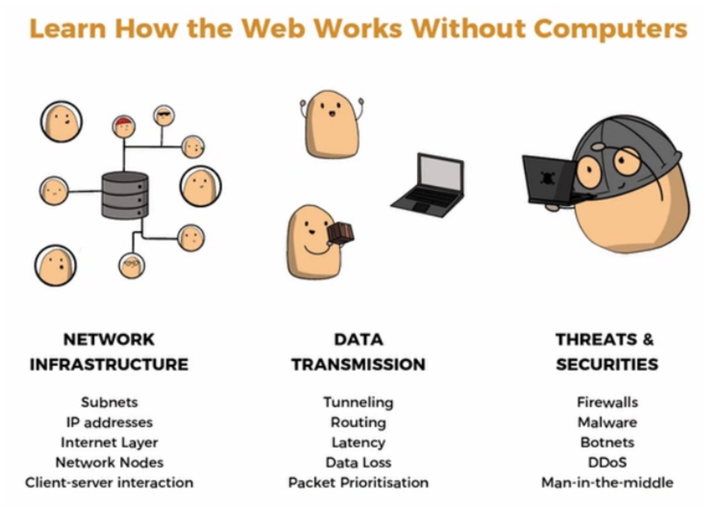


Figure 2.9: Potato Pirates 2: Enter The Spudnet gameplay

The gameplay can be competitive or cooperative, with each game style giving rise to its own strategies and approaches. As players move their potatoes (i.e. data packets) across the shipping network, they will face various network hindrances such as navigating inconvenient firewalls and frustrating connection slowdowns when warehouse nodes get overloaded. The game illustrates the concepts and introduces various cybersecurity-related concepts.

[13]Image downloaded from: https://potatopirates.game/products/enter-the-spudnet-board-game in August 2021

2.4 Games for Learning Software Development

There are a couple of games that teach the basics of the software development process. In this section, such games are described.

The Scrum Card Game [32] supports learners of the agile framework. Players act in a collaborative team environment during gameplay. Multiple teams play against each other. The game's objective is that a team of players plan and work on a product's abstract working packages aiming to complete as many working packages as possible.

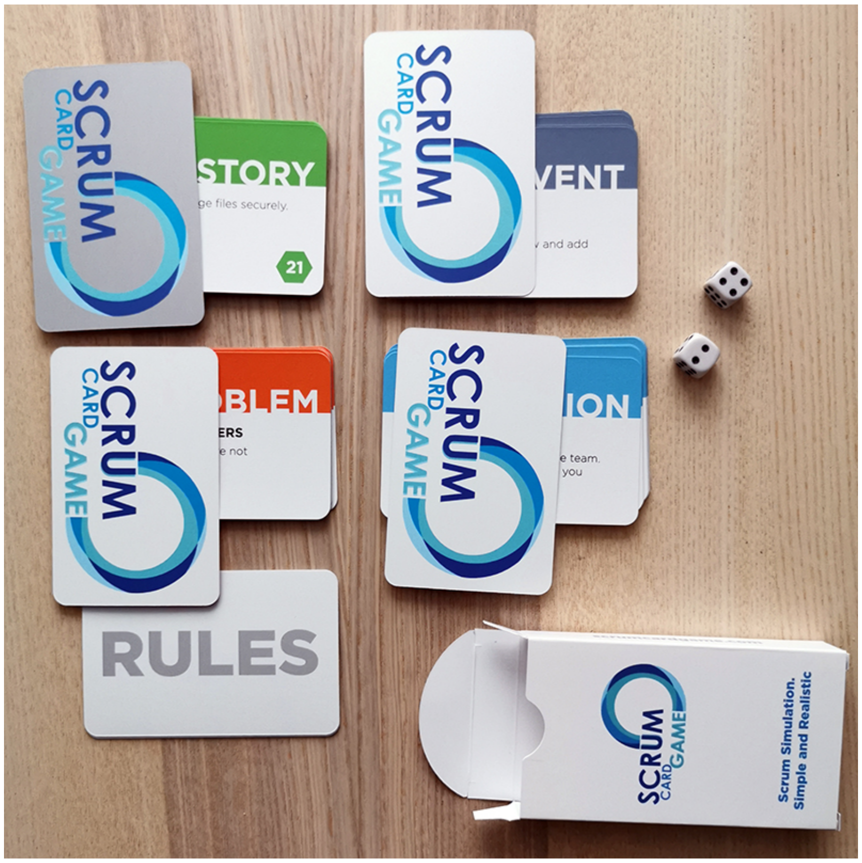


Figure 2.10: Scrum card game

The card deck of the Scrum Card Game consists of four types of cards: Story, Event, Problem and Solution. The Story cards describe features that the team wants to create for a product. Event cards deal with a positive, neutral, or negative impact on gameplay and progress. Problem cards describe permanent obstacles on planned Stories. Finally, Solution cards help players with Problems that occur. Figure 2.10[14] shows the setup of the game. A player starts the game by rolling two dice, indicating the working hours per day spent on a working package. Every three simulated days (which is called a Scrum Sprint), the players re-plan their work in an agile way. The game ends after three Scrum Sprints. The game is used for player to experience work in a simulated SCRUM sprint scenario. It allows reflection of many aspects and topics that happen in real life while using Scrum as a team.

The card game PlayScrum [33] covers learning the Scrum agile method. The game represents a first attempt at using a physical card game to teach students about the Scrum agile method. PlayScrum addresses many of the weaknesses of more traditional learning approaches and brings additional benefits in the form of face-to-face learning and enjoyable play. It claims to be the successor of Problems and Programmers[34]. Instead, each player slips into the role of a Scrum Master following the practices of Scrum. A die is used to determine the number of cards drawn from a card pile on a player's turn. The team working on open tasks from a Backlog is represented abstractly by so-called developer cards the Scrum Master controls. This means that the team experience different roles while working in collaboration.

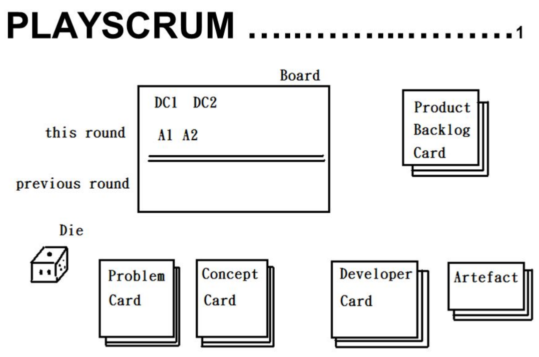


Figure 2.11: PlayScrum card game

The results of the research show that some PlayScrum cards can be improved. Problem

[14]Image downloaded from: https://agilelean.pro/en/scrum-card-game/ in August 2021

and concept cards can be more explicit in their scope. Also, there is a couple of areas where the game does not cover Scrum's features. Figure 2.11[15] shows an example of the game setup.

2.5 Summary

Based on the related research works examined, practical implementation (i.e. learning an actual programming language) and visualization of program results are essential in software engineering education. Also, learning through gameplay has more sustainable effects [35]. The main challenge of Game-Based Learning for software engineering is to maintain a balance between learning and engagement [36]. The gameplay should be presented in a structured way such that the user can relate the outcome to real-world programming. Most of the research deals with a specific part of software engineering like debugging, the user interface, and project management.

[15]Image downloaded from: https://slideplayer.com/slide/4429957/ in August 2021

Chapter 3

Program Wars 1.0

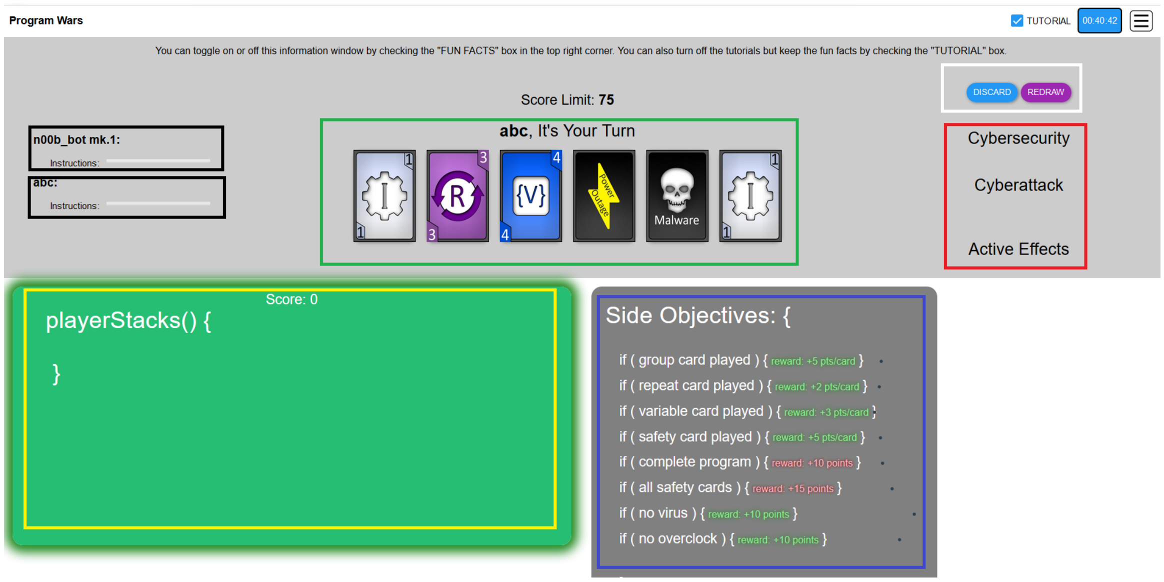
Among the games studied, Program Wars [37] was chosen as the most suitable for integrating software engineering concepts as it is an ongoing research project focused on the learning of the basics of computer programming. Learning basic computer programming has two main obstacles. *First*, the associated syntax issues can frustrate novice learners. For example, in a programming language like C, a programmer needs to declare a data type before the variable name (e.g. `int` varname). Different programming languages have distinct notations and methods for representing the computer program. As a result, a beginner might find it challenging to understand all the notations and feel less motivated to continue learning. *Second*, logical understanding is an essential factor in learning. Yet, different logical concepts are represented differently in various programming languages. This can make it hard for a novice programmer to understand. For example, the `if-else` statement has a different notation for programming languages C and Python. Finally, sometimes it is difficult to relate the programming language terminology with real-world software development scenarios.

Program Wars is a programming-language independent game. So the understanding of programming concepts a player develops by playing the game can be transferred to any programming language. The game is designed in such a way that the players need not be worried about syntactical issues. It does this by adopting a card-game modality, meaning that a player can not play a card unless it is valid. This ensures the ‘program’ created by the player is free from syntax errors. The game has straightforward rules. However, the user study for v.1.0 showed that some of the users found it complex.

3.1 Overview of Program Wars 1.0

The game can be played as a “hot-seat” (i.e. two human players taking turns at a same computer), or the player can choose to play against a simple AI. The player who reaches the target score first wins the game. The target points range from 35 to 105, which changes the duration of the gameplay.

The game starts with each player having six cards in their hand. On every turn, the player receives a new card randomly from the deck. Each player builds stacks of instruction cards and possibly cards that multiply the value of the instruction card (i.e. repetition cards) to obtain points. The total points are calculated by the cumulative sum of the number of instructions that all of the player’s stacks have.



You can toggle on or off this information window by checking the "FUN FACTS" box in the top right corner. You can also turn off the tutorials but keep the fun facts by checking the "TUTORIAL" box.

```
playerStacks() {

}
```

```
Side Objectives: {

if ( group card played ) { reward: +5 pts/card }
if ( repeat card played ) { reward: +2 pts/card }
if ( variable card played ) { reward: +3 pts/card }
if ( safety card played ) { reward: +5 pts/card }
if ( complete program ) { reward: +10 points }
if ( all safety cards ) { reward: +15 points }
if ( no virus ) { reward: +10 points }
if ( no overclock ) { reward: +10 points }
```

Figure 3.1: Play area of Program Wars *v.*1.0

Figure 3.1 shows the interface for Program Wars with additional coloured squares to highlight the different areas of the game. The black outlines show where the player’s progress towards the target score is given. The white outline shows the ‘Redraw’ and ‘Discard Hand’ buttons which can be used during a player’s turn. The green outline presents the current hand of the player, which consists of six cards. The yellow outline shows where the current player can create instruction stacks. The player can gain bonus points by completing objectives that are outlined in the blue frame. In addition to instructions cards, there are cards used to hinder an opponent's progress. These represent cyberattacks on the player's program. Finally, there are active effect cards, which can boost the total of the player's stacks or protect a player's stacks from attack. The red outline highlights the area where these will appear.

During a player's turn, each player builds their stacks using different `Instruction` cards, `Group` cards, `Repeat` cards and `Variable` cards. The player can also launch a cyberattack at an opponent or prepare a defence against a cyberattack. The following items describe the card types in the game and how they relate to computer programming. Figure 3.2 shows different programming related cards in Program Wars *v.*1.0 .

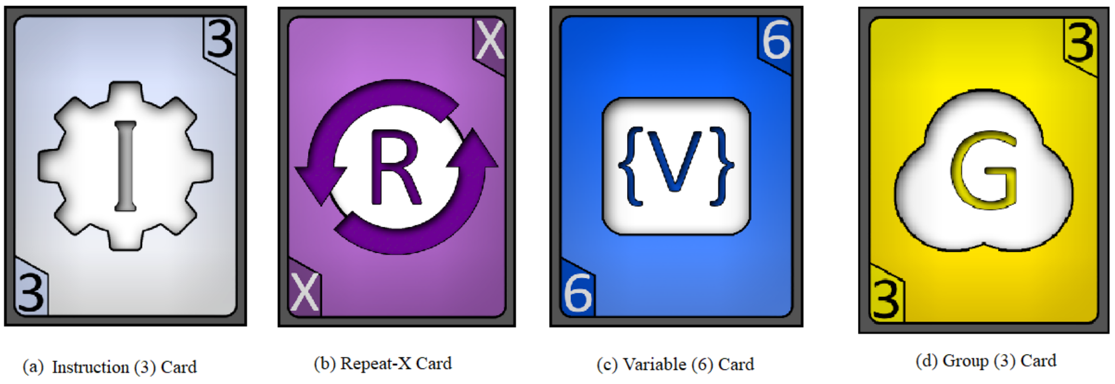
- **Instruction cards** – In the game, the `Instruction` cards have fixed values (1, 2, or 3). Each player will develop their stacks using the `Instruction` cards. The values of the `Instruction` card are added to the total points of the player.

  Relation to computer programming: Instruction statements are the most fundamental concept of a programming language. To execute any instruction, a programmer has to write a statement.

- **Variable cards** – `Variable` cards represent the concept of a variable in a programming language. Placing a `Variable` card on a `Repeat-X` card increases its multiplicative power. The `Variable` cards have values of 3 through 6 inclusive.

  Relation to computer programming: In a programming language, a variable's value can be changed.

- **Repeat cards** – This card increases the value of the instruction by multiplying its effect. There are three sizes of `Repeat` cards: 2, 3, and 4. A loop inside a loop (i.e. nested loop) can be created if one `Repeat` card is placed on top of another. There is



(a) Instruction (3) Card (b) Repeat-X Card (c) Variable (6) Card (d) Group (3) Card

Figure 3.2: Program Wars *v.*1.0 programming related cards

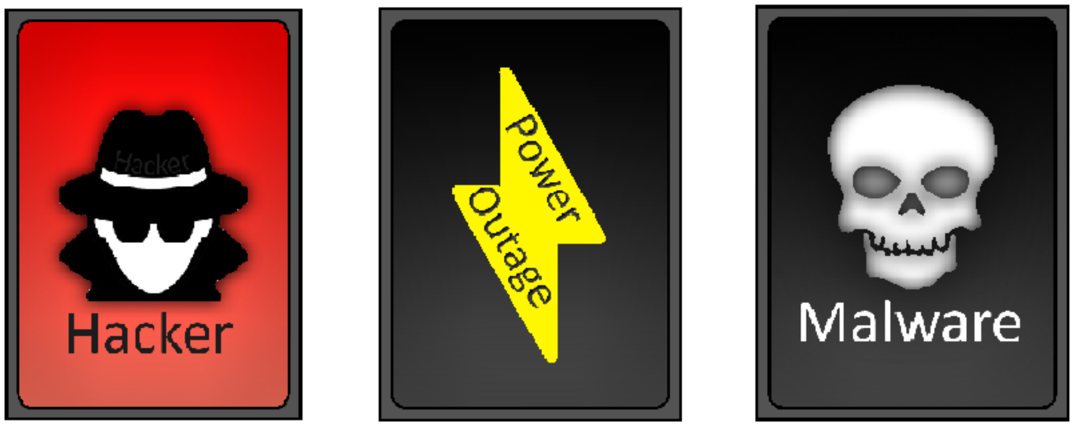
also a concept of variable-sized loops using a special `Repeat` card (called `Repeat-X`). A `Repeat-X` card does not affect an `Instruction` card until it combines with a `Variable` card.

Relation to computer programming: `Repeat` cards symbolize the loop concept in computer programming. If a statement or group of statements needs to be repeated in a programming language, a loop structure is used.

- **Group cards** – The player can use a `Group` card to group a collection of instructions (i.e. multiple cards) or a single instruction. The cards have fixed values of 2 through 6. These cards protect a portion of the player's program from a `Hack` card, which is a cyberattack.

  Relation to computer programming: By their use, players are indirectly introduced to the software engineering practice of modular decomposition and/or source code refactoring [38].

- **Cyberattack cards** – Cyberattack cards provide a social aspect to the game between players. In Program Wars *v.*1.0 , three Cyberattack cards are introduced: `Hack`, `Power Outage` and `Malware`. The `Hack` card allows a player to remove a stack of their opponent's cards that are not contained in a `Group` card. The `Power Outage` card prevents an opponent from playing cards until they can "restore power," and the `Malware` card reduces the efficiency of the opponent by reducing total score 25%. Figure 3.3 shows different cyberattack related cards in Program Wars *v.*1.0 .



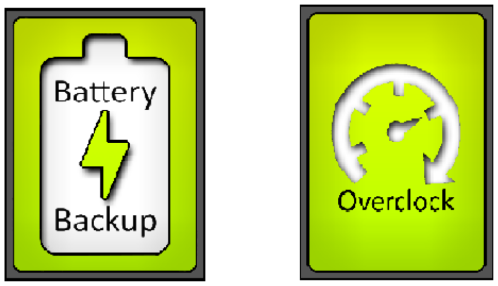
(a) Hacking Card (b) Power Outage Card (c) Malware Card

Figure 3.3: Different cyberattack related cards in Program Wars *v.*1.0

Relation to real-world Cyberattack: Increasingly, cybersecurity plays a vital role in software engineering. In the real world, malware reduces the efficiency of the affected machine by slowing computer or web browser speeds. Hacking is an attack on a computer program that changes the program in some way. A power outage disrupts the running of software as the computer is not able to function.

- **Remedy cards** – To counteract cyberattacks, there are two types of cards. One is the `Battery Backup` card, and another is the `Overclock` card. `Battery Backup` cards counter the `Power Outage` card effect by representing the power to the player's CPU. For a performance increase, a player can play the `Overclock` card, which simulates increasing a CPU's computational efficiency by increasing total score 25%. This card is used to counteract the `Malware` card. Also, by playing these cards, the player can receive some end-of-game bonus points. Figure 3.4 shows the two different remedy cards in Program Wars *v.*1.0 .

  Relation to real-world performance: CPU overclocking can be performed when more efficiency is required from the CPU. The `Battery Backup` cards represent the

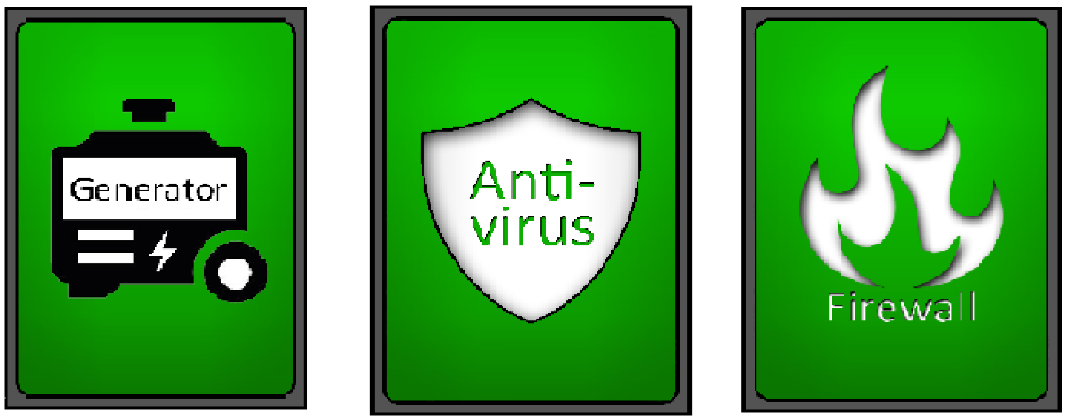


(a) Battery Backup Card (b) Overclock Card

Figure 3.4: Different remedy related cards in Program Wars *v.*1.0

idea of an uninterruptible power supply (a.k.a. UPS) in the real world.

- **Safety cards** – There are three cards in this category. A `Generator` card is used by a player to prevent or stop an opponent from playing a `Power Outage` card. The `Antivirus` card protects against a malware attack, and the `Firewall` card prevents a player's program from being 'hacked.' Figure 3.5 shows the different safety cards in Program Wars *v.*1.0 .



(a) Generator Card (b) Antivirus Card (c) Firewall Card

Figure 3.5: Different safety related cards in Program Wars *v.*1.0

Relation to cyberdefense: In the real world, an antivirus program protects a user from malware attacks represented in the game by an `Antivirus` card. Either a hardware or software firewall is used to protect against a computer hacking attack. The generator is usually used to prevent power failure.

3.2 Analysis of Program Wars *v.*1.0

The following is an assessment of the strengths and weaknesses of Program Wars.

3.2.1 Strengths

In many ways, Program Wars is better than many previous games for teaching programming. The following are the identified strengths of the game.

Conditional Statements Conditional statements are decision points in a program that control the flow of execution. Examples of conditional statements include `if-then`, `if-then-else` and `switch-case`. In Program Wars, conditional statements are represented as `goals` (the items in the blue box of Figure 3.1). A player can relate conditional statements in a programming language with this game element.

Repeat Statements Repeat statements occur when a program executes a certain set of instructions multiple times, which is also known as a `loop-structure`. Examples of repeat statements in programming languages includes `for`, `while`, `do-while` and `jump`. In Program Wars, repeat statements are implemented with a `Repeat` card where the repetition depends on the card, or in the case of `Repeat-X` , a `Variable` card. In Program Wars the `Repeat` card multiplies the score of an `Instruction` card, giving the player an idea of how loops work in real programs.

Method/Function structure In a programming language, a method (function) is a block of organized, reusable code that is used to perform a single, related action. The concept of the Method/Function structure in a programming language is to reduce the redundancy of statements of a program. Usually, the body of the function (i.e. the statements of a function) needs to be defined in a separate portion of a program, and the specific function needs to be called from the main program. Program Wars represents the function structure with `Group` cards. By using a `Group` card, multiple statements (i.e. `Instruction` cards) can be presented as a function. Moreover, if a `Group` card has been applied, the sequence of instructions is protected from various cyberattacks. Like in a real-word function/method, the details (i.e. Instructions) are hidden by the card and only the cumulative effect is presented.

Variables concept In programming languages, a variable is used to store values in the computer's memory. There are different types of variable like `int`, `float`, `char`, etc. In the game, the `Variable` card is associated with the `Repeat-X` card. The combined effect of `Repeat-X` and `Variable` cards depends on the value of the variable. This gives the player a fundamental idea about variables in real programming.

Syntax−error free One of the significant barriers to learning programming is a syntax error. A programmer has to follow strict syntax rules for the programming language so that the program can be successfully compiled and translated into machine code. However, in Program Wars, a player does not directly write a program. As long as a player plays a valid card, the game will go on. Also, because it is a card game, the player cannot make typographical errors, which is a common problem when novice programmers enter text into an editor. Along with that the semantic error will not possible during the gameplay.

Programming language independence There are many different programming tools (i.e. Codeblocks, Visual Studio) and programming languages (i.e. Java, C++) for coding. For a beginner to become competent with such tools and all the syntax of a programming language might be difficult. Program Wars presents an language-independent platform where the player does not need to be concerned about the specifics of a programming language's syntax or peculiarities of a software development tool.

Competitive elements Program Wars has competitive elements that are important for engaging gameplay. Two players can play against each other, or a single player can play the game against an AI. Also, the game allows the player to score in various ways, which leads to multiple ways to win the game.

Attributes for engaging the player Program Wars has various cyberattack and cyberdefense-related cards, which make the gameplay more exciting and challenging. By introducing these cards, the game ensures players engage properly towards the gaming components.

Basic cybersecurity concepts Program Wars has some basic cards to introduce cyberattacks, like `Malware` attack cards that can be played to "infect". Again, cybersecurity is increasingly becoming an important part of the development of software systems as more and more of these are being built for the online environment where there is more risk of attack. Most games and SE curriculum either do not include this aspect, or it is taught late in the program. Program Wars reinforces the importance of considering system security from the very beginning of software creation.

3.2.2 Weaknesses

Program Wars has some areas where the game can be improved, some of which were identified by the previous user study [37]. The identified weaknesses of the game are described below.

Complex gameplay Although the game was intended to be easy to understand, for a person with a non-programming background, initially, the gameplay might be challenging to understand.

Limited use of functional concepts Although Program Wars has the `Group` card to create the concept of `method/function calling`, the idea of a general method is not reflected well through the `Group` card. In a real-world programming scenario, the method concept is not only binding a set of instructions together but rather is also used to hide specific details and redundancy in the program. As a result, the full understanding of the use of function calls is not reflected in the gameplay.

Elimination of Semantic error As the gameplay is designed to be played with error free environment which also eliminate the probability for semantic error.

3.3 Summary

Program Wars *v.*1.0 introduces several basic programming concepts via a card game. The game introduced a player to the programming concepts of instructions, loop, functions, and variables. It also introduces some concepts of cybersecurity like hacking, malware, antivirus and firewall. The use study of version 1.0 showed that maximum participant has intermediate knowledge regarding computer programming. Also the result analysis provide a indication that the gameplay had improved the knowledge of the participants regarding computer programming. Participants found `Group` card concept more complex. Again there was no evaluation regarding participants cybersecurity knowledge. However, there are several areas where the game can be improved, including the refinement of method concepts. The gameplay can be more interesting and challenging by adding various cyberattacks and cyberdefense-related cards.

Chapter 4

Program Wars 2.0

As previously discussed, one of the objectives of this research is to improve the educational aspects of Program Wars. This was done by making modifications to the User Interface (UI) and the gameplay. The introduction to the concept of algorithms was also a part of the modification.

There are two different play modes that have been introduced into the gameplay of Program Wars *v.*2.0 [39]. The different modes provide various combinations of cyberattack and cyberdefense cards. Also, the playing area for a player was divided into two parts which present the concepts of main program and method. The `Method` card is a new edition to this version, replacing the previous `Group` card. More specific cyberattack and cyberdefense cards are also introduced in this version of Program Wars. The details about the cards are described in Section 4.3. The concept of a library function containing an algorithm is also introduced via Sorting and Searching cards. Adding specific Cyberattack and Cyberdefense related cards along with the Algorithm cards makes Program Wars *v.*2.0 gameplay more exciting and challenging to the players. By introducing these cards, players get more engaged towards the gameplay. Also, the game score sheet is updated to match the modifications. This chapter provides details for all of these modifications.

4.1 Improvements to the User Interface (UI)

The User Interface (UI) for Program Wars was updated in the following manner.

- **Introduction of Beginner and Advanced Modes**

  Program Wars *v*.2.0 expands the original game with two modes of gameplay: *Beginner* and *Standard*. For each gameplay mode, the player can choose to play with one of four different sets of cyberattack and cyberdefense cards, adding them to the game's basic deck.[16] Figure 4.1 shows the initial home screen menu for Program Wars *v*.2.0 . In the figure, both the 'Game Type' and 'Level' use the drop-down menu. The available options (for both 'Game Type' and 'Level') are displayed in the black area indicated by an arrow.

