DEVELOPING A GAME THAT TEACH ALGORITHMIC THINKING FOR EDUCATION PURPOSES

by

Muhammad Shafri Syamsuddin (何伟光)

A Thesis

Submitted to the Faculty of Nanjing Xiaozhuang University

in Partial Fulfillment of the Requirements for the degree of

Bachelor of Software Engineering

School of Information Engineering

Fangshan, Nanjing

May 2022

To: Dean Xiangjun Zhao
School of Information Engineering

This thesis, written by Muhammad Shafri Syamsuddin, and entitled "Developing A Game That Teach Algorithmic Thinking for Education Purposes", having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

李青
Li Qing

徐晓昭
Xu XiaoZhao

倪震
Ni Zhen

Date of Defense: June 7, 2022

The thesis of Muhammad Shafri Syamsuddin is approved.

赵向军
Dean Xiangjun Zhao
School of Information Engineering

[illegible]
Zhao Juan
Chief of Foreign Affairs Office

Nanjing Xiaozhuang University, 2022

PERNYATAAN

Dengan ini saya menyatakan dalam skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaan di suatu perguruan tinggi dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain kecuali yang secara tertulis diacu dalam naskah ini dan diterbitkan dalam daftar pustaka.

Yogyakarta, 6 Juni 2022

Penulis,

[illegible]

METERAI TEMPEL

776AJX948369419

M. Shafri Syamsuddin

ACKNOWLEDGMENTS

The author would like to express the whole heart thanks to his family for financial supports in the author’s entire life and this pursue of bachelor’s degree in Nanjing Xiaozhuang University and Universitas Islam Indonesia.

The author would like to express an immeasurable appreciation for all the help and support that has been given, either directly and indirectly during the preparation for the completion of this thesis. In particular, to:

1. Certain someone who have been very impactful in these recent months to how I see life, to how I see motivations, to how I see belief, to how I overcome my depression, who accompany me talks about what is the purpose of life, how to take control and a lot of emotional supports. I am very fortunate that I met this human, this anomaly in my life that somehow saved me.
2. Myself, for not giving up.
3. Li Qing (李青) Laoshi as my advisors, who helped me from the beginning of the first draft till the completion of the thesis, he is such a nice guy with great attitude, great voice too.
4. Mrs. Sheila Nurul Huda as my favorite lecturer in University Islam Indonesia, Gotta catch em all, amirite?
5. Mrs. Sophie Mou , the best *cici* in Nanjing Xiaozhuang University, who always take jokes lightly , cheers up the mood in the university wechat groups, really talkative person on meetings, thank you for the companion for these 2 years online.
6. To my fellow classmates in both Nanjing Xiaozhuang University and Universitas Islam Indonesia.
7. Last but not least, to anyone, any people, that create any sort of content/education/philosophy that impact me in any way and make me a person that I am right now, thank you.

TABLE OF CONTENTS

List of tables

List of figures

Abstract

Owing to their ease of engagement and motivational nature, especially for younger age groups, games have been omnipresent in education since earliest times. More recently, computer video games have become widely used, particularly in secondary and tertiary education, to impart core knowledge in some subject areas and as an aid to attracting and retaining students.

Gaming community also have a niche group called speedrunners, where they compete to complete a game in the shortest way possible. Academics have proposed that game based learning to teach Algorithmic Thinking or Computational Thinking. And there are a lot of research suggesting that learning basic programming language such as pascal improves Algorithmic Thinking Ability of a person significantly.

Keywords : Computational Thinking, Algorithmic Thinking, Game, Education, Programming.

CHAPTER 1. INTRODUCTION

1.1 Background of The Study

Gaming is a term for an activity which the user run a specialized applications known as electronic games or video games on game consoles like X-Box, Playstation and Nintendo or on Personal Computers (in which case the activity is known as online gaming). The term “gaming” comes from a synonym for “gambling” although most electronic games today do not involve gambling in the traditional sense. Nowadays, gaming is a term that people uses regularly to suggest regular gameplay or possibly as a hobby. A person who is into gaming is often called a gamer or hardcore gamer.

On the other hand of the spectrum, we have education. Education is the process of which someone learns something; an acquisition of knowledge, skills, values, morals, beliefs, habits, and personal development.

Many research shows that video games have a negative impact on student grades. Because of the addiction that usually occurs when people play video games. Studies show that some children as young as 4 years old have become addicted. Video game addiction in children does not happen undetected. The negative effects of video game addiction are always apparent to the naked eyes of other people to see. Both parents and teachers may notice decreased performance at school coupled with lower grades and failing classes.

With that said, as with everything there is in the world, there are two sides to the object in question, video game apparently also have a positive impact. People that play games daily usually excels at problem solving and IQ test in general, although, it is still difficult to prove that video games are the one that cause the improvement, because there may be a bias that people who excels at problem solving and higher IQ are interested in playing video games because it makes their brain think more often and fast paced rather than learning school subjects.

I propose a solution which then we can monopolize on gaming addiction and use it as a tool to improve education. The solution is a game where the participants will learn how to think algorithmically, in hope that it will helps with students studying instead of having negative effects.

Algorithmic thinking is a way of thinking that every problem can be solved through the clear definition of the steps needed, rather than coming up with a single answer to a problem, we develop a set of instructions or rules that if followed precisely leads to answers to the original and similar problems. This way of thinking if applied correctly can improve on students ability to solve general problems.

This game is intended for students that are aware that they like to play games but want to improve their ability in academics.

1.2 Problem Statement

Based on the background, the author can identify the problem that needs to be solved by this application, those are:

a. How can a game counteracts it’s negative effects?
b. How to design a game that can helps in students studies without making it a boring game?
c. How to integrate a command-line like interface into a game?

1.3 Purpose of the Study

The purposes of this application are:

a. This application is a game that teaches students how to think algorithmically in hope that it helps with their studies.
b. This application help people use their addiction energy to gaming towards something more productive.

1.4 Limitation of Study

Some limitations of this study are as follows:

1. The application is only targeted at people who already like gaming.

2. The application can only be run on windows operating system.
3. The application will only be available in English.

1.5 Benefit of The Study

The benefits of the application are:

1. The application teaches participants how to think algorithmically and hopefully apply them in real life situations.
2. The application is designed with minimal graphics so participants can focus on the problem solving aspect of the game.
3. The application is lightweight so any computer with windows can run it.

1.6 Writing Structure

The thesis will be written with a this structure, divided into five chapters, description for each chapter are as follows:

CHAPTER 1. INTRODUCTION

In this chapter, the author discuses the background of the study, problem statement, limitation of the study, benefits of the study, writing structure and schedules.

` **CHAPTER 2. REVIEW OF LITERATURE**

This chapter discusses existing literature that supports the arguments of this paper, and used to elaborate the technology used in creating this paper

CHAPTER 3. ANALYSYS AND DESIGN

This chapter discusses the research methodology that is used to create this paper. The software development method used, the application wireframe, system needs, and system design.

CHAPTER 4. IMPLEMENTATION AND TESTING

This chapter discusses the results of the analysis and design of the chapter before. This chapter will explain the process of coding the application until it can be used by the users. It also illustrates the graphic of the system, and conducts limited testing of the application to see if it is already qualified or still in need of redesign or improvements.

CHAPTER 5. SUMMARY AND CONCLUSION

This chapter contains the conclusion to the paper, it talks about how theoretically the system achieve it goals, and talks about recommendations that can improve that for future developments.

1.7 Schedule

Scheduling for creating this thesis are as follows:

Table 1.1: Schedule

No	Activities	Weeks								
		1	2	3	4	5	6	7	8	9
1	Analysis of Needs	■								
2	Design	■	■							
3	Wireframe		■	■	■					
4	Coding					■	■	■	■	
5	Testing					■	■	■	■	
6	Documentation									■

It takes approximately 9 weeks for the writer to finish this paper, starting on February 22th, 2022 until April 30th, 2022.

CHAPTER 2. REVIEW OF LITERATURE

This review of literature will discuss about topics, theories and technologies that are used by the author during the development of the game:

2.1 Fundamental Theory

2.1.1 Algorithmic and Computational Thinking

Mind or machine, which trumps which? Debates about reasoning and self perfection, automation and political order did not begin with Big Data. Early modern natural philosophers, who developed mechanical calculating machines, embraced mathematical practice for the bettering of moral character (Ksenia Tatarchenko, 2019). Computational Thinking (CT) was first introduced by Papert (1990). Wing (2006) emphasized that CT is one of the daily life skills that everyone needs, rather than just being a programming skill used only by computer scientists. Wing (2010) further defined operational thinking as the process of problem-solving, so that the message processing agent can be effectively executed and the problem solved. Computers can help us solve problems via the following two steps: First, consider the steps to solve the problem, then use technical skills to control the computer to help solve the problem. For example, one must understand the mathematical formula and explain the problem, and use simple methods or formulas to solve the problem via the computer's computation. Besides , creating animation, the designer need to plan the way for the animation to be completed and have a good understanding of how the animation will flow before completing the task using compuer softwares. In these two examples, CT is the thinking process one needs to engage in before beginning the computer and machine operation [2][6][7][8][18][19].

CT is the fundamental skill for everyone, not just programmers. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking (Wing 2008). Wing (2008) stated that thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.

Computer science is the study of computation— what can be computed and how to compute it. CT thus has the following characteristics:

1. Conceptualizing, not programming.
2. Fundamental, not rote skill.
3. A way that humans, not computers, think.
4. Complements and combines mathematical and engineering thinking.
5. Ideas, not artifacts.
6. For everyone, everywhere (Wing 2008).

Heather Skinner Robin Croft (2009) found that Neurolistic Programming; a form of computational thinking, offers a more detailed approach to goal setting than other frameworks with which business and management students may already be familiar. Within this framework students are able to identify both motivation and means, and are then encouraged to take up the opportunity of making the first step towards successfully undertaking the dissertation project.

While Doleck, et al (2017) findings suggest a lack of association between computational thinking skills and academic performance (except for a link between cooperativity and academic performance). This is noteworthy given the importance that has been placed on teaching and learning twenty-first-century skills in various curricular reforms implemented since the turn of the millennium. If there is no relationship between computational thinking skills and academic performance, we must ask whether these curriculum-mandated skills are being explicitly taught at all. More distressingly, we must wonder at measures of academic performance that are negatively associated with cooperativity.

