LONDON METROPOLITAN UNIVERSITY

ITAHARI INTERNATIONAL COLLEGE

Module Code & Module Title

FC6P01NT - Final Year Project

Assessment Weightage & Type

40% FYP Final Report

Year and Semester

2021 Autumn

Student Name: Nischal Rai

College ID: 19031772

Assignment Due Date: April 27th, 2022

Assignment Submission Date: April 27th, 2022

Internal Supervisor: Mr. Santosh Parajuli

External Supervisor: Mr. Rubin Thapa

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Acknowledgement

Working on the Final Year Project has given me the opportunity to learn about various technical terms and processes involved in the development of a complete system. It has aided me in the development of both my creative and technical abilities. I'd like to thank IIC College for giving me the opportunity to learn new skills.

I would also like to express my heartfelt appreciation to my project supervisor, Mr. Santosh Parajuli and Mr. Rudin Thapa for guiding me throughout the project. They inspired and encouraged me to excel in all aspects of the project. They provided me with all of the guidance and assistance I required to complete the project. Being under their supervision has been a wonderful experience.

Abstract

Almost all youngsters play video games, and children who play too many video games become distracted from their studies. It's not like video games are the cause of the problem; even when there were no video games, youngsters would always procrastinate by watching television or engaging in other enjoyable activities. This project of an educational game is introduced to overcome this problem. Conan: The Wonder Kid is the title of this educational game. As part of its educational component, this game features Newton's law of motion and basic physics. This educational game is appropriate for children in grades 6 and up. This project is for every solo game creator because it is a solo game development project.

Table of Contents

Table of figures

Table of Tables

Chapter 1 Introduction

1. Introduction

1.1.Project Description

Game is a structured form of play that is usually played for fun or entertainment and is sometimes used as an educational tool. Games differ from work, which is usually paid, and art, which is often an expression of aesthetic or ideological elements (Ansari, 2011). According to Accenture, the total value of the gaming industry has surpassed the combined markets for movies, music, and movies. The report, which is based on data collected by 4,000 gamers in four of the largest gaming markets, explores how the industry is shaping up and how it will continue to grow.

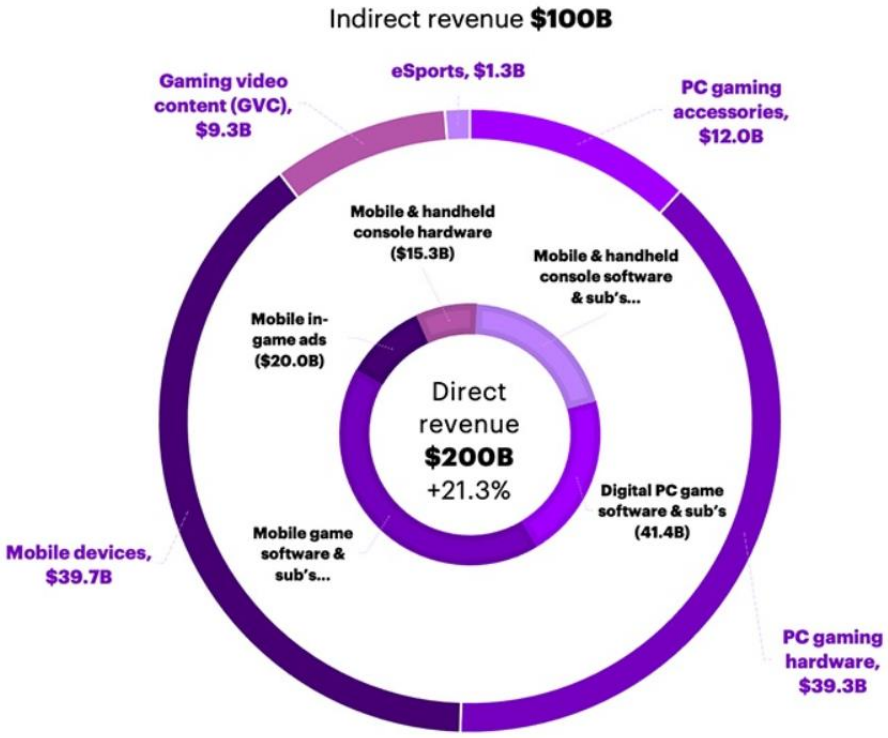


Figure 1: Current scenario of the game marketing

The gaming industry has expanded by half a billion gamers in the previous three years, reaching 2.7 billion individuals globally. According to the estimate, more than 400 million additional gamers are expected by the end of 2023. The demographics of these arrivals are changing as well: 60% are women, 30% are under the age of 25, and one-third are non-white. Long-term gamers, on

the other hand, are 61% male, 79 percent above the age of 25, and 76% Caucasian (Market Report Analysis, 2020).

The social side of gaming is becoming an increasingly important aspect of gamers' overall experiences as the gaming community grows. 84 percent of respondents believe video games help them connect with others who share their interests, and three-quarters acknowledge that gaming platforms now account for more of their social contacts. The data for the study came from an online poll of 4,000 people who spend at least four hours a week playing video games. China, Japan, the United States, and the United Kingdom are evenly represented in the sample. Over a dozen in-depth interviews with gaming executives from Activision Blizzard, EA, Everton, Niantic, Razer, Square Enix, Samsung, Splash Damage, and Tencent were also done (Wiley, 2020).

Educational games are either educational games designed specifically for the aim of education or entertainment games with educational value. Educational games are designed to help people learn new concepts, gain domain knowledge, and improve their problem-solving skills. Almost half of all youngsters spend more than 10 hours each day online, and by the age of twenty, they will have logged more than 30,000 hours of gaming time. Rather than trying to distract children from the technology that is such an integral part of their lives, many educators have learned to embrace high-tech methods of education, such as educational video games, game-based learning, and "blended learning" (the process of combining both technology and more traditional methods of teaching). According to a 2013 research, games can help students boost their grades by two grades. Co-op is the way to go. Children's results increase by two standard deviations when they play together, according to research on motivation.

1.2. Current Scenario

1.2.1. World

Video games are a different form of entertainment in that they encourage players to become characters in the story. Although video games have been around for more than 30 years, today's sophisticated options necessitate players paying close attention to the game at all times. Players engage on a more physical and emotional level than people watching a movie or television. According to a 2018 survey, two-thirds of US teens aged 13 and up considered themselves gamers. According to 2017 data, children aged 12 to 15 play video games for approximately 12.2 hours per week. For older teenagers, the figure is much higher (Jacquemet, 2019).

Many psychologists and scientists believe that playing video games has some advantages, especially in terms of teaching higher-level and abstract thinking skills. Playing video games alters the physical structure of the brain in the same way that learning to play the piano or read a map alters the brain. The brain is a muscle that can be strengthened through exercise. When playing games, the combination of concentration and neurotransmitter surges helps to strengthen neural circuits, giving the brain a real workout (Jacquemet, 2019).

1.2.2. Nepal

Video games have a significant impact in Nepal, just as they do in other countries. The number of people who play video games is growing by the day. People earn money for video games by participating in video game tournaments and streaming video games.

Revenue on Game:

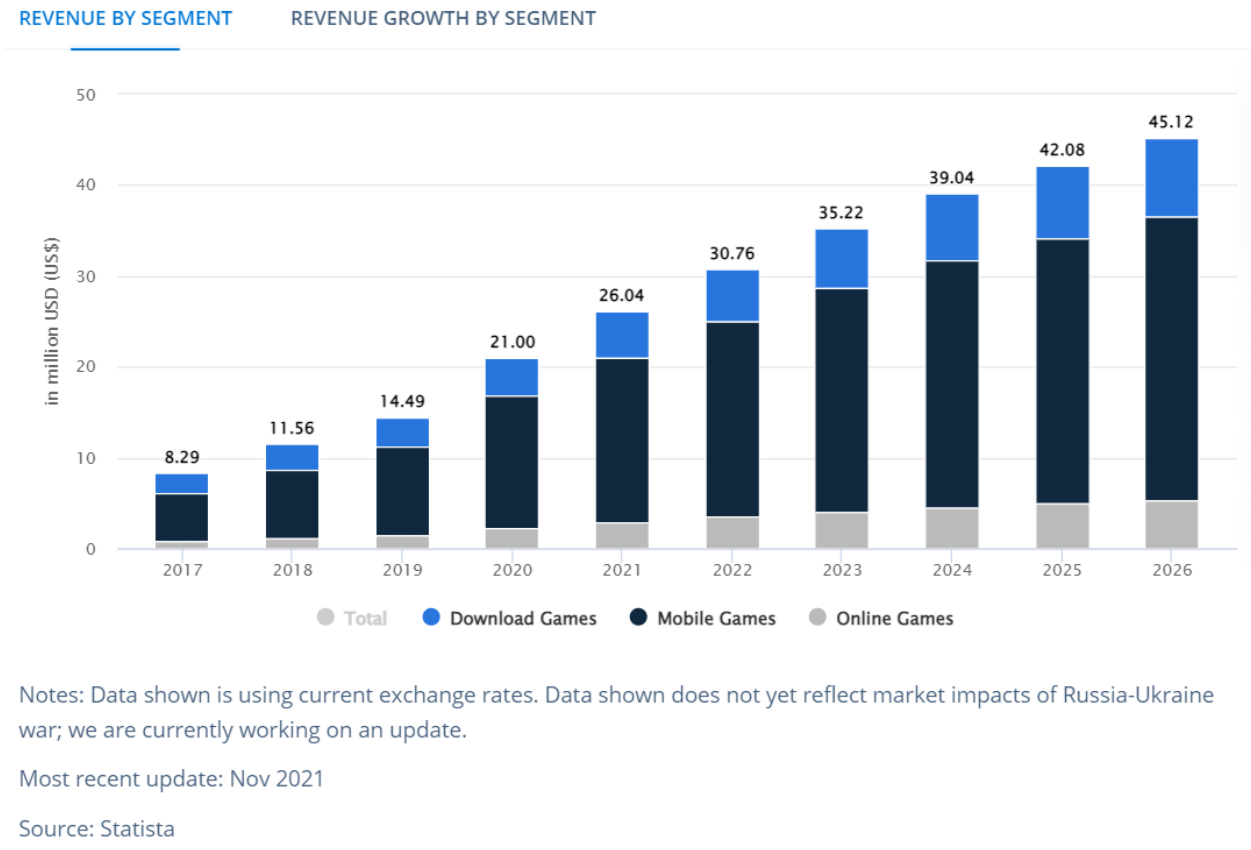


Figure 2: Revenue on game by Segment

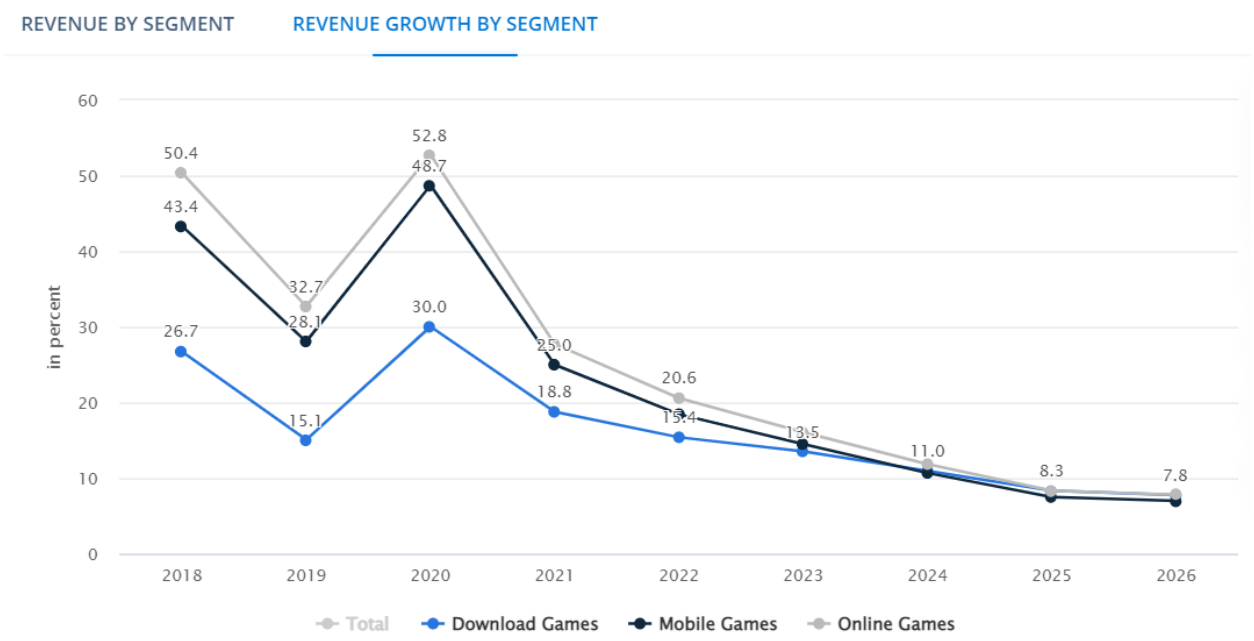


Figure 3: Revenue growth by Segment

1.3.Problem Domain

The Covid-19 epidemic made a huge impact on the educational system, with institutions across the country essentially crumbling overnight. Schools and colleges use a unique educational approach in which students are unable to physically attend classes and must instead attend sessions online, creating a new challenge. The majority of survey participants reported internet problems and lacked the ability to use and handle technology-related concerns, according to the findings.

During online classes, there are a variety of issues that might develop. As communication between students in schools and universities became more challenging, efforts were undertaken to devise ways for resolving basic issues that developed during online education.

Due to the epidemic, parents were having problems watching their children, assisting them with their online studies, providing a learning environment for their children, and so on. During the epidemic, children were often busy with other things and slacked off during online courses, and they were fascinated with online gaming and video games.

Many gaming companies are now investing millions of dollars on game development. Many gaming businesses invest millions of dollars in in-game graphics, coding, marketing, and other areas, which is almost impossible in Nepal. Young people are likewise shelling out large sums of money to play these games. Even if game firms invest a large sum of money in developing such games, players in Nepal will not purchase or spend money on them. They will just play a crack game with other foreign firms. It is quite tough to produce and fund such a large game in Nepal. In a survey, it is found out that almost all of the participants pay games at least one time at their daily life.

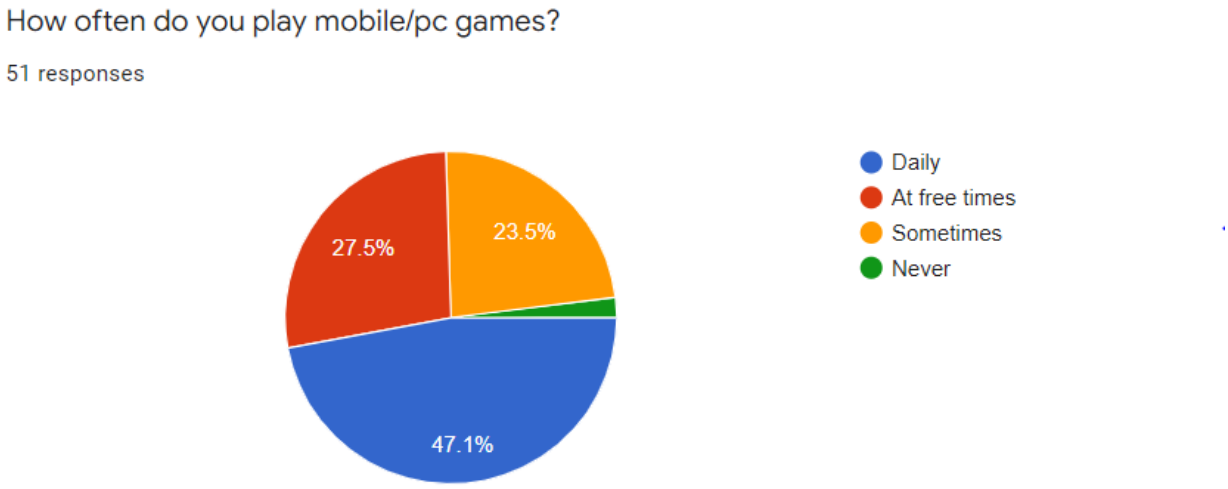


Figure 4: Survey for game

1.4.Project As Solution

We can completely resolve the current problem by developing a children's educational game. As we all know, games are quite popular among children; hence, including education into games allows children to learn while having fun. There is a lot of worthless content in games, but if we replace it with education, we can teach kids about educational concerns using the gaming content. Playing educational games may benefit children in a variety of ways, including improving hand-eye coordination, problem-solving, and strategic thinking abilities, as well as helping children build their memory. It can also aid children with attention disorders. Traditional learning is outperformed by games, especially those with simulation components, by 23%. Games are powerful motivators, according to an outstanding 2011 research, but they work best when learning is the pleasurable part, not just a side note. In the current situation, when parents worry that their children are distracted by games and are obtaining poorer marks in school, and where children just want to have fun while playing games, education games would undoubtedly be the most effective solution.

Solo game creation is the most effective technique to address and lessen the risk of game development where corporations must spend a large amount of money to generate games. Players will not have to spend as much money on design and visuals while developing a solo game; instead, they may use the available resources to create a free game. The number of downloads and the commercials played in-game are two ways for developers to make money. As a result, we will be able to build games in Nepal, and Nepalese gamers will be able to enjoy games created by Nepalese creators.

Almost all of the survey participants liked the idea of an educational game and agreed on the idea of learning educational things from video games.

Figure 5: Survey asking what people will think about educational game

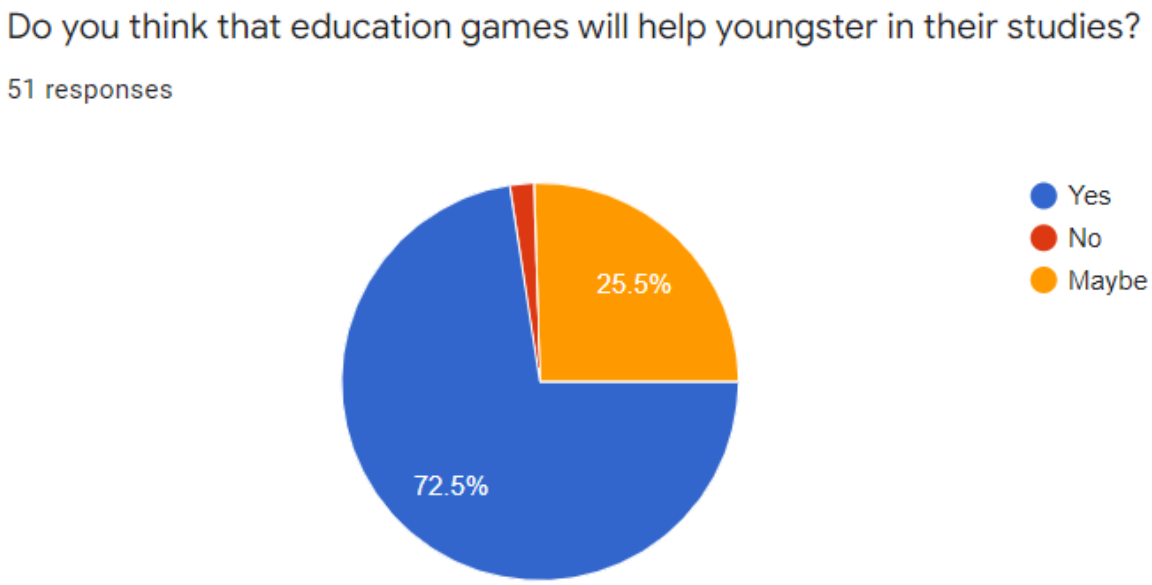


Figure 6: Survey asking what they think about educational game for youngsters

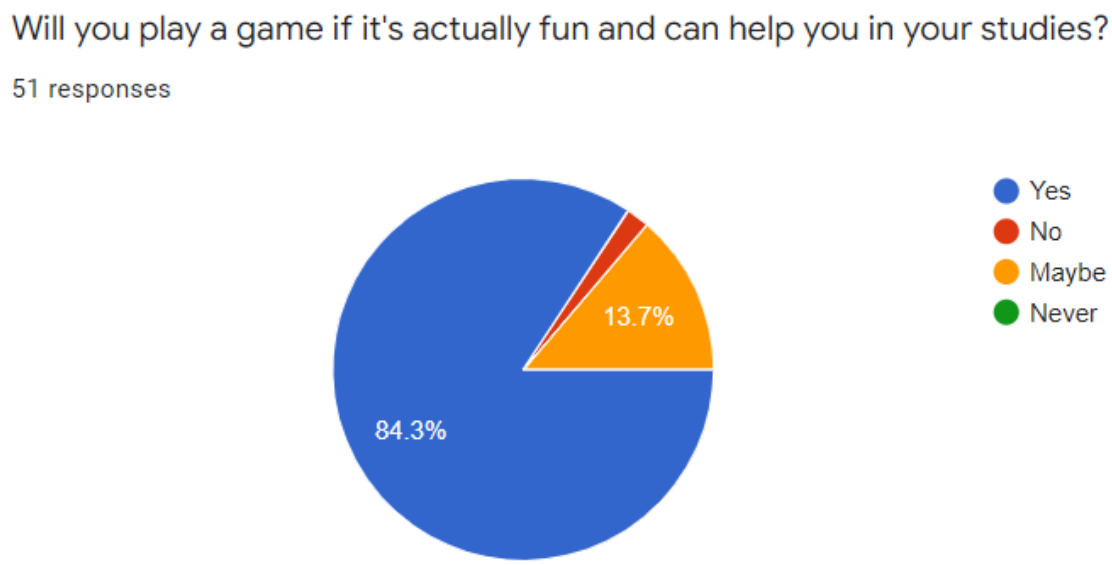


Figure 7: Survey asking people if they will play educational game

1.5. Aims and Objective

The project's major goal is to assist all parents and educators who are having difficulty educating their children and students, as well as to assist younger children in learning in their preferred manner. Only a few companies manufacture games in Nepal's present IT industry, and with this project, we can tackle two problems: the cost of producing games and the difficulty of youngsters studying in school.

Some objectives of this project are given below:

- To develop an educational game using unity C#, which enhanced the skills and knowledge in programming and game development.
- To understand how the game object works.
- To increase the thinking skills and creativity in game designing.
- To learn the implementation of the unity engine.
- To understand the working mechanism of the unity engine and the implementation of physics in-game objects.
- To be creative enough to put educational topics inside a fun game.
- To document the details of the creation and implementation of this game.

1.6. Structure of the report

1.6.1. Background

The project elaboration and literature evaluation are principally illustrated in the report's background. The project's elaboration offers a full explanation of the project's technical features, such as hardware and software needs. In the literature review, a full overview of the survey results and end-user interviews is offered, as well as a quick comparison of similar systems.

1.6.2. Development

The methodology used in this project is clearly explained in this part, as is the purpose of choosing this methodology. In addition, the stages involved in the chosen methodology are briefly described, along with a work breakdown structure and Gantt chart.

1.6.3. Testing and Analysis

The testing and analysis section includes the system's various test cases. It includes Functionality testing, Performance box testing, and Compatibility testing, as well as the necessary tables and screenshots. It also includes end-user testing.

1.6.4. Conclusion

The conclusion section includes a list of additional tasks that must be completed in the future. It also includes a final project evaluation and a critical analysis of the entire project. It provides a discussion of the issues that have been and may be encountered by the application.

Chapter 2 Background

2. Background

2.1. About the end user

To begin with, this educational game is designed for children in grades 6 and up, however teachers and parents are more interested in the educational game than the children themselves. The educational games that are being produced on the internet are appreciated by not only children and teenagers, but also by parents and teachers. I looked through the educational game review area and articles and discovered that parents and teachers are more likely to play or test the games before their children and students. The majority of the time, teachers and parents will provide feedback on educational games. Some of the educational games were also very interesting and fun to children and youngster.

2.2.Understanding the Solution

2.2.1. Project Elaboration

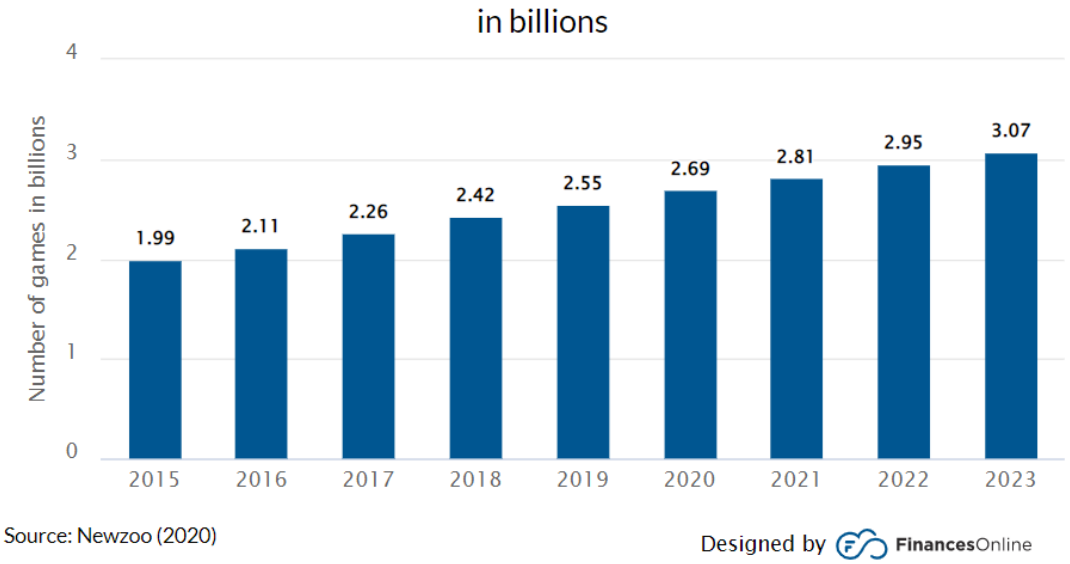


Figure 8: Number of active video gamers worldwide

Every year, the number of people who play video games are growing rapidly. Nearly one-third of all youngsters play video games. Many young people are influenced by both offline and online video games, which has a negative impact on their education. Addiction to video games causes children to lose concentration on their schoolwork. This condition is extremely difficult for children, and their parents and teachers are quite concerned. In order to level up or get stronger in the game, players will go to any length. If we include education into the game and have them complete educational chores in a pleasant fashion that will help them become stronger in the game or level up, they will undoubtedly complete those assignments and learn a great deal. Even if no education is included in the game, there are numerous things that players may learn from particular types of games. By keeping the player occupied, the game helps to prevent addiction to other harmful habits while also reducing anxiety. An educational game is an excellent method to address this issue. Youngsters may use their procrastination tendencies in an educational game to help them study while still having fun.

2.2.2. System Architecture

The project is being introduced to address a contemporary issue that every parent, teacher, and student is facing. This project will assist young children in learning educational topics in a fun and interesting manner, and their parents and teachers will not have to worry about their children's procrastination habits. In the case of Nepal, a solo game will solve the financial problem while simultaneously providing opportunity for skilled game developers to create their own solo games, so addressing the problem of unemployment. In this circumstance, the educational game will be revolutionary.

In the game there only user to system interaction where user gives the instruction to the game and game will analyze the order of the user and pass the instruction through different states and display result to the user through GUI/Screen. The calculation of the user order and result is very quick so that player will be able to play the game in real time.

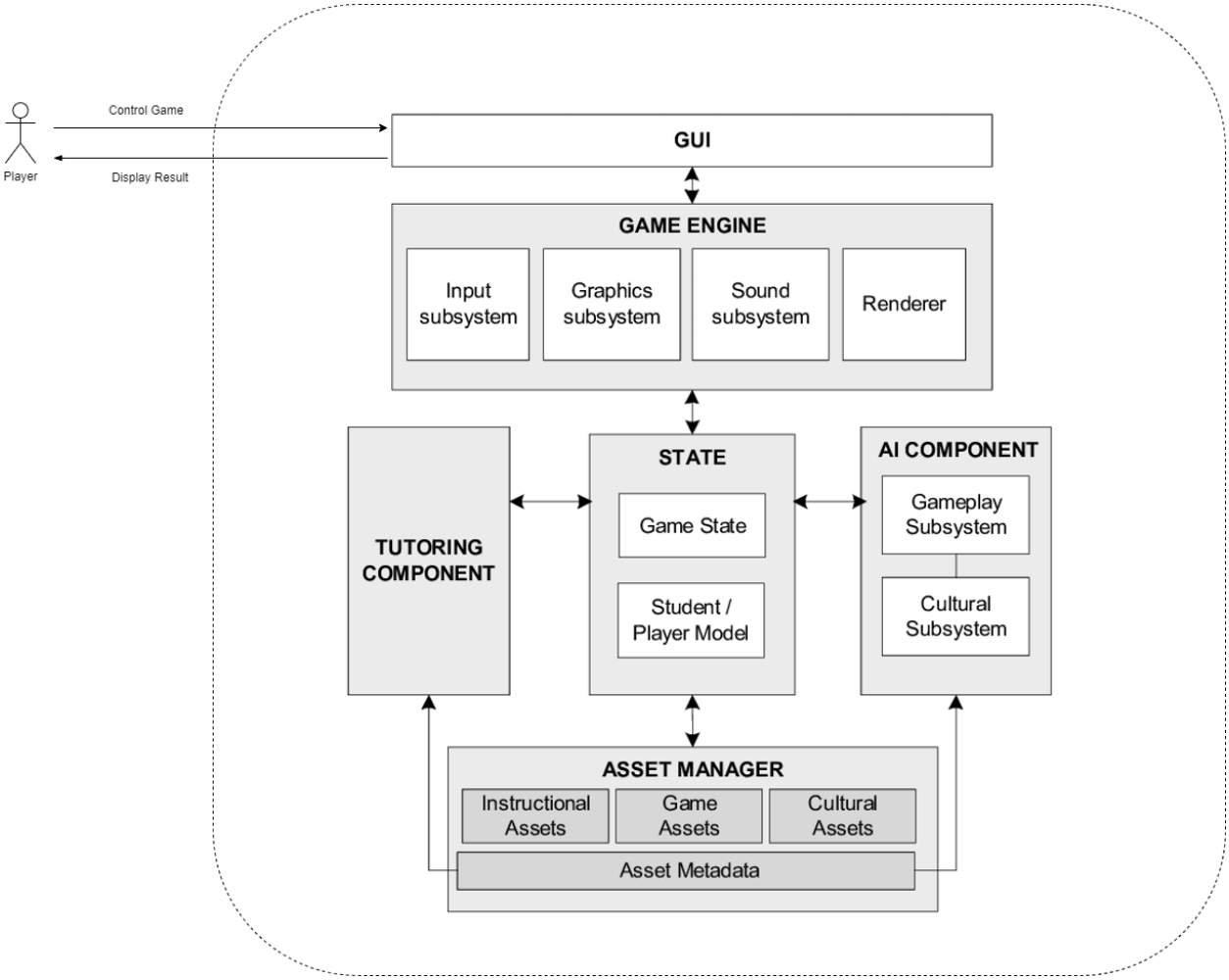


Figure 9: System Architecture Diagram

2.2.3. Review of technical aspects

Since the project's goal is to create an educational game, it requires lots of software. The following is a list of the software that will be used in this project:

1. **Windows**

Windows 10 is a Microsoft operating system designed for personal computers, tablets, embedded devices, and internet of things devices. Microsoft released Windows 10 in July 2015 as a follow-up to Windows 8. The company has stated that it will continuously update Windows 10 rather than releasing a new, full-fledged operating system as a successor. Windows 10 includes built-in capabilities that enable corporate IT departments to secure and control devices running the operating system using mobile device management (MDM) software. Traditional desktop management software, such as Microsoft System Center Configuration Manager, can also be used by organizations (Bigelow, 2021).

2. **Visual Studio 2019**

Microsoft Visual Studio is an integrated development environment (IDE) developed by Microsoft for several forms of software development, including computer programs, websites, web applications, online services, and mobile apps. Completion tools, compilers, and other capabilities are included to make the software development process easier. Visual Studio is used to write code for my system.

3. **Adobe Illustrator 2021**

Adobe Illustrator is a vector-based design and drawing application for professionals. Illustrator may be used to create everything from single design components to whole compositions when used as part of a wider design workflow. Illustrator is used by designers to make posters, symbols, logos, patterns, and icons, among other things. Adobe Illustrator is used to design game objects and game characters.

4. **Unity**

Unity is a robust cross-platform IDE for developers and a 3D/2D game engine. Unity can provide many of the most crucial built-in elements that make a game work as a game engine. Physics, 3D rendering, and collision detection are all examples of this. Unity technologies created a cross-platform game engine that is mostly used to create video games and simulations for computers, consoles, and mobile devices. Unity is used to develop this education game.

2.2.4. Functions and Features

The required features for the game application are as follows:

- Saving the settings information:

  The settings panel's customized settings are saved in cache memory even after the game application is closed and reopened.

- AI Feature:
  The AI in the game application is Villain and Crawling Enemy. Crawling Enemy is a preset AI enemy that will follow the player wherever he goes.
- Tutorial
  There is a game tutorial in game application which will teach player before he or she play the real game.
- Storyline
  Storyline is one of the main features of the game application. The story lines tell the player about the journey.
- Obstacles
  There are different obstacles in the game application which make it very interesting and challenging.
- Educational Part
  The education part of the game is physics. The education content is provided in the tutorial as well as in levels.
- Levels
  Different level with different graphics and different design is available.
- Animations
  Animation is one of the most important features of the game application. Animation is added in almost every “Game Objects”.
- Messages
  Pop up as well as normal messages are there in every level of the game which guide the player to complete the game and as well as provide the educational content to the player.

2.3.Review of similar systems

2.3.1. Colors & Shapes Game - Fun Learning Games for Kids

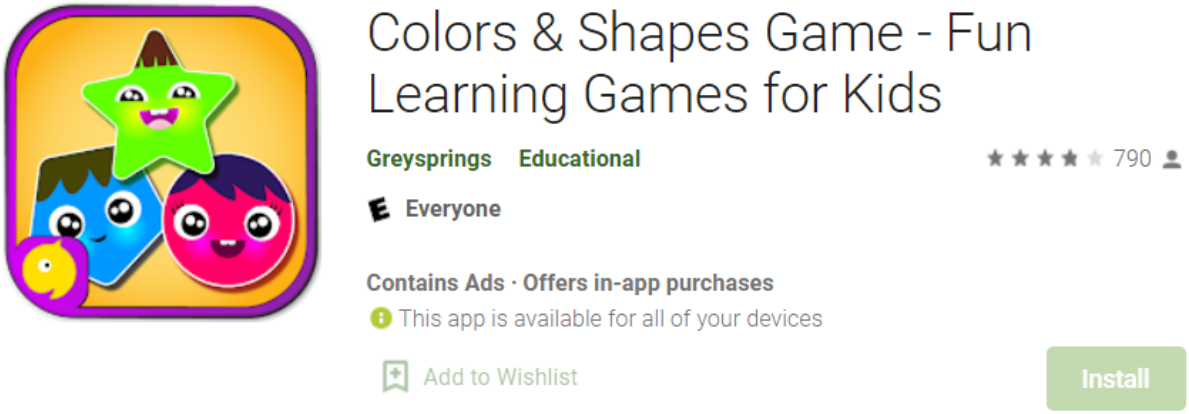


Figure 10: Colors & Shapes Game- Fun Learning Games for Kids

Colors & Shapes Game is an education games for kindergarteners (age 2-6) years. This game includes educational games activities to teach kindergarteners about shapes & colors using vegetable/fruit and other real life household objects. Kids Learn Shapes & Colors is a fun and educational software that children will enjoy. The game's experience is enthralling to children since there is no winning or losing. The activities are meant to help kids learn shapes and colors while also expanding their early learning and discovering new sounds. Kindergarten children enjoy cartoon coloring pages, thus free coloring books are creative games for them which is included inside this game (Grey Springs, 2021).

✨Learning Shapes & Colors Games for Kids✨

📍 Attractive and enticing designs and pictures for learning shapes and colors for preschool children
📍 Engaging kids learning activities for toddlers for early childhood education about Shapes and colours
📍 An interactive and engaging way, meant for early learning for kids
📍 Colors for kids is a great learning game for children
📍 The tracing of geometric shapes for toddlers enhances hand eye co-ordination and motoring skills. It not only enhances the hand eye co-ordination but also prepare the child for writing, without even knowing the she is being trained for writing
📍 So Many free coloring pages for kids to enjoy unlimited fun
📍 Painting using pencils and crayons to color the Geometric shapes for toddlers and so many coloring pictures and sheets
📍 Learning Shapes and colors for toddlers has been presented in the form of fun learning activities like Dragging the monster for space jump, feeding hungry frog, Balloon pop quiz, Odd one out, Honey bee etc.
📍 Many interactive activities to teach about different geometric shapes like Circle,Square,Triangle,Heart,Diamond,Star,Semicircle,Oval,Rectangle,Pentagon,Hexagon etc.
📍 In the activity Robot factory, Kindergarteners learn about shapes and their usage.
📍 Hidden object Games- Scratch and reveal different colorful Geometric shapes.

Figure 11: Specialties of Colors and Shapes games

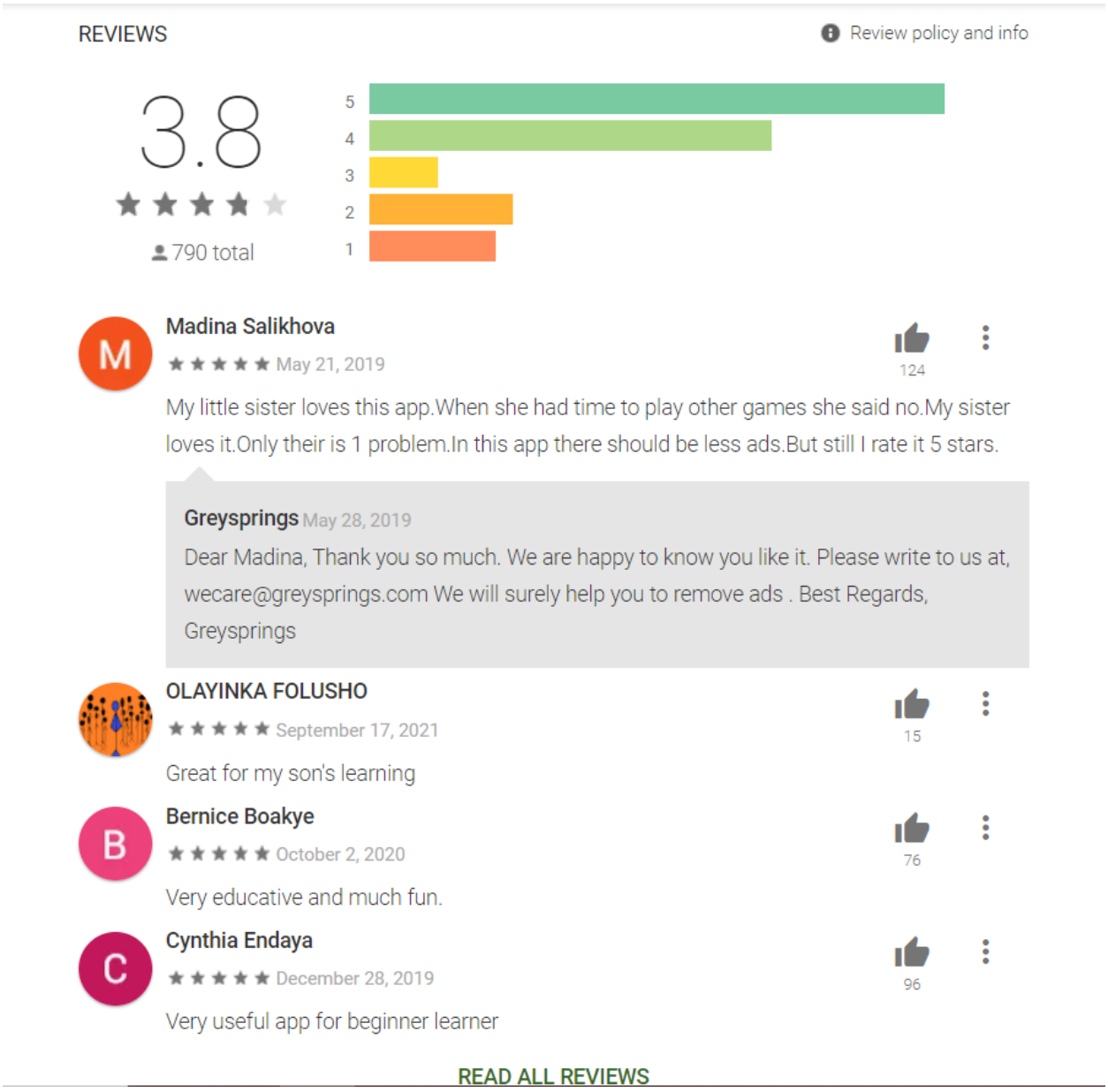


Figure 12: Some reviews on Colors and Shapes games

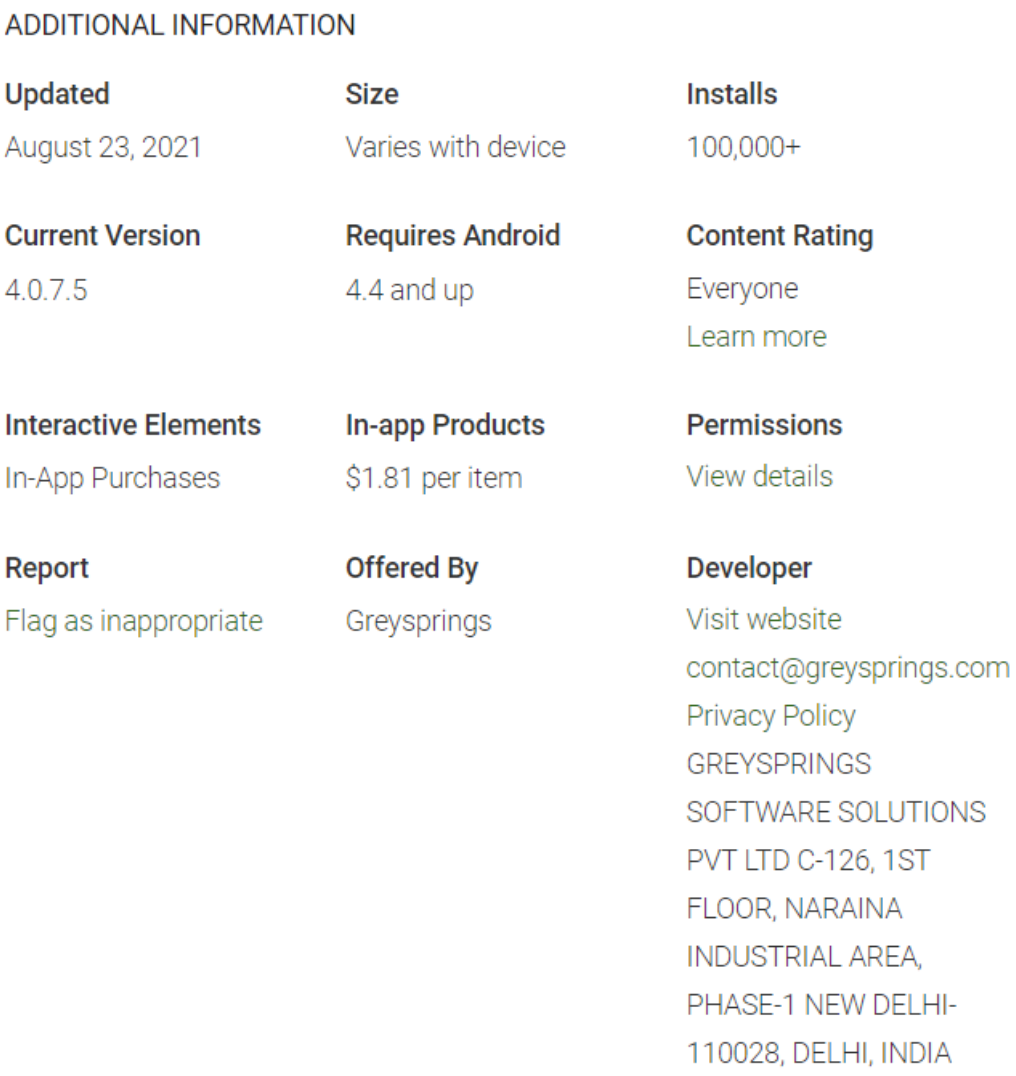


Figure 13: Additional Information on Colors and Shapes Games

2.3.2. Masha and the Bear: Water game

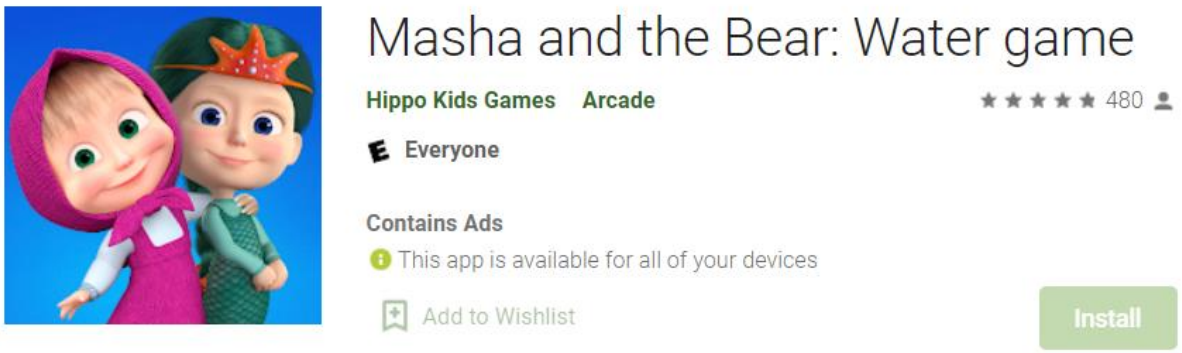


Figure 14: Masha and the Bear: Water game

Masha and the Bear: Water Game is an educational game for kids whose age is 3-7. This is an easy educational game that will help player to learn colors, find the same items and make various puzzles. The characters from children's favorite cartoons are use in this game to make it more fun and interesting for children (Hippo Kids Games, 2021).

👧 🐻 If you like cartoons and funny puzzles, Masha and the Bear are ready to bring you to their team. Hurry up! This game set about Masha has both educational and entertaining elements. That's why these educational games for kids from 3 to 7 years are perfect for toddlers and their parents.

🌊 Download this app and visit a magical underwater world with cartoon characters. Educational mini games create a real fairy tale. Golden fish, sunken ships, treasure chests, mermaid and a lot of other creatures from the underwater world.

💰 Find pirate treasures. The Bear uses her fishing rod to find coins at the sea bottom. But don't touch fish, we are interested in the golden treasures.

🚣 Wooden trunks will help us to get to the other side of the river. Be an attentive player not to let Masha fall into water.

🧩 Collect puzzles with amazing marine pictures. Beautiful puzzles are created especially for preschoolers.

🍳 Feed friends. Choose food for every animal. We are going to cook in the nature.