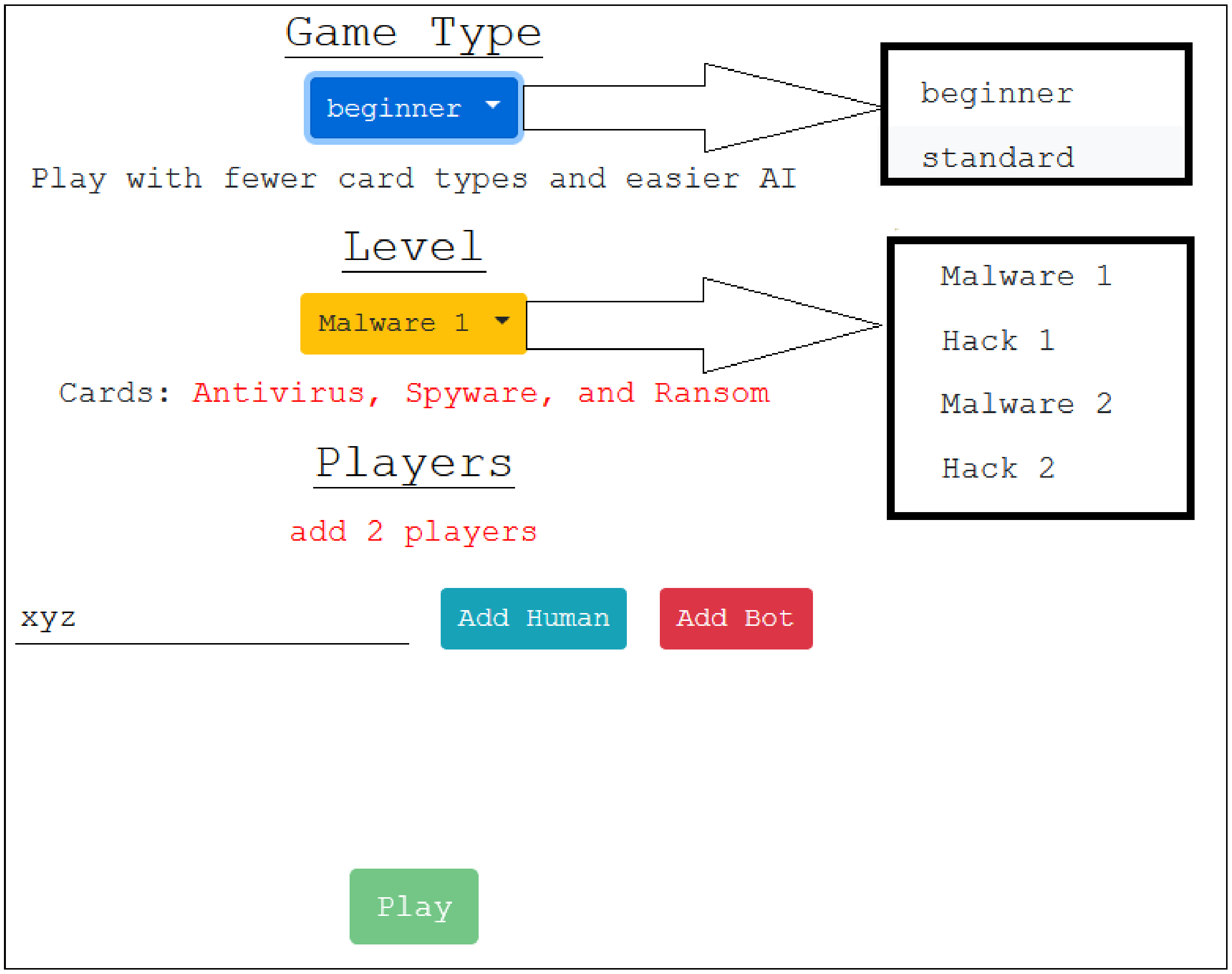


Figure 4.1: Initial home-screen for Program Wars *v*.2.0

In each mode, there are different combination sets for cyberattacks. A player can

[16]A basic Program Wars deck is comprised of only `Instruction` , `Method` , `Repeat` , `Variable` , `Search` , `Sort` , and `Computer Scan` cards.

Table 4.1: Game modes and Card sets in Program Wars *v*.2.0 .

Type	Card	Beginner				Standard			
		Malware 1	Hack 1	Malware 2	Hack 2	Malware	Hack	Combined 1	Combined 2
Safety	AntiVirus	X		X		X		X	X
	Firewall		X		X		X	X	X
Malware	Spyware	X				X		X	
	Ransomware	X				X			X
	Virus			X		X		X	
	Trojan			X		X			X
Hack	Buffer Overflow		X				X	X	X
	Cross-site Scripting				X		X		
	DoS Attack		X				X		X
	SQL Injection				X		X	X	

choose any combination based on the mode of the game. A detailed description of the combinations of cyberattack cards that are present in the various modes is given in Table 4.1.

- **Indicating playable cards**

  The user interface provides a view of the player's hand. Among those cards, all are not eligible for playing at all times. This feature provides feedback for the players as to what cards can currently be played by toggling or glowing. For example, as mentioned in Section 3.2.2, Program Wars has a restriction of using the `Variable` card. Unless the `Repeat-X` card has been played, the variable can not be applied. For this reason, if `Variable` cards appear in the player's hand, it remains disabled until a `Repeat-X` card is played.

- **Separating functions from the 'main program'**

  In Program Wars *v*.1.0 there is only one playing area for cards. One of the gameplay modifications is to facilitate the introduction of a new card for representing when a method/function is called in a program. To support the action, the play area was separated into two areas (Figure 4.2). Now, the main part of the program is played in the main stack, and the `Instruction` cards for a `Method` card is played in a separate area.

4.2 Gameplay Overview

At the start of the game, a player can choose to play against another human on the same computer (i.e. hot-seat play) or against provided computer opponents for up to a total of 4 players. Program Wars *v.*2.0 expands the original game with two modes of gameplay: *Beginner* and *Standard*. The two gameplay modes were created in response to feedback from the user study [37], where participants commented that "once I got the hang of the basics, there wasn't much room to improve".

In *Beginner* mode, only one of the two types of cyberattack cards are added to the deck - either two malware cards or two hack cards. In *Standard* mode, the player can choose to play with either all of the malware cards, all of the hack cards, or two different combinations of two malware and two hack cards. Each card set includes the corresponding cybersecurity cards that block the attacks. These card sets change the gameplay difficulty through the different combinations of cybersecurity and cyberattack cards. The game uses these card sets to progressively introduce cards representing more complicated concepts.

The game starts with each player being dealt five cards, with players receiving another card from the deck at the start of each turn. Program Wars is played in a series of rounds where each player takes a turn to either play or discard a card from their hand or draw a new hand. The goal is to create a program that reaches a specific number of points. The points represent the total number of instructions that would be executed by the computer based on the playing cards.

If either player has reached or exceeded the goal number of points, the game will finish at the end of the current round. This makes it possible for both players to reach the goal number of points in a game. In *Beginner* mode, the player's instruction score is used to determine the winner, and players can tie. In *Standard* mode, bonus points are awarded for reaching specific objectives, such as the use of `Repeat` and `Variable` cards or ending the game without being under the influence of cyberattacks. If both players have the same total score, these bonus points are used to break ties (e.g. the player who used the most `Variable` cards wins). If this method cannot determine a winner, the game is declared a tie.

4.2.1 Gameplay Areas

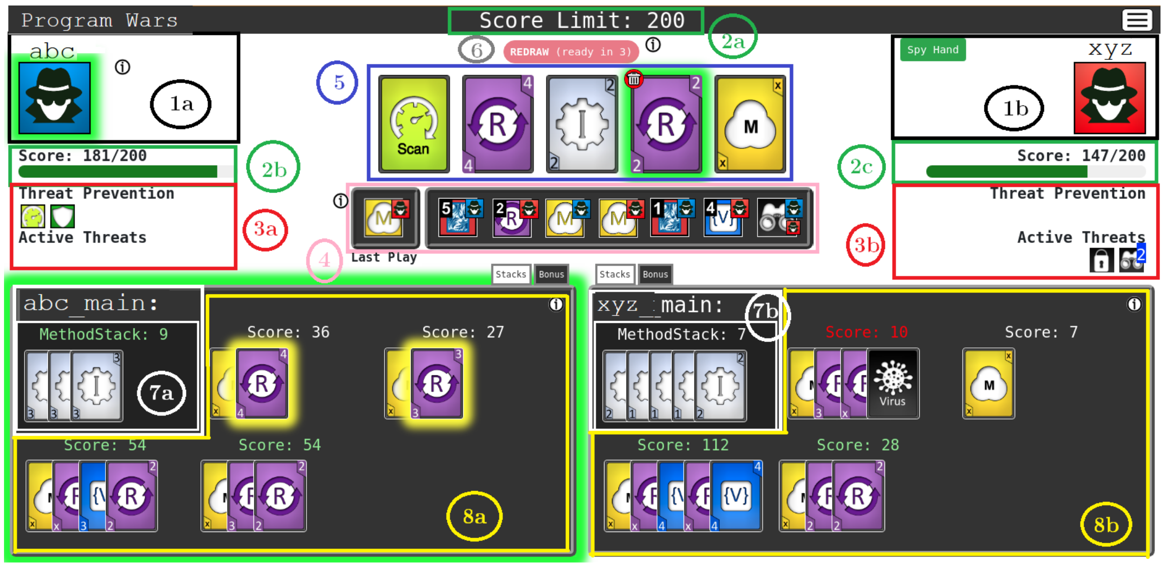


Figure 4.2: An annotated view of a Program Wars game in progress.

Figure 4.2 shows part of the way through a two-player game. The top right and left corners of the screen show the status of each player. The player's name is shown along with a unique image that is highlighted with green on their turn (*Areas 1a* and *1b*). It is currently `abc`'s turn. Under each player's image are that player's score and a progress bar showing their progress towards the goal number of points (*Areas 2b* and *2c*).[17] Below the player's score, the player's current status effects are displayed (*Areas 3a* and *3b*) represented by small icons, usually a smaller version of the image from the card that caused the effect.

[17]The bar is red if the player is below 50% of the goal number of points, yellow if below 75%, and green above 75%.

The *Threat Prevention* section shows the current cyberdefense effects active for the player. *Area 3a* shows that `abc` has a `Computer Scan` and `Antivirus` effect active. *Area 3b* shows where `xyz` has been attacked by `Ransomware` and `Spyware`. The effects that last for a specific number of turns are shown with the number of turns remaining over the top right corner. In *Area 3b*, the `Spyware`'s effect on `xyz` has only two turns until it expires. The status effects are added and removed as cyberattack and cybersecurity cards are played.

Each of the gameplay play areas contains an *Information* icon (an *i* within a circle) that, when clicked on, provides a short explanation of that play area.

4.2.2 Player's Hand

Area 5 of Figure 4.2 shows the current player's hand for the player whose image card is highlighted in green. When a card is selected, a small trash can icon appears in the upper lefthand corner of the card to allow the player to discard it. Above the cards is a button to enable the player to redraw their hand. If a player redraws their hand, they must wait for three (3) turns to do so again. *Area 6* shows the *Redraw* button is inactive and that there are three turns until `abc` can use it again.

Most cards are played by dragging the card from the hand and dropping the card where it is to be added - either the *Main* or *Method Stack* areas of the Program Editor (see Section 4.2.4). The Algorithm, cybersecurity, and cyberattack cards (excluding `Virus`) are played by clicking on them. When these cards are selected, a small overlay appears over the card allowing the player to make a choice. For algorithm cards, the overlay has a single button to activate the card. Safety cards have the same `Activate` button when the card can be activated. However, if the same effect is already applied to the player, then the overlay will indicate that the effect is already active, and the card will not be playable. For cyberattack cards, the overlay will give a set of buttons for valid targets of the attack. The overlay will display "No Targets" if there are no valid targets for the attack. Some cyberattack effects will not allow certain cards to be played. Cards that cannot be played will be highlighted red while in the player's hand and will not be draggable or show an overlay when selected.

4.2.3 Game History

Area 4 of Figure 4.2, shows a visual representation where the last eight turns history is reflected. Each icon represents a card that was played. In the top right corner of each of these icons is the image of the player that played that particular card. The leftmost icon appears in a separate box to reinforce that it was the most recent card played; it can also be seen that `xyz` played a `Method` card last turn. If a card had a target player, the target's image is placed on the card icon's bottom right corner. This can be seen on the rightmost icon in the turn history, where it shows that `abc` played a `Spyware` card on `xyz`. Some cards may include another card or effect, such as `Computer Scan` when it removes an impact. In this case, a small icon for the card or effect will be placed on the bottom left corner of the card image.

4.2.4 Program Editor

The bottom half of the screen contains representations of a source code editor where the player builds their program (Figure 4.2 *Areas 7* and *8*). Programs are built by creating stacks of `Instruction`, `Method`, `Repeat` and `Variable` cards. Each stack of cards represents a portion of the player's program. Stacks are created by dragging an `Instruction` or `Method` card onto the Program Editor. `Repeat` and `Variable` cards are played by dropping them onto an existing stack. A stack with a highlight around the top card indicates that the currently selected card can be played on that stack. If the valid stack is in the current player's Program Editor, the highlight will be yellow. For cyberattack cards, such as `Virus`, the proper stack will be in an opponent's Program Editor and the highlight will be red. In *Area 8a*, the currently selected `Repeat` card from `abc`'s hand can be played on one of the two highlighted stacks.

Each stack shows its score above it, telling the player how many points that stack is contributing to the player's score. The *Method Stack* (*Areas 7a* and *7b*) is surrounded by a white border and is a special stack that only accepts `Instruction` cards. The *Method Stack* score is used as the value of the `Method` card. The *Method Stack* score is capped at nine (9) points, and the stack can hold a maximum of six (6) `Instruction` cards. Placing too many low values `Instruction` cards in a *Method Stack* can leave a player unable to reach the stack's maximum score. As the `Method` cards total value is multiplied with each added `Instruction` card's value, a player's score can get a sudden boost. By setting up a limit on `Instruction` cards, the game has been balanced. A player's total score is the sum of all of their stack scores, excluding the *Method Stack*. Stacks with green scores indicate that the stack is complete[18] and no more cards can be added to it. A stack with a red score indicates that the stack does not contribute its full value to the total score. For example, in *Area 8b* of Figure 4.2, the stack in `xyz`'s play area should contribute 21 points (7 for the `Method` card times 3 for the `Repeat`-3). However, with the use of the `Virus` card, the stack's value reduces by half (10).

4.2.5 Gameplay Goals

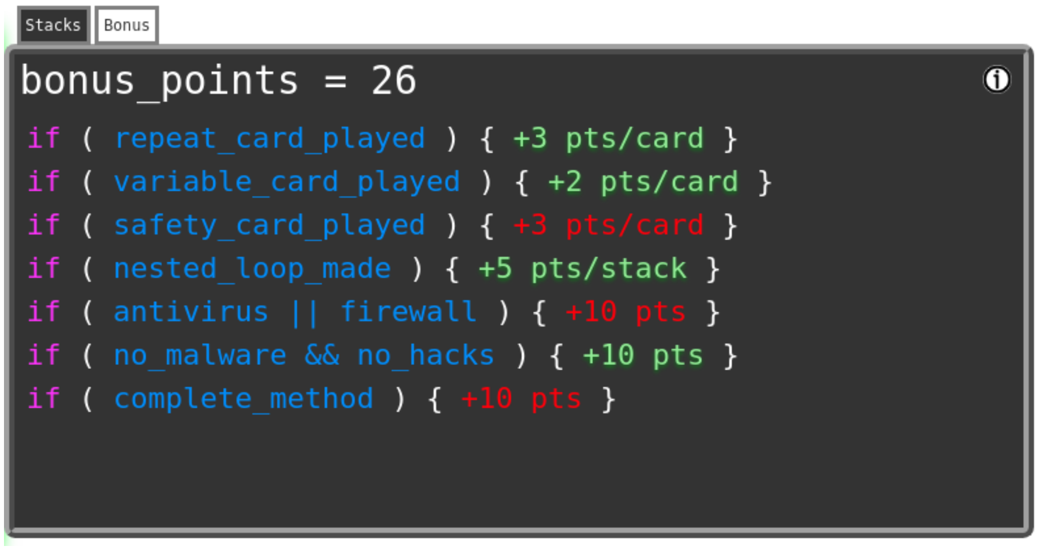
In *Standard* mode, a player can achieve bonus points by reaching certain sub-goals during a game. Their current bonus points can be seen by selecting the *Bonus* tab. The tabs are in the center of the screen attached to the Program Editor of each player (*Area 8a* of Figure 4.2). This tab is not shown for a computer opponent to prevent human players from seeing a computer opponent's bonus progress. The *Bonus* area, shown in Figure 4.3, has a set of conditional statements written in a C-like pseudocode.

The conditional portion of the statement identifies the bonus that is awarded. Some bonuses are for playing cards (i.e. `True` when playing a `Repeat` card), and others are for maintaining a certain status (i.e. not being affected by cyberattacks). The body of the condition shows the number of points a player will receive to satisfy the requirement. If the text is red, it means the condition is not met, and the text will turn green when the

[18]A stack is considered complete when it has two (2) `Repeat` cards played on it. `Repeat-X` cards only count towards completing a stack if they are paired with a `Variable` card.

player satisfies the condition. Some conditions, such as the `(no_malware && no_hacks)` condition, may be gained or lost during the game.

At the end of each turn, the player's bonuses are re-calculated, and the total bonus score is updated and shown at the top of the *Bonus* tab. Bonus scores are only added to the player's instruction score once the game has finished and do not apply to reaching the goal score during the game. These bonuses are intended to help reinforce certain concepts and motivate the player towards what can be considered good programming practices.



```
bonus_points = 26
if ( repeat_card_played ) { +3 pts/card }
if ( variable_card_played ) { +2 pts/card }
if ( safety_card_played ) { +3 pts/card }
if ( nested_loop_made ) { +5 pts/stack }
if ( antivirus || firewall ) { +10 pts }
if ( no_malware && no_hacks ) { +10 pts }
if ( complete_method ) { +10 pts }
```

Figure 4.3: The *Bonus* tab for a player.

4.3 Card Additions and Improvements

To improve the representation of programming and cybersecurity concepts in Program Wars, several new cards have been introduced in Program Wars *v.*2.0. Also, some of the effects of cards have been modified to enhance gameplay. Table 4.2 provides a comparison among the cards between Program Wars *v.*2.0 and Program Wars *v.*1.0 .

The player builds their program using the basic building blocks of instructions, methods and repetition. Also, in their turn, a player can launch a cyberattack at an opponent or prepare their defence. This section describes each of the new cards that are introduced for computer programming, cyberattack and cybersecurity-related concepts.

Table 4.2: Cards comparison in Program Wars *v.*2.0 vs Program Wars *v.*1.0

Type	Card Name	*v.*2.0	*v.*1.0
Basic Programming Card	Instruction	✓	✓
	Repetition	✓	✓
	Variables	✓	✓
	Group	×	✓
	Method	✓	×
Cyberdefense/Safety Card	Computer Scan	✓	×
	AntiVirus	✓	✓
	Firewall	✓	✓
	Generator	×	✓
Attack Card	Spyware	✓	×
	Ransomware	✓	×
	Virus	✓	×
	Trojan	✓	×
	Malware (Basic)	×	✓
	Power Outage	×	✓
Hack	Buffer Overflow	✓	×
	Cross-site Scripting	✓	×
	DDoS Attack	✓	×
	SQL Injection	✓	×
	Hacking (Basic)	×	✓
Backup Cards	Battery Backup	×	✓
	Overclocking	×	✓
Algorithms / Library Functions	Sorting	✓	×
	Searching	✓	×

4.3.1 Computer Programming

The cards in Program Wars *v.*1.0 primarily focused on computer programming, and many of these cards carry over into Program Wars *v.*2.0. The `Instruction`, `Repeat` and `Variable` cards remain unchanged in their effect and working procedure in Program Wars *v.*2.0 gameplay. In Figure 4.2, the example where an `Instruction` card, `Repeat` card and `Variable` card are used is shown. For computer programming, the `Method` card replaced the `Group` card in the new version of the game.

`Method:` In Program Wars *v.*1.0 , the `Group` card represented the concept of a procedure, function or method in a programming language. However, from the user study of Program Wars *v.*1.0, participants had some confusion related to the concept of a method and game's `Group` card. Program Wars *v.*2.0 replaces the `Group` card with the `Method` card in an effort to address this issue.

The `Method` card acts as a proxy for the contents of the *Method Stack* area, with the player's total score being adjusted accordingly. If a new 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` and `Variable` cards to increase the effect of a `Method` card.

In Program Wars *v.*2.0 , the player can add a `Method` card to the play-field, and the card reflects the total scores of the method stack. Figure 4.4 shows the new `Method` card.

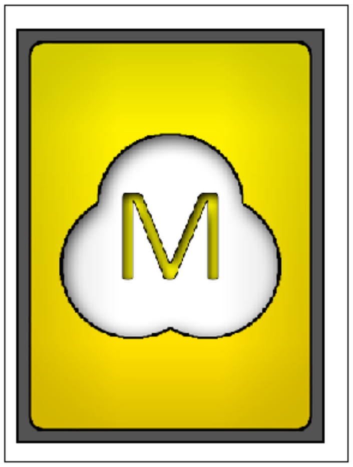


Figure 4.4: Program Wars *v.*2.0 `Method` card.

Relation to computer programming: Computer programs are commonly broken up into functions, methods or procedures. All of these names mean the same thing – a group of instructions. This card also introduces the software engineering principle of refactoring. Refactoring is restructuring code without changing what it does. Refactoring code into small chunks that can be reused makes software easier to change or maintain.

4.3.2 Cybersecurity

In Program Wars *v.*1.0 , there are two cybersecurity concepts. As cybersecurity is an important aspect of modern-day software development, Program Wars *v.*2.0 includes more expanded cybersecurity concepts. Also, these cards provide for an interactive experience between the players. This section presents the cards that teach about two types of cyberattacks, malware and system attacks, and their corresponding cyberdefenses.

4.3.2.1 Malware

Program Wars *v.*1.0 represented the cyberthreat of malware with a single card – the `Malware` card. In Program Wars *v.*2.0 , the `Malware` card was replaced with cards that more directly represent four of the most common types of malware: spyware, ransomware, virus, and trojan. Figure 4.5 shows these new cards.

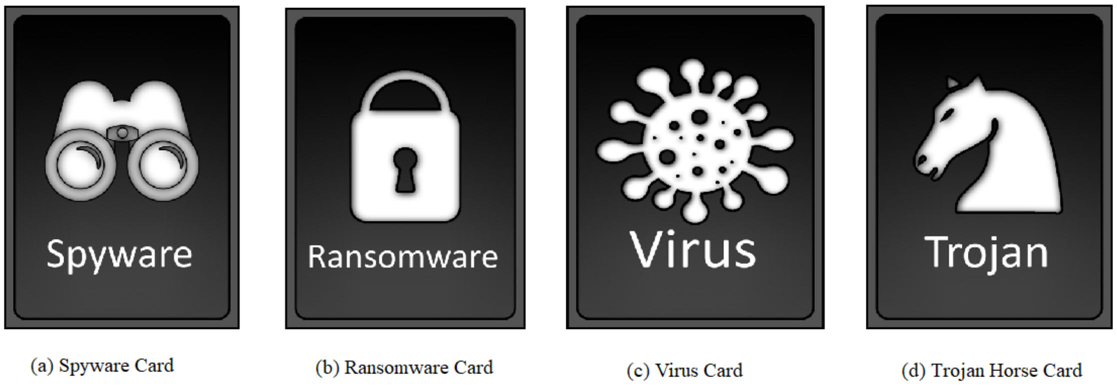


Figure 4.5: Program Wars *v.*2.0 Malware cards

`Spyware`: Spyware is used to gather and send information to another party without the target's consent. The `Spyware` card represents this same situation in the context of the game. When a player plays a `Spyware` card against their opponent, a 'spy' button will appear beside the opponent's name for the next three turns. It will allow the attacker to view an opponent's hand. Affected players can remove the effect by playing `Computer Scan` or `Antivirus` card (see in Section 4.3.2.3).

`Ransomware :` This card's effect reflects the real-world concept of ransomware, where an attacker blocks access to a target's files, such as encrypting them, and threatens to publish or delete them unless a ransom is paid. When a player plays a `Ransomware` card on an opponent, the targeted player loses 10 points from their total score and the points added to the attacker scores. This can result in an opponent's score becoming negative. Unlike real-world ransomware, recovering from this attack is simple, as an affected player can recover their points by either using a `Computer Scan` or `Antivirus` card.

`Virus :` A computer virus is a computer program that replicates itself by modifying other programs. The `Virus` card is used to reduce the effect of a stack of cards in the *Main* area by reducing the points of a card stack by 50%[19]. This card is the most similar to the `Malware` card from Program Wars *v.*1.0 , where the `Malware` card reduced a player's total score by 25%. The attacking player can play viruses on an opponent's stacks. If the `Virus` card is played on a stack that starts with an `Instruction` card, it reduces a stack score to 0 and prevents any more cards from being played until an `Antivirus` card is played. If the stack starts with a `Method` card, the reduction is by 50% instead.

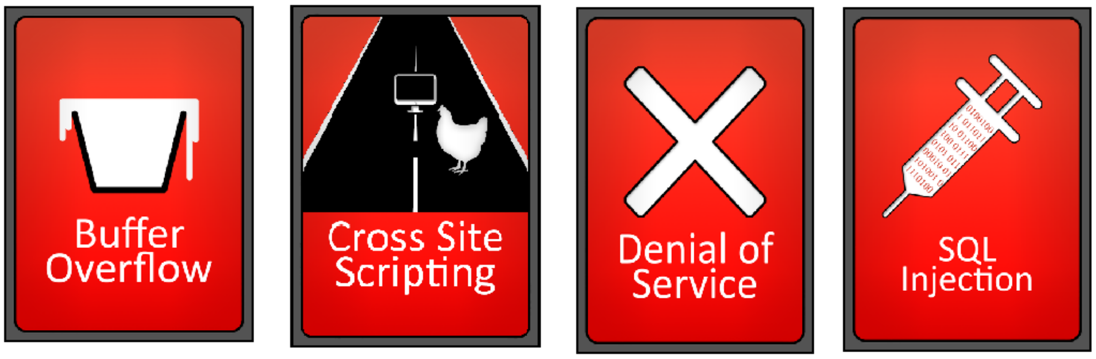
`Trojan Horse:` In the real world, a Trojan Horse is a computer program that misleads users as to its real intent. When a `Trojan Horse` card is played against an opponent, a random card in the opponent's hand is replaced with one that mimics it. The actual effect of the mimic card depends on what card is replaced. If the replaced card is either the `Method`, or the `Instruction` card, then when the opponent plays the mimicked card, it has the same effect as if a `Ransomware` card was played on the opponent. If the replaced card is a cyberattack card, the opponent will receive a `Spyware` effect when the card is played. Cards that are only added to stacks in the *Main* area (i.e. `Repeat` and `Variable`) become a `Virus` card. Human players can tell which cards are being mimicked when playing against a computer opponent, as they will have a horse head logo on top of the card

[19]For reduction the floor is taken if the points of the card stack is odd.

when active.

4.3.2.2 Hacking

Program Wars *v*.1.0 contained a single `Hack` card that represented an intrusion into a computer system. The effect of the `Hack` card was to remove one of the stacks of cards on an opponent's playfield. Program Wars *v*.2.0 refines this idea by adding specific cards to represent common ways whereby computer systems are intruded or affected by an intrusion. These four cards provide representations of the effects of four types of system attacks: causing a buffer overflow, cross-site scripting, a denial of service attack (DoS), and injection of malicious SQL code. Figure 4.6 shows these cards.



(a) Buffer Overflow Card (b) Cross Site Scripting Card (c) Denial of Service Card (d) SQL Injection Card

Figure 4.6: Program Wars *v*.2.0 Hacking cards.

`Buffer Overflow:` A common system attack is to send data to a program such that a memory buffer overflows and causes program instructions to be overwritten by malicious code, which in turn are executed. In Program Wars *v*.2.0 , the `Buffer Overflow` card prevents an opponent from playing any `Instruction`, `Repeat`, `Variable` or `Method` cards for two turns. The concept behind this card's effect is that if a program tries to utilize more space than is available, then the call stack overflows, and the system protects itself by allowing no more code to be run. This is similar to real-world solutions where a system checks that the stack has not been altered when a function returns and exits the program with a segmentation fault [40, 41].

`Cross-site Scripting (CSS):` Cross-site scripting is a code injection attack. The attack happens when the victim visits a web page or web application that administers the harmful code [42]. The visited web page or service acts as a carrier to deliver the malicious code to the affected browser. In Program Wars *v.*2.0 , the `Cross-site Scripting` card stops a player from playing any algorithm or cyberattack cards. The concept behind this card is to make a player familiar with this type of attack by preventing the advantages given by certain cards, including attacking an opponent. When affected by a CSS card, a player can only play a firewall and scan card. If a player cannot play any card from their hand, there is a “pass” button beside the *Redraw* button that can pass the turn to the next player.

`Denial of Service (DoS):` A Denial of Service (DoS) attack occurs when a computer system connected to a network is intentionally flooded with requests so that the system can no longer handle legitimate requests. In Program Wars *v.*2.0 , the `Denial of Service` card prevents a player from redrawing new cards at the end of their turn and changes the *Redraw* button to a *Pass* button. This effect lasts for three turns resulting in the player having fewer cards to choose from their hand in subsequent turns. In the worst-case scenario, the player has no playable cards for three turns and has to pass.

`SQL Injection :` In an SQL injection attack, malicious SQL code is entered into a data field such that the code is run on a back-end database. The result of such an attack is to obtain information that was not intended to be disclosed or delete and/or corrupt the data in the database. In Program Wars *v.*2.0 , the `SQL Injection` card can be used to slow down the progress of an opponent by reducing the total of the *Method Stack* area by two points. This deduction from the method stack affects all of the `Method` cards in the playfield. Cyberdefense-related cards remove the effect of a SQL injection. The concept behind this card is that of infiltration of a program’s method by malicious code.

4.3.2.3 Cyberdefense

Program Wars *v.*1.0 provided three cards for cyberdefense. Two of the cards were "permanent" cards, meaning that they remained on a player's playfield when played. These two cyberdefense cards were referred to as *Safeties*. The first of these cards was the `Antivirus` card that prevented the `Malware` card from being played on a player. The second of these cards was the `Firewall` card which protected against the `Hack` card. The third card was the `Overclock` card, which combated the `Malware` card by increasing the player's total score by 25%. However, it was observed that the gameplay effect of the `Overclock` card did not match well with real-world cybersecurity concepts. Program Wars *v.*2.0 continues the use of the two safety cards and adds a new one-time-use cyberdefense card: `Computer Scan` . Figure 4.7 shows Program Wars *v.*2.0's cyberdefense cards.