Other authors such as Tissenbaum, et al (2019) stated that a new way to look at CT is by converting it to Computational Action, they conclude that with rapid changes happening in both computing and computing education landscapes, we have an opportunity to reconsider how students learn computing. Young learners have the capacity to develop computational products that have authentic impact in their lives from the moment they begin to code. They simply need contexts that allow them to have such impact. Computational action starts to define what these contexts should look like. With more computing instructors coming online, we have a unique opportunity to work with them as they develop skills and practices necessary to engage in computational action with their students. We are excited about a world in which young learners see the world as full of opportunities for them to digitally create the future they (and we) want to inhabit.

There are also other forms of computational thinking suggested by Ross D. Arnold and Jon P. Wade (2105) called Systems Thinking. They described it as with most systems, systems thinking consists of three kinds of things: elements (in this case, characteristics), interconnections (the way these characteristics relate to and/or feed back into each other), and a function or purpose5 (Meadows 2008). Notably, the least obvious part of the system, its function or purpose, is often the most crucial determinant of the system's behavior5 (Meadows, 2008). Though not all systems have an obvious goal or objective, systems thinking does. In order to convey its definition, especially to those unfamiliar with the concept, it is critically important to communicate this goal.

2.1.2 Games on Improving Algorithmic and Computational Thinking

Although puzzle-based learning can be used as a fundamental strategy when developing an online learning system, it remains essential to have a proper strategy or mechanism that can further increase students' motivation and engagement when they are solving and practicing puzzles. Engaging students in repeatedly solving a puzzle not only ensures that they truly understand how to solve that puzzle but also helps students practice the essential skills of puzzle solving. Integrating game-based learning with puzzle-based learning to create a puzzle-based game learning system that could make puzzle solving identical to game playing may help to fulfill the above requirements of repetition and practice (Chih-Chao Hsu and Tzone-I. Wang 2017-2018) .

With the wide application of technology in education, digital games holding the inherent characteristics of goals, interaction, feedback, and entertainment have been given distinct educational purpose and developed to promote students' cognitive and intelligent abilities. However, it is not sufficient to simply transform puzzle-solving into the form of a puzzle-solving game that only possesses the inherent characteristics of digital games (Chih-Chao Hsu and Tzone-I. Wang 2017-2018).

Chih-Chao Hsu and Tzone-I. Wang (2017-2018) found that in all four puzzle-solving performances, the results of algorithmic thinking skills and the review of the pseudocodes were both reasonable. Using game mechanics, this study encouraged members of the PGM and PGS groups not only to try and solve a puzzle more times but also to try and solve more puzzles.

Other autors such as Kazigmolu, et al (2012) conduct experiments such as making their own game, they found that twenty five students participated in an exercise to evaluate program your robot and it was found that participants enjoyed playing the game. Furthermore, participants reported that this type of approach can enhance the problem solving abilities of students who are learning introductory computer programming.

Other orthodox way of this is what Topalli and Cagiltay (2018) do , instead of teaching CT using games , they conduct an experiment that teach CT by making games using scratch. They found that by slightly improving the course content through real-life game development in the Scratch environment, students' performance in the Senior-projects improved significantly which may be an indicator that the game projects improve students' motivation towards the introduction to programming concepts. Additionally, as they were able to see the big picture of the software development cycle and the place of coding in that cycle their understanding of programming concepts was most probably better conceptualized enabling them to better integrate the other concepts in their program within this big picture. The Scratch visual programming environment also most probably helped them not just concentrate on the syntax problems but on the design and development of the general system and algorithms. By solving a real-life problem and getting feedback from domain experts, they most likely are better motivated and work harder to solve the problem that is defined by the domain experts, in turn, improving their problem-solving skills.

2.1.3 Games Addiction

For the past few years , Online Games are really popular. Everyone plays it, child and adult alike. Game Addiction can be used as an advantage for this method of learning. However, making an education game addicting can be hard. Here are some of the findings with game addiction.

Dal Yong Jin and Florence Chee (2008) said that with the ever-increasing presence of online games in Korean mainstream culture, the corresponding consequences of games eclipsing other activities have also garnered much attention in recent years to become a very pertinent issue. Gamers in Korea have repeatedly made world headlines with reports on their perceived level of pathological use of games. The controversies have largely revolved around the compromise of “real-life” social activities because of their addictions to all things having to do with games, at home, and especially at PC bang. This acceptance into mainstream popular culture is evident in many ways, such as the success of games such as StarCraft and Lineage, the existence of celebrity professional gamers, and organized league play of online games, which are often broadcast on cable television (Kline et al., 2003). As D. Lee (2006) noted, the broader populace is however only starting to realize that gaming itself is not just for trivial fun but has become another channel of human relationships, in other words, part of people’s actual lives.

The section “part of people’s actual lives” is the critical part, games nowadays have taken over some people life, because it is where most people can use their knowledge in a niche community. Being good at a specific something have a really deep impact on psychology, usually we will feel proud that we can do something that not everybody can (Chappel et al ,2006).

2.2 Theoretical Basis

2.2.1 Windows Operating System

Microsoft Windows, usually called Windows, is an operating system made by Microsoft. It is a graphical operating system shell for MS-DOS in response to the growing interest in graphical user interfaces. Microsoft Windows allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet.

2.2.2 Unity 3D

Unity Technologies is a company that is created on 2004 by David Helgason, Nicholas Francis and Joachim Ante. This game engine is made because three of them care about indie game developers that can’t afford to buy an expensive game engine. The mission of this company is to make a game engine that can be used by everyone.

Unity 3D is a great cross-platform 3D game engine and a user friendly development environment. It uses C# to handle code and scripts, in hand with a number of classes and APIs. Just like other game engine, Unity is capable of providing many of the most important features to make a game work. It already includes physics, 3D rendering, and collision detection. From a developer’s eye, this means that they do not need to code all of that anymore. Rather than starting a new project just to create physics, Unity 3D already provide it.

CHAPTER 3. ANALSYIS AND DESIGN

3.1 Research Methodology

3.1.1 Introduction

The aim of this game is to educate people on how to think algorithmically using a serious game that can teach people how to think ahead of time and how well that type of thinking can aid in problem solving every aspect of life.

As teaching algorithmic thinking have a direct effect on improving someone problem solving skills, it would possibly improve their grades in school and therefore decrease the problem that games usually brought; scoring low in academic tests.

3.1.2 Software Development Method

The development process used in this project is something that every game studio follows to create a good game. The process is iterative, meaning that features can be prototyped over and over.

However because limitation of time, the author of this paper decided to implement Rapid Application Development (RAD) mindset to this development method.

The development method is divided into 5 phases:

1. **Game Design:** Here is where most of the prototyping or designing of the game takes place, the developer creates a game by writing a document called Game Design Document (GDD) which is a document that every developers in the team need to follow for the rest of the project so that the game created will be what the game designers want exactly. The result of this step is usually some drawings of the game or maybe a prototype.
2. **Implementation:** This step is also called production, and is the main stage of the development process. The game assets and source code are created. In this part we can divide the team into three major groups, the first are the programmers, they will create the mechanic of the game without graphics for rapid testing, the second are graphic designers, they usually create the 3D assets of the game, the last are sound engineers, they are obliged to create and search for sound effects

that are suitable for the game. The product of this stage is called an alpha version of the game. It contains all the major feature of the game, only some minor feature will be added or removed later.

3. **Beta phase:** This version is where the game is already complete, the only part left to do is bug-fixing by getting feedback from users or beta testers. No new feature is added, only bug-fixing.
4. **Post production:** In the post production the game is released to public.
5. **Maintenance:** As all software that are created in this world, game also needs a maintenance phase after it's released, in order to keep up with new devices or fixing bugs that are not discovered on the beta phase of the development.

In this paper, the author only work alone, he will take all the roles, because of this limitation, the game will only be made until the beta phase, it will not be released nor get maintenance.

3.2 Project Planning

The project plan will discuss about the main objectives of the project, specifications and/or responsibilities. As mentioned, this plan will include everything involving the development of the game, but it can still be altered in the future with reviews, evaluations and feedbacks.

These are the project plans:

3.2.1 Requirements

The main project requirements for the author is to learn how to design and develops a video game. All the phases inside it like preproduction, production and post production. The author also need to learn how to make 3D assets, compose simple music for the game, scripting, level creation and user interface.

3.2.2 Specifications

Because the project will be tested on the author computer, the specifications will be focused on minimum specifications to run the software needed to create this projects. The software are:

1. Unity 3D
2. Blender
3. Visual Studio 2019
4. Audacity

All these software run smoothly on the author laptop , Acer swift sf314-54g , with the technical specs:

Table 2: Specifications

1.8 GHz Intel Core I5-8250U
8 GB 2666 MHz DDR4
Nvidia MX150
15,4” monitor with 1920*1080 resolution
NVME SSD 256GB Storage
Windows 11 Home Single Language 64 bit

3.2.3 Project Scope

The project will be based on creating a sokoban-like 3D game with instructions as a mean of controlling the player rather than direct inputs, the in-scope features are:

- Single Player
- Windows
- 3D Low Poly Art
- Multi Level
- 3D sokoban-like game with instruction as a way to control the player
- Minimalist GUI

- Unity3D using C# as programming language

3.3 Game Design Document

The game design document (GDD) is the core of the game. The designer create this in extreme details to explain how the game play mechanics play and what the game world based on. This document is a reference for all the people in the team to work with and to create the game. In this instance, since the author work on the game alone, this serves purpose as a guideline so the time spent developing is more efficient.

3.3.1 Introduction

Escape from Area 51 is a sokoban-like puzzle game for windows developed in Unity3D with 3D low-poly art. The player will control the character through a lot of levels avoiding spikes and holes by giving it a set of instructions that it will follows, just like programming!

Beat every level and set the character free from the area 51

3.3.2 Background

Escape from Area 51 is inspired by two existing game that are already available. These games are called *sokoban* and *baba is you*, both are the same genre with that of *Escape from Area 51*.

This game is inspired by these titles because they are essentially a grid-based movement style of game, by restricting the movement of characters to a grid, the instructions to move the character will be more understandable and fluid. *Baba is You* is also inspiring this game but in a different way, in that title, the game teaches player how to use logic equation to solve the puzzle, by teaching logical thinking to player, that game can make someone better at problem solving too, this game is inspired by that but took another approach, by teaching algorithmic thinking.

3.3.3 Character

In this game there will only be 1 character , the main character, which is:

- Robot : This robot is a very strong robot, it can push blocks, how much? Infinite, but it’s not very smart, you need to give it instructions, detailed instructions, to help him get out of the maze!

3.3.4 Gameplay

The gameplay is sokoban-like puzzle with 3D style, the player controls the main character from the start of the level until the exit. You need to push boxes, avoid spikes, block lasers and navigate through rotating platforms to get to the exit. The player control the character by giving it movement instructions that the character will follow.

