VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Game for Teaching and Learning Programming

Sanjeeta Singh

Supervisor(s): : Karsten Lundqvist, Simon McCallum

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

This project explores the designs of previous work that has been done for Game-Based learning and analyses the effects the design approaches have on the users. A design for the project is created based on the explored design approaches. A browser-based game is implemented focusing on teaching users programming concepts that are found in most traditional imperative languages such as Java or C. The project conducts user testing with University students to obtain a better understanding of the user interaction and the impact the game has. The final product from the project is spoken about and future work and suggestions are presented.

Contents

Chapter 1

Introduction

Computer programming involves problem-solving skills and is considered an essential skill for today's digital world [26]. Game-based learning has been considered an effective way of helping others to construct knowledge by playing games. By taking this approach learners are more engaged in their learning and have high learning motivation which they can use to apply their gained knowledge to real-life situations [26]. Sometimes it is hard to keep individuals interested or motivated when it comes to learning and teaching technology-related courses. Many people might find it dry and boring which lowers their motivation and interest in learning the concepts of programming [14]. Over the years an increasing number of teachers have endeavored to integrate educational computer games into training and teaching as it has been recognized that well-designed game-based learning is an effective means to help individuals construct knowledge and skill [5].

1.1 Overview of Project

This project will require research on past work that has been implemented to carry out game-based learning and create a design that is based on the research done. From there a game that teaches programming or programming concepts must be implemented. The implementation of the project is broad and any framework or language can be used for the implementation stage of the project. User testing is required throughout the project to gain feedback on the design and implementation of each level and the user testing will be done by 400-level students. The intended result of this project is to create a browser or mobile-based game that will teach the basic concepts of programming languages.

Summary of Project Objective:

- Research about previous work that has been done and the effects it has on users.
- Create a design for a game to teach programming and programming concepts based on the research that has been done.
- Implement a game for teaching programming concepts.
- User testing to be done on students to gather feedback on game functionality and effectiveness.

Chapter 2

Background and Related Work

Some people may find the process of learning programming unpleasant and difficult [14]. There are many theories on why programming education can be difficult, such as the traits of the individuals, for example, lack of motivation or becoming impatient with the lack of immediate results. Individuals may not understand how their program relates to the underlying system and are not able to create a clear model of the program. Programming does not only require one skill it is a complex cognitive activity where individuals must simultaneously build and apply their skills [12]. This section will discuss what game-based learning is and discuss some approaches that are taken to design a game to teach programming.

2.1 Game-Based Learning

Game-based learning (GBL) describes an environment where game content and game play enhance knowledge and skills. These games involve problem-solving spaces and challenges that provide players and learners with a sense of achievement. GBL maintains motivation and user engagement in comparison to some traditional techniques which can be dull and uninspiring [21]. Some of these traditional techniques are using big textbooks having multiple worksheets and more. Researchers and developers are trying to integrate educational content within game-based contexts, to transform the educational process to be a fun and engaging activity for learning [25]. Studies have considered that the use of digital games is very important in supporting education as students who used video games for their subjects reported significant improvements in subject understanding, motivation, and diligence [20]. These studies have shown that two well-known approaches have been designed to create GBL, Authoring-Based and Play-based approaches.

2.2 Authoring-Based Approach

This approach is when a program has pre-programmed elements and is partially implemented. This approach gets users to learn programming concepts by either constructing, completing, or modifying the existing code base the game provides. Previous research has stated that this technique has been proven to be very effective as this has helped many users to be able to learn and demonstrate a greater understanding of the programming concepts [12]. There have been past systems that have taken this approach to develop a game and this section will introduce and explain some of the previous systems that have used the authoring-based approach.

2.2.1 Scratch

Scratch is one of the previous systems that have been created with this approach in place. Scratch is a programming environment that lets users create an interactive media-rich project [13]. Scratch is a system that is mainly targeted at users at the beginner level and is used in schools, which allows students to create a simple game by scripting actions with sprites with drag-and-drop boxes. These drag-and-drop functions are colourful command blocks that control 2D graphical objects called sprites. Scripts are built by snapping together blocks that represent statements, expressions, and control structures. These block shapes suggest a way they fit together and will not allow the user to merge in a way that is meaningless, which prevents users from making unnecessary mistakes and allows them to make more progress [13]. Figure 2.1 shows the Scratch window that consists of the main command palette with a button that is always there to guide the user on what commands are available to them. A section shows the currently selected sprites for the script, another section shows where the game action occurs and towards the bottom is a section that allows the users to see the thumbnails that are available for the project.

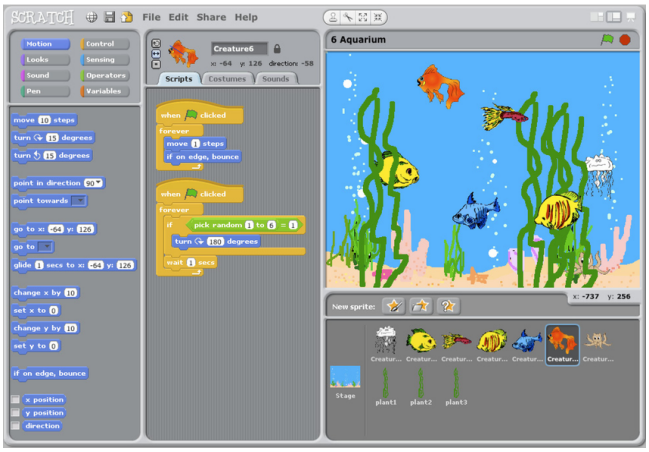

Figure 2.1: Scratch Interface

2.2.2 Alice

Alice is an open education resource made available to the teaching community to make it easy for non-programmers to use an interesting 3D environment to develop interactive games. Alice uses this environment to create computer animations using 3D models and uses animations, storytelling, and game construction to introduce object-based concepts to the users [12]. Similarly to Scratch, Alice uses a drag-and-drop environment that consists of statements and expressions. Users can see immediately how the program runs which can lead them to understand the actual program functions and different programming language constructs [27]. This also helps to sustain the user's interest and involvement. The target audience for Alice is non-programmers to use a 3D environment to develop a game. It has been stated in previous research that Alice can be a useful tool for supporting users and obtaining problem solving and arithmetic thinking as the user needs to build animations to solve a set of problems [27]. Figure 2.2 shows a similar interface to scratch, there are predefined blocks that help to develop a game.

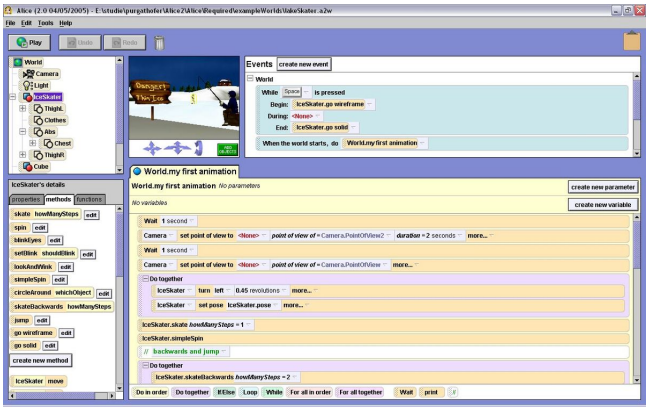

Figure 2.2: Alice Interface

2.2.3 Greenfoot

Greenfoot is an educational integrated development environment aimed at learning and teaching programming [11]. Greenfoot combines interactive graphical output with the programming language Java. The target audience is students around the age of 14 years old however, this tool is also suitable for students in college and university education. One of the goals of Greenfoot is a design that explicitly visualizes important concepts of object-orientated programming [11]. Unlike Scratch and Alice Greenfoot does not use drag-and-drop elements. Greenfoot has source code that has some base code already provided that helps the user to get started with programming. Figure 2.3 and 2.4 shows us the interface of Greenfoot which is slightly different from Scratch and Alice. Figure 2.3 shows the main window which is the Greenfoot world and where the program executes. This window holds the project scenario and the classes for the project and their interface relations. When double-clicking the class the Greenfoot editor opens which is what is shown in Figure 2.4. The editor has some predefined functions which are in Java that need to be modified by the user. Once some code has been written the project can be executed and the user can see immediate results, which again just like Alice can help sustain users' interest and involvement.

Figure 2.3: Greenfoot main window

Figure 2.4: Greenfoot Editor

2.2.4 Robocode

Robocode is similar to Greenfoot as it involves the user inserting code in an editor to progress with the game. Robocode is an easy-to-use robotics battle simulator that operates across various types of computer platforms supporting Java. [10] The goal of the game is to develop a robot battle tank to battle against other tanks in Java or .NET [7]. The player must write code for the AI robot telling it how to behave and react to events occurring in the battle arena. The battle occurring in the game is in real-time and on-screen, which makes the activity more dynamic and keeps users motivated and more engaged just like in Alice and Greenfoot [7]. This program allows students to define strategies and implement them according to the multiple programming procedures available. The editor involves some predefined functions in Java that need to be completed or modified just like Greenfoot. Figure 2.5 shows the battle arena that shows all the tanks and Figure 2.6 shows the editor that appears and needs to be completed to progress in the game.

Figure 2.5: Greenfoot main window

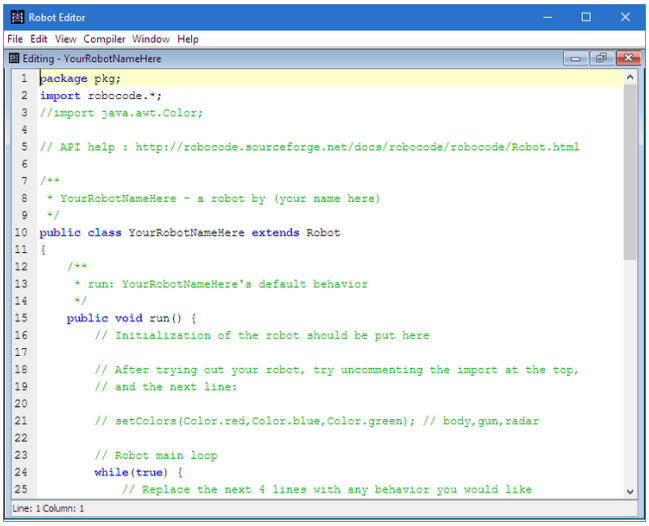

Figure 2.6: Greenfoot Editor

2.3 Play-Based Approach

The play-based approach is when there are a series of missions and tasks which is related to a concept. From this approach, the user can gain knowledge from the concepts by developing programming skills to complete the tasks and missions provided by the game [12]. Even though this technique may require users to develop code for the system as their solution, this differs from the authoring-based approach as there is a smaller emphasis on coding. The code given is usually restricted to completing a function or completing a certain task. Systems that use this approach also try to target a specific programming concept, unlike most authoring systems that try to cover a full language or concept. Keeping it concise this way helps users to focus on smaller areas of the language instead of overwhelming them with too much information. There has been some previous work done with this approach in place.