Figure 15: Specialty of Masha and The Bear: Water Game

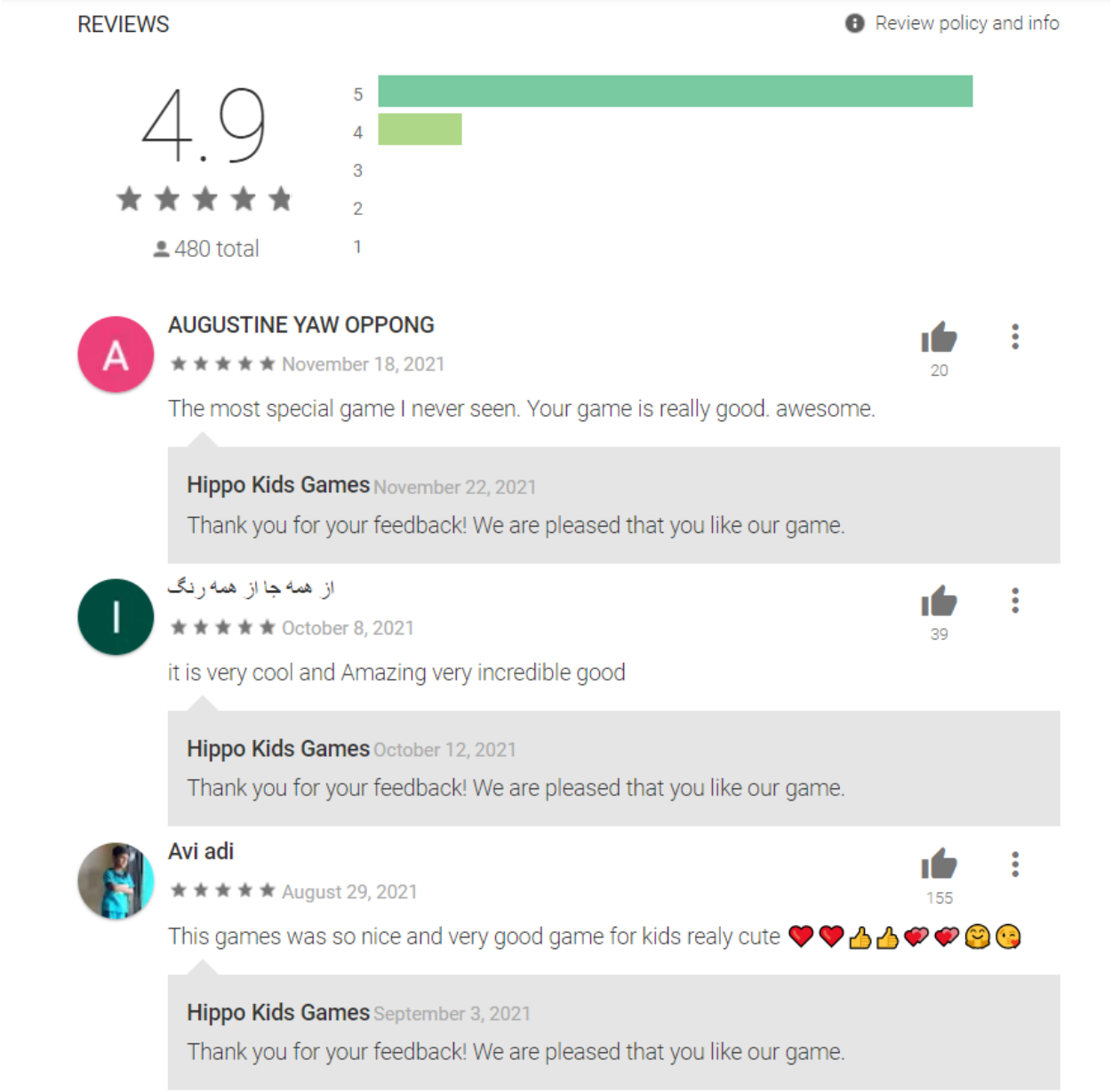


Figure 16: Some Reviews for Masha and the Bear: Water game

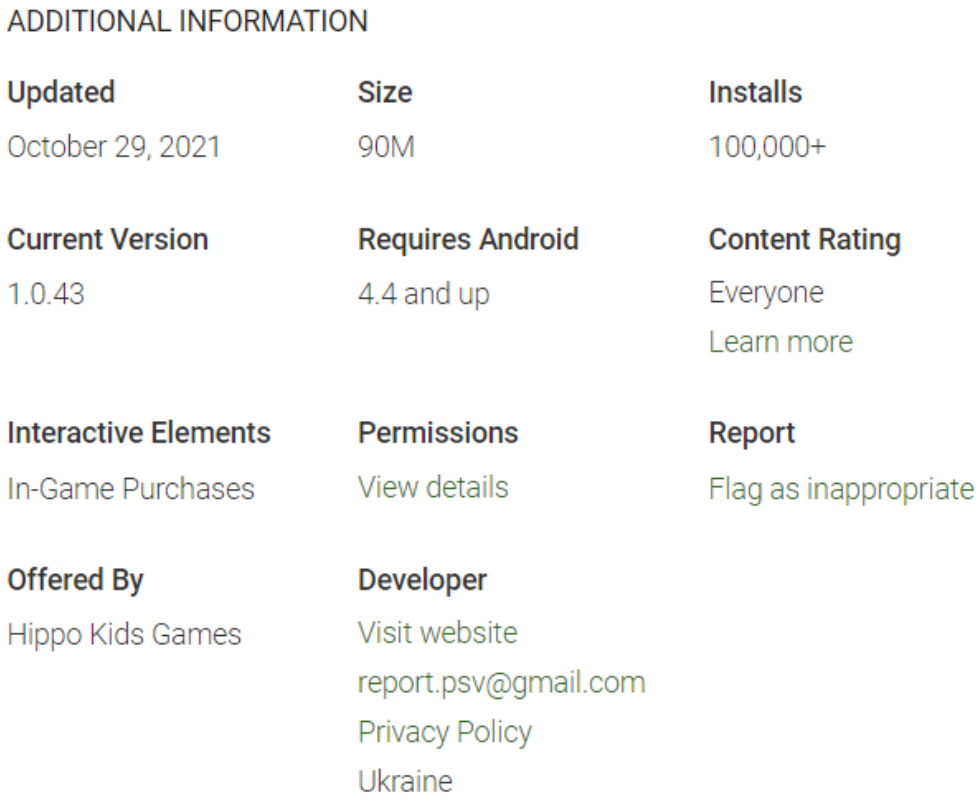


Figure 17: Additional features of Masha and the Bear: Water Game

2.3.3. Adventure Academy

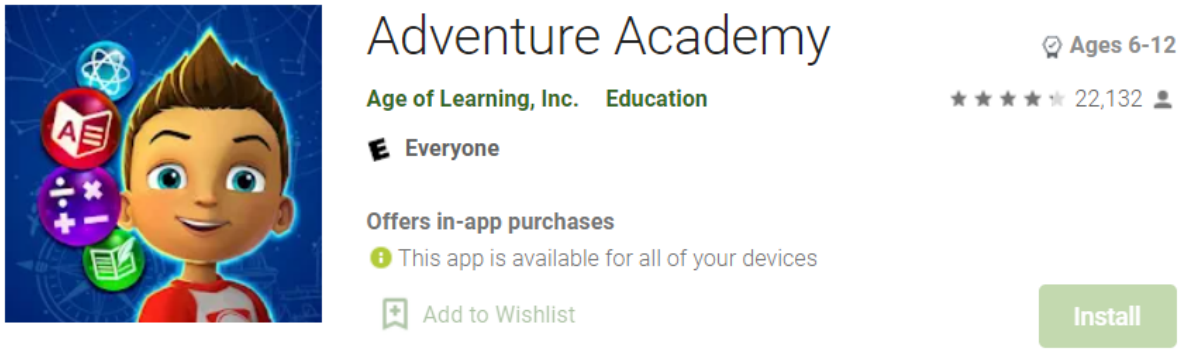


Figure 18: Adventure Academy

Adventure Academy is an educational adventure game which is playable by the age group 6-12. In Adventure Academy, players will travel around the academy and solve the problems of math, science, grammar, etc. to level up. This game will increase the player knowledge about academy studies and also help them to increase their problem-solving skills. Not only kids but also the early teenagers are also suggested to play game (Age of Learning, 2021).

Key Features:

Discover and play thousands of learning activities to help boost key skills and abilities across many subjects, including math, language arts, reading, science, social studies, and more

Share amazing experiences with your best friends in a safe and secure environment

Play simultaneously with up to three other scholars within the same account on your individual smartphone, tablet, or computer

Crafted by teachers and early learning education experts with a curriculum-first approach

Complete quests and learning challenges with your friends to level up and advance your character's abilities

Great for homeschoolers and learning on the go

As you continue your journey through Adventure Academy, you can also earn amazing rewards for your accomplishments. Hundreds of clothing items, hair styles, and looks give a seemingly unlimited number of choices to customize your character. Also, you can earn your own 3D virtual home within the game that you can decorate with items you earn from completing learning activities.

Trusted by teachers and designed by early learning education experts, Adventure Academy provides a safe learning environment for students. Parents can monitor progress, control the level of interaction through filtered chat, or block in-game communication entirely.

Figure 19: Key Features of the Adventure Academy

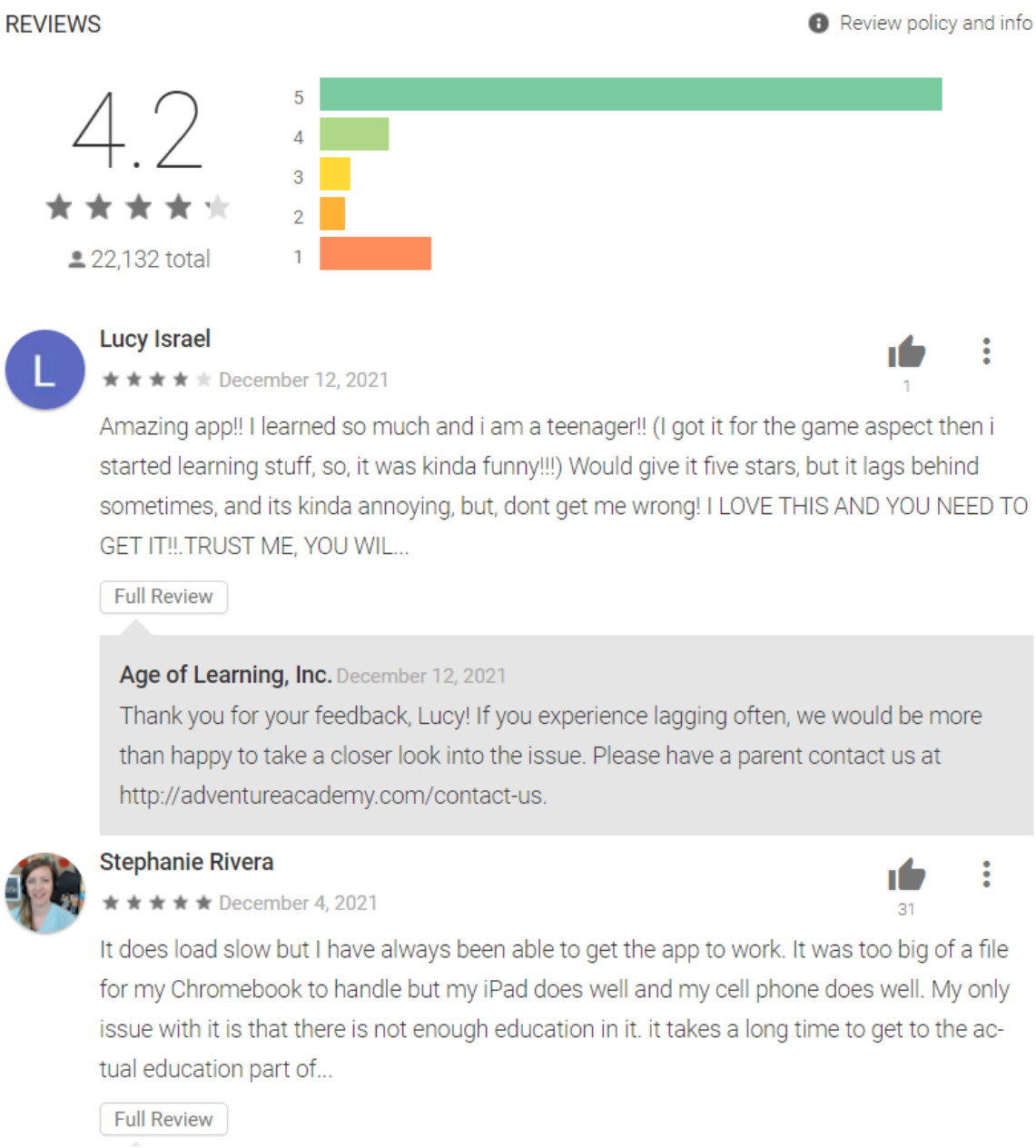


Figure 20: Reviews for Adventure Academy

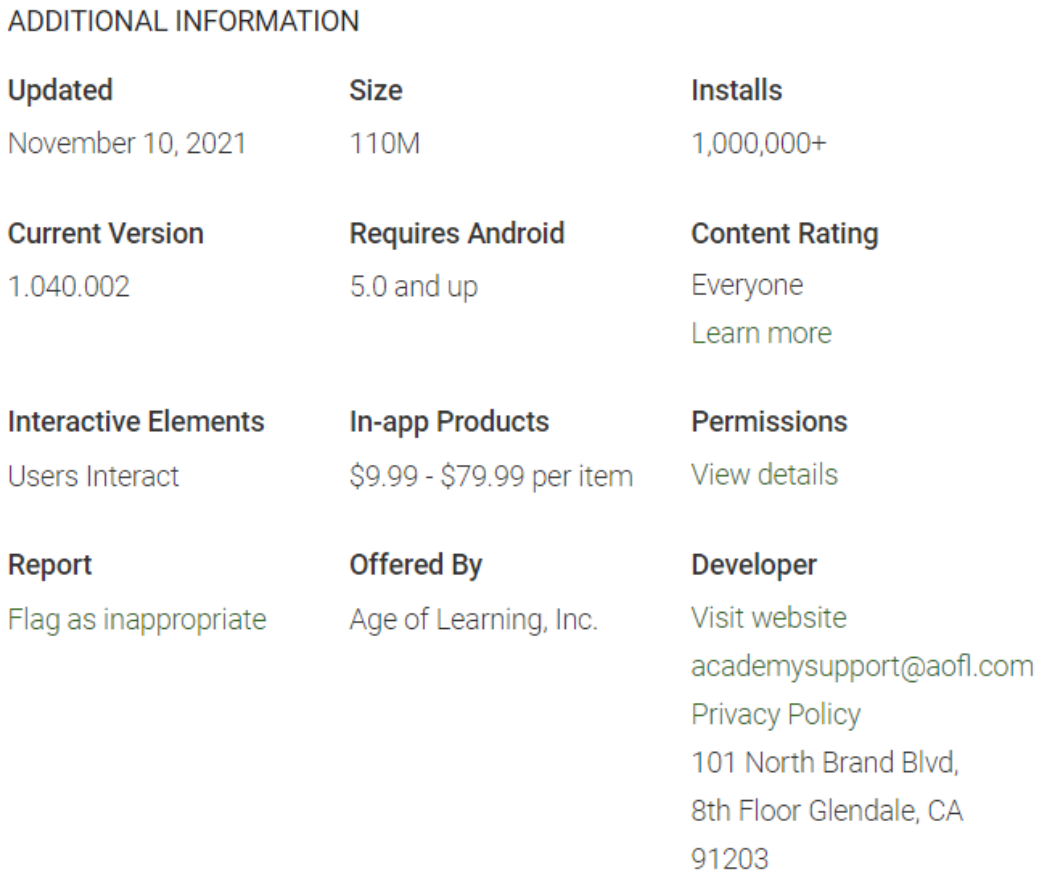


Figure 21: Additional Information for Adventure Academy

2.3.4. Baby Games for 1+ Toddlers

Figure 22: Baby Game for 1+ Toddlers

Baby Games for 1+ Toddlers is a game application for kids of the age group of 2-4 years. In this game, Kids will learn colors, shapes and follow through simple thematic stories. Like caring for cute animals, or sorting things by size, color and shapes (kidsEducational, 2022).

15 Ads Free learning games for kids under 5!
Educational activities in the games are planned and tested by experts.

Free toddler games in the app, provide great **learning and entertainment experience** for preschool children, both **baby boys and girls**. They will spend time flexing and improving their **motor skills**, star recognizing **colors and shapes**, follow through the baby games with **simple storyline**.

Learning experience is planned and tested by toddler development experts, to be best suited for kids under 5 years and to be **simple, fun and educational at the same time!**

Additionally all the settings and outbound links are **protected and inaccessible for babies.**

Please support us by writing reviews if you like the app and let us know about any issue or suggestions too.
This baby games app is free and without any ads!
Enjoy your time with your kids!

Figure 23: Specialty of Baby Games

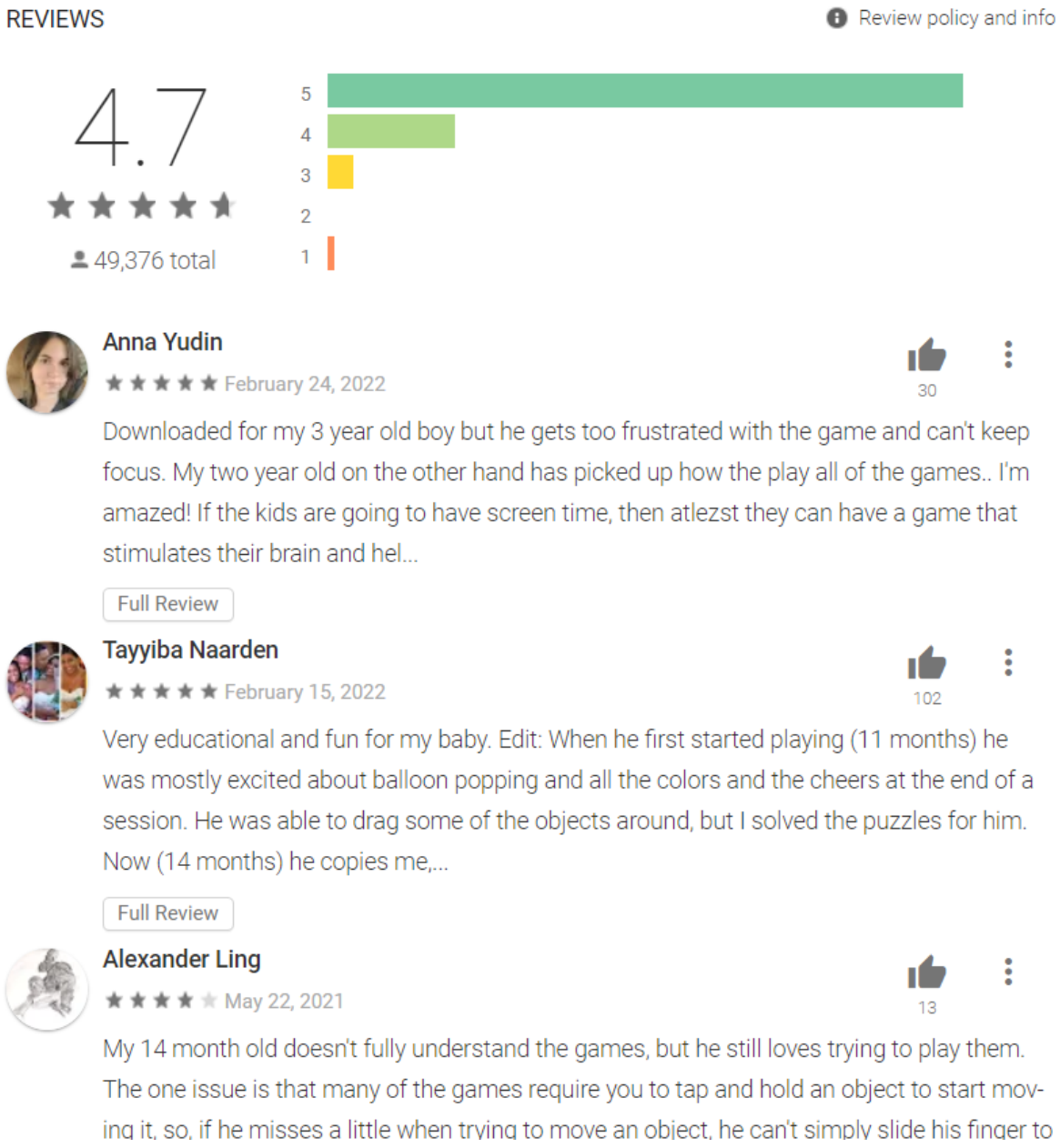


Figure 24: Reviews on Baby Games

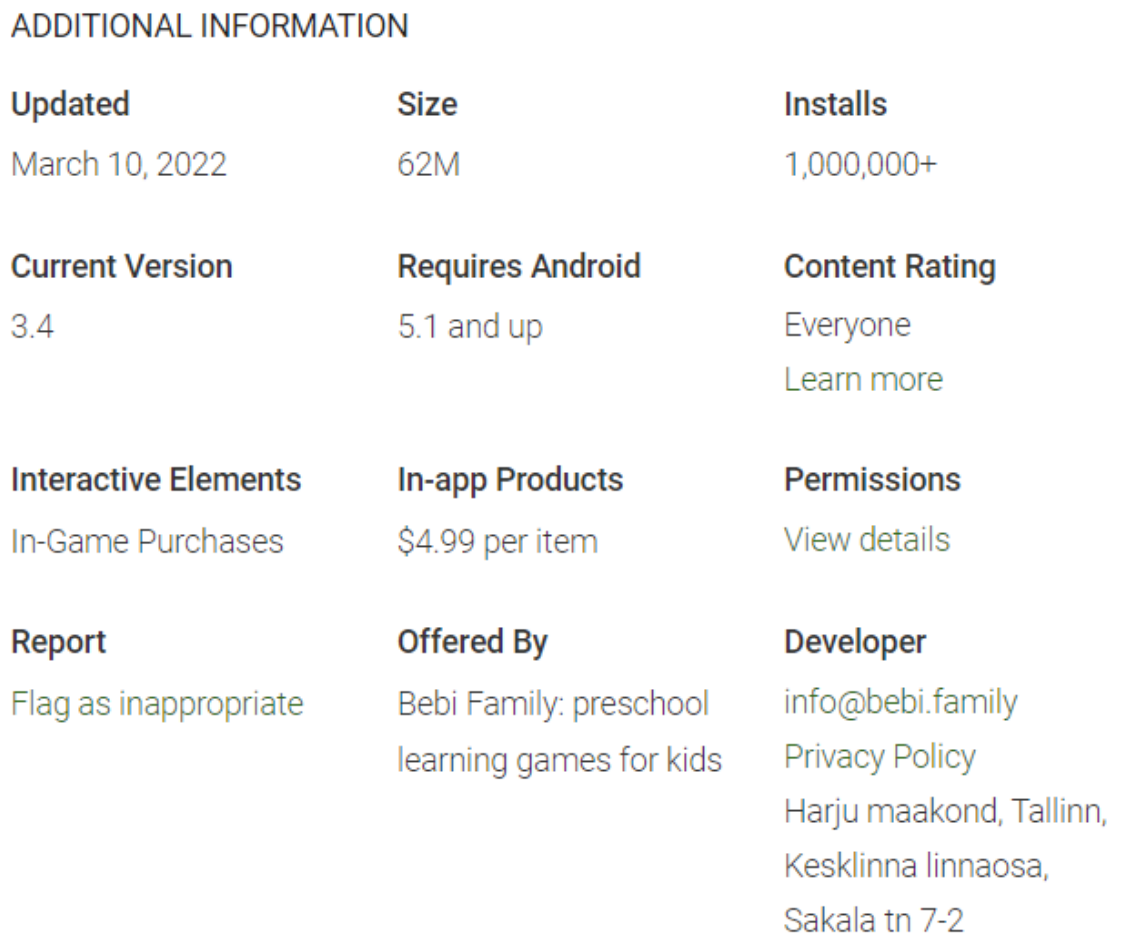


Figure 25: Additional Information on Baby Games

2.3.5. Toddler Games for 2–3-Year-Old

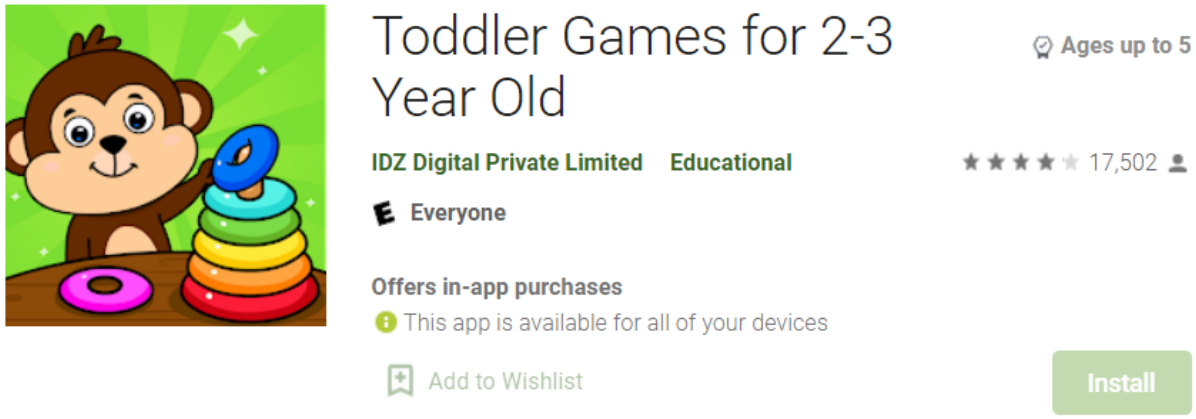


Figure 26: Toddler Games 2–3-Year-Old

Toddler Games for 2–3-Year-Old is education game for kids of age group 2-5 years. Kidlo Toddler Games includes puzzles, popping bubble games, coloring, join the dots, dress up, match the pairs, sorting, tracing games and more. These games will boost cognitive skills, hand-eye coordination, concentration, and imagination of children (IDZ Digital Private Limited, 2022).

Kidlo Toddler Games is a fun educational app for 2 year olds and 3 year olds with **1000+ interactive games to play** and learn easily. Begin your child's early learning journey with these exciting educational games!

Kidlo Toddler Games includes puzzles, popping bubble games, coloring, join the dots, dress up, match the pairs, sorting, tracing games and more. These games will **boost cognitive skills, hand-eye coordination, concentration, and imagination** of your children. You will be amazed to see how quickly your 2 year old or 3 year old toddler learns different things.

These games are a great way for you to have some fun learning time with your two year old. Watch your toddler complete these fun games while learning at the same time. It is filled with **cute, colorful graphics and funny animations** which encourage your three year old to learn shapes and colors easily.

Features of Kidlo Toddler Games are:

Learning Games for two year olds and three year olds.
These games are ideal for toddlers to learn animals, numbers, shapes, colors and much more!

Kid Friendly
All the educational games in this app are completely kid-friendly. 2 year old and 3 year old Toddlers can play them easily.

Different Categories
Choose from a wide range of game categories like Farm, Animals, Fun, Insects, Space, City, Marine, Vehicles and Birds.

Available Offline & No Ads
These games can be played anytime and anywhere without internet. Completely ad-free.

Figure 27: Specialty of Toddler Games for 2-3-years old

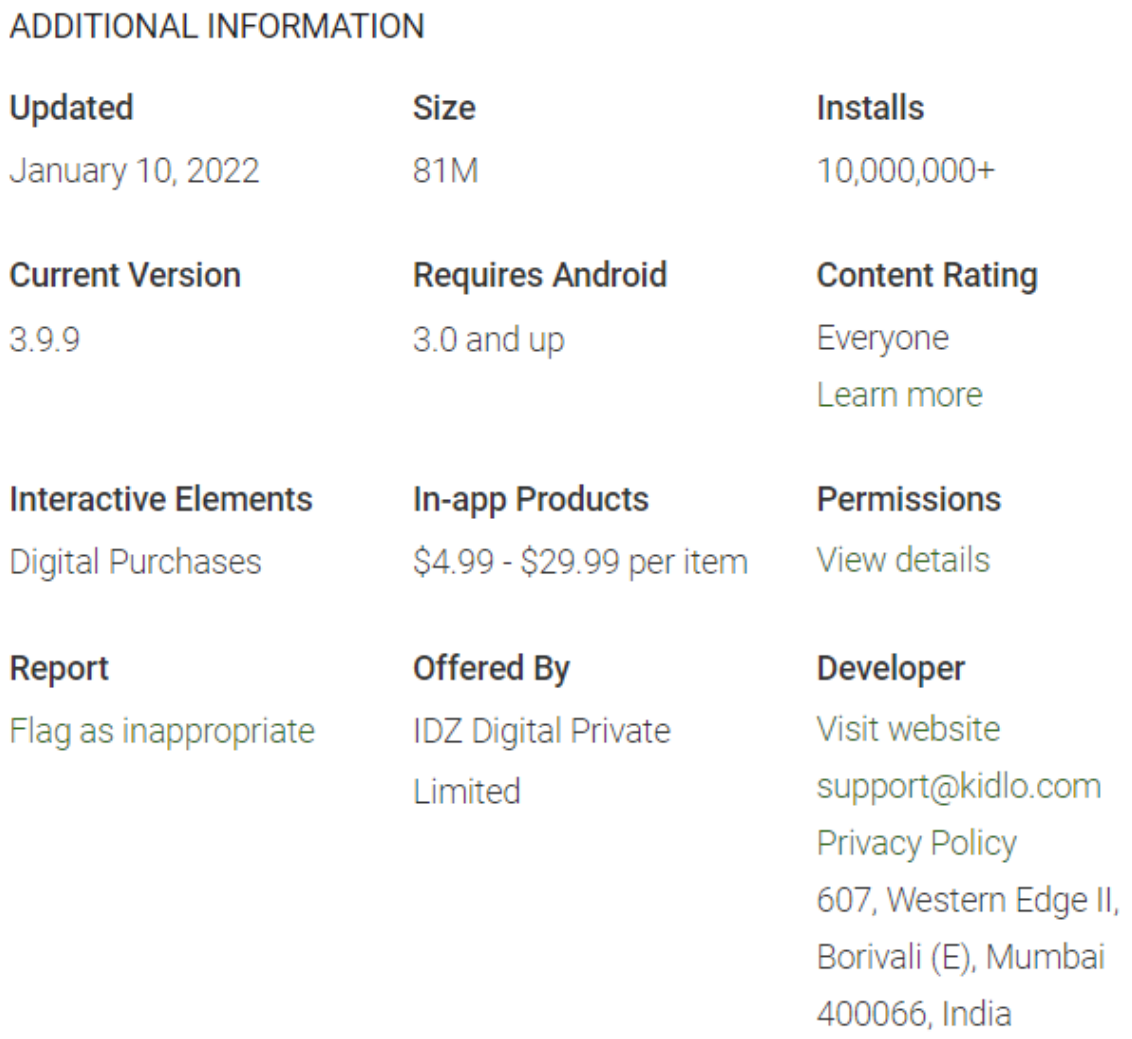


Figure 28: Additional Feature of Toddler games for age of 2-3 years old

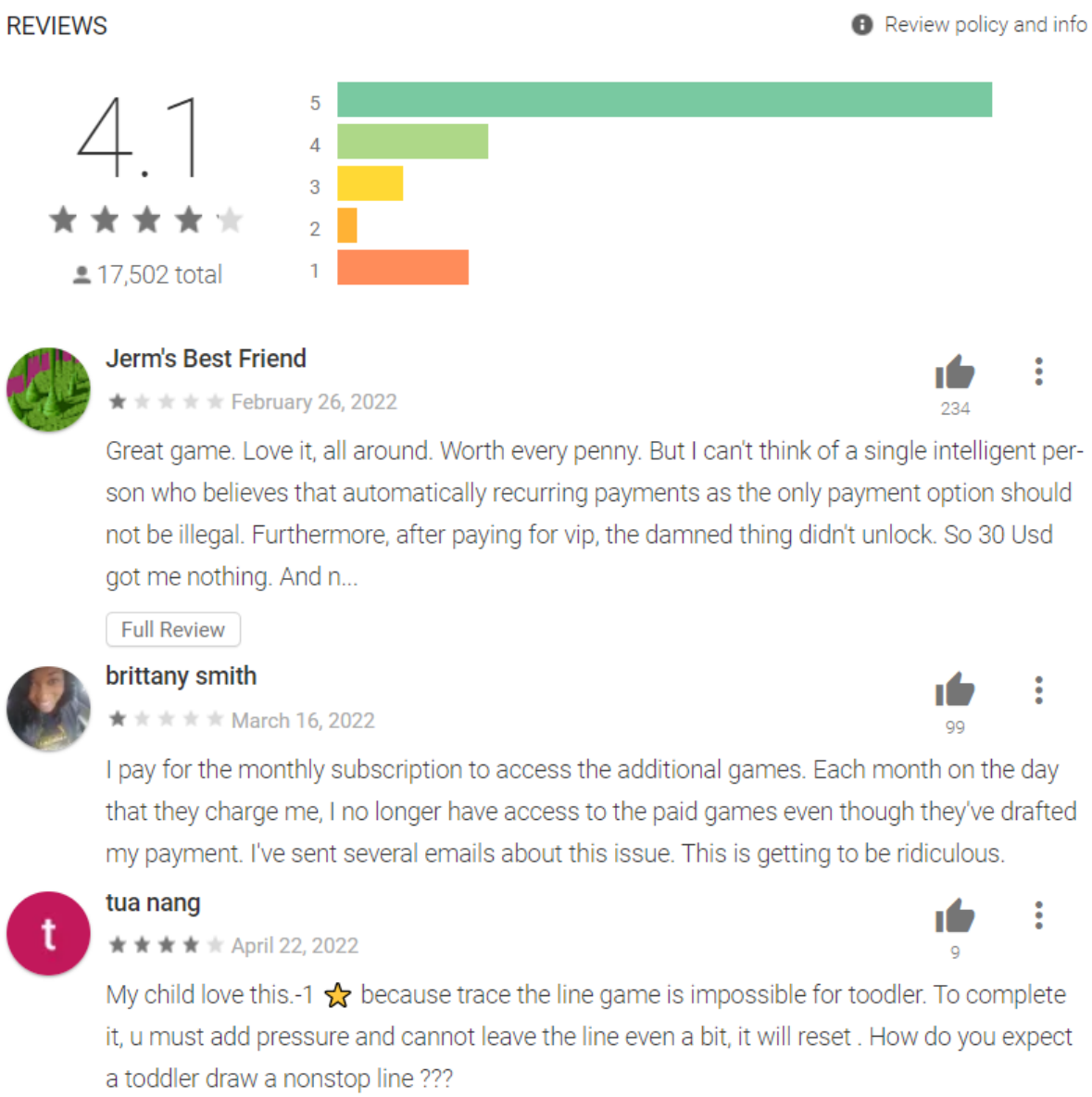


Figure 29: Reviews on Toddler Game for 2-3-years old

2.4.Comparison

2.4.1. Comparison between Similar Systems

All three games are educational games aimed at various age groups. They were all posted to the Play Store. Many people enjoy these games, and many of them have received great reviews. All of these games are developed by various companies and have different game ideas. Each one of these games has different educational component to it. All of the games are designed with the age range of children in consideration. Below is a comparison of the features of these games with my game:

Comparison List						
Features	My Game	Colors and Shapes Games	Masha and the Bear	Adventure Academy	Baby Game	Toddler Games
2D or 3D	2D	2D	2D	2D	2D	2D
For Age group	13 and above	2-6 years	3-7	8-12	2-4	2-5
Education About	Newton's law of motion and Physics	Colors and Shapes	Colors and Items	Math, Science and Grammar	Colors, shapes and animals	Puzzles and colors
Mobile Application	No	Yes	Yes	Yes	Yes	Yes
Windows	Yes	No	No	No	No	No
Solo game	Yes	No	No	No	No	No
AI	Yes	No	No	No	No	No
Levels	Yes	Yes	Yes	Yes	Yes	Yes

Table 1: Comparison List Table

2.4.2. Conclusion from Similar System

Analyzing the similar the other similar application can reveal the result of the difference between the features application. All of the educational games mentioned above for comparison came from the Google Play store. They were highly rated by users, so I believe my application is just as useful as all of the applications mentioned above.

When compared to similar applications, my application has more and some additional unique features that make it more interesting.

Chapter 3 Development

3.1. Considered Methodology

A game is a type of software application that is designed to give amusement. When it comes to starting and creating games, merely following the software development life cycle (SDLC) isn't enough, because game developers encounter several obstacles during the production process. (Examples of difficulties include graphics, visuals, sounds, animations, physics, collisions, AI, gestures, and user inputs, among others). A new approach known as GDLC (Game Development Life Cycle) will be introduced to solve the difficulty that every game creator faces.

3.2. Selected Methodology

To develop strong software architecture for your game on all platforms, GDLC relies on common streamlined engineering concepts (Pressman, 2009). The game development process is extremely complicated and building a product for numerous platforms will always necessitate a collaborative multi-skilled/talented team. Game producers, game/art directors, a technical team, game designers, a game artwork team, a game quality team, a game programming team, a game testing team, and a game marketing (post-Production) team are all involved in the process (T.Fullerton, 2008)

3.3. Phases of GDLC

To develop a successful game, any new gaming project should follow all the stages outlined in this image:

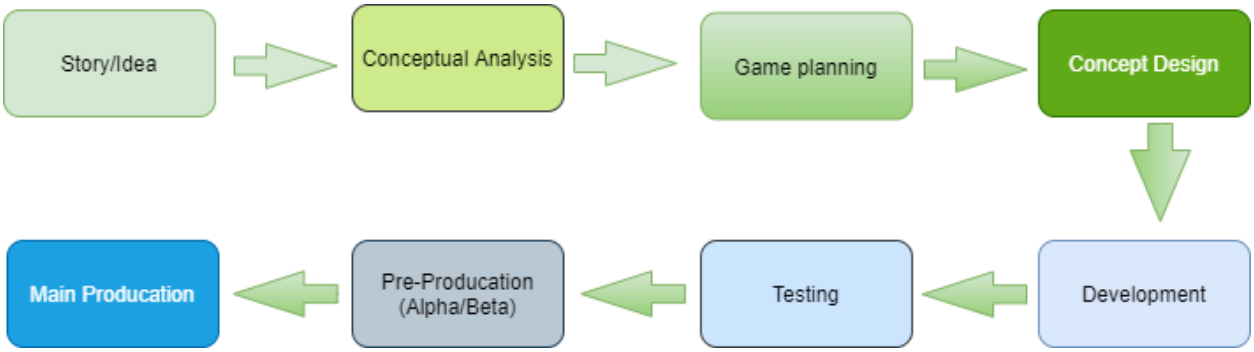


Figure 30: GDLC Chart

a. Story/Idea

The creators decide on the sort of game they want to make in the first stage, such as the target audience or players, the type of game, the hero, the protagonist of the game, the game's theme, and so on. The actual game concept and need of our project is the story/idea. All of this should be stated in the project/requirement document. Ideas are merely snatches of information from your game. It's the game's genuine prototype, which includes all of the concepts. Game Concept/Game Story refers to a collection of concepts.

Task	Estimated Duration
Game Idea	3 days
Game Story	3 days
Game Character Idea	2 days
Game Character Description	2 days

Table 2: Story/Idea Phase

b. Conceptual analysis

According to the story or concept, a thorough analysis of the need is required. Before beginning game production, a feasibility study should be completed. As a result, we'll need to look at a few areas.

- Actual Requirements
- Pricing
- Technical Capabilities
- Organizational, Cultural or Legal Issues and Solutions
- Skills and Scope of the project

Task	Estimated Duration
Requirement Analysis	3 days
Price Analysis	3 days
Technical Analysis	3 days
Skill Required Analysis	3 days
Solution Analysis	3 days

Table 3: Conceptual Analysis Phase

c. Game planning

Following the collection of all needs and data from studies, game development planning is required. The game's project plan or blueprint must be created. On paper or a chart, write down all of the features, tasks, and ideas.

- Make a task list (graphics, animations, sounds, etc.)
- Scheduling and estimating the duration of each task
- Create a document/chart depicting the workflow
- Determine the workflow as well as the test cases and test strategies

Task	Estimated Duration
Task List Creation	1 day
Scheduling	1 day
Document/Chart of Workflow Creation	2 days
Test Case Determination	3 days

Table 4: Game Planning Phase

d. Concept Designing

The term "concept design" refers to the creation of a design prototype for a certain demand, idea, or story. It is also known as "game design." The core of each game/product is the design. Game Design demonstrates mastery of the trade-in bringing a concept to life. Any game's most inventive, creative, and complicated process. It is required to create decent and high-quality games. It necessitates critical interactive thinking, comprehension, implementation, execution, behavior, and user interface design. Before beginning production, the game designer must produce a document known as the "Game Design Document (GDD)" (Chandler, 2010).

Game Design Elements:

- UI Interface
- Game Data
- Player Data & Characteristics
- Level Design
- Gameplay & Mechanism
- 3D/2D Game Arena
- Game Objects/Powers/Properties
- Sound Music

Task	Estimated Duration
UI Design	15 days
Game Data	5 days
Level Design	10 days
Game Play/Mechanics	12 days
Game Arena Design	15 days
Game Object Design	10 days
Sound/Music	6 days

Table 5: Concept Designing Phase

e. Development

Following the completion of the GDD, it is now time to begin developing the actual game concept/idea as described in the Game Design. The programming for the development should begin with the selection of the Game Engine and its supporting modules/plug-ins/frameworks/platforms. The lead programmer is responsible for the game's development progress and quality; the lead programmer should create a checklist of pending, working, and finished tasks based on the developer's tasks. The work of each programmer/developer must be submitted to the main programmer.

The Developer should be knowledgeable about the coding system and its vocabulary.

This part includes:

- Abstraction
- Modularity
- Pattern of Design
- The architecture of Software and Games
- Structure and Style of Coding
- OOP skills
- Robust programming
- Use fewer resources & generate more output
- Feel end-user experience

Task	Estimated Duration
Abstraction	64 days
Modularity	15 days
Pattern of Design	10 days
Architecture of software and game	20 days
Robust Programming	16 days

Table 6: Development Phase

f. Testing

The most crucial aspect of the GDLC is testing. In any game/concept development architecture, testing and game design are equally important. Testing is more than just playing a game at work or in the arena. It's about the real end-user experience with our product. It is a repeated and interactive process of receiving the same screen flow input and anticipating the user's response in terms of game quality.

While doing QA testing, it must work on two papers, one under the Bugs Tracker Report and the other under the Testing Report.

1. Documents for Test Cases
2. Documents for Test Plans

Every testing procedure must include certain papers and file systems:

- Defect/Bug
- Reproduce
- Module
- Frequency
- Bug Log Number
- Status/Occurrence
- Screenshots
- Platforms
- Date Time

Task	Estimated Duration
Functionality Testing	3 days
Performance Testing	5 days
Compatibility Testing	7 days

Table 7: Testing Phase

g. Pre-Production (Alpha/Beta Release)

The Alpha/Beta Release is a wonderful approach to learn about user experience and acceptance of our product. If we made any problems after completing the testing phase, we may discover them using pre-production techniques before moving on with the main production. We need to produce a report for the Alpha/Beta Release's output; if the rework is required, complete it for our product and ready it for Main Production/Main Release.

Task	Estimated Duration
Alpha/Beta Release	13 days
End User Testing	13 days

Table 8: Pre-Production Phase

h. Main production

We need to prepare a short movie that looks like a trailer for our game and take some appealing screenshots during the creation stage. Finally, correctly deploy the game, press publish, and make it available in the relevant shops! We prepare a release note, privacy rules, and terms and conditions of use for our product, and include them in a well-organized main marketing document.

Task	Estimated Duration
Main Game Release	10 days
Reports and Documentation	23 days

Table 9: Main Production Phase

3.4. Survey Result

3.4.1. Pre-Survey Result

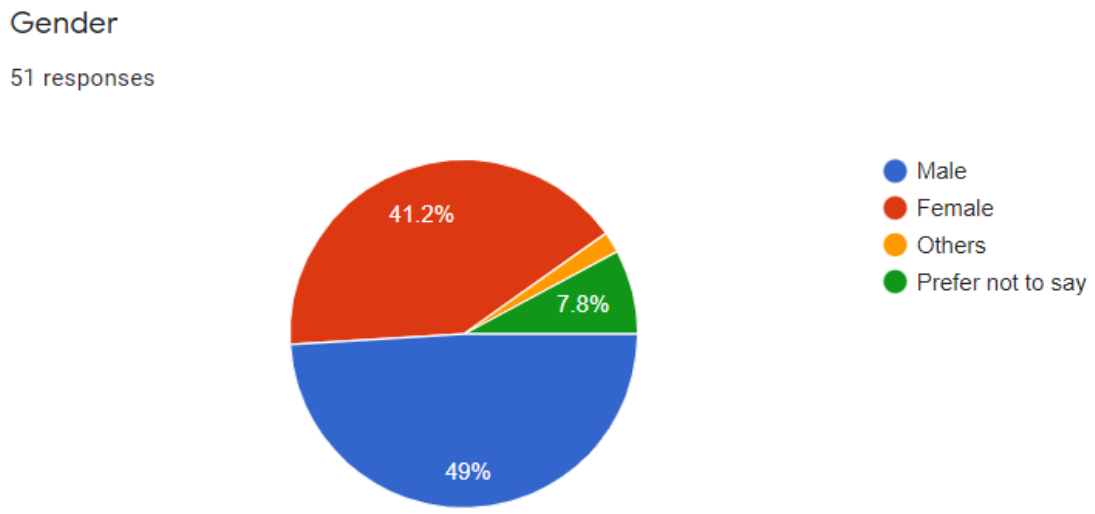


Figure 31: Pre-Survey Result 1

This survey shows the gender of people who took part in the survey.