(a) Computer Scan Card

(b) Antivirus Card

(c) Firewall Card

Figure 4.7: Program Wars *v.*2.0 Cyberdefense cards.

`Computer Scan`: The `Computer Scan` card represents the action of a user explicitly scanning all of their files to find any infected items using an antivirus tool. If the player is under the influence of multiple malware and/or hack cards, then the `Computer Scan` card allows the players to choose which card effect to remove. If the player is not under the influence of a cyberattack card, the effect is saved until the player is attacked, at which time the cyberattack is neutralized.

`Antivirus:` An antivirus program is a program or set of programs designed to prevent, search for, detect, and remove malware from a computer system. The `Antivirus` card reflects this real-world tool by protecting a player from the effect of any of the malware attack cards. If the player is already under the effect of one or more malware cards, all of the effects are removed when this card is played. Unlike the `Computer Scan` card, the effect of this card is permanent once it is played, thereby protecting the player from any future malware card attacks.

`Firewall:` A firewall is a network security device that controls incoming and outgoing network traffic and grants or prevents data packets based on a set of security rules, thereby protecting a computer system from various intrusion attacks. Like the `Antivirus` card, the `Firewall` card reflects this real-world tool by preventing hack cards from being played on the player. Similar to the `Antivirus` card, if the player is affected by any hack cards, these effects are removed, and the effect of this card is permanent once played.

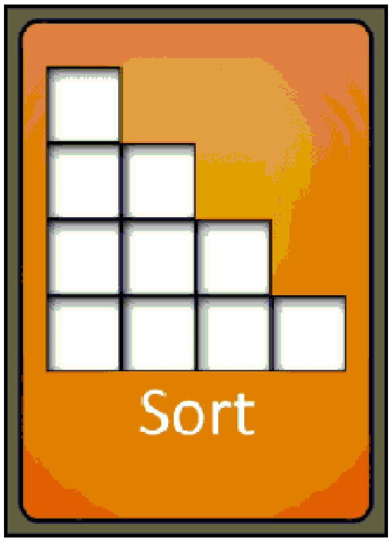
4.3.2.4 Algorithms / Library Functions:

The use of algorithms, often from libraries, is an essential part of computer programming. Two key categories of algorithms are searching and sorting, and both of these are introduced in Program Wars *v.*2.0 . Figure 4.8 shows the two new sorting and searching cards.

`Sort:` Sorting is the arrangement of items into an ordered sequence. In Program Wars *v.*2.0 , the `Sort` card allows a player to rearrange the top five (5) cards of the deck into whatever order they choose. This can allow a player to control what cards they will draw for their next turns. [20] When the card is played, an overlay is opened showing the top five cards, and the player can drag a card onto another card to cause them to swap places.

Relation to computer programming: Sorting is important for optimizing the effi-

[20]A minimum of 2 turns in a 4 player game and a maximum of 3 turns in a 2 player game



(a) Sorting Card

(b) Search Card

Figure 4.8: Program Wars *v.*2.0 Algorithms cards.

ciency of other algorithms (such as the merge algorithm [43]) that require input data to be in sorted lists.

`Search`: Searching is the process of locating a particular element present in a given set of elements. In Program Wars *v.*2.0 , the `Search` card allows a player to search for a specific card within the top ten (10) cards of the deck. Playing the card results in an overlay being opened that shows these cards, and the player selects one to immediately put into their hand for their next turn.

Relation to computer programming: A search algorithm prevents a user from having to look through lots of data to find the specific information.

4.4 Summary

Program Wars *v.*2.0 is a modification of Program Wars *v.*1.0 in such a way that it better serves the user for learning different programming and cybersecurity concepts. With respect to the programming aspect, a `Method` card was introduced to represent the method concept. For cybersecurity concepts, several new cards have been introduced to make players familiar with real-world cybersecurity scenarios. Also, the new version introduced the algorithm/library function concept with `Algorithm` cards. There are several concepts that have been changed in the new version of the game. Table 4.3 provides a summary of the changes made between Program Wars *v.*1.0 and Program Wars *v.*2.0 . Also, the table presents how various game elements align with The *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering* [44], specifically in the knowledge areas of *Computing Essentials* and *Security*.

Table 4.3: Features of the two versions of Program Wars with alignment of game elements to curriculum guidelines.

Concepts	*v.*2.0	*v.*1.0	Curriculum Guidelines
Programming language basics	`Instruction`, `Repeat` cards	`Instruction`, `Repeat` cards	CMP.cf.8
Control Flow	Bonus goals	Division of play-field into two section with one section randomly chosen as playable.	CMP.cf.1
Method concept	`Method` card	`Group` card	CMP.cf.4
Cyberdefense/Safety	Added `Computer Scan` card	`Antivirus` & `Firewall` cards	SEC.net.3
Cyberattack	`Virus`,`Ransomware`, `Spyware`,`Trojan Horse` cards	`Malware` card	SEC.net.1
Hacking	`Buffer Overflow`, `Cross-site Scripting`, `SQL Injection` & `Denial of Service` cards	`Hack` card	SEC.net.1
Algorithms / Library Functions	`Search` & `Sort` cards	Not present	CMP.cf.2
Gameplay Versatility	Two modes (Beginner & Standard)	Not present	N/A

Chapter 5

User Study of Program Wars 2.0

This chapter presents the detailed results of a user study of Program Wars *v.*2.0 . First, the experimental procedures for the user study are described. After that, the results of the study are presented. A detailed analysis of the results and findings has given later in this chapter.

5.1 Research Methodology

The contributions of Program Wars *v.*2.0 for learning programming and cybersecurity are measured based on the results of a user study. A similar user study of Program Wars *v.*1.0 contained questions that had programs written in one of four different programming languages (C, Pascal, Python, and FORTRAN). The Program Wars *v.*2.0 user study used a C-like pseudo-code. Also, the previous study did not evaluate the participants learning outcomes for cybersecurity concepts. The Program Wars *v.*2.0 user study participants are particularly chosen from non-CS major areas, so their learning outcomes can be measured more accurately, whereas the Program Wars *v.*1.0 participants had already had some exposure to programming concepts. In the Program Wars *v.*2.0 user study, the programming questions sought to measure the same knowledge, but were not the same as those used in the previous study. Otherwise, the two studies procedures for assessing the knowledge outcome remains similar. For measuring the cybersecurity concepts, several real-world scenarios were presented to the participants. Participant's knowledge change is calculated based on the responses they provided for the specific topics.

5.2 Detail of User Study

To assess if Program Wars *v*.2.0 improves a player's knowledge of programming and cybersecurity concepts, a user study was conducted. The user study was conducted following the policy[21] and guidelines provided by the University of Lethbridge Human Participant Research Committee (HPRC). The HPRC committee analyzed all the material of the user study, including all the questions and approved the study[22]. The user study was conducted using Qualtrics[23]. There were three (3) stages in the user study,

- **Pre-game (Stage # 1) –** In the pre-game questionnaire, the participant was asked for their consent regarding the study and some demographic questions. Along with that, some multiple-choice questions were asked related to computer programming and cybersecurity concepts to assess the participant's previous knowledge of programming and cybersecurity.

- **Play Program Wars *v*.2.0 (Stage # 2) –** After completing the pre-game questionnaire, the participant was asked to play Program Wars *v*.2.0 against the AI at least three (3) times.

- **Post-game (Stage # 3) –** In this stage, participants were asked to complete a post-game questionnaire. They were asked some knowledge questions similar to those from the pre-game questionnaire related to computer programming and cybersecurity to identify the learning outcomes. Also, participants were asked questions to assess if they could map the game mechanics (e.g. cards) to real-world scenarios, such as a computer program or a cybersecurity situation.

5.2.1 Participants

As Program Wars *v*.2.0 focuses on teaching cybersecurity and programming concepts, the target demographic for participants was those with very little to no programming or

[21] https://ethics.gc.ca/eng/home.html
[22] HPRC Protocol Number: 2020-113
[23] https://uleth.qualtrics.com

cybersecurity background (i.e., non-CS majors). To participate in the user study, a participant must be 18 years of age or older.[24] The primary approach to recruit participants was via electronic communication. An invitation (Appendix B) was sent via e-mail to the administrative assistants of the Department of Biological Sciences, Kinesiology & Physical Education, Geography Environment, Anthropology and Psychology[25] at the University of Lethbridge to be distributed among the respective department students, alumni, faculties, and staff. No contact between the participant and the researcher(s) was made after the invitation unless it was initiated by the participant.

5.2.2 Study Procedure Details

Initially, all of the participants were invited via e-mail. In that invitation, the user study URL was provided. Before starting the questionnaire, participants were presented an informed consent web form (Appendix A). In order to continue with the questionnaire, a participant needed to click on the 'Agree' button. If they (the participant) agreed to participate in the study, they were directed to another web page where separate links were provided for the three (3) stages of the study. In the Pre-game questionnaire (Stage # 1), there were two (2) Demographic Questions, two (2) Experience Questions, four (4) Knowledge Questions regarding computer programming and two (2) Knowledge Questions related to cybersecurity. To complete this stage required approximately 15 minutes.

After completing the pre-game questionnaire, the participant was redirected to Stage # 2, where they played Program Wars *v*.2.0 several times until the participant was comfortable and felt they understood the game's concepts well enough to answer the post-game questions. It was recommended that participants play the game a minimum of three (3) and a maximum of ten (10) times. The average time required to complete this stage was approximately 30 minutes.

[24]For PSYC1000, some of the participants may be 17-year-old because the study does not contain any explicit or graphic content that is unsuitable for a minor.

[25]For Psychology department the study was held in a separate platform 'SONA', which was also approved by the similar department.

After completing Stage # 2, the participants were asked to complete a post-game questionnaire (Stage # 3) comparable to the pre-game questionnaire. There were three (3) questions related to the game elements mapping with programming concepts, two (2) programming comprehension questions, two (2) cyberattack questions, and associated with these two cyberattack questions, two (2) cyber defence questions were asked. After that, six (6) feedback questions were asked. The participants were also given an opportunity to provide comments regarding the game. Finally, the study concluded by asking users about their rating of Program Wars *v*.2.0 . To complete this stage required approximately 15 minutes.

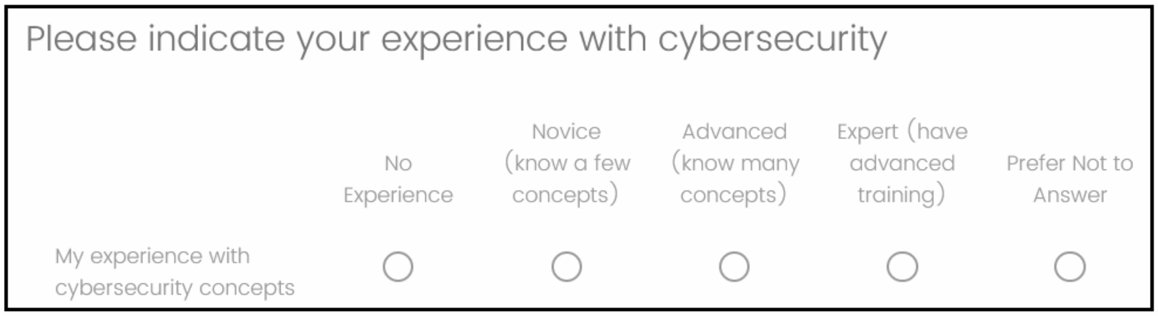
Participation in the study was expected to take 60 $\sim$ 90 minutes, depending on the number of times a participant played the game. Participants were identified by a unique study_id, which replaced the requested e-mail address. The unique study_id was used to link up the pre-game and post-game responses of the participant. As a result, there was no personal identifying information kept after the data from the questionnaire was downloaded. If a participant did not complete either of the pre-game or post-game questionnaires, it was decided that the participant did not complete the whole user study, and the data was incomplete. The incomplete responses were not taken into consideration in the final study results. After all the filtering, there were twenty-six (26) participants who completed all the stages of the study.

5.2.3 Questions Asked

Throughout the user study, there were several questions asked in both the pre-game and post-game questionnaires. The details about the questions are discussed below.

5.2.3.1 Demographic and Previous Experience Questions:

The participants were asked two (2) demographic questions related to their age range and level of education. Also, participants were asked two (2) experience questions related to their prior programming and cybersecurity knowledge. One of the experience questions is presented below, and the rest of the questions are presented in Appendix C.



Please indicate your experience with cybersecurity

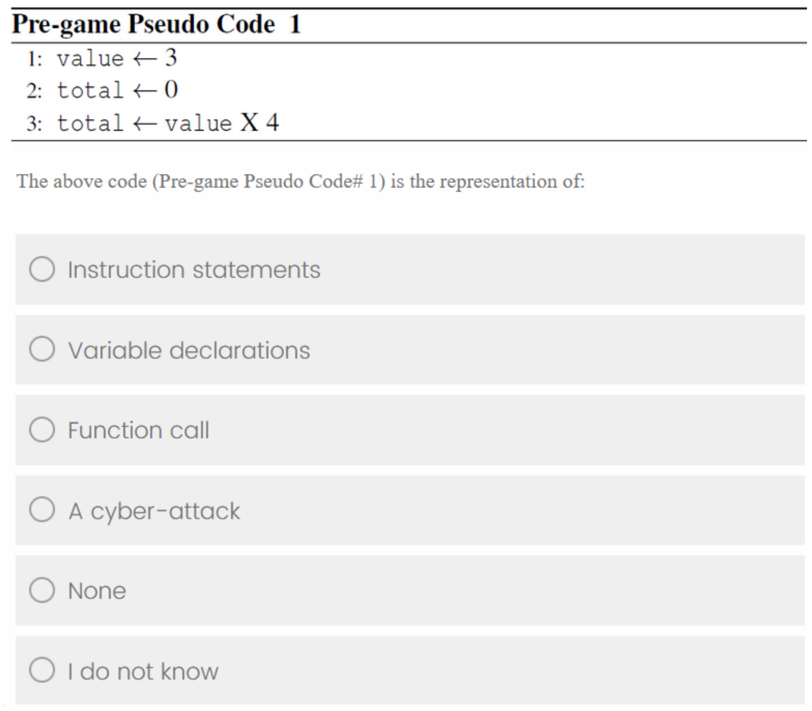
	No Experience	Novice (know a few concepts)	Advanced (know many concepts)	Expert (have advanced training)	Prefer Not to Answer
My experience with cybersecurity concepts	○	○	○	○	○

5.2.3.2 Programming Comprehension:

The participants were asked several computer programming questions both in the pre-game and post-game questionnaire. They were given some pseudo code and asked questions about the outcomes. Each question was designed to test one or two areas of computer programming knowledge. The questions testing specific knowledge areas are now presented.

Variable Concept: To understand a participant's knowledge about *variables* in computer programming, participants were asked questions in both pre-game and post-game questionnaires. They were given some pseudo code and asked questions that require them to understand the use of variables. Below, an example of the questions from the pre-game and post-game questionnaires is presented.

- **Pre-game Programming Question #3:**



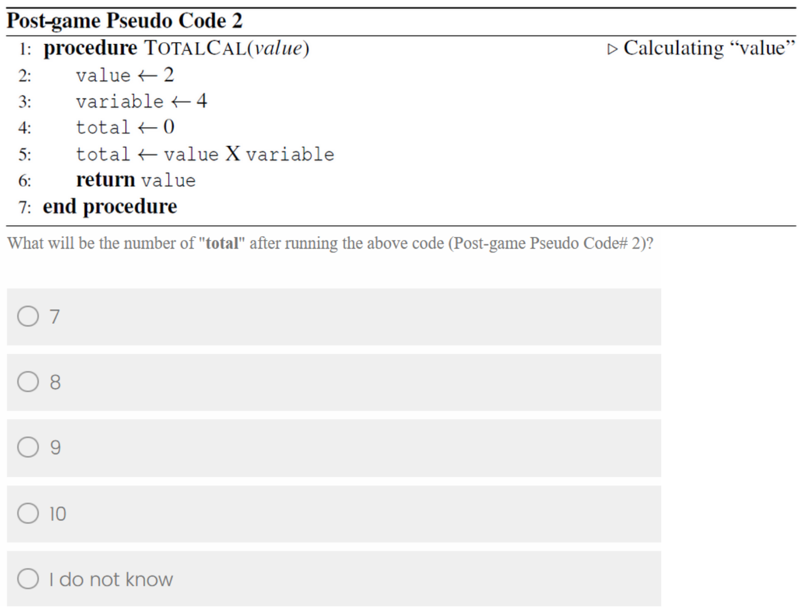
Pre-game Pseudo Code 1

```
value ← 3
total ← 0
total ← value X 4
```

The above code (Pre-game Pseudo Code# 1) is the representation of:

- ◯ Instruction statements
- ◯ Variable declarations
- ◯ Function call
- ◯ A cyber-attack
- ◯ None
- ◯ I do not know

- **Post-game Programming Question #5:**



Post-game Pseudo Code 2

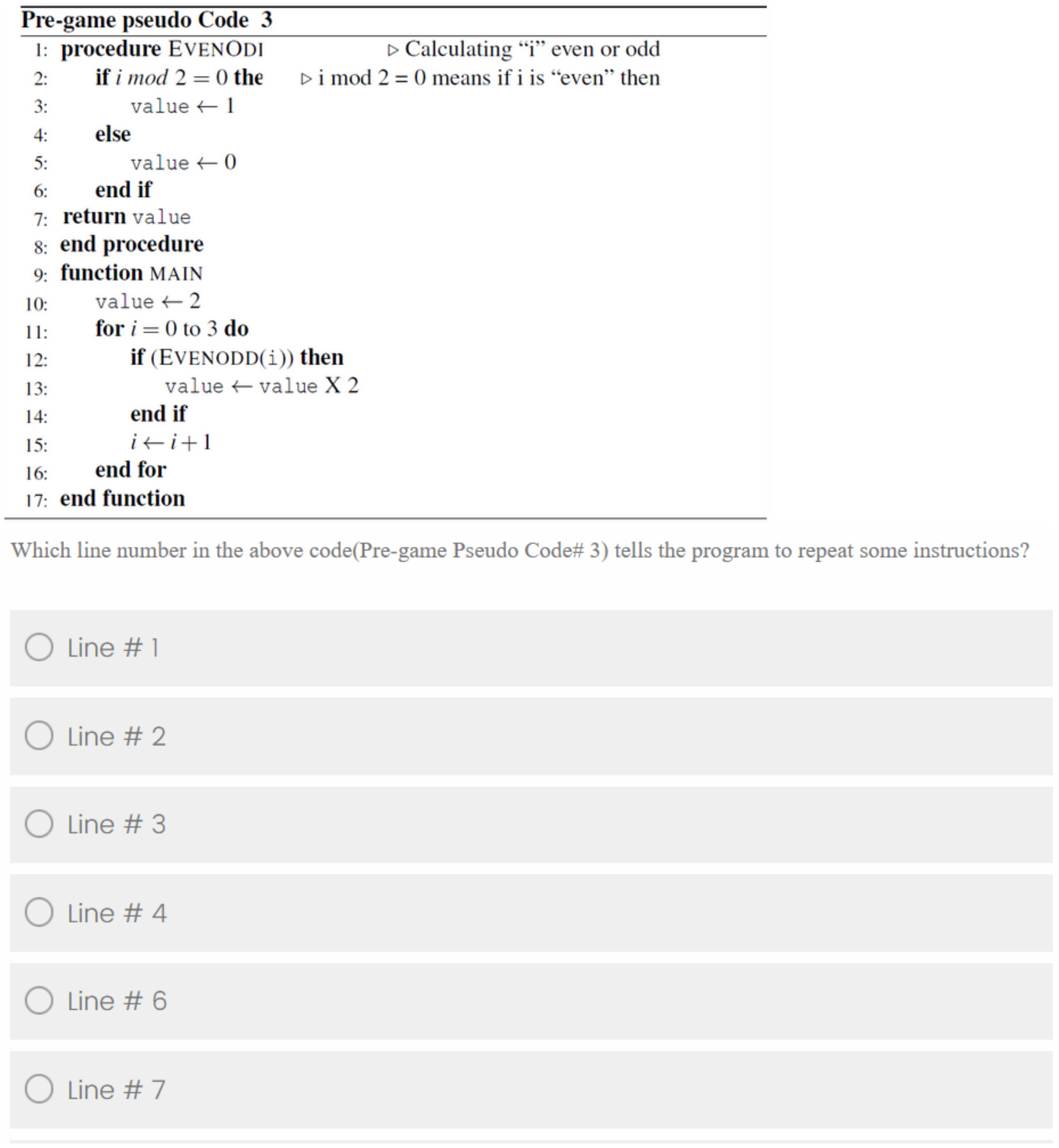
```
procedure TOTALCAL(value)                ▷ Calculating "value"
    value ← 2
    variable ← 4
    total ← 0
    total ← value X variable
    return value
end procedure
```

What will be the number of "**total**" after running the above code (Post-game Pseudo Code# 2)?

- ◯ 7
- ◯ 8
- ◯ 9
- ◯ 10
- ◯ I do not know

Loop Concept: To understand a participant's knowledge about *loops* in computer programming, participants were asked questions in both the pre-game and post-game questionnaire. They were given some pseudo code and asked questions that require them to understand the use of loops. Below, an example of the questions from the pre-game and post-game questionnaires is presented.

- **Pre-game Programming Question #2:**



Pre-game pseudo Code 3

```
1:  procedure EVENODI                          ▷ Calculating "i" even or odd
2:      if $i$ mod $2 = 0$ the        ▷ i mod 2 = 0 means if i is "even" then
3:          value ← 1
4:      else
5:          value ← 0
6:      end if
7:  return value
8:  end procedure
9:  function MAIN
10:     value ← 2
11:     for $i = 0$ to 3 do
12:         if (EVENODD(i)) then
13:             value ← value X 2
14:         end if
15:         $i \leftarrow i + 1$
16:     end for
17: end function
```

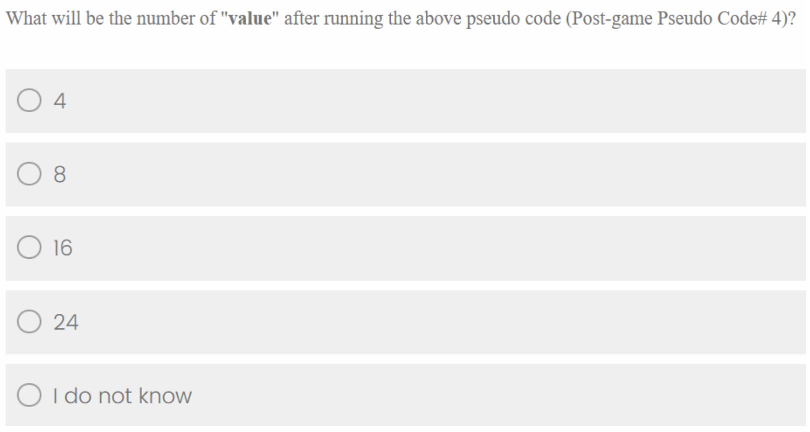
Which line number in the above code(Pre-game Pseudo Code# 3) tells the program to repeat some instructions?

- ○ Line # 1
- ○ Line # 2
- ○ Line # 3
- ○ Line # 4
- ○ Line # 6
- ○ Line # 7

- **Post-game Programming Question #4:**

Post-game pseudo Code 4

```
value ← 2
for $i = 0$ to 3 do
    value ← value X 2
    $i \leftarrow i + 1$
end for
```

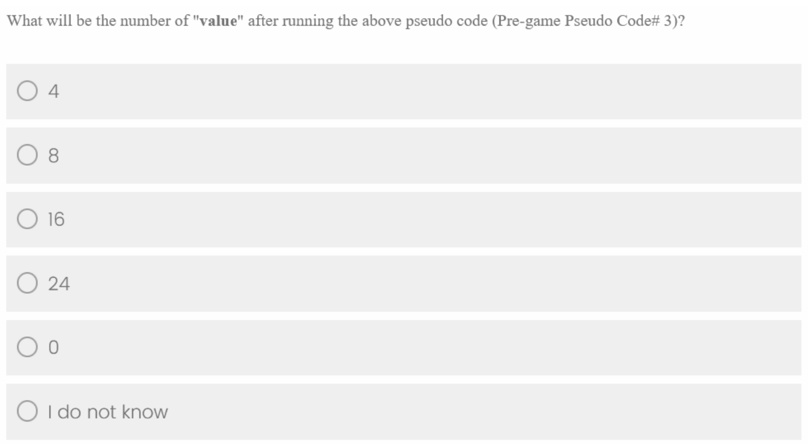


What will be the number of "**value**" after running the above pseudo code (Post-game Pseudo Code# 4)?

- ○ 4
- ○ 8
- ○ 16
- ○ 24
- ○ I do not know

Method Concept: To understand a participant's knowledge about *method* in computer programming, participants were asked questions in both pre-game and post-game questionnaires. They were given some pseudo code and asked questions that require them to understand the use of methods. Below, the detailed questions from the pre-game and post-game questionnaires are presented.

- **Pre-game Programming Question #1:** Pre-game Pseudo code # 3, was given and the question was:



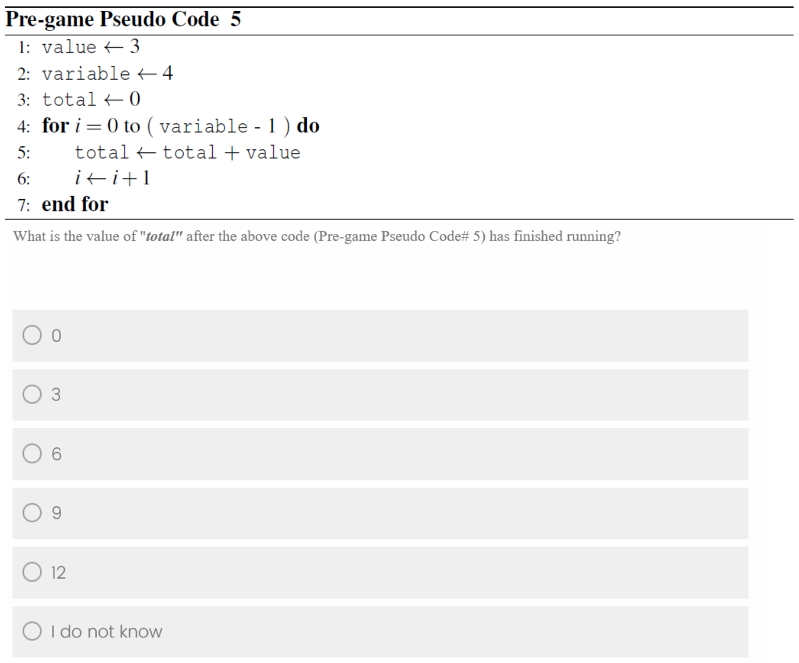
What will be the number of "**value**" after running the above pseudo code (Pre-game Pseudo Code# 3)?

- ○ 4
- ○ 8
- ○ 16
- ○ 24
- ○ 0
- ○ I do not know

- **Post-game Programming Question #5:** was used and already discussed above.

Combined Knowledge: To understand how variables and loops are used in combination in computer programming, participants were asked questions in the pre-game questionnaire. They were given a pseudo code and asked questions that required both variable and loop structure knowledge. The question is presented below.

- **Pre-game Programming Question #4:**



Pre-game Pseudo Code 5

```
value ← 3
variable ← 4
total ← 0
for $i = 0$ to ( variable - 1 ) do
    total ← total + value
    $i \leftarrow i + 1$
end for
```

What is the value of "***total***" after the above code (Pre-game Pseudo Code# 5) has finished running?

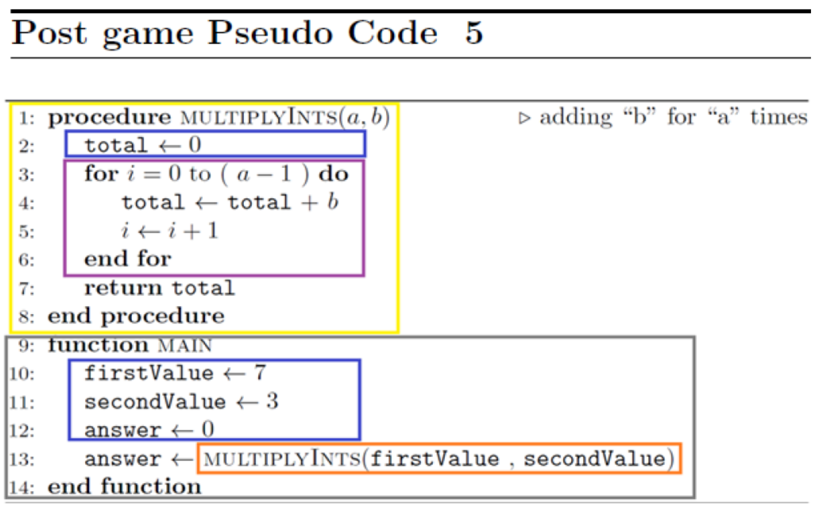
- 0
- 3
- 6
- 9
- 12
- I do not know

5.2.3.3 Conceptual Mapping of Programming Constructs:

The participants were asked several computer programming-related questions in the post-game questionnaire that are related to Program Wars *v*.2.0 gameplay. The main objective of these questionnaires is to find out whether participants could make connections with game cards and real-world programming. Post-game pseudo code # 5 was provided to answer post-game programming questions # 1 to # 3, and similar options are given for each question. The details of the conceptual mapping questions are given in the following section.