2.3.1 Saving Sera

The game Saving Sera is a 2D game where the user must save princess Sera who has been abducted. The users must complete several tasks to progress in the plot of the game [8]. These tasks involve the user completing lines of code that will result in an executable program, or they have to map existing lines of code to their proper position or order [8]. This game introduces concepts such as simple and nested loops, recursion, and control structures all in the process of trying to complete tasks. Figure 2.7 shows one of the screens from the game Saving Sera, we can see a task given to the user as well as a dialog screen that lets the user enter a bit of code to complete the task.

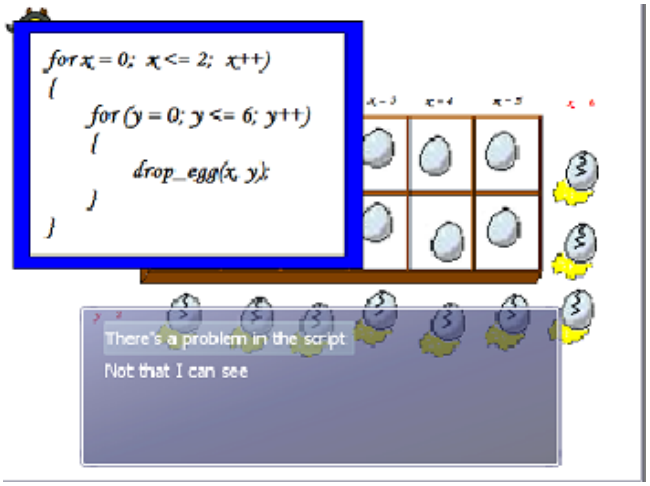

Figure 2.7: Saving Sera

2.3.2 Light-Bot

Light-Bot is a puzzle game where the objective is to program a small robot to light up all the blue blocks on a grid-based board [4]. The objective is completed by giving the robot a set of instructions from a limited set of commands, with a finite instruction space [4]. As the levels progress the board becomes increasingly complicated, and the user is required to create a combination of commands that are more simplified to achieve the task. Figure 2.8 shows the game on level 1 with some of the limited set of commands.

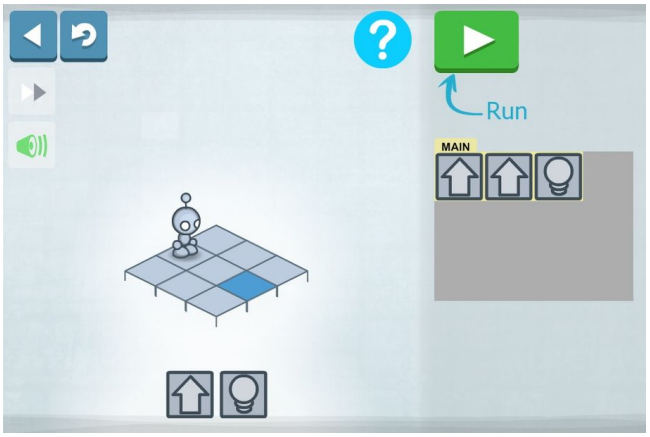

Figure 2.8: Greenfoot Editor

The program gives the user predefined command tiles as we can see in Figure 2.8, and the user can only hold a certain number of commands. This will push the user to use additional functions for repetitive actions or to develop a recursive solution [12].

2.3.3 Human Resource Machine

Human Resources Machine is a visual programming game with the concept of office workers performing simple tasks which represent machine code instructions [2]. Players a required to create a visual programming loop that allows them to automate the task that is produced by the boss. As you succeed the player is shown how many instructions it took and how long it took to process that program on average and they get promoted to the next

level. The program includes predefined commands to create a list of instructions to perform each task, these predefined commands include picking up and dropping items, performing addition or subtraction with the carried item, and making decisions based on the value of the carried item. These elements mimic assembly language, which is any low-level programming language with a strong correspondence between the instructions in the language and the architecture's machine code language. The interface of the game is a simple office environment with predefined buttons that allow the user to create instructions for the player which can be seen in Figure 2.9.

Figure 2.9: Human Resources Machine Interface

2.3.4 Codemancer

Codemancer is an educational game with a target audience of 6-12 year old's however it is still considered fun for adults too. Players are expected to code their way through a fantasy world full of rival sorcerers and their minions. Players will position magical runes which represent commands, some of these commands are direct movement commands such as "move forward", other commands are what are considered "flow control" that determine the execution of the spells such as loops [9]. Arranging these runes in order allows the users to move in any desired pattern and respond to unexpected events within the game. Figure 2.10 shows the possible defined commands present in the game.

Figure 2.10: Commands available in Codemancer

The gameplay takes place in a hexagon world so it is easier for the users to estimate angles and distance. In figure 2.11 we can see the hexagon world that is Codemancer's User

Interface and a sequence has been added to complete the task.

Figure 2.11: Codemancer User Interface

2.3.5 TIS-100

TIS-100 is an assembly language in a form of a puzzle game. Assembly language is the lowest level of programming language and it is what the CPU on your computer is running, and due to this reason, the game is mainly targeted at those that have some experience in programming or knowledge of computers. Programmers rarely write in assembly language and often write a high level like C++ and then programs called compilers break those high-level languages down into assembly. In the game, each level involves users writing a program that completes certain tasks given. The tasks could include copying data from one field to another, adding numbers, storing numbers, and so on. On top of this, the user is given one or more inputs feeding into a grid of processing nodes and one or more outputs and the user must process those inputs into appropriate outputs by writing instructions in each of the nodes [23]. The game does not really have many visuals, the interface mostly is text or coloured blocks/bars on a screen however this interface makes sense for the concept of the game, an example of an interface can be seen in figure 2.12.

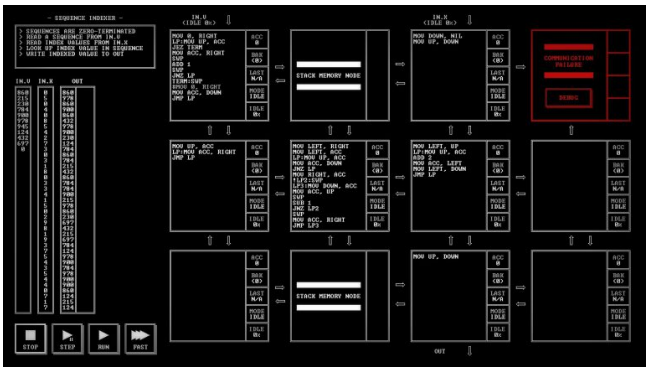

Figure 2.12: Interface of TIS-100

2.4 Summary

Table 2.1 summarizes the key differences between the two approaches. Both approaches have some drawbacks and some benefits. The authoring approach allows users that are at a beginner level to understand concepts and gives them a base and structure to start with, however, the play-based approach does not require as much coding, so they tend to not

produce as much structure as authoring provides. However, this is a drawback to play-based as users that are at a beginner level can get confused while using the game.

Play-Based provides a scenario where tasks need to be completed to progress through the game, however, authoring does not provide a story or plot to develop and usually does not provide tasks. This has its pros and cons, letting users have the freedom to construct and develop what they want gives them exposure to several things however confining users to a task or mission helps them to focus on specific concepts or language without overwhelming them.

For authoring users are usually provided a program that is a game-themed assignment alongside a set of development libraries and partial implementation which can cause users to be less involved and interested, however, for the play-based approach, the program is set out as a game, and requires very little amount of coding. It usually is a program that needs functions completed or specific commands such as Light-Bot for example. This makes the user feel less like they are doing something educational and gains their attention and grows their interest.

Both approaches focus on programming however, play-based tends to focus on a specific language or concept where in most cases authoring tends to be quite broad in what they try to cover. Keeping it concise will help the user to focus on specific things and have a deeper understanding of them instead of having a shallow understanding of a broader concept.

Authoring Approach	Play-Based Approach
Provides base code that the user can construct, complete or modify.	Given scenarios where users have to complete tasks by implementing code.
Allows users to construct and modify the given code or structure as much as they like.	The approach provides code that restricts users from completing a function or a certain task.
Focus on larger concepts and how programming works instead of a single language.	Focus on one specific language or concept
Requires some sort of code or works withdrag-and-dropp functionalities.	Doesn't require as much code to complete tasks.
Let user see their progress immediately which sustains their interest and involvement.	The design makes it feel like users are playing a game and not implementing code or learning which keeps them interested.

Table 2.1: Comparison between the approaches

2.4.1 Approach Chosen

Based on the findings and the research, the most appropriate approach for this project would be the play-based approach. The target audience of this project is beginner-level programmers and this approach would be most suitable for them. This approach requires very little coding and is focused more on problem-solving and programming ideas which will allow beginners to create a better foundation in programming. Having a narrative and creating tasks and missions based on them for users keep the users engaged as well as guide the users on what's required for each level instead of confusing or overwhelming them.

Chapter 3

Design

After the research had taken place there were two kinds of design approaches found that can be taken to implement some systems, authoring and play-based approach. Based on the considerations in section 2.4.1, play-based was the design approach that was decided on. Play-based allows users to focus on one concept or language rather than making the programming concept being taught too broad. Taking this approach for this project, the teaching focus for the game will be on introducing programming concepts that are found in most traditional imperative languages, such as Java, C, C++, Python, and Ruby.

The design approach for play-based includes a plot or a scenario for a game which helps create tasks and missions for users to complete. Following that approach, for this project, the scenario would be the user helping a dog, and the user's task would include getting the dog to his food or bone with the use of the programming concept they learn. A reason why a dog was selected as the main character was due to how previous work mentioned in sections 2.2 and 2.3 created the main character according to their game plot. For example, in Human Resource Machine the game is based on an office environment, therefore, the main characters were zombie humans that had to be moved around to complete tasks. Another example is Light-bot where the plot is to light up all the blue blocks on a grid-based board which leads them to create a robot as their main character. Inspired by this pattern found in past work, the initial plan was to have a user complete tasks on a grid-based board, and the reason a dog was selected was that it was perceived to allow easy creation of scenarios for each level.

In the proposal the rough idea was to allow the users to input code in an editor, however, after researching, the initial idea changed, and instead of an editor, the user will control the dog with button commands that will be predefined for them. The reason for this change to occur was this allows the game to be more restricting in the possible things the user can do and prevents the users to go off track. Also, it can be considered more inviting to have buttons in comparison to having an editor with lines of code. This change was also influenced by the play-based approach from previous related work mentioned such as Light-bot, Human Resource Machine, and Codemancer which limit their users to buttons that help perform programming concepts. These button commands will provide the users with programming commands and help the user complete the given tasks. The dog will be moving around a board made out of tiles to simplify the movement for the users as the dog will move a tile at a time. For some levels when complexity increases, users will be able to locate the dog's actual position and the tiles they are on. The game will have multiple levels that introduce new concepts and new button commands. Levels start becoming challenging with obstacles being added and the puzzles will need to be solved with the knowledge gained about programming. To add more complexity the board sizes will change for some of the levels as this will make the levels less predictable.