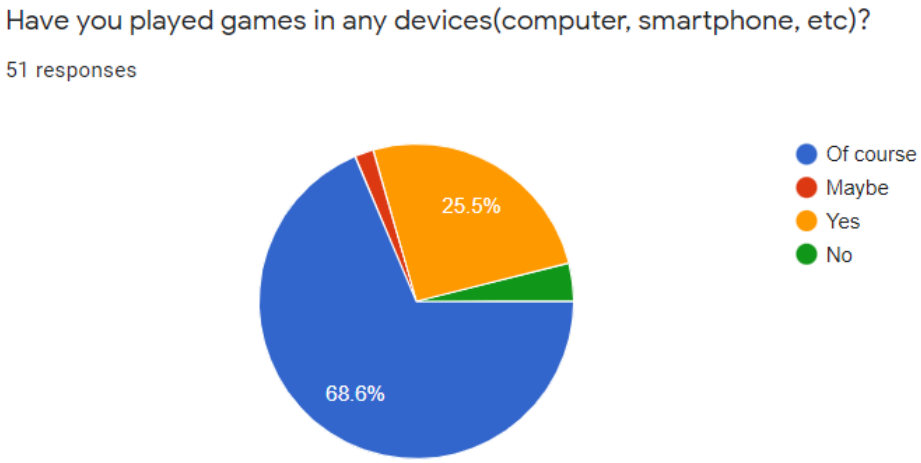


Figure 32: Pre-Survey Result 2

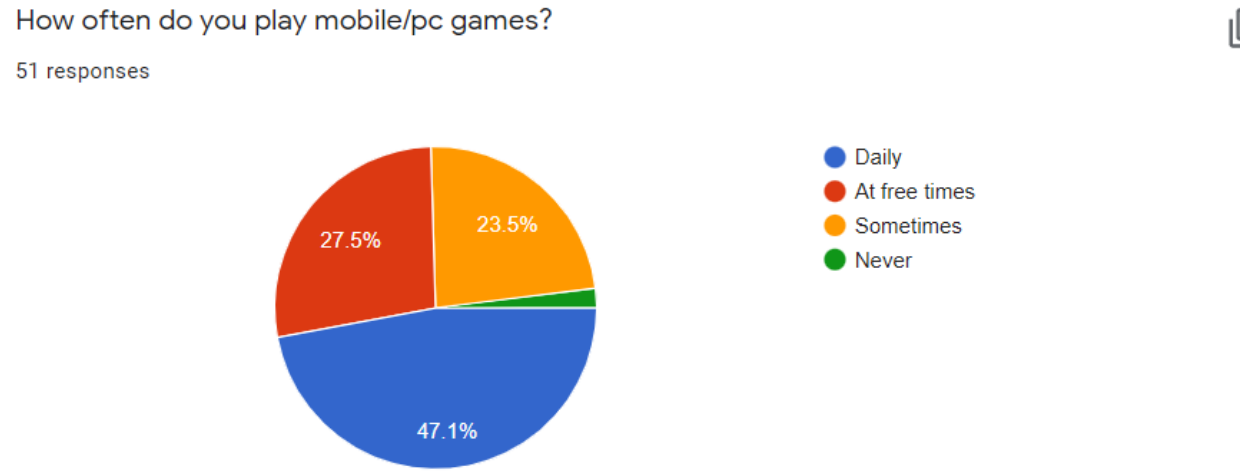


Figure 33:Pre- Survey Result 3

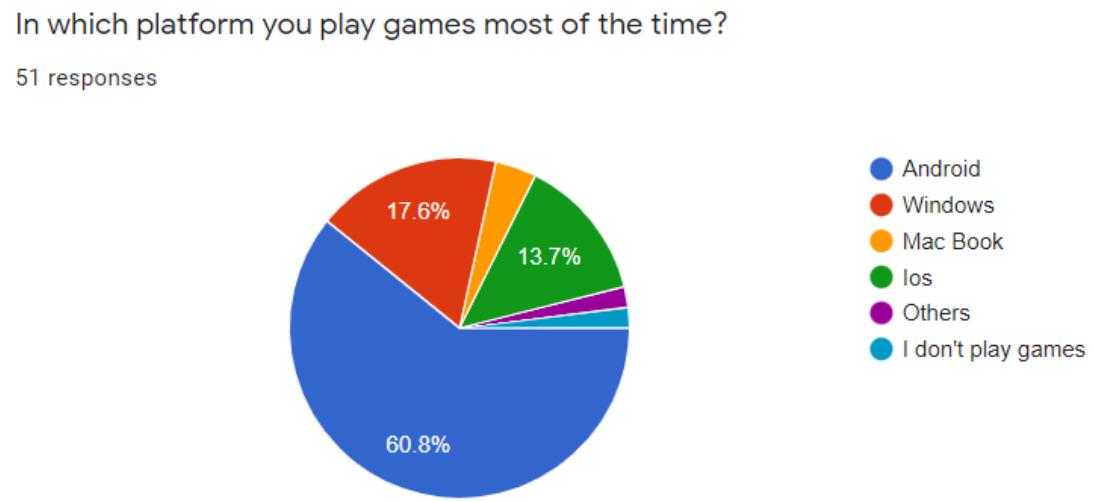


Figure 34:Pre- Survey Result 4

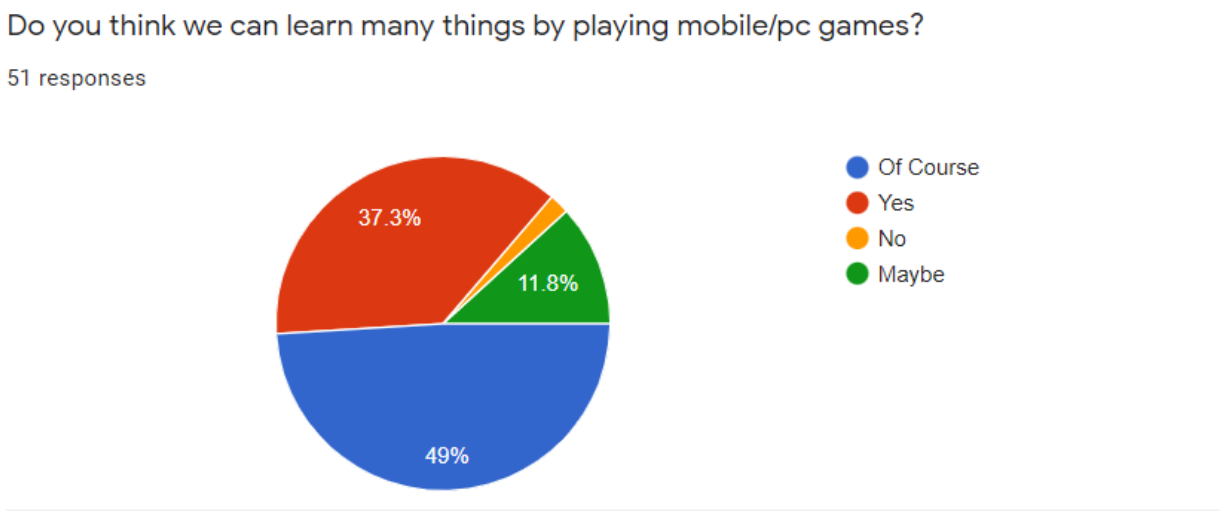


Figure 35:Pre- Survey Result 5

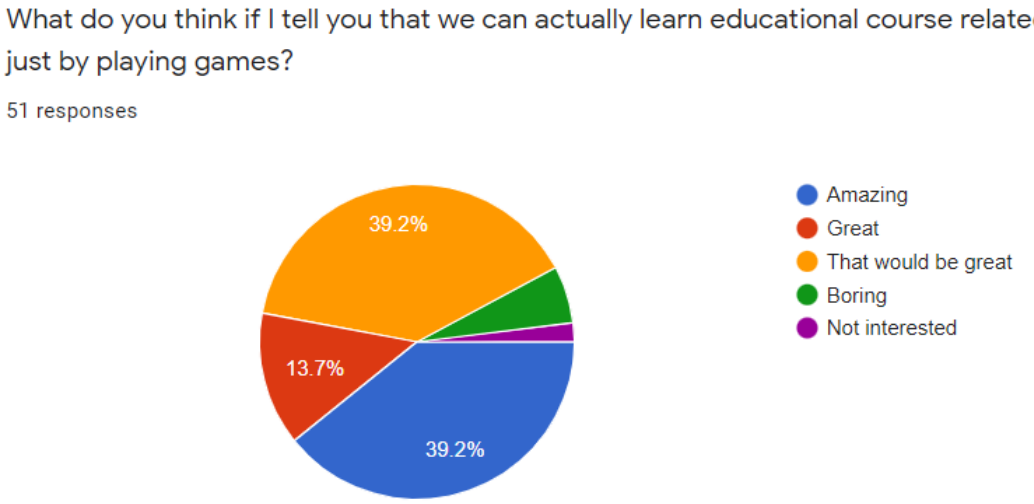


Figure 36: Pre-Survey Result 8

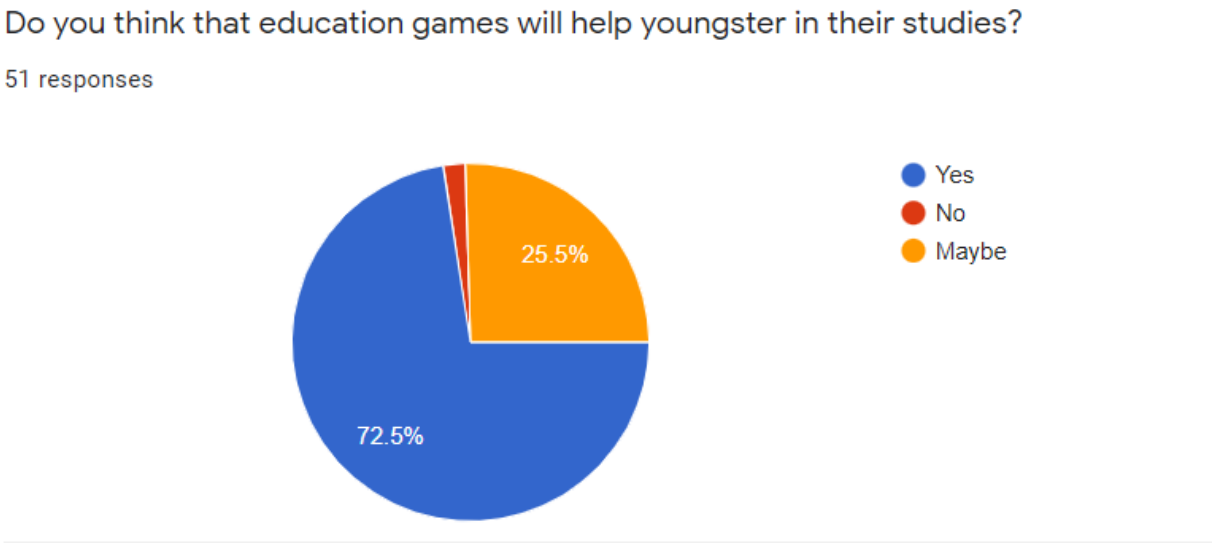


Figure 37:Pre- Survey Result 7

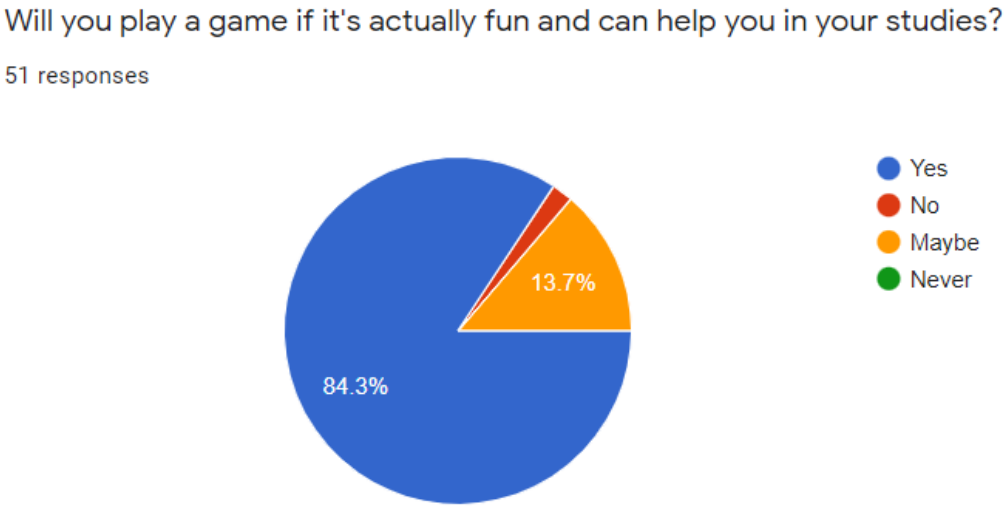


Figure 38: Pre-Survey Result 8

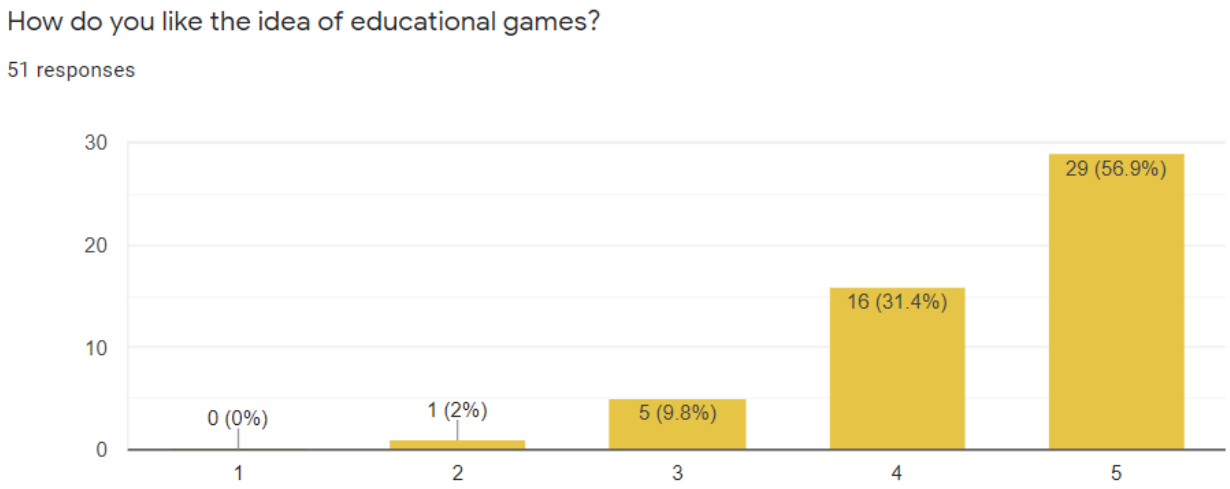


Figure 39:Pre- Survey Result 9

3.4.2. Post-Survey Result

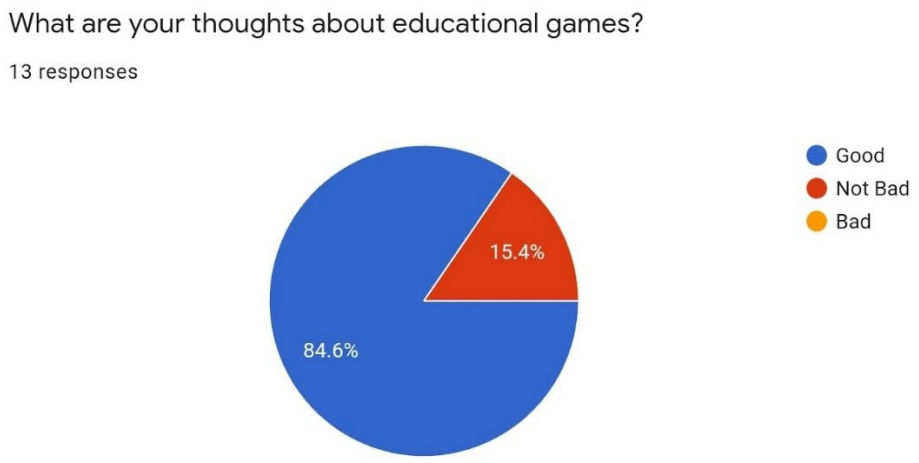


Figure 40: Post-Survey Result 1

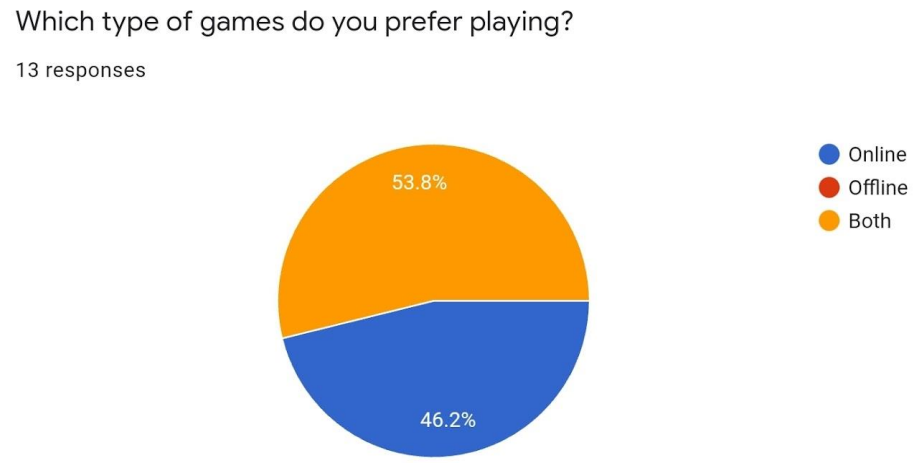


Figure 41: Post-Survey Result 2

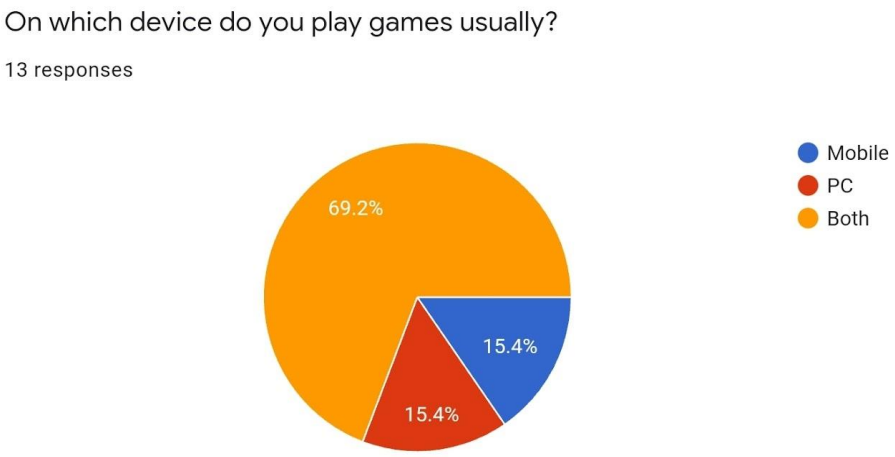


Figure 42: Post-Survey Result 3

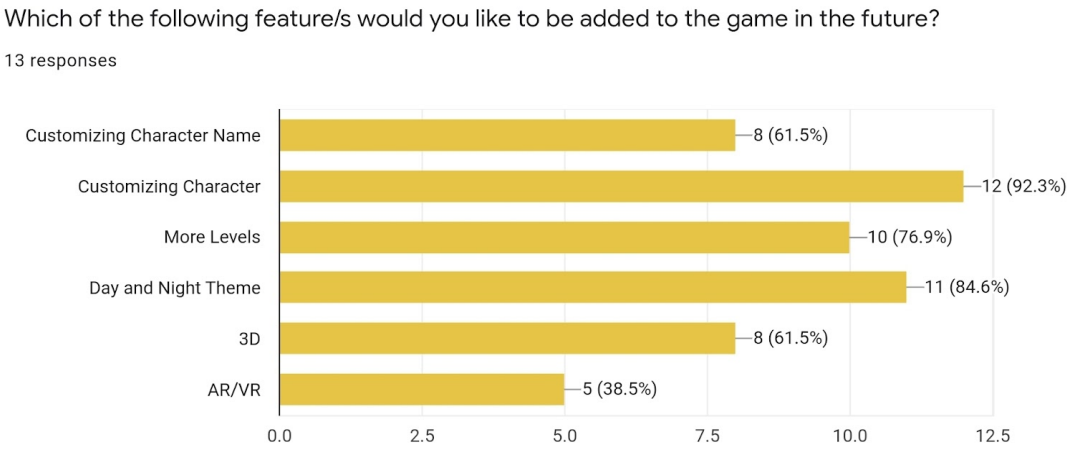


Figure 43: Post-Survey Result 4

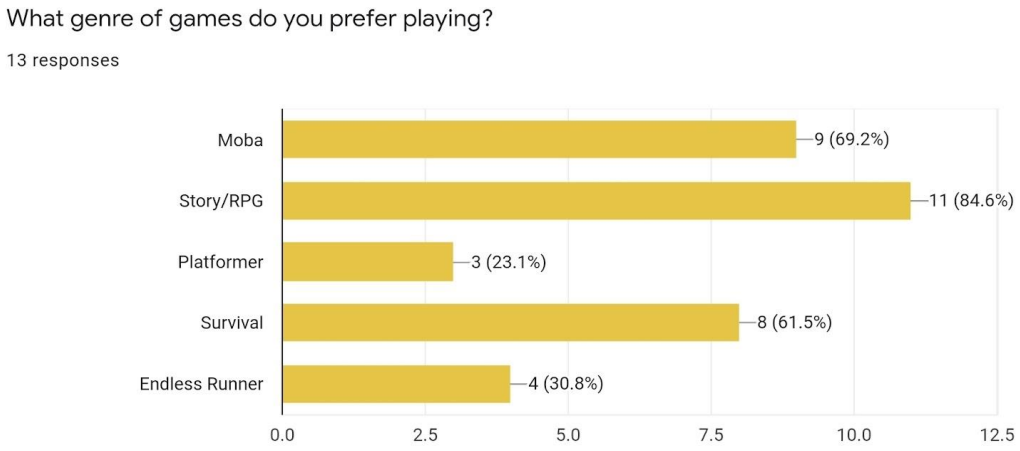


Figure 44:Post-Survey Result 5

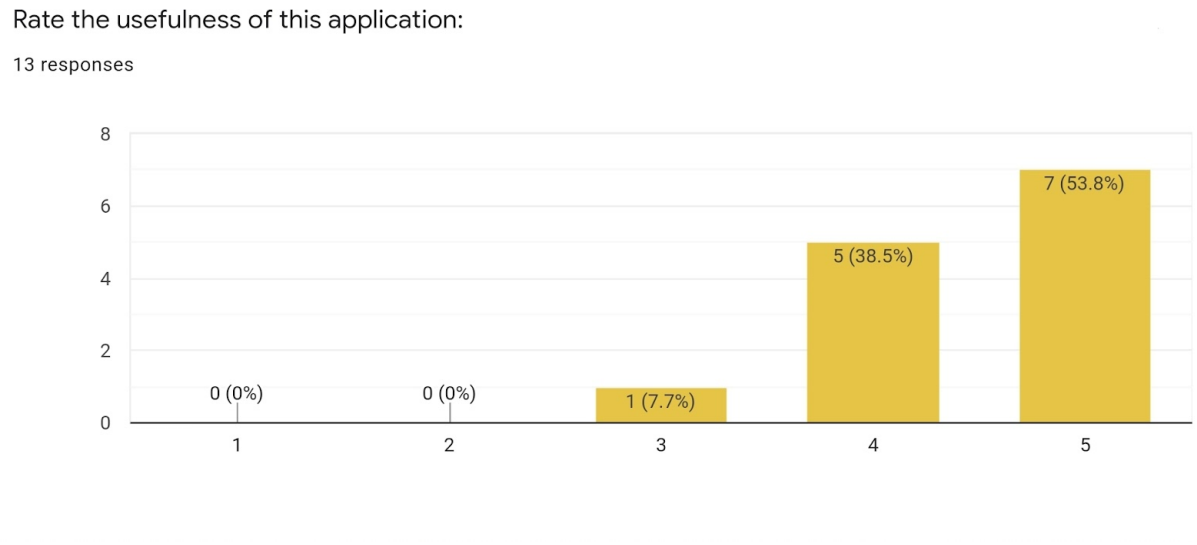


Figure 45: Post-Survey Result 6

3.5. Requirement Analysis

3.5.1. Overall requirements

As this project is about an educational game application, it is a 2D game that can be played on devices with lower frame rates. This application is compatible with the majority of devices. The following are some requirements for running this application:

- Windows 7 and above
- MacBook

3.5.2. Generalized list of requirements

3.5.2.1. Functional

- User can customize the settings
- User can quit the game
- User can play the game character
- User can use hover in UI of the game
- User can use the mechanics of the character like movement, jump, shooting, etc.

3.5.2.2. Non-Functional

- User can play different levels to play
- User can see the key score.
- User will be message in every tutorial and game levels which help the user for completing levels

3.5.2.3. Useability

- The game will go to next level after the player save the girl
- User can use the mechanics of the character as he or she wants
- User can read the messages and learn
- User can listen to the instruction given by the AI of the game

3.6. Design

3.6.1. Mind Map

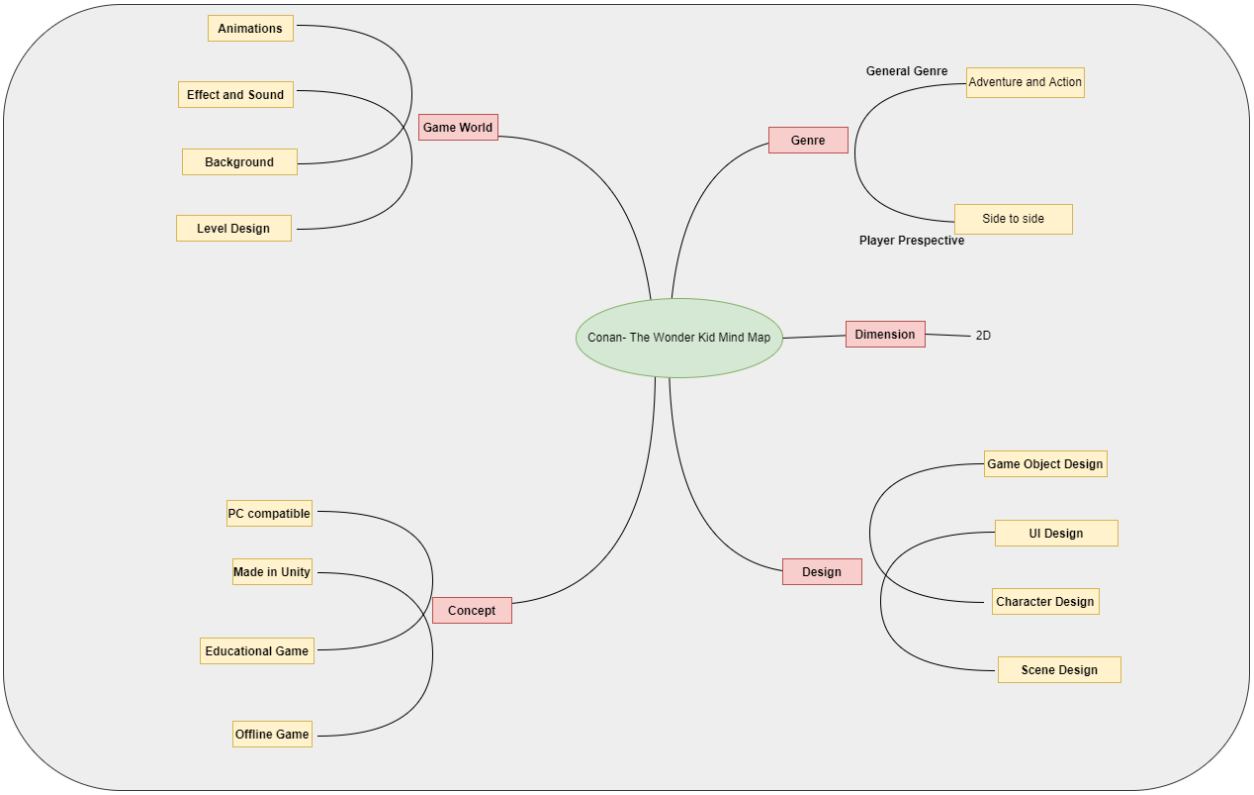


Figure 46: Mind Map

3.6.2. Work Breakdown Structure

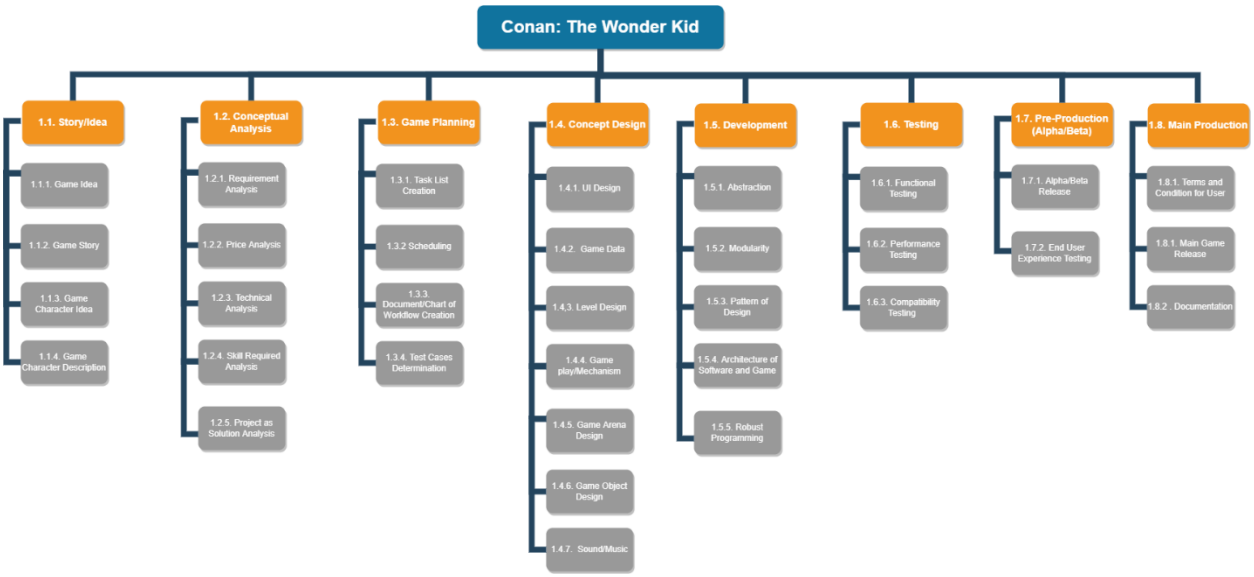


Figure 47: Work Breakdown Structure

3.6.3. Use Case Diagram

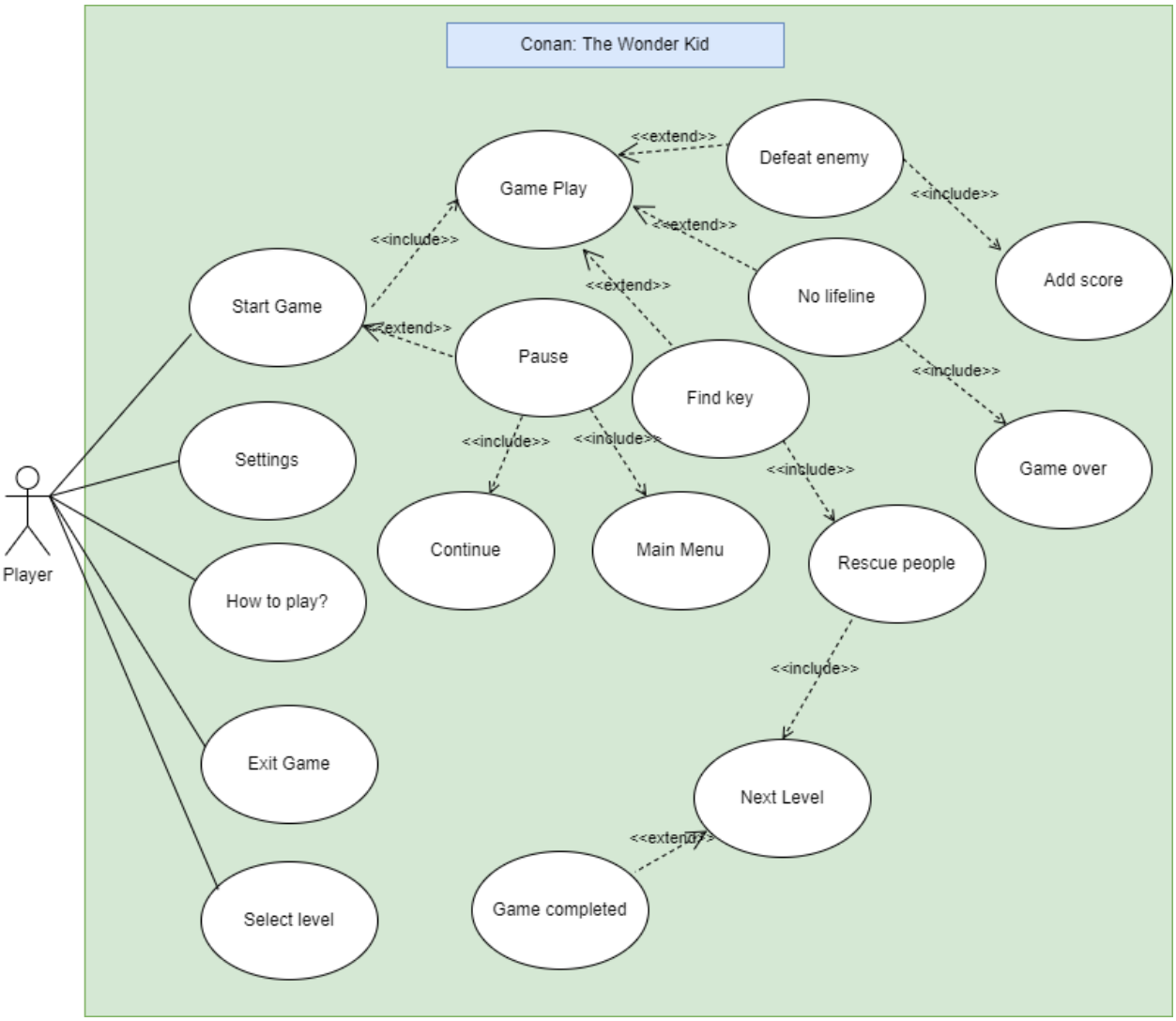


Figure 48: Use Case Diagram of player and the system

3.6.4. Wireframes

q Wireframes are useful for a variety of things, including connecting the site's information architecture to its visual design by displaying routes between pages. Clarify how specific types of information should be shown on the user interface in a consistent manner. Determine the interface's intended functionality. The wireframe of my game is given below:

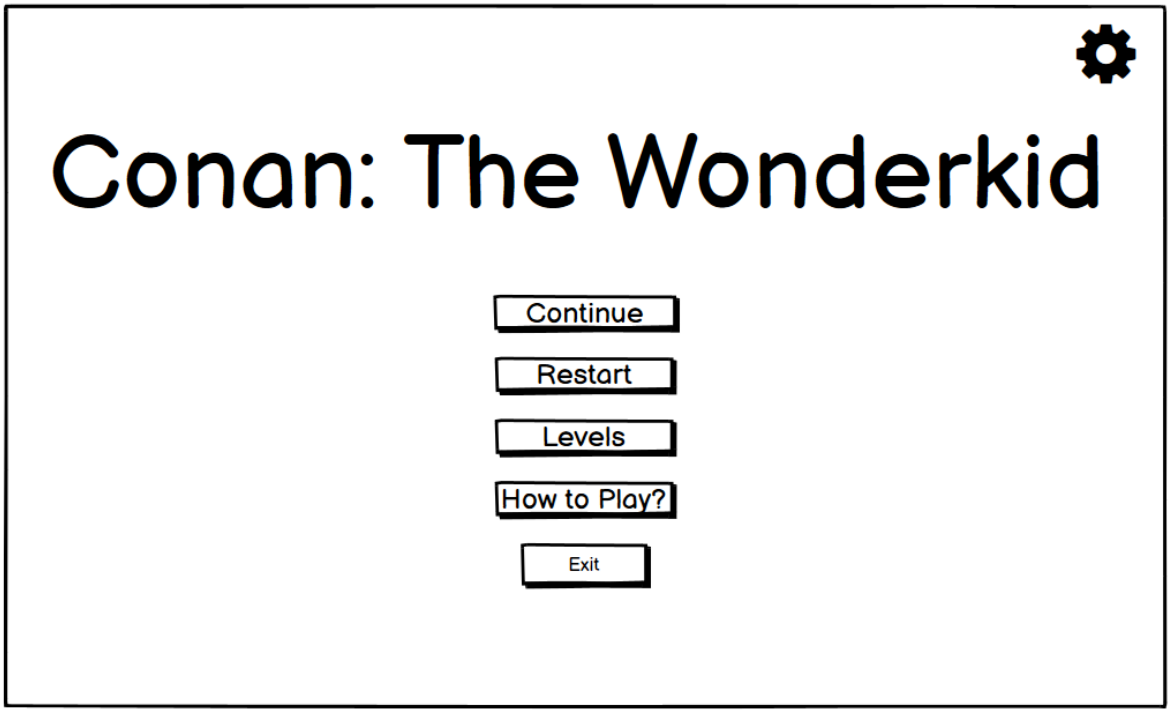


Figure 49: Wireframe of the Main Menu

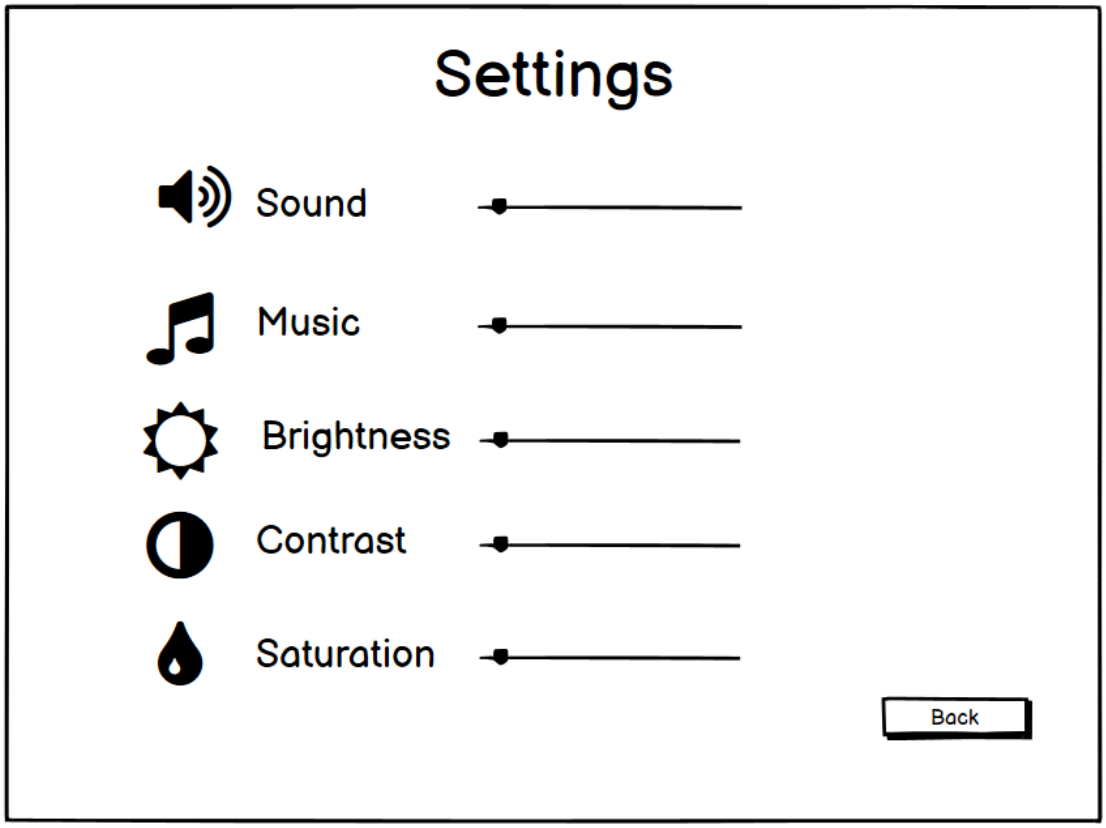


Figure 50: Wireframe of Settings Panel

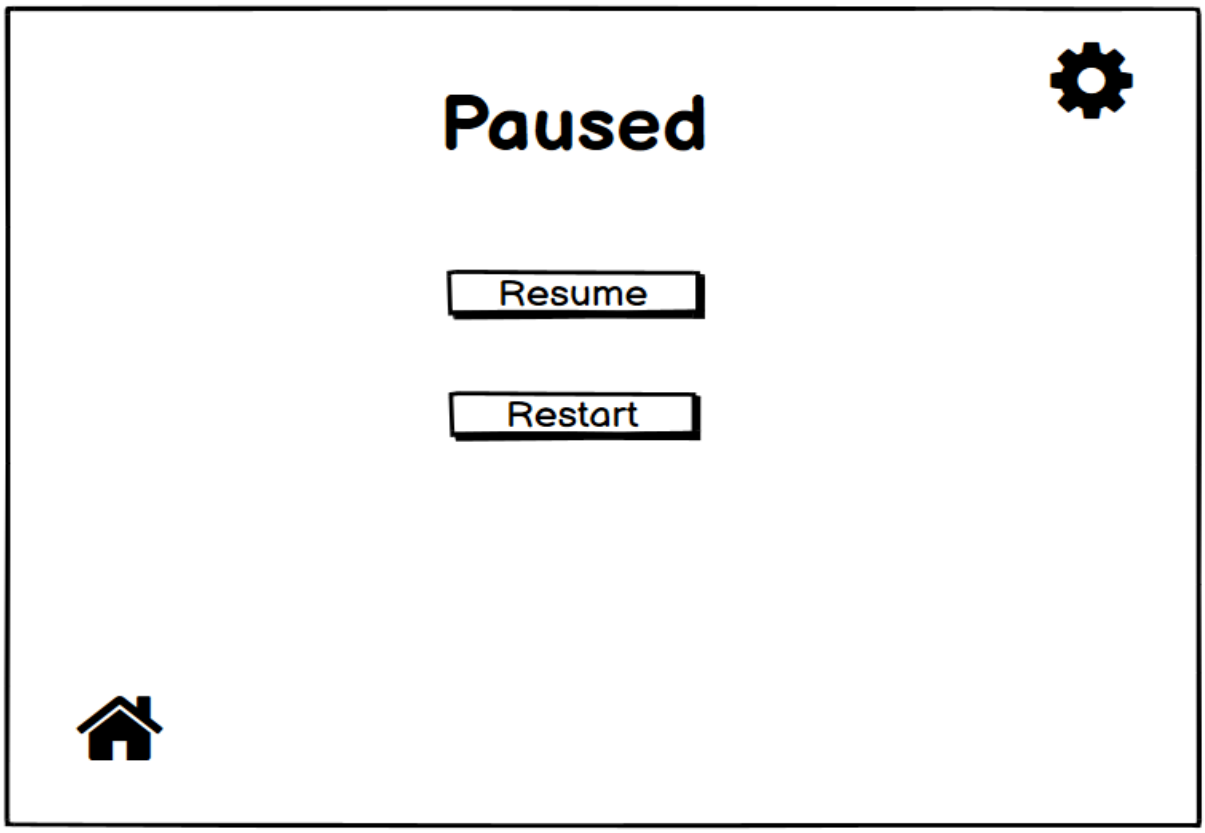


Figure 51: Wireframe for pause section

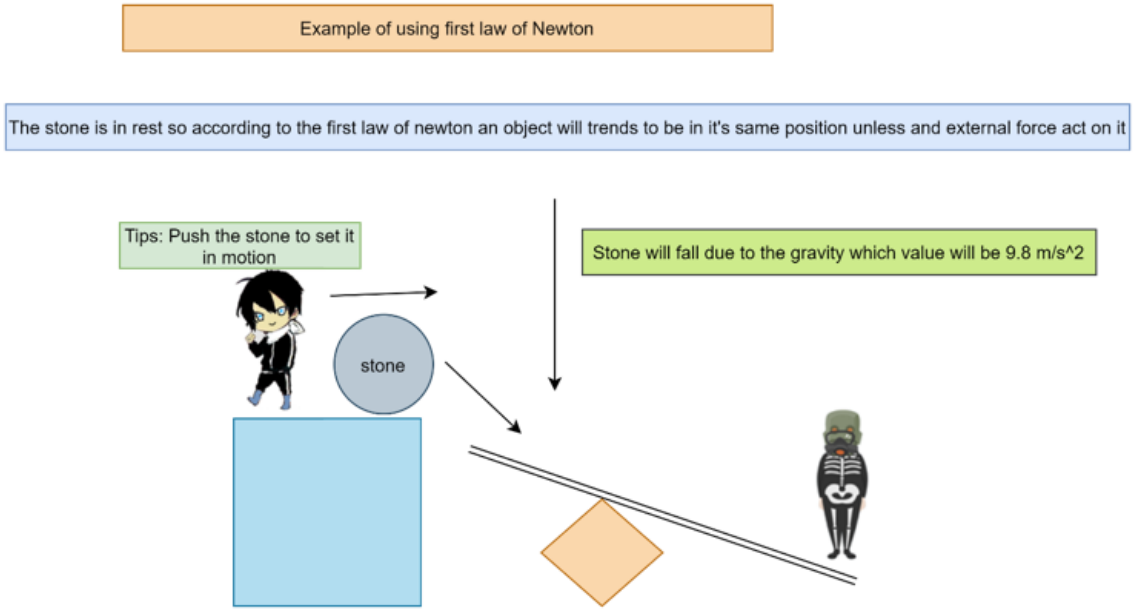


Figure 52: Wireframe of Tutorial Scene

Figure 53: Wireframe of Game World

3.6.5. Graphical Representation

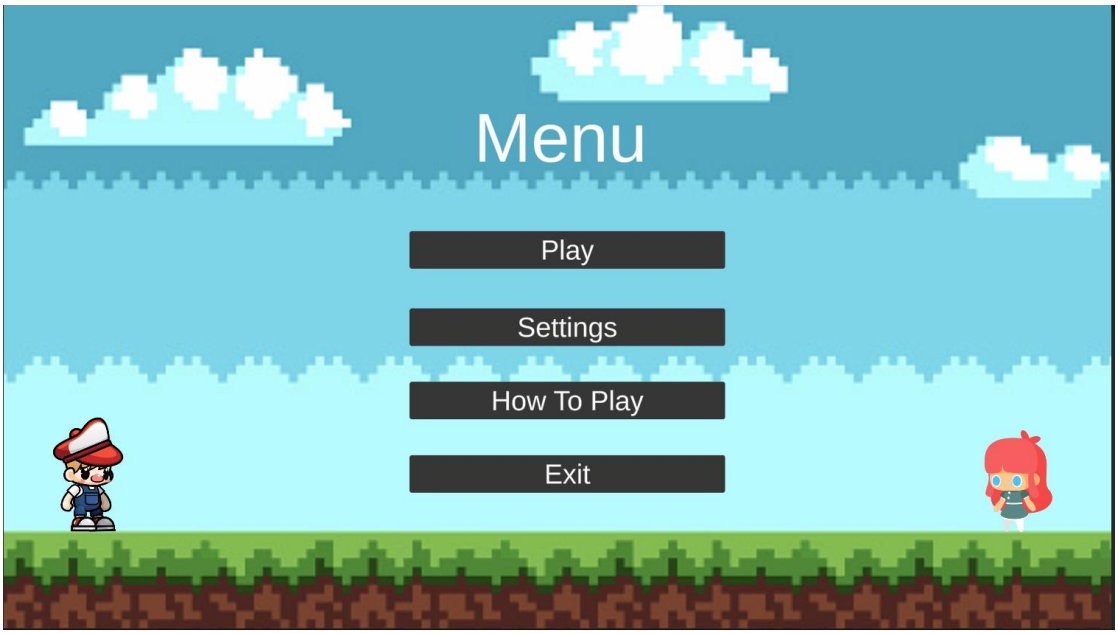


Figure 54: Main Menu

Figure 55: Settings Panel

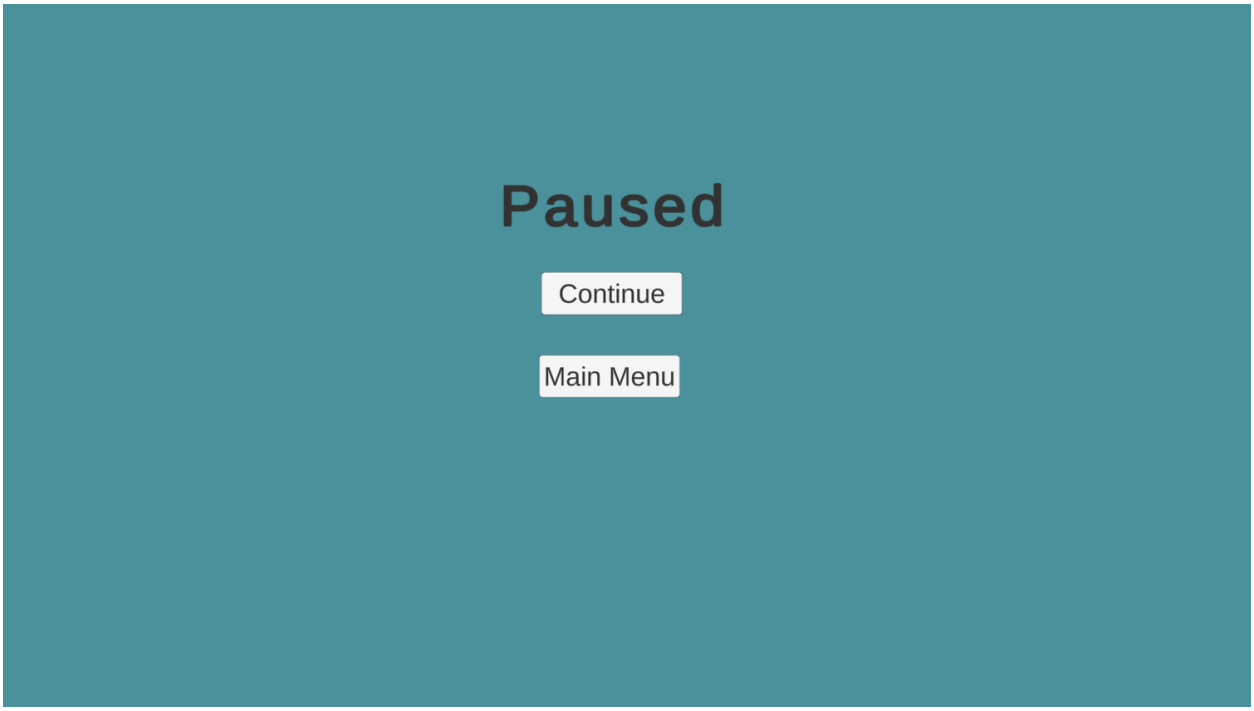


Figure 56: Paused Panel

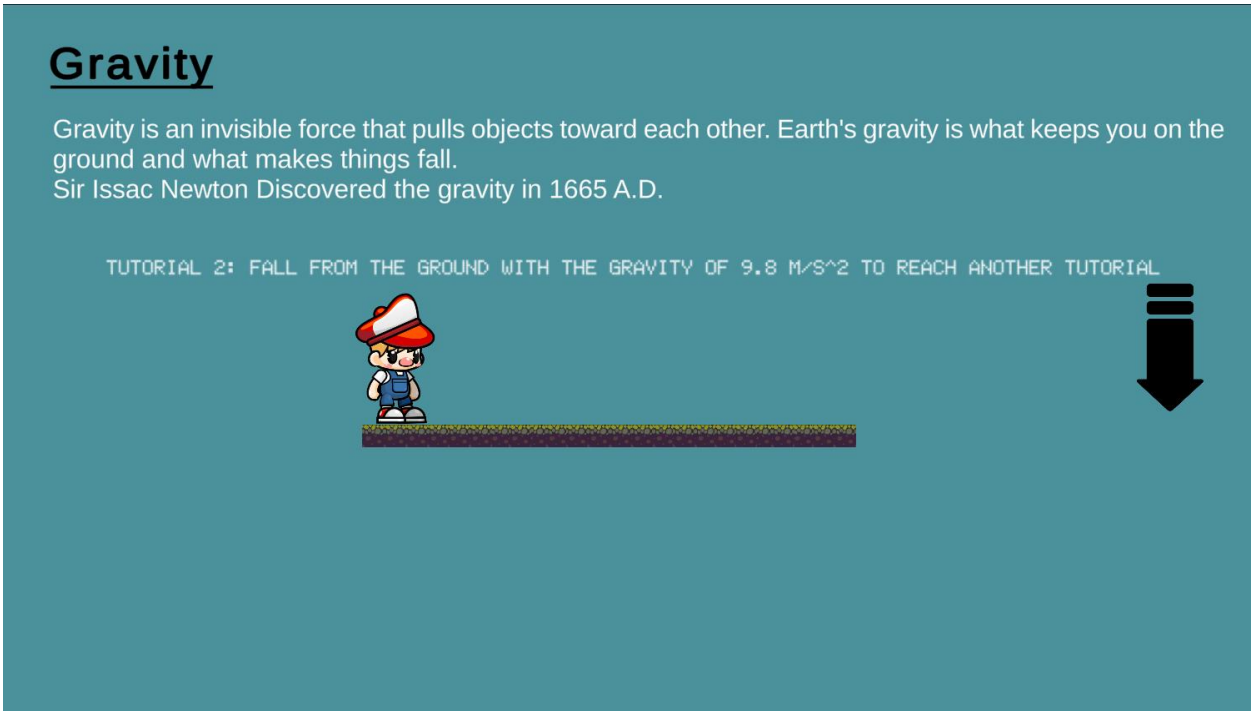


Figure 57: Tutorial Scene

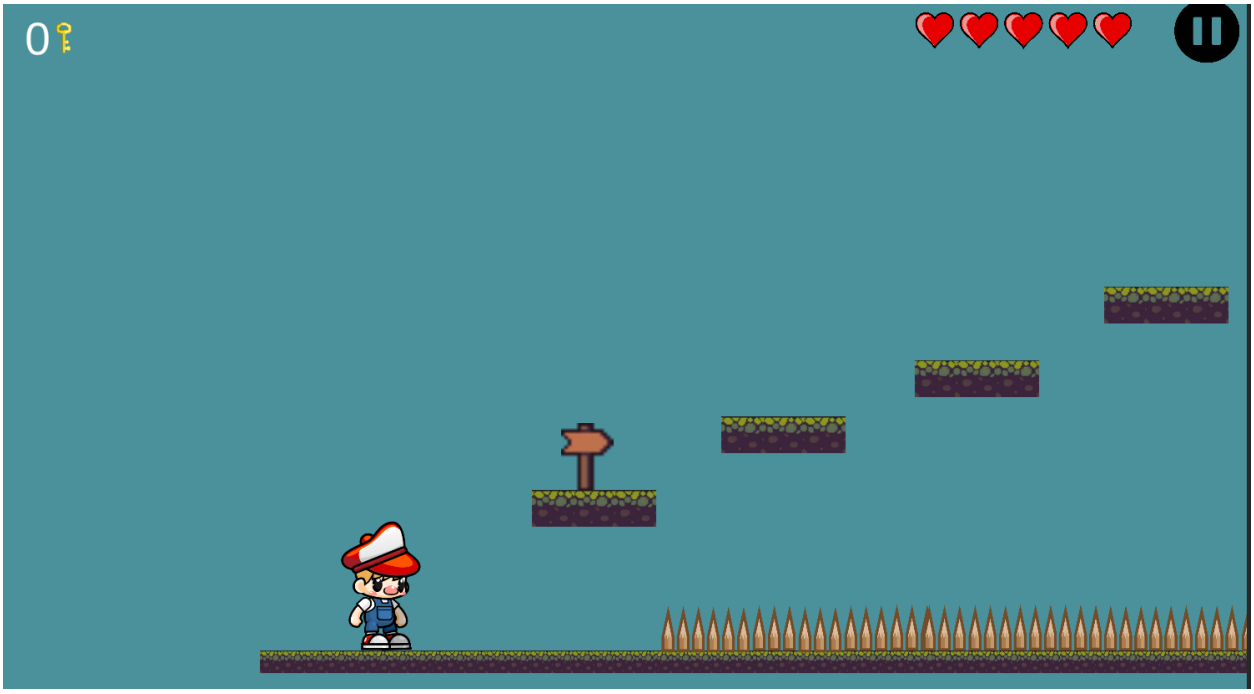


Figure 58: Game World

3.6.6. Overall System Overview

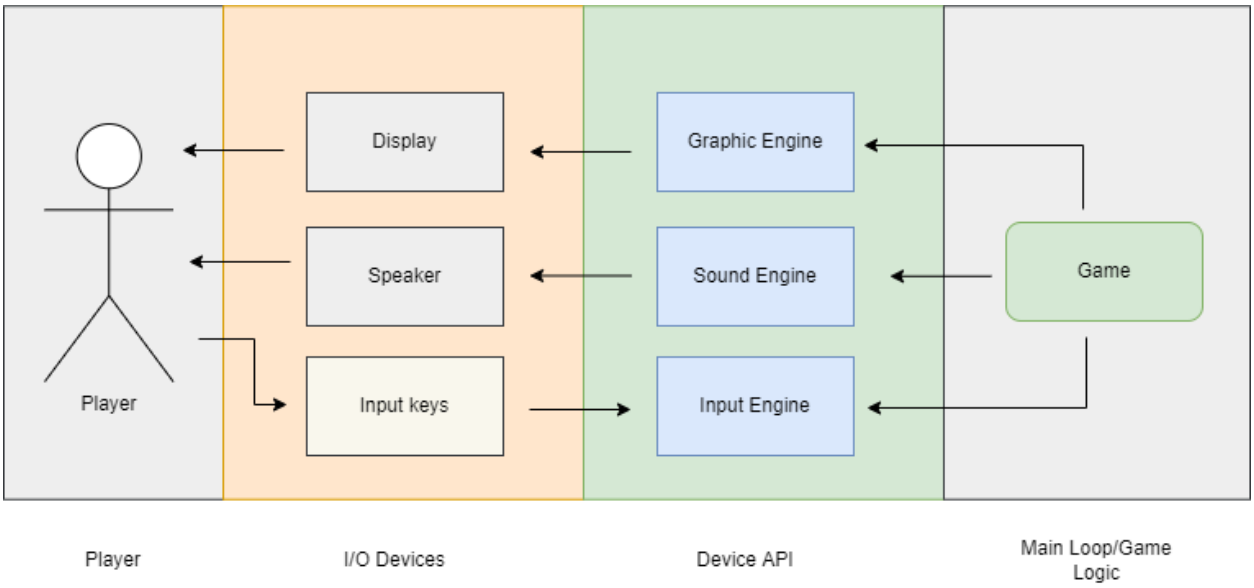


Figure 59: Overall System Overview

3.6.7. UML Diagrams

3.6.7.1. Play Button

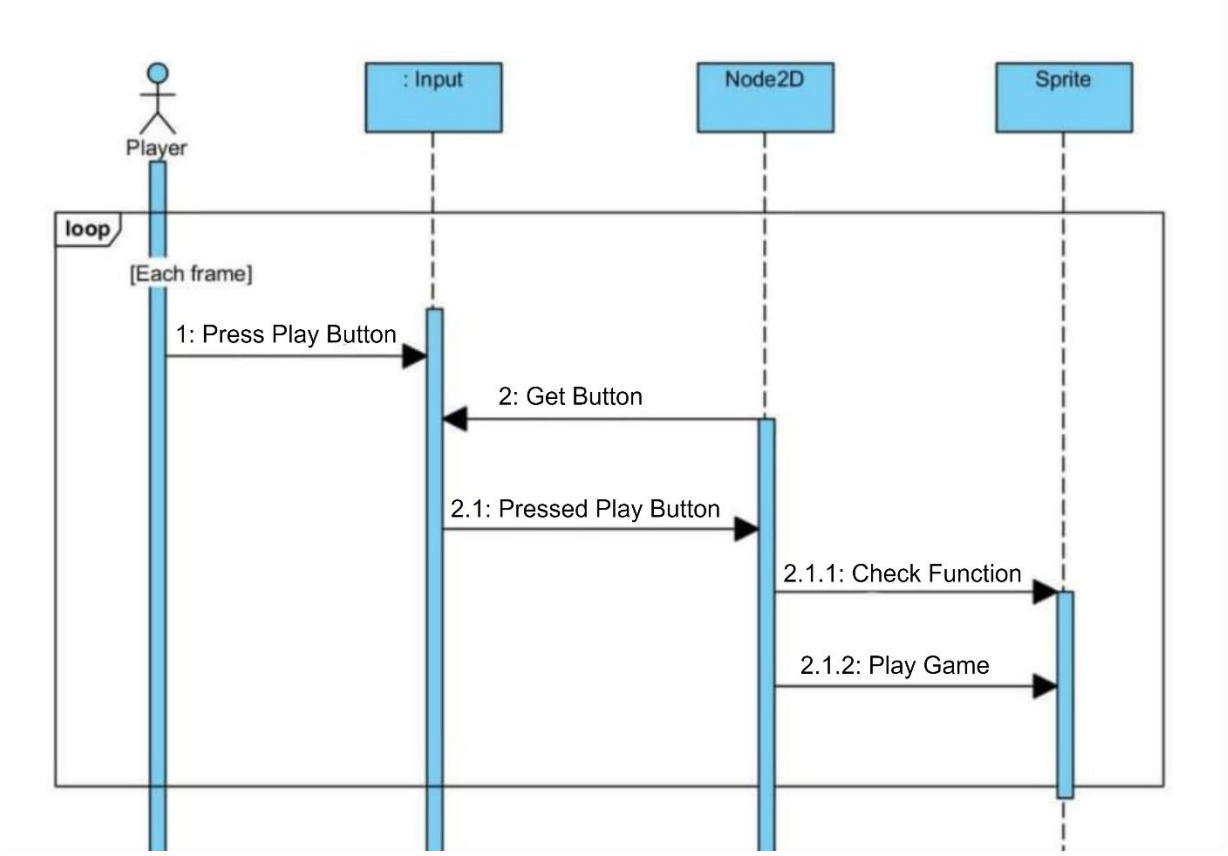


Figure 60: Sequence Diagram for Play Button

3.6.7.2. Settings Button

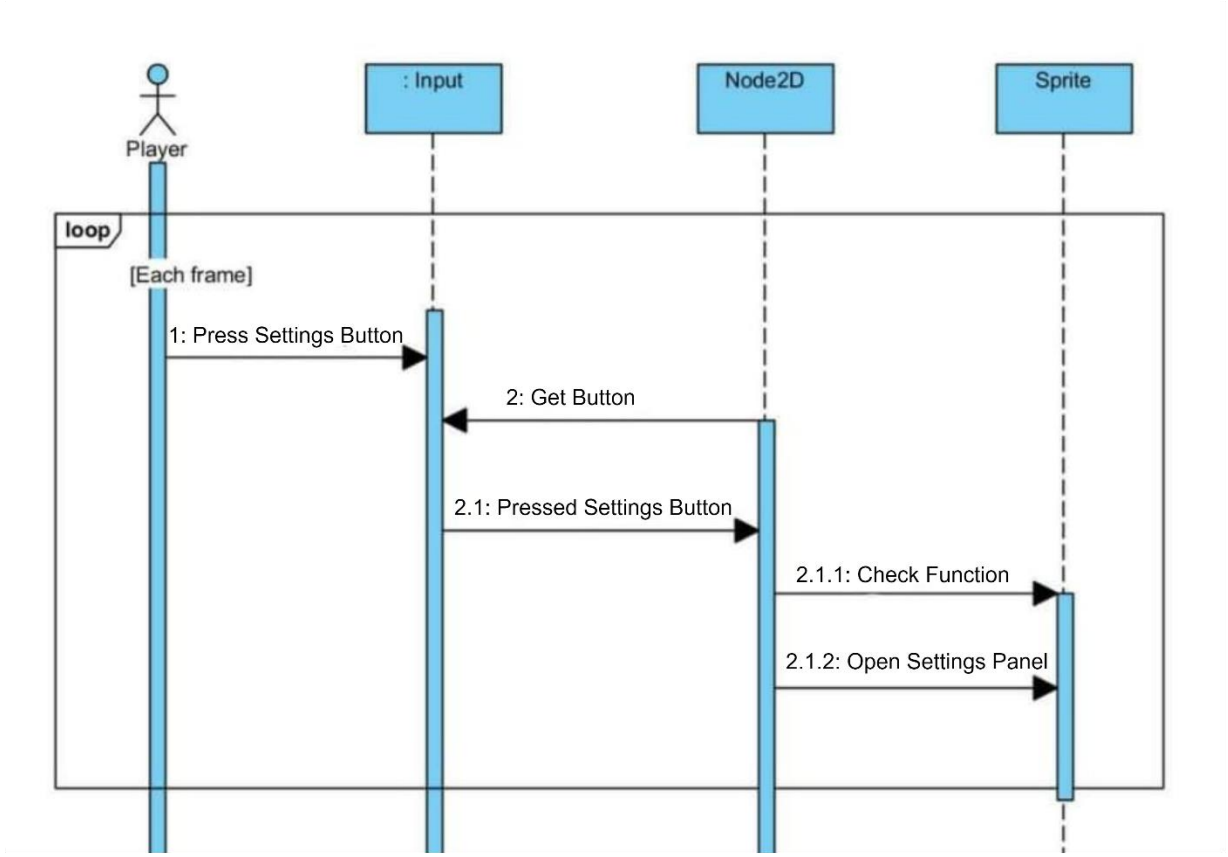


Figure 61: Sequence Diagram for Settings Button

3.6.7.3. Settings Panel

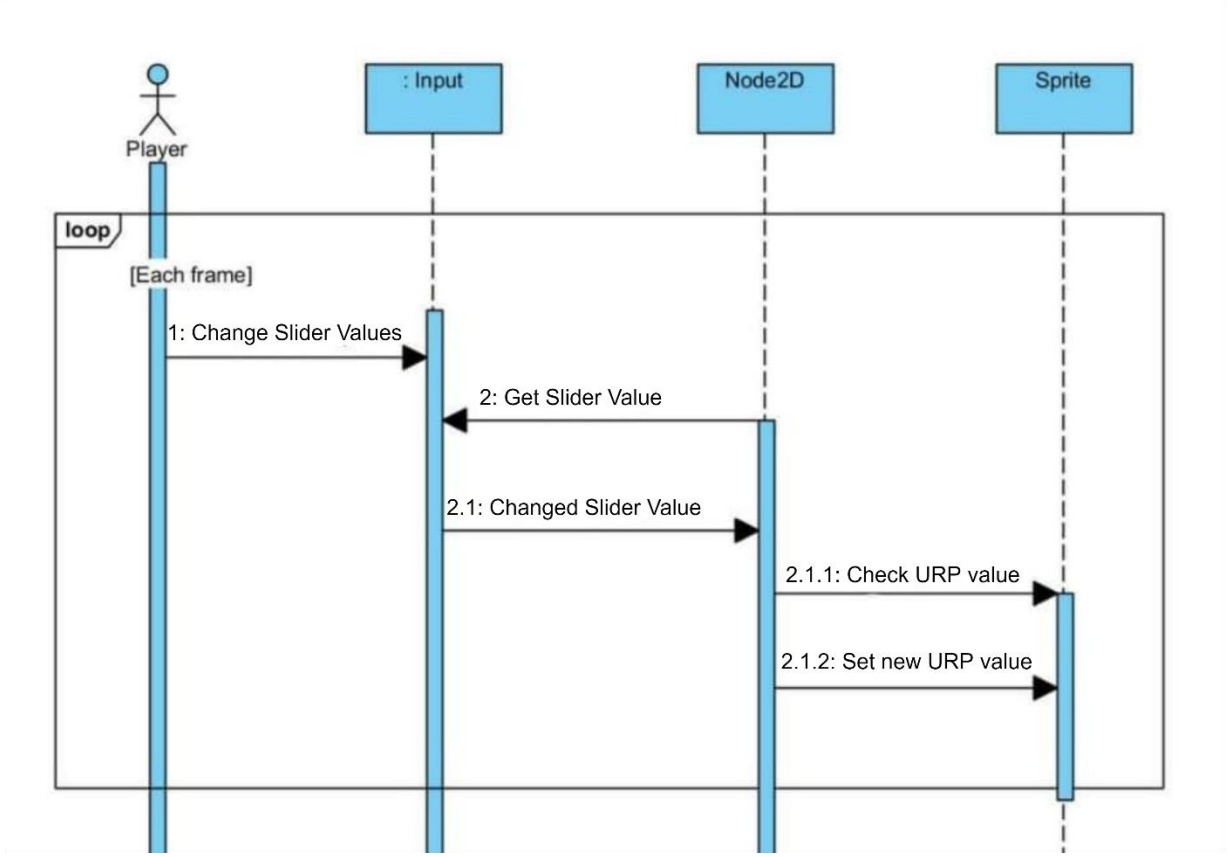


Figure 62: Sequence Diagram for Settings Panel

3.6.7.4. Exit button

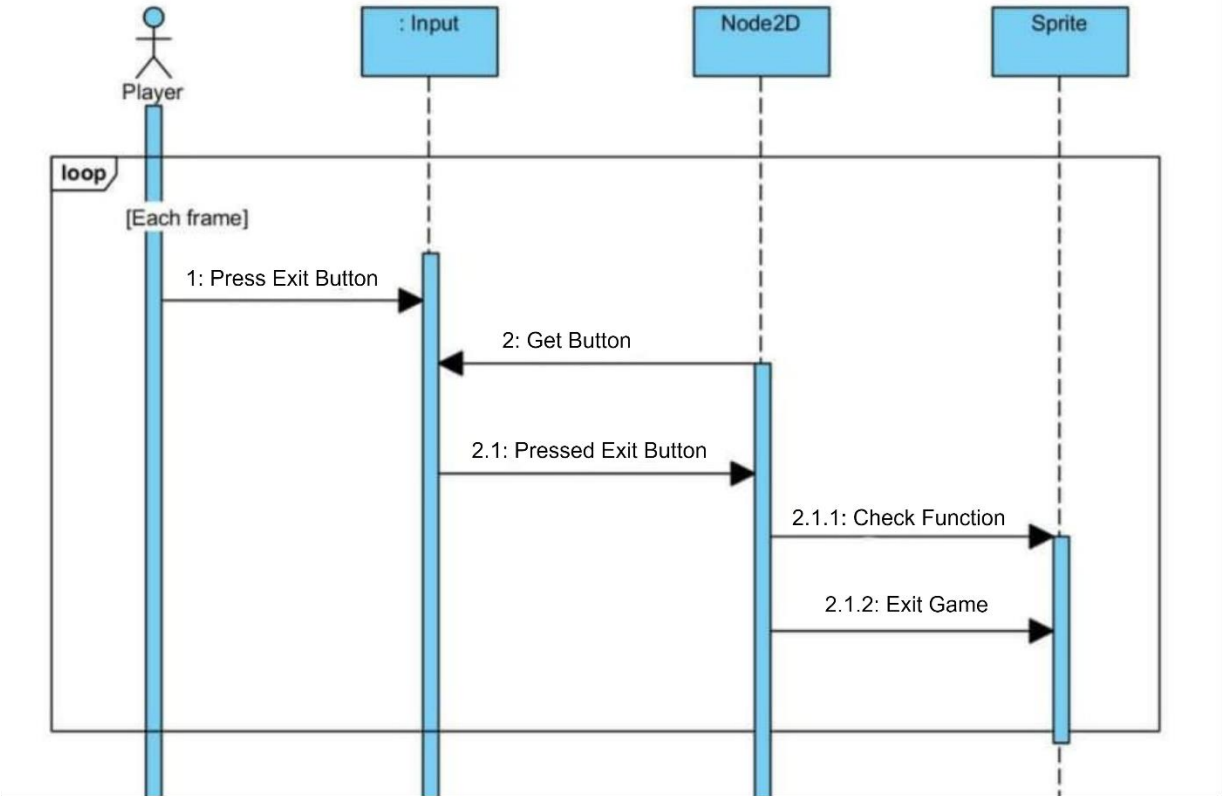


Figure 63: Sequence Diagram for Exit Button

3.6.7.5. Pause button

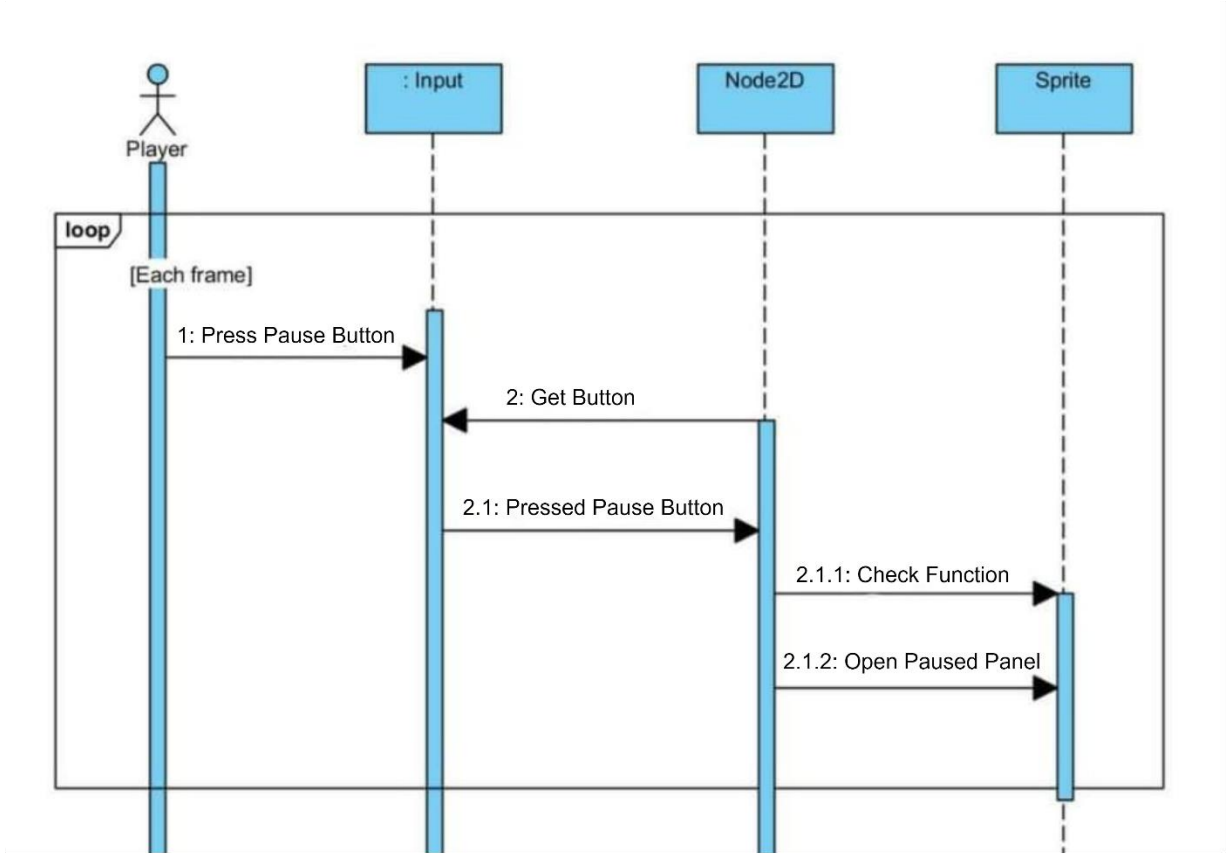


Figure 64: Sequence Diagram for Pause Button

3.6.7.6. Continue Button

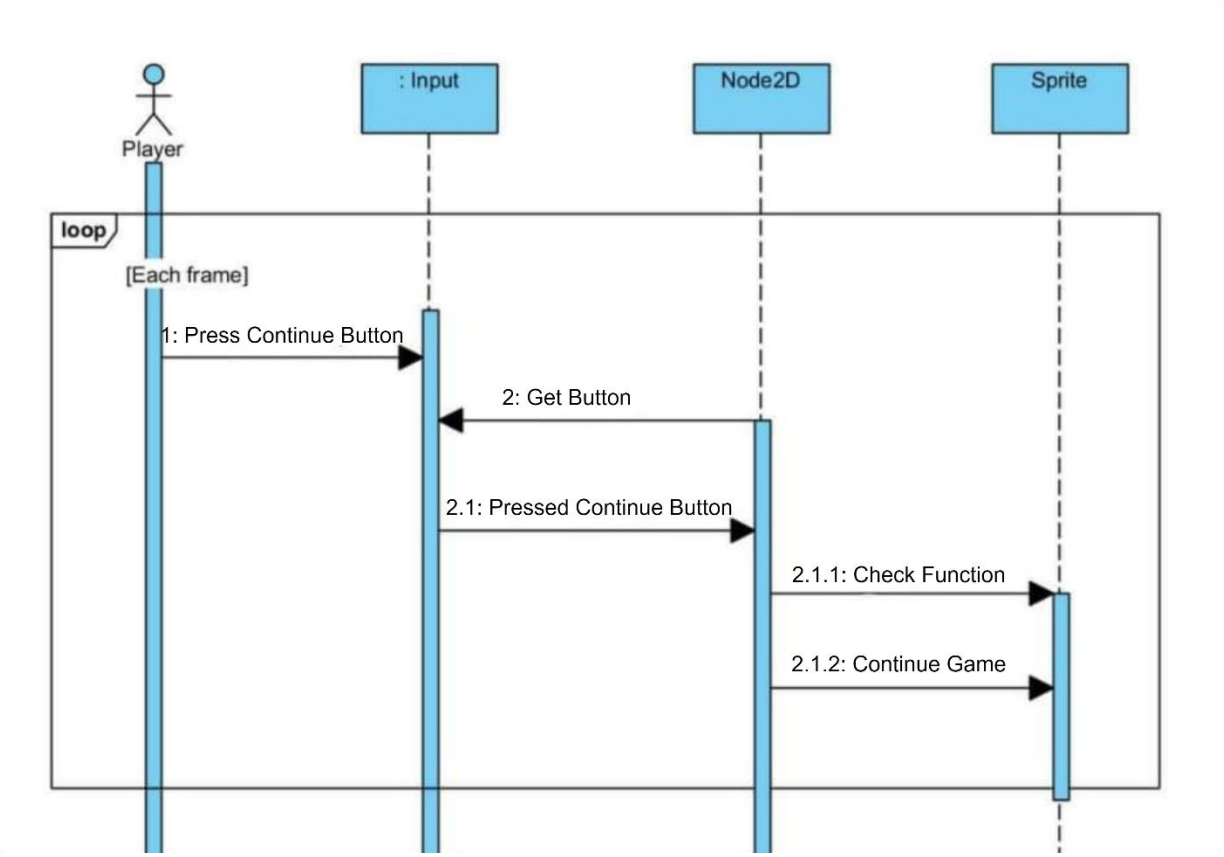


Figure 65: Sequence Diagram for Continue Button

3.6.7.7. Main Menu Button

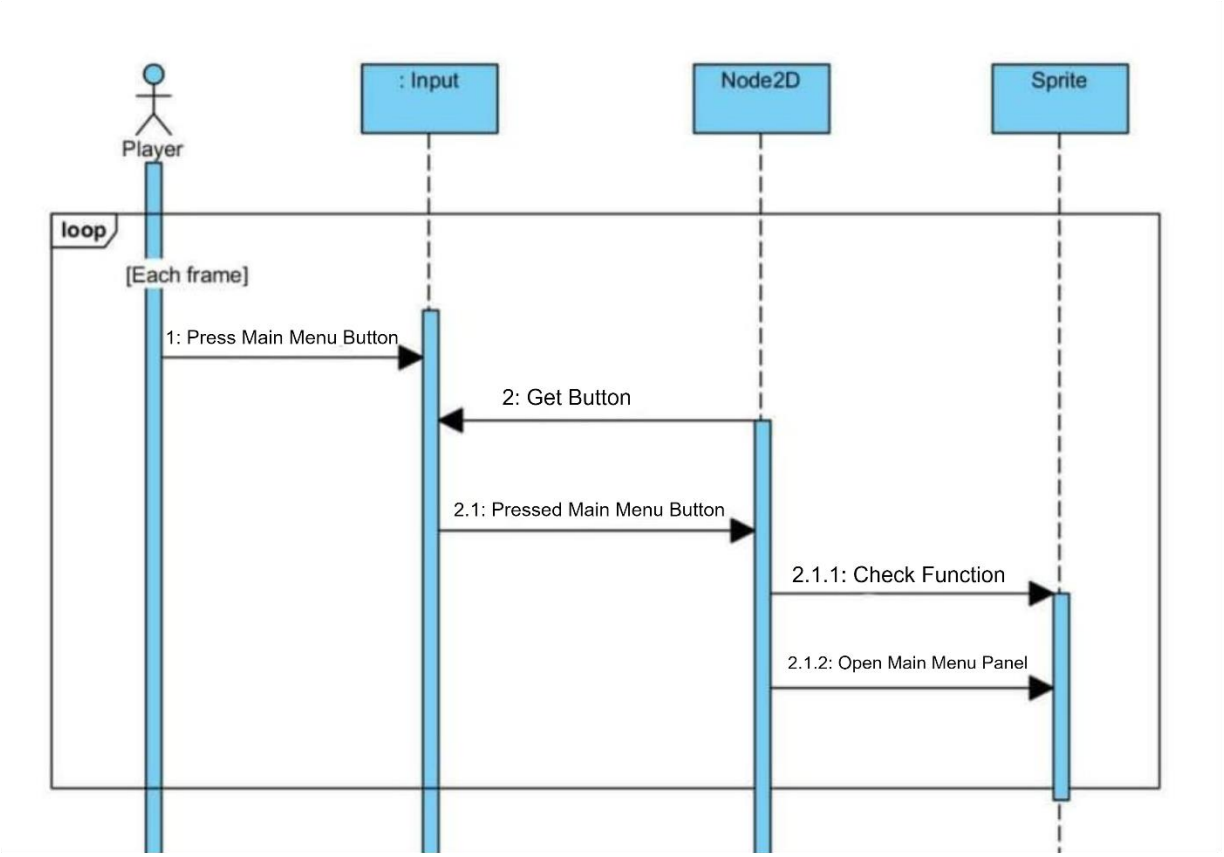


Figure 66: Sequence Diagram for Main Menu Button

3.6.7.8. Player movement and jump

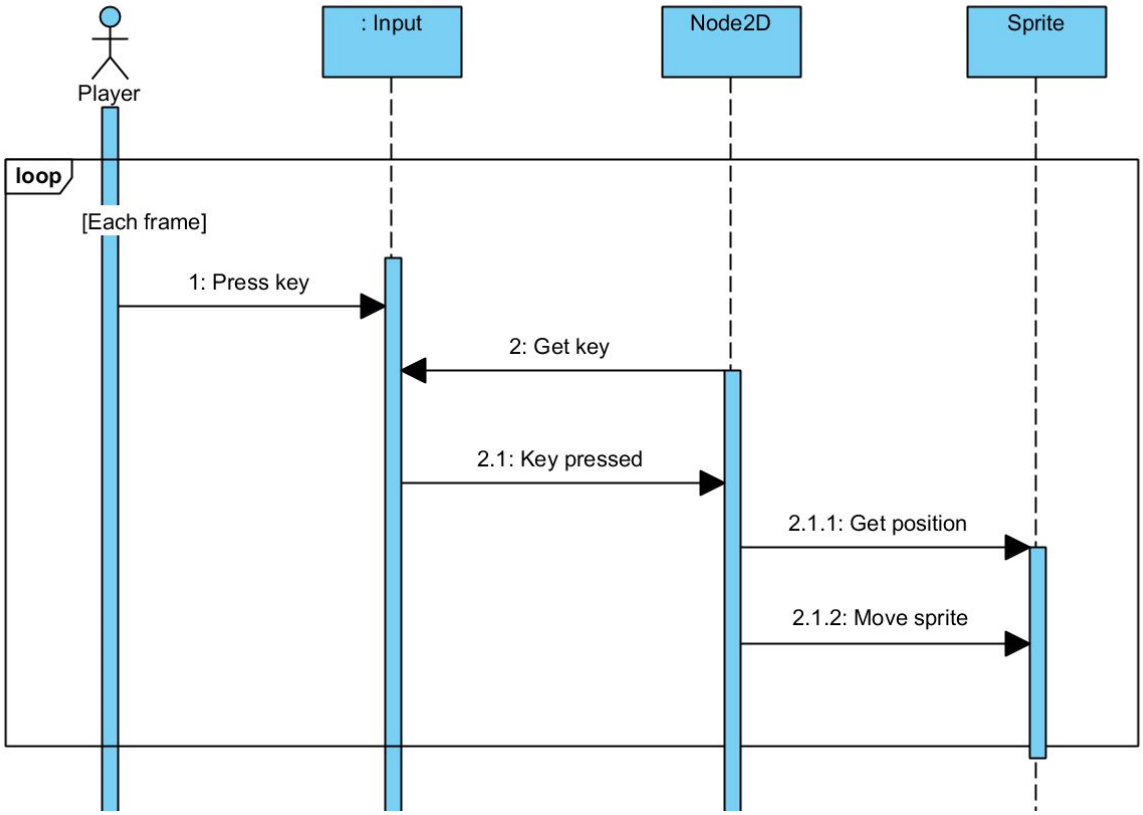


Figure 67: Sequence Diagram for Movement and Jump

3.6.7.9. Shooting

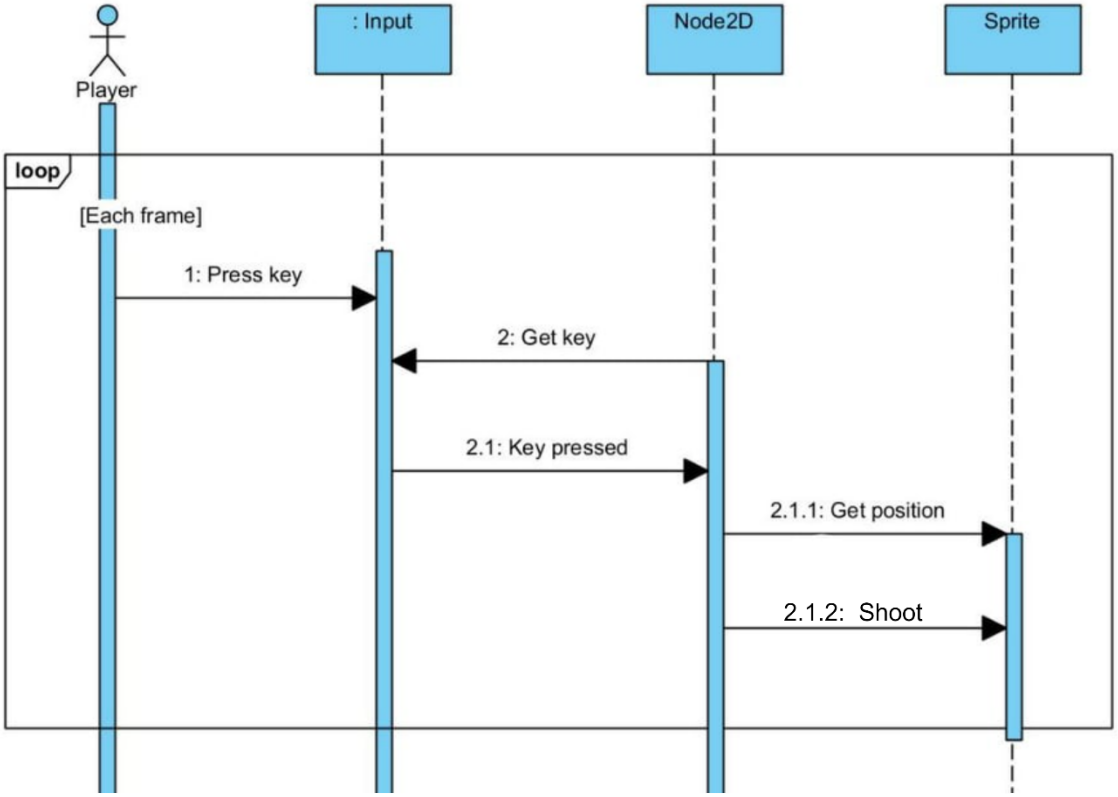


Figure 68: Sequence Diagram for Shooting

3.6.8. Game Loop Diagram

Figure 69: Game Loop Diagram

3.6.9. Game-Mechanics Loop Diagram

Figure 70: Game-Mechanics Loop Diagram

3.7. SRS Report

3.7.1. Introduction

A software requirements specification (SRS) is a document that describes the software system that will be produced. It lays out functional and non-functional requirements, as well as a set of use cases that define how the product must interact with users.

3.7.2. Purpose

Educational games are those that are meant to help individuals learn about certain subjects, broaden concepts, reinforce development, understand a historical event or culture, or educate them in learning a skill while they are playing.

3.7.3. Scope

The purpose of “Conan: the wonder kid” is to make enjoyable as well as educational game for the kids in grade 6 to above. In this game, kids can learn about the Newton’s law of motion and physics. Education game like this is perfect solution for the current situation where parents want their children to study, and children wants to play game.

SRS Link for Appendix

3.8. Implementation

3.8.1. Implementation of Core Feature

3.8.1.1. Implementation of Packages in Unity

A package is a container for any combination of Assets, Shaders, Textures, plug-ins, icons, and scripts that enhance different aspects of your project. Unity packages are newer, more tightly integrated versions of Asset Store packages that can provide a wide range of Unity enhancements (Unity, 2022).

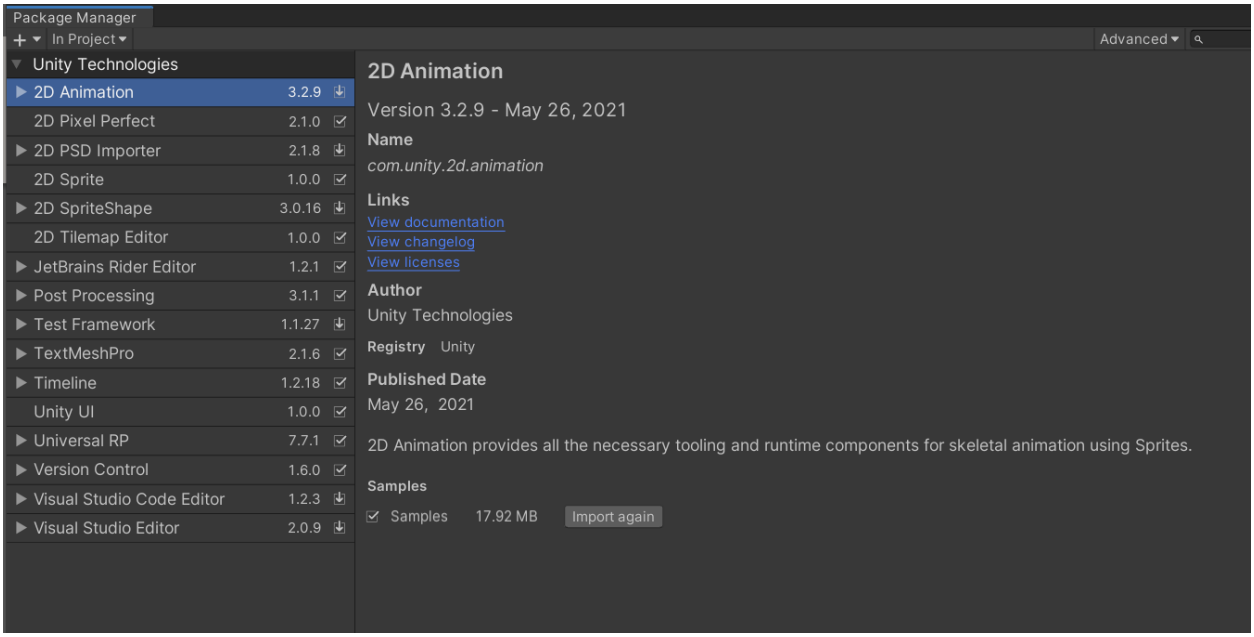


Figure 71: Implementation of Packages in Unity

3.8.1.2. Implementation of Assets in Unity

A Unity asset is a piece of software that you can use in your game or project. An asset can be derived from a file created outside of Unity, such as a 2D model, an audio file, an image, or any of the other file types supported by Unity. You can also create asset types within Unity, such as an Animator Controller, an Audio Mixer, or a Render Texture (Unity, 2022).

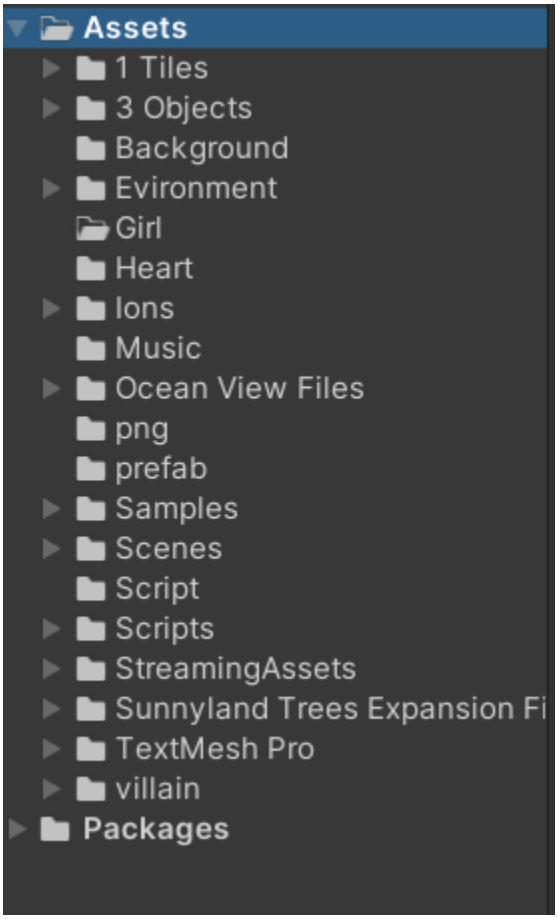


Figure 72: Implementation of Assets in Unity

3.8.1.3. Implementation of animator control and animation in game objects

An Animator Controller allows you to create and manage a collection of Animation Clips and Animation Transitions for a character or object. In most cases, having multiple animations and switching between them when certain game conditions occur is normal. For example, whenever the spacebar is pressed, we could switch from a walk Animation Clip to a jump Animation Clip (Unity, 2022).