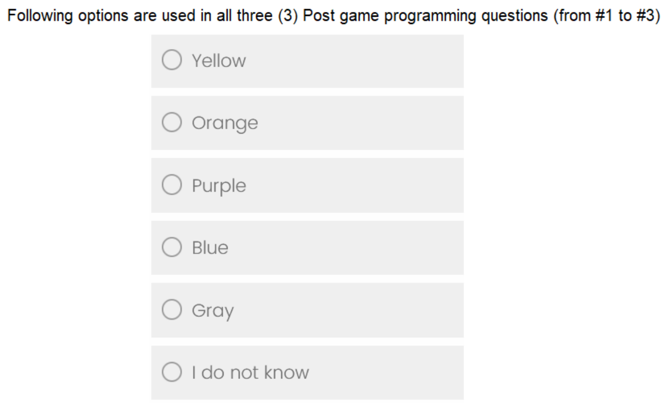
Variable Mapping: To understand whether a player could make the connection between a real world *variable* and the game's `Variable` card, the following question was asked in the post-game questionnaire.

- **Post-game Programming Question #2:** In the above pseudo code (Post-game Pseudo Code# 5), which box represents the `Variable` card from the game?



Post game Pseudo Code 5

```
procedure MULTIPLYINTS($a, b$)                         ▷ adding "b" for "a" times
    total ← 0
    for $i = 0$ to ( $a - 1$ ) do
        total ← total + $b$
        $i \leftarrow i + 1$
    end for
    return total
end procedure
function MAIN
    firstValue ← 7
    secondValue ← 3
    answer ← 0
    answer ← MULTIPLYINTS(firstValue , secondValue)
end function
```



Following options are used in all three (3) Post game programming questions (from #1 to #3)

- ◯ Yellow
- ◯ Orange
- ◯ Purple
- ◯ Blue
- ◯ Gray
- ◯ I do not know

Loop Mapping: To understand whether a player could make the connection between a real world *loop structure* and the game's `Repeat` card, the following question was asked in the post-game questionnaire.

- **Post-game Programming Question #3:** In the above pseudo code (Post-game Pseudo Code# 5), which box represents the `Repeat` card from the game?

Method Mapping: To understand whether a player could make the connection between a real world *method structure* and the game's `Method` card, the following question was asked in the post-game questionnaire.

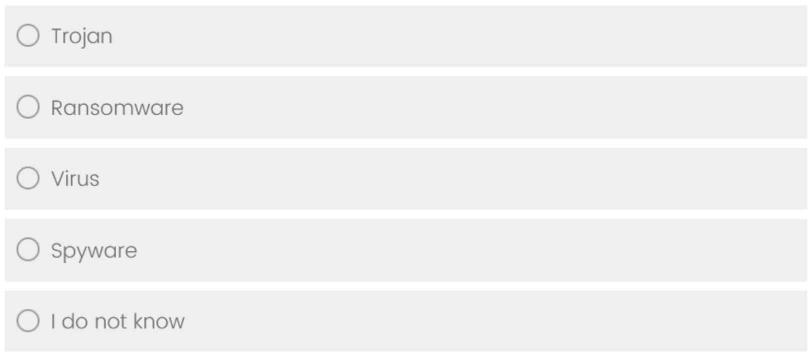
- **Post-game Programming Question #1:** In the above pseudo code (Post-game Pseudo Code# 5), which box represents the `Method` card from the game?

5.2.3.4 Cybersecurity Knowledge:

The participants were asked several cybersecurity questions both in the pre-game and post-game questionnaire. They were given some real-world scenarios and asked questions about the outcomes. Below, the questions related to cybersecurity are described.

Cybersecurity Scenario # 1:

- **Pre-game Cybersecurity Question # 1:** Suppose you received an email from someone you know and click on what looks like a legitimate attachment. After downloading and opening the file, you can not access your files. Every time you click on your folder, it shows an error message mentioning $100 needs to be paid to access all your files. What type of cyberattack is that?

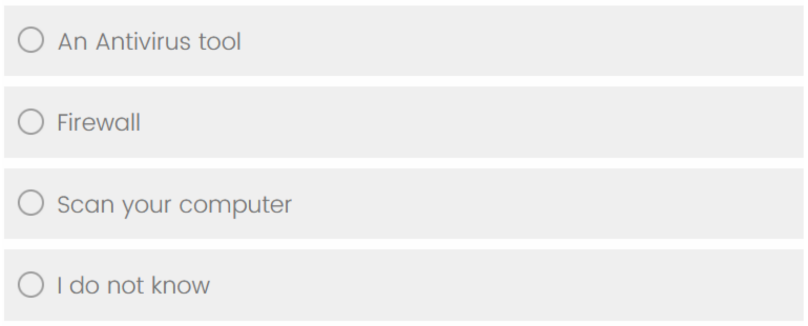


○ Trojan
○ Ransomware
○ Virus
○ Spyware
○ I do not know

- **Post-game Cybersecurity Question # 1:** Suppose you download a small game from a website that you found through Google. After downloading and installing the game, your computer locks up with a message that has a link to a webpage. When you go to the web page, it asks for your credit card information in order to unlock your computer. What type of cyberattack happened to you? [26]

Also, in the post-game questionnaire, participants were asked about the cyber defence technique related to the post-game Cybersecurity Question # 1.

- **Post-game Cybersecurity Question # 2:** Which cyberdefense tools could you have used to prevent the attack mentioned in Post-game Cybersecurity Question # 1?



○ An Antivirus tool
○ Firewall
○ Scan your computer
○ I do not know

Cybersecurity Scenario # 2:

- **Pre-game Cybersecurity Question # 2:** Suppose you want to open an online account on a website. After providing your personal information, it requires your bank account number and password for background verification. After providing that information, you lose $500 of money from your account. What type of cyberattack is that?[26]

- **Post-game Cybersecurity Question # 3:** Suppose you want to purchase a book from an unauthorized website. After providing your personal information, it requires your credit card information to confirm the order. After providing that information, you lose a certain amount of money from your account. What type of cyberattack is that?[26]

[26]Similar choices are given that has been provided for Pre-game Cybersecurity Question # 1.

Also, in the post-game questionnaire, participants were asked about the cyber defence technique related to the Post-game Cybersecurity Question # 3.

- **Post-game Cybersecurity Question # 4:** Which cyberdefense tools could you have used to prevent the attack mentioned in post-game Cybersecurity Question # 3?[27]

5.3 Results and Analysis

In this section, the results from the user study are discussed. The results are used to provide the answers to the following research questions:

R.Q. # 1: Do the refinements to the UI and gameplay of Program Wars improve a player's understanding of basic computer programming concepts?

This Research Question (R.Q.) is divided into three (3) sub-questions to evaluate the learning outcomes more precisely.

(a) Do the refinements to the UI and gameplay of Program Wars improve a player's knowledge of the Variable concept?

(b) Do the refinements to the UI and gameplay of Program Wars improve a player's knowledge of the Loop concept?

(c) Do the refinements to the UI and gameplay of Program Wars improve a player's knowledge of the Method concept?

R.Q. # 2: Does the refinement of the cybersecurity aspects of Program Wars lead a player to better understand real-life cybersecurity threats and how to combat them?

As earlier mentioned, a total of twenty-six (26) participants participated in the user study. In this section, the results of the user study, along with an in-depth analysis of the results, are presented. First, demographic information about the participants is discussed.

[27] Similar choices are given that has been provided for Post-game Cybersecurity Question # 2.

After that, knowledge regarding programming and cybersecurity is discussed. Then concept mapping results for programming knowledge are presented. Finally, the in-depth analysis regarding the programming knowledge is presented. The response of all the questionnaires is presented in Appendix D.

5.3.1 Demographic and Previous Experience

From the results, out of the 26 participants, 23 (88%) were from the 17 to 24 age group, and only 3 (12%) belonged to the age group 25 to 34. Participants were also asked about their education, and 22 (85%) reported completing their high school degree, 2 (8%) have their bachelor's degree, 1 (4%) had a graduate level of education, and 1 (4%) had an associates (2-years) degree.

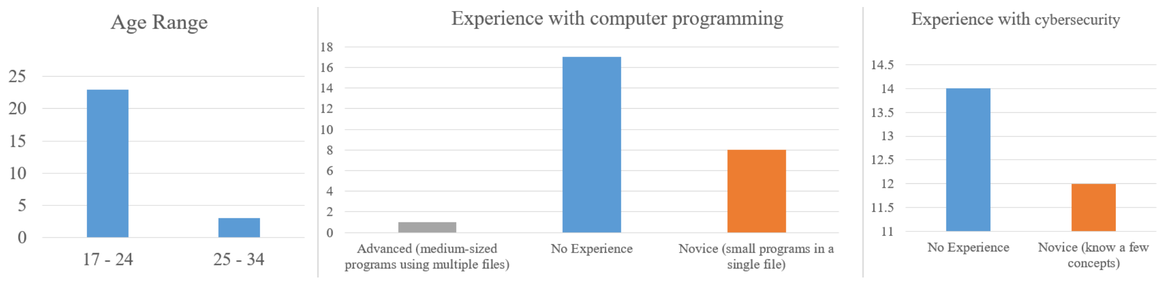


Figure 5.1: Demographic and Previous Experience results

Also, the results show that 17 (65%) have no experience with computer programming, 8 (31%) have little experience (experience with small computer programming files), and 1 participant had advanced programming knowledge (experience with a medium-size computer program using multiple files). In other words, almost all of the participants (95%) did not have any knowledge about computer programming or knew very little about it.

Regarding cybersecurity experience, 14 (54%) participants did not have any experience with cybersecurity, and 12 (46%) were novices (i.e. know little about cybersecurity). So, according to the data, no participant has a moderate or advanced level of knowledge in cybersecurity. Figure 5.1 showed the overall result for demographic analysis among the participants.

5.3.2 Change in Knowledge

The results regarding knowledge accumulation are divide into two categories. First is *Programming Comprehension* where paired programming questions from the pre-game and post-game questionnaire are analyzed. Second is *Cybersecurity* knowledge, where cybersecurity-related pre-game and post-game questions pairs are analyzed.

5.3.2.1 Programming Comprehension

To identify changes in the participant's programming knowledge, three fundamental concepts of programming have been focused on in this study. They are the *variable* concept, the *loop* concept and the *method* structure. One (1) subject reported advanced programming knowledge and was removed from the Programming Comprehension part, so the total participants for this part were twenty-five (25).

Variable Knowledge: As previously discussed in section 5.2.3.2, to identify *variable* knowledge among the participants *Pre-game Programming Question # 3* and *Post-game Programming Question # 5* were paired. Results show that in pre-game programming questions, out of 25 participants, a total of 9 (36%) participants were able to answer correctly.
Observation: Almost one-third of the participants showed pre-game knowledge of variables.
Also, the data shows that for the post-game programming question, out of 25 participants, a total of 16 (64%) participants gave the correct answer.
Outcome: It was observed that at least 7 (28%) participants have gained knowledge regarding variable concepts after playing Program Wars v.*2.0* .

Loop Knowledge: As previously discussed in section 5.2.3.2, to identify *loop structure* knowledge among the participants *Pre-game Programming Question # 2* and *Post-game Programming Question # 4* were paired. Results show that in the pre-game programming questions, a total of 4 (16%) participants were able to correctly answer the question.

Observation: Very few participants showed prior knowledge about loops.

Also, the data shows that for the post-game programming question, a total of 5 (20%) participants gave the correct answer.

Outcome: From the result, it was observed that 1 (4%) participant had gained knowledge regarding loop concepts after playing Program Wars v.*2.0 . However, after re-examining the questions asked regarding loops in the pre-and post-questionnaire, the questions may have been too challenging to understand for those with little prior programming experience (i.e. those who may not have seen a loop structure before).*

Method Knowledge: As previously discussed in section 5.2.3.2, to identify *method structure* knowledge among the participants *Pre-game Programming Question # 1* and *Post-game Programming Question # 5* were paired. Results show that in pre-game programming questions, no one was able to correctly answer the question.

Observation: No participants showed prior knowledge about method concept. After re-examining the questions asked regarding the method in the pre-questionnaire, participants required an understanding of logical operator concepts, which may have been too challenging to understand for those with little prior programming experience.

Also, the data shows that for the post-game programming question, a total of 16 (64%) participants gave the correct answer.

Outcome: From the result, it was observed that 16 (64%) participants had gained knowledge regarding method structure after playing Program Wars v.*2.0 .*

Combined Knowledge: To understand the combined concept of variable and loop structure, participants were asked question *Pre-game Programming Question # 4*. A total of 3 (12%) participants could answer this correctly.

Observation: Most of the participants found the question challenging because a significant portion of participants did not have prior knowledge about programming.

Summary

Figure 5.2 shows the changes in the correct answers between the pre-game and post-game questionnaire for various programming knowledge areas. From the diagram, it is observed that 28% of the participants improved in their knowledge of the variable concept. Also, the improvement in knowledge for the loop concept is 4%, and in the area of method concept, the improvement in knowledge is 62% among the participants. To answer **R.Q.# 1** it can be said that by playing Program Wars *v*.2.0 participants certainly gained knowledge about method structure. For the variable concept is not clear, and for loop structure, it is hard to tell whether the game did help or not.

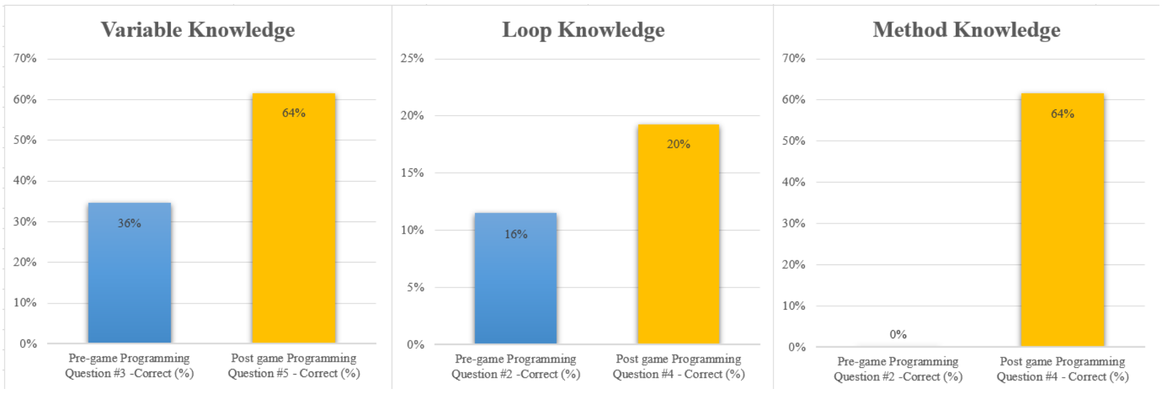


Figure 5.2: Correct answer of pre-game and post-game pair questions.

5.3.2.2 Programming Concept Mapping:

As previously mentioned, the three fundamental concepts of programming languages taught in Program Wars are the *variable* concept, the *loop* concept and the *method* structure. After participants completed Stage # 2 (playing Program Wars *v*.2.0 several times) of the user study, in Stage # 3 (Post-game questionnaire), they were asked questions related to the game's cards. These questions were asked to determine whether participants could connect their gameplay knowledge to real-world scenarios. The conceptual mapping between the gameplay experience and real-world knowledge is discussed in this section. Recall that three post-game programming questions (#1 to #3, details in Section 5.2.3.3) were based on their gameplay experience. In all of the three questions, the same pseudo code was used, with different colour rectangles used to highlight different areas of the pseudo code. Participants would have to use their gameplay knowledge to identify the appropriate concept (i.e. loop, variable or method structure) in the pseudo code section; each highlighted rectangle was a similar colour as the game's card that represents the corresponding concept.

Variable Concept Mapping [R.Q.#1(a)]: Post-game programming question # 2 was about the variable concept. A total of 15 participants were able to correctly connect the `Variable` card with the real-world programming structure.

Loop Concept Mapping [R.Q.#1(b)]: Post-game programming question # 3 was about the loop concept. A total of 14 participants were able to correctly connect the `Repeat` card with the real-world programming structure.

Method Concept Mapping[R.Q.#1(c)]: Post-game programming question # 1 was about the method concept. A total of 11 participants were able to correctly connect the `Method` card with the real-world programming structure.

Figure 5.3 shows the conceptual mapping outcomes from the post-game questions # 1 to # 3. Based on the participants' responses, Table 5.1 shows the total number of concepts that participants were able to correctly map between the game and a real-world scenario.

Table 5.1: Post-game Computer programming related mapping

	Relation Count	Correct %
Could relate all concept (Variable, Loop, Method) (100%)	8	32%
Could relate two-third concept (66%)	6	24%
Could relate one-third concept (33%)	4	16%
Could not relate any concept (0%)	7	28%
Total	**25**	

Out of 25 participants, 8 participants were able to relate all three (variable, loop and method) concepts with the gameplay, 6 participants were able to relate two concepts from

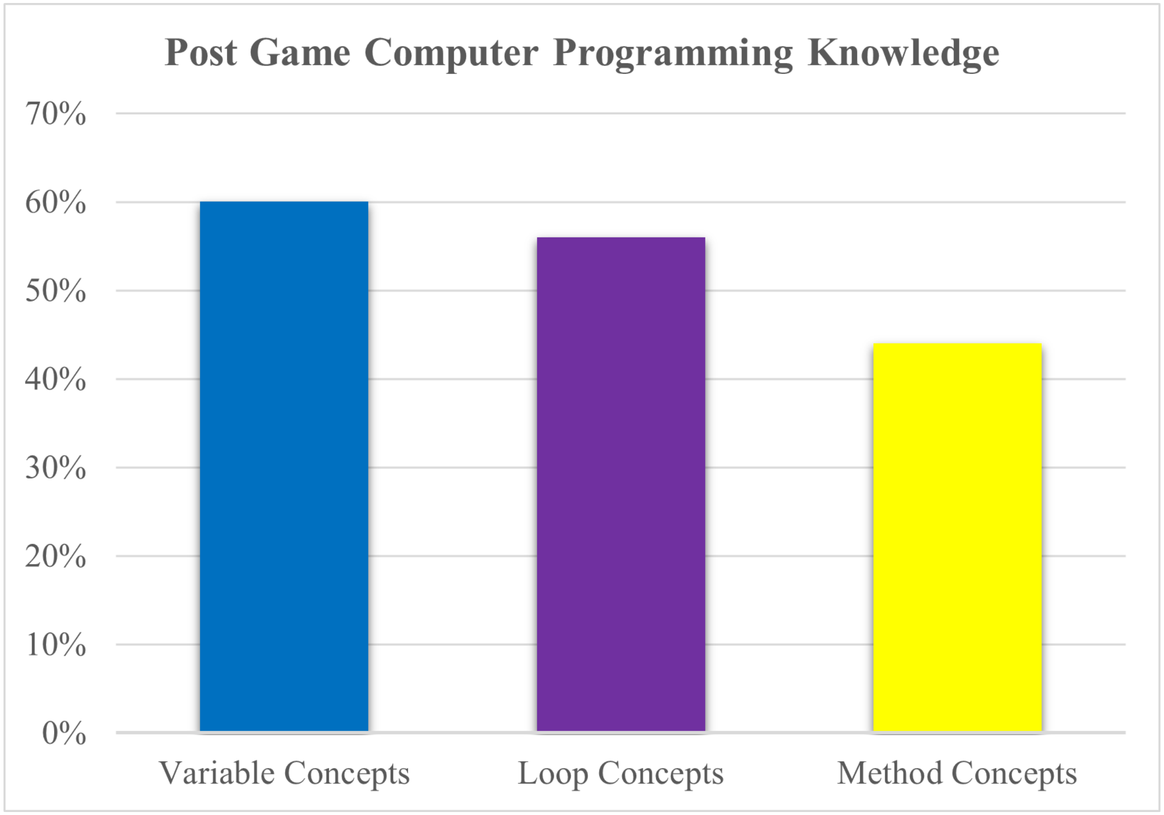


Figure 5.3: Post-game programming concept related mapping.

the gameplay, 4 participants were able to relate one concept from the gameplay. Finally, there were 7 participants who could not relate any concept with real-world programming after playing the game.

Summary

After analyzing the data, it was observed that among the three concepts, the highest number of participants, 15 (60%), missed out on the concept of method structure. Also, the participants who were able to make two connections maximum (4 participants out of 6) missed the method concept. Again 50% (2 participants out of 4) of the participants who could connect one concept successfully were able to identify the variable structure. By further examining the post-game programming questions, some of the pseudo code might have been challenging for the participants who had little knowledge about programming, considering that almost 30% of the participants could not connect a programming structure correctly with the gameplay.

5.3.3 Cybersecurity Knowledge:

To identify the cybersecurity knowledge of the participants, two (2) different real-world scenarios were given to the participants in both the pre-game and post-game questionnaire. The following section discusses the results of the cybersecurity knowledge among the participants.

Cybersecurity Scenario # 1: As previously discussed in Section 5.2.3.4, cybersecurity knowledge related questions *Pre-game Cybersecurity Question # 1* and *Post-game Cybersecurity Question # 1* were paired. Both of the scenarios represent the concept of Ransomware. The results show that for the pre-game cybersecurity question, a total of 6 (23%) participants were able to give the right answer.

Observation: The demographic result shows, most of the participants had very little concept regarding cybersecurity, which was demonstrated in the pre-game cybersecurity question where a large number of participants could not provide a correct answer.

In the corresponding post-game cybersecurity question, a total of 18 (69%) participants gave the correct answers.

Outcome: It is observed that at least 12 (46%) participants gained knowledge regarding the cyberattack (Ransomware) concepts after playing Program Wars v.*2.0* .

Also, in *Post-game Cybersecurity Question # 2*, participants were asked about the cyber defence technique regarding the cyberattack scenario presented in *Post-game Cybersecurity Question # 1*. A total of 14 (54%) participants were able to identify the cyberdefense technique correctly for this cyberattack after playing the game.

Cybersecurity Scenario # 2: As previously discussed in Section 5.2.3.4, cybersecurity knowledge related *Pre-game Cybersecurity Question # 2* and *Post-game Cybersecurity Question # 3* were paired. Both the scenarios represent the concept of Spyware. The result shows that in the pre-game cybersecurity question, a total of 5 (19%) participants were able to give the right answer.

Observation: As most of the participants had very little concept regarding cybersecurity, in the second scenario, a large number of participants could not provide the right answer.

In the post-game cybersecurity question, a total of 16 (62%) participants gave the correct answers.

Outcome: It is observed that at least 11 (42%) participants have gained knowledge regarding cyberattack (Spyware) concepts after playing Program Wars v.*2.0* .

Post-game Cybersecurity Question # 4, participants were asked about the cyber defence technique regarding the cyberattack scenario presented in *Post-game Cybersecurity Question # 3*. A total of 16 (62%) participants were able to identify the cyberdefense technique for this cyberattack correctly after playing the game.

Summary

Figure 5.4 shows the changes in the correct answers between the pre-game and post-game questionnaire for the two cybersecurity-related knowledge questions. From the diagram, it is observed that 46% of participants improve in their knowledge of the Ransomware cyberattack. Also, the improvement in knowledge for the Spyware cyberattack among the participants is 43%. It is also observed that after playing the game, more than 60% of participants were able to provide correct answers to the post-game questionnaire for preventing specific cyberattacks. To answer **R.Q. # 2**, it can be said that playing Program Wars *v.*2.0 improved the player's cybersecurity knowledge.

5.3.3.1 Cyberattack Concept Mapping:

The post-game cybersecurity questions (#1 to #4) were based on common real-world cyberattack (i.e. Trojan, Ransomware, Virus) and cyber defence (i.e. Computer Scan, Firewall) scenarios.

Program Wars *v.*2.0 was designed in a manner so that if the participant played the game

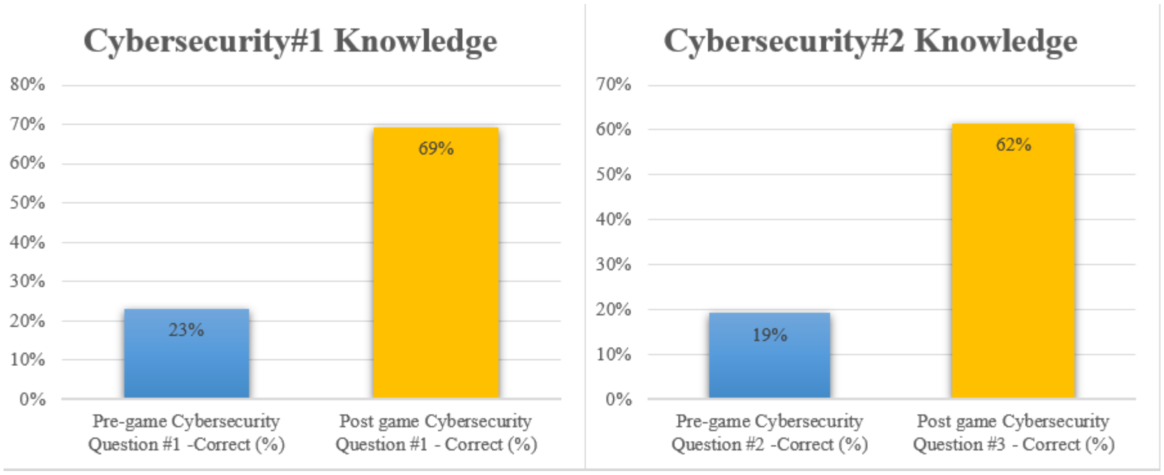


Figure 5.4: Correct answer of pre-game and post-game pair questions.

Table 5.2: Cyberattack related knowledge

	Count	Correct %
Correctly identified both cyberattacks	10	38%
Correctly identified one cyberattack	7	27%
Could not Correctly identified any cyberattacks	9	35%
Total	**26**	

several times, then he/she will be able to understand the basic concept of cybersecurity, which can be later used in a real-world scenario. Based on the participant's responses, a total of 10 participants were able to identify both cyberattacks accurately, 7 participants were able to identify at least one cyberattack accurately, and 9 participants were not able to identify any cyberattack correctly. Table 5.2 shows the overall statics of identifying different cyberattack.

Summary

The data for cybersecurity knowledge shows that 38% of participants successfully identified the cyberattacks in both of the scenarios, and 27% could identify at least one cyberattack successfully. So, the data shows that 65% of participants were able to relate the gameplay to a real-world cyberattack scenario.

5.3.4 Categorization of Learning Outcome

Based on the pre-game and post-game question responses participants knowledge change can be categorized into one of four states of learning according to the following formula.

$$\text{Knowledge Change (X)} = \text{Pre-game Question (X)} \cap \text{ Post-game Question (X)}$$
$$Where, X = \text{variable, loop, method, cyberattack, cyberdefense}$$

(a) **Previous Knowledge:** If the paired pre-game response and the post-game response are both 'Correct', then it will be assumed that the participant has previously known about the concept, and it will be marked as 'Previous Knowledge'.

(b) **Unclear:** If the pre-game response from a participant is 'Correct' and the paired post-game response is 'Incorrect' then it is unclear why the participant got the answer 'Correct' in pre-game questionnaire but then 'Incorrect' after playing the game. So, the knowledge change is marked as 'Unclear'.

(c) **Improved:** If the pre-game response from a participant is 'Incorrect' and the paired post-game response is 'Correct' then it is assumed that after playing the game their knowledge is updated. As a result, the knowledge change is marked as 'Improved'.

(d) **No Change:** If the pre-game response from a participant is 'Incorrect' and the paired post-game response is also 'Incorrect' then it is assumed that the player knowledge did not change by playing the game. As a result, the knowledge change is marked as 'No Change'.

A general overview of this categorization are presented in Table 5.3

5.3.4.1 Knowledge Change: Programming Concepts

Based on the Table 5.3 knowledge change detailed findings are discussed below.

Table 5.3: Knowledge Change Categories

Input # 1 (Pre-Game Question)	**Input # 2 (Post-game Question)**	**Knowledge Change**
Correct	Correct	**Previous Knowledge**
Correct	Incorrect	**Unclear**
Incorrect	Correct	**Improved**
Incorrect	Incorrect	**No Change**

Variable Concept: Based on the analysis, a total of 9 participants were considered to have 'Previous Knowledge' about the *variable* concept, and none of the participants was categorized as 'Unclear'. A total of 7 participants were categorized as 'Improved'. Finally, there were 9 participants whose knowledge regarding the *variable* concept did not improve after playing Program Wars *v*.2.0 . As a result, they were categorized as 'No Change'.

Loop Concept: Based on the analysis, a total of 2 participants were considered to have the 'Previous Knowledge' concept about the *loop* concept and 2 participants were categorized as'Unclear'. A total of 3 participants were categorized as 'Improved'. Finally, there were 18 participants whose knowledge regarding the *loop* did not improve after playing Program Wars *v*.2.0 . As a result, they were categorized as 'No Change'.

Method Concept: From analyzing the data, no participants were categorized as 'Previous Knowledge' or 'Unclear' about the *method* concept. Among the total participants, 16 participants were categorized as 'Improved'. Finally, there were 9 participants whose knowledge regarding the *method* did not improve after playing Program Wars *v*.2.0 . As a result, they were categorized as 'No Change'.