There was a base design layout that was implemented for the game which is shown in Figure 3.1. This figure was used as a template to locate all the components of the game on the screen. The instructions and command buttons are dependent on the level and what programming concepts are being taught. The command sequence allows the user to view the commands they have selected for the dog to complete. This template would change further on due to more command buttons being available, instructions being either larger or smaller, or the board being different sizes. An example of this change is seen in Figure 3.2.

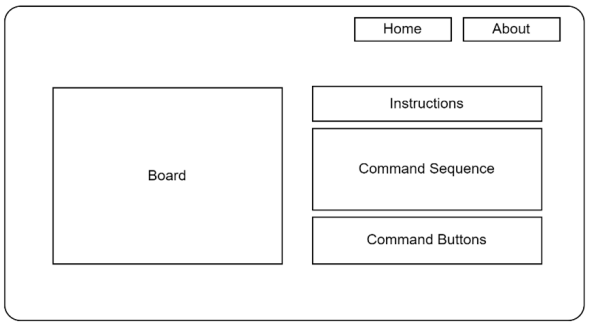

Figure 3.1: Simple design template

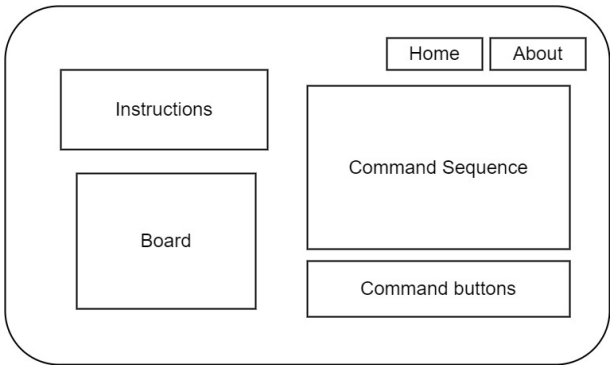

Figure 3.2: Changed design template

3.1 Design Iterative Changes

To be sure that the design approaches that are considered for the project are in fact user friendly and fulfil the purpose of teaching users programming, user testing was undertaken. This section will speak about the iterative approach done with usability testing to design a user-friendly interface, however, more technical testing and a more in-depth discussion will take place in the Evaluation section, section 5.

Usability testing is when participants are asked to perform tasks usually with one or more specific user interfaces, and while participants complete the tasks the researcher observes the participants and records feedback [15]. Four users were selected to user test the interface. The reason for only selecting four users was to follow similar principles mentioned by Jakob Nielsen. Nielsen established the discount usability engineering movement for fast and cheap improvements of user interfaces and has invented several usability methods which include the 10 usability heuristics [18]. Nielsen recommends testing with a smaller number of users because it distributes the budget for user testing across many small tests and keeping the numbers small will help to test more effectively and efficiently. As for this project, the cost is not a concern however in a real-case scenario companies would benefit a lot from this principle as having many iterations of user testing with a large number of user testers can be costly.

A graph created by Jakob Nielsen shows how effective a small number of users with more iterations of user testing occurring can be.

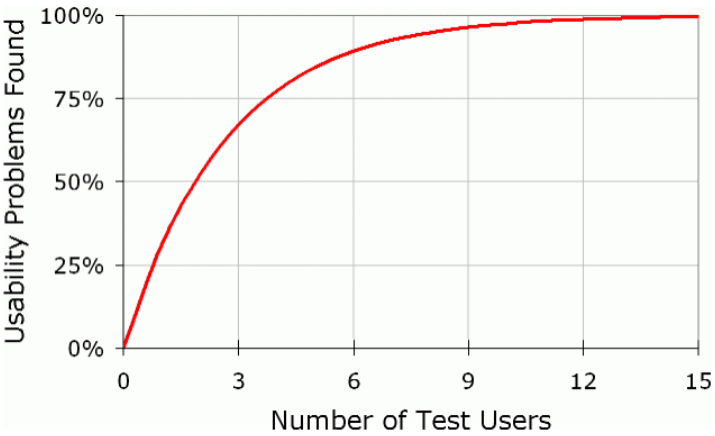

Figure 3.3: Graph created by Jakob Nielsen

With the use of this graph, we can see that when you collect data from a single user the insight about your user interface increases and you learn almost a third of what there is to know about the usability of the design. The more users added the less you learn because there will be a repetition of the things you see and there is no need of observing the same thing multiple times [16]. That is why for this project four users were selected to effectively test the interface.

3.1.1 User Design Feedback

The user testing was conducted several times to be able to receive valuable feedback on the interface and observe the interactions users had. There were four 400-level students selected to test the game, and the feedback received by these participants generally was positive. However, the target audience initially had been beginner programmers around first-year students or even high school, but as the game progressed the target seemed to change from teenagers/young adults to children. This was confirmed by the users as they were asked the question ”Who do you think the target audience is for this game?” and two users thought the aim was for children due to the design of the interface. However, the programming concepts that are introduced are done well and can help anyone learn. The design of the interface was said to be bright and something to be eye-catching to children in primary or intermediate, this can be seen in figure 3.4.

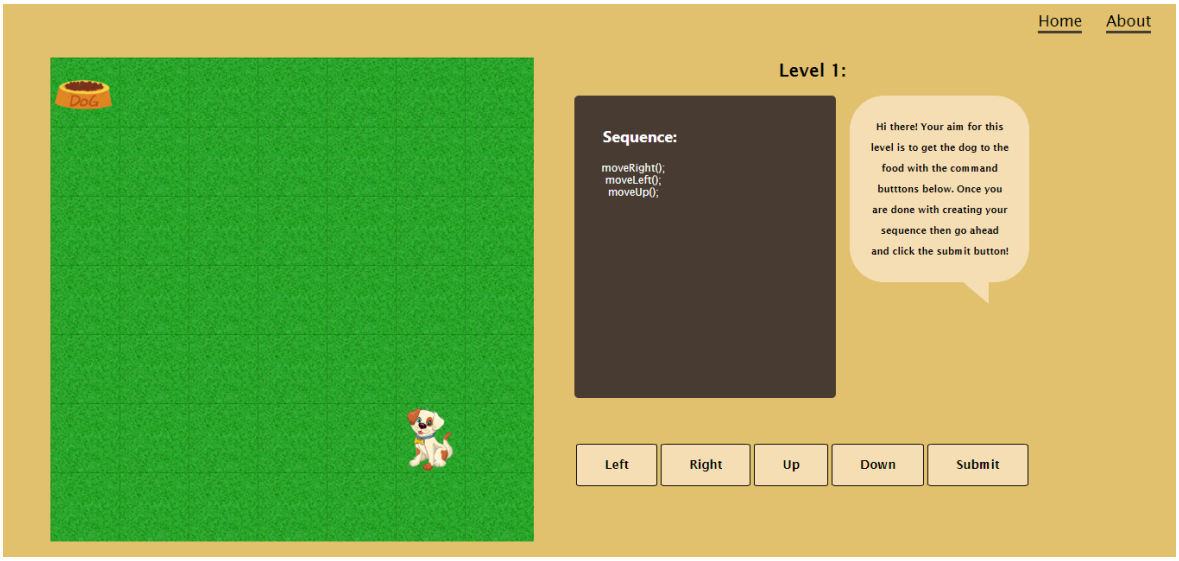

Figure 3.4: First design of the interface

The feedback for the interface design was considered and darker tones and colours were used to give a more grownup effect and images were changed to give a less cartoon style.

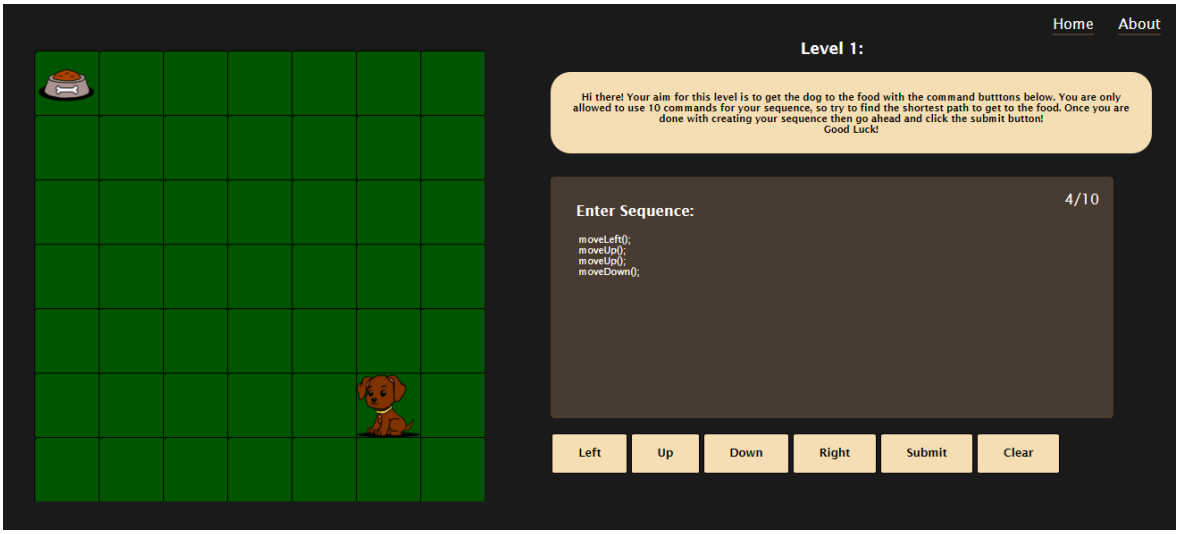

Figure 3.5: The improved design of the interface

One of the final iterations for the interface was a minor accessibility issue. A user mentioned they had a hard time distinguishing between the command buttons and the non-command buttons and suggested making them different to be easier to notice. Below is the feedback implemented.