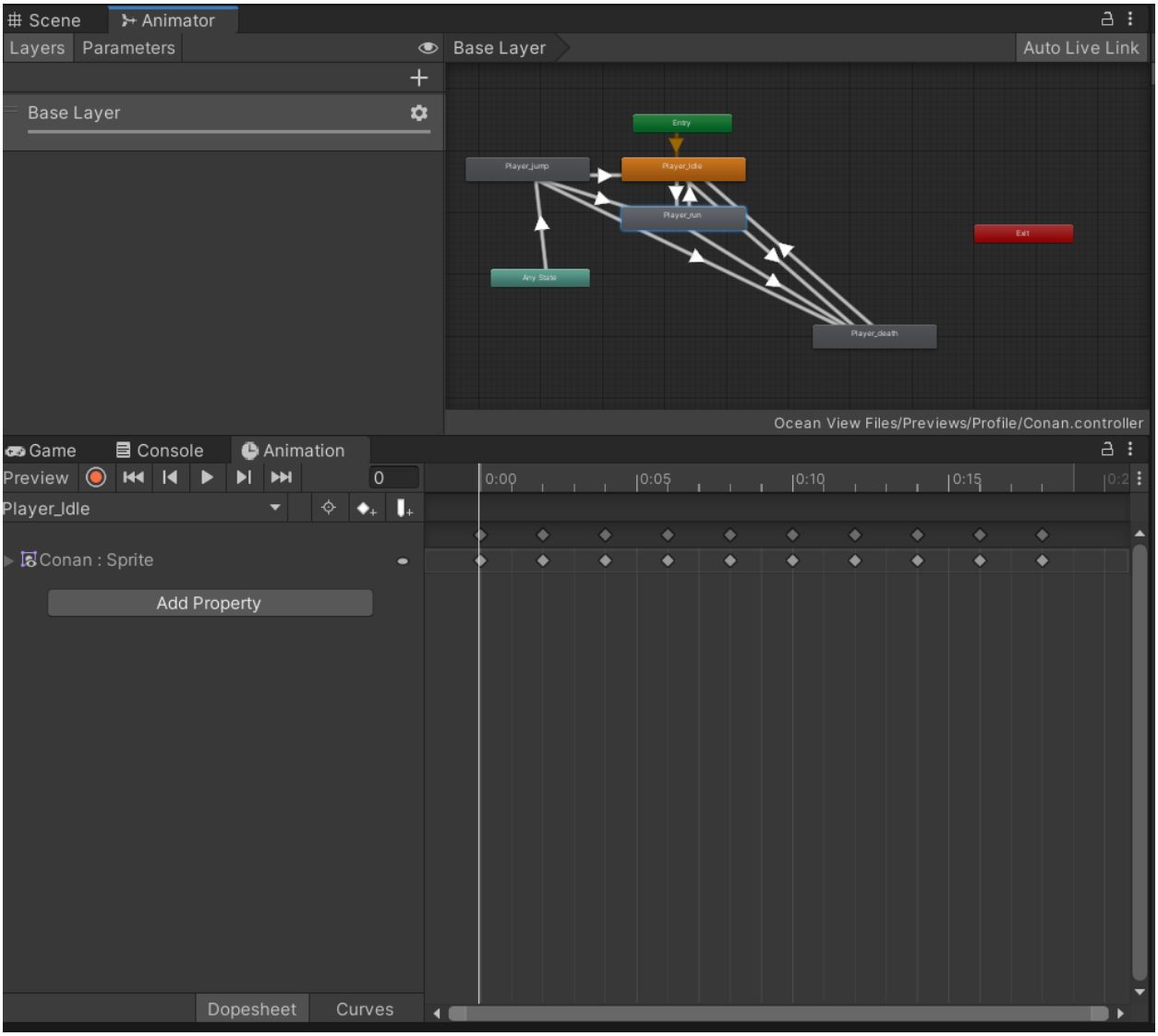


Figure 73: Implementation of Animator and Animation in Unity

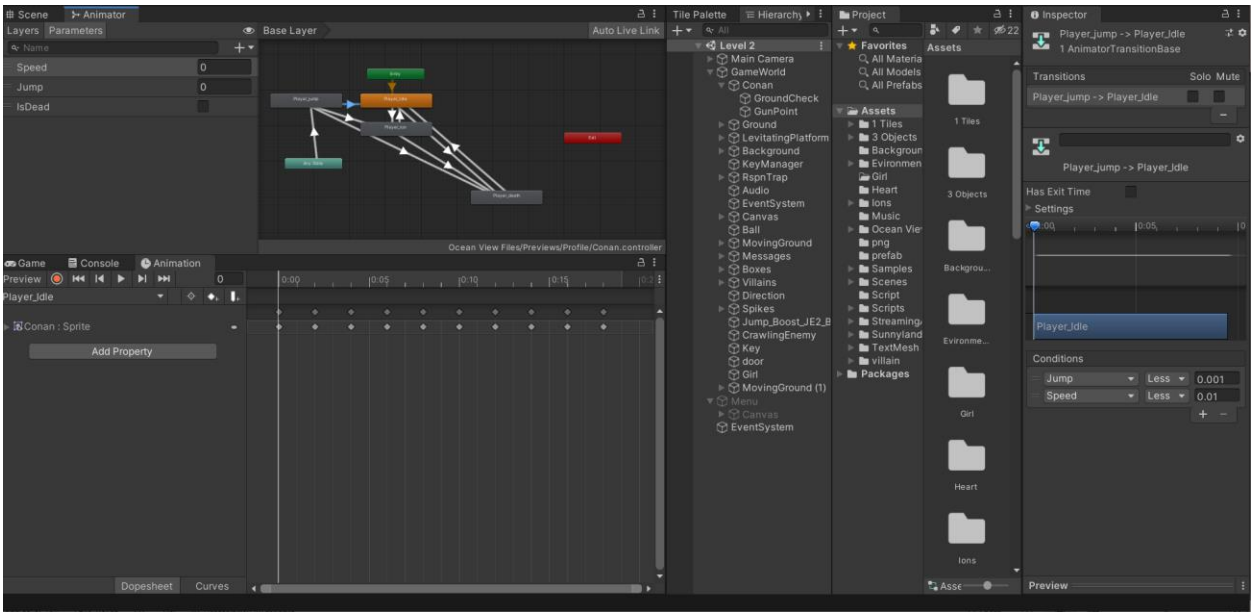


Figure 74: Parameter and Condition for Player_jump to Player_Idle animation

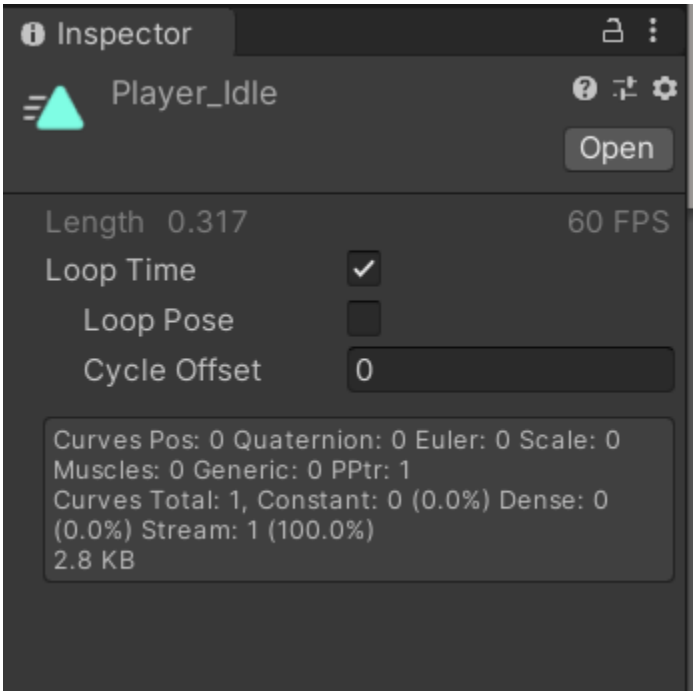


Figure 75: Inspecting Player_Idle Animation property

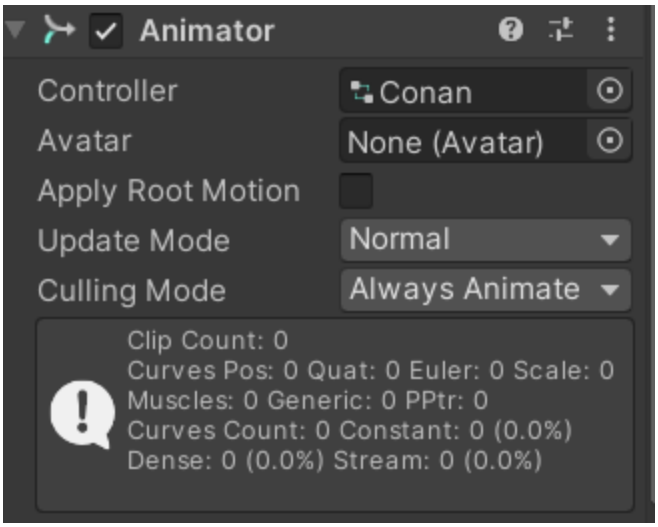


Figure 76: Animator Component in Player

3.8.1.4. Implementation of physics components in game objects

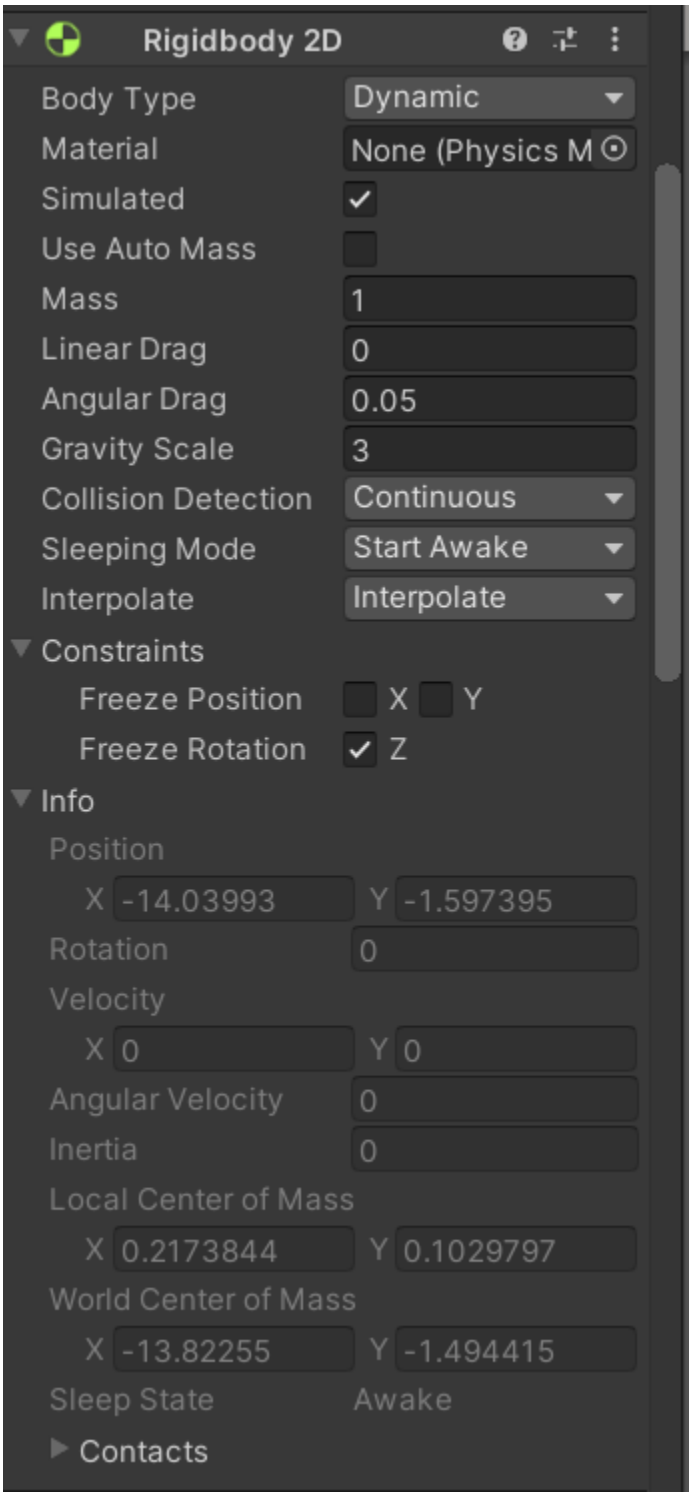


Figure 77: Rigidbody 2D component in Player

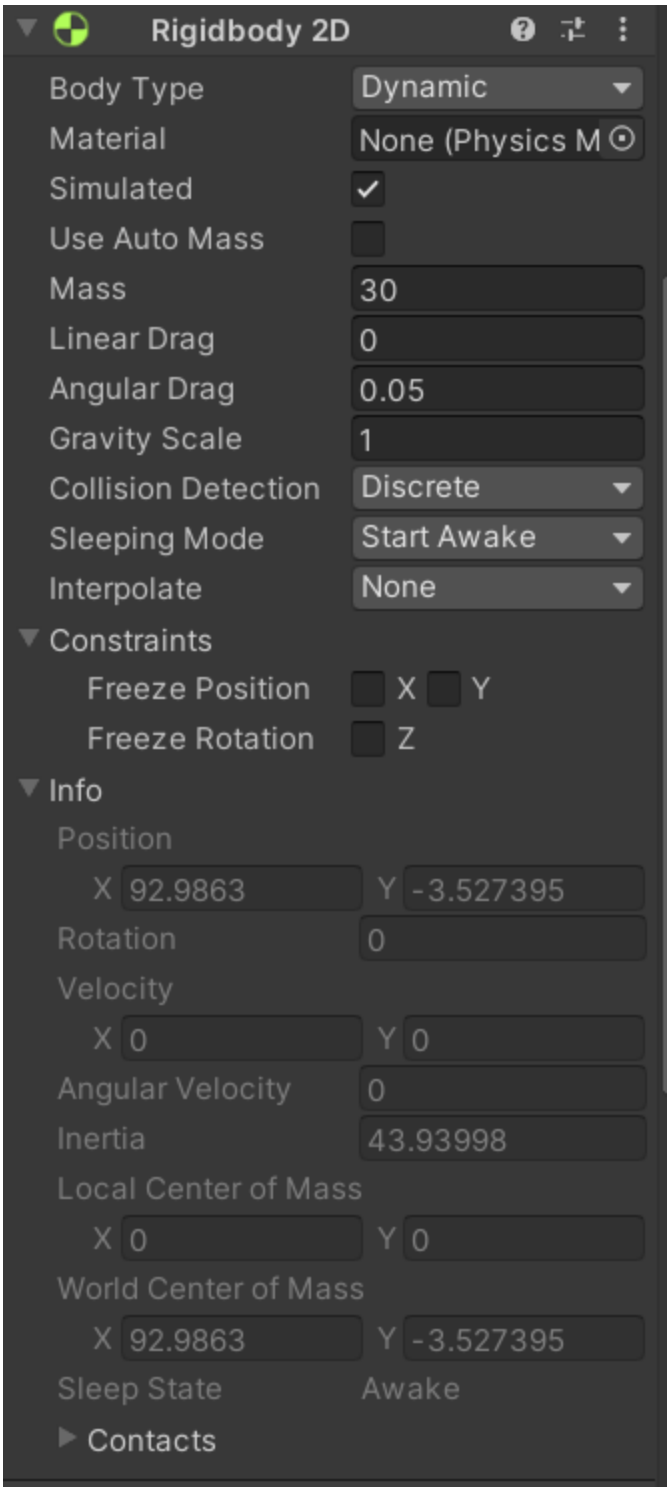


Figure 78: Rigidbody 2D component in Ball

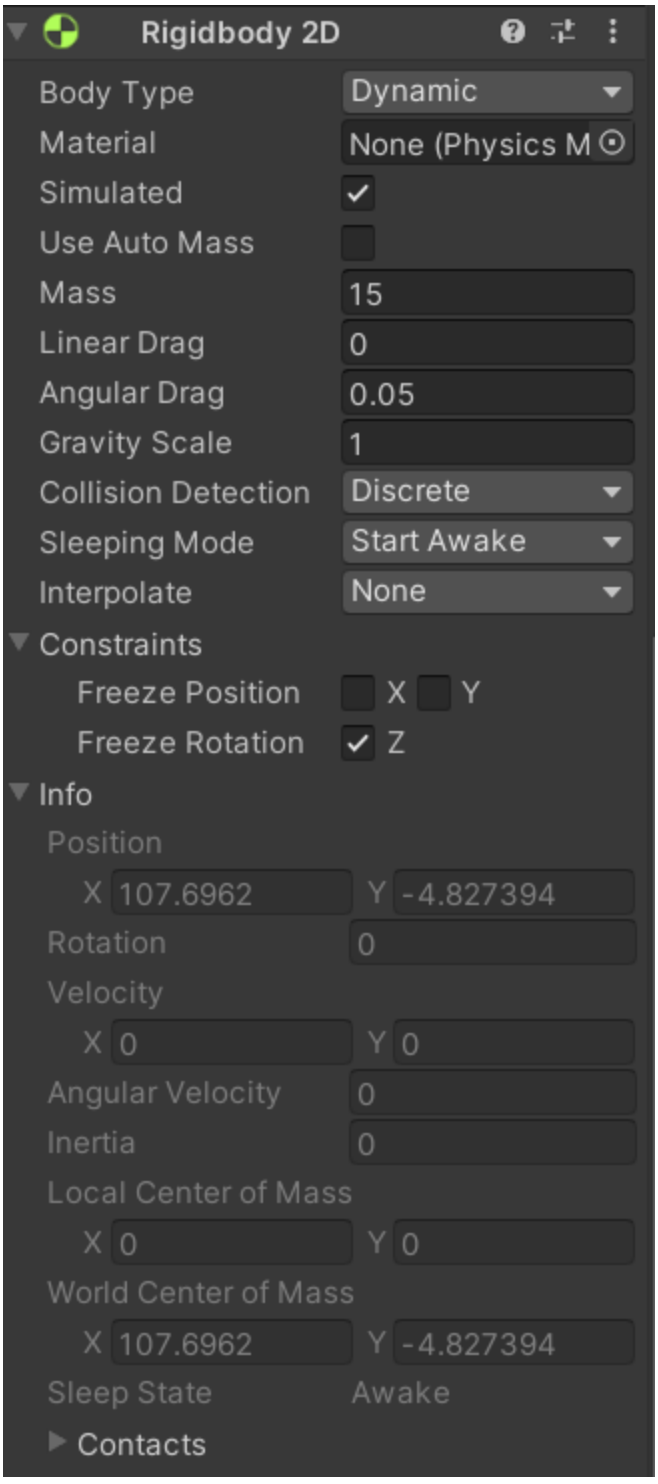


Figure 79: Rigidbody 2D component in Boxes

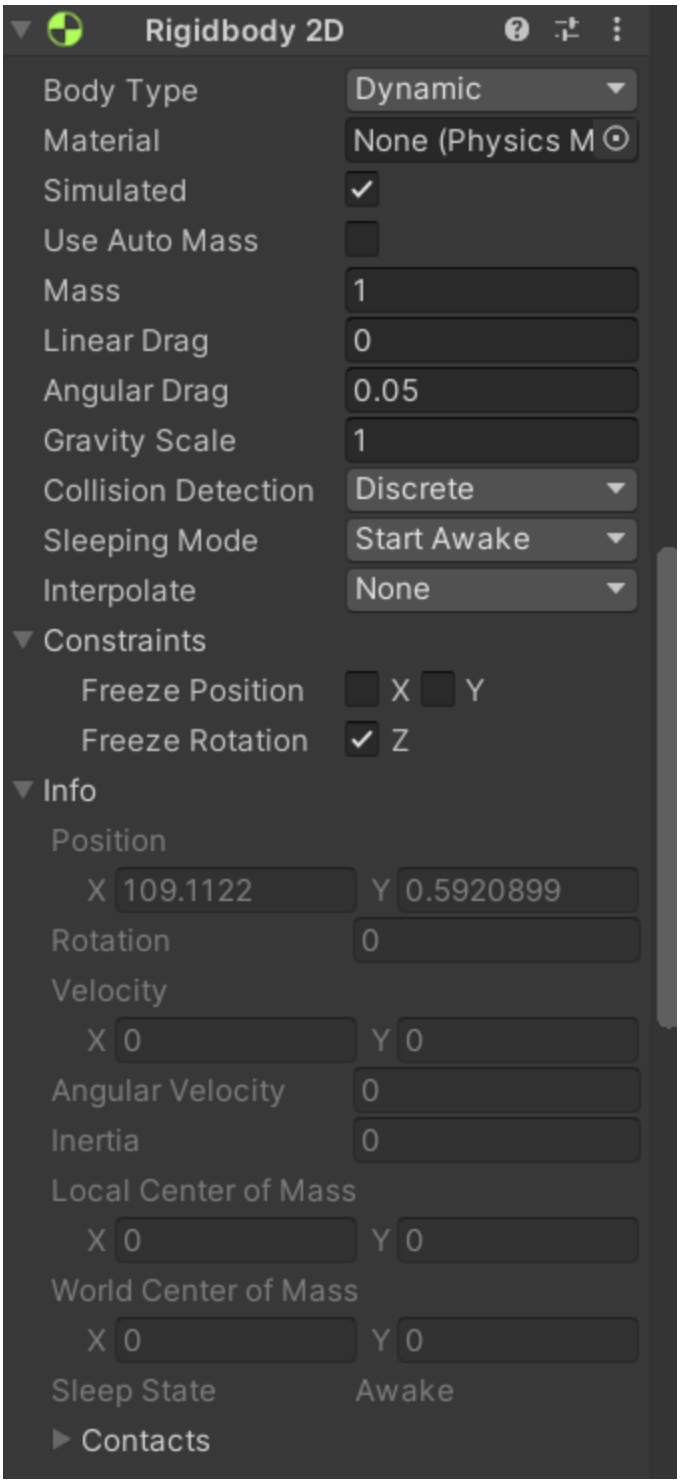


Figure 80: Rigidbody 2D component in Crawling Enemy

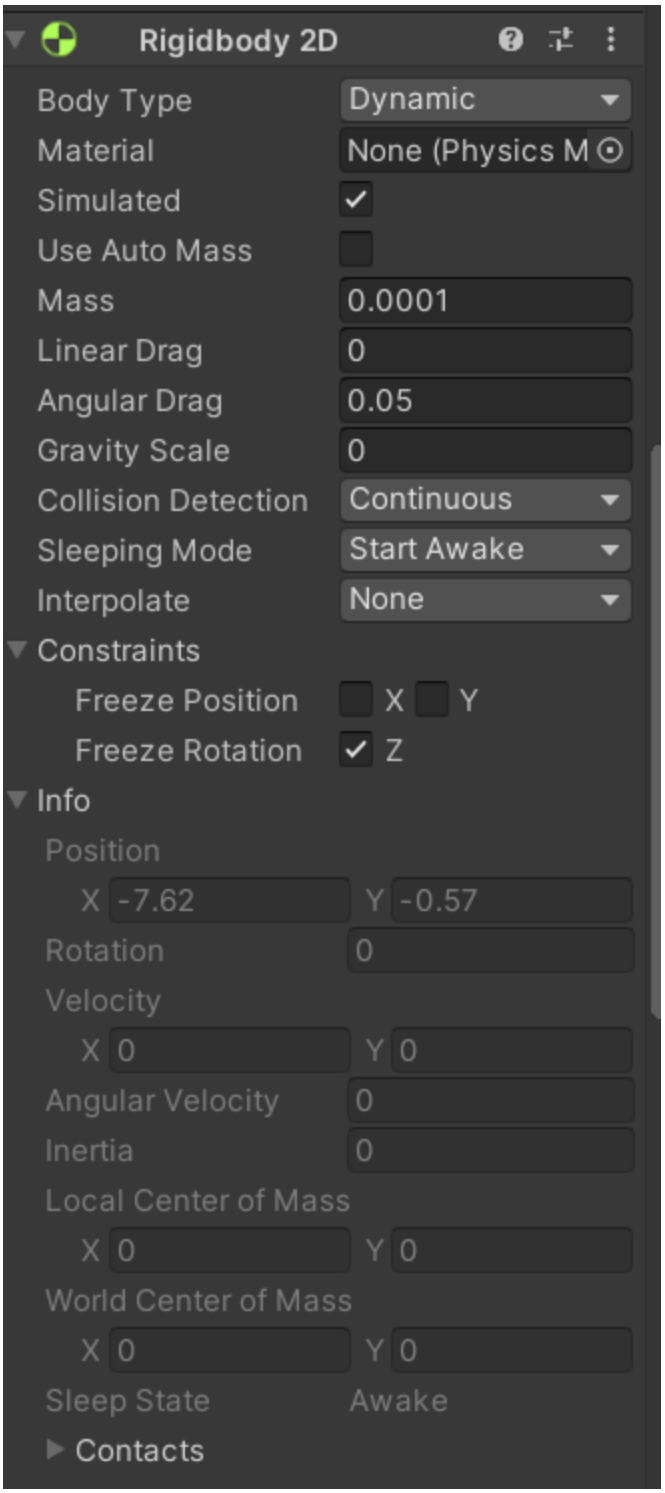


Figure 81: Rigidbody 2D component in Bullet

3.8.1.5. Implementation of Scripts in Unity

Scripts are behavior components that can be added to GameObjects and edited in the Unity Inspector. A script is made up of C# code that is executed while the game is in the "play" state. Scripts can also be used to create tools in Unity that can be used to alter the development process (Unity, 2022).

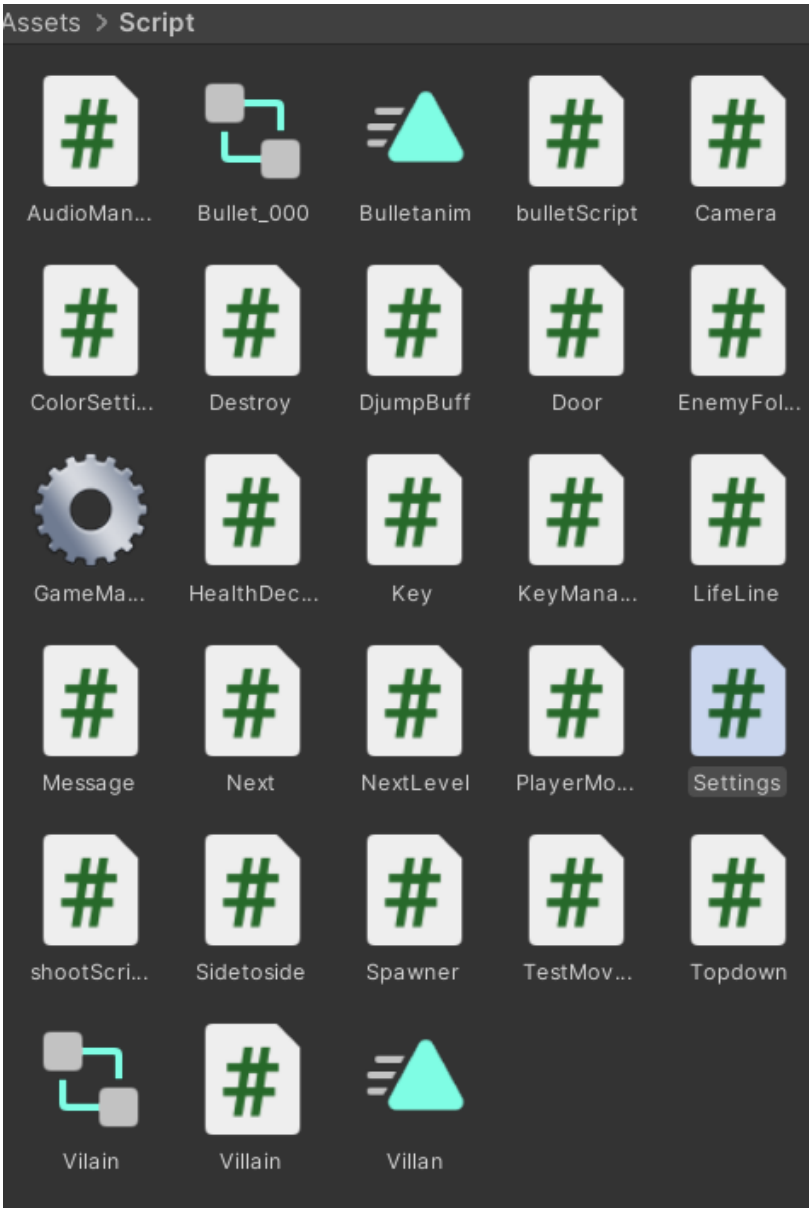


Figure 82: Implementation of Scripts in Unity

```
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script (7 asset references) | 5 references
public class PlayerMovement : MonoBehaviour
{
    //SerializedFeild to protect the value
    //Setting the data type for these variables
    //For Speed
    [SerializeField] float speed;

    //For jumpspeed
    [SerializeField] float jumpspeed;

    //For movement
    [SerializeField] float movement;

    //For jump
    [SerializeField] float jump;

    //Rigidbody2D for Conan as Conan is Ridgibody2D GameObject
    [SerializeField] Rigidbody2D Conan;

    //Animator for animator as animator is Animator used for adding animation frame by frame
    [SerializeField] Animator animator;

    //To check if the player is touching the ground
    [SerializeField] bool isTouchingGround;

    //To take the transform postion of the empty GameObject inside the player
    [SerializeField] Transform groundCheck;

    //To check the layer
    [SerializeField] LayerMask Layer;

    //Adding radius to the empty GameObject to check the LayerMask
    [SerializeField] float R;

    //To check if player is alive
    [SerializeField] bool isAlive;

    //To take the location of the player
    Vector2 location;

    //To keep Game World as GameObject
    [SerializeField] GameObject Gameworld;

    //To check if Double Jump Buff is available
    public bool DoubleJump;

    //To check if the Double Jump buff is available
    public bool DJBuff;

    //Taking DjBuff as the DjumpBuff object
    DjumpBuff djBuff;
```

Figure 83: PlayerMovement Script Part 1

```
56      // Start is called before the first frame update
        Unity Message | 0 references
57      void Start()
58      {
59          //Setting is alive true
60          isAlive = true;
61
62          //Taking the current position
63          location = this.transform.position;
64
65          //To get component that have Rigidbody2D component i.e. Conan
66          Conan = GetComponent<Rigidbody2D>();
67
68          //Finding object type DjumpBuff
69          djBuff = FindObjectOfType<DjumpBuff>();
70      }
71
72      // Update is called once per frame
        Unity Message | 0 references
73      void Update()
74      {
75          //Stopping Move of Player when he is dead
76          if (isAlive == false)
77          {
78              Conan.velocity = new Vector2(0, Conan.velocity.y);
79              return;
80          }
81
82          //Making circle to check the Layer
83          isTouchingGround = Physics2D.OverlapCircle(groundCheck.position, R, Layer);
84
85          //Calling
86          Movement();
87          Jump();
88      }
89
        1 reference
90      private void Movement()
91      {
92          //Getting axis as "Horizontal" and multiplying with speed everytime a or d or side arrows key
93          float movement = Input.GetAxis("Horizontal") * speed;
94
95          //Animator movement
96          animator.SetFloat("Speed", Mathf.Abs(movement));
97
98          //Local positon
99          if (movement > 0)
100         {
101             transform.localRotation = Quaternion.Euler(0, 0, 0);
102         }
```

Figure 84: PlayerMovement Script Part 2

```
        //Turning
        else if (movement < 0)
        {
            transform.localRotation = Quaternion.Euler(0, 180, 0);
        }
        Conan.velocity = new Vector2(movement, Conan.velocity.y);

    }

    1 reference
    private void Jump()
    {
        //Getting Axis as "Jump" and multiplying every time with jumpspeed everytime we use jump button
        float jump = Input.GetAxis("Jump") * jumpspeed;

        //Setting animation for jump
        animator.SetFloat("Jump", Mathf.Abs(jump));

        //Condition for double jump
        if (Input.GetKeyDown(KeyCode.Space) && isTouchingGround == true)
        {

            Conan.velocity = new Vector2(Conan.velocity.x, jump);
            DoubleJump = DJBuff;
        }
        else if (Input.GetKeyDown(KeyCode.Space) && DoubleJump)
        {
            Conan.velocity = new Vector2(Conan.velocity.x, jump);
            DoubleJump = false;

        }
    }

    //For Parent and Child
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.tag=="RightToLeft")
            this.transform.parent = col.transform;
    }

    //For removing parent and child relation of "RightToLeft" GameObject and player
    Unity Message | 0 references
    private void OnCollisionExit2D(Collision2D col)
    {
        if (col.gameObject.tag == "RightToLeft")
            this.transform.parent = Gameworld.transform;
    }
```

Figure 85: PlayerMovement Script Part 3

```
150     //For Trigger
        Unity Message | 0 references
151     private void OnTriggerEnter2D(Collider2D collison)
152     {
153         //For PlayerDeath Animation
154         if (collison.gameObject.tag == "Dmg")
155         {
156             PlayerDeath();
157         }
158
159         //For PlayerDeath Animation
160         if (collison.gameObject.tag == "Enemy")
161         {
162             PlayerDeath();
163         }
164
165         //To go to the next scene
166         if (collison.gameObject.tag == "Next")
167         {
168             //Loading next scene
169             SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
170         }
171
172         //Destorying Key after getting triggered by player
173         if (collison.gameObject.tag == "Key")
174         {
175             //Destorying Key
176             Destroy(collison.gameObject);
177         }
178     }
179
180
181     //For PlayerDeath
        2 references
182     public void PlayerDeath()
183     {
184         //Setting player as dead
185         isAlive = false;
186
187         //Play death animation
188         animator.SetFloat("Jump", 0);
189         animator.SetBool("IsDead", !isAlive);
190
191         //Wait for 2 sec
192         //Respawn player
193         Invoke("RespawnPlayer", 2f);
194
195     }
196
```

Figure 86: PlayerMovement Script Part 4

```
197     //For RespawPlayer
        0 references
198     public void RespawnPlayer()
199     {
200         //Setting Player as alive
201         isAlive = true;
202         animator.SetBool("IsDead", !isAlive);
203
204         //Respawing at the first position and in player facing the same direction
205         transform.position = location;
206         transform.rotation = Quaternion.identity;
207
208         //Enabling SpriteRenderer of the double jump buff
209         djBuff.GetComponent<SpriteRenderer>().enabled = true;
210
211         //Enabling CircleCollider2D of the double jump buff
212         djBuff.GetComponent<CircleCollider2D>().enabled = true;
213     }
214
215
216 }
```

Figure 87: PlayerMovement Script Part 5

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
Unity Script (9 asset references) | 0 references
public class Next : MonoBehaviour
{
    Unity Message | 0 references
    private void Start()
    {
        //Time scale to 1
        Time.timeScale = 1;
    }

    //For Play
    0 references
    public void PlayGame()
    {
        //Next Scene Loading
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    //For Exit
    0 references
    public void ExitGame()
    {
        //Quiting application
        Application.Quit();
    }

    //For MainMent
     0 references
     public void MainMenu()
    {
        //Loading Scene 0
        SceneManager.LoadScene(0);
    }
}
```

Figure 88: Next Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (7 asset references) | 0 references
public class Camera : MonoBehaviour
{
    [SerializeField] GameObject Conan;

    //Starting Point for horizonatal
    [SerializeField] float startingPoint;

    //Starting point for vertical
    [SerializeField] float sPoint;

    //Ending point for horizontal
    [SerializeField] float endingPoint;

    //Ending point for vertical
    [SerializeField] float ePoint;

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        //Claming camera for horizontal Movement of player
        float newpoint= Mathf.Clamp(Conan.transform.position.x, startingPoint, endingPoint);

         //Claming camera for vertical Movement of player
         float npoint = Mathf.Clamp(Conan.transform.position.y, sPoint, ePoint);

         //For freezing Z axis
        transform.position = new Vector3(newpoint, npoint,transform.position.z);
    }
}
```

Figure 89: Camera Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 1 reference
public class EnemyFollow : MonoBehaviour
{
    //Speed
    [SerializeField] float speed;

    //Target that this AI going to follow
    private Transform target;

    //Health
    public float health = 100;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        //Taking Player as target to follow
        target = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        //Taking the position of the playre and moving towards it
        transform.position = Vector2.MoveTowards(transform.position, target.position, speed * Time.deltaTime);

        //Condition
        if (health == 0)
        {
            //Self Destroy
            Destroy(gameObject);
        }
    }

    //On collision
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        //if collides with Player
        if (collision.gameObject.tag == "Player")
        {
            //Self Destroy
            Destroy(gameObject);
        }
    }
}
```

Figure 90: EnemyFollow Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class Spawner : MonoBehaviour
{
    [SerializeField] GameObject ObjectToBeSpawn;
    BoxCollider2D Spw;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        //Get Object Having BoxCollider2D i.e. Spw
        Spw = GetComponent<BoxCollider2D>();
    }

    0 references
    public BoxCollider2D GetBox(bool tempBool)
    {
        Spw.enabled = tempBool;
        return Spw;
    }

    1 reference
    private void SpawnObject()
    {
        //Instantiating object
        Instantiate(ObjectToBeSpawn, transform.position, transform.rotation);
    }

    //On Trigger
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            SpawnObject();
            Spw.enabled = false;
        }
    }
}
```

Figure 91: Spawner Script

```
using UnityEngine;

Unity Script (5 asset references) | 1 reference
public class Villain : MonoBehaviour
{
    //Starting Point
    [SerializeField] float StartPoint;

    //Ending Point
    [SerializeField] float EndPoint;

    //Speed
    [SerializeField] float speed;

    //Check Move right
    [SerializeField] bool moveRight = true;

    //Health
    public float health;
    Unity Message | 0 references
    void Update()
    {
        if (health == 0)
        {
            //Destroying Villain
            Destroy(gameObject);
        }

        //Condition for rotation
        if (transform.position.x >= EndPoint)
        {
            transform.localRotation = Quaternion.Euler(0, 180, 0);
            moveRight = false;
        }
        if (transform.position.x <= StartPoint)
        {
           transform.localRotation = Quaternion.Euler(0, 0, 0);
            moveRight = true;
        }
```

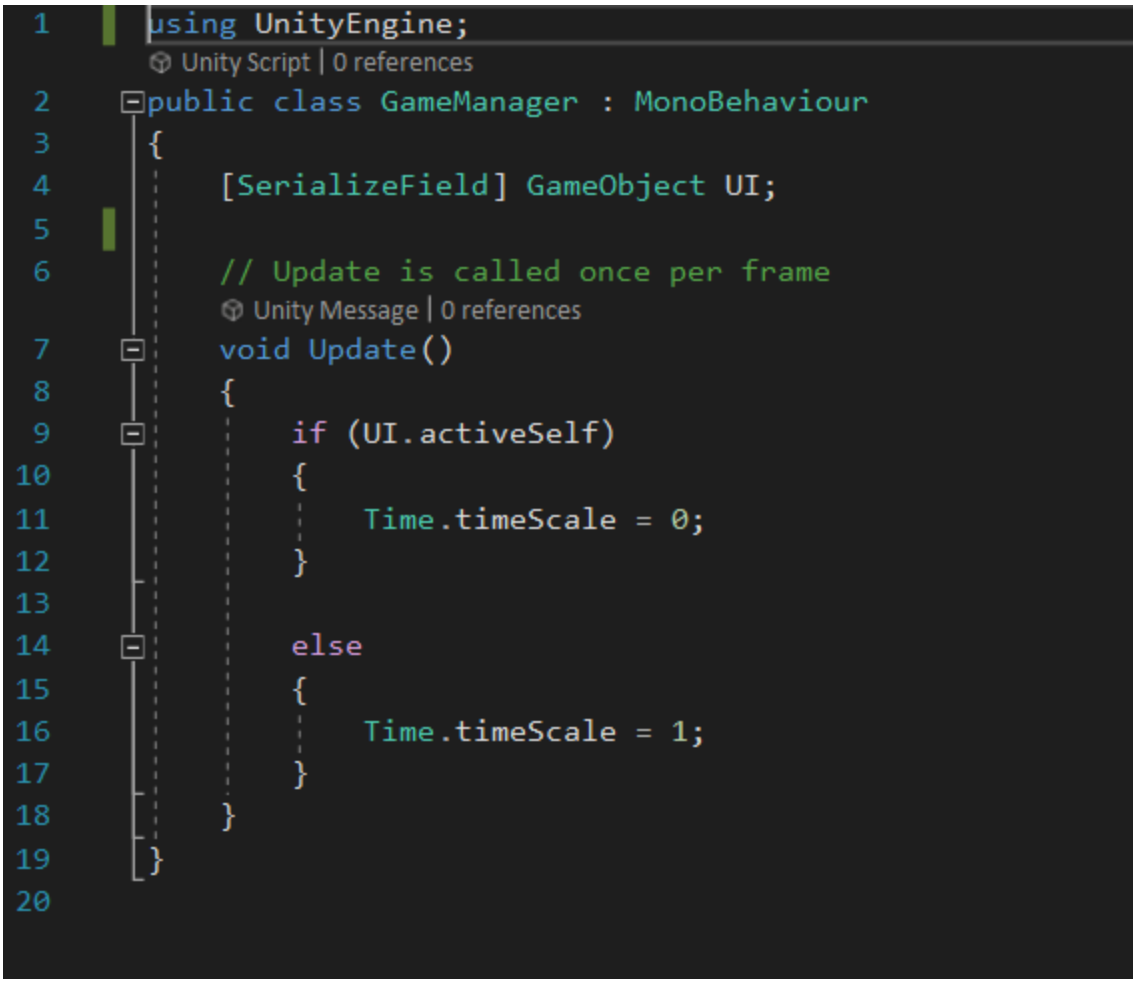
Figure 92: Villain Script Part 1

```
        //Condition for Move right
        if (moveRight)
            transform.position = new Vector2(transform.position.x + speed * Time.deltaTime, transform.position.y);
        else
            transform.position = new Vector2(transform.position.x - speed * Time.deltaTime, transform.position.y);

    }

    //On Collison
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        //If collides with Player
        if (collision.gameObject.tag=="Player")
        {
            //Self destruction
            Destroy(gameObject);
        }
    }
}
```

Figure 93: Villain Script Part 2

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class Topdown : MonoBehaviour
{
    //Starting point
    [SerializeField] float StartPoint;

    //Ending Point
    [SerializeField] float EndPoint;

    //Speed
    [SerializeField] float speed;

    //Moving top check
    [SerializeField] bool moveTop = true;
    Unity Message | 0 references
    void Update()
    {
        //Conditions
        if (transform.position.y >= EndPoint)
        {
            moveTop = false;
        }
        if (transform.position.y <= StartPoint)
        {
            moveTop = true;
        }

        if (moveTop)
            transform.position = new Vector2(transform.position.x, transform.position.y + speed * Time.deltaTime);
        else
            transform.position = new Vector2(transform.position.x, transform.position.y - speed * Time.deltaTime);

    }
}
```

Figure 94: Topdown Script



```
using UnityEngine;
Unity Script | 0 references
public class GameManager : MonoBehaviour
{
    [SerializeField] GameObject UI;

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (UI.activeSelf)
        {
            Time.timeScale = 0;
        }

        else
        {
            Time.timeScale = 1;
        }
    }
}
```

Figure 95: GameManager Script

```
using UnityEngine;
using TMPro;

Unity Script (2 asset references) | 4 references
public class KeyManager : MonoBehaviour
{
    //Setting KeyManger as static
    public static KeyManager instance;

    //Using TMPROGUI text
    public TextMeshProUGUI text;

    //KeyScore
    public int keyScore;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        //Conditon for instance
        if(instance == null)
        {
            instance = this;
        }
    }

    //or ChangeKeyValue
    1 reference
    public void ChangeKeyValue(int keyValue)
    {
        //KeyScore added as keyValue
        keyScore += keyValue;

        //KeyScore to string to show in message
        text.text = keyScore.ToString();
    }
}

```

Figure 96: KeyManager Script

```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class Destroy : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        Destroy(gameObject, 0.5f);
    }


}
```

Figure 97: Destroy Script

```
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class Key : MonoBehaviour
{
    //Setting Key Value to 1
    public int keyValue = 1;

    //On Trigger
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D other)
    {
        //Contion if it gets trigger by Player
        if (other.gameObject.tag =="Player")
        {
            //Changing the KeyValue
            KeyManager.instance.ChangeKeyValue(keyValue);
        }
    }
}
```

Figure 98:Key Script

```
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class bulletScript : MonoBehaviour
{
    //SerizlizeFeild to protect value
    //Taking BulletRB as RigidBody2D
    public Rigidbody2D bulletRB;

    //Bullet Speed
    [SerializeField] float bulletSpeed;

    //For Damage Value
    public float dmg;

    //For impactEffect
    [SerializeField] GameObject impactEffect;
    Unity Message | 0 references
    void Start()
    {
        //getting GameObject that have Rigidbody2D component i.e. bulltetRB
        bulletRB = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        //Adding Velocty in bullet
        bulletRB.velocity = transform.right * bulletSpeed;

        //Destorying bullet after 2 second
        Destroy(gameObject, 2f);

    }
```

Figure 99: BulletScript Script Part 1

```
34      //On Collison
        Unity Message | 0 references
35      private void OnCollisionEnter2D(Collision2D collision)
36      {
37          //Condition if the bullet collides with Ground
38          if (collision.gameObject.tag == "Ground" )
39          {
40              //Destroy bullet
41              Destroy(gameObject);
42
43              //ImpactEffect got instantiate where the bullet hit the ground
44              Instantiate(impactEffect, transform.position, transform.rotation);
45          }
46          if (collision.gameObject.tag == "Enemy")
47          {
48              //Destroy Bullet
49              Destroy(gameObject);
50
51              //ImpactEffect got instantiate where the bullet hit the Enemy
52              Instantiate(impactEffect, transform.position, transform.rotation);
53
54              //Finding the Vllain Script and dealing dmg
55              FindObjectOfType<Villain>().health -= dmg;
56
57              //Finding EnemyFollow Script and dealing dmg
58              FindObjectOfType<EnemyFollow>().health -= dmg;
59          }
60      }
61
62  }
```

Figure 100: BulletScript Script Part 2

```
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class Door : MonoBehaviour
{
    private KeyManager kValue;
    Unity Message | 0 references
    private void Start()
    {
        kValue = FindObjectOfType<KeyManager>();
    }
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            if (kValue.keyScore==1)
            {
                Destroy(gameObject);
            }

        }
    }
}
```

Figure 101: Door Script

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class enemyScript : MonoBehaviour
6   {
7
8       public float health;
9
10      // Update is called once per frame
11      void Update()
12      {
13          if(health == 0)
14          {
15              Destroy(gameObject);
16          }
17      }
18  }
19
```

Figure 102: enemyScript Script

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script (2 asset references) | 2 references
public class LifeLine : MonoBehaviour
{
    //Setting public for varibale so the we can change it values using other script
    //Taking GameObjets
    public GameObject h1, h2, h3, h4, h5, over;

    //Health of the Player
    public int health;

    //End Time
    public float endtime;

    //Value
    public float Value;

    //To check if the Game is Over
    [SerializeField] bool isGameOver;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        health = 5;
        h1.gameObject.SetActive(true);
        h2.gameObject.SetActive(true);
        h3.gameObject.SetActive(true);
        h4.gameObject.SetActive(true);
        h5.gameObject.SetActive(true);
        over.gameObject.SetActive(false);
        Value = 2;
        endtime = 1;
    }
```

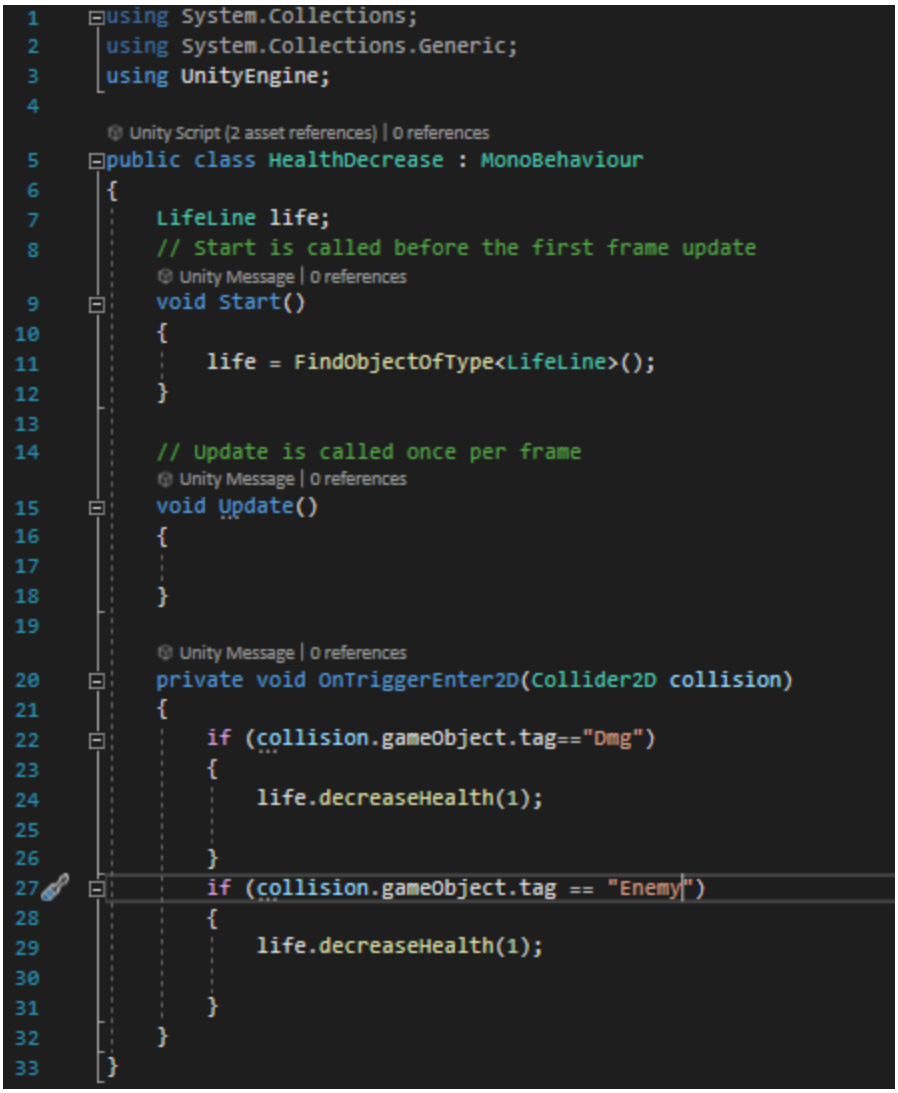
Figure 103: LifeLine Script Part 1