By analyzing the categorization of participants, 62% participants improved their knowledge about the *method* concept after playing Program Wars *v*.2.0 . A total of 28% participants improved their knowledge about the *variable* concept. Finally, 12% of the participants improved their knowledge about the *loop* concept. The overall statistics for each learning

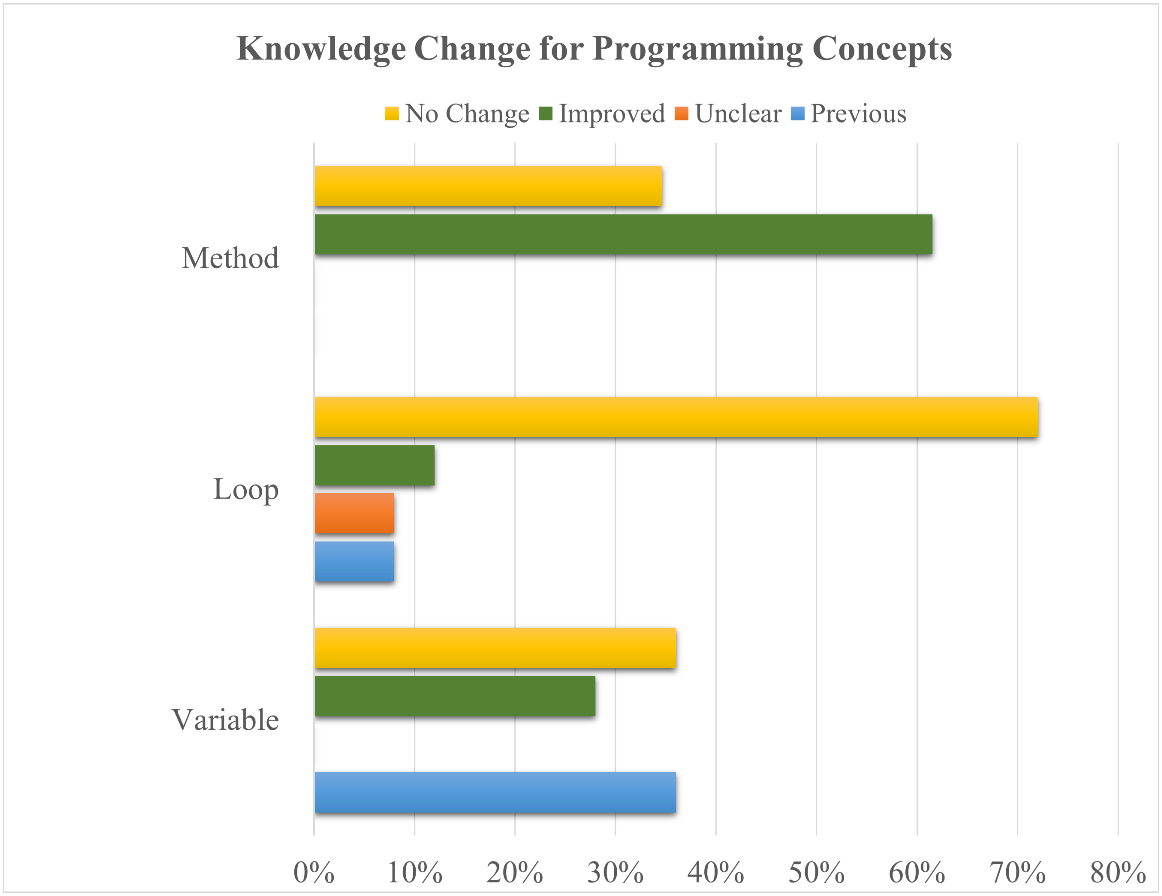


Figure 5.5: Categorization of Participants Knowledge Change (Programming Concepts)

outcome are presented in Figure 5.5.

As the gameplay might not be familiar to the participants and most of them were not familiar with programming logic, it was recommended that players should play the game at least three times. A total of 2 participants who played the game more than three times showed improvements in their computer programming knowledge for all three concepts (Variable, Loop and Method). A total of 7 participants played the game more than three times and improved their computer programming knowledge for at least two concepts. Five (5) of the participants played the game more than three times and improved their computer programming knowledge for at least one concept. Only 1 participant who played the game at least three times could not connect any of the programming concepts with a real-world scenario.

These results are shown in Figure 5.6. In all but one case, participants who played Program Wars at least three times saw some improvements in terms of programming knowledge. This would seem to indicate that if participants continued to play the game, they might see further improvements in their knowledge about programming.

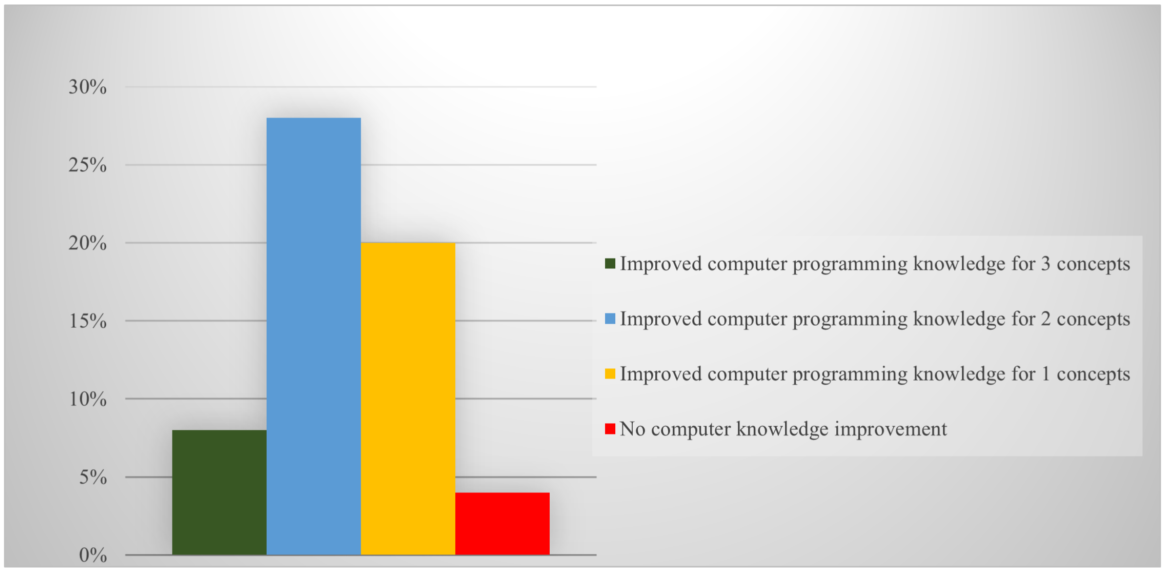


Figure 5.6: Computer Programming knowledge after playing the game at least three times.

5.3.4.2 Knowledge Change: Cybersecurity

In Program Wars *v.*2.0, two (2) cyberattack scenarios were presented. Based on the Table 5.3 knowledge change categorizations, the results are discussed below.

Cyberattack Scenario#1 (Ransomware): Based on the analysis, out of 26 participants, 4 participants were considered to have 'Previous Knowledge' about the *Ransomware* attack, and 3 participants were categorized as 'Unclear'. A total of 14 participants were categorized as 'Improved. Finally, there were 5 participants whose knowledge did not improve after playing Program Wars *v.*2.0 . As a result, they were categorized as 'No Change'.

Cyberattack Scenario#2 (Spyware): Based on the analysis, 1 participant was considered to have 'Previous Knowledge' about the *Spyware* attack, and 4 participants were categorized as 'Unclear'. A total of 15 participants were categorized as 'Improved'. Finally, there were 6 participants whose knowledge did not improve after playing Program Wars *v*.2.0 . As a result, they were categorized as 'No Change'.

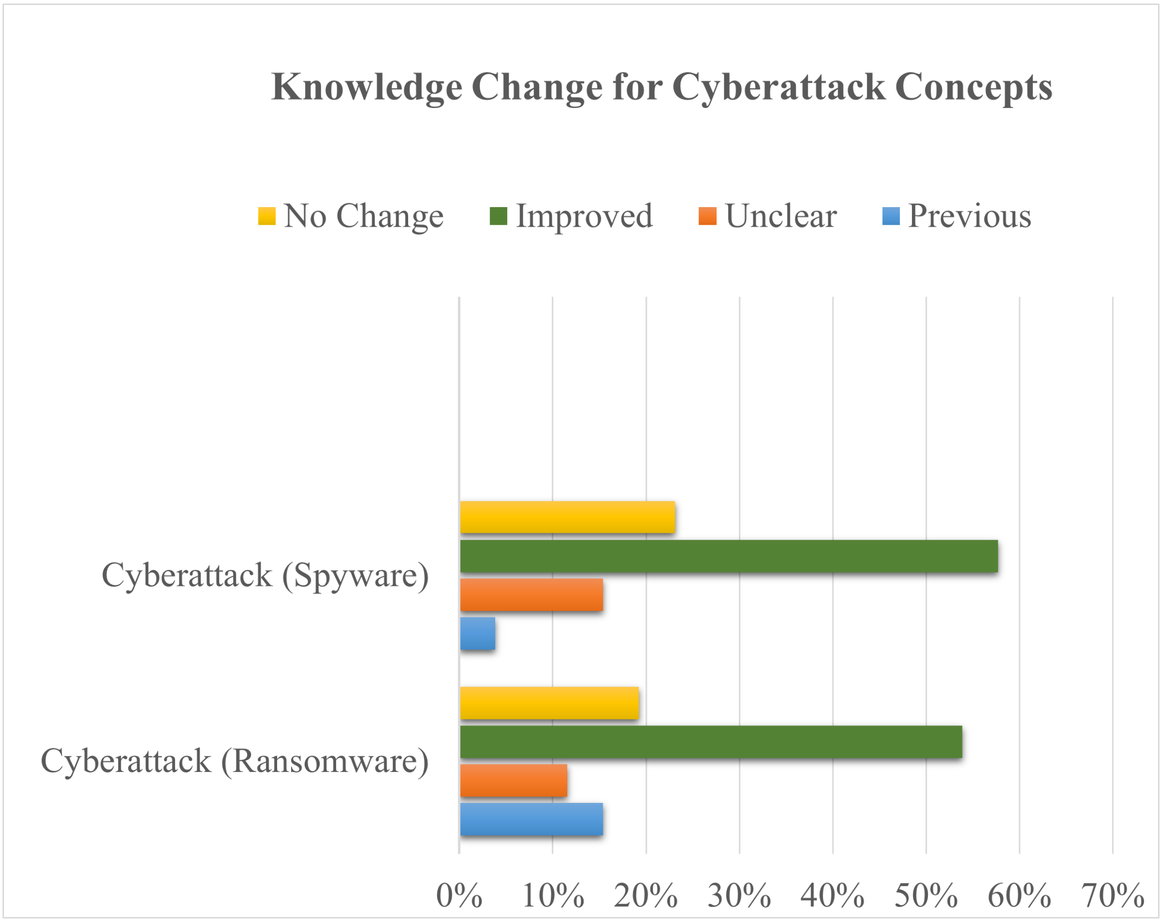


Figure 5.7: Categorization of Participants Knowledge Change (Cyberattacks)

By analyzing the categorization of participant knowledge changes, 54% of participants improved their knowledge about the *Ransomware cyberattack* after playing Program Wars *v*.2.0 , and 58% of participants improved their knowledge about the *Spyware cyberattack*. An overall statistics for each learning outcome is presented in Figure 5.7. As each participant showed improvement in knowledge about at least one cyberattack. This further confirms the findings from previous section and answered **R.Q. # 2** as "Program Wars does improve player's knowledge of cybersecurity".

As the gameplay might not be familiar to the participants and most of them were not familiar with various cyberattacks, it was recommended that players should play the game at least three times. A total of 6 participants who played the game more than three times showed improvements in their cyberattack knowledge for the two scenarios. Twelve (12) of the participants played the game more than three times and improved their cyberattack knowledge for at least one scenario. Eight (8) participants who played the game at least three times could not connect any of the cyberattack knowledge with a real-world scenario. These results are shown in Figure 5.8. Most of all who played Program Wars at least three times saw some improvements in terms of cyberattack knowledge. This would seem to indicate that if participants continued to play the game, they might see further improvements in their knowledge about the cyberattack.

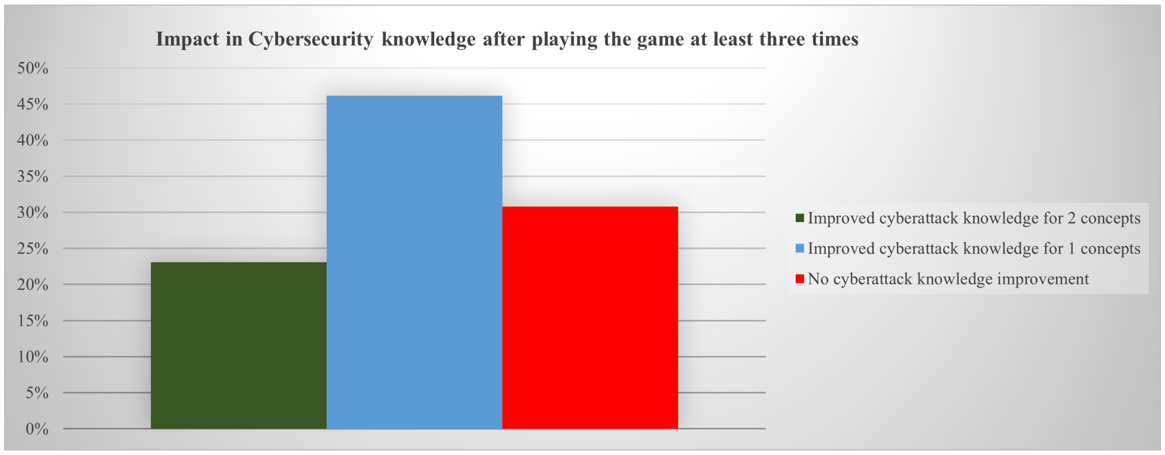


Figure 5.8: Cyberattack knowledge after playing the game at least three times.

5.3.5 Answering R.Q.# 1

To answer **R.Q.# 1:** “Do the refinements to the UI and gameplay of Program Wars improve a player’s understanding of basic computer programming concepts?” – as completely as possible, The following analysis was done.

(a) **Playing Program Wars helped:**

If a participant was able to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘Improved’, then it is considered as evidence that Program Wars does help in learning that concept and can be referred to ‘True Positive (TP)”. Similarly, if a participant was able to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘Previous Knowledge’, then it is considered as evidence that the participant understood the game, but their knowledge may not have improved. This situation can also be referred to as a “True Positive (TP)”. These results are taken to answer **R.Q.# 1** in the positive.

(b) **Understood Program Wars**

If a participant was able to conceptually map a Program Wars concept to a real-world programming structure, but the same participant’s knowledge of that concept was categorized as ‘Unclear’, then it is considered as evidence that participants understood the game, but their knowledge improvement is unclear. This also refer that participants can understand the gameplay with the real world scenario and in pre-game questionnaire related to the subject they did answer correctly and somehow could not answer correctly in post game questionnaire related to the subject. As result it can be conclude that the participant’s did understand the concept of Program Wars and can be referred to as a “False Negative (FN)”. Similarly, if a participant was able to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘No Change’, then it is considered as evidence that the participant knowledge is contradictory about Program Wars. This scenario occurred when in both pre and post game questionnaire’s participants failed to answer correctly related to the subject. But they did understand the game logic and connect the concept with real world scenario which can also be considered as a “False Negative (FN)”. These results are taken to answer **R.Q.# 1** in the positive.

(c) **Unclear Outcome**

If a participant was unable to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘Improved’, then it is considered as contradictory evidence and can be referred to as a “False Positive (FP)”. Similarly, if a participant was unable to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘Previous Knowledge’, then it is considered as evidence that the participant’s knowledge about Program Wars is confusing. This state can also be referred to as a “False Positive (FP)”. These results are taken to answer **R.Q.# 1** in the negative.

(d) **Program Wars did not help**

If a participant was unable to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘Unclear’, then it is considered as that Program Wars was not helpful and is referred to as a “True Negative (TN)”. Similarly, if a participant was unable to conceptually map a Program Wars concept to a real-world programming structure and the same participant’s knowledge of that concept was categorized as ‘No Change’, then it is considered as evidence of no knowledge change. This state can also be referred to as a “True Negative (TN)”. These results are taken to answer **R.Q.# 1** in the negative.

Table 5.4 summarises this analysis methodology.

Table 5.4: Summary of Analysis for answering **R.Q.# 1**

Knowledge Change	**Conceptual Mapping**	**Answer to R.Q.# 1**	**Support for Conclusion**
Improved	Able to Connect	Playing Program Wars helped	True Positive (TP)
Previous	Able to Connect		
Unclear	Able to Connect	Understood Program Wars	False Negative (FN)
No Change	Able to Connect		
Improved	Unable to Connect	Unclear Outcome	False Positive (FP)
Previous	Unable to Connect		
Unclear	Unable to Connect	Program Wars did not help	True Negative(TN)
No Change	Unable to Connect		

R.Q. #1 (a): Variable Concept

According to the analysis, Program Wars helped a total of 10 participants in learning about the variable concepts, and 5 participants understood the variable concepts from playing Program Wars.

R.Q. #1 (b): Loop Concept

According to the analysis, Program Wars helped a total of 2 participants in learning about the loop concepts, and 12 participants understood the loop concepts from playing Program Wars.

R.Q. #1 (c): Method Concept

According to the analysis, Program Wars helped a total of 7 participants in learning about the method concepts, and 4 participants understood the method concepts from playing Program Wars.

Table 5.5: Results of analysis for answering **R.Q.# 1**.

Decision	**Variable Concept**	**Loop Concept**	**Method Concept**
Playing Program Wars helped (TP)	10	2	7
Understood Program Wars (FN)	5	12	4
Unclear Outcome (FP)	4	8	5
Program Wars did not help (TN)	6	3	9

The overall statistics for each programming concept (i.e. variable, loop, method) are shown in Table 5.5. Figure 5.9 shows a different view of these results.

To answer **R.Q. # 1**, the following equations were used:

$$TP + FN = \text{ Program Wars helped in improving knowledge}$$

$$TN + FP = \text{ Program Wars did not help in improving knowledge}$$

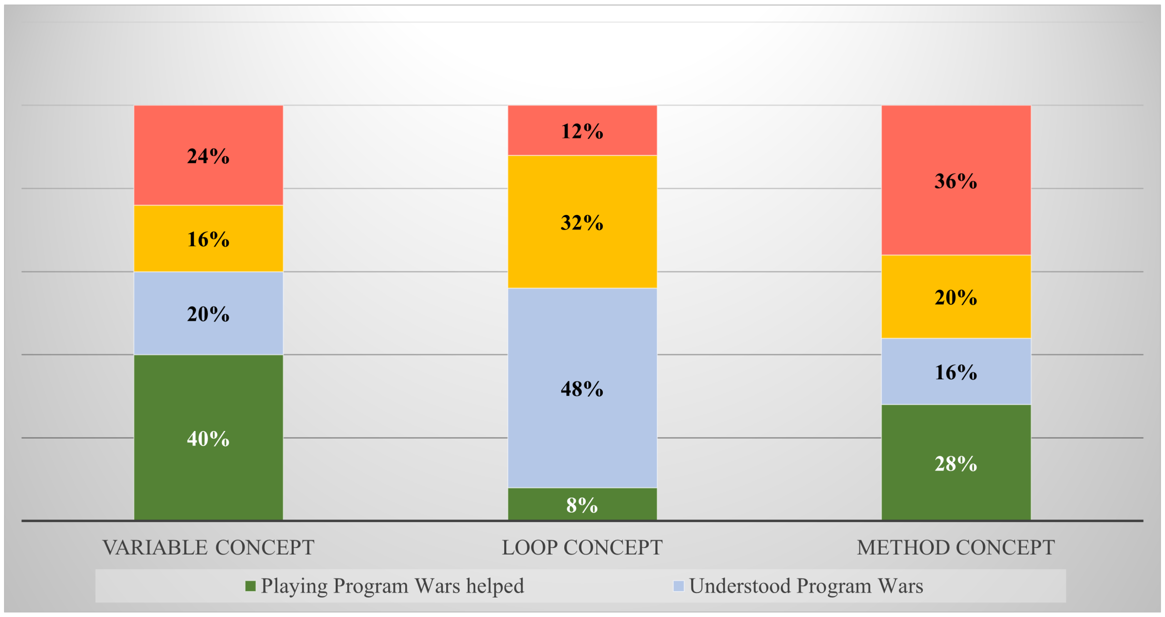


Figure 5.9: Programming concept learning outcome

From Table 5.5, it can be stated that Program Wars helped 15 participants to understand the *Variable* concept and Program Wars did not help 10 participants. Also, Program Wars helped 14 participants to understand the *Loop* concept, and Program Wars did not help 11 participants. Finally, Program Wars helped 11 participants to understand the *Method* concept and Program Wars did not 14 participants. Figure 5.10 shows the statics about how Program Wars helped to learn the three programming concepts.

Summary

From the results, it is observed that for 60% of the participants, Program Wars helped to understand the *Variable* concepts. Therefore, to answer **R.Q.# 1(a)** it is most likely true that Program Wars helped to understand the *Variable* concept. Similarly, for 56% of the participants, Program Wars helped to understand the *Loop* concepts. Therefore, to answer **R.Q.# 1(b)** that it is most likely true that Program Wars helps to understand the *Loop* concept. Finally, it was found that for 44% of the participants, Program Wars helped to understand the *Method* concepts. This means that the answer to **R.Q.# 1(c)** is that Program Wars may not have helped to understand the *Method* concept.

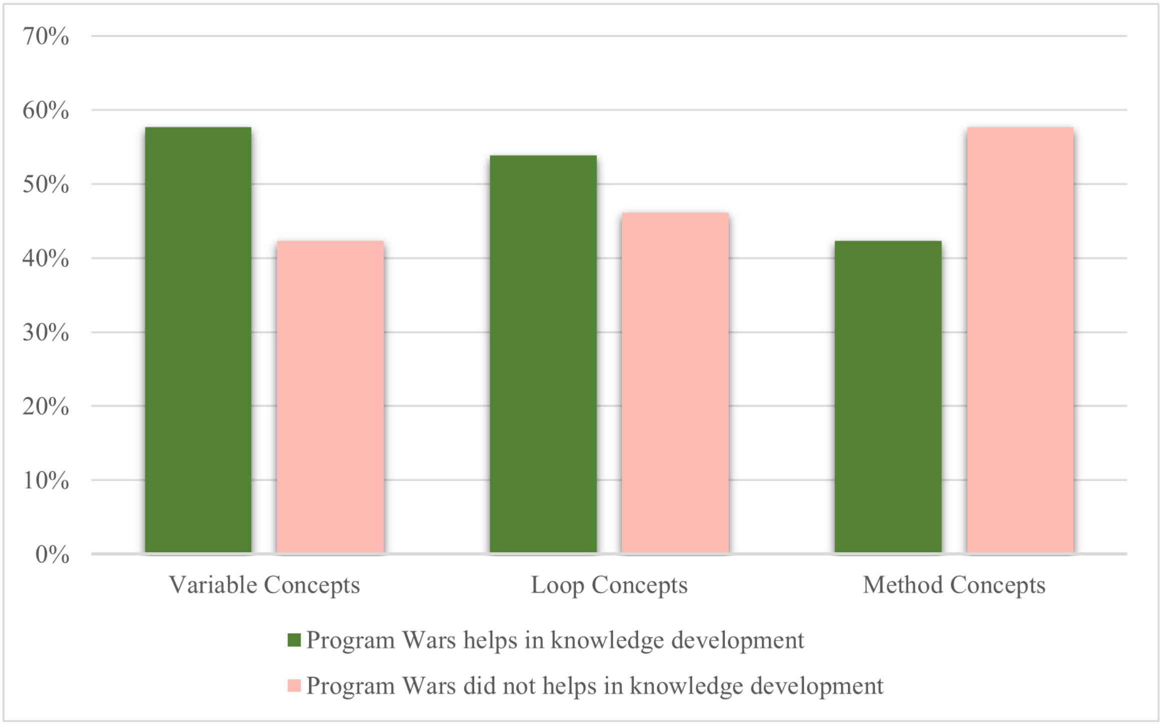


Figure 5.10: Learning outcome for Programming Concepts (Variable, Loop, Method)

5.3.6 Result Comparison between *v*.1.0 and *v*.2.0

In Program Wars *v*.1.0 [37] there was a user study similar to Program Wars *v*.2.0 . In this section two user study results are compared.

Table 5.6: Result comparison between two version of Program Wars

Programming Concept	**v 1.0**	**v 2.0**
Variable	**67%**	60%
Loop	47%	**56%**
Method/Group	31%	**44%**

Table 5.6 presents the detailed comparison between Program Wars *v*.1.0 and Program Wars *v*.2.0 in terms of three programming concepts (i.e. variable, loop, method). Based on this comparison, Program Wars *v*.1.0 was performed better for learning the variable concept. But after analyzing the participant's demographic responses for v.1.0, it is observed that most of the participants had some experience with programming concepts. On the

other hand `v.2.0` participants' knowledge about programming was very limited or almost none. So, in terms of numerical value `v.1.0` might looks better but `v.2.0` improvement is also significant. Also, Program Wars *v.*2.0 performed better for learning about loops and methods.

5.3.7 Threats to validity

In answering **R.Q. #1**, it should be noted that the 'False Negative' and 'False Positive' results could be considered to fall into a "weak support" category. This means that Program Wars may be more effective at teaching the programming concepts than the analysis shows. Similarly, Program Wars may be less effective than the analysis shows. Examining the results for each concept, it is observed that for the variable concept, 16% of participants fall into the "weak support" category, 32% of participants fall into the "weak support" category for the loop concept, and 20% of participants fall into this category for the method concept.

In answering **R.Q. #2**, participants may not choose the Ransomware and Spyware cyberattack cards at the beginning of the game (the game setting was set to independent to ensure the unsupervised concept of the user study). As a result, participants might not be familiar with the above-mentioned cyberattack.

5.3.8 Participant Feedback

In the post-game questionnaire, participants were asked several feedback questions regarding their experience playing Program Wars.

It is found out that 88% of participants liked Program Wars, and 65% agreed that they would suggest playing Program Wars to their friends. Regarding self-assessment in the area of computer programming and cybersecurity, a large portion was not sure that playing Program Wars improved their understanding. A possible reason is that they were not provided immediate feedback on answered questions, whether they got questions answers correct or not after they completed the questionnaire. However, judging by the results in Section 5.3, a lot of the participants showed improvements in their knowledge.

Table 5.7: Summary of post-game Feedback

	Strongly Agree	Agree	Unsure	Disagree	Strongly Disagree
Enjoyed playing Program Wars	11	12	1	1	1
Percentage (%)	42%	46%	4%	4%	4%
Would recommend playing Program Wars to friends	5	12	4	3	2
Percentage (%)	19%	46%	15%	12%	8%
Playing Program Wars improved individual knowledge of computer programming	1	3	13	7	2
Percentage (%)	4%	12%	50%	27%	8%
Playing Program Wars improved individual knowledge of cybersecurity	1	10	7	7	1
Percentage (%)	4%	38%	27%	27%	4%
After playing Program Wars, I am more confident in my programming knowledge	0	3	8	10	5
Percentage (%)	0%	12%	31%	38%	19%
I would continue to play Program Wars to improve my programming and cybersecurity knowledge	2	9	7	5	3
Percentage (%)	8%	35%	27%	19%	12%

Table 5.7 shows the overall feedback from the participants. A total of 11 participants strongly agreed that they enjoyed playing Program Wars and 12 participants also agreed with the statement. Only 1 participant did not like Program Wars. In response to recommending Program Wars to friends, a total of 17 participants agreed with the statement, and 2 participants said they would not recommend it to their friends. Responses regarding how participants felt that Program Wars improved their individual knowledge of computer programming showed a total of 4 participants agreed it improves their skills in programming and 13 participants were not sure about it. A total of 2 participants did not think the game improved their programming knowledge. Regarding improvements of individual knowledge to cybersecurity, responses show that 11 participants felt that their knowledge improved, and 1 participant felt strongly that his/her knowledge did not improve. Regard-

ing feeling confident in programming knowledge after playing Program Wars, 3 participants felt more confident in their programming knowledge after playing Program Wars, and 8 participants were not sure after playing Program Wars. Participants were asked whether they will play Program Wars in future, with a total of 11 participants providing positive feedback that they will play Program Wars in future to improve their knowledge in programming and cybersecurity concepts and a total of 3 participants stating that they will not play Program Wars in the future.

Finally, participants were asked to rate Program Wars. Out of 26 participants, 5 participants gave 5 *Star* to Program Wars, eleven (11) participants gave 4 *Star*, 8 participants gave 3 *Star*. And the game got 2 *Star* and 1 *Star* from 1 participant each. Figure 5.11 shows the overall rating of Program Wars among those provide by the participants.