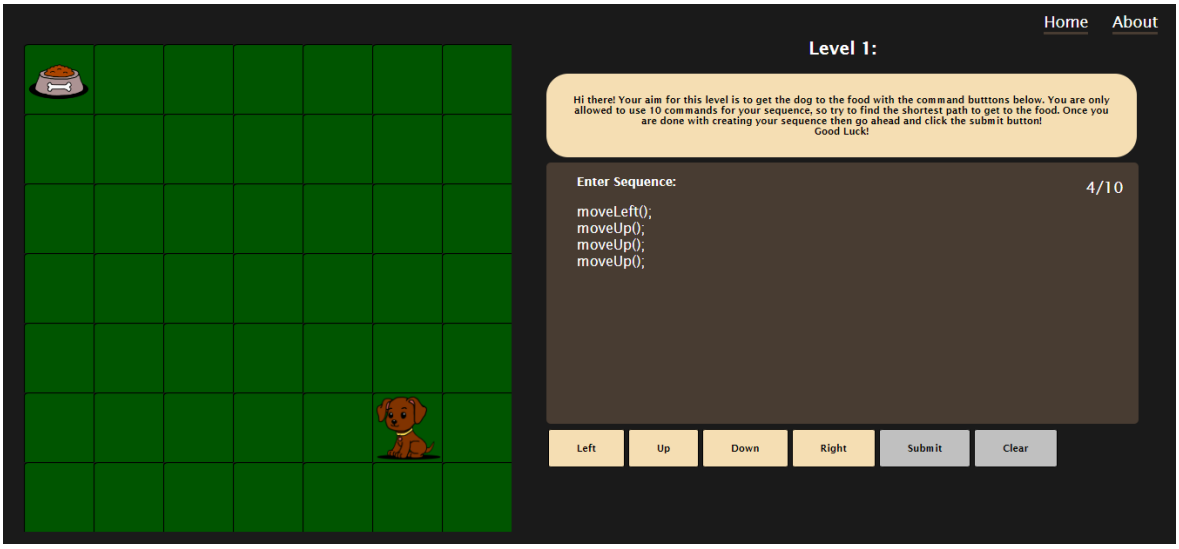

Figure 3.6: The final design of the interface

There were two other users that said the narrative of the game, which was helping a dog to complete tasks, could make the game more children focused however the game does teach good basic programming concepts which makes it targeted at anyone who is learning to program.

When doing the first user testing it was suggested that the game lacked complexity and didn't get the user to think too hard about the command sequence. For each level instead of leaving the command sequence open-ended, a limit was added to control how many commands are allowed to be selected. Adding this complexity to the levels forced the user to think about the shortest route they can take and think more about the sequence instead of mindlessly selecting commands, which kind of mimics how some programming algorithms work. An example of the command limits is shown in the figures mentioned above in the top right corner of the command sequence component which shows how many commands are allowed for that level.

The starting few levels were good introductions to get the user familiar with the game, but in order to start challenging the users more to prevent them from losing interest in the game, the levels had to start incorporating some complexity. After the first few user tests, feedback was provided that the game could include some obstacles. The first obstacle that was created according to the narrative was a cat, this obstacle allowed the introduction of if statements and create scenarios where users had to implement harder sequences. As levels increased more complexity such as holes, multiple cats, unknown food bowl targets, random moving cats, multiple end goals, and even multiple dogs were added. After these obstacles were added, users really liked how the complexity increased each time they moved up a level and they were still able to learn a programming concept. For example, a level included obstacles with a cat and holes, the user had to bark if there was a cat present and jump if there was a hole in the dog's way, with this narrative the game introduced the concept of using if else statements and required the users to use that knowledge to complete the level. An example of this can be seen in the figure below.

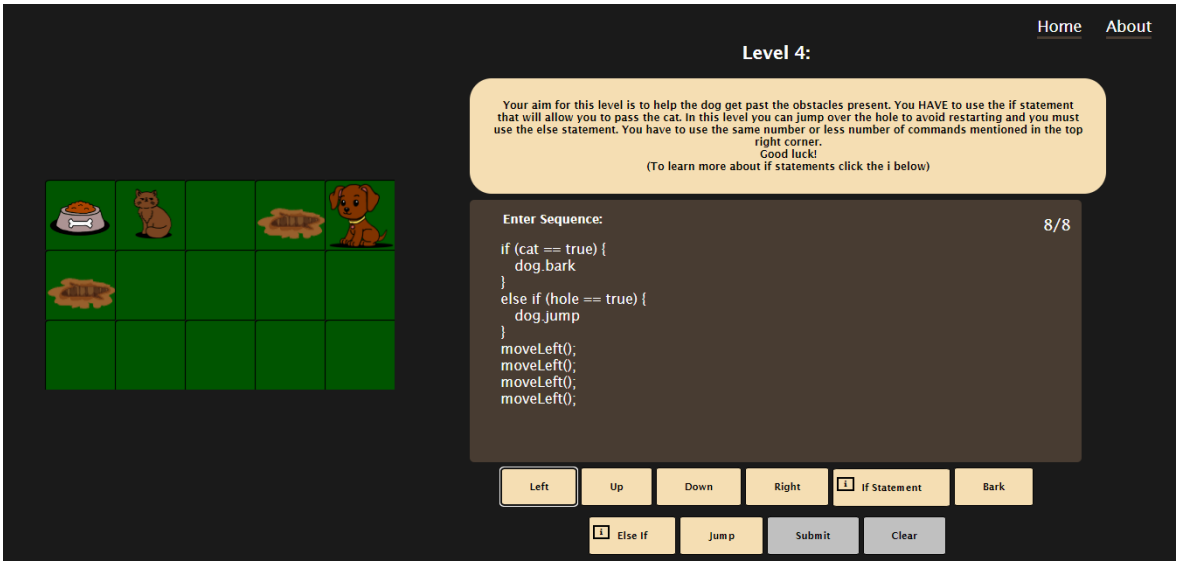

Figure 3.7: Example of an obstacle

These obstacles that were added to the game made the puzzles harder for the users. Creating harder puzzles was important as it would challenge the user to implement their understanding of the computational thinking behind the task. These obstacles allowed to create levels that incorporated multiple programming concepts into one level which would allow users to understand how each concept works with one another. The engagement of these users is very important for the learning process as it has been said that this increases the user's attention and focus and motivates them to engage in higher-level critical thinking [19].

From the feedback provided by the users the design of the game does indeed help to introduce programming concepts and teaches the users, however, the game scope did end up being more suitable for a younger audience. Due to it being too far into the year and the initial design being for beginner programmers and not children an ethics approval was not requested at the start of the year and it was too late to request one later on. If ethics approval had been part of the project then the game would have been tested on children to be able to grasp a better understanding of their interaction with the interface and design. It is recognized that this aspect of the project is not fully tested.

3.1.2 Usability Heuristics for User Interface Design

Jakob Nielsen had ten general principles for interaction design which are broad rules and guidelines for a better interface. For this project, some of those principles were followed and met to produce a user-friendly interface.

Principle 1. Visibility of system status: The system should always keep users informed about what is going on with the use of appropriate feedback within a reasonable time [17].

In the game, it was made sure that instructions about what to do and how to operate the game were clearly stated right at the top. And if the user failed the game, it was made clear with a popup box to let them know what happened and what they need to work on. With the help of user testing for levels, clarity was given to some instructions as some users sometimes found them too wordy or hard to understand.

Principle 3. User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. [17].

Taking this principle into consideration a "Clear" button was added to help the user to erase the sequence if a mistake was made without having to reload the page or go back.

Principle 6. Recognition rather than recall: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate [17].

This principle was implemented in the design by having the instructions thoroughly explained and if something was continued from the previous level it was mentioned briefly again as a part of the current level. Also, small increments were tried to be made from one level to another to help the user understand each level and get more practice on it.

Principle 8. Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility [17].

The design of the interface was created to be as simple as possible as it could easily overwhelm the users. If there needed to be more information added to the level extra buttons were added to keep the interface as minimalist as possible. This can be seen in Figure 3.8 where the instructions are moved to a button.

Figure 3.8: Interface with extra buttons to keep it minimalist

Principle 9. Help users recognize, diagnose, and recover from errors Error messages should be expressed in plain language, precisely indicate the problem, and constructively suggest a solution [17].

When a user fails a level, a prompt would let the user know what they could have done wrong and guide them in the right direction to try again. For example, if the user hit the boundary of the board, it would notify the user and not be too vague. User testing helped to align to this principle better as the failed popup was originally generic and did not inform the user what they did wrong.

Principle 10. Help and documentation Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation.

This was implemented into the interface design by adding information buttons for all the programming concepts such as while loops, if statements, variables, etc. These information buttons would produce more information about the chosen concept and give a breakdown of the syntax or provide a definition. This can be seen in Figure 3.9.

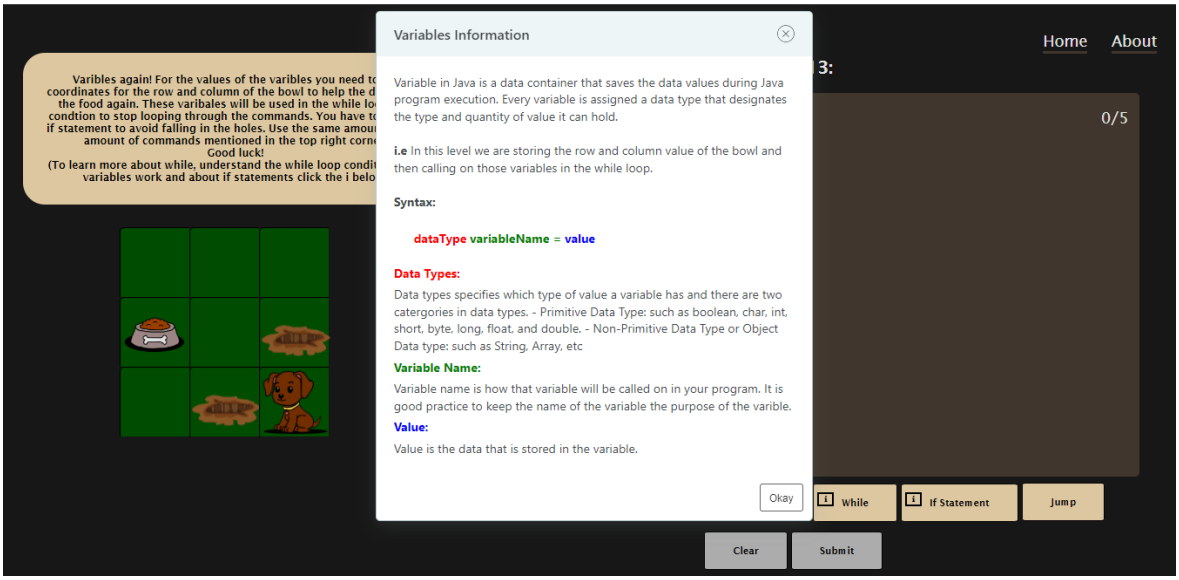

Figure 3.9: Information popup when variable information button is pressed.

3.2 UML Diagrams

A UML Diagram was created to describe the attributes and operations involved in a class and to model the system. The UML can be seen in Appendix A. It shows the base classes needed with the necessary methods and fields. To clarify each class, the controls class is where most of the functionality occurs for the commands. The Board class creates the number of tiles and creates the board. The Panel class takes the controls class as a component and calls that in the board class and styles how the controls will be placed on the browser. The App class is the main method that runs on the web browser and shows the board. The check class is essentially getting the dog and bowl element id from the board class and checking if the user has won or failed the level. The getElementById is an inbuilt JavaScript method that allows access to elements from different HTML files. Finally, the Pages class navigates to the About page and Home page.