The player mechanic will be a very specific instruction as follows :

1. Turn Left
   The Robot will rotate 90 degree counter-clockwise.
2. Turn Right
   The Robot will rotate 90 degree clockwise.
3. Move Forward
   The Robot will move to the block in front of it.

The game is composed of 3 main stages where the mechanics will be introduced one by one:

- Stage 1 : Here the player will be introduced to the basic mechanics of the game , controlling the player , learning to navigate through the level and pushing boxes.
- Stage 2 : The player will be introduced to lasers , they will learn that the character will die if the lasers touches them , they will learn that they can push blocks to interrupt the lasers so that they can make it until the exit without touching the lasers.
- Stage 3 : Rotating platform will be introduced in this stage , if the character touches the yellow square on the middle of the platform , the

platform will rotates , so the player need to think more of the sequences of their instructions.

CHAPTER 4. IMPLEMENTATION & TESTING

The implementation phase is where the author try to develop the game according to the GDD previously made and the conception of the game in the author mind. After reading and watching courses about the tools that are needed for the development of the games. The first step is to create some 3D models and then start programming with them. The graphic section is not taking up too much time, so the author think he should start with it.

Then the author will create the basic mechanics of the game such as movement, death, respawn, etcetera. After the mechanics are created in a test level, the author will start designing the levels and create a fully playable game.

4.1 Graphics and Animations

The graphic in this game is made by using Blender, a 3D modelling app, the models that are created as follows:

1. The Robot

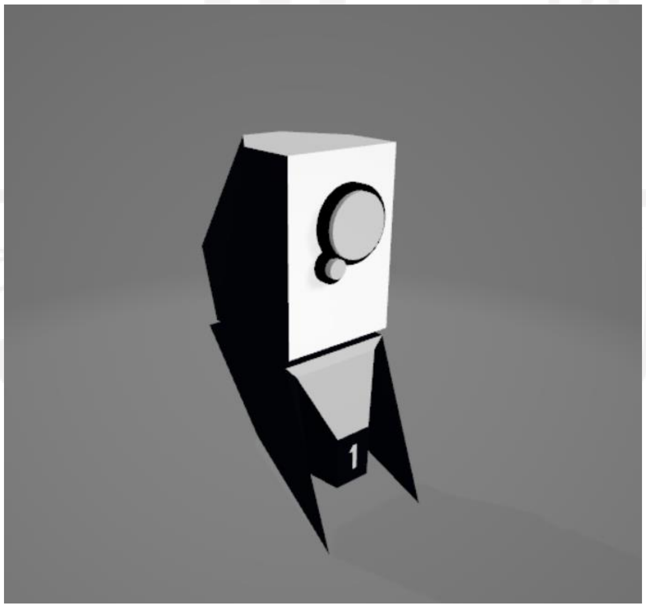


Figure 1: The Robot

This is the 3D model of the robot, the character which the player will give instructions to, the model is inspired by a game called *portal*. There is a camera like robot that will shoot the player in that game, the design of this game robot is mimicking that of the *portal* game. The robot doesn't have a strafing function, instead it can only turn right or left in place and move forward. The design of the robot also implies that it can only turn in place and move forward. This way the control is a bit different from other *sokoban-like* games.

2. The Spikes

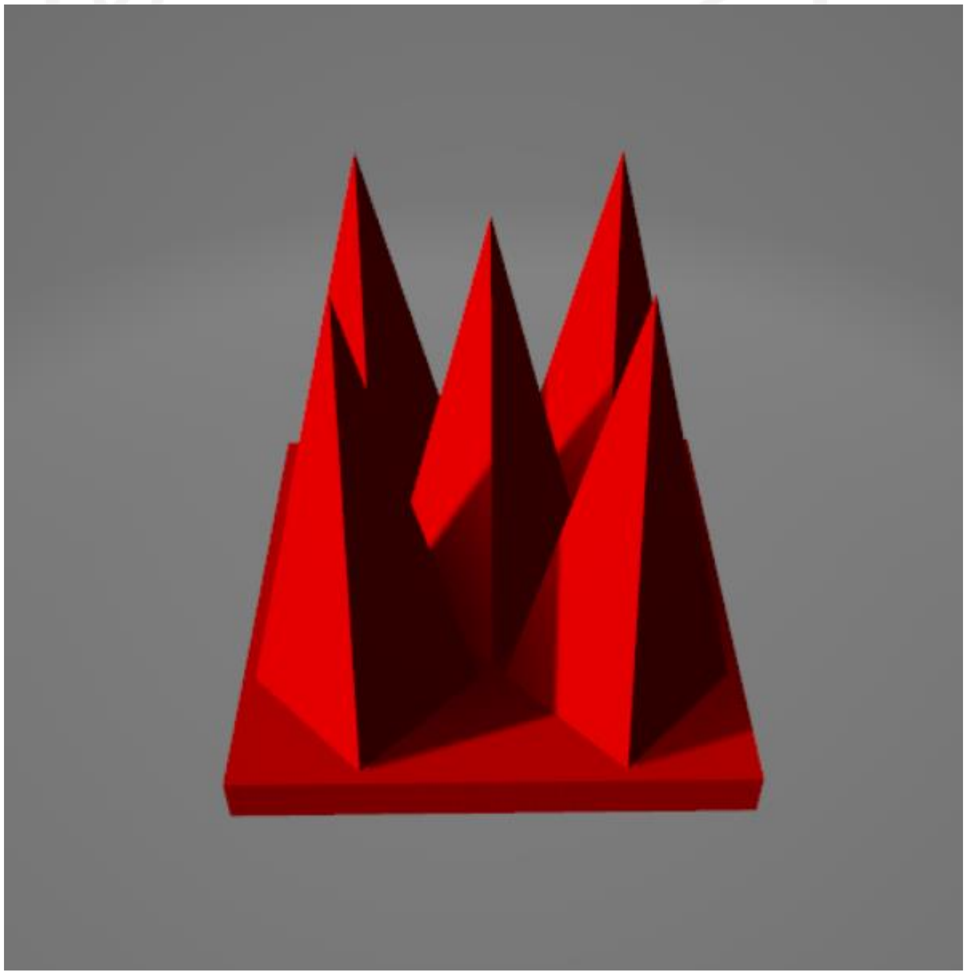


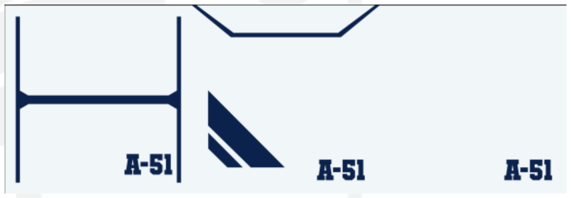
Figure 2: The Spike

This is the spike 3D model , the robot will die and respawn if it touches the spike. It is made with a simple square at the bottom and 5 protruding pyramids at the top of it to mimic a characteristic of a spike, the square bottom allows the author to easily fit it in the *sokoban-like* grid system in the game.

The spike also have a Collider detection, it is a Unity3D built-in feature that allows developer to assign a detector on any 3D model, if another 3D model enters the area it will give away a Boolean signal, this signal then can be used to execute certain functions, developer can also assign specific 3D model that will trigger the function, in this case the author make it so that the Collider only detects the robot, if the robot enters the collider it will send a true signal to the robot that it touches the spike, this will make the robot send another signal to the game to activate the respawn function that will reset the entire level to its original state.

3. The box

The box doesn't have a 3D model that is made with blender, instead I use Unity Built-in feature of wrapping a 3D box with 2D texture. The way unity does it is that we provide the unity with a 2D picture that have 6 faces. Here's what it looks like, the box can be pushed by player, dropped to the spike and block laser beam.



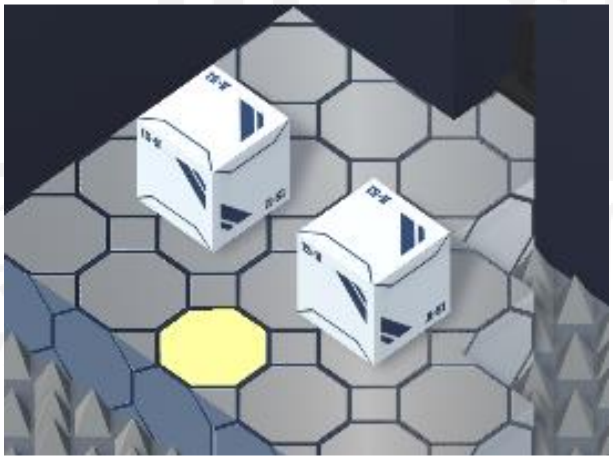


Figure 3: The Box

4. The Rotating Platform

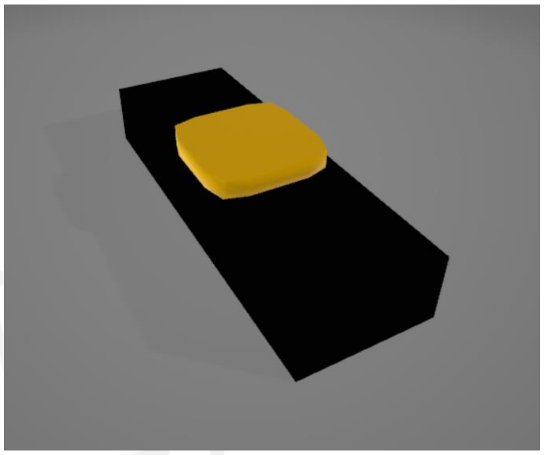


Figure 4: The Rotating Platform

This is the rotating platform, It will rotate by 90 degree if the robot touches the yellow button in the middle of the platform. If the robot wants to turn it again, it must leave the button and click it again. How it works is that there is a Collider in the center of the button, it detects if there are another 3D Rigid Body that enter its area, it will send a true or false signal, that then can be used by a script to determine whether or not the platform rotates.