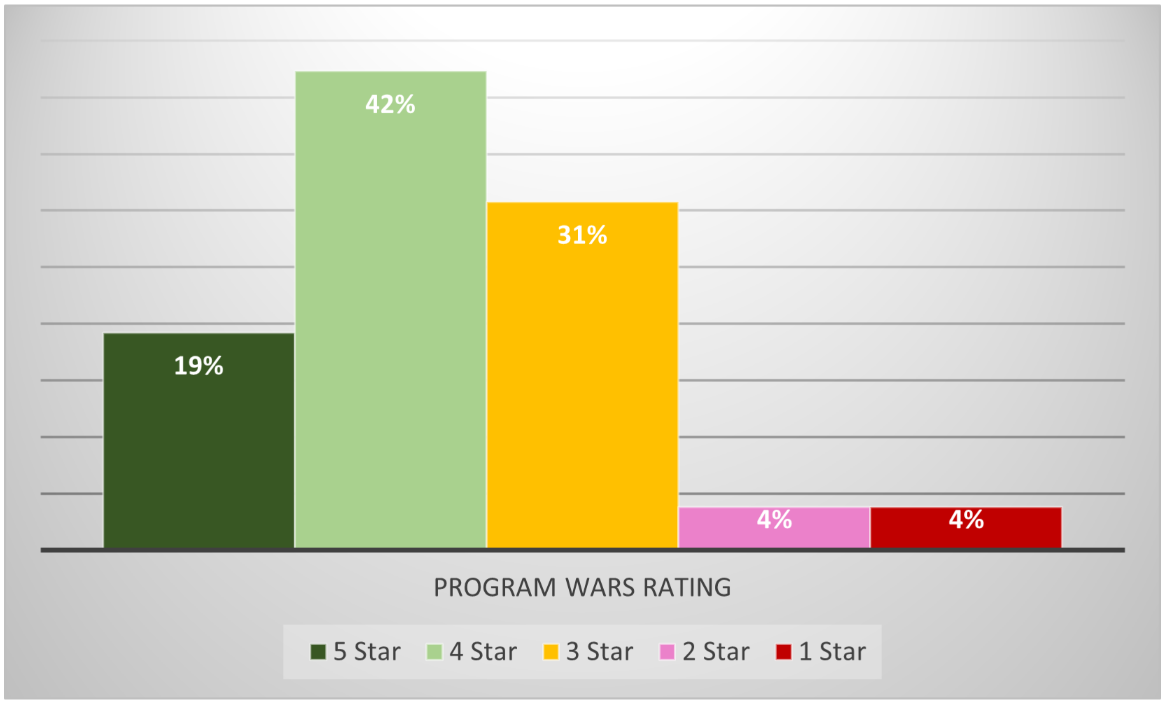


Figure 5.11: Program Wars Rating.

5.3.9 Participant Comments

When asked for any additional comments, a couple of participants indicated it was a 'good game,' and they had fun while playing the game. A couple of the participants found the User Interface (UI) to be complex, along with the game rules. However, it should be noted that for any new card game, the initial couple of rounds are typically a challenge as players learn the rules of that game. One of the participants remarked that the game has “A very good concept to teach programming & cybersecurity basics and ideas, The game is really enjoyable, and it enhanced my knowledge regarding basic programming and cybersecurity.”

5.4 Summary

The results of the user study showed, after playing Program Wars *v.*2.0 , participant’s knowledge about variables, loops and methods improved. Overall it can be said that Program Wars *v.*2.0 helps the participants to improve their knowledge about programming (**R.Q.# 1**). A comparison also showed that Program Wars *v.*2.0 performed better than Program Wars *v.*1.0 for both in loop and method concepts. In terms of cybersecurity concepts (**R.Q.# 2**), participants knowledge improved after playing Program Wars *v.*2.0 . Finally, feedback showed that more than 80% of the participants liked Program Wars *v.*2.0 and 61% of the participants gave the game 4 or more stars.

Chapter 6

Program Wars 3.0

The objective of Program Wars *v.*3.0 is to explore further the effectiveness of teaching fundamental software engineering concepts using GBL. Specifically, Program Wars *v.*2.0 was augmented to introduce the fundamental concepts of the Software Development Life Cycle (SDLC) and the Iterative Software Development Methodology (ISDM) (a.k.a. Agile software development), thus creating Program Wars *v.*3.0 . Although there have been other card-based games that concentrated on teaching ISDM, these efforts focus either on a specific Agile methodology, such as Scrum, teamwork aspects, or both [32, 33, 45]. But Program Wars *v.*3.0 is different in allowing the user to practice the ISDM by implementing a representation of a computer program, thereby making it independent of any specific agile methodology.

Software Development Life Cycle (SDLC): This can be referred to as a method that produces software with the essential quality in the least amount of time. With the help of SDLC, an institution can build a quality output that is well-tested and ready for production use. SDLC is a process followed for a software project within a software organization.

It consists of a detailed plan describing how to develop, maintain, replace, or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process. Figure 6.1 shows the phases of SDLC.

Iterative Software Development Methodology (ISDM): In the Iterative model, the iterative process starts with a simple implementation of a small set of software requirements.

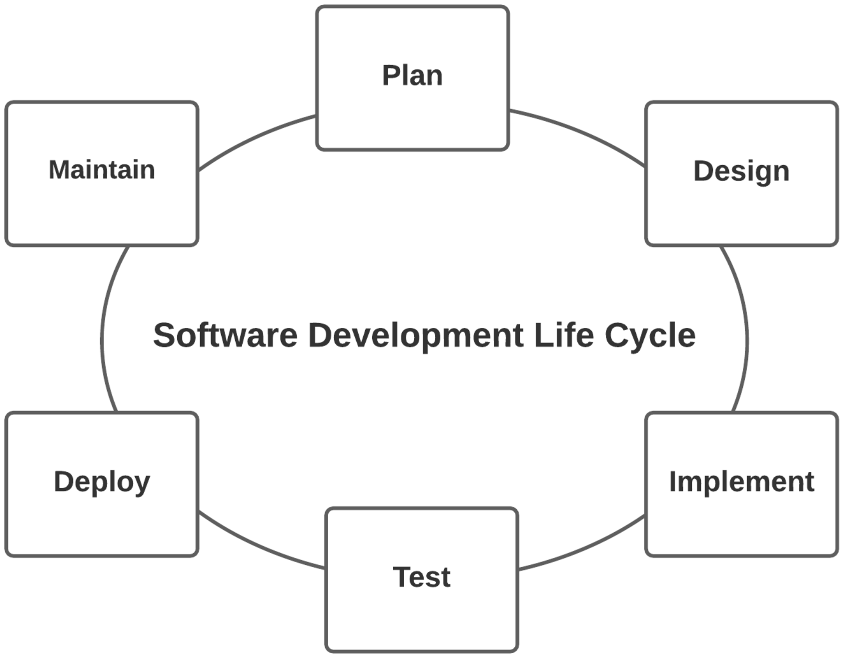


Figure 6.1: Software Development Life Cycle (SDLC)

After that, it iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed. An iterative life cycle model does not attempt to start with a complete specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further needs. This process is then repeated, producing a new version of the software at the end of each iteration. Figure 6.2 shows a general concept of ISDM model.

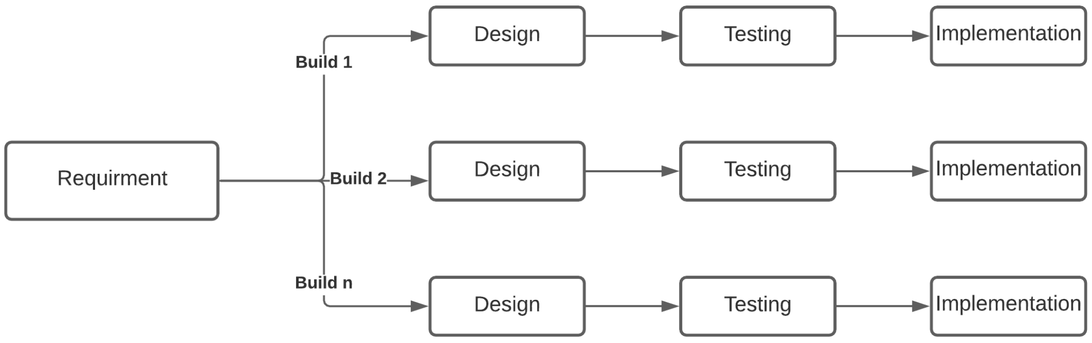


Figure 6.2: Iterative Software Development Methodology (ISDM)

The question to be answered by this research is:

R.Q. # 3 How can Program Wars be modified to teach the basics of the SDLC and ISDM?

The answer to **R.Q. # 3** is discussed in the rest of this chapter.

6.1 Introducing SDLC and ISDM using Program Wars

Program Wars *v.*3.0 provides experiential learning of the SDLC and ISDM concepts. In Program Wars *v.*3.0 , the gameplay has been changed, and a new mode is added, which is called the *ISDM mode*. The new mode breaks the game into four phases. The phases are: *Requirement*, *Design*, *Implementation* and *Testing*. *ISDM* is introduced by the overall gameplay (i.e. the sprints). Each sprint is composed of the phases from the SDLC. In the new mode, players still need to compete for points, but the game is broken into three rounds called *sprints*[28]. The game ends when all three sprints have been completed. In *ISDM mode*, players still need to build programs to get points, but previous objectives are replaced by individual requirements with one objective for each sprint. Completing these objectives awards additional points to a player. Each player is given a specific requirement in each sprint to complete and builds their own customized deck to help them complete those requirements. A key focus of the *ISDM* mode is to familiarize the player with the SDLC. As a result, Program Wars *v.*3.0 , does not end immediately after one round rather, it gives the player several chances to improve their coding concepts by setting up various objectives. Figure 6.3 shows a flow diagram of the Program Wars *v.*3.0 gameplay. The additional elements and modified concepts are described in the appropriate section below.

6.2 Game Setup

In Program Wars *v.*3.0 , the mode drop-down has an additional entry for *ISDM mode*. Unlike Program Wars *v.*2.0 where players immediately start building their program, players

[28]In Agile product development, a sprint is a set period of time during which specific work has to be completed and made ready for review [46]

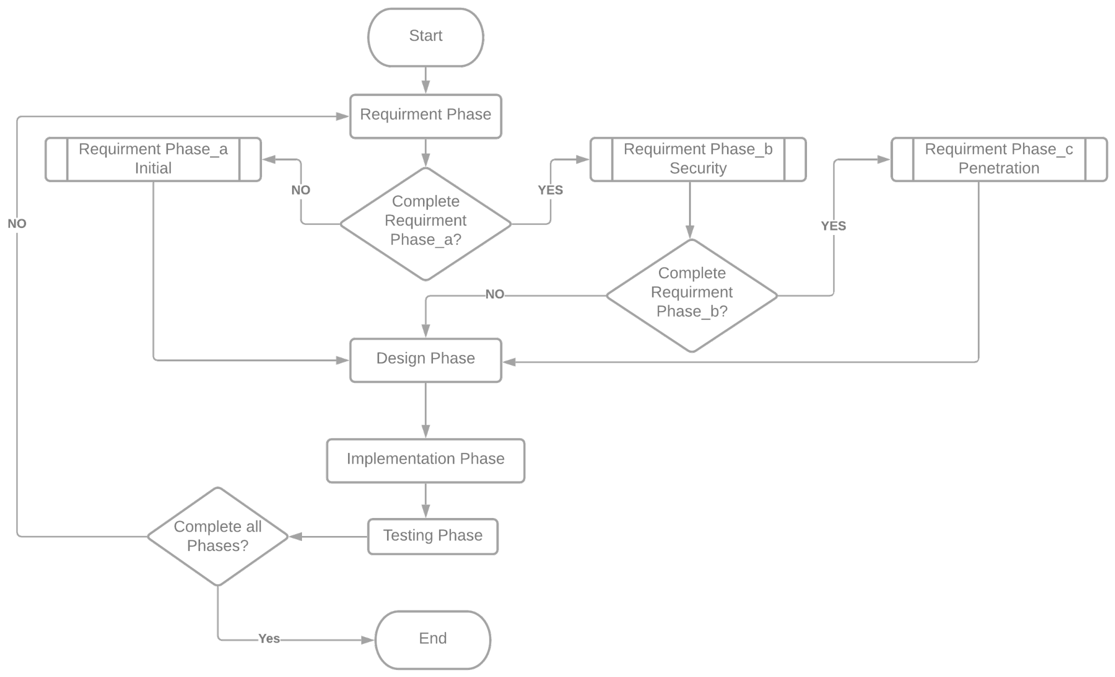
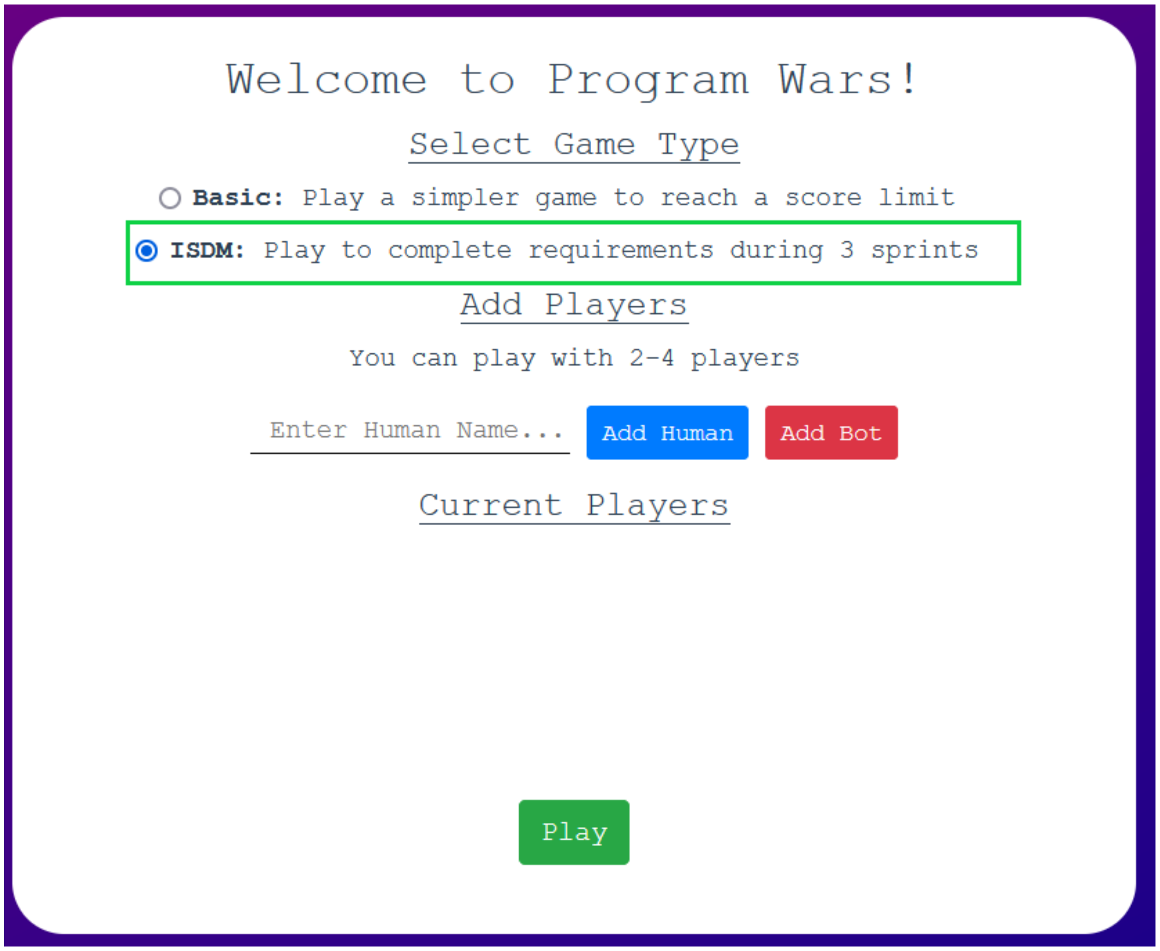

Start
Requirment Phase
Requirment Phase_a Initial
NO
Complete Requirment Phase_a?
YES
Requirment Phase_b Security
Requirment Phase_c Penetration
NO
YES
Complete Requirment Phase_b?
NO
Design Phase
Implementation Phase
Testing Phase
Complete all Phases?
Yes
End

Figure 6.3: Gameplay diagram of Program Wars *v.*3.0

will be advanced to the *Requirement* phase. Figure 6.4 shows the initial game setup screen for Program Wars *v.*3.0 .

The phases make up each sprint. Initially, players will be presented Sprint 1, where they focus on the requirements for the `Initial` stage. In the *Design* phase, players have to design their deck based on the requirements that have been provided by the `Initial` stage. In this phase, players will select cards from the main deck that they believe will help them to achieve their sprint goals. After completing the *Design*, phase player will be directed to the *Implementation* phase, where all players will have the same number of turns to either achieve the sprint goal or get as close as they can. This phase is similar to the game of Program Wars *v.*2.0 , except that the game will not end when the players reach a certain score. After Sprint 1 is completed, the player will be directed to *Testing* phase, where sprint objectives will be tested, and sprint points (i.e. including bonus points) will be calculated and added to the player score. After the *Testing* phase, the player's progress will be saved, and players will be returned to the *Requirement* phase for Sprint 2. In this


Welcome to Program Wars!

Select Game Type

○ **Basic:** Play a simpler game to reach a score limit

● **ISDM:** Play to complete requirements during 3 sprints

Add Players

You can play with 2-4 players

Figure 6.4: Starting for Program Wars *v.*3.0

sprint, in the *Requirement* phase player will be presented with `Security` objectives. The same workflow that has been maintained in Sprint 1 will be followed in Sprint 2. After Sprint 2, players again returned to the requirements phase for the final sprint. In this sprint `Penetration` objectives are given as requirements. After the *Design* and *Implementation* phases, the player will be directed to the final *Testing* phase where bonus points will be calculated.

6.2.1 Requirement Phase

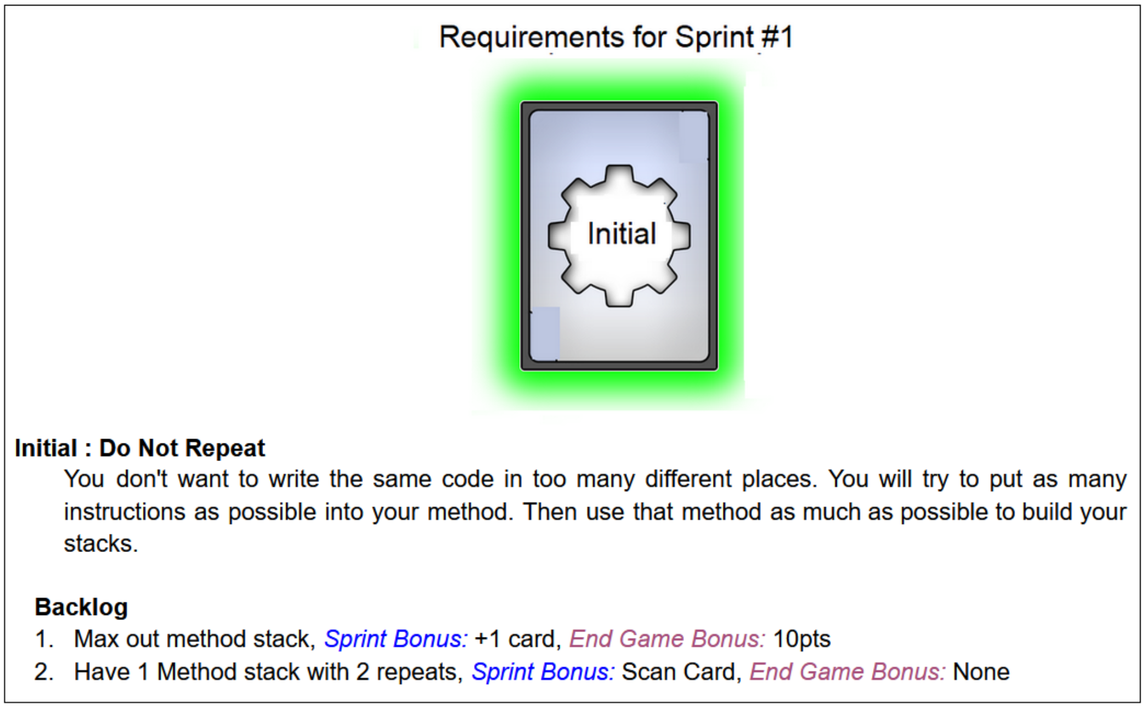
In each requirement phase, players are given requirements based on the sprint goal. These requirements are sets of individual objectives that award points and other bonuses to the player when completed. They represent the requirements that would be given to a developer or team by a customer for a new software project or as a part of an existing software project. Each requirement has three objectives to complete during the game. The given objectives in each sprint are checked in the *Testing* phase to record a player's progress. If a player completes an objective before the end of the sprint it is associated with, the player receives an additional bonus. These bonuses are not points but instead give the player a useful card or status effect. Some sprint bonuses are awarded immediately upon completing the objective, and others are awarded at the end of the sprint, based on how they might help the player. When players start the *Requirements* phase, they are taken to a new page that has the requirements presented as a card. The card appears on the top portion of the screen and can be scrolled through and selected to see the specific objectives for the sprint and bonuses granted for their completion. Each sprint has its own requirements. The details of the three sprints requirements are discussed below.

6.2.1.1 Sprint 1 - "Initial"

In the 'Initial' sprint, the requirement for the players is to use the method card so that redundant code can be reduced. This teaches the SE design principle of 'Do Not Repeat Yourself' (a.k.a. "D.R.Y.").

Figure 6.5 shows the requirement card for this phase, which explains the overall objective of the sprint. The main objective, game plan and bonus for `Initial` are:

- Objective: The main objective of this sprint is to encourage a player to create a program that is modular by creating methods. Also introduces the loop concept into the program.
- Game plan: For achieving the goals of Sprint 1, the player has to use `Method` and `Repeat` cards several times. So, the player has to concentrate on these cards in the *Design* phase.
- Bonus: If a player maximizes the method stack, then they can play with an extra card only for the current sprint, and 10 points will be added to their total. Also,


Requirements for Sprint #1

Initial

Initial : Do Not Repeat

You don't want to write the same code in too many different places. You will try to put as many instructions as possible into your method. Then use that method as much as possible to build your stacks.

Backlog

1. Max out method stack, *Sprint Bonus:* +1 card, *End Game Bonus:* 10pts
2. Have 1 Method stack with 2 repeats, *Sprint Bonus:* Scan Card, *End Game Bonus:* None

Figure 6.5: The Requirements Card of Sprint 1(Initial)

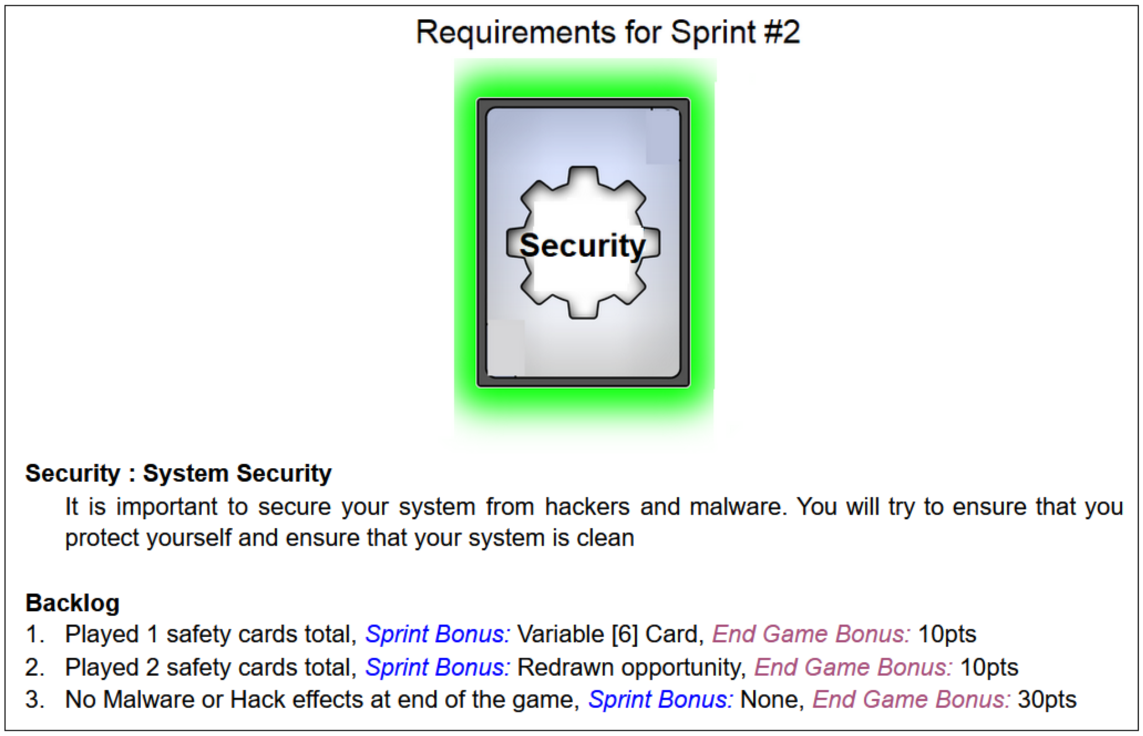
if the player can create a method stack with two `Repeat` cards, then they receive a `Computer Scan` card as a bonus.

6.2.1.2 Sprint 2 - "Security"

In the 'Security' sprint, the requirement for the players is to use the cyberdefense cards so that the system can be secured from various cyberattacks. This teaches the SE design principle of 'System Security'.

Figure 6.6 shows the requirement card for this phase, which explains the overall objective of the sprint. The main objective, game plan and bonus for `Security` are:

- Objective: The main objective of this sprint is to encourage a player to keep their system safe from different hacks and malware attacks.
- Game plan: For achieving the goals of Sprint 2, the player has to use various cyber defence cards.


Requirements for Sprint #2

Security

Security : System Security

It is important to secure your system from hackers and malware. You will try to ensure that you protect yourself and ensure that your system is clean

Backlog
1. Played 1 safety cards total, *Sprint Bonus:* Variable [6] Card, *End Game Bonus:* 10pts
2. Played 2 safety cards total, *Sprint Bonus:* Redrawn opportunity, *End Game Bonus:* 10pts
3. No Malware or Hack effects at end of the game, *Sprint Bonus:* None, *End Game Bonus:* 30pts

Figure 6.6: The Requirements Card of Sprint 2(Security)

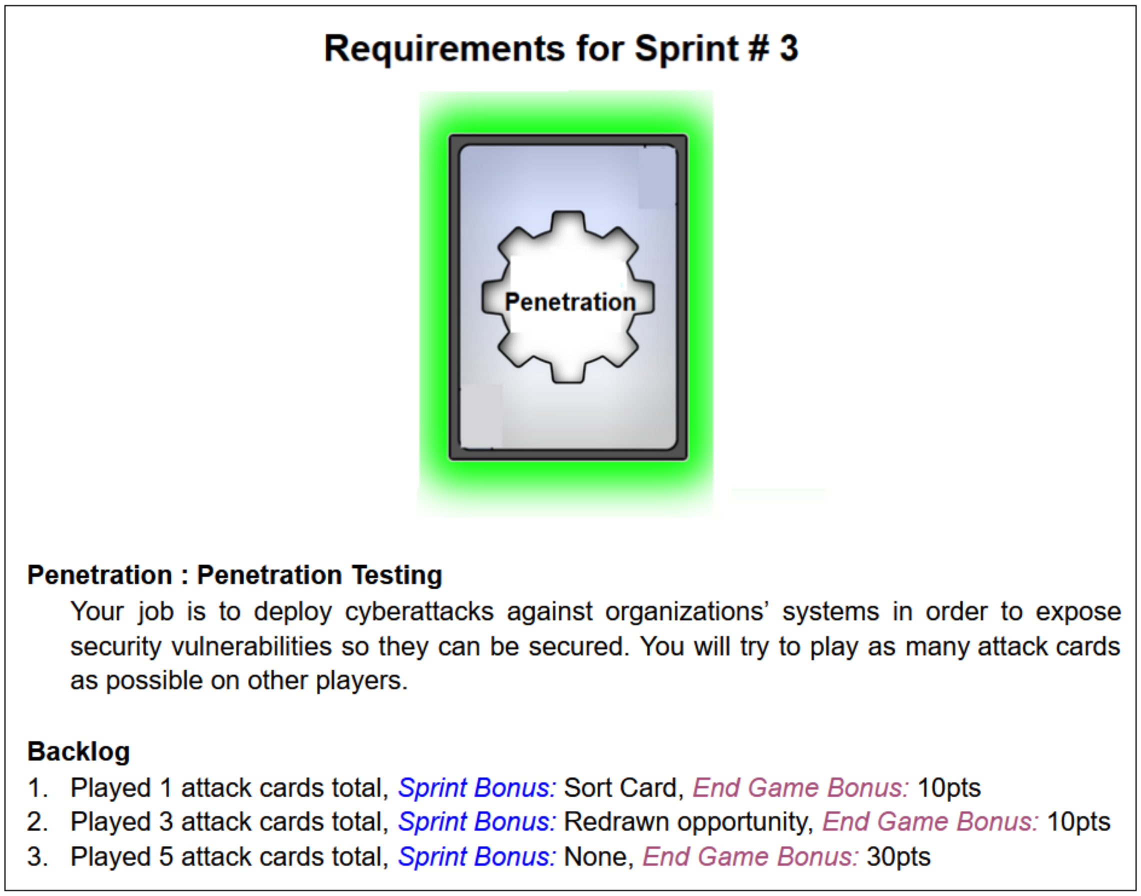
- Bonus: There are several bonus criteria in this sprint. If player could play 1 `Cyberdefense` cards they will be awarded with `Variable` [6] card[29] and bonus 10 points. If a player could play 2 safety-related cards (i.e. Computer Scan, Firewall and Antivirus cards), they will have the chance to redraw[30] along with 10 bonus points. At the end of the sprint, if they are immune (a.k.a. played both Firewall and Antivirus cards) from Malware or Hacking, they receive 30 bonus points.