Throughout the project, the control class had become very large and most of the logic had been added to that class. However later in the project, the class had numerous functions added, and there was quite a lot of repetition of code when more levels had been created. Therefore refactoring the system would have been time-consuming and could have affected the implementation of the previous levels. If the system could have been created differently the control class would be separated into multiple classes which would reduce code repetition for each level as well as help to organize the code structure better. For example, having a different class for movements and having different classes for conditions such as, if and else if statements. This would have helped with redundant code as well as allowed easy access to the functions that were needed. The current code structure would probably get very messy if a large number of levels were made or if multiple people collaborated on it. Considering a real-world scenario, the code structure would be very expensive as most companies make sure that the code implemented is as efficient as possible, and if this project was much bigger the repetition of code could be very costly.

3.3 Use Case Diagrams

To understand the system requirements a few Use Case diagrams were designed. A Use Case Diagram is used in system analysis to identify, clarify, and organize system requirements. The first diagram, Figure 3.10, lays out the general sequences of interaction between systems and users.

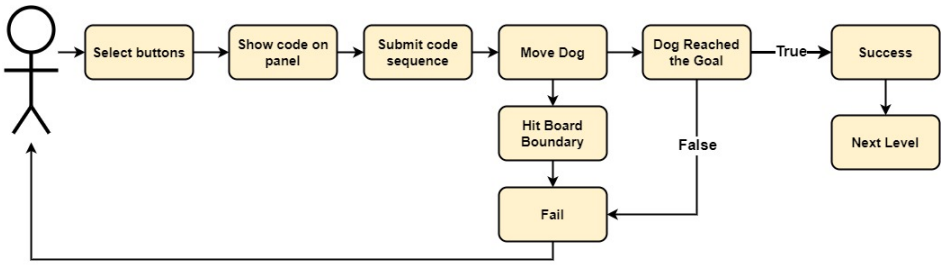

Figure 3.10: Use Case Diagram simple sequence example

The second use case, Figure 3.11, is an example of a user using the move commands to create a sequence to reach the end goal, which is the food, and what the possible outcomes could be.

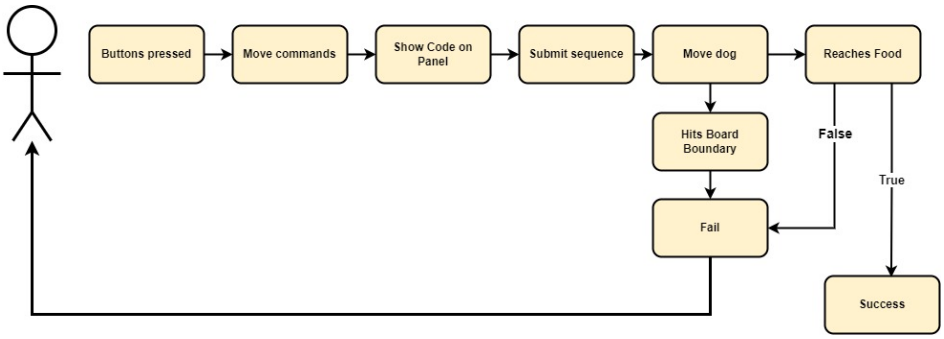

Figure 3.11: Use Case Diagram move command example

The third use case, Figure 3.12, is an example of the user using the while loop command and the movement commands to move the dog to reach the food. This use case shows how the user could fail or succeed at this.

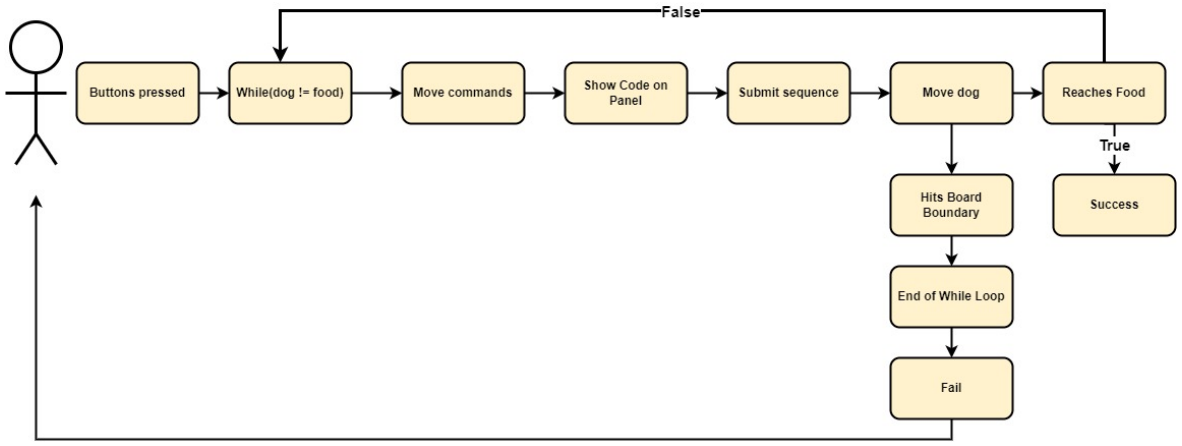

Figure 3.12: Use Case Diagram while command with move example

The last use case, Figure 3.13 is an example of the user using the while loop, if condition, and move commands to reach the food.

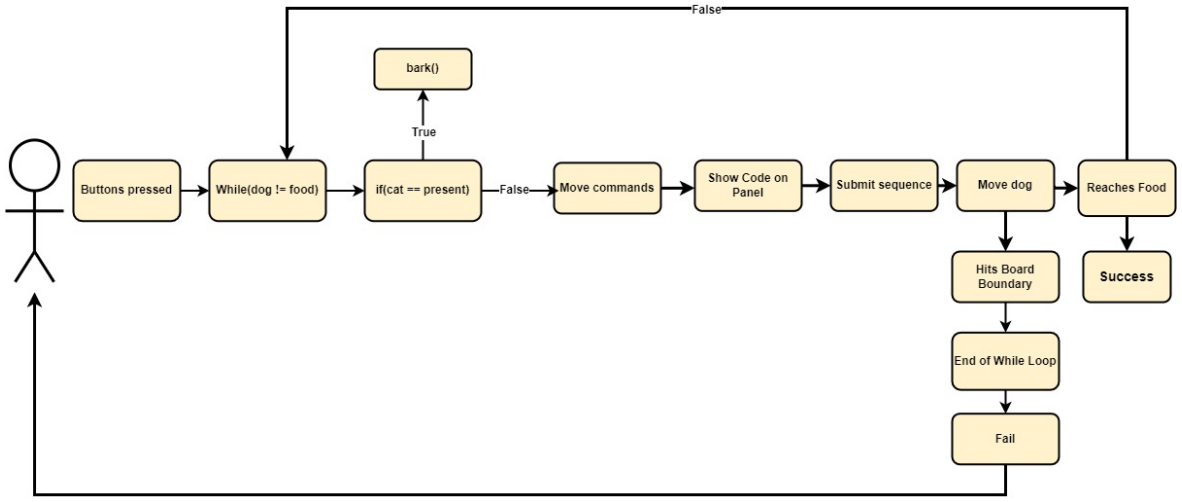

Figure 3.13: Use Case Diagram while and if statement with move command example

With the help of these use case diagrams, it can clearly identify the possible scenarios and outcomes as well as identify the system requirements which would help to test levels and implement more levels.

Chapter 4

Implementation

4.1 Technology chosen

As this project will be a web-based game the framework chosen was heavily influenced by the most commonly used frameworks for browsers. There are three popular front-end frameworks React, Angular, and Vue. Angular is a typescript-based free open-source web application framework that offers excellent user experience and responsiveness. Vue is a progressive framework for JavaScript used to build web interfaces and single-page applications. React is an open-sourced JavaScript library used to build user interfaces based on UI components. When looking at these frameworks and libraries, they all had their pros and cons. Angular offers faster load time, and framework elements and modules are fully customizable which gives more freedom to developers and designers. However, what brings Angular down is this framework is more suitable for larger projects and if the size of the project is not appropriate and not complex enough, this framework may weigh the performance down [28]. Vue framework is lightweight, which makes it convenient to use for developing projects of various complexity, and due to it being so lightweight the loading and installation of libraries are very fast. The cons of Vue are, there is a lack of plugins compared to Angular and React and this leads to developers needing to switch to other resources. There are also limitations in making use of the community as many elements of the framework are only available in Chinese or not many people use the framework, this results in a big disadvantage of this framework as the help provided online is limited[3]. React improves performance enhancement due to virtual DOM. The DOM is a cross-platform and programming API that deals with HTML, XML, or XHTML. With most frameworks when the DOM updates it slows down the performance of the application, however, with React because of the virtual DOM it exists entirely in memory and is a representation of the browser. So when a React component is created it will not update the DOM directly, it instead writes virtual components and turns them into the DOM which produces a smoother and faster performance [6]. Other advantages of React are this library is very well known and help is more accessible than other frameworks. A con with React would be that it can be high maintenance, due to the environment continuously changing this results in some developers having to relearn processes or new mechanics constantly, this also leads to a lack of documentation sometimes, as there are updates and new releases which leads to documentation sometimes not being up to date [6].

After weighing out the pros and cons of these frameworks React was chosen as the front-end library for the project. React and Angular were both great options, however, due to not being aware of how large and complex the project could be, it couldn't be determined if the Angular framework would weigh down the performance of the application. Also, due to Angular being in Typescript this would have made the learning curve quite steep if the

framework was chosen. Vue has a big disadvantage of lack of community and documentation this would make the development process difficult as finding resources would take up most of the time. React is a library that I know well and am comfortable using as well. Also React is a popular framework in local industries and it would be beneficial to use it to gain more experience. However, other great game development technologies that are available such as Unity, Solar2D, Unreal etc which would help create better user interfaces.

A Browser-based application was chosen over a mobile-based application as I have had more experience with mobile applications in comparison to browsers. This project would give me the chance to gain further experience with it. The development process however for both would be similar but for mobile, React Native library would need to be used instead which is an open-source UI software framework used to develop applications for iOS, and Android. Another reason to choose a browser-based application was that it would allow the UI to be more spaced out and not as crowded as it could get with mobile applications, which could hinder the usability aspect of the game.

4.2 Implemented Design

Essentially the plan for the game was to implement about 4-5 levels and each level would introduce a different concept. The reason a small number of levels were mentioned initially was due to being unsure of how long each level may take to implement. However, after the first two levels that took the most time, the base code had been implemented and further implementation become easier to carry out. After some user testing was conducted users suggested adding obstacles to the levels to add more complexity and depth. After the feedback, the initial plan was reviewed and more levels were designed to add complexity to the programming concepts being introduced. The main character allowed easier creation of tasks that could be completed for each level.