5. The Laser

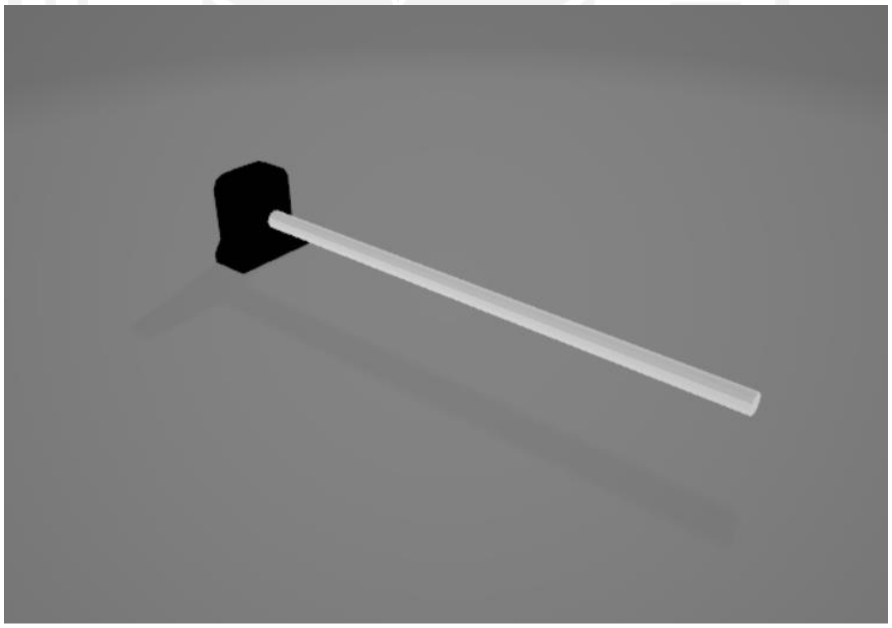


Figure 5: The Laser

This is the laser, the black part of the laser is the only 3D model that is used in the laser, the white part is for demonstration purposes, the actual laser in the game later will be made using Unity3D Particle System, the black part of the 3D model will be assigned with the Particle System, this then will shoot a particle constantly that it will look like a laser, it also contains a Collider much like the spike, so if the robot touches it the entire level will be reset to its original state.

All this 3D models will later look a bit different in game because of texture and lighting applied on it.

4.2 Programming

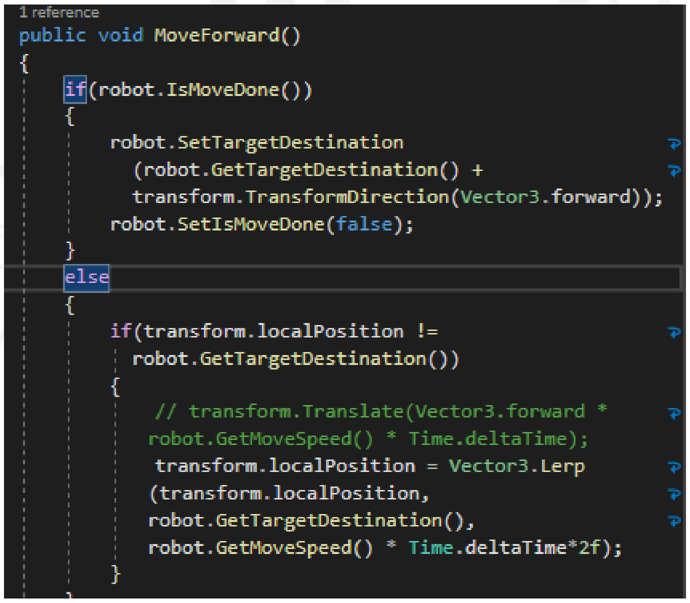
The programming part have been done using Unity3D and Visual Studio Community 2019. Unity have a large community filled with forums, hence the project was done with helps from various of people online. Another advantage of using Unity is that it is an environment with useful tools and libraries that make it easier to start into the game, for example, developer doesn’t need to create basic functions like gravity, movement, lighting etc. The game engine already does it for the developer.

The programming of the game was divided into several sections:

1. Character Movements

This part is to create and control the behavior of The Robot. The actions are Moving Forward and Turning in Place.

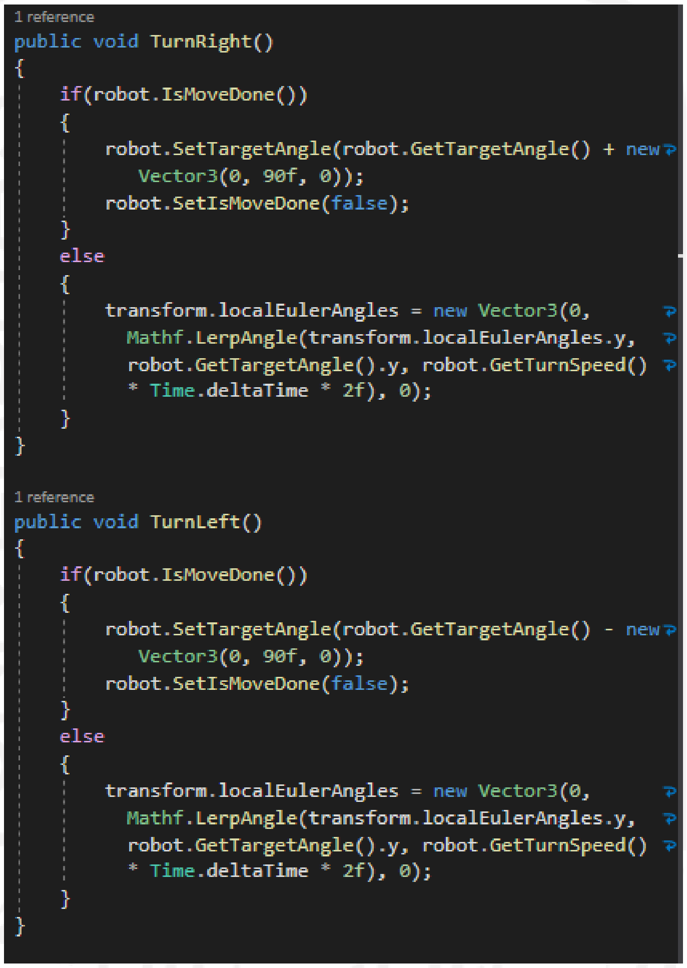
- Move Forward: This mechanic is made by taking the position of the robot and adding movement speed to it, transform() is a Unity3D basic function, basically it will change the position of the robot every 2 frame of the game moving it according to how much movement speed is assigned on it , then checking if it’s already in its destination position yet, if it’s already in its destination , the movement will be stopped.



```
1 reference
public void MoveForward()
{
    if(robot.IsMoveDone())
    {
        robot.SetTargetDestination
          (robot.GetTargetDestination() +
          transform.TransformDirection(Vector3.forward));
        robot.SetIsMoveDone(false);
    }
    else
    {
        if(transform.localPosition !=
          robot.GetTargetDestination())
        {
            // transform.Translate(Vector3.forward *
           robot.GetMoveSpeed() * Time.deltaTime);
            transform.localPosition = Vector3.Lerp
           (transform.localPosition,
           robot.GetTargetDestination(),
           robot.GetMoveSpeed() * Time.deltaTime*2f);
        }
```

Figure 6: Character Movement Code

- Turning in place: In here, there are two functions, turn left and turn right, it works the same way as move forward, also using the same transform() function but instead of changing the X and Y position of the robot, this change the rotation of the robot, every time TurnRight() or TurnLeft() is called, the robot will turn 90 degree clockwise or counter-clockwise.



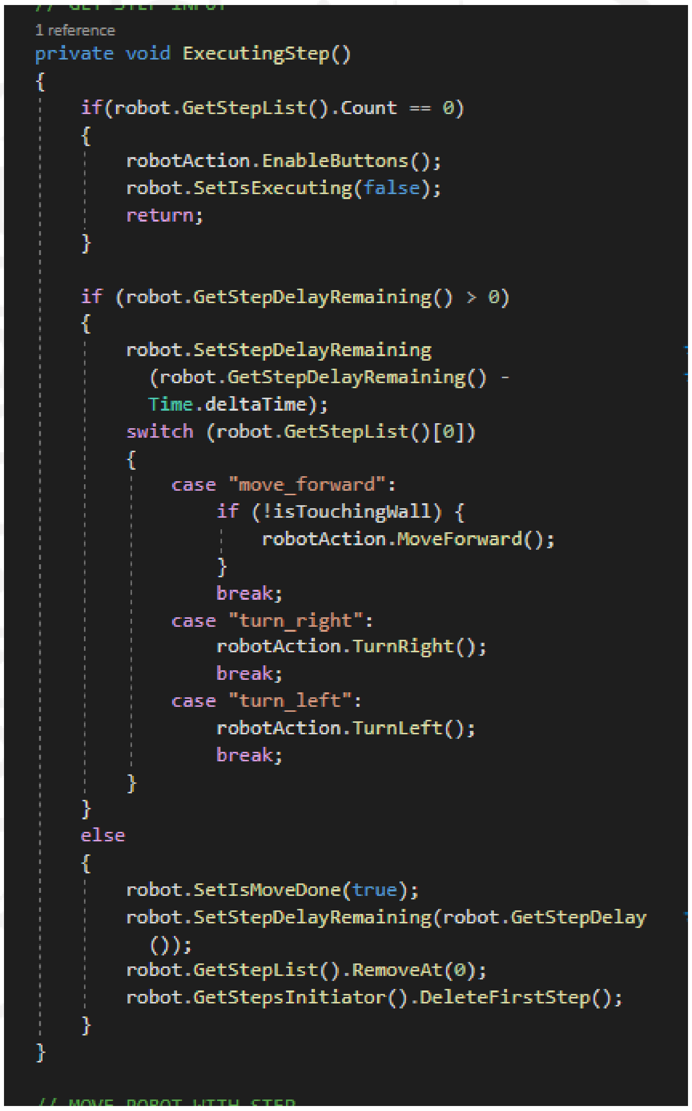
```
1 reference
public void TurnRight()
{
    if(robot.IsMoveDone())
    {
        robot.SetTargetAngle(robot.GetTargetAngle() + new
           Vector3(0, 90f, 0));
        robot.SetIsMoveDone(false);
    }
    else
    {
        transform.localEulerAngles = new Vector3(0,
          Mathf.LerpAngle(transform.localEulerAngles.y,
          robot.GetTargetAngle().y, robot.GetTurnSpeed()
          * Time.deltaTime * 2f), 0);
    }
}

1 reference
public void TurnLeft()
{
    if(robot.IsMoveDone())
    {
        robot.SetTargetAngle(robot.GetTargetAngle() - new
           Vector3(0, 90f, 0));
        robot.SetIsMoveDone(false);
    }
    else
    {
        transform.localEulerAngles = new Vector3(0,
          Mathf.LerpAngle(transform.localEulerAngles.y,
          robot.GetTargetAngle().y, robot.GetTurnSpeed()
          * Time.deltaTime * 2f), 0);
    }
}
```

Figure 7: Character Movement Code 2

2. Character Control

This part is where the player controls the movements of the character, the GDD says that it will be an instruction-based movements, the way the author programmed this is, by having a set of instructions stored in an array, with minimum of 5 instructions and 14 maximums, then a function will read the arraylist and then deleting it one by one as it calls the movement functions