6.2.1.3 Sprint 3 - "Penetration"

In the 'Penetration' sprint, the requirement for the players is to use the cyberattack cards so that the opponent's system can be tested against various cyberattacks. This teaches the testing practice of 'Penetration Testing'.

Figure 6.7 shows the requirement card for this phase, which explains the overall objective of the sprint. The main objective, game plan and bonus for `Penetration` are:

[29] Player can use this in the *Implementation* phase
[30] Player get the chance to discard and redraw cards from their hand in the *Implementation* phase


Penetration

Requirements for Sprint # 3

Penetration : Penetration Testing

Your job is to deploy cyberattacks against organizations' systems in order to expose security vulnerabilities so they can be secured. You will try to play as many attack cards as possible on other players.

Backlog

1. Played 1 attack cards total, *Sprint Bonus:* Sort Card, *End Game Bonus:* 10pts
2. Played 3 attack cards total, *Sprint Bonus:* Redrawn opportunity, *End Game Bonus:* 10pts
3. Played 5 attack cards total, *Sprint Bonus:* None, *End Game Bonus:* 30pts

Figure 6.7: The Requirements Card of Sprint 3(Penetration)

- Objective: The main objective of this sprint is to encourage a player to play different cyberattack cards so that they can exploit the vulnerability of the opponent's system.
- Game plan: For achieving the goals of sprint 3, a player has to use various cyberattack cards.
- Bonus: There are several bonus criteria in this sprint. If the player could play 1 attack card, they will be awarded `Sort` card[29] and bonus 10 points. If a player could play 3 cyberattack cards, they will have the chance to redraw[30] along with 10 bonus points. At the end of the sprint, if they were able to make 5 cyberattacks, they receive 30 bonus points.

6.2.2 Design Phase

After a player receives the requirements for each sprint, they move to the next phase of the game, where a player develops a customized deck based on the sprint requirements. Since each of the sprints has different objectives, it is necessary for players to make some choices about how they can best complete these objectives. The *Design* phase represents the part of a software project where developers make decisions about how the software will be built and what tools they should use. In the *Design* phase, players get a base set of cards for their deck. These cards are those that are essential for playing Program Wars, such as `Instruction` and `Repeat` cards.

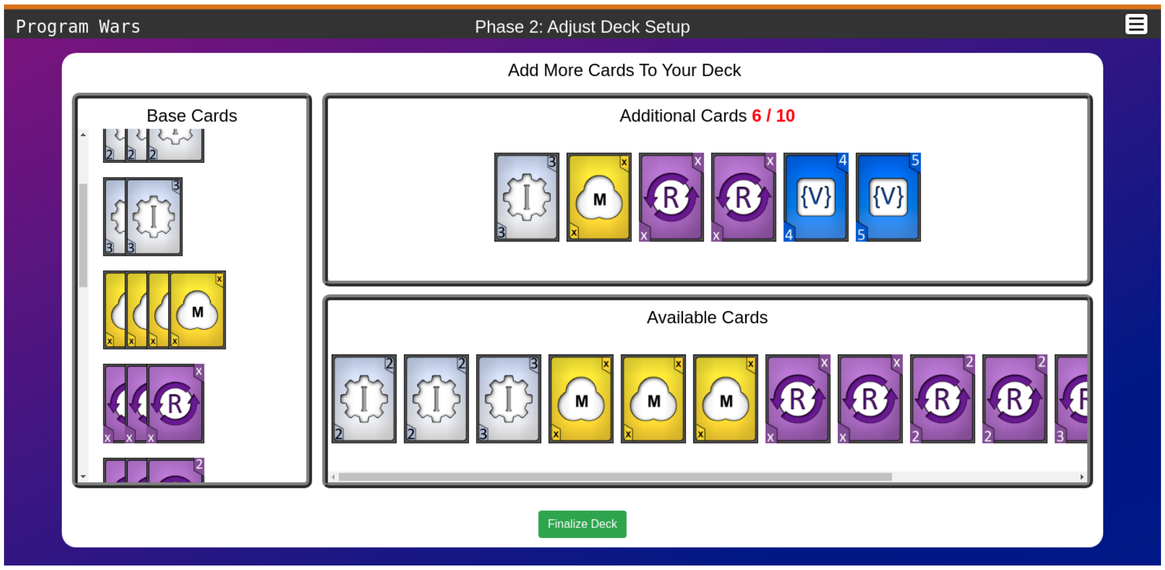


Figure 6.8: The Design Phase Page

In this phase, players are given a pool of cards, from where they have to choose a number of cards to meet the requirements. The card pool is unique to each of the requirements so that players can have access to certain cards that are most useful for that requirement. However, there are also some cards that may not be essential, but a player can still have those cards as a part of their strategy. More powerful cards are limited in number and do not appear in the card pools for all requirements.

When players start the *Design* phase, they are taken to a screen where they can build their own deck. On the left side of the screen (in Figure 6.8), there is a vertical list of card types. Each type has a pile showing how many cards of that type, and their value, if applicable, which are included in the deck automatically. This way, the player knows what cards are already in their deck and can act accordingly. The rest of the screen is split into two horizontal lists of cards. The cards are on the top of the screen are those they have selected, and on the bottom of the screen is the pool of cards the player can pick from. Cards can be dragged between these two lists to move them around. The upper list has an indicator of how many cards have been added and how many can be added in total. Once a player has added the maximum number of cards to this list, they are able to advance to the *Implementation* phase.

6.2.3 Implementation Phase

The *Implementation* phase, for the most part, is played the same as Program Wars *v.*2.0 . The major change is that the game no longer ends when a player reaches a specific point. Instead, the players each get 10 turns. The other major change to the gameplay is that a player is trying to complete the set of requirements for an ongoing sprint. In Program Wars *v.*2.0 , it was possible to ignore the bonus objectives and win as the player that reaches the point total first wins. In Program Wars *v.*3.0 , the player needs to complete at least two (2) of their requirements in order to win the game. This allows beginner players an opportunity to get used to the format of the game without excelling immediately. The sprint bonuses are useful in making it easier to get more points or to complete subsequent objectives before the end of the game.

6.2.4 Testing Phase

The *Testing* phase is a replacement for the score modal that is displayed at the end of the Program Wars *v.*2.0 game. This phase represents the sprint acceptance testing phase for the program the player built. The player has requirements to fulfill for the sprint, so they are judged on their progress. This is where the bonus points for requirements are added to a player's current score. The detailed progress for each player toward their requirements is compared in this phase. The player with the most points after three sprints will be declared the winner. Tiebreaks will favour the player that completed more requirements in total or by the end of the appropriate sprint. The points given for requirements are intended to be balanced to allow strategies that do not focus as much on the total instruction score.

6.3 Summary

This chapter has provided a possible answer to **R.Q. # 3**. In Program Wars *v.*3.0 , the concepts of the SDLC and ISDM are integrated into the modified version of Program Wars *v.*2.0 . By introducing several sprints, Program Wars *v.*3.0 allows the player to experience the SDLC and ISDM. This knowledge is for those who seek to become software developers. Also, by introducing different phases of SDLC, players are habituated with different phases in Software Development which is really important in the real world. For future research work, a user study can be proposed to measure the performance for Program Wars *v.*3.0 to teach the SDLC and ISDM.

Chapter 7

Conclusion

Various activities, such as gaming that integrates both social and educational engagement, have been found to be an effective way of learning. Among these, Game-Based Learning (GBL) is one of the popular methods for teaching conceptual knowledge in a practical way. Learning computer programming can be monotonous. Moreover, a novice who wants to learn programming may find the learning procedure really hard as most of the computer programming tools use specific syntax. For a beginner, it is really hard to memorize all the appropriate syntax of a required compiler. As a result, a certain amount of students may find computer programming hard to learn.

Again, it is observed that in the software industry, a successful project largely depends on a developer's software engineering (SE) education. A person's knowledge about the software development process often depends on his/her adequate knowledge of computer programming.

This research work aims to explore further the effectiveness of teaching fundamental software engineering concepts through GBL. Specifically, Program Wars, a web-based card game for learning programming language and cybersecurity concepts, was extended into two versions.

Program Wars *v.*2.0 presented an evolution of the game elements and concepts introduced by Program Wars *v.*1.0 . Regarding programming concepts, the `Group` card was changed to the `Method` card to better represent procedures/functions/methods in programs. Also, the conditional statement concept was made more explicit with the use of gameplay goals. Similarly, the general cybersecurity concepts in Program Wars *v.*1.0 were specialized in Program Wars *v.*2.0 . Finally, Program Wars *v.*2.0 introduced the concepts of algorithms and library function concepts through the addition of the `Search` and `Sort` cards. Regarding gameplay, two modes of play (*Beginner* and *Standard*) were introduced to provide a learning path for players, and the user interface was revised to provide more player and game information. To evaluate Program Wars *v.*2.0, a user study was conducted. After analyzing the data from the user study, it was found that in terms of programming knowledge, participant's understanding of variables improved. For the loop (repeat) structure, more than fifty percent of participants showed knowledge improvement after playing Program Wars *v.*2.0 , and for the method structure, more than forty percent of participants showed their knowledge was improved. In cybersecurity concepts, more than sixty percent of participants showed knowledge gains regarding various cyberattacks and cyberdefense.

Regarding Program Wars *v.*3.0 , both the concept of ISDM and SDLC are integrated to provide knowledge about software development. The game is comprised of several sprints, which allow the users to refine their knowledge of several programming concepts. Also, the game has different phases of the SDLC, which match with the different phases in software development. It is believed that the gameplay of Program Wars *v.*3.0 helps a user to understand the workflow of a real-world software development project. To evaluate the performance of Program Wars *v.*3.0 , a user study is also proposed in the future work section.

7.1 Future Work

A few ideas and directions regarding future work are presented in this section.

7.1.1 Gamplay and Card behaviour

In Program Wars, the `Variable` card cannot be played independently of the `Repeat-X` card. In future, this card should be examined to see if it can be made more independent and demonstrate other uses of variables in programming.

The notion of a semantic error is not integrated into the current version of the game. In the future, integrating this type of error into the game could make the game more challenging.

Control flow is only introduced using the bonus objectives. Consideration should be given into integrating control flow more directly into the game play.

7.1.2 Program Wars *v.*3.0 User Study

Similar to Program Wars *v.*2.0 , to assess whether Program Wars *v.*3.0 improved participant's knowledge about the SDLC and ISDM, a separate user study can be performed. There will be three stages in the proposed user study,

- **Pre-game (Stage # 1)** – In the pre-game questionnaire, the participant will be asked some demographic questions. Participants will also be asked about their previous experience related to software development. After that, some multiple-choice questions will be asked related to the SDLC and ISDM concepts to assess the participant's previous knowledge about the subjects.
- **Play Program Wars *v.*3.0 (Stage # 2)** – After completing the pre-game questionnaire, the participant will be asked to play Program Wars *v.*3.0 at least five (5) times.
- **Post-game (Stage # 3)** – In this stage, participants will be asked to complete a post-game questionnaire. They will be asked some knowledge questions similar to those from the pre-game questionnaire related to the SDLC and ISDM to identify the knowledge gains. Also, participants will be asked questions to assess if they could co-relate the game mechanics to the real-life software development process.

7.1.2.1 Participants

To participate in the Program Wars *v.*3.0 user study, participants must have an understanding of the basic programming concepts. Also, Someone who is interested in software development could also be part of the user study. University undergraduates who have experience with basic computer programming-related courses will also be a part of this user study. For recruiting participants, different communication will be made, like posters (both on-campus and digital), email or invitations in social media.

7.1.2.2 Questions Type

Throughout the user study, there will be several questions asked in both the pre-game and post-game questionnaires. An overview of the questions is given below.

- **Demographic Questions:** These will be some basic demographic questions (i.e. age, gender).
- **Previous Experience Questions:** The participant will be asked questions related to their experience with software development and programming experience.
- **Knowledge questions:** The participant will be asked questions related to the SDLC and ISDM in both pre-game and post-game questionnaires.
- **Mapping questions:** In the post-game questionnaire, participants will be asked some questions to identify if they could map the knowledge of the game with real-world software development processes.
- **Feedback questions:** Participants will be asked to provide feedback regarding Program Wars *v.*3.0 .

7.1.2.3 Privacy

Anonymity will be partially provided because the participant's background (i.e. software development skills) is vital regarding the feedback they provide.

Consents: Before completing the pre-game questionnaire, participants will be presented with an informed consent web form.

Bibliography

[1] M. S. L. A. S.-R. W. Jacqueline E. Maloney, *A Mindfulness-Based Social and Emotional Learning Curriculum for School-Aged Children: The MindUP Program*. New York, NY: Springer, 2016.

[2] J. P. Gee, *What Video Games Have to Teach Us About Learning and Literacy*. Palgrave Macmillan, 2004.

[3] J. McGonigal, "Gaming can make a better world," 2010.

[4] N. Whitton, "Motivation and computer game based learning," 01 2007.

[5] S. Deterding, R. Khaled, L. Nacke, and D. Dixon, "Gamification: Toward a definition," *Proceedings of CHI 2011 Workshop Gamification: Using Game Design Elements in Non-Game Contexts*, pp. 6–9, 2011.

[6] V. Uskov and B. Sekar, "Gamification of software engineering curriculum," *2014 IEEE Frontiers in Education Conference (FIE) Proceedings, Madrid*, pp. 1–8, 2014.

[7] C. Pettey, "Gartner predicts over 70 percent of global 2000 organisations will have at least one gamified application by 2014." http://www.gartner.com/newsroom/id/1844115, 2011. [Online; accessed 05-July-2021].

[8] A. Calderón, M. Ruiz, and E. Orta, "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)*, pp. 21–27, 2017.

[9] A. Baker, E. Oh Navarro, and A. van der Hoek, "An experimental card game for teaching software engineering processes," *Journal of Systems and Software*, vol. 75, no. 1, pp. 3–16, 2005. Software Engineering Education and Training.

[10] M. R. De Almeida Souza, L. Furtini Veado, R. Teles Moreira, E. Magno Lages Figueiredo, and H. A. X. Costa, "Games for learning: bridging game-related education methods to software engineering knowledge areas," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, pp. 170–179, May 2017.

[11] J. Pieper, O. Lueth, M. Goedicke, and P. Forbrig, "A case study of software engineering methods education supported by digital game-based learning: Applying the semat essence kernel in games and course projects," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1689–1699, April 2017.

[12] L. H. B. Jacobson, Ivar, P.-W. Ng, P. E. McMahon, and M. Goedicke, *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons*. Association for Computing Machinery and Morgan and Claypool, 2019.

[13] N. T. T. Xie and J. de Halleux, "Educational software engineering: Where software engineering, education, and gaming meet," *2013 3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change (GAS), San Francisco, CA*, pp. 36–39, 2013.

[14] J. B. Swapneel Sheth and G. Kaiser, "Halo (highly addictive socially optimized) software engineering," *In Proceedings of the 1st International Workshop on Games and Software Engineering (GAS '11). Association for Computing Machinery, New York, NY, USA*, pp. 29–32, 2011.

[15] M. A. Miljanovic and J. S. Bradbury, "Robobug: A serious game for learning debugging techniques," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, (New York, NY, USA), p. 93–100, Association for Computing Machinery, 2017.

[16] C. Szabo, "Evaluating gamedevtycoon for teaching software engineering," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, (New York, NY, USA), p. 403–408, Association for Computing Machinery, 2014.

[17] D. Pickton and S. Wright, "What's swot in strategic analysis?," *Strategic Change*, vol. 7, pp. 101–109, 03 1998.

[18] Y. Khazaal, A. Chatton, R. Prezzemolo, F. Zebouni, Y. Edel, J. Jacquet, O. Ruggeri, E. Burnens, G. Monney, A.-S. Protti, J.-F. Etter, R. Khan, J. Cornuz, and D. Zullino, "Impact of a board-game approach on current smokers: A randomized controlled trial," *Substance abuse treatment, prevention, and policy*, vol. 8, p. 3, 01 2013.

[19] "Battle bots v2." https://www.curufea.com/doku.php?id=games:board:battlebots. [Online; accessed 05-July-2021].

[20] I. J. Timm, T. Bogon, A. D. Lattner, and R. Schumann, "Teaching distributed artificial intelligence with roborally," in *Multiagent System Technologies* (R. Bergmann, G. Lindemann, S. Kirn, and M. Pěchouček, eds.), (Berlin, Heidelberg), pp. 171–182, Springer Berlin Heidelberg, 2008.

[21] "Robot turtles." http://www.robotturtles.com. [Online; accessed 05-July-2021].

[22] "Code master." https://www.thinkfun.com/products/code-master. [Online; accessed 05-July-2021].

[23] P. Chen, R. Kuo, M. Chang, and J.-S. Heh, "Designing a trading card game as educational reward system to improve students' learning motivations," *T. Edutainment*, vol. 3, pp. 116–128, 08 2009.

[24] "Potato pirates." https://potatopirates.game/. [Online; accessed 05-July-2021].

[25] R. Wetzel, L. Blum, and L. Oppermann, "Tidy city: A location-based game supported by in-situ and web-based authoring tools to enable user-created content," in *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, (New York, NY, USA), p. 238–241, Association for Computing Machinery, 2012.

[26] "Codecombat." https://codecombat.com. [Online; accessed 05-July-2021].

[27] "Blockly maze." https://www.brainpop.com/games/blocklymaze. [Online; accessed 05-July-2021].

[28] "Kodable." https://www.kodable.com/. [Online; accessed 05-July-2021].

[29] "Codingame." https://www.codingame.com/start. [Online; accessed 05-July-2021].

[30] "Cyber threat defender." https://cias.utsa.edu/ctd.php. [Online; accessed 17-August-2021].

[31] "Potato pirates 2: Enter the spudnet." https://potatopirates.game/products/enter-the-spudnet-board-game. [Online; accessed 17-August-2021].

[32] "Scrum card game." https://www.tastycupcakes.org/2016/06/scrum-card-game/. [Online; accessed 25-June-2021].

[33] J. M. Fernandes and S. M. Sousa, "Playscrum - a card game to learn the scrum agile method," in *2010 Second International Conference on Games and Virtual Worlds for Serious Applications*, pp. 52–59, 2010.

[34] A. Baker, E. Navarro, and A. van der Hoek, "Problems and programmers: An educational software engineering card game," pp. 614– 619, 06 2003.

[35] J. M. Randel, B. A. Morris, C. D. Wetzel, and B. V. Whitehill, "The effectiveness of games for educational purposes: A review of recent research," *Simulation & Gaming*, vol. 23, no. 3, pp. 261–276, 1992.

[36] P. Moreno-Ger, D. Burgos, I. Martínez-Ortiz, J. L. Sierra, and B. Fernández-Manjón, "Educational game design for online education," *Computers in Human Behavior*, vol. 24, no. 6, pp. 2530 – 2540, 2008. Including the Special Issue: Electronic Games and Personalized eLearning Processes.

[37] J. Anvik, V. Cote, and J. Riehl, "Program wars: A card game for learning programming and cybersecurity concepts," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, (New York, NY, USA), p. 393–399, Association for Computing Machinery, 2019.

[38] *Refactoring: Improving the Design of Existing Code*. USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[39] “Program wars.” https://program-wars.firebaseapp.com/. [Online; accessed 05-July-2021].

[40] T. Nooning, “Lock it down: Use libsafe to secure linux from buffer overflows.” https://www.techrepublic.com/article/lock-it-down-use-libsafe-to-secure-linux-from-buffer-overflows/, 2002. [Online; accessed 13-August-2020].

[41] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, “Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks,” in *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM’98, (USA), p. 5, USENIX Association, 1998.

[42] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, “Automatic creation of sql injection and cross-site scripting attacks,” in *2009 IEEE 31st International Conference on Software Engineering*, pp. 199–209, 2009.

[43] S. Paira, S. Chandra, and S. S. Alam, “Enhanced merge sort- a new approach to the merging process,” *Procedia Computer Science*, vol. 93, pp. 982–987, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[44] I. C. S. . A. f. C. M. Joint Task Force on Computing Curricula, “Software engineering 2014: Curriculum guidelines for undergraduate degree programs in software engineering.” https://www.acm.org/binaries/content/assets/education/se2014.pdf, 2015. [Online; accessed 16-Oct-2020].

[45] G. Rodriguez, Soria, and M. Campo, “Virtual scrum: A teaching aid to introduce undergraduate software engineering students to scrum,” *Computer Applications in Engineering Education*, vol. 23, no. 1, pp. 147–156, 2015.

[46] M. A. Boschetti, M. Golfarelli, S. Rizzi, and E. Turricchia, “A lagrangian heuristic for sprint planning in agile software development,” *Computers Operations Research*, vol. 43, pp. 116–128, 2014.

Appendix A

User Study Consent Form

Please read the following information carefully before beginning the survey.

Program Wars v.2.0: A Game-based Approach to Teaching Programming and Cybersecurity Concepts.

What is this study about?
You have been invited to participate in a research study at the University of Lethbridge. Through your participation in this research study, we hope to understand the better effect that a web-based card game has on learning fundamental programming and cybersecurity concepts. No programming or Computer Science background is required to participate in the study.

In the study, we are testing whether a participant's knowledge of programming and cybersecurity concepts increases after playing a few rounds of the game, as well as finding ways to improve the educational nature of the game.

What are the benefits of participating?
There are no direct benefits associated with participating in this study. You may, however, gain a better understanding of programming and cybersecurity concepts.

What are the risks and benefits of participating?
There are no anticipated risks from participating in this study.

What is expected of you?
Your participation in the research study is expected to take 60-90 minutes, depending on how many times you play the game. Participating in this study involves three parts:

1. Completing an initial survey containing demographics questions. You will also be given some examples of computer programs and asked to answer some questions. This portion is expected to take ten to fifteen (10-15) minutes.
2. After the pre-game survey has been completed, you will be asked to play Program Wars v 2.0. It's recommended that participants should play the game minimum of three (3) and a maximum of five (5) times. The average time required to complete a level is approximately ten (10) minutes. The outcome of the games (i.e., if you win or lose) is not important to the study and is not recorded. This portion is expected to take about 30-60 minutes, depending on how many times you play the game.
3. After you have completed playing some games, you will complete a post-game survey. The post-game survey will ask similar questions as were asked in the pre-game survey. You will also be asked about your experience playing the game. This portion is expected to take ten to fifteen (10-15) minutes.

You will be asked to provide a legitimate e-mail address for both the pre-and post-game survey (e-mail addresses should be the same for both pre-and post-game surveys). You will be given an option to opt-in to receive the study results once the analysis is completed. If you choose not to opt-in, you will never receive an e-mail from us. Your e-mail address will be deleted from the collected data within 12 months of taking the survey.

What are the anticipated uses of the data collected?
The responses of the surveys will be aggregated and presented in one or more academic reports and presentations, including a Master's thesis. At no time will identifying information be used in the reports or presentations.

How will your confidentiality and anonymity be protected?
Participation is voluntary. However, as with any on-line survey, neither anonymity nor confidentiality can be completely guaranteed. The survey is being conducted using Qualtrics, and their privacy policy can be accessed at https://www.qualtrics.com/privacy- statement/.

You will be asked to provide a legitimate e-mail address for both the pre-and post-game survey (e-mail addresses should be the same for both pre-and post-game surveys). Your e-mail address will be replaced with a unique study_id for the pre-and post-game questionnaires so that your responses cannot be linked back to you. Individuals who complete the study will be given an option to receive a summary of the study results once the analysis is completed. If an individual chooses not to opt-in, he/she will never receive an e-mail from us. In any event, your e-mail address will be deleted from the collected data within 12 months of taking the survey.

How can a participant withdraw?
Your participation is completely voluntary. You can withdraw from the study at any time while taking the survey or playing the game if you decide you no longer want to participate. You may choose not to complete the questionnaires at any time by simply closing your browser before you submit your responses. If you do so, the responses you entered will be destroyed and not included in the study. Once the data has been extracted from Qualtrics, the pre-game and post-game questionnaires have been linked, and your e-mail address replaced by a numeric number, it will not be possible to withdraw your responses, as at this point, there is no longer a means to link responses to an individual person.

Who is conducting this research?
If you require any additional information about this study, please contact Md. Hasan Tareque at mdhasan.tareque@uleth.ca . Questions regarding your rights as a participant in this research may be addressed to the Office of Research Ethics, University of Lethbridge (Phone: 403-329-2747 or E-mail: research.services@uleth.ca).

This research project has been reviewed for ethical acceptability and approved by the University of Lethbridge Human Participant Research Committee. Thank you for your consideration.

If you wish to participate in the study, please check the "I agree" checkbox below and click "Submit". Submission of your responses will be accepted as implied consent to participate. Thank you in advance for your participation.

Are you 18 years or above? (You must be 18 years or older to participate in this study)

☐ Yes
☐ No

I have little or no programming concepts

☐ Yes
☐ No

Click the "Submit" button to proceed.

Appendix B

User Study Invitation Letter

Hello everyone, my name is Md. Hasan Tareque. I am a graduate student in Computer Science at the University of Lethbridge. I would like to invite you to participate in a research study. The study is called "Program Wars v.2.0: A Game-based Approach to Teaching Programming and Cybersecurity Concepts".

In the study, we are exploring whether your knowledge of programming and cybersecurity concepts increases after playing a few rounds of the game, as well as finding ways to improve the educational nature of the game.

No programming or Computer Science background is required to participate in the study.

Your participation in the study will require the use of a computer with internet access and approximately 60-90 mins of your time. The study is not designed to be done on a mobile device, so you will need to use laptops or desktop computers.

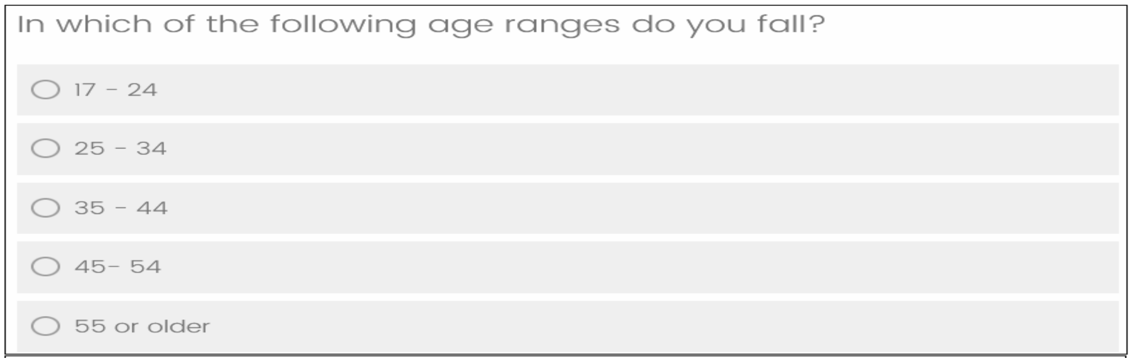
I hope participating in the study will enrich your knowledge regarding programming and cybersecurity concepts.

Link:
the URL to direct the participant to the consent form

If you require any additional information about this study, please contact Md. Hasan Tareque at mdhasan.tareque@uleth.ca

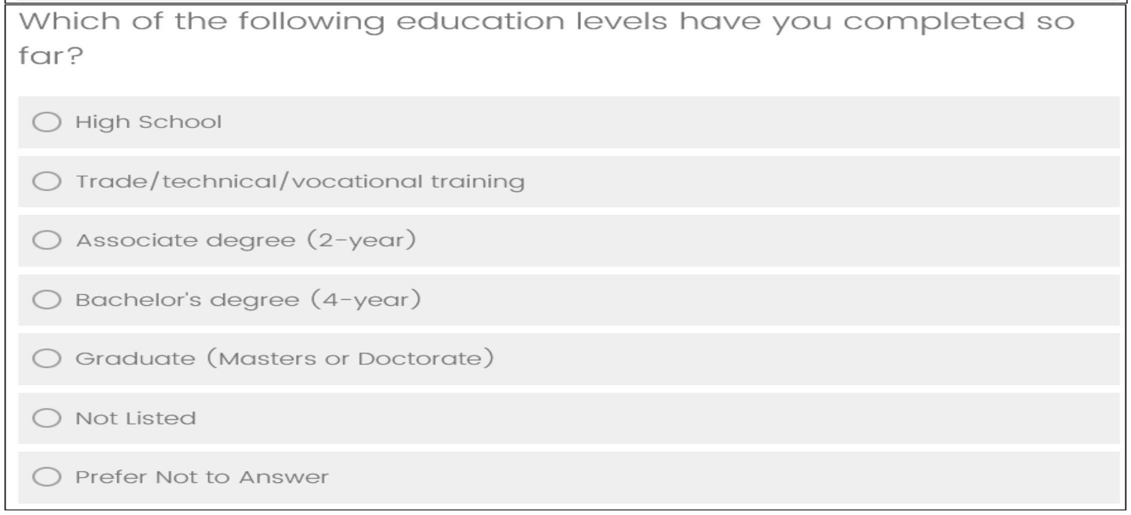
Appendix C

User Study Demographic Questions



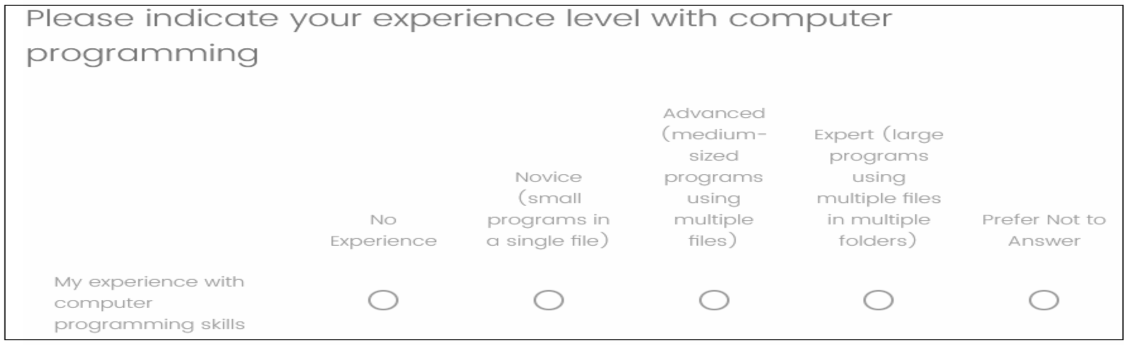
In which of the following age ranges do you fall?