```
38      //For GameOver
        1 reference
39      private void GameOver()
40      {
41          //Condition for GameOver
42          if (isGameOver ==true)
43          {
44              //Starting Coroutine For Message
45              StartCoroutine(MessageTimer());
46
47          }
48      }
         1 reference
49       IEnumerator MessageTimer()
50      {
51        //Stopping time
52         Time.timeScale = 0;
53         yield return new WaitForSeconds(3);
54         Time.timeScale = 1;
55         SceneManager.LoadScene(0);
56      }
57
58      //For Decreasing Health
        2 references
59      public void decreaseHealth(int dmg)
60      {
61          //Switch Conditons
62          health -= dmg;
63          if (health > 5)
64              health = 5;
65
66          switch (health)
67          {
68              case 5:
69                  break;
70              case 4:
71                  h5.gameObject.SetActive(false);
```

Figure 104: LifeLine Script Part 2

```
70              case 4:
71                  h5.gameObject.SetActive(false);
72                  break;
73              case 3:
74                  h4.gameObject.SetActive(false);
75                  h5.gameObject.SetActive(false);
76                  break;
77              case 2:
78                  h3.gameObject.SetActive(false);
79                  h4.gameObject.SetActive(false);
80                  h5.gameObject.SetActive(false);
81                  break;
82              case 1:
83                  h2.gameObject.SetActive(false);
84                  h3.gameObject.SetActive(false);
85                  h4.gameObject.SetActive(false);
86                  h5.gameObject.SetActive(false);
87                  break;
88              case 0:
89                  h1.gameObject.SetActive(false);
90                  h2.gameObject.SetActive(false);
91                  h3.gameObject.SetActive(false);
92                  h4.gameObject.SetActive(false);
93                  h5.gameObject.SetActive(false);
94                  over.gameObject.SetActive(true);
95                  isGameOver = true;
96                  GameOver();
97                  break;
98          }
99      }
100 }
```

Figure 105: LifeLine Script Part 3



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class HealthDecrease : MonoBehaviour
{
    LifeLine life;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        life = FindObjectOfType<LifeLine>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag=="Dmg")
        {
            life.decreaseHealth(1);

        }
        if (collision.gameObject.tag == "Enemy")
        {
            life.decreaseHealth(1);

        }
    }
}
```

Figure 106: HealthDecrease Script

```
using System.Collections;
using UnityEngine;
using TMPro;

Unity Script (6 asset references) | 0 references
public class Message : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI Msg;
    [TextArea(10, 4)]
    [SerializeField] string textMessage;
    [SerializeField] float messagetimer = 1;

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        StartCoroutine(MessageTimer());
    }

    1 reference
    IEnumerator MessageTimer()
    {
        Msg.text = textMessage;
        yield return new WaitForSeconds(messagetimer);
        Msg.text = " ";
    }
}
```

Figure 107: Message Script

```
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script (2 asset references) | 0 references
public class NextLevel : MonoBehaviour
{
    //This work on collison with Player
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        //Sending to the next scene
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
```

Figure 108: NextLevel Script

```
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class shootScript : MonoBehaviour
{
    public Transform shotArea;
    public GameObject bullet;


    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        Shot();
    }

    1 reference
    void Shot()
    {
        if (Input.GetKeyDown(KeyCode.F))
        {
            Instantiate(bullet, shotArea.position, shotArea.rotation);
        }
    }

}
```

Figure 109: shootScript Script

```
using UnityEngine;

Unity Script (3 asset references) | 0 references
public class Sidetoside : MonoBehaviour
{
    [SerializeField] float StartPoint;
    [SerializeField] float EndPoint;
    [SerializeField] float speed;
    [SerializeField] bool moveRight = true;
    Unity Message | 0 references
    void Update()
    {
        if (transform.position.x >= EndPoint)
        {
            moveRight = false;
        }
        if (transform.position.x <= StartPoint)
        {
            moveRight = true;
        }

        if (moveRight)
            transform.position = new Vector2(transform.position.x + speed * Time.deltaTime, transform.position.y );
        else
            transform.position = new Vector2(transform.position.x - speed * Time.deltaTime, transform.position.y);

    }
}
```

Figure 110: Sidetoside Script

```
using System.Collections;
using UnityEngine;
using TMPro;

Unity Script (2 asset references) | 2 references
public class DjumpBuff : MonoBehaviour
{
    //Message as TMPROGUI
    [SerializeField] TextMeshProUGUI Msg;

    //Setting TextArea
    [TextArea(10, 4)]

    //Taking Message
    [SerializeField] string textMessage;

    //Player Collider
    public Collider2D Player;

    //Duration
    public float duration = 1;

    //Random
    public float random = 0;

   //Taking PlayerMovement Object as J
    public PlayerMovement J;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        //Finding PlayerMovement Copmponent
        PlayerMovement J = Player.GetComponent<PlayerMovement>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        //Conditon for random
        if (random == 2)
        {
            //Duration value goes down with real time by -0.2
            duration = duration - 0.2f * Time.deltaTime;
        }
        if (duration <= 0)
        {
            random = 0;
            PlayerMovement m = Player.GetComponent<PlayerMovement>();

            //Double jump buff off
            J.DJBuff= false;
            duration = 1;
```

Figure 111: DjumpBuff Script Part 1

```
            duration = 1;
        }
    }

    //On Trigger
    Unity Message | 0 references
    public void OnTriggerEnter2D(Collider2D other)
    {
        //If triggered by Player
        if (other.gameObject.tag == "Player")
        {
            //JumpTwice Coroutine started
            StartCoroutine(JumpTwice(other));

            //Value of random set to 2
            random = 2;

            //MessageTimer Coroutine started
            StartCoroutine(MessageTimer());
        }
    }

    //Inenumerater for JumpTwice
    1 reference
    IEnumerator JumpTwice(Collider2D player)
    {
        //Setting double jump buff on
        J.DJBuff = true;

        //Turning off the components
        GetComponent<SpriteRenderer>().enabled = false;
        GetComponent<CircleCollider2D>().enabled = false;
        yield return new WaitForSeconds(2);
    }

    //Inenumerater for JMessageTimer
    1 reference
    IEnumerator MessageTimer()
    {
        Msg.text = textMessage;
        yield return new WaitForSeconds(5);
        Msg.text = " ";
    }
}
```

Figure 112: DjumpBuff Script Part 2

```
using UnityEngine;
using UnityEngine.UI;

Unity Script (1 asset reference) | 0 references
public class AudioManager : MonoBehaviour
{
    [SerializeField] AudioSource gameMusic;
    [SerializeField] Slider musicSlider;

    Unity Message | 0 references
    private void Start()
    {

        if (PlayerPrefs.GetInt("FIRSTTIMEOPENING", 1) == 1)
        {
            //Debug.Log("First Time Opening");
            //Set first time opening to false
            PlayerPrefs.SetInt("FIRSTTIMEOPENING", 0);
            gameMusic.volume = musicSlider.value = 0.5f;

        }
        else
        {
            gameMusic.volume = musicSlider.value = PlayerPrefs.GetFloat("Music");

        }
    }

    Unity Message | 0 references
    private void Update()
    {
        SetVolume();
    }

    1 reference
    private void SetVolume()
    {
        gameMusic.volume = musicSlider.value;
    }


    Unity Message | 0 references
    public void OnDestroy()
    {
        PlayerPrefs.SetFloat("Music", musicSlider.value);
    }


}
```

Figure 113: AudioManager Script

```
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;
using UnityEngine.UI;

Unity Script (1 asset reference) | 0 references
public class ColorSettings : MonoBehaviour
{
    //Sliders
    Slider brightnessSlider;
    Slider contrastSlider;
    Slider saturationSlider;

    //Volume
    private Volume v;

    //For color adjustment
    private ColorAdjustments setting;

    //Color Setting
    Settings colourSettings;
    Unity Message | 0 references
    private void Awake()
    {
        //Taking object type Settings
        colourSettings = FindObjectOfType<Settings>();

        //Finding Sliders
        brightnessSlider = colourSettings.GetBrightness();
        contrastSlider = colourSettings.GetContrast();
        saturationSlider = colourSettings.GetSaturation();
    }
    Unity Message | 0 references
    private void Start()
    {
        v = GetComponent<Volume>();
        v.profile.TryGet(out setting);

        //Default Value for first time
        brightnessSlider.maxValue = 1f;
        brightnessSlider.minValue = -2f;

        //Default Value for first time
        contrastSlider.maxValue = 100f;
        contrastSlider.minValue = -50f;

        //Default Value for first time
        saturationSlider.maxValue = 100f;
        saturationSlider.minValue = -100f;
```

Figure 114: ColorSettings Script Part 1

```
49          //PlayerPrefs for value
50          if (PlayerPrefs.HasKey("Brightness"))
51          {
52              setting.postExposure.value = brightnessSlider.value = PlayerPrefs.GetFloat("Brightness");
53              setting.contrast.value = contrastSlider.value = PlayerPrefs.GetFloat("Contrast");
54              setting.saturation.value = saturationSlider.value = PlayerPrefs.GetFloat("Saturation");
55          }
56          else
57          {
58              setting.postExposure.value = brightnessSlider.value = 0f;
59              setting.contrast.value = contrastSlider.value = 0f;
60              setting.saturation.value = saturationSlider.value = 0f;
61          }
62      }
63
64      //For color reset
        0 references
65      public void resetColour()
66      {
67          setting.postExposure.value = brightnessSlider.value = 0.01194698f;
68          setting.contrast.value = contrastSlider.value = 0.83559f;
69          setting.saturation.value = saturationSlider.value = 35.4568f;
70      }
71
        Unity Message | 0 references
72      public void Update()
73      {
74          setting.postExposure.value = brightnessSlider.value;
75          setting.contrast.value = contrastSlider.value;
76          setting.saturation.value = saturationSlider.value;
77      }
78
79      //Saving value on destroy (Cache memory)
        Unity Message | 0 references
80      private void OnDestroy()
81      {
82          PlayerPrefs.SetFloat("Brightness", setting.postExposure.value);
83          PlayerPrefs.SetFloat("Saturation", setting.saturation.value);
84          PlayerPrefs.SetFloat("Contrast", setting.contrast.value);
85      }
86
87      //Saving value while going back to setting (Cache memory)
        0 references
88      private void Save()
89      {
90          PlayerPrefs.SetFloat("Brightness", setting.postExposure.value);
91          PlayerPrefs.SetFloat("Saturation", setting.saturation.value);
92          PlayerPrefs.SetFloat("Contrast", setting.contrast.value);
93      }
94  }
```

Figure 115: ColorSettings Script Part 2

Chapter 4 Testing

4.1. Functionality Testing

4.1.1 Hover testing for Main Menu

Test Case	1
Action	Hovering mouse in the buttons.
Expect Result	Color of the hovered button to be changed.
Actual Result	Color of the hovered button was changed.
Conclusion	Successful

Table 10: Testing hover for buttons in Main Menu

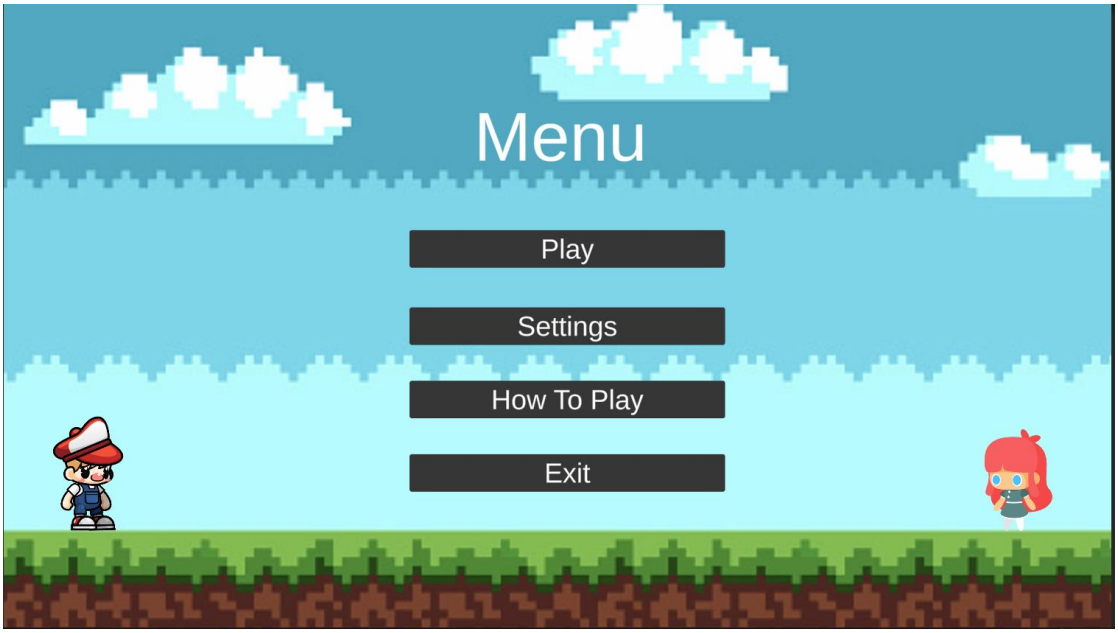


Figure 116: Play Button before hover

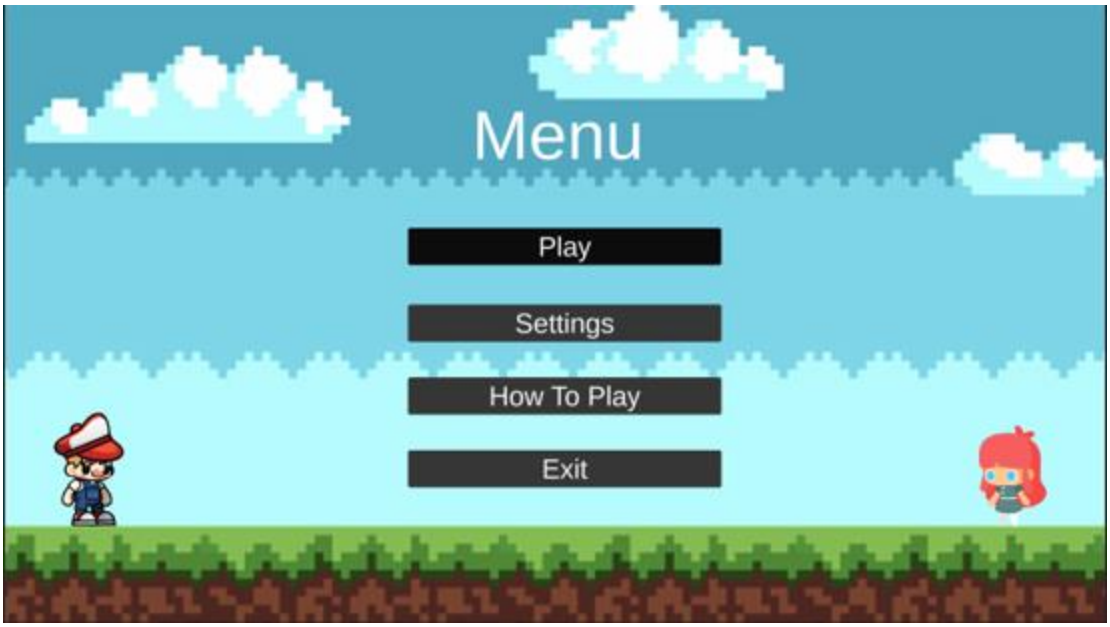


Figure 117: Play Button after over

4.1.2. Play Button testing

Test Case	2
Action	Play button is pressed.
Expect Result	Next scene to be opened.
Actual Result	Next scene was opened.
Conclusion	Successful

Table 11: Testing for Play Button

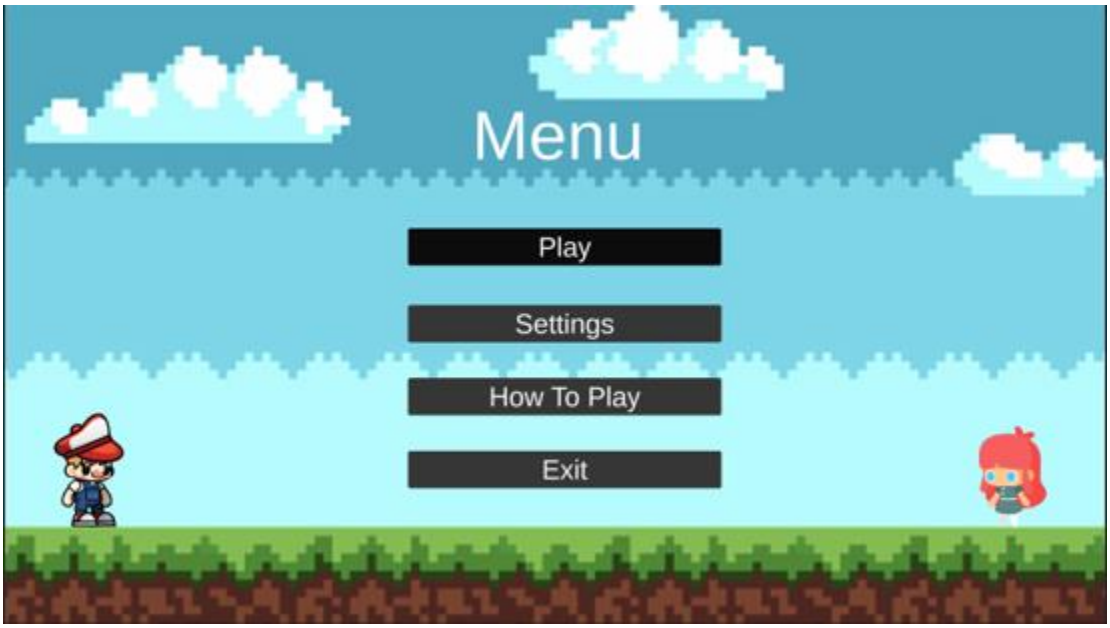


Figure 118: Before pressing the Play Button

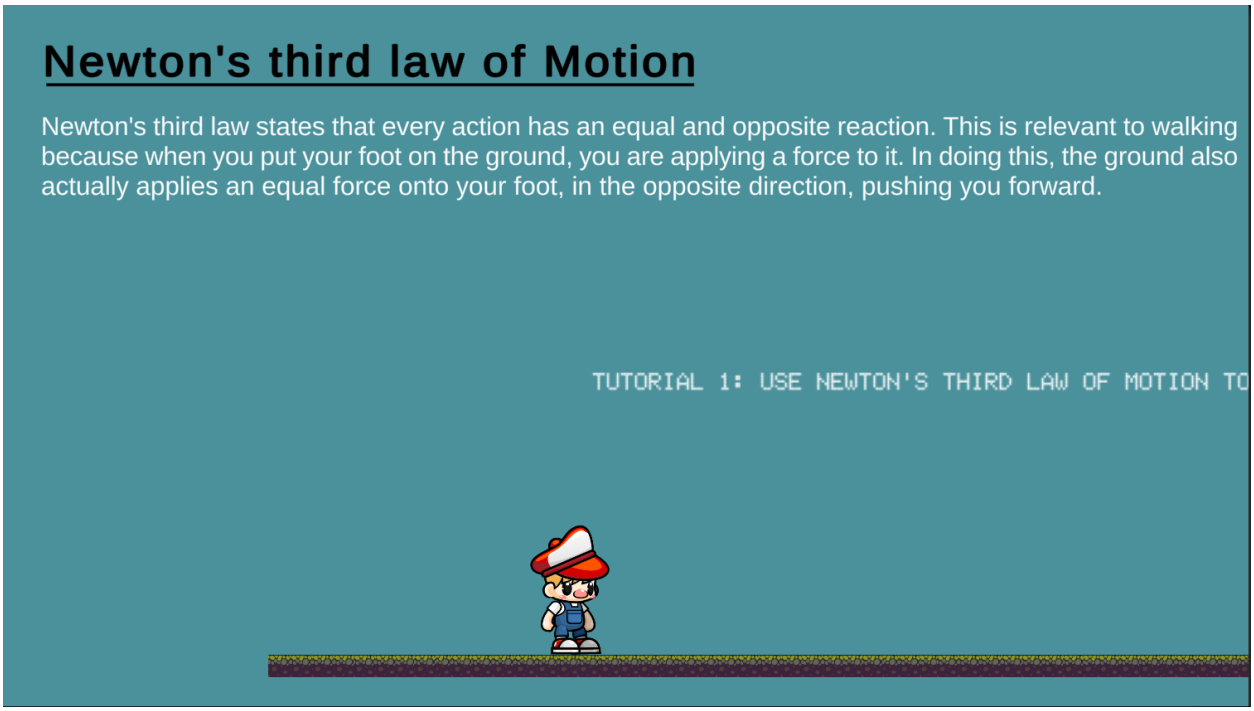


Figure 119: After pressing the Play Button

4.1.3. Settings Button Testing

Test Case	3
Action	Settings button is pressed.
Expect Result	Settings Panel to be opened.
Actual Result	Settings Panel was opened.
Conclusion	Successful

Table 12: Testing Settings Button

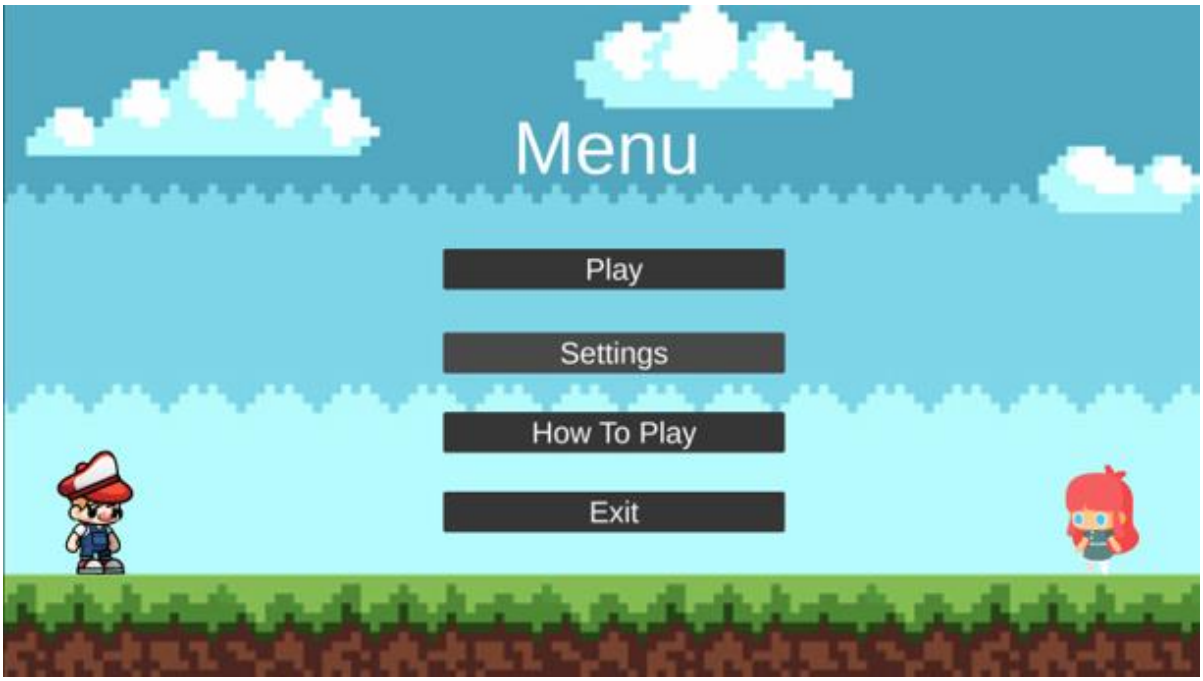


Figure 120: Before Pressing Settings button

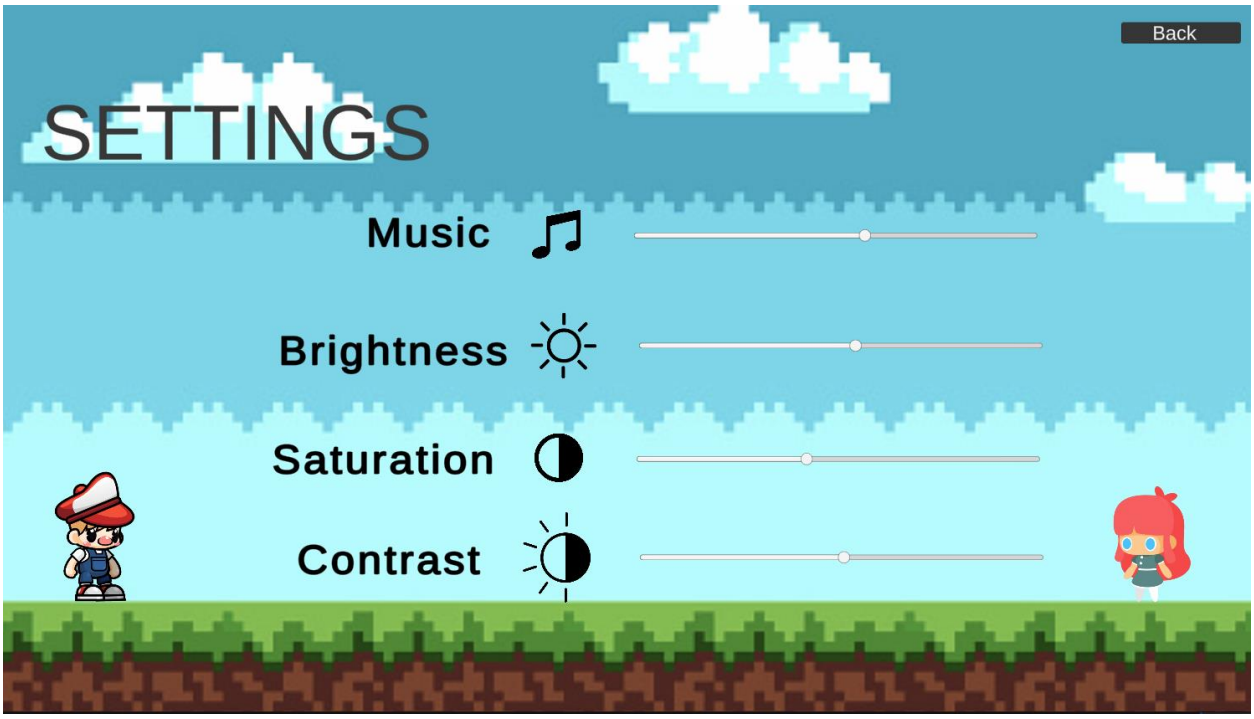


Figure 121: After Pressing the Settings Button

4.1.4. Music Slider Testing

Test Case	4
Action	Music Slider value is changed.
Expect Result	Slider value to change and Music loudness to change.
Actual Result	Slider value and Music loudness was changed.
Conclusion	Successful

Table 13: Testing Music Sider

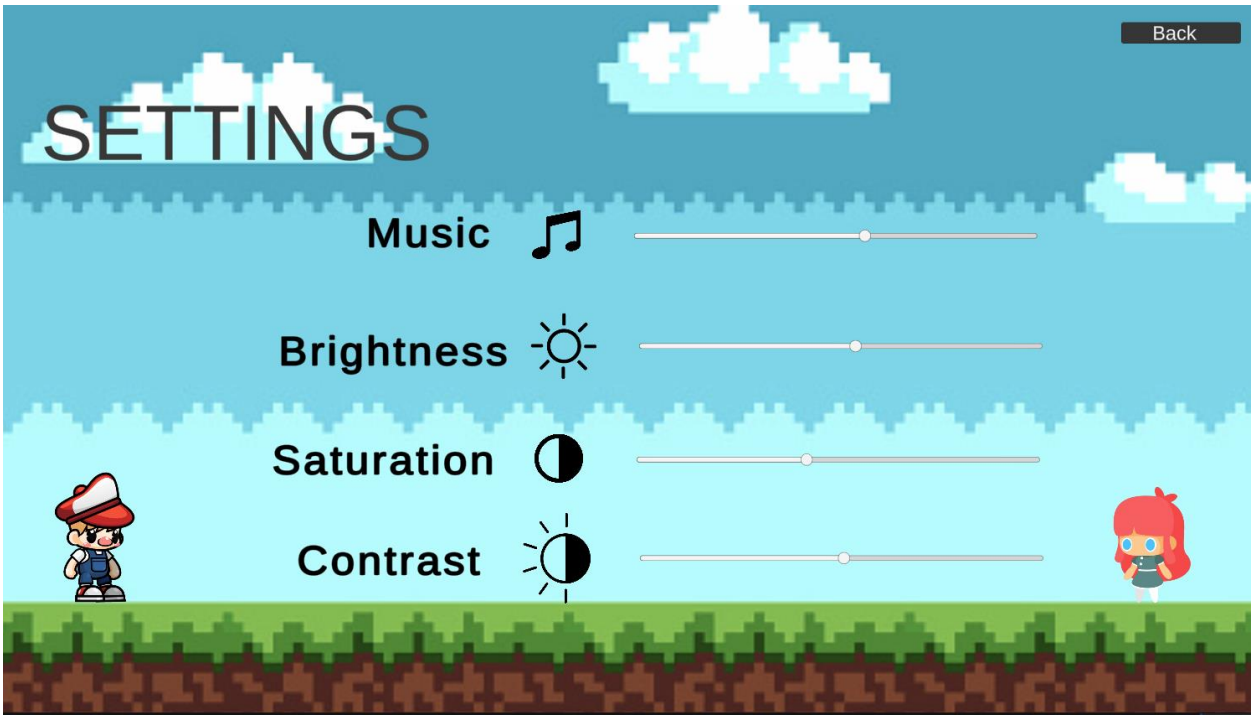


Figure 122: Before changing the value of the Music Slider

Figure 123: After changing the value of the Music Slider

4.1.5. Brightness Slider Testing

Test Case	5
Action	Brightness Slider value is changed.
Expect Result	Slider value to change and Brightness to change.
Actual Result	Slider value and Brightness was changed.
Conclusion	Successful

Table 14: Testing Brightness Slider

Figure 124: Before changing the value of the Brightness Slider

Figure 125: After changing the value of Brightness Slider

4.1.6. Saturation Slider Testing

Test Case	6
Action	Saturation Slider value is changed.
Expect Result	Slider value to change and Saturation to change.
Actual Result	Slider value and Saturation was changed.
Conclusion	Successful

Table 15: Testing Saturation Slider

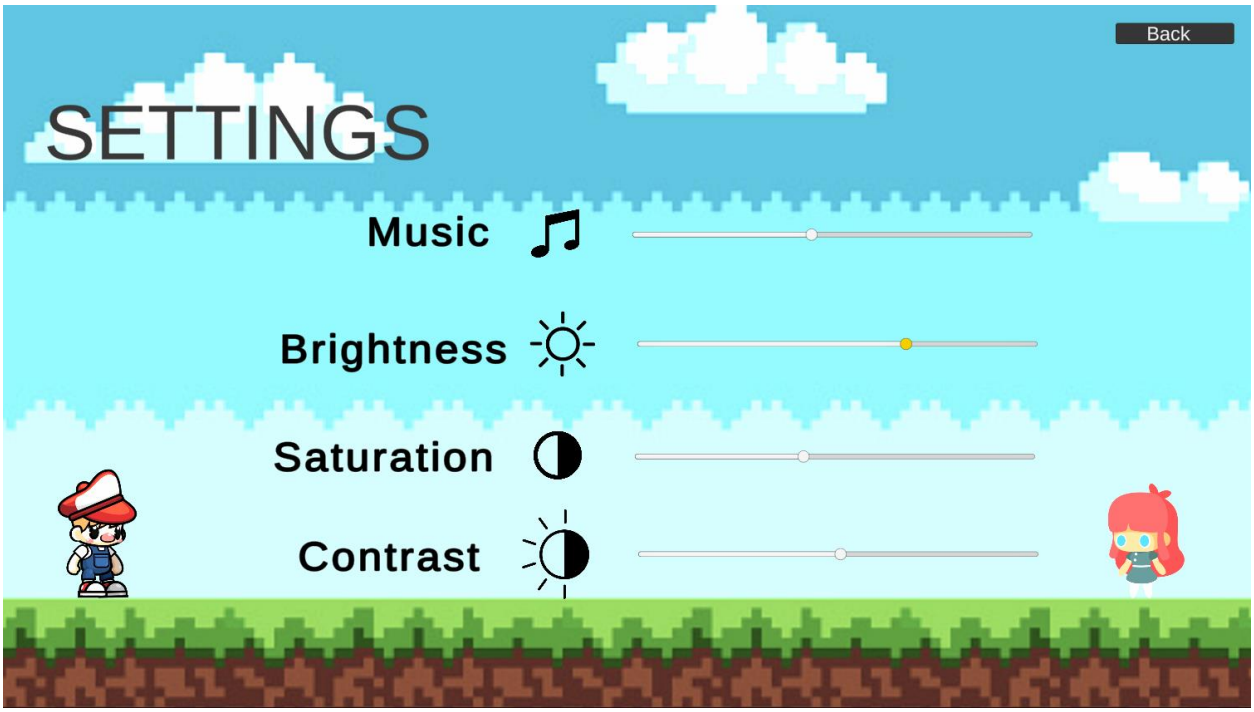


Figure 126: Before changing the value of Saturation Slider

Figure 127: After changing the Value of Saturation Slider

4.1.7. Contrast Silder Testing

Test Case	7
Action	Contrast Slider value is changed.
Expect Result	Slider value to change and Contrast value to change.
Actual Result	Slider value and Contrast was changed.
Conclusion	Successful

Table 16: Testing Contrast Slider

Figure 128: Before Changing the Value of Contrast

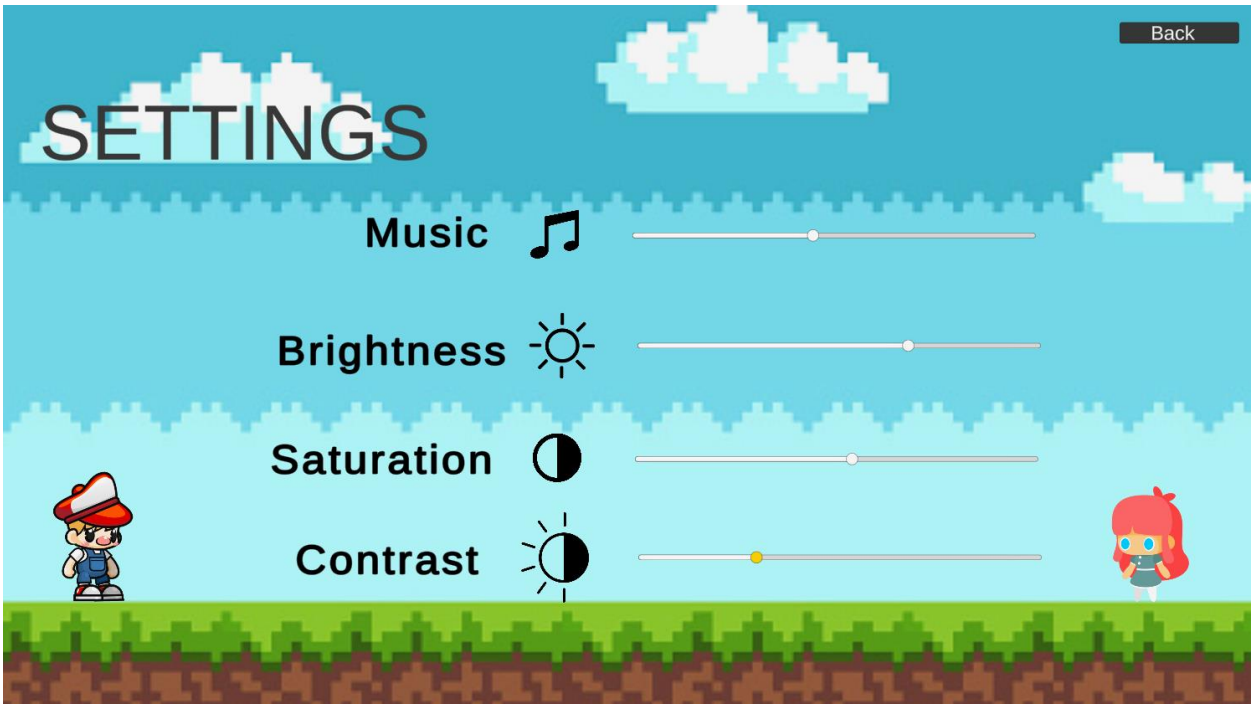


Figure 129: After changing the Value of Contrast

4.1.8. Back button of Settings Testing

Test Case	8
Action	Back button was pressed.
Expect Result	Main Menu to be open.
Actual Result	Main Menu was opened.
Conclusion	Successful

Table 17: Testing back button of Settings

Figure 130: Before pressing Back Button

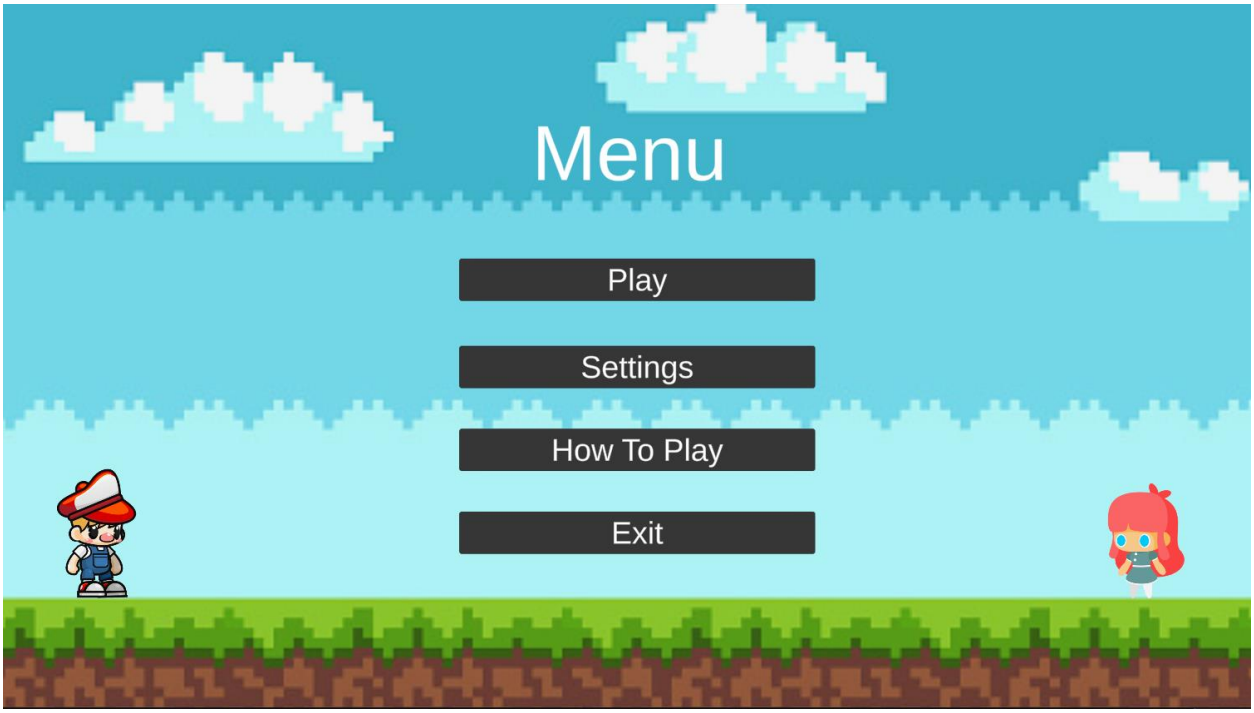


Figure 131: After Pressing Back Button

4.1.9. How to Play Button Testing

Test Case	9
Action	How to Play button was pressed.
Expect Result	How To Play Panel to open.
Actual Result	How to Play Panel was opened.
Conclusion	Successful

Table 18: Testing How to Play Button

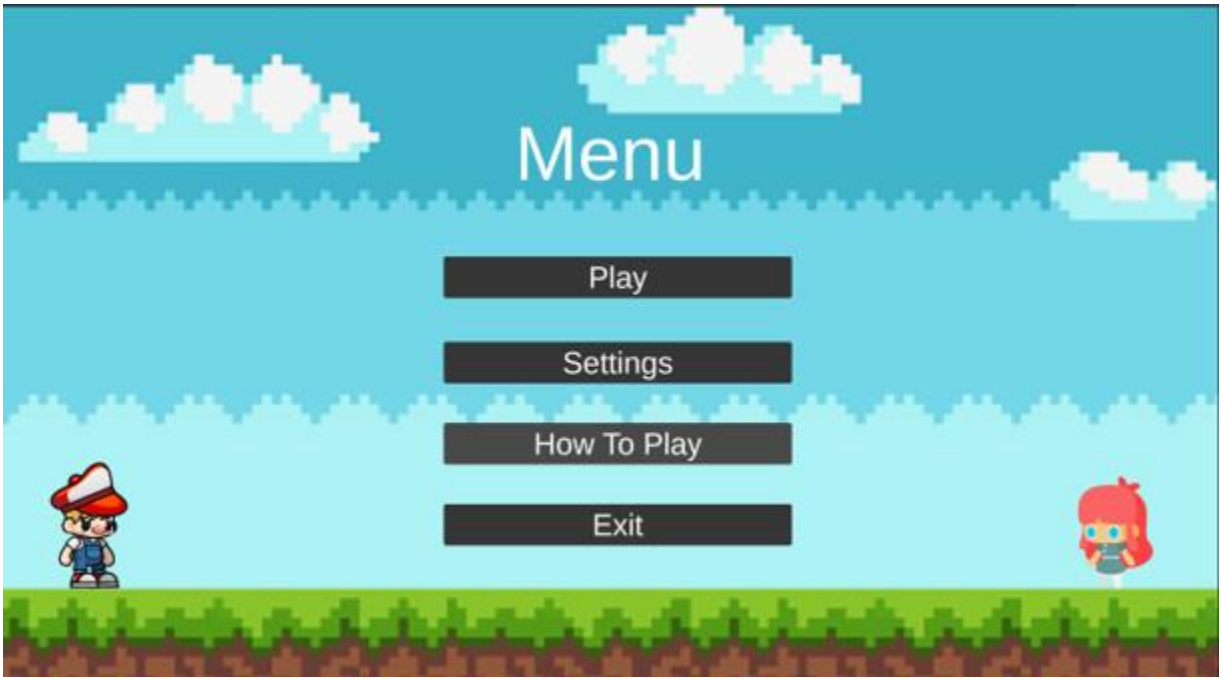


Figure 132: Before Pressing the How to Play button

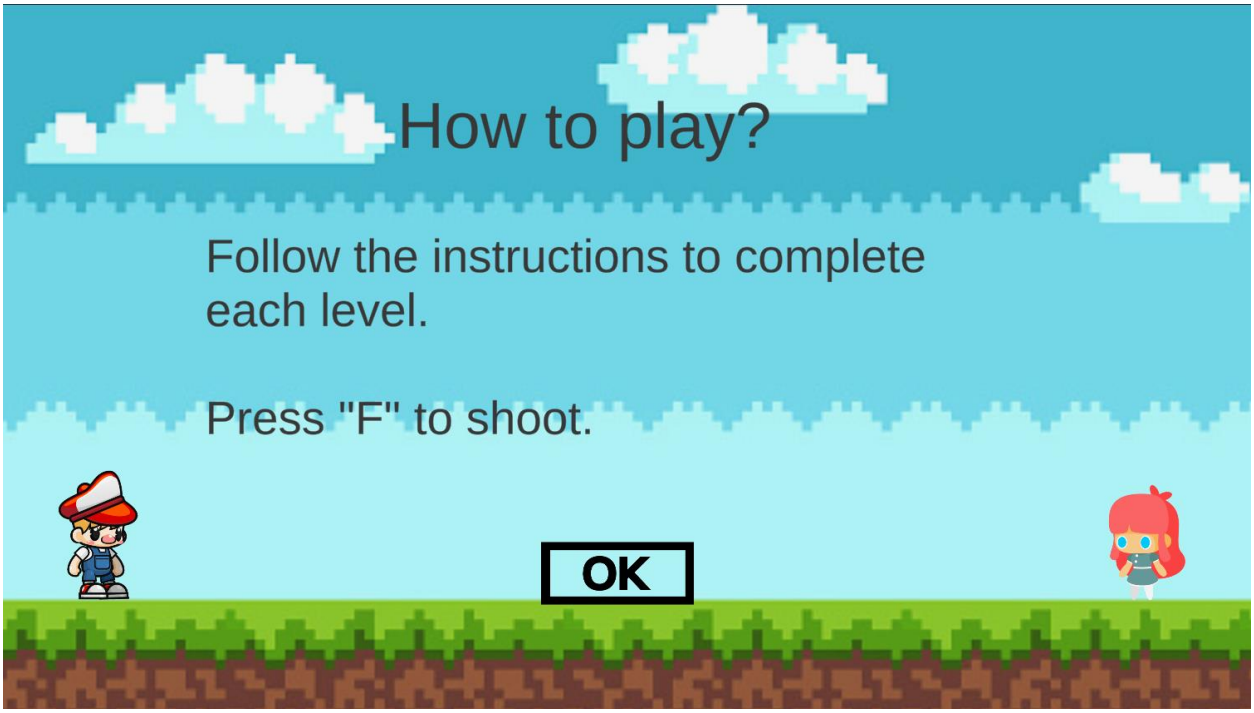


Figure 133: After Pressing the How to Play Button

4.1.10. OK Button of How to Play Panel Testing

Test Case	10
Action	OK was pressed.
Expect Result	Main Menu to open.
Actual Result	Main Menu was opened.
Conclusion	Successful