There was a total of 21 levels that were implemented and the programming concepts that were covered were, if statements, if-else statements, while loops, variables, Boolean, for loops and, ArrayList collection. Figures of some of these levels working can be seen in Appendix B. These concepts help build the basic programming foundation needed for imperative languages. The order in which these concepts are introduced was also taken into consideration. This is important, for example, having a while loop introduced after the if statement because a while loop is essentially an extension of an if statement. Another example is making the users aware of the idea of variables before introducing for loops as for loops work with initialization within the loop. Along with the visual aspect of the programming concepts, there were also information popups that were implemented to educate the users more about what they were visually seeing. This aspect of the game was needed as it is important for users to understand the theory and syntax of what they are trying to learn as that knowledge will help them create a better foundation.

The system architecture for the project was structured using a method of using layers, which is separating the infrastructure of the application into individual sections and having each section responsible for its own tasks [24]. There are 5 layers in software architecture, Presentation Layer which handles the UI and interaction of the users with the software, Application Layer which handles the main programs of the architecture, Business Layer where the application business logic operates, Persistence Layer is also known as the Data Access Layer contains the code to access the Database Layer and finally, Database Layer where the system stores all the data [24]. The project structure can be seen in Appendix A and using it as a reference the Presentation layer was represented by the App and Board class which handles the user interface and user interaction and the Control class was the Application

and Business layer which handed the main logic of the game. The project did not have a Persistence and Database Layer as the project did not use a database at all, however, in the future, further levels could be stored in a JSON format and stored in a database as this can help to reduce code repetition as well as separate the layers out. Dividing an application into these layers has its advantages such as each layer performing its own functions which allows you to change a layer of the architecture without those changes directly affecting other components [24]. Also, when the application is divided into layers it becomes easier to test individual units rather than a larger system, which is very beneficial as it lets you gather critical information about every layer [24].

4.3 Implementation Methodology

For the implementation, an Agile Methodology was carried out. Agile Methodology is a method that is based on iterative and incremental development, it allows working on projects in an effective and efficient manner. For the Agile approach, customer satisfaction is the highest priority and the customer has direct involvement in evaluating the software [22]. There are methodologies that can be implemented in Agile Projects which are focused on different aspects of the software development cycle, such as some focus on the practices, while others focus on managing the software projects. For the implementation of this project, the focus was on managing the software project, therefore the Scrum methodology was taken. Scrum is based on an incremental software development process where the entire development cycle is divided into a series of iterations which is called a sprint [22]. This method started off by collecting user requirements however not all requirements should come from the users at the beginning, here users can provide their input and add or remove features. The next phase is listing the requirements also known as the project backlog. The sprints are planned out which discusses how many sprints are needed, the duration of the sprint, and what requirements from the product backlog need to be implemented each sprint. Due to there only being one person working on this project there weren't any meetings and end of spirit reviews. The sprints were documented in GitLab with the help of iterations and boards feature. The sprints were created with a duration of 2 weeks, and issues were assigned with the requirements to each sprint which were visible on the board in GitLab. After each sprint, it was reviewed what requirements for the sprint have been implemented and what have not. The issues that have not been completed move on to the next sprint. The advantage of Scrum is that it reduces the risk of development, these increments occur on the small software that is delivered to the user, where feedback is taken and it warns the developers about the upcoming problems which may occur in the later stages of development and allows them to see bugs and errors early too [22]. After each iteration, the user is given the developed project where their feedback is taken into consideration and the customer's feedback is welcome at any stage of development, this can be an advantage and a disadvantage as well [22]. Customer interaction is key because the project is developed according to the requirements however if the customers are not clear about product features this can cause miscommunication between developers and customers creating an end goal that does not satisfy the requirements as well as consuming more time for fixes and changes.

4.4 Implementation Issues

There were a few setbacks during the implementation process and sometimes a few iterations are needed to overcome them. One of the issues was how to get the dog to move a tile at a time that would not require to be refactored later on in the project. Due to there being

more commands that needed to be added later on in the project, a flexible code structure needed to be created. Therefore it was important that the solution that is taken would not require constant refactoring. The reason why this was an issue initially, was because the command sequence was stored in a list and once the user submitted their sequence it would loop through the list and move the dog to the tile that the dog would have ended with, instead of iterating each command at a time and move the dog accordingly. This would have caused issues when programming concepts would have been added to the game because the concepts would not be visually visible to the users. For example for the if statement, if the user had to jump over the hole the user would not see that condition actually rendering and instead see the dog locating from the first tile to the last tile. An early solution was to use a finite-state machine to allow the dog to move commands at a time. The definition of a finite state machine is a mathematical model for a system that has a limited number of conditional states. For the finite state machine, all possible states the component could be in needed to be defined, all allowed transitions between the states, and a list of the actions that could occur and when they should occur had to be implemented. Before starting to implement this research needed to be done on how finite state machines work, and how to implement it in JavaScript with a React library. To have a better understanding a small program to do with finite state machines was created, however, when integrating finite state machines into the game it introduced bugs and errors, and the movement of the dog was not as intended. After spending some time with finite state machines, switch cases were used as a different solution. Switch cases were a simpler alternative that allowed to move the dog according to the commands in the list. From what we had initially with the sequence being put into a list that was then used in these switch cases, we iterated through the list, and with the help of switch cases according to the cases, actions were executed.

Adding sound to the project required some unexpected extra work. Initially adding HTML audio tags worked however it required manually pressing a play and pause button to trigger the sound which was not exactly what was wanted. Another implementation was done with React hooks, which store the sound in a state and when a button is pressed it triggers the sound. Using React Hooks made the sound play inconsistently and sometimes would not play at all, this was due to threading. Threading is the execution of running multiple tasks or programs at the same time. The React Hooks were not allowing both the sound and movement of the dog to run at the same time. After looking at a few React libraries, the library Howler.js played sounds asynchronously, which means that it helps to run the sound in the background or tasks while other events are running. This library solves the issue of threading that was encountered while using React Hooks. The library provided was very helpful and the sound was successfully added to the project using Howler.js.

Chapter 5

Evaluation

The project specification was to implement a game that would help to learn programming or programming concepts. The game implemented focuses on basic programming concepts that help to build a good foundation to learn how to program. Each level challenges the user to think and use the knowledge gained to solve the puzzles.

5.1 Technical Testing

User testing was largely talked about in section 3, however, alongside user testing, functional testing was also executed as well. Functional testing is a type of testing that checks whether each application feature works as per usual and as according to system requirements. Some functional testing took place to test some of the front-end aspects and some functions as well.

To carry out functional testing the library Jest was used. Jest is a JavaScript framework that allows you to write tests to ensure the correctness of any JavaScript codebase. Snapshot testing was done using Jest, snapshot testing is useful when you want to make sure the UI does not change unexpectedly. This was carried out to make sure that the UI for each level stayed consistent with the last time it was updated and that nothing was changed by accident, this helps to not have unexpected bugs and errors and having to search through lines of code. However, snapshot tests did need constant updating and due to there being one person working on the code base it did get slightly frustrating. This is due to the tests failing and manual verification needing to be done to make sure that the application is working fine, then the tests would need to be updated. However, if there were multiple collaborators, snapshot testing would be helpful as it insures that the code base that is worked on by multiple people works correctly.

With Jest you have access to rich Mock functions API to test implementation. With these Mock Functions, test cases were created to test specific aspects of the project. For example, testing if the user is using the right amount of commands for a specific level and what happens if they don't. As well as Jest, React Testing library was used as additional tests, this library helps to test React components without relying on the implementation details.

A simple test plan for the functional testing was created due to the functionalities being tested with the use of user testing. A test plan refers to a detailed document that has the test strategy, objective, schedule, estimations, and deadlines [1]. The test plan that was followed for this project was to mainly test the business layer, which was the controls class that handles most of the logic. The use of functional testing for this project was to make sure that the functions created worked as intended and if there were any bugs in execution. Majority of the user testing covered testing the functionalities of the game requiring less

functional testing to take place.

5.2 User Testing

User testing also was carried out, as mentioned before four 400-level students were selected for usability testing. To evaluate the effectiveness of the game all the users were asked questions after they had completed each level. These questions helped to reflect on what they had learned, provide feedback on the UI and functionality as well as give back some suggestions for improvements. The questions asked were:

1. Did you like the level?
2. What did you not like about the level?
3. What's something that could be done better?

A table is created of the similar responses gathered for each of the questions. This table can be seen in Appendix C. The responses gathered with these questions allowed us to take in feedback and implement changes. From the feedback, it was clear that the project specification was met as the users were able to learn about basic programming concepts and it also was implemented in a way that kept the users engaged and interested.

One of the most common responses to the questions was, the UI was great and had a very fast response time. This was due to the framework selected, React allowed to create a well-designed web application and the react components and virtual DOM kept the response time fast and smooth. With the help of user testing, bugs and errors were also found. For example, for some levels, it was found that the dog went off the board and boundary checks were broken, a concept for a few levels did not quite make sense, and there were instructions that were too vague or did not make sense. And with all the feedback provided it was easier to fix and implement these changes and make a better version of the game.

The buttons that provided programming commands to the user to move the dog and complete tasks were said to be a great way to visually see how the programming concepts work, as well as restrict the user from going off track. This idea was acquired from the research done about the play-based approach and with the use of user testing, it was seen to work well for the project.

There was an additional question asked during one of the user testing sessions which was "who do you think the target audience is for the game." This question allowed us to understand that this game could potentially be for a younger audience as the UI and the plot of the game seems to be more suitable for a child or teenager. However, there were some users that believed that even though the game plot was suitable for a younger audience the concepts taught in the game are suitable for anyone who is learning how to program.

As mentioned before due to the intent of this project to be aimed at beginners and young adults ethics approval was not requested at the start of the project and it was too late to request one later on, therefore if the user testing were done again, an ethics approval would have been requested to be able to retrieve feedback from a younger audience and be able to observe their interaction with the game and if they are able to learn programming concepts from the game or not.

Chapter 6

Conclusions and Future Work

The game introduces some of the basic concepts that are needed for a good foundation to learn how to program and as well as keep the learning process engaging and enjoyable.

During this project, there were a number of 21 levels created for this project and those 21 levels helped build a basic foundation of imperative programming languages. The introduced concepts were if statements, if else statements, while loops, variables, Booleans, for loops and ArrayList collections.

The game can always have new aspects added to it therefore there is no defiant end to the implementation of the project. For future work, students or researchers can explore to introduce more concepts, such as more levels about classes, all collection types and how they work, threading, recursion, and more. Introducing these concepts at further levels will allow the user to gain more knowledge of imperative languages as well as practice the knowledge that they learn.