- ○ 17 - 24
- ○ 25 - 34
- ○ 35 - 44
- ○ 45- 54
- ○ 55 or older



Which of the following education levels have you completed so far?

- ○ High School
- ○ Trade/technical/vocational training
- ○ Associate degree (2-year)
- ○ Bachelor's degree (4-year)
- ○ Graduate (Masters or Doctorate)
- ○ Not Listed
- ○ Prefer Not to Answer



Please indicate your experience level with computer programming

	No Experience	Novice (small programs in a single file)	Advanced (medium-sized programs using multiple files)	Expert (large programs using multiple files in multiple folders)	Prefer Not to Answer
My experience with computer programming skills	○	○	○	○	○

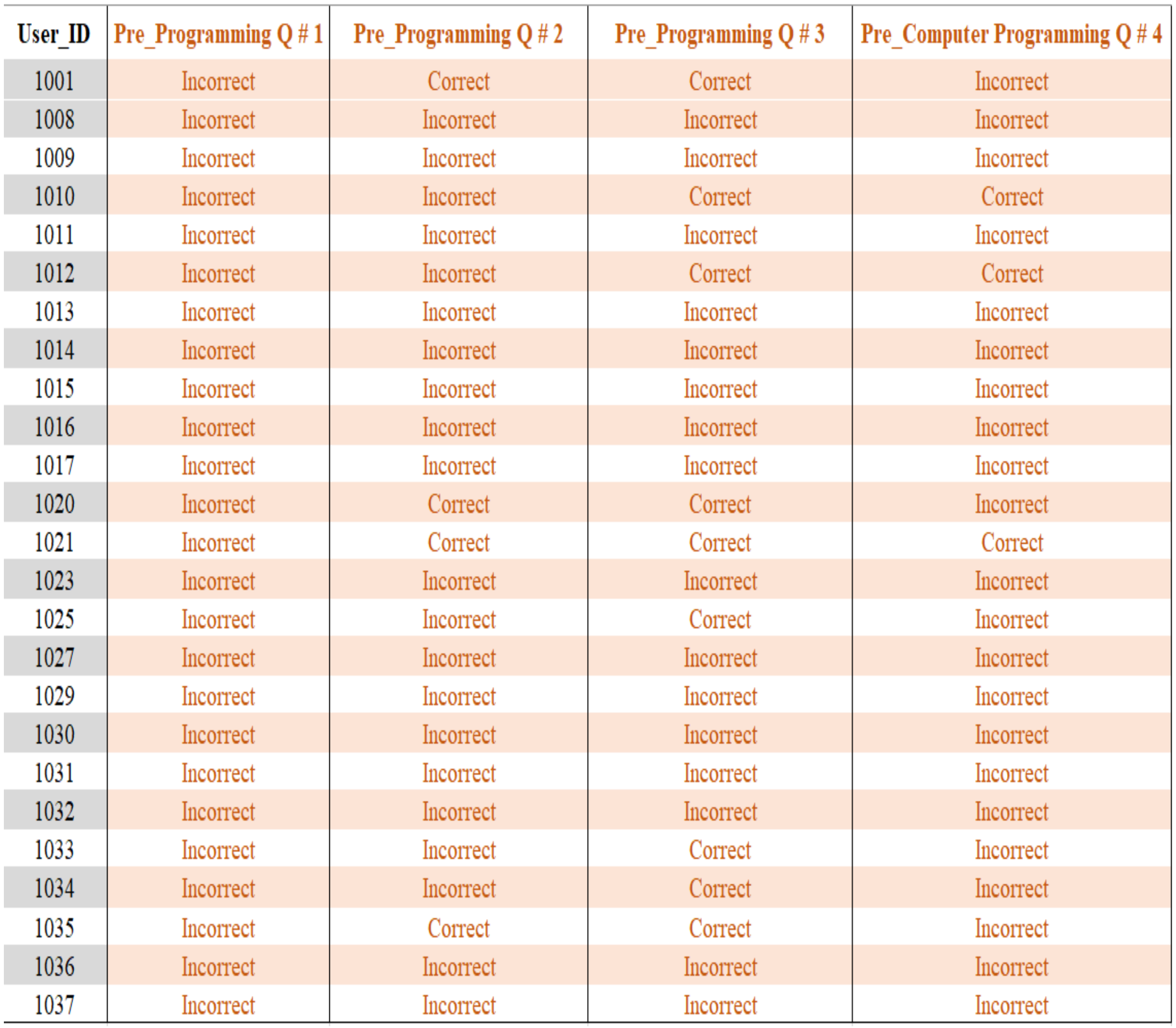
Appendix D

User Study Responses

Table D.1: Demographic responses from the participants

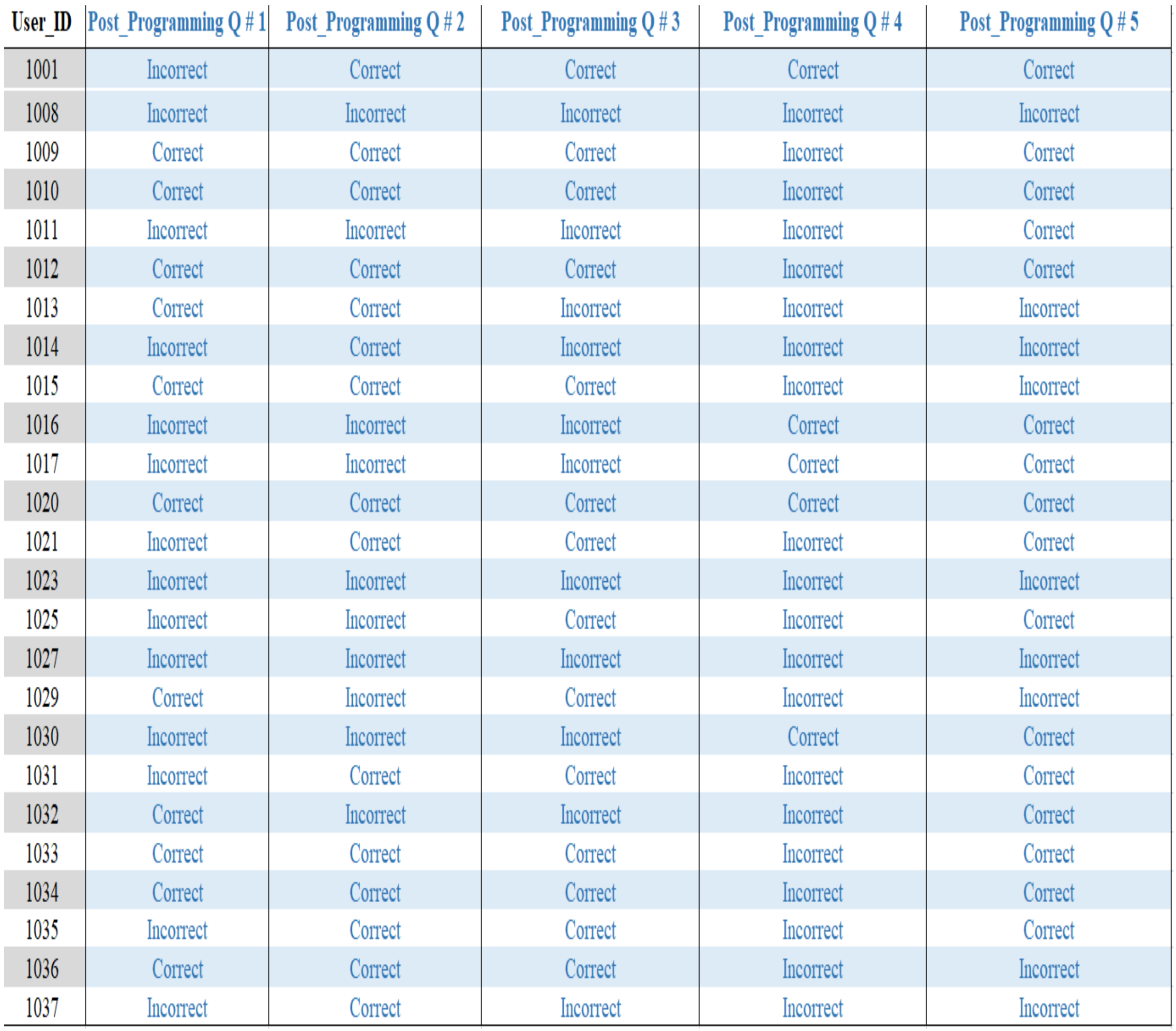
User_ID	Age Range	Education	Experience with computer programming	Experience with cybersecurity
1001	17 - 24	High School	Novice (small programs in a single file)	Novice (know a few concepts)
1006	17 - 24	High School	Advanced (medium-sized programs using multiple files)	Novice (know a few concepts)
1008	17 - 24	High School	No Experience	Novice (know a few concepts)
1009	17 - 24	High School	No Experience	No Experience
1010	17 - 24	High School	No Experience	No Experience
1011	17 - 24	High School	No Experience	No Experience
1012	17 - 24	High School	No Experience	No Experience
1013	17 - 24	High School	Novice (small programs in a single file)	Novice (know a few concepts)
1014	17 - 24	High School	No Experience	Novice (know a few concepts)
1015	17 - 24	High School	No Experience	No Experience
1016	17 - 24	High School	Novice (small programs in a single file)	No Experience
1017	17 - 24	High School	Novice (small programs in a single file)	Novice (know a few concepts)
1020	17 - 24	High School	Novice (small programs in a single file)	No Experience
1021	17 - 24	Bachelor's degree (4-year)	Novice (small programs in a single file)	Novice (know a few concepts)
1023	17 - 24	High School	No Experience	No Experience
1025	25 - 34	High School	No Experience	Novice (know a few concepts)
1027	17 - 24	High School	No Experience	Novice (know a few concepts)
1029	17 - 24	High School	Novice (small programs in a single file)	No Experience
1030	17 - 24	High School	No Experience	No Experience
1031	17 - 24	High School	No Experience	No Experience
1032	17 - 24	Graduate (Masters or Doctorate)	No Experience	Novice (know a few concepts)
1033	17 - 24	Bachelor's degree (4-year)	Novice (small programs in a single file)	No Experience
1034	17 - 24	High School	No Experience	Novice (know a few concepts)
1035	25 - 34	High School	No Experience	No Experience
1036	25 - 34	High School	No Experience	No Experience
1037	17 - 24	Associate degree (2-year)	No Experience	Novice (know a few concepts)

Table D.2: Pre-game computer programming related responses



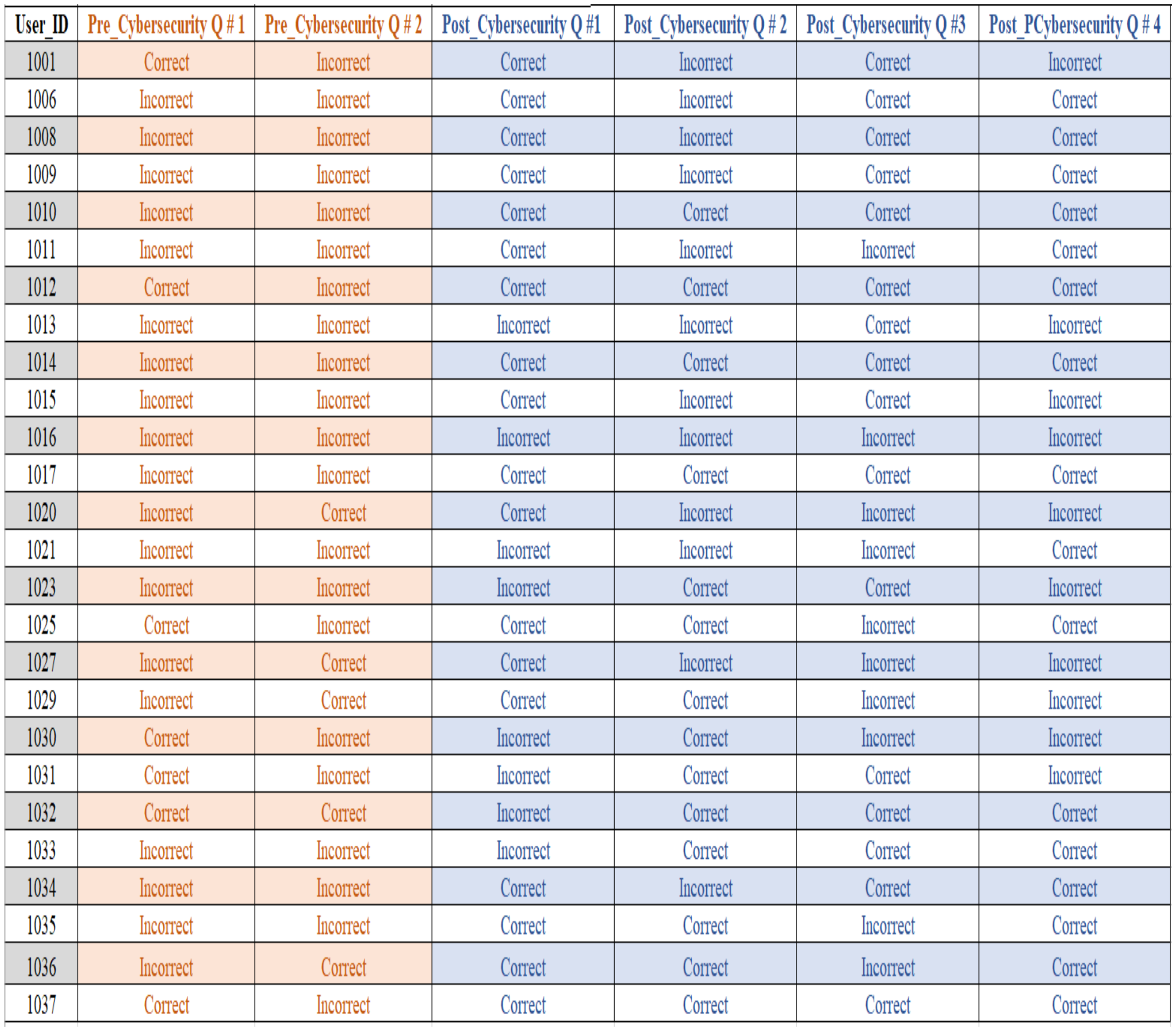
User_ID	Pre_Programming Q # 1	Pre_Programming Q # 2	Pre_Programming Q # 3	Pre_Computer Programming Q # 4
1001	Incorrect	Correct	Correct	Incorrect
1008	Incorrect	Incorrect	Incorrect	Incorrect
1009	Incorrect	Incorrect	Incorrect	Incorrect
1010	Incorrect	Incorrect	Correct	Correct
1011	Incorrect	Incorrect	Incorrect	Incorrect
1012	Incorrect	Incorrect	Correct	Correct
1013	Incorrect	Incorrect	Incorrect	Incorrect
1014	Incorrect	Incorrect	Incorrect	Incorrect
1015	Incorrect	Incorrect	Incorrect	Incorrect
1016	Incorrect	Incorrect	Incorrect	Incorrect
1017	Incorrect	Incorrect	Incorrect	Incorrect
1020	Incorrect	Correct	Correct	Incorrect
1021	Incorrect	Correct	Correct	Correct
1023	Incorrect	Incorrect	Incorrect	Incorrect
1025	Incorrect	Incorrect	Correct	Incorrect
1027	Incorrect	Incorrect	Incorrect	Incorrect
1029	Incorrect	Incorrect	Incorrect	Incorrect
1030	Incorrect	Incorrect	Incorrect	Incorrect
1031	Incorrect	Incorrect	Incorrect	Incorrect
1032	Incorrect	Incorrect	Incorrect	Incorrect
1033	Incorrect	Incorrect	Correct	Incorrect
1034	Incorrect	Incorrect	Correct	Incorrect
1035	Incorrect	Correct	Correct	Incorrect
1036	Incorrect	Incorrect	Incorrect	Incorrect
1037	Incorrect	Incorrect	Incorrect	Incorrect

Table D.3: Post-game computer programming related responses



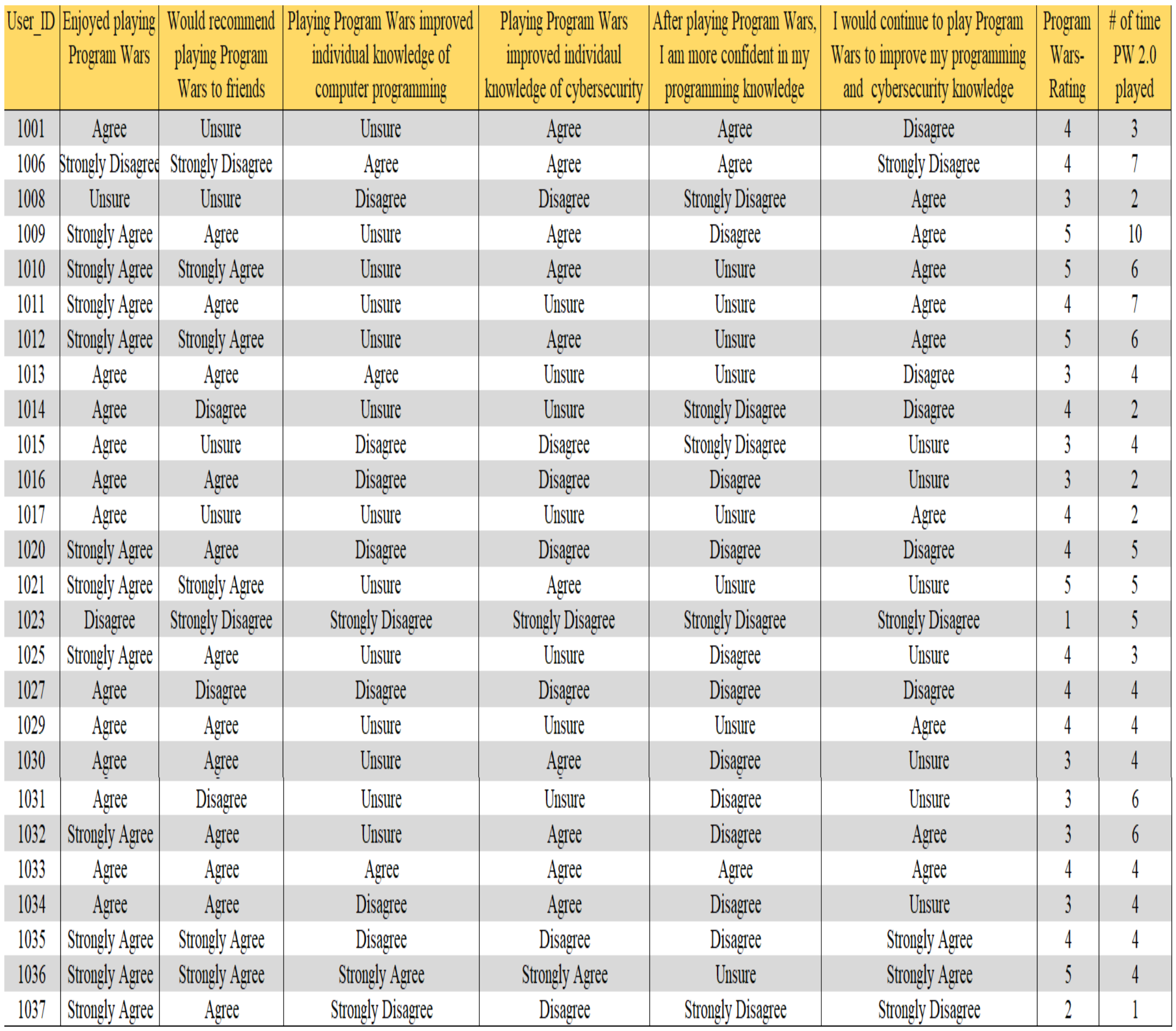
User_ID	Post_Programming Q # 1	Post_Programming Q # 2	Post_Programming Q # 3	Post_Programming Q # 4	Post_Programming Q # 5
1001	Incorrect	Correct	Correct	Correct	Correct
1008	Incorrect	Incorrect	Incorrect	Incorrect	Incorrect
1009	Correct	Correct	Correct	Incorrect	Correct
1010	Correct	Correct	Correct	Incorrect	Correct
1011	Incorrect	Incorrect	Incorrect	Incorrect	Correct
1012	Correct	Correct	Correct	Incorrect	Correct
1013	Correct	Correct	Incorrect	Incorrect	Incorrect
1014	Incorrect	Correct	Incorrect	Incorrect	Incorrect
1015	Correct	Correct	Correct	Incorrect	Incorrect
1016	Incorrect	Incorrect	Incorrect	Correct	Correct
1017	Incorrect	Incorrect	Incorrect	Correct	Correct
1020	Correct	Correct	Correct	Correct	Correct
1021	Incorrect	Correct	Correct	Incorrect	Correct
1023	Incorrect	Incorrect	Incorrect	Incorrect	Incorrect
1025	Incorrect	Incorrect	Correct	Incorrect	Correct
1027	Incorrect	Incorrect	Incorrect	Incorrect	Incorrect
1029	Correct	Incorrect	Correct	Incorrect	Incorrect
1030	Incorrect	Incorrect	Incorrect	Correct	Correct
1031	Incorrect	Correct	Correct	Incorrect	Correct
1032	Correct	Incorrect	Incorrect	Incorrect	Correct
1033	Correct	Correct	Correct	Incorrect	Correct
1034	Correct	Correct	Correct	Incorrect	Correct
1035	Incorrect	Correct	Correct	Incorrect	Correct
1036	Correct	Correct	Correct	Incorrect	Incorrect
1037	Incorrect	Correct	Incorrect	Incorrect	Incorrect

Table D.4: Pre and Post-game cybersecurity related responses



User_ID	Pre_Cybersecurity Q # 1	Pre_Cybersecurity Q # 2	Post_Cybersecurity Q #1	Post_Cybersecurity Q # 2	Post_Cybersecurity Q #3	Post_PCybersecurity Q # 4
1001	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
1006	Incorrect	Incorrect	Correct	Incorrect	Correct	Correct
1008	Incorrect	Incorrect	Correct	Incorrect	Correct	Correct
1009	Incorrect	Incorrect	Correct	Incorrect	Correct	Correct
1010	Incorrect	Incorrect	Correct	Correct	Correct	Correct
1011	Incorrect	Incorrect	Correct	Incorrect	Incorrect	Correct
1012	Correct	Incorrect	Correct	Correct	Correct	Correct
1013	Incorrect	Incorrect	Incorrect	Incorrect	Correct	Incorrect
1014	Incorrect	Incorrect	Correct	Correct	Correct	Correct
1015	Incorrect	Incorrect	Correct	Incorrect	Correct	Incorrect
1016	Incorrect	Incorrect	Incorrect	Incorrect	Incorrect	Incorrect
1017	Incorrect	Incorrect	Correct	Correct	Correct	Correct
1020	Incorrect	Correct	Correct	Incorrect	Incorrect	Incorrect
1021	Incorrect	Incorrect	Incorrect	Incorrect	Incorrect	Correct
1023	Incorrect	Incorrect	Incorrect	Correct	Correct	Incorrect
1025	Correct	Incorrect	Correct	Correct	Incorrect	Correct
1027	Incorrect	Correct	Correct	Incorrect	Incorrect	Incorrect
1029	Incorrect	Correct	Correct	Correct	Incorrect	Incorrect
1030	Correct	Incorrect	Incorrect	Correct	Incorrect	Incorrect
1031	Correct	Incorrect	Incorrect	Correct	Correct	Incorrect
1032	Correct	Correct	Incorrect	Correct	Correct	Correct
1033	Incorrect	Incorrect	Incorrect	Correct	Correct	Correct
1034	Incorrect	Incorrect	Correct	Incorrect	Correct	Correct
1035	Incorrect	Incorrect	Correct	Correct	Incorrect	Correct
1036	Incorrect	Correct	Correct	Correct	Incorrect	Correct
1037	Correct	Incorrect	Correct	Correct	Correct	Correct

Table D.5: Post Game Feedback



User_ID	Enjoyed playing Program Wars	Would recommend playing Program Wars to friends	Playing Program Wars improved individual knowledge of computer programming	Playing Program Wars improved individaul knowledge of cybersecurity	After playing Program Wars, I am more confident in my programming knowledge	I would continue to play Program Wars to improve my programming and cybersecurity knowledge	Program Wars-Rating	# of time PW 2.0 played
1001	Agree	Unsure	Unsure	Agree	Agree	Disagree	4	3
1006	Strongly Disagree	Strongly Disagree	Agree	Agree	Agree	Strongly Disagree	4	7
1008	Unsure	Unsure	Disagree	Disagree	Strongly Disagree	Agree	3	2
1009	Strongly Agree	Agree	Unsure	Agree	Disagree	Agree	5	10
1010	Strongly Agree	Strongly Agree	Unsure	Agree	Unsure	Agree	5	6
1011	Strongly Agree	Agree	Unsure	Unsure	Unsure	Agree	4	7
1012	Strongly Agree	Strongly Agree	Unsure	Agree	Unsure	Agree	5	6
1013	Agree	Agree	Agree	Unsure	Unsure	Disagree	3	4
1014	Agree	Disagree	Unsure	Unsure	Strongly Disagree	Disagree	4	2
1015	Agree	Unsure	Disagree	Disagree	Strongly Disagree	Unsure	3	4
1016	Agree	Agree	Disagree	Disagree	Disagree	Unsure	3	2
1017	Agree	Unsure	Unsure	Unsure	Unsure	Agree	4	2
1020	Strongly Agree	Agree	Disagree	Disagree	Disagree	Disagree	4	5
1021	Strongly Agree	Strongly Agree	Unsure	Agree	Unsure	Unsure	5	5
1023	Disagree	Strongly Disagree	Strongly Disagree	Strongly Disagree	Strongly Disagree	Strongly Disagree	1	5
1025	Strongly Agree	Agree	Unsure	Unsure	Disagree	Unsure	4	3
1027	Agree	Disagree	Disagree	Disagree	Disagree	Disagree	4	4
1029	Agree	Agree	Unsure	Unsure	Unsure	Agree	4	4
1030	Agree	Agree	Unsure	Agree	Disagree	Unsure	3	4
1031	Agree	Disagree	Unsure	Unsure	Disagree	Unsure	3	6
1032	Strongly Agree	Agree	Unsure	Agree	Disagree	Agree	3	6
1033	Agree	Agree	Agree	Agree	Agree	Agree	4	4
1034	Agree	Agree	Disagree	Agree	Disagree	Unsure	3	4
1035	Strongly Agree	Strongly Agree	Disagree	Disagree	Disagree	Strongly Agree	4	4
1036	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree	Unsure	Strongly Agree	5	4
1037	Strongly Agree	Agree	Strongly Disagree	Disagree	Strongly Disagree	Strongly Disagree	2	1

Table D.6: Post Game Comments

User_ID	Comments
1001	Images in instructions do not all match game images
1009	Very good game
1011	Provide more of an explanation to what the cards do.
1017	I had started the survey once and when I clicked on a ”search” card, no selection came up and I had to refresh the page (which restarted my survey) in order to exit the selection window.
1020	Although it is a fun card game within itself, it is very easy to look past any coding aspect and see it as purely a game and not give coding a second thought. I’m not sure how to improve this besides by being more explicit without being more blunt about coding.
1023	Could be fun and the game has potential but, the in game UI is awkward and would be confusing for people who have never played a turn based card game. Also there was a couple times I would play one of the red cards and the bot would just not play its turn. Like I said, it has potential but by a game standards, not beginner friendly. Although most things with the word ”cyber security” are not very beginner friendly I guess.
1027	I feel the video should have a small list of cards and their meanings at the side.
1031	There was no rules to start the game and it was very confusing on what exactly each card does as well as just knowing what the purpose to win
1035	You only know what the cards are and how you are suppose to play when you go over to the rules. The link puts you straight into the game without explaining anything so I was very lost at first. The connection of the different viruses and what not need to be more distinctly obvious or else what is the point other than just playing a game. Thanks a ton, I had a lot of fun!
1036	A very good concept to teach programming & cyber security basics and ideas, The game is really enjoyable and It enhanced my knowledge regarding basic programming and cyber security.
1037	Program Wars 2.0 seemed like a random game to me and I did not learn much of anything. I believe videos, one on cyber security threats and how to spot them, and one on basic programming, would be a better educational tool. I appreciate you trying to educate us.