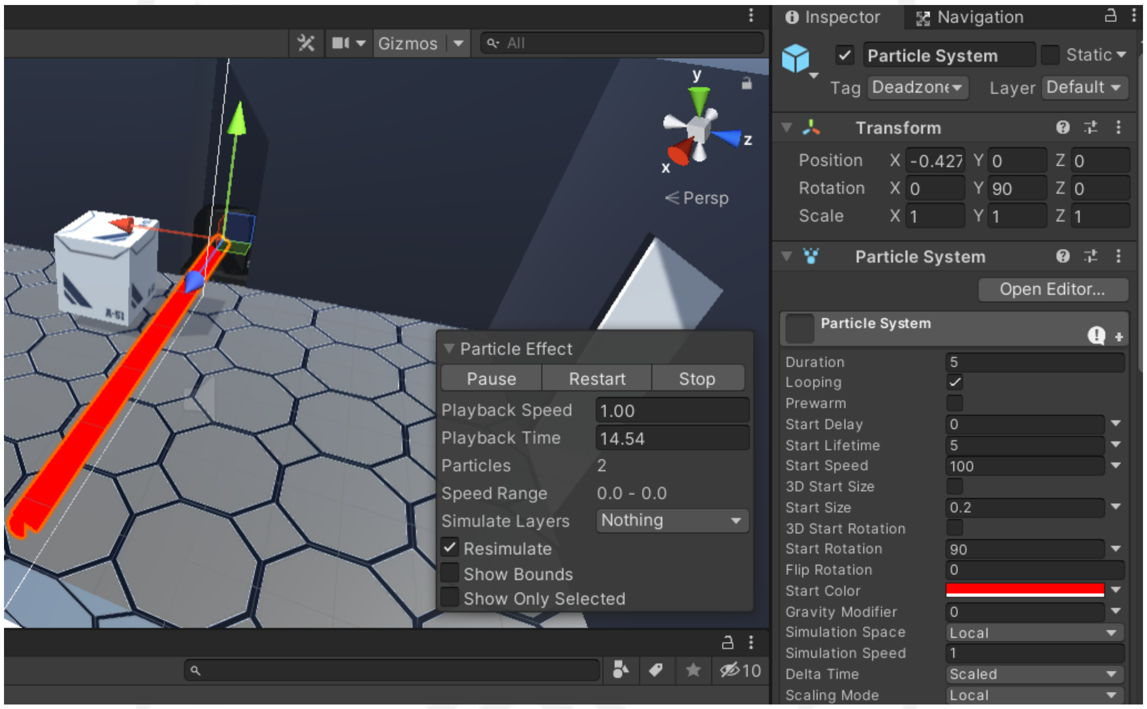
```
1 reference
private void ExecutingStep()
{
    if(robot.GetStepList().Count == 0)
    {
        robotAction.EnableButtons();
        robot.SetIsExecuting(false);
        return;
    }

    if (robot.GetStepDelayRemaining() > 0)
    {
        robot.SetStepDelayRemaining
          (robot.GetStepDelayRemaining() -
          Time.deltaTime);
        switch (robot.GetStepList()[0])
        {
            case "move_forward":
                if (!isTouchingWall) {
                    robotAction.MoveForward();
                }
                break;
            case "turn_right":
                robotAction.TurnRight();
                break;
            case "turn_left":
                robotAction.TurnLeft();
                break;
        }
    }
    else
    {
        robot.SetIsMoveDone(true);
        robot.SetStepDelayRemaining(robot.GetStepDelay
          ());
        robot.GetStepList().RemoveAt(0);
        robot.GetStepsInitiator().DeleteFirstStep();
    }
}
```

Figure 8: Character Control Code

3. Laser

The laser is not coded at all, because there is a built-in feature called Particle System in Unity, Developer can use this feature to create laser, then by tagging the laser as a “Deadzone”, when the robot collide with it, we can run the respawn function, the particle system also have the added benefit of being blockable by 3D objects without coding , so when the player pushes the block into the laser, it will automatically be blocked.



Particle Effect
Pause Restart Stop
Playback Speed 1.00
Playback Time 14.54
Particles 2
Speed Range 0.0 - 0.0
Simulate Layers Nothing
✓ Resimulate
Show Bounds
Show Only Selected

✓ Particle System Static
Tag Deadzone Layer Default

Transform
Position X -0.427 Y 0 Z 0
Rotation X 0 Y 90 Z 0
Scale X 1 Y 1 Z 1

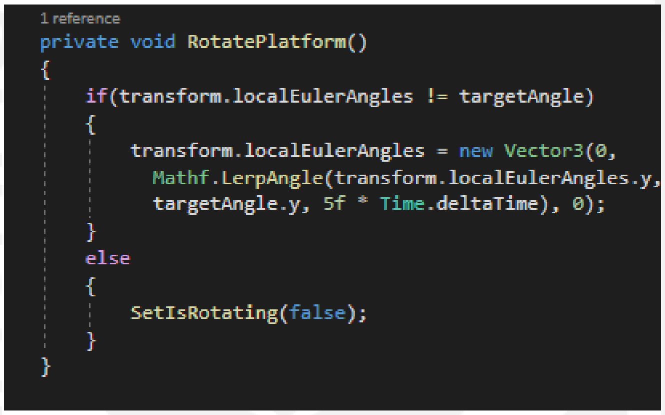
Particle System

Duration	5
Looping	✓
Prewarm	
Start Delay	0
Start Lifetime	5
Start Speed	100
3D Start Size	
Start Size	0.2
3D Start Rotation	
Start Rotation	90
Flip Rotation	0
Start Color	🟥
Gravity Modifier	0
Simulation Space	Local
Simulation Speed	1
Delta Time	Scaled
Scaling Mode	Local

Figure 9: The Laser Particle System

4. Rotating Platform

The rotating platform mechanic is the same mechanic as the one that turn the robot, the difference is the trigger mechanism, it will rotates every time the robot enters the yellow button collision zone in the middle of the rotating platform. This uses Unity3D built-in function transform(), using this we just insert a new vector angle that we want, and then executing transform every 2 frame to get the desired rotation which is 90 degree clockwise rotation.



```
1 reference
private void RotatePlatform()
{
    if(transform.localEulerAngles != targetAngle)
    {
        transform.localEulerAngles = new Vector3(0,
          Mathf.LerpAngle(transform.localEulerAngles.y,
          targetAngle.y, 5f * Time.deltaTime), 0);
    }
    else
    {
        SetIsRotating(false);
    }
}
```

Figure 10: Rotate Platform Code

5. Trigger Collision

Trigger Collision is the most important aspect in any game programming, it controls and decide what happens when objects touch another object. In this case the only collision trigger needed is on the Robot, when the robot touches certain objects that are tagged in certain ways, it reacts differently, such as:

A. Deadzone

   The player will trigger the respawn function, this will reset the whole level to its original state.

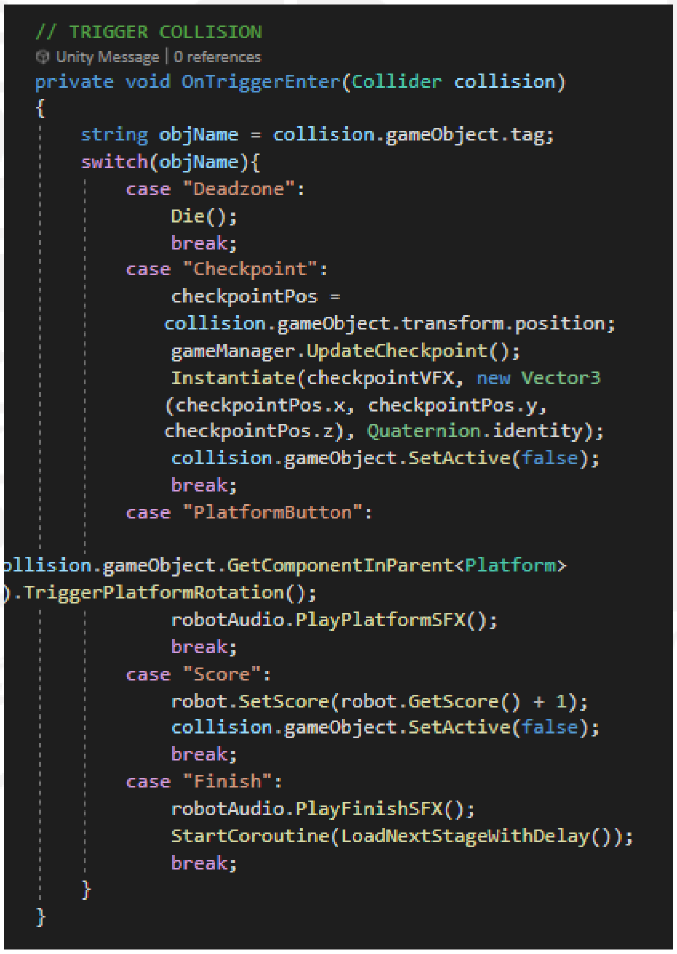
B. Checkpoint

   The player will trigger the update checkpoint function, this will change the original state of the level into the current state of the level, so when the robot touches the “Deadzone” the whole level will not reset.

We can see there collision.gameObject.transform.position it update the checkpoint position to where the collision is happening, we can also see that it then disable the checkpoint using collision.gameObject.SetActive(false) so that if the robot somehow touches the checkpoint again we avoid any error that might happens.

C. PlatformButton

Here when the robot collides with the orange button on the middle of the rotating platform, it will call the Rotate Platform function



```
// TRIGGER COLLISION
Unity Message | 0 references
private void OnTriggerEnter(Collider collision)
{
    string objName = collision.gameObject.tag;
    switch(objName){
        case "Deadzone":
            Die();
            break;
        case "Checkpoint":
            checkpointPos =
           collision.gameObject.transform.position;
            gameManager.UpdateCheckpoint();
            Instantiate(checkpointVFX, new Vector3
           (checkpointPos.x, checkpointPos.y,
           checkpointPos.z), Quaternion.identity);
            collision.gameObject.SetActive(false);
            break;
        case "PlatformButton":

ollision.gameObject.GetComponentInParent<Platform>
).TriggerPlatformRotation();
            robotAudio.PlayPlatformSFX();
            break;
        case "Score":
            robot.SetScore(robot.GetScore() + 1);
            collision.gameObject.SetActive(false);
            break;
        case "Finish":
            robotAudio.PlayFinishSFX();
            StartCoroutine(LoadNextStageWithDelay());
            break;
    }
}
```

Figure 11: Trigger Collision Code

6. General User Interface

   Last but not least was the programming of the GUI. There is a HUD that shows the controls of the player and the instructions button, All these elements are inside a Canvas and controller. There is also a Pause Menu that allows the player to reset, restart the level, quit the game or go back to the main menu. Unity3D implemented a UI system since 4.6 that makes really easy and fast to create those menus. The function onClick() allows the button to execute any function of a script attached to an object.