Along with introducing new concepts in future work, harder puzzles can be created as well with existing concepts. Due to the time constraint, there were only a few hard puzzles implemented however great concepts were able to be implemented and introduced within the game. For future work, those concepts could be taken further such as combining everything that has been taught into a few levels that would really challenge the users as well as allow them to use all the knowledge they have learned into one level. This would also show what actual coding can be like with all of the concepts present in one task.

If this project was carried on it would be better to move to a game development technology such as Unity or Solar2D. This is because game development technologies, like Unity, provide amazing graphics and visual effects that can make the user experience much better. There are also asset stores for them that allow you to download game assets, ready-made solutions, and functional extensions. However, Unity allows free downloads whereas with Solar2D you must purchase graphics from their store. Just like React, for these game engines you code the logic and some of the components so essentially they are similar but game engines create a better UI in comparison to React.

Overall the project met the brief of carrying out research, designing a game based on the research, and implementing a game that is able to teach programming concepts. The project can still be improved in future work and more levels can be implemented with further complex levels and adding multiple programming concepts.

Bibliography

[1] BROWSERSTACK. Test planning: A detailed guide. https://www.browserstack.com/guide/test-planning, 2020. Accessed: 2022-10-19.

[2] CHACHANIDZ, E. Serious games in engineering education. https://www.proquest.com/docview/2213791345?pq-origsite=gscholar&fromopenview=true, 2019. Accessed: 2022-09-12.

[3] DEVELOPMENT, D. Pros and cons of vue.js framework programming. https://ddi-dev.com/blog/programming/the-good-and-the-bad-of-vue-js-framework-programming/, 2020. Accessed: 2022-10-14.

[4] GOUWS, L., BRADSHAW, K., AND WENTWORTH, P. An evaluation of the educational game light-bot. *Computational Thinking in Educational Activities* (2013), 10–15. Accessed: 2022-05-21.

[5] HUNG, C., SUN, J., AND YU, P.-T. The benefits of a challenge: student motivation and flow experience in tablet-pc- game-based learning. *Interactive Learning Environments* (2015), 172–190. Accessed: 2022-05-18.

[6] JAVATPOINT. Pros and cons of reactjs. https://www.javatpoint.com/pros-and-cons-of-react, 2021. Accessed: 2022-10-14.

[7] JESUS, A., AND SILVEIRA, I. A study of battle strategy for the robocode. Gamebasedcollaborativelearningframeworkforcomputationalthinkingdevelopment, 2021. Accessed: 2022-05-29.

[8] KARAGIANNIDIS, C., POLITIS, P., AND KARASAVVIDIS, I. *Research on e-Learning and ICT in Education Technological, Pedagogical and Instructional Perspectives*, vol. 3. Springer, New York, 2014. Accessed: 2022-05-21.

[9] KICKSTARTER. Codemancer: A fantasy game that teaches the magic of code. https://www.kickstarter.com/projects/bobbylox/codemancer-a-fantasy-game-that-teaches-the-magic-o, 2020. Accessed: 2022-09-12.

[10] KOBAYASHI, K., UCHIDA, Y., AND WATANABE, K. A study of battle strategy for the robocode. *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734) 3* (2003), 3373–3376. Accessed: 2022-05-21.

[11] KOLLING, M. The greenfoot programming environment. *ACM Transactions on Computing Education 10* (2010), 1–21. Accessed: 2022-05-21.

[12] LI, F. W., AND WATSON, C. Game-based concept visualization for learning programming. *Proceedings of the third international ACM workshop on Multimedia technologies for distance learning* (2020), 1–6. Accessed: 2022-05-19.

[13] MALONEY, J., RESNICK, M., RUSK, N., SILVERMAN, B., AND EASTMOND, E. The scratch programming language and environment. *ACM Transactions on Computing Education 10* (2010), 1–15. Accessed: 2022-04-29.

[14] MATHRANI, A. CHRISTIAN, S., AND PONDER-SUTTON, A. Playit: Game based learning approach for teaching programming concepts. *Journal of Educational Technology & Society 19* (2016), 5–17. Accessed: 2022-05-17.

[15] MORAN, K. Usability testing 101. `https://www.nngroup.com/articles/usability-testing-101/`, 2019. Accessed: 2022-10-13.

[16] NIELSEN, J. Why you only need to test with 5 users. `https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`, 2000. Accessed: 2022-10-13.

[17] NIELSEN, J. Ten usability heuristics. `https://pdfs.semanticscholar.org/5f03/b251093aee730ab9772db2e1a8a7eb8522cb.pdf.`, 2005. Accessed: 2022-10-14.

[18] NIELSEN, J. World leaders in research-based user experience. `https://www.nngroup.com/people/jakob-nielsen/`, 2019. Accessed: 2022-10-13.

[19] OF WASHINGTON, U. Engaging students in learning. `https://teaching.washington.edu/topics/engaging-students-in-learning/`, 2022. Accessed: 2022-10-19.

[20] PELLAS, N., FOTARIS, P., KAZANIDIS, I., AND WELLS, D. Augmenting the learning experience in primary and secondary school education: a systematic review of recent trends in augmented reality game-based learning. *Virtual Reality* (2018), 1–18. Accessed: 2022-05-28.

[21] QIAN, M., AND CLARK, K. Game-based learning and 21st century skills: A review of recent research. *Computers in Human Behavior 63* (2016), 50–58. Accessed: 2022-05-28.

[22] SHARMA, S., SARKAR, D., AND GUPTA, D. Agile processes and methodologies: A conceptual study. *International journal on computer science and Engineering 4*, 5 (2012), 892. Accessed: 2022-10-18.

[23] STEFFEN, D. Game review: Tis-100. `https://www.diabolicalplots.com/game-review-tis-100/`, 2017. Accessed: 2022-09-12.

[24] TEAM, I. E. What are the 5 primary layers in software architecture? `https://www.indeed.com/career-advice/career-development/what-are-the-layers-in-software-architecture`, 2022. Accessed: 2022-10-19.

[25] TEKINBAŞ, K. Learning and games. *The ecology of games: connecting young, games and learning* (2008), 2–25. Accessed: 2022-05-28.

[26] WANG, L., AND CHEN, M. The effects of game strategy and preference-matching on flow experience and programming performance in game-based learning. *Interactive Learning Environments* (2010), 39–52. Accessed: 2022-05-17.

[27] WANG, T., MEI, W., LIN, S., CHIU, S., AND LIN, J. *Teaching programming concepts to high school students with Alice*. IEEE Frontiers in Education Conference, 2010. Accessed: 2022-05-20.

[28] XING, Y., HUANG, J., AND LAI, Y. *Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development*. Association for Computing Machinery, 2019. Accessed: 2022-10-14.