Table 19:Testing Okay button of How to Play Panel

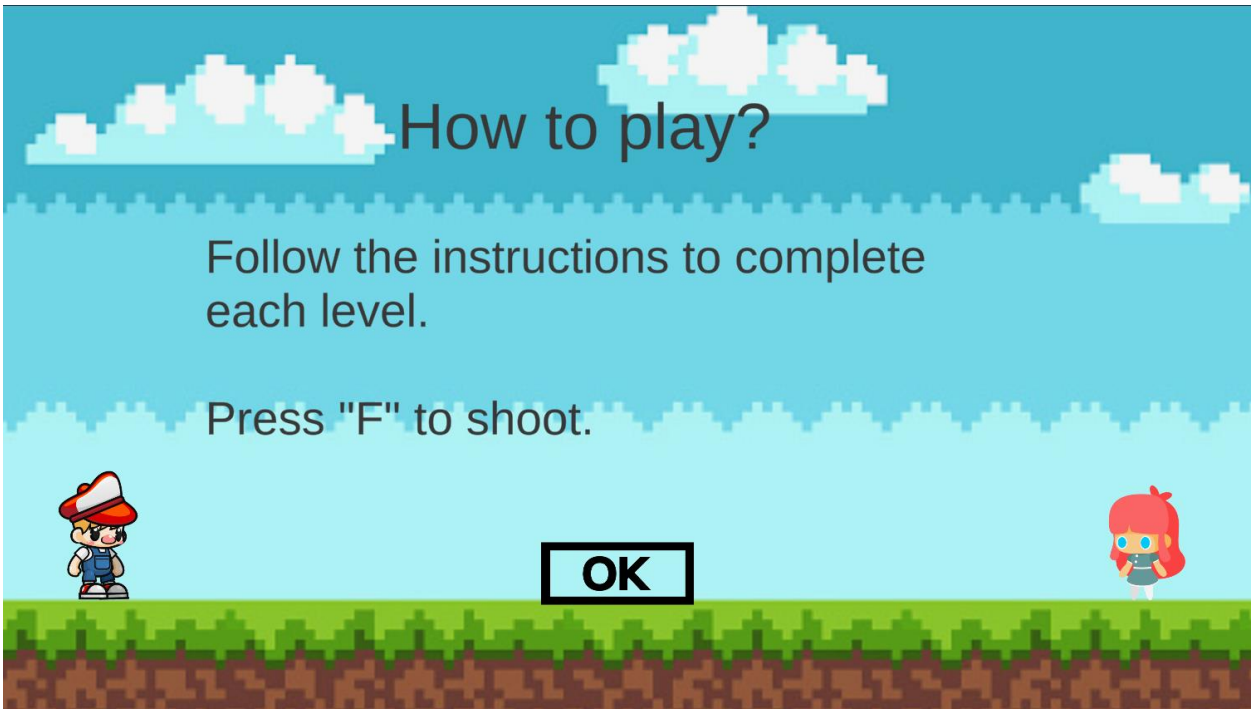


Figure 134: Before Pressing OK Button

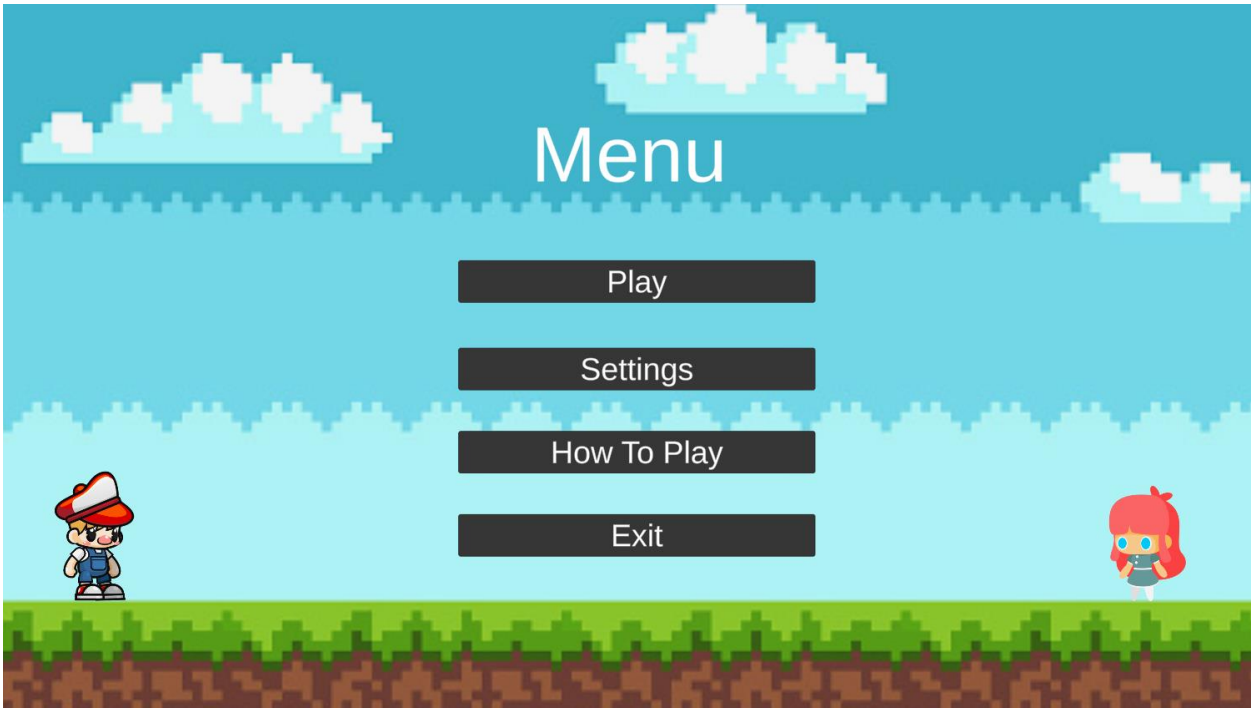


Figure 135: After Pressing OK Button

4.1.11. Exit Button Testing

Test Case	11
Action	Exit was pressed.
Expect Result	Game to closed.
Actual Result	Game was closed.
Conclusion	Successful

Table 20: Testing Exit Button

Figure 136: Before Pressing the Exit Button

4.1.12. Hover for Next button Testing

Test Case	12
Action	Next Button was hover.
Expect Result	Color of the button to be change.
Actual Result	Color of the button was changed.
Conclusion	Successful

Table 21: Testing Hover for Next button

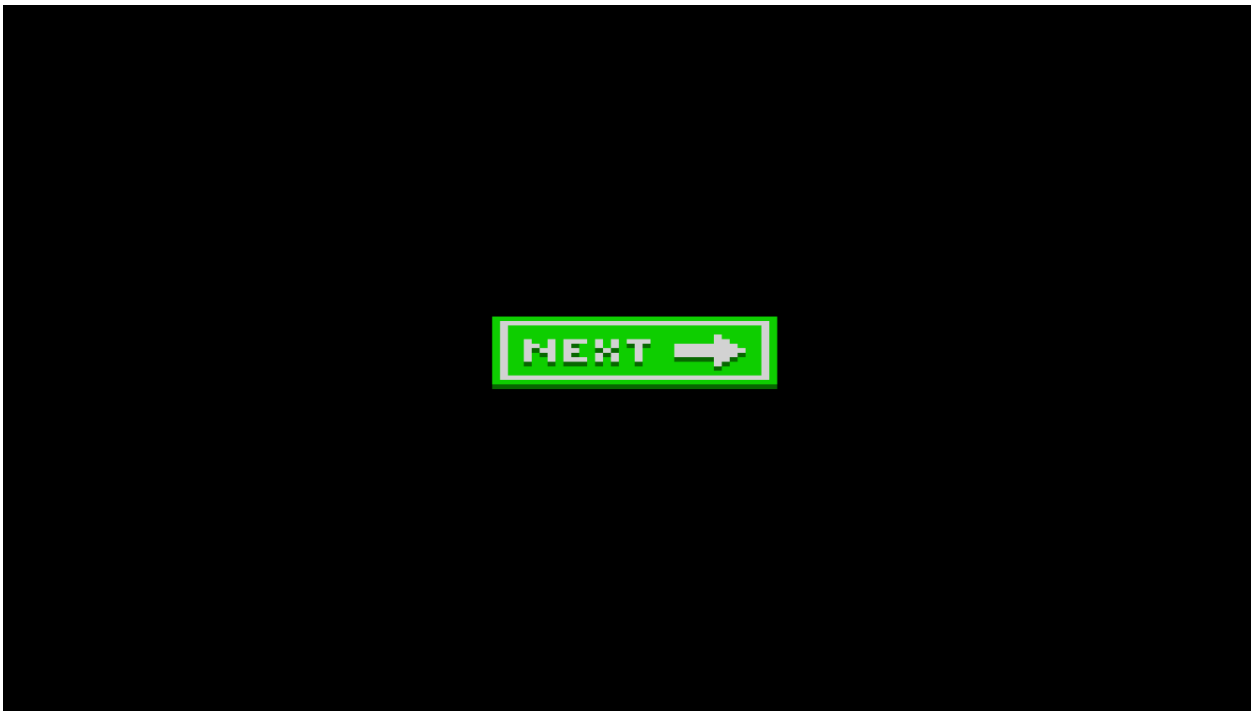


Figure 137: Before Hover

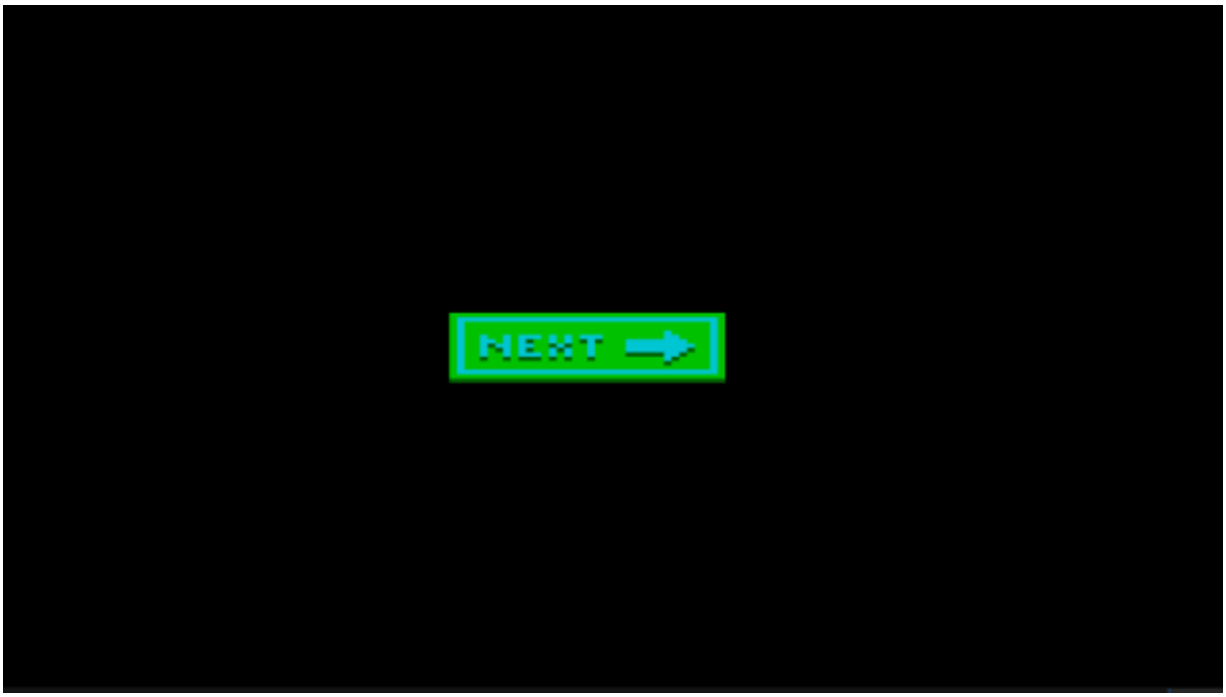


Figure 138: After Hover

4.1.13. Next Button Testing

Test Case	13
Action	Next Button was pressed.
Expect Result	Next Scene to be open.
Actual Result	Next scene was opened.
Conclusion	Successful

Table 22: Testing Next Button

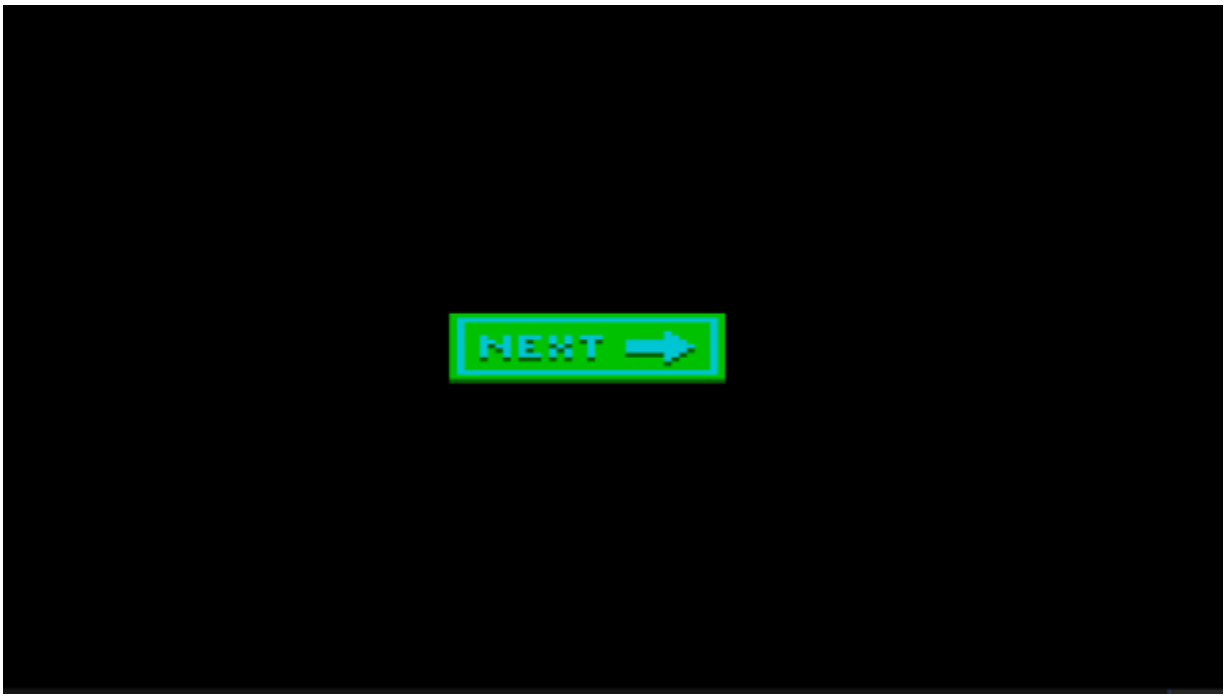


Figure 139: Before Next Button was pressed

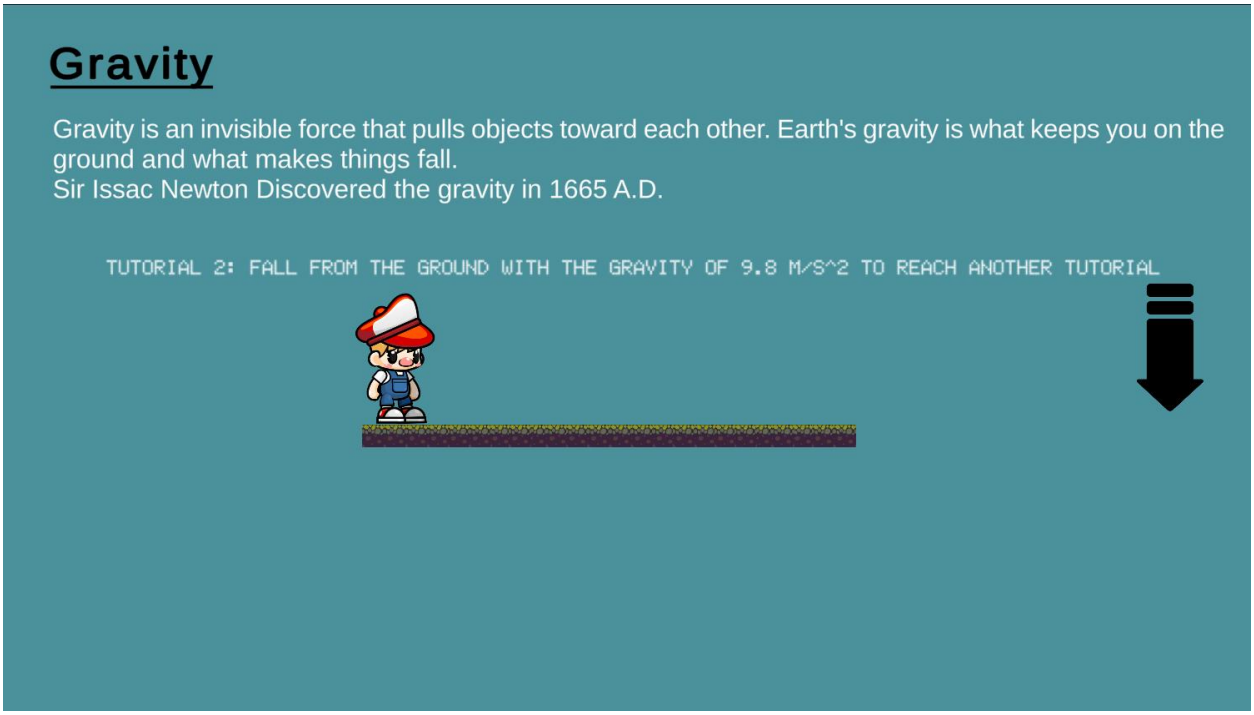


Figure 140: After Next Button Was Pressed

4.1.14. Movement Button Testing

Test Case	14
Action	Movement was pressed.
Expect Result	Player to move.
Actual Result	Player moved.
Conclusion	Successful

Table 23: Testing Movement Button

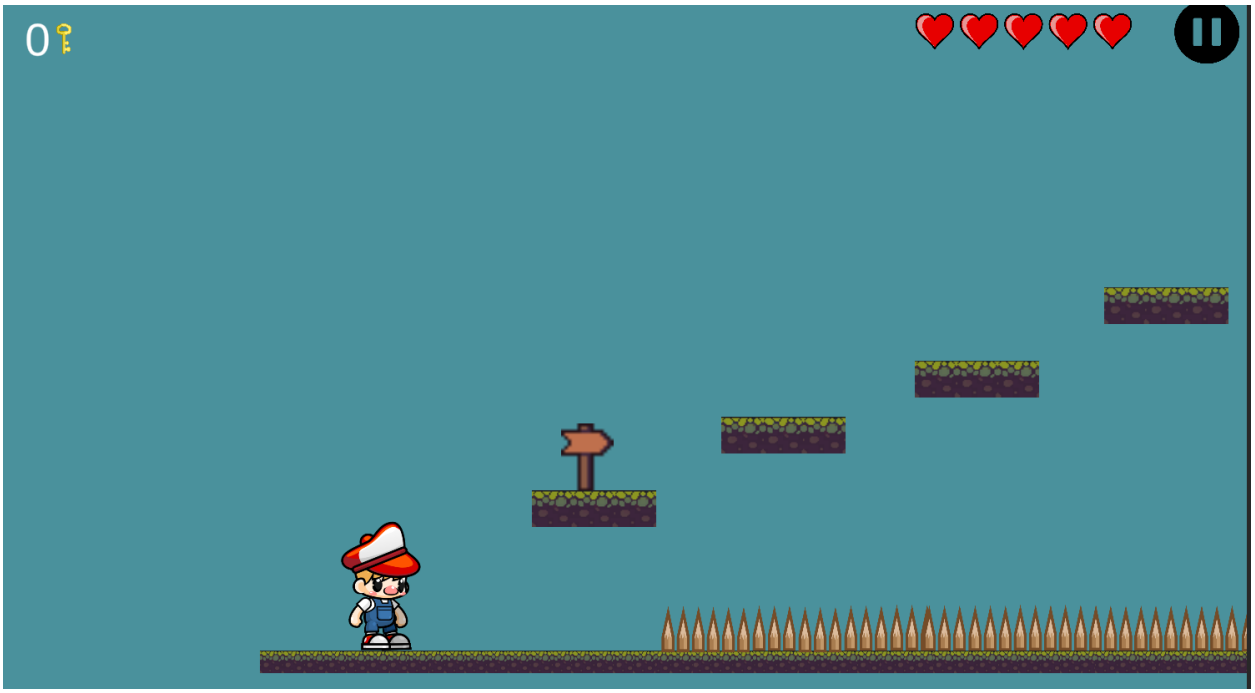


Figure 141: Before Pressing Movement Button

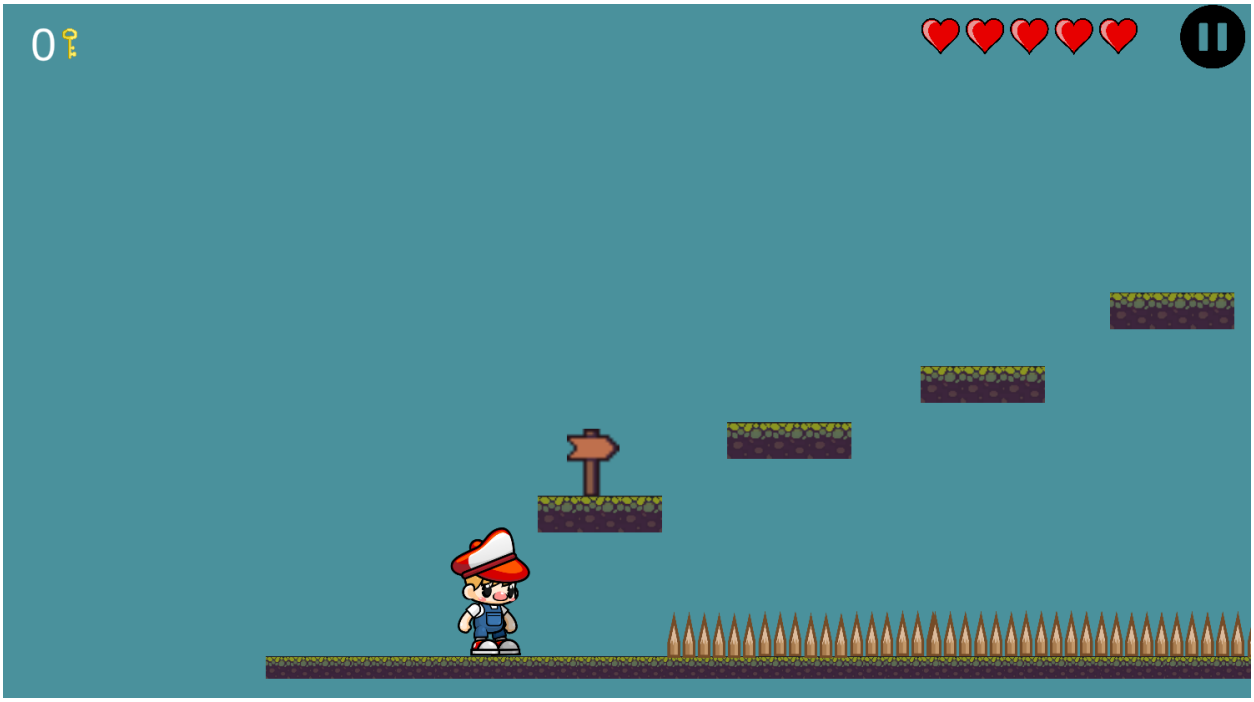


Figure 142: After Pressing Movement Button

4.1.15. Jump Button Testing

Test Case	15
Action	Jump was pressed.
Expect Result	Player to jump.
Actual Result	Player jumped.
Conclusion	Successful

Table 24: Testing Jump Button

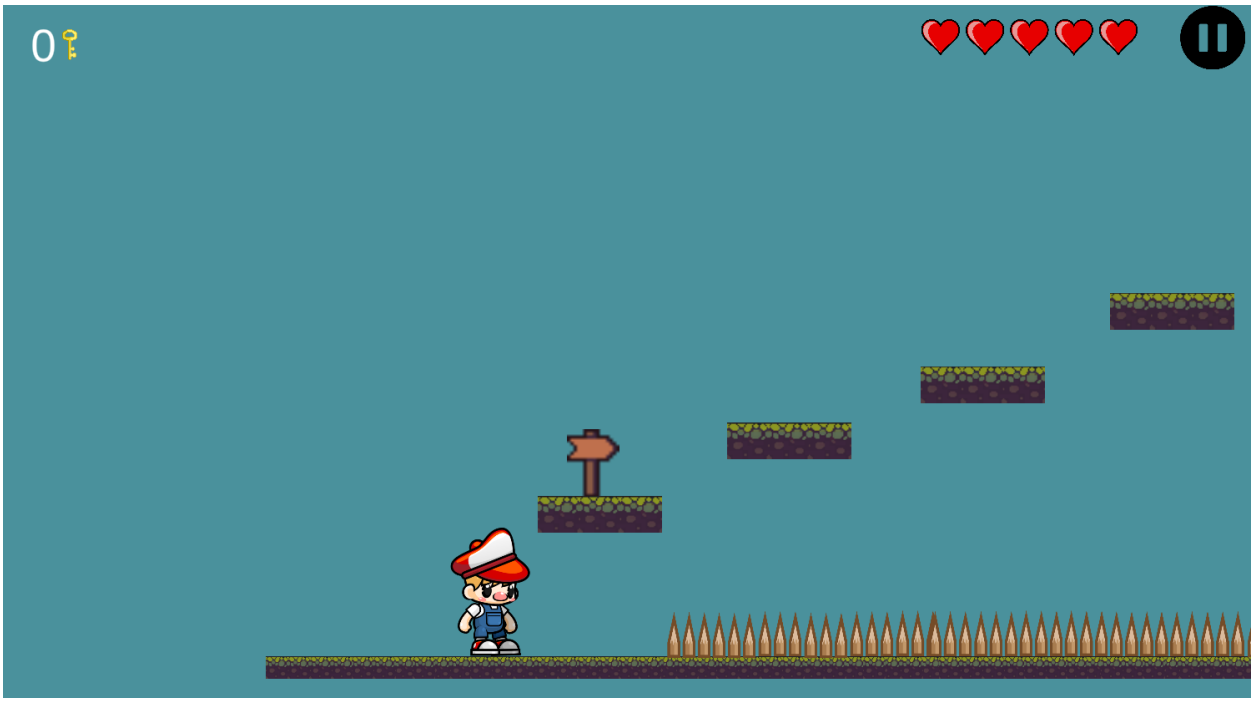


Figure 143: Before the Jump Button was pressed

Figure 144: After the Jump Button was Pressed

4.1.16. Pause Button Testing

Test Case	16
Action	Pause was pressed.
Expect Result	Paused Panel to Open.
Actual Result	Paused Panel is Open.
Conclusion	Successful

Table 25: Testing Paused Button

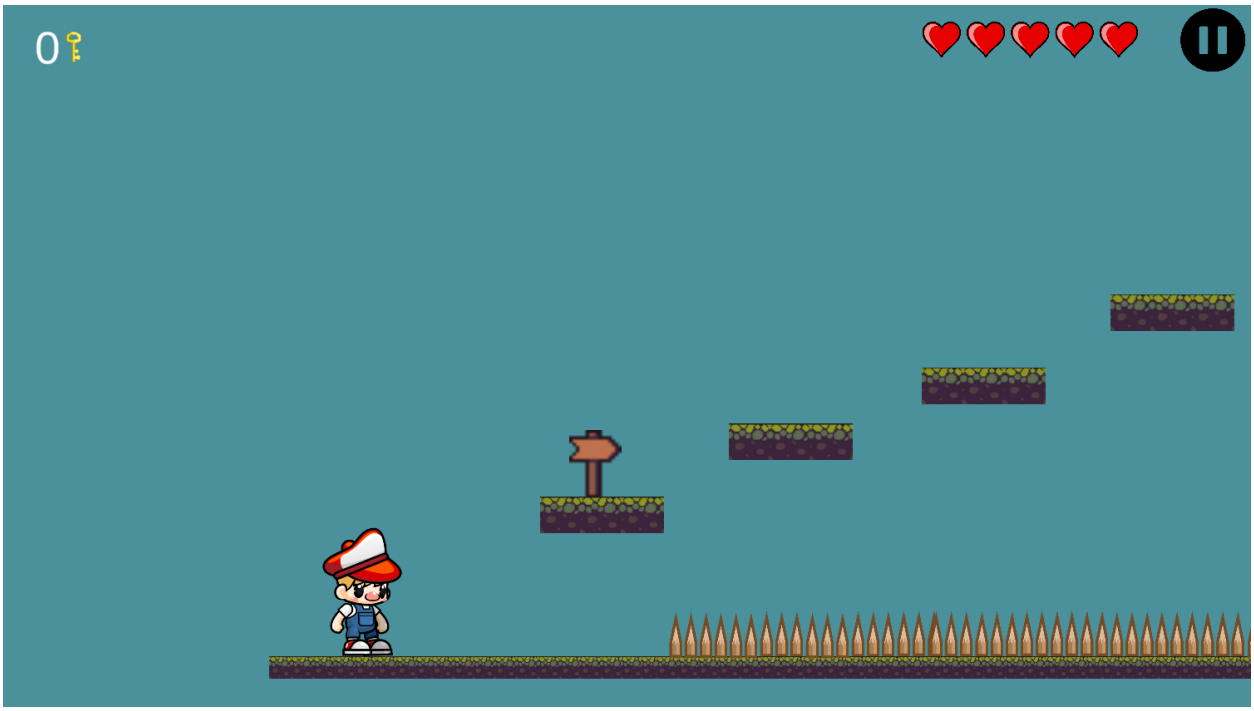


Figure 145: Before Paused Button

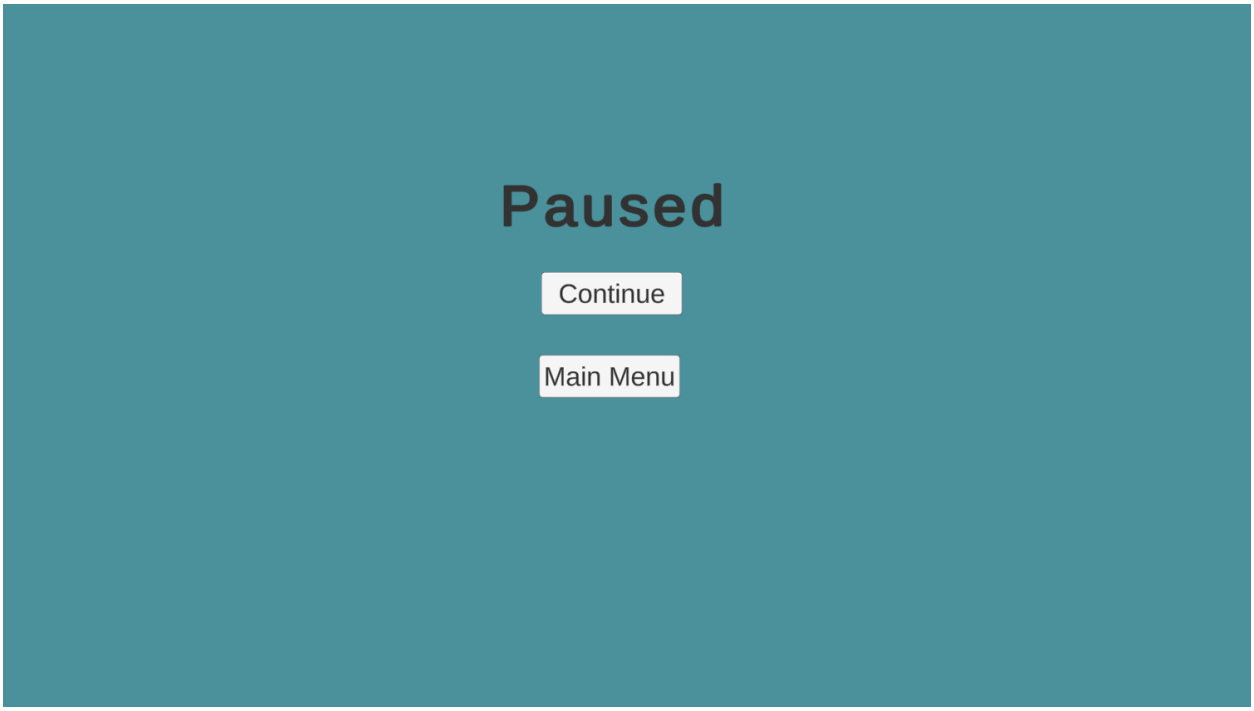


Figure 146: After Pressing Paused Button

4.1.17. Hover for Paused Panel Testing

Test Case	17
Action	Hover was over button.
Expect Result	Color of the button to be changed.
Actual Result	Color of the button is changed.
Conclusion	Successful

Table 26: Testing Hover for Paused Panel

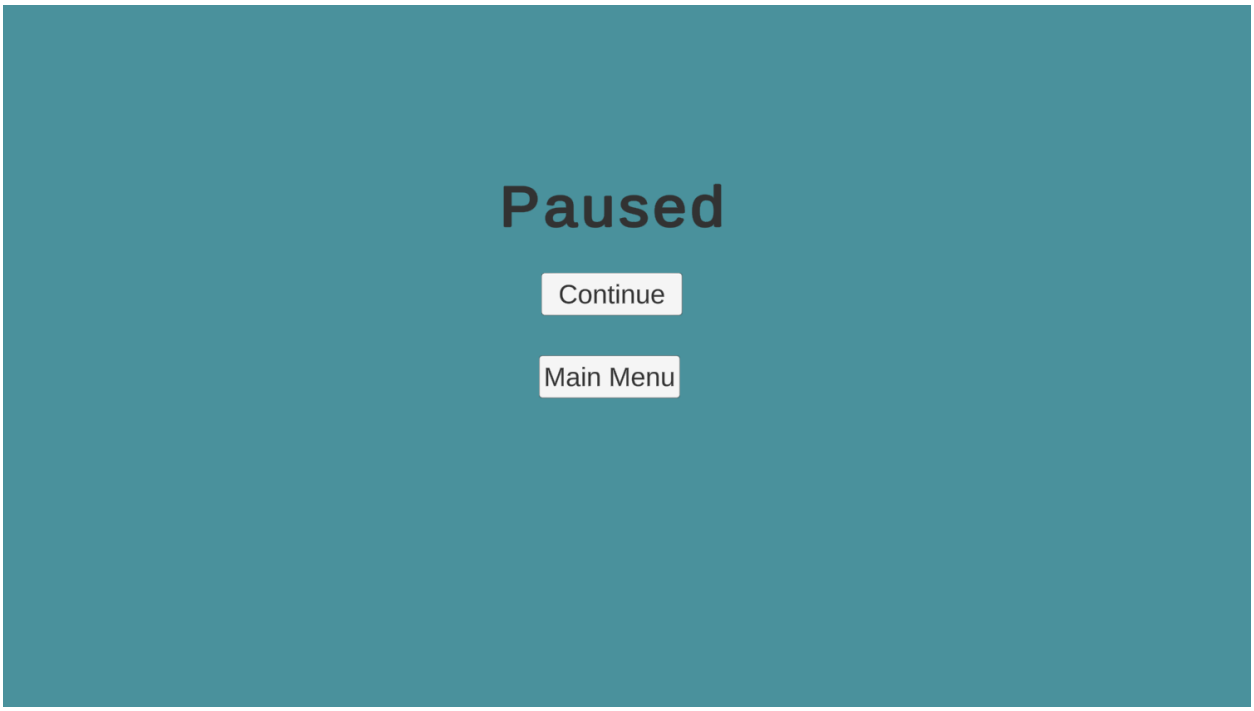


Figure 147: Before Hovering Continue Button

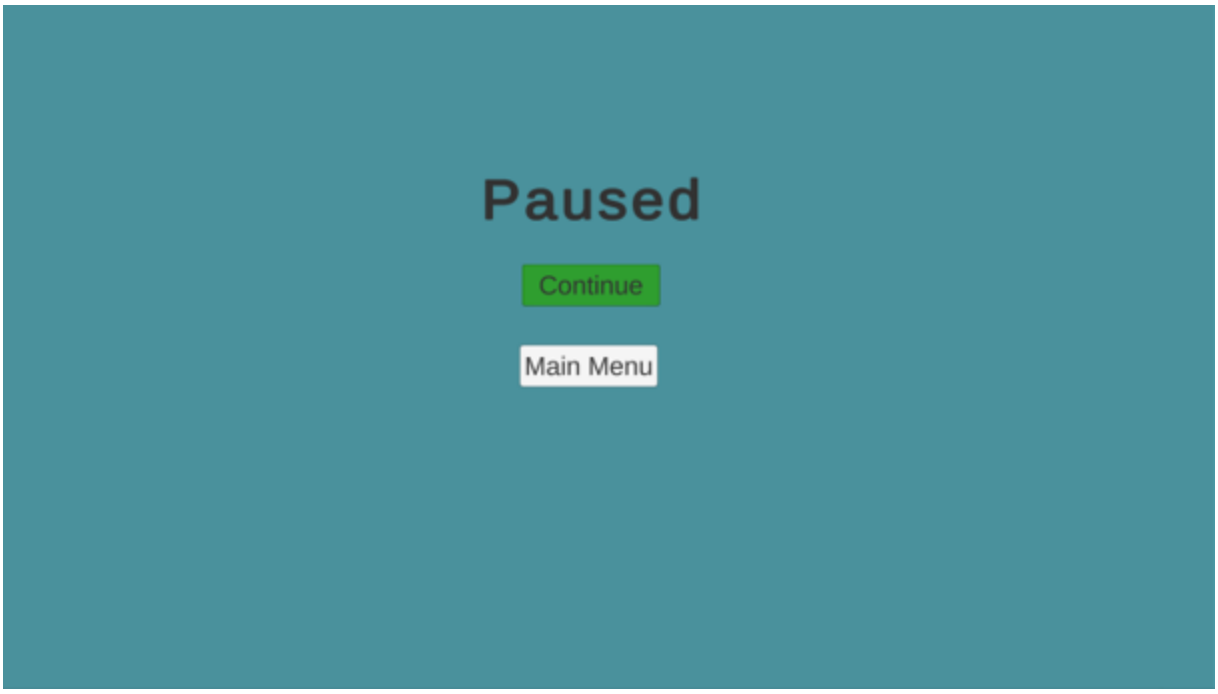


Figure 148: After Hovering Continue Button

4.1.18. Continue Button of Paused Panel Testing

Test Case	18
Action	Continue Button was pressed.
Expect Result	To take back to the Game scene.
Actual Result	Nothing happened.
Conclusion	Failed

Table 27: Testing Continue Button of Paused Panel

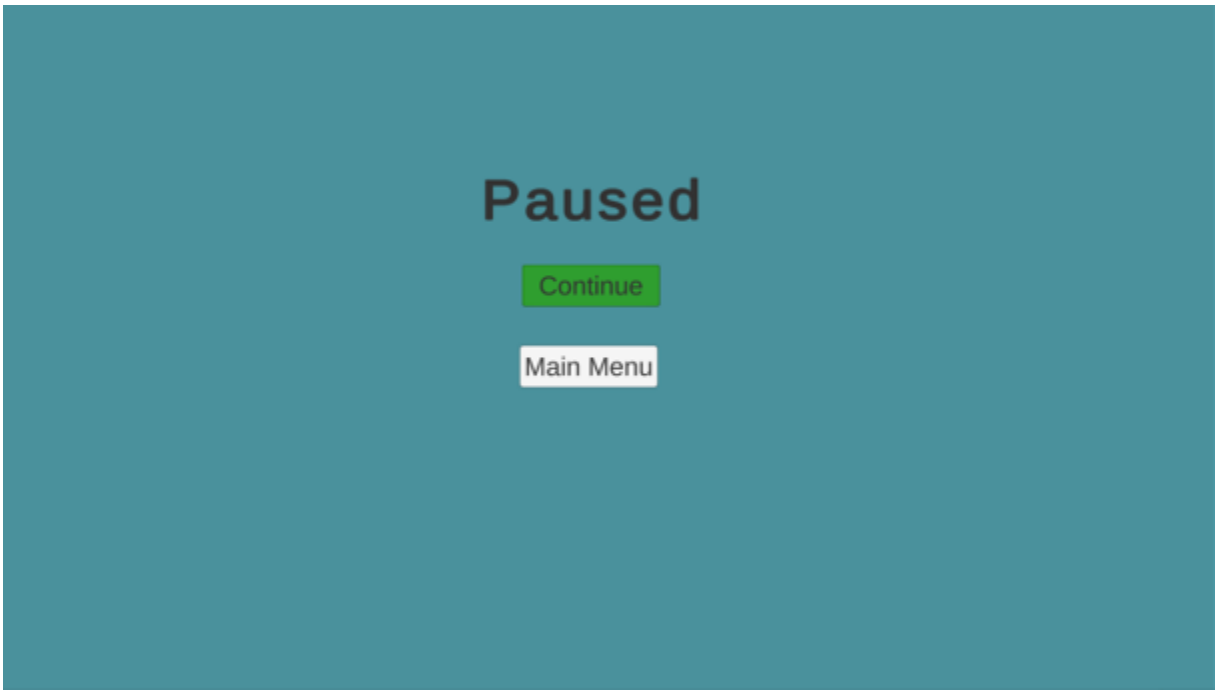


Figure 149: Before Pressing Continue Button

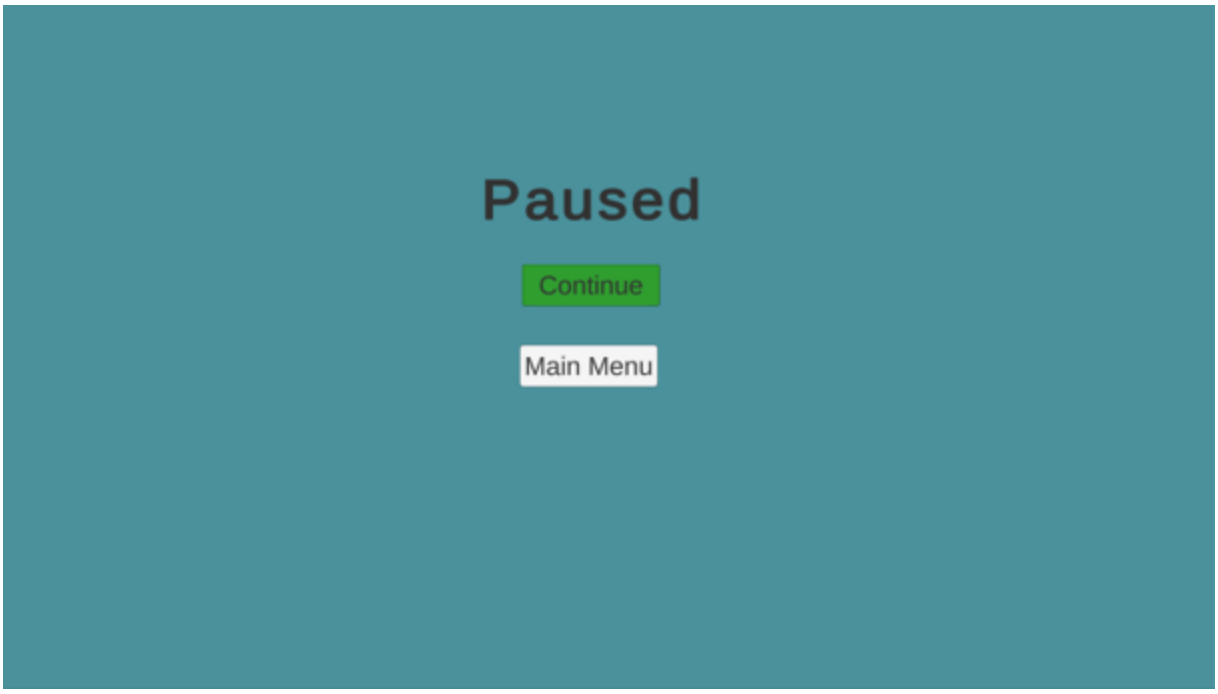


Figure 150: After Pressing Continue Button

4.1.18.1. Fixing the Continue Button:

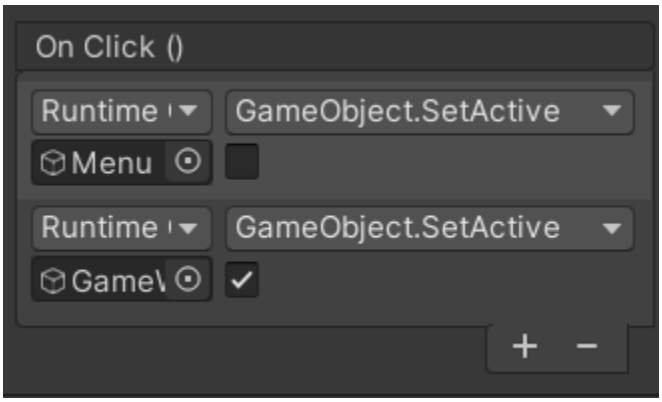


Figure 151: Setting on click () Function for Continue button

4.1.18.2. Testing again after fixing the on click () function for the Continue Button:

Test Case	19
Action	Continue Button was pressed.
Expect Result	To take back to the Game scene.
Actual Result	Continued back to Game scene.
Conclusion	Successful

Table 28: Testing Continue button of Settings Panel After Fixing

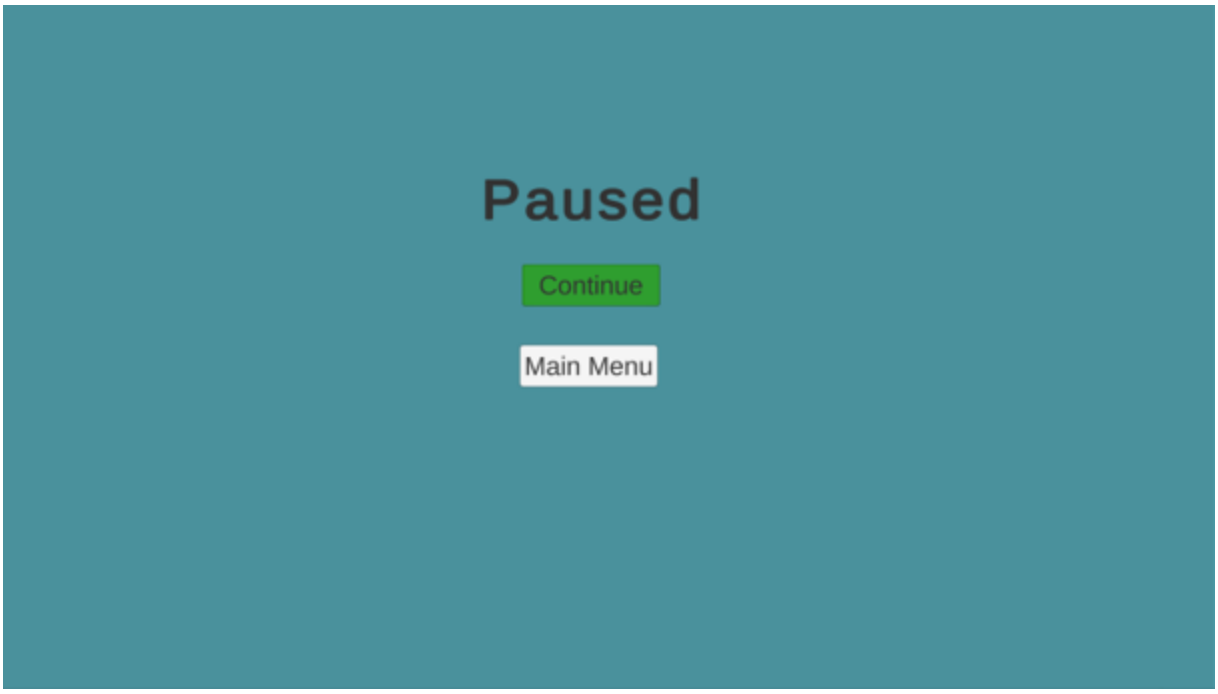


Figure 152: Before Pressing Continue Button, after getting fixed

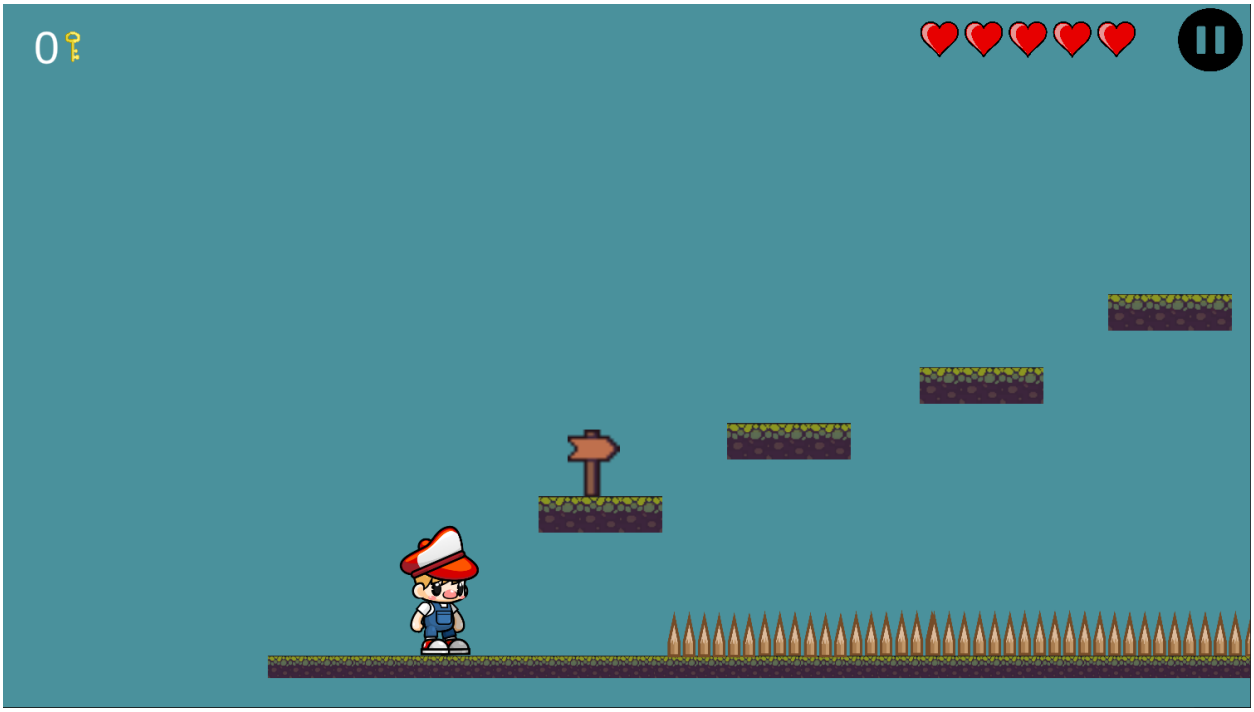


Figure 153: After Pressing the Continue Button, after getting fixed

4.1.19. Main Menu button Testing

Test Case	20
Action	Main Menu Button was pressed.
Expect Result	To take back to the Main Menu.
Actual Result	Nothing happened.
Conclusion	Failed