4.3 Testing

Testing is the most important part of the game development process. By testing the game developers can discover certain unknown bugs or if one of the feature is not as good as the player expected.

These are the picture of the tested mechanics or functions in the game:

1. Main Menu

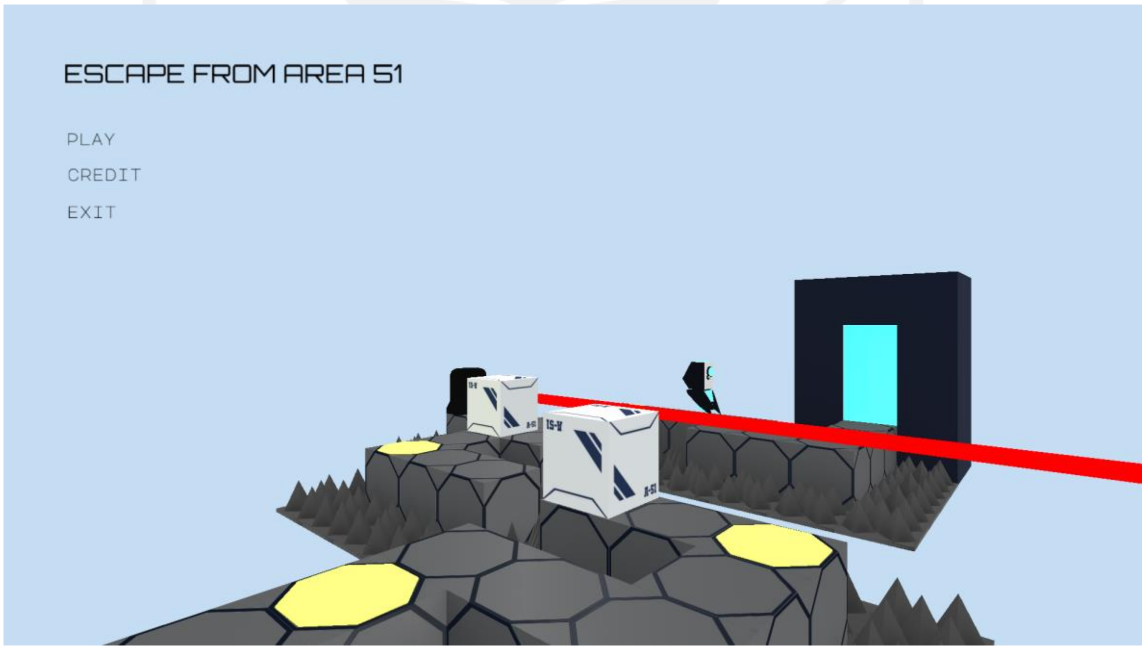


Figure 12: Main Menu

Main Menu is a landing interface that the user will encounter at the start of the game, It consists of 3 buttons: Play, Credit and Exit. When the player clicked on the Play button they will be redirected to the 1st stage of the game where they will play the game, Credit will take them to credit for the game, and Exit button will make them quit the game. The background of this Main Menu is not an image either , rather a live 3D model that is created for the Main Menu. There are no bugs found here.

2. Stage 1-1

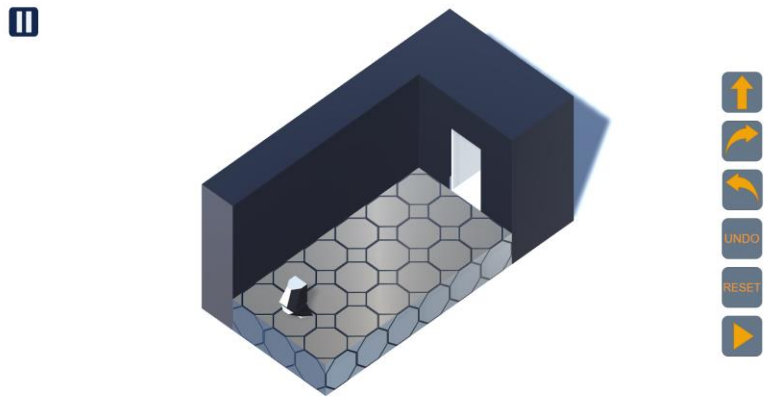


Figure 13: Stage 1-1

This is stage 1-1; this is included in the test part to see whether the user can enter a stage successfully or will there be errors. The result is that the player can enter the stage without any problem. We can also see that the instruction button is showing up perfectly.

3. Creating the instructions

`In here we can see the player already made a set of instructions for the robot to follow. All the buttons work perfectly and the inputs are recorded without a problem.

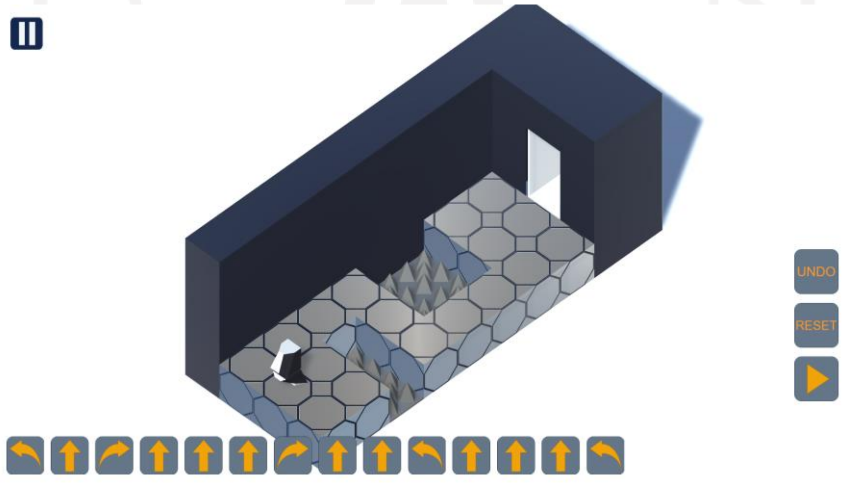


Figure 14: Instruction Demo

4. Executing the Instructions

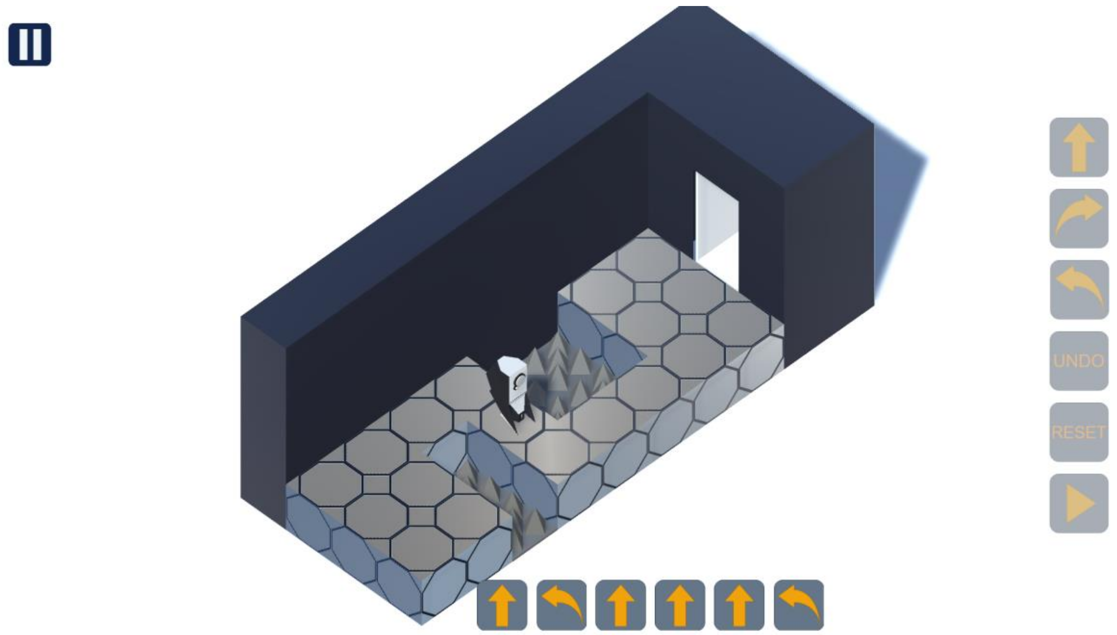


Figure 15: Executed Instructions

Here we can see that the robot followed the player instruction perfectly as inputted , move forward , turn left , move forward , turn right , move forward and so on. It will then stop after it does all the instructions.

5. Pause menu

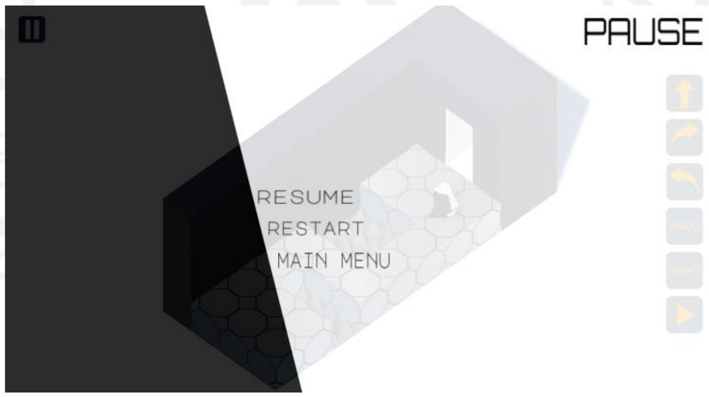


Figure 16: Pause Menu

When the player click the pause button on the top left side of the screen , this pause menu will show up , player can then resume to exit the pause menu , restart to reset the stage to the original state , or go to main menu if the player decided to stop playing the game.