Appendix A

Diagram

A.1 UML Diagram

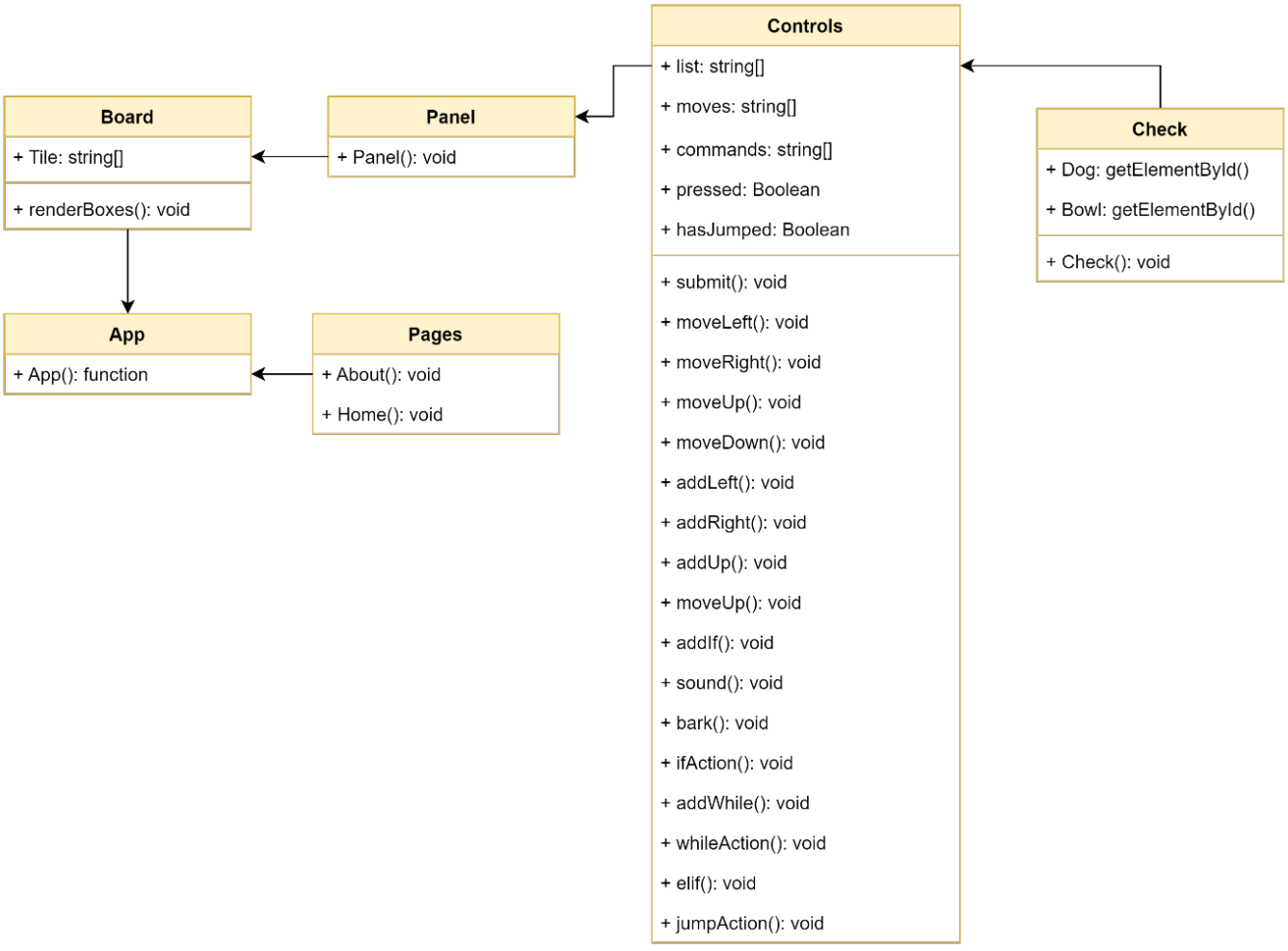

Figure A.1: UML Diagram

Appendix B

Implementation of levels

B.1 Figures of working levels

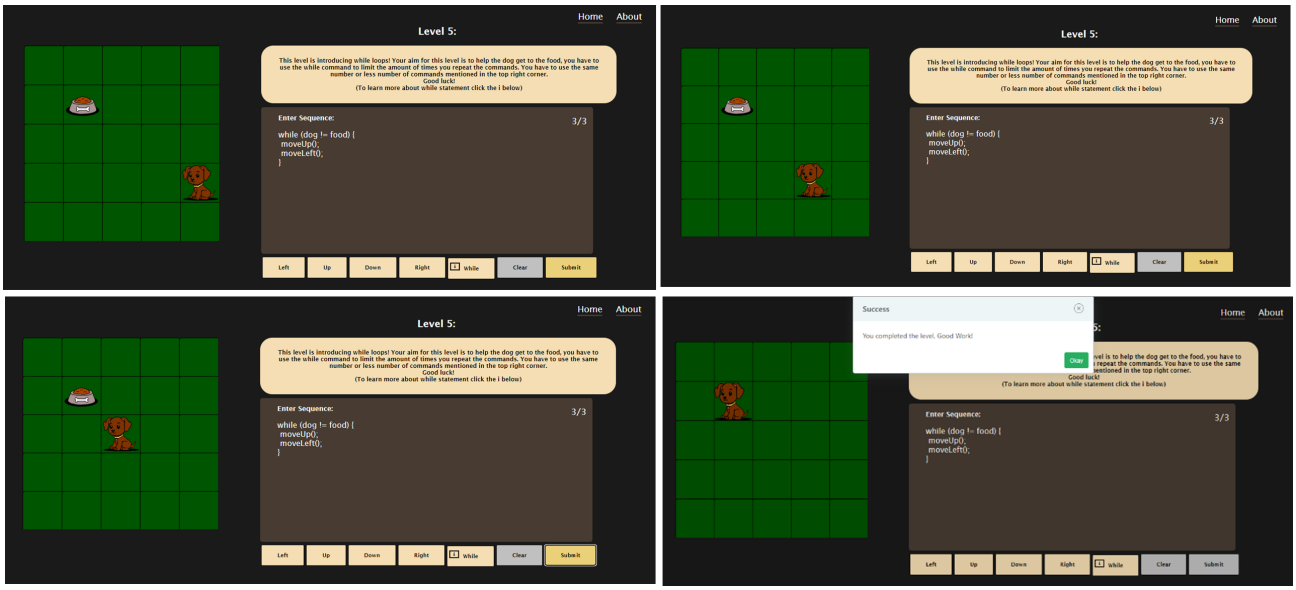

Figure B.1: Implmentation of a level using while loop

Figure B.2: Implementation of a level with a if statement

Figure B.3: Implantation of a level with variables

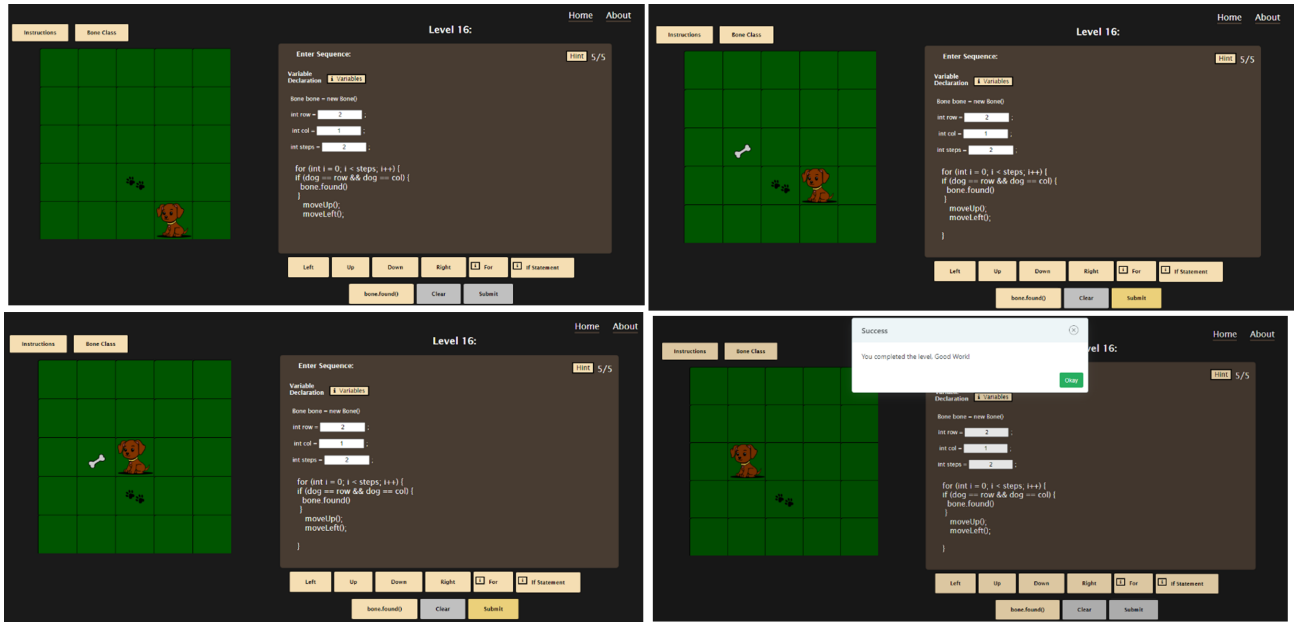

Figure B.4: Implementation of a level with variables and for loop

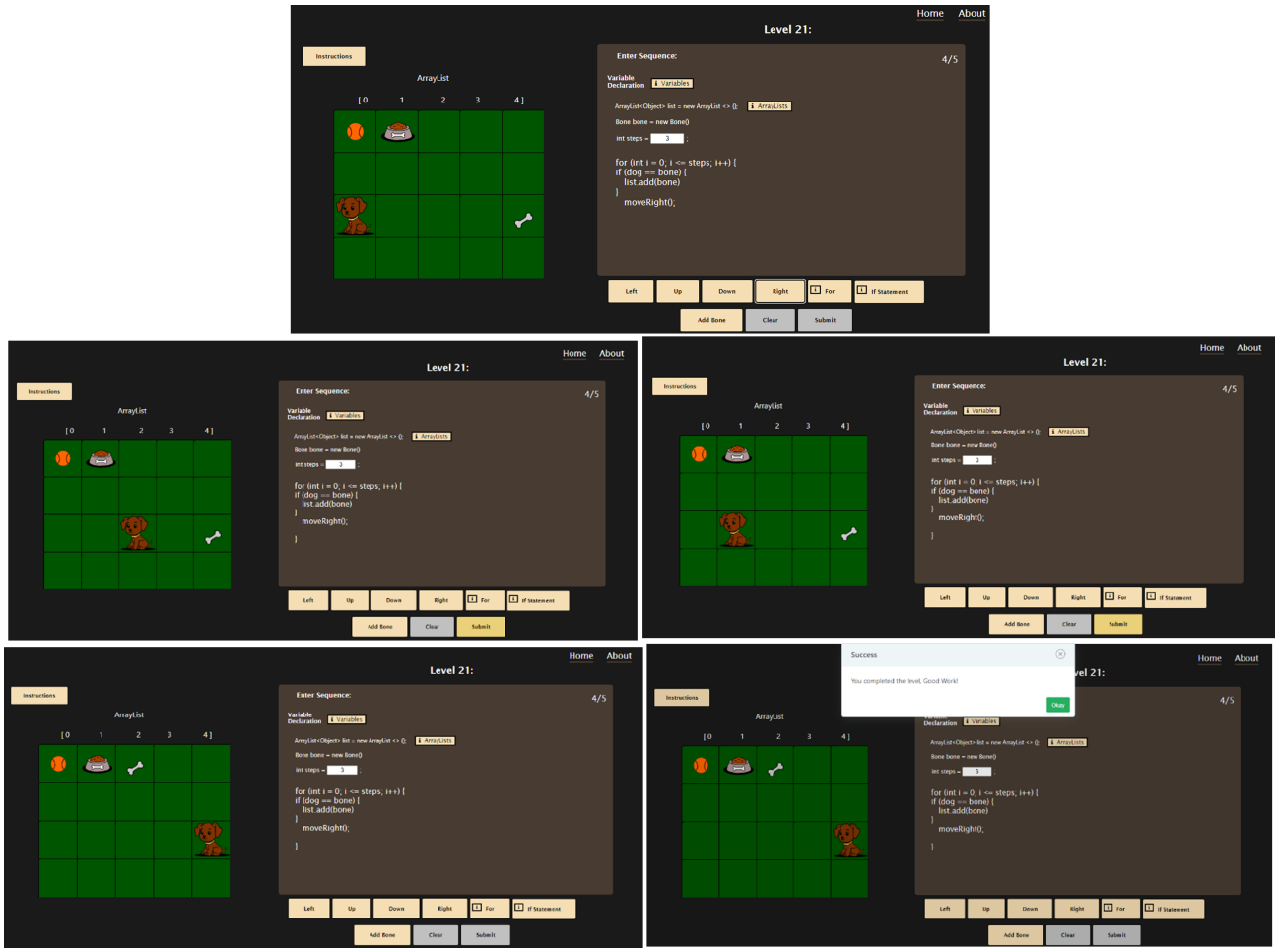

Figure B.5: Implementation of a level using collections

Appendix C

User Experience

C.1 User Testing Results

Question 1 – What they liked about the level?	**Question 2 – What they didn't like about the level?**	**Question 3 – Further Suggestions**
Good visuals	Some levels were easier than the previous levels	Making the board bigger as a 5x5 is a bit too easy
Response time was good	There weren't limits on some of the levels which confused some of the users on how to go about the levels.	Add limits to commands on every level to stay consistent.
The board got smaller and made it harder	There was a misunderstanding on the variables inputs field and the user wasn't sure what to put in it.	Add placeholders in the fields to let the user know what goes in them.
Users liked how different programming concepts were introduced.	Earlier levels didn't have extra information on the programming concepts being taught.	Add something to inform the user about the programming concept.
Liked how some levels combined what had already been taught to create a complex level.	Can't revert things to the previous state which can be annoying sometimes when you make small mistakes.	Maybe try adding states so users can redo and undo them.
Instructions on most levels were very well explained.	Misunderstood the instructions and started to look from the dog's point of view to figure out the row and column for the variables level.	Adding values on the board to help the user get the row and col values.
The information popup dialogues explain the programming concept.	Bugs found with for loops and the dog going off the board.	Fix the bugs with the for loops and fix boundary checks.
User-friendly interface	Spelling mistakes with some of the instructions	Fix spelling in instructions
Liked how the complexity kept going up	Hard to distinguish between the commands and the submit and clear	Make the buttons slightly different for accessibility

Table C.1: The similar feedback responses for all levels for each question