6. The laser

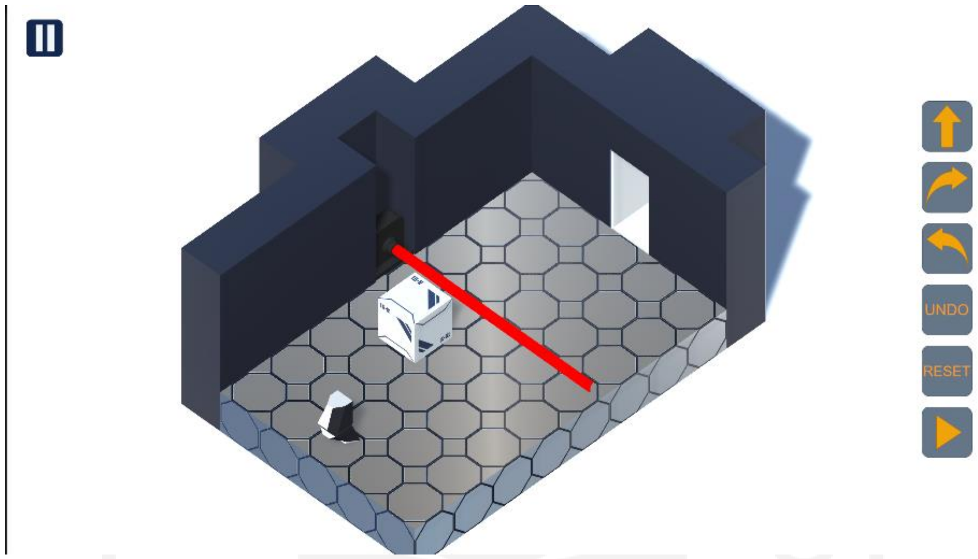


Figure 17: The Laser

We can see that the laser particle work as intended, it fires out of the device until it hit a wall, in this case it hit an invisible wall.

7. The Laser (Blocked)

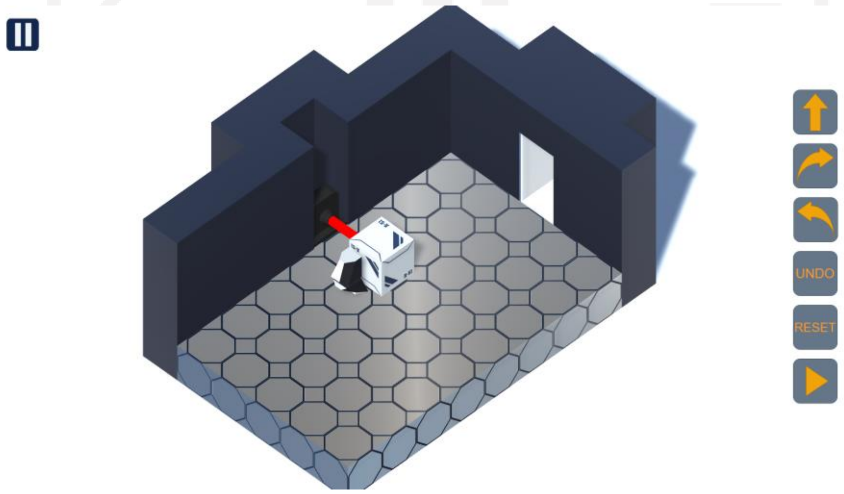


Figure 18: Blocked Laser

The laser should stop if it hit other object or rigidbody beside the wall because it is a particle system, in this case the object is the box , we can see that the laser doesn't go through the box that have been pushed into position by the robot.

8. The Robot dies from Touching the Laser

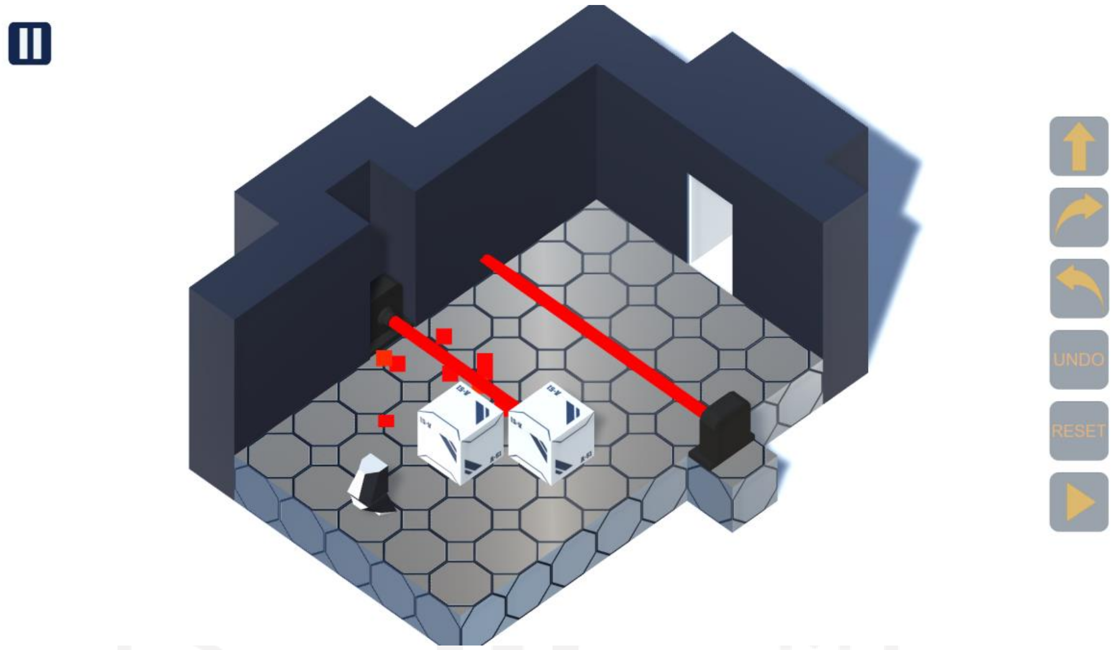


Figure 19: Robot Dies Demo

In this particular screenshot, the robot is purposely given instruction so that we can test the laser , as seen in the picture there are particles comping out from the laser, that is the robot death animation particle effect, the respawn also happened, so we see the robot is already on it's original state but we still see the death effect on the laser.

9. Rotating platform

   In here , we test the mechanic of the rotating platform , the rotating platform should rotate 90 degree clockwise if the robot trigger the collider in the middle of the platform

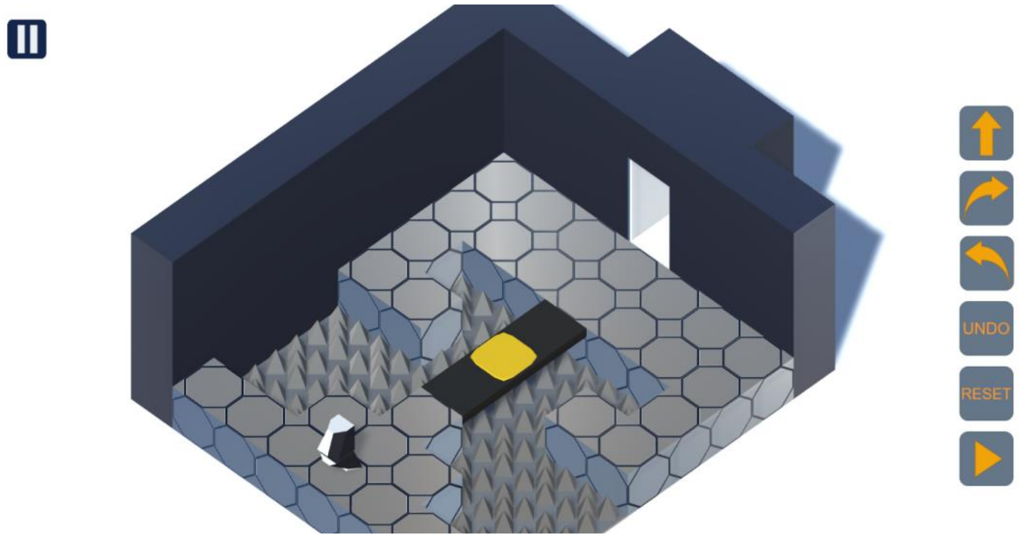


Figure 20: Platform

10. Rotating Platform (Rotated)

    After the robot touches the middle of the platform or trigger the collider, the platform rotates 90 degree clockwise as expected.

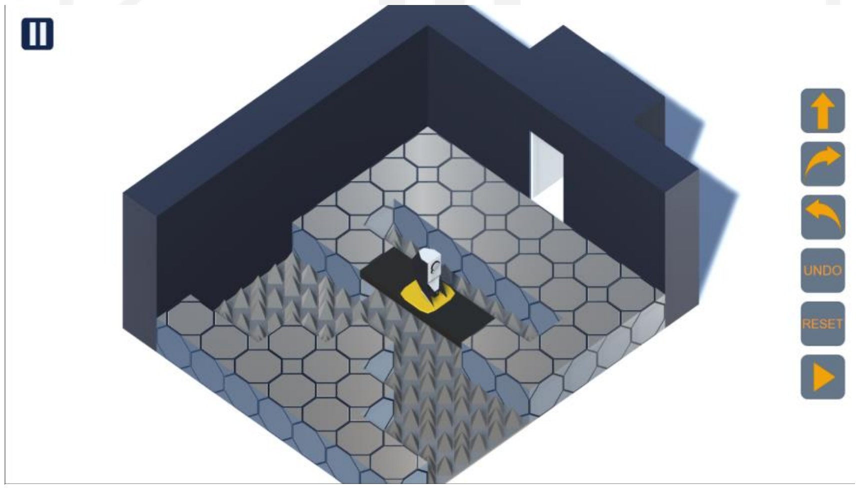


Figure 21: Rotated Platform

Those are the major functions, that are already tested. The testing went well without any bugs discovered.

4.4 Stage Testing

In this section, the author will explain important stages and how does each stages teach the player to think algorithmically and the challenges that comes with it. What the designer of the level intended the level to be.

4.4.1 Introducing Movement

First, we need to make sure the player understand the control of the game, so we give the player a basic maneuvering challenge like in the stage 1-2 , this will give the player time to train the controls without thinking that they are being told. Player will likely make mistakes at first and then learn that falling into the spikes kill the robot.

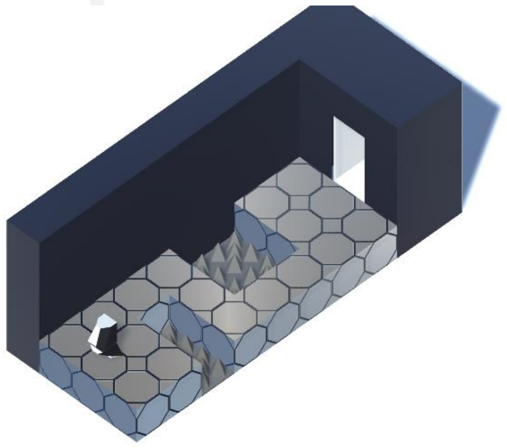


Figure 22: Stage 1-2

Then, after we make sure that the player learned about the basic maneuver of the robot , we can introduce them to the boxes.

Boxes is just a 3D square with *rigidbody*, meaning that it can be pushed around by another rigidbody; another box or the robot itself. It can also fall and cover spikes on the ground, that way the player can walk above the boxes itself.

In sokoban like game usually there is an unspoken-rule that , boxes that are already pushed onto a wall , cannot be recovered anymore because the main character or in our case the robot can't pull boxes. This can be an addition to the level designing as well. Like in level 1-4 the level is designed in a way that the player need to think creatively if they don't want to be blocked by their own boxes.

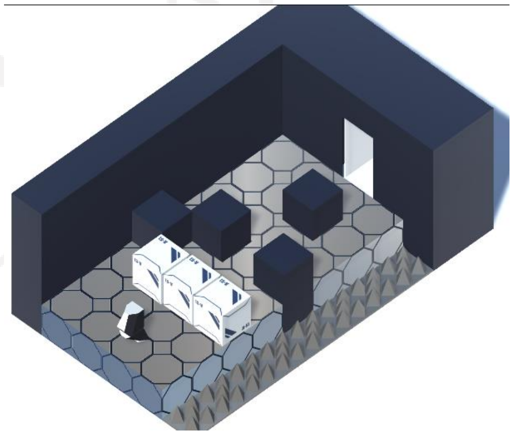


Figure 23: Stage 1-4

The player is expected to be competent at the basic mechanic of the game now. We can introduce the player to more advanced mechanic after this.

4.4.2 Introducing Lasers

In here we will start introducing lasers to the player, we need to make sure that the player knows what a laser does, so the 1st level of this stage, level 2-1 will only consist of a box, and a laser, to teach the player that they need to block the laser with a box to progress through the level. This will be implemented more like in level 2-3, it's still the same idea of blocking lasers with boxes but with more lasers, boxes and spikes.

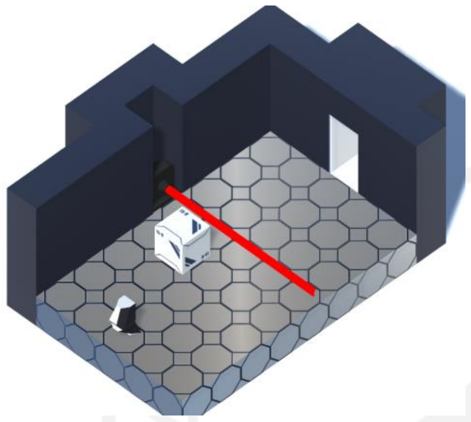


Figure 24: Stage 2-1

In that level the player needs to construct some sort of blocking stair out of the boxes to navigate through the lasers safely, in this instance the player will then need to think in advance on how do they position the cubes.

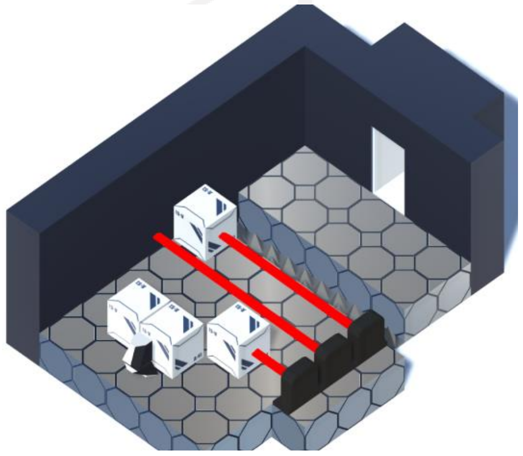


Figure 25: Stage 2-3

After the player complete this stage, we expect that the player already understands the mechanic of using the boxes for blocking lasers.

4.4.3 Rotating Platforms

Basically, the same principle as we did before, we introduce a single platform on the first level of the stage, the unique thing about the platform is that when it rotates, the robot won't rotate with it, so the player needs to take that into consideration before making instructions for the robot. We also introduce a new mechanic called a checkpoint block here, the function is that if the robot dies after it already touches the checkpoint block, it will respawn on the checkpoint block instead of restarting the whole level again.

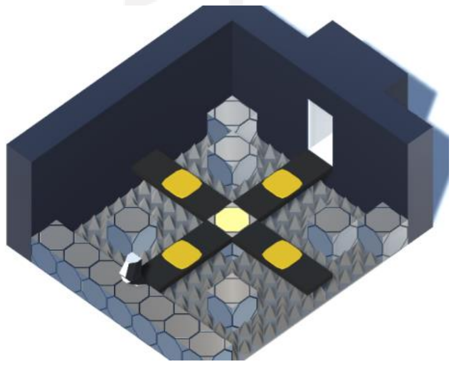


Figure 26: Stage 3-3

CHAPTER 5. CONCLUSION

5.1 Conclusion

Developing a game is not an easy task, as mentioned it is an iterative process where when making the game it is not constantly like that, sometimes there are some mechanics that are unrealistic to make without helps from other, features that are not as good as theorized when made. Being a team of one person also means that the author had to do all the roles in a normal game development settings.

Planning on how to make the game is a hard part for the author because he need to define milestones and tasks without knowing much if certain task consumes more time than another certain tasks. Thus, some aspects of the GDD is not very realistic for the time needed to do all the things planned. Clearly the time was worst when doing the project due to lack of experience as a student, also the mental state of the author at the time of making. Such as no repetitive function to teach recursion, no logical function to teach If…else , etc.

The application is completed now and hopefully satisfy the purpose of it. Teaching how to do algorithmic thinking, with this game the players need to think ahead of time before acting, and set out certain instructions to the robot.

5.2 Recommendations

Like said in the conclusion, the author can't make some feature that are pretty crucial for teaching algorithmic thinking. And there are no tests done to see if real people actually learn how to do algorithmic thinking by playing this game. Some recommendation from the author in order to make this game better can be seen:

1. Logical and Recursion function can be added to further improve how to think algorithmically.
2. Simulating sensors like in a real robot can also be done so that the player can create his own AI.
3. The plan for the game originally was to create a compiler so that the player can create his own code for the robot, however the author haven't finished this part yet, so he did the instructions approach instead.

4. Testing can be done to see if game actually improve algorithmic thinking on students.

REFERENCES

1. Topalli D. & Cagiltay N.E., Improving programming skills in engineering education through problem-based game projects with Scratch, *Computers & Education (2018), doi: 10.1016/j.compedu.2018.01.011.*

2. Hsu T.-C., Chang S.-C. & Hung Y.-T., How to learn and how to teach computational thinking: Suggestions based on a review of the literature, *Computers & Education (2018), doi: 10.1016/j.compedu.2018.07.004.*

3. Michael Hemmingsen (2020): Code is Law: Subversion and Collective Knowledge in the Ethos of Video Game Speedrunning, Sport, Ethics and Philosophy, DOI: 10.1080/17511321.2020.1796773.

4. Heather Skinner Robin Croft, (2009),"Neuro-linguistic programming techniques to improve the self-efficacy of undergraduate dissertation students", *Journal of Applied Research in Higher Education, Vol. 1 Iss 1 pp. 30 – 38.*

5. Hsu C.-C. & Wang T.-I., Applying game mechanics and student-generated questions to an online puzzle-based game learning system to promote algorithmic thinking skills, *Computers & Education (2018), doi: 10.1016/j.compedu.2018.02.002.*

6. Louise Amoore (2019) Introduction: Thinking with Algorithms: Cognition and Computation in the Work of N. Katherine Hayles, *DOI: 10.1177/0263276418818884*

7. Ksenia Tatarchenko (2019) : Thinking Algorithmically: From Cold War Computer Science to the Socialist Information Culture. *Historical Studies in the Natural Sciences, Vol. 49, Number 2, pps. 194–225. ISSN 1939-1811, electronic ISSN 1939-182X. © 2019 by the Regents of the University of California.*

8. Doleck, T., Bazelais, P., Lemay, D. J., Sazena, A. & Basnet, R. B. (2017) : Algorithmic thinking, cooperativity, creativity, critical thinking, and problem

solving: exploring the relationship between computational thinking skills and academic performance.

9. Kazimoglu, C., Kiernan, M., Bacon, L. & Mackinnon, L. (2012) : A serious game for developing computational thinking and learning introductory computer programming.

10. Fokker, J.D., Egges, A. & Toll , W.V. (2019) : Learning C# by Programming Games.

11. Patrick Jagoda (2018) : On Difficulty in Video Games: Mechanics, Interpretation, Affect.

12. Dal Yong Jin & Florence Chee (2008) : A Critical Interpretation of the Korean Online Game Industry

13. Chappell, D., Eatough, V., Davie, M. N. O. & Griffiths, M. (2006) : EverQuest—It_s Just a Computer Game Right? An Interpretative Phenomenological Analysis of Online Gaming Addiction

14. Carley et al. (2018) : Digital Education to Limit Salt in the Home (DELISH) Program Improves Knowledge, Self-Efficacy, and Behaviors Among Children. *Journal of Nutrition Education and Behavior, Volume 50, Number 6, 2018*.

15. Catherine So-Kum Tang, Yee Woen Koh, & YiQun Gan. (2017) : Addiction to Internet Use, Online Gaming, and Online Social Networking Among Young Adults in China, Singapore, and the United States. *Asia Pacific Journal of Public Health 1-10.*

16. Eijnden, R. V. D., Koning, I., Doornwaard, S., Gurp, V. P. & Bogt, T. T. (2018) : The impact of heavy and disordered use of games and social media on adolescents' psychological, social, and school functioning. *Journal of Behavioral Addictions 7(3), pp. 697–706 (2018) DOI: 10.1556/2006.7.2018.65*

17. Isabella Kotini and Sofi a Tzelepi (2015) : A Gamification-Based Framework for Developing Learning Activities of Computational Thinking. *T. Reiners, L.C.*

Wood (eds.), Gamifi cation in Education and Business, DOI 10.1007/978-3-319-10208-5_12.

18. Jeannette M. Wing (2006) : Computational Thinking. *Communications of The ACM, March 2006/Vol. 49, No. 3.*
19. Mike Tissenbaum, Josh Sheldon, & Hal Abelson (2019) : From Computational Thinking to Computational Action. *Communications of The ACM, March 2019, Vol. 62, No. 3*
20. Ross D. Arnold & Jon P. Wade (2015) : A Definition of Systems Thinking: A Systems Approach. *Procedia Computer Science 44 (2015) 669 – 678*
21. Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CS0 course using scratch. *Journal of Computing Sciences in Colleges, 26(3), 19-27.*