Table 29: Testing Main Menu Button

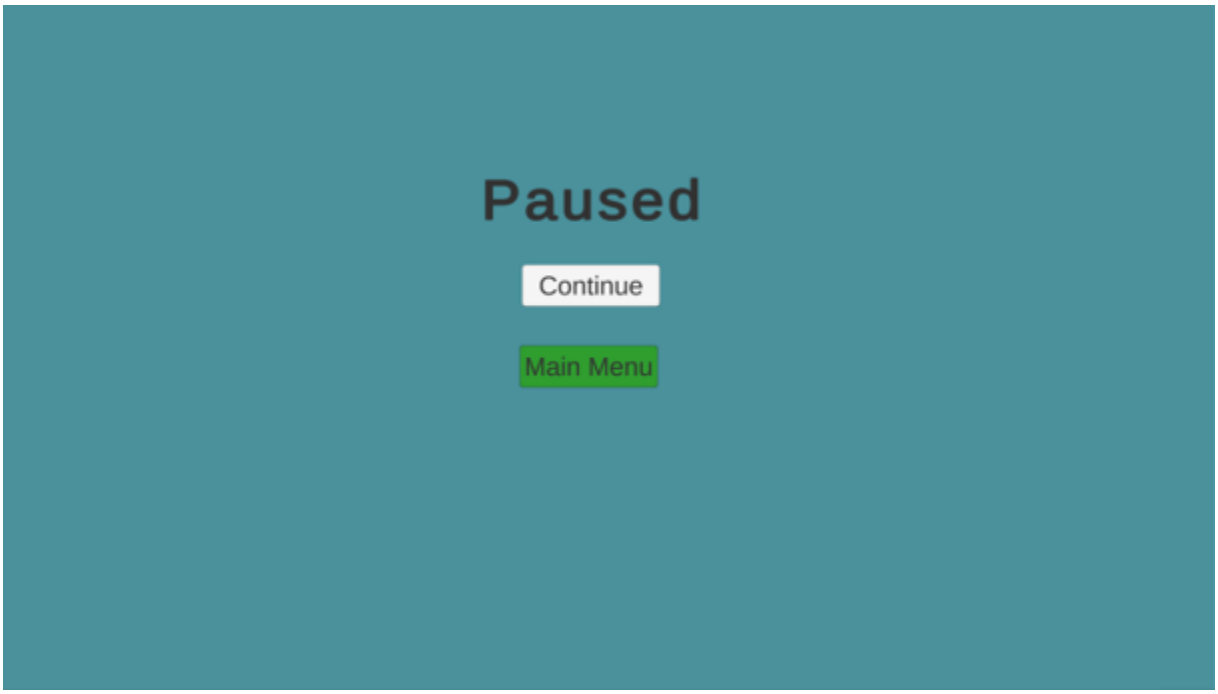


Figure 154: Before Pressing Main Menu Button

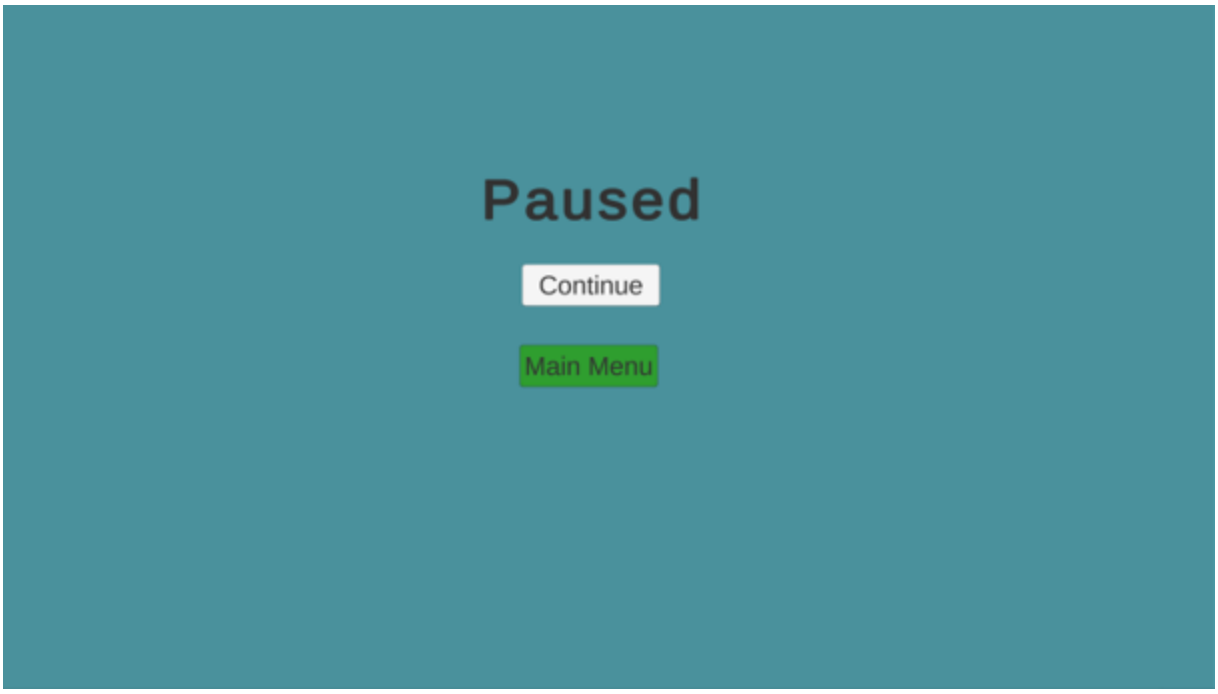


Figure 155: After Pressing Main Menu Button

4.1.19.1. Fixing the Main Menu Button:

```
public void MainMenu()
{
    SceneManager.LoadScene(0);
}
```

Figure 156: Function is created in the script

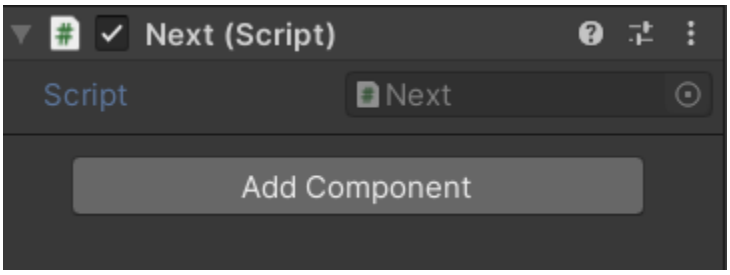


Figure 157: Putting Script in Menu Object

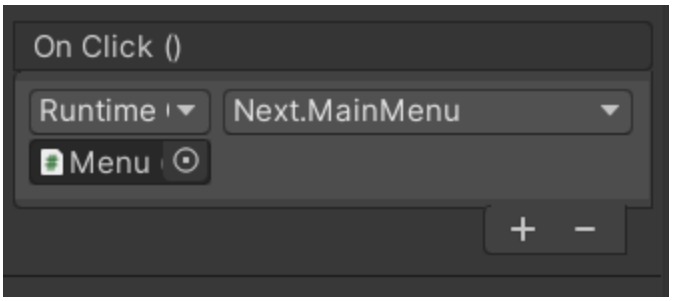


Figure 158: Using the MainMenu () Function from Next script which is the component of Menu Object

4.1.19.2. Testing after fixing the Main Menu Button

Test Case	21
Action	Main Menu Button was pressed.
Expect Result	To take back to the Main Menu.
Actual Result	Main Menu scene is opened.
Conclusion	Successful

Table 30: Main Menu button Testing after fixing it

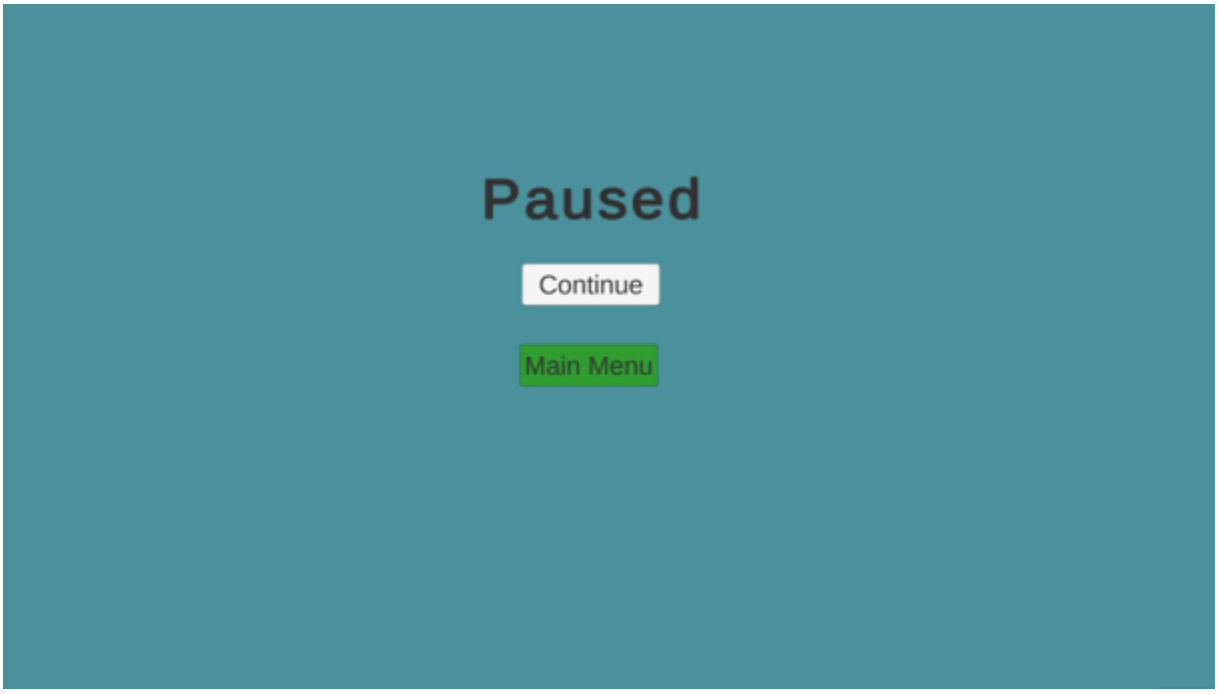


Figure 159: Before pressing the Main Menu Button, after fixing it

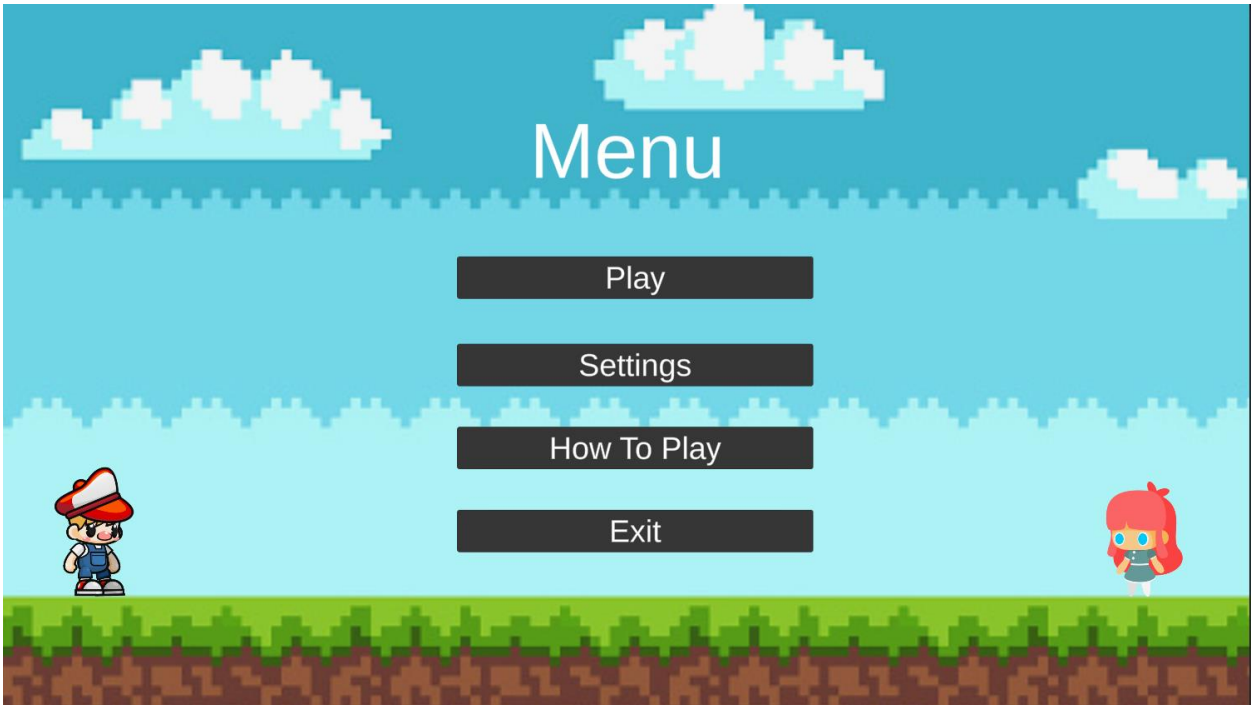


Figure 160: After pressing the Main Menu Button After fixing it

4.1.20. OK Button Testing in Congrats Scene

Test Case	21
Action	OK was pressed.
Expect Result	To take back to the Main Menu.
Actual Result	Main Menu scene is opened.
Conclusion	Successful

Table 31: Testing OK Button in Congrats Scene

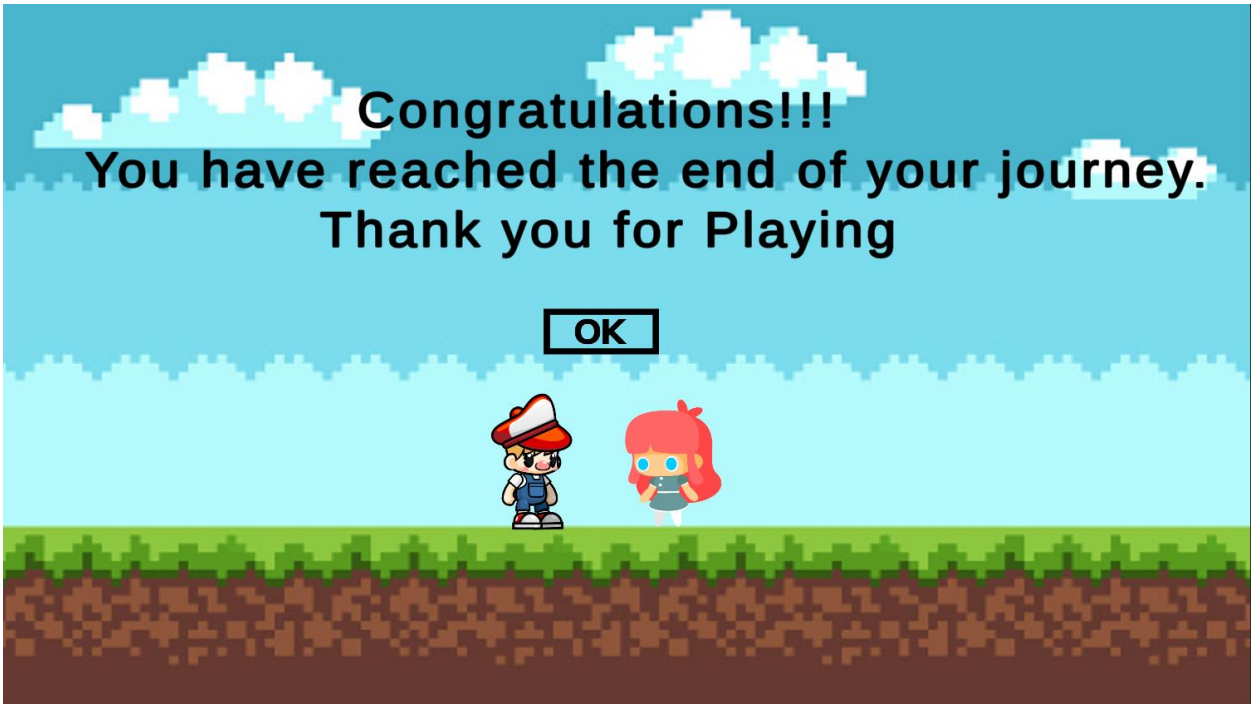


Figure 161: Before Pressing OK Button

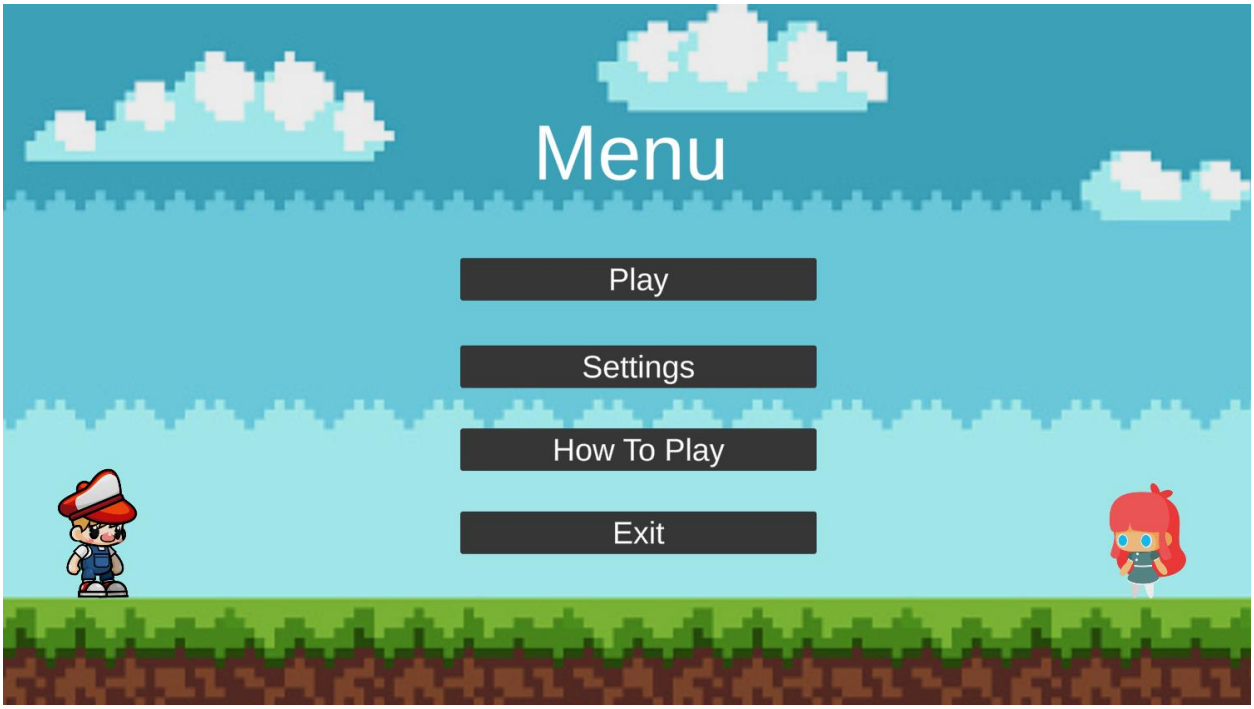


Figure 162: After Pressing Okay Button

4.2. Performance Testing

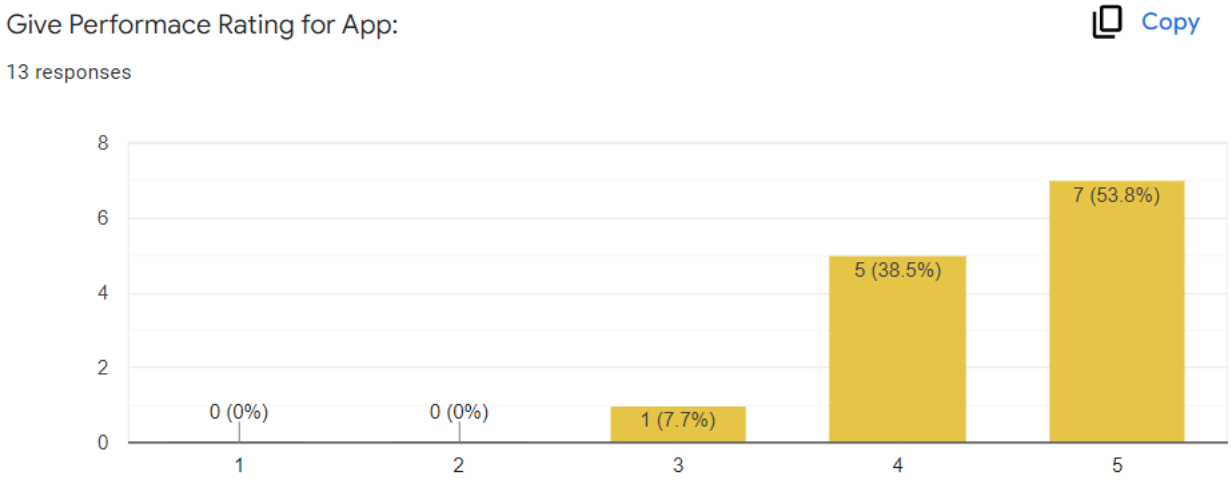


Figure 163: Performance Rating by Users

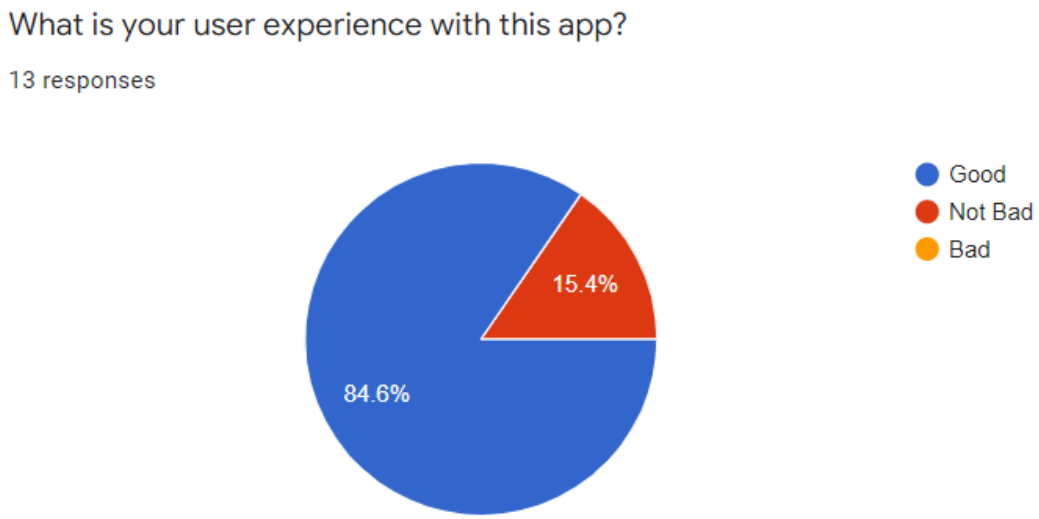


Figure 164: User experience of the Users

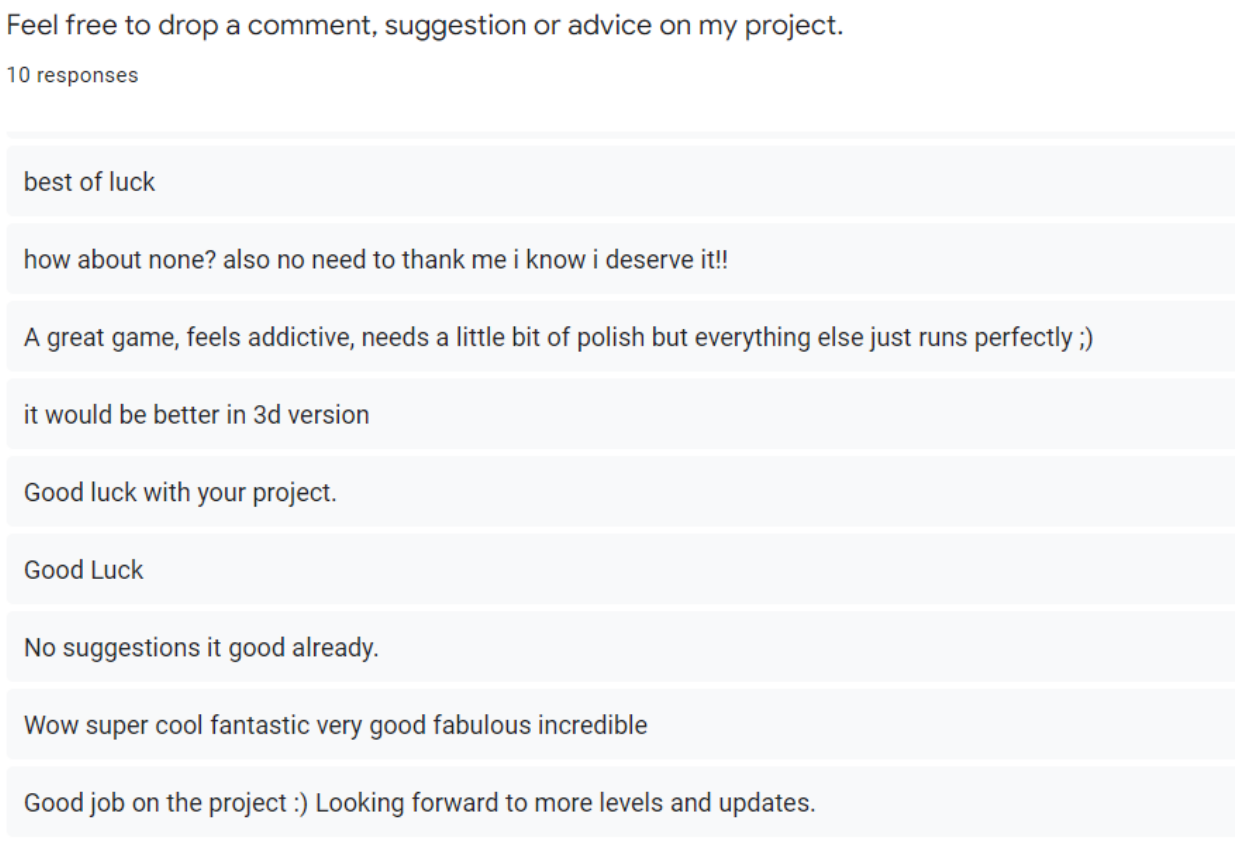


Figure 165: User's comment on app

4.3. Compatibility Testing

Runs on: Windows 10/ Windows 8/ Windows 7/ Windows Vista/ XP

Storage Requirements: 35.5 MB (37,232,640 bytes)

Hardware requirement: Keyboard, Screen and Sound-system

Processor requirement: HP intel core and above

Graphic card: Not needed

Some of the client devices that were used are as follows:

- Dell vostro 3500
- Acer Nitro 5
- Acer Nitro 7
- Mac Book Air M1
- HP spectre x360
- Lenovo Yoga 510
- Acer Aspire 5
- MSI GF75
- Dell Inspiron 15 5000 series

Chapter 5 Conclusion

5.1. Project Evaluation

The proposal to create an educational game application was made with the goal of assisting students in their studies and resolving teacher and parent concerns. Several studies have been conducted on educational games and the responses of teachers and parents to educational games. This application was created from the ground up to meet all of the users' functional, non-functional, and usable requirements. The application was finished on time and with all of the features specified in the proposal.

The GDLC model was used in the development process. The use of the same model proved to be extremely beneficial in completing the project on time. After extensive research, the GDLC model appeared to be the best fit for the project out of all the methodologies considered.

The project's completion provided a large number of learning outcomes and experience. The project took two iterations to complete. The first iteration focused on completing the project's main features such as sending reports, listing the reports, listing the articles, and so on. The second iteration was more focused on generating the required ideas for the game and designing the game. The second iteration focused more on improving the animation, implementing the educational component, and making the experience fun and entertaining for users.

Completing the entire application in the allotted time frame was not an easy task. There was a lot of pressure because there were so many mistakes. Some of the topics required extensive research, and the majority of them were also novel. Coming up with educational components that could be included in the game was a difficult part of this project because it was an educational game. However, by dedicating sufficient time and effort, the application was successfully completed.

5.2. Critical Analysis

When I attempted to start the application, it appeared to be a massive and extremely difficult task. Education games are a relatively new phenomenon in the world. It's difficult to strike a balance between fun, entertainment, and educational content. Many ideas were proposed, and many were rejected. Many studies and analyses were conducted. Many YouTube videos were watched in order to learn more about game designing and development and managing the time. The application was successfully completed as a result of this extensive research and effort.

The completion of the application resulted in several learning outcomes; they are:

- Working with unity
  Almost 80% of the work of the project is done in the unity. The knowledge of unity UI and experience about unity if gained form this project.
- Working with Animations
- Scripting
- Working with adobe illustrator
- Working with unity components
- Error Handling and debugging
- Working with lighting and Universal Renderer Pipeline in unity

5.3. LEGAL, SOCIAL AND ETHICAL ISSUES

5.3.1. Legal Issues

The following Legal issues may arise as a result of the application:

- Copyright: If anyone try to use my game assets without my permission then it will lead to copyright.

5.3.2. Social Issues

The following legal issues may arise as a result of the application:

- Social Skills: Players can be too addicted to the game and may not interact with other people, which may directly or indirectly affect their social skills.

5.3.3. Ethical Issues

The following Ethical issues may arise as a result of the application:

- Game addiction: It can be an issue.
- Violence: Player may try to idolize the main character of the game and may try to practice violence.

5.4. Advantages

- It Helps with Hand-Eye Coordination.
- It Improves Problem-Solving & Strategic Thinking.
- It Expands Memory Capacity.
- It helps Kids Who Struggle with Attention Disorders.
- It helps Kids Become More Computer-Literate.
- It helps kids to learn about science.
- It helps kids to spend their free time in learning things.
- It helps kids to improve their grade in science.

5.5. Limitations

- Offline game.
- Limited level.
- Player can use only one character in game.
- Player cannot customize their name.
- Player cannot customize their character.
- Game includes the education part about the physics only

5.6. Future work

Even though the application was completed, and the client was completely satisfied, there are still some things that need to be done in the future to improve the application:

i. Adding levels: There are only limited levels in the game.

ii. Customizing character: In the feedback given by users, they want the character customizing feature.

iii. Customizing name: In the feedback given by users, they want the character name feature.

iv. Skins or New character: In the feedback given by users, they want the new playable character and skin for characters.

v. Education content about other topics: Some of the teacher and parents wants other educational contents in game.

Chapter 6 References

Age of Learning, I., 2021. *google playstore.* [Online] Available at: https://play.google.com/store/apps/details?id=com.aofl.adventureacademy&hl=en&gl=US [Accessed 12 Dec 2021].

Ansari, M., 2011. *Game Development Tools.* Boca Raton: FL: Taylor and Francis Group.

Bigelow, S. J., 2021. *techtarget.* [Online] Available at: https://www.techtarget.com/searchenterprisedesktop/definition/Windows-10#:~:text=Windows%2010%20is%20a%20Microsoft,follow%2Dup%20to%20Windows%208. [Accessed 2 April 2022].

Chandler, H. M., 2010. *Game Production Handbook.* New York: Jones and Bartetts Publishers.

Grey Springs, 2021. *Google Play.* [Online] Available at: https://play.google.com/store/apps/details?id=com.greysprings.shapesandcolors&hl=en&gl=US [Accessed 10 December 2021].

Hippo Kids Games, 2021. *google playstore.* [Online] Available at: https://play.google.com/store/apps/details?id=com.kbpro.UnderwaterFairyTale&hl=en&gl=US [Accessed 12 Dec 2021].

IDZ Digital Private Limited, 2022. *PlayStore.* [Online] Available at:

https://play.google.com/store/apps/details?id=com.baby.toddler.games&hl=en&gl=US

[Accessed 12 March 2022].

Jacquemet, A. N. a. J., 2019. The Impact of Video Games. *The Impact of Video Games*, Nov.

kidsEducational, B. F. p. l. g. f., 2022. *PlayStore.* [Online] Available at: https://play.google.com/store/apps/details?id=com.educational.baby.games&hl=en&gl=US

[Accessed 12 March 2022].

Market Report Analysis, 2020. *Market Report Analysis,* San Franscisco: Region, And Sement Forecasts.

Pressman, R. S., 2009. *Software Engineering Challeneges in Game Development.* 5th ed. New York: Book Style.

T.Fullerton, 2008. *Game Design Workshop.* 3rd ed. Burlington: Elsevier: Book Style.

Unity, 2022. *Learn Unity.* [Online] Available at: https://learn.unity.com/tutorial/working-with-scripts#

[Accessed 03 July 2022].

Unity, 2022. *Unity3D.* [Online] Available at: https://docs.unity3d.com/Packages/com.unity.package-manager-ui@1.8/manual/index.html#:~:text=A%20package%20is%20a%20container,range%20of%20enhancements%20to%20Unity.

[Accessed 2 April 2022].

Unity, 2022. *Unity3D.* [Online] Available at: https://unity3d.com/quick-guide-to-unity-asset-store#:~:text=the%20website%20version.-,What%20is%20a%20Unity%20Asset%3F,of%20file%20that%20Unity%20supports. [Accessed 3 April 2022].

Unity, 2022. *Unity3D.* [Online] Available at: https://docs.unity3d.com/Manual/class-AnimatorController.html#:~:text=Unity%20automatically%20creates%20an%20Animator,and%20click%20Create%20%3E%20Animator%20Controller. [Accessed 3 April 2022].

Wiley, A., 2020. *advancementcourses.* [Online] Available at: https://blog.advancementcourses.com/articles/educational-benefits-video-games/ [Accessed 02 November 2021].

Chapter 7 Bibliography

Age of Learning, I., 2021. *google playstore.* [Online] Available at: https://play.google.com/store/apps/details?id=com.aofl.adventureacademy&hl=en&gl=US [Accessed 12 Dec 2021].

Ansari, M., 2011. *Game Development Tools.* Boca Raton: FL: Taylor and Francis Group.

Bigelow, S. J., 2021. *techtarget.* [Online] Available at: https://www.techtarget.com/searchenterprisedesktop/definition/Windows-10#:~:text=Windows%2010%20is%20a%20Microsoft,follow%2Dup%20to%20Windows%208. [Accessed 2 April 2022].

Chandler, H. M., 2010. *Game Production Handbook.* New York: Jones and Bartetts Publishers.

Grey Springs, 2021. *Google Play.* [Online] Available at: https://play.google.com/store/apps/details?id=com.greysprings.shapesandcolors&hl=en&gl=US [Accessed 10 December 2021].

Hippo Kids Games, 2021. *google playstore.* [Online] Available at: https://play.google.com/store/apps/details?id=com.kbpro.UnderwaterFairyTale&hl=en&gl=US [Accessed 12 Dec 2021].

IDZ Digital Private Limited, 2022. *PlayStore.* [Online] Available at:

https://play.google.com/store/apps/details?id=com.baby.toddler.games&hl=en&gl=US

[Accessed 12 March 2022].

Jacquemet, A. N. a. J., 2019. The Impact of Video Games. *The Impact of Video Games*, Nov.

kidsEducational, B. F. p. l. g. f., 2022. *PlayStore.* [Online] Available at: https://play.google.com/store/apps/details?id=com.educational.baby.games&hl=en&gl=US

[Accessed 12 March 2022].

Market Report Analysis, 2020. *Market Report Analysis,* San Franscisco: Region, And Sement Forecasts.

Pressman, R. S., 2009. *Software Engineering Challeneges in Game Development.* 5th ed. New York: Book Style.

T.Fullerton, 2008. *Game Design Workshop.* 3rd ed. Burlington: Elsevier: Book Style.

Unity, 2022. *Learn Unity.* [Online] Available at: https://learn.unity.com/tutorial/working-with-scripts#

[Accessed 03 July 2022].

Unity, 2022. *Unity3D.* [Online] Available at: https://docs.unity3d.com/Packages/com.unity.package-manager-ui@1.8/manual/index.html#:~:text=A%20package%20is%20a%20container,range%20of%20enhancements%20to%20Unity.

[Accessed 2 April 2022].

Unity, 2022. *Unity3D.* [Online] Available at: https://unity3d.com/quick-guide-to-unity-asset-store#:~:text=the%20website%20version.-,What%20is%20a%20Unity%20Asset%3F,of%20file%20that%20Unity%20supports. [Accessed 3 April 2022].

Unity, 2022. *Unity3D.* [Online] Available at: https://docs.unity3d.com/Manual/class-AnimatorController.html#:~:text=Unity%20automatically%20creates%20an%20Animator,and%20click%20Create%20%3E%20Animator%20Controller. [Accessed 3 April 2022].

Wiley, A., 2020. *advancementcourses.* [Online] Available at: https://blog.advancementcourses.com/articles/educational-benefits-video-games/ [Accessed 02 November 2021].

Chapter 8 Appendix

8.1. Appendix A: Revised System Requirement Specification (SRS)

8.1.1. Overview

This document is intended to explain all the details about the overall system description, as well as the functional, non-functional, and system description. This article is divided into three sections: the first section has a quick introduction to the system; the second section contains a detailed description of the system; and the third section contains a list of references. The second section covers an overview of the system, as well as product perspectives, functionalities, user characteristics, constraints, assumptions, and dependencies, as well as requirement apportionment. Specific needs, data and behavioral model description of the system, functional and non-functional criteria, and so on are all included in the final section.

8.1.2. Overall Description

8.1.2.1. Productive Perspective

This product is an education game which is suitable for kids that are in grade 6 and above. It is an education game related to physics.

8.1.2.2. System Interfaces

The system interface is a two-dimensional instructional game. This educational game was created with Unity. All the physical materials in the game objects are inserted using the Unity engine. Unity is used for over 60% of the development process. For game and character design, I used Adobe Illustrator. To write code, Visual Studio is used.

8.1.2.3. User Interfaces

Users are most likely to play game through hardware (from windows or android).

8.1.2.4. Hardware Interfaces

Hardware that can be used are windows and android.

8.1.2.5. Software Interfaces

Unity

Name: Unity

Specification: For game development

Version: Unity 2019.4.29f1 (64-bit)

Source: https://unity3d.com/get-unity/download/archive

Visual Studio

Name: Visual Studio

Specification: For coding

Version: Visual Studio 2019

Source: https://visualstudio.microsoft.com/downloads/

Adobe Illustrator

Name Adobe Illustrator

Specification: For Designing game object and game characters

Version: Adobe Illustrator 2021

Source: https://igetintopc.com/adobe-illustrator-2021-free-download/

8.2. Appendix B: Pre-Survey Form

Link: https://forms.gle/LG2r77UMQKs5CuQm6

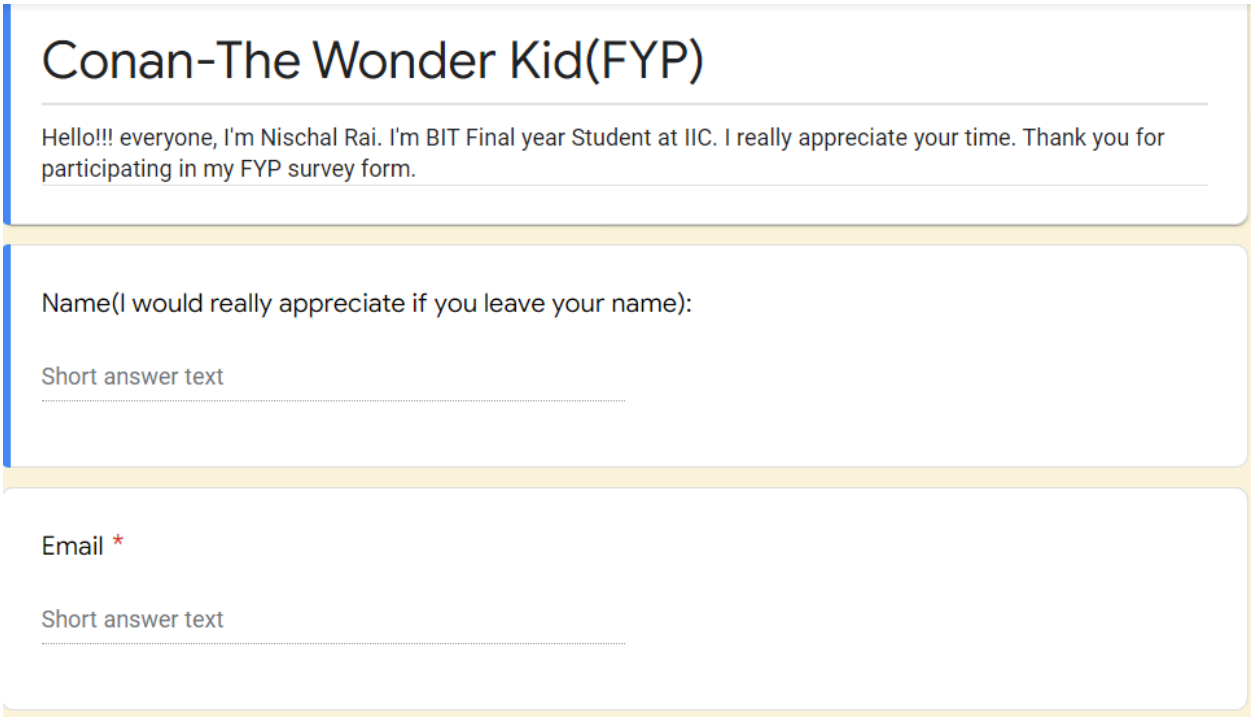


Figure 166: Pre-Survey Form Part 1

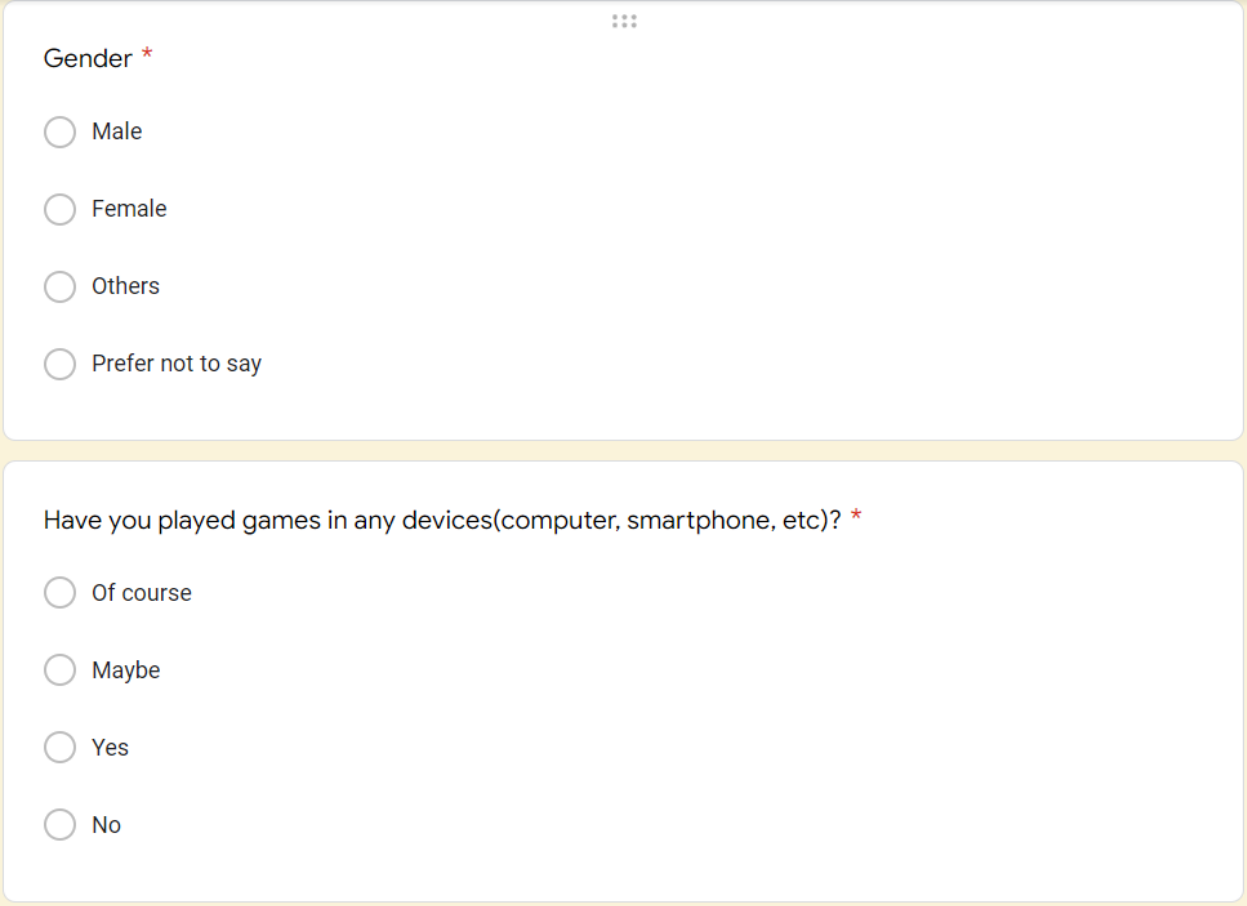


Figure 167: Pre-Survey Form Part 2

How often do you play mobile/pc games? *

- ◯ Daily
- ◯ At free times
- ◯ Sometimes
- ◯ Never

In which platform you play games most of the time? *

1. Android
2. Windows
3. Mac Book
4. Ios
5. Others
6. I don't play games

Figure 168: Pre-Survey Form Part 3

Do you think we can learn many things by playing mobile/pc games? *

- ○ Of Course
- ○ Yes
- ○ No
- ○ Maybe

What do you think if I tell you that we can actually learn educational course related things just by playing games? *

- ○ Amazing
- ○ Great
- ○ That would be great
- ○ Boring
- ○ Not interested

Figure 169: Pre-Survey Form Part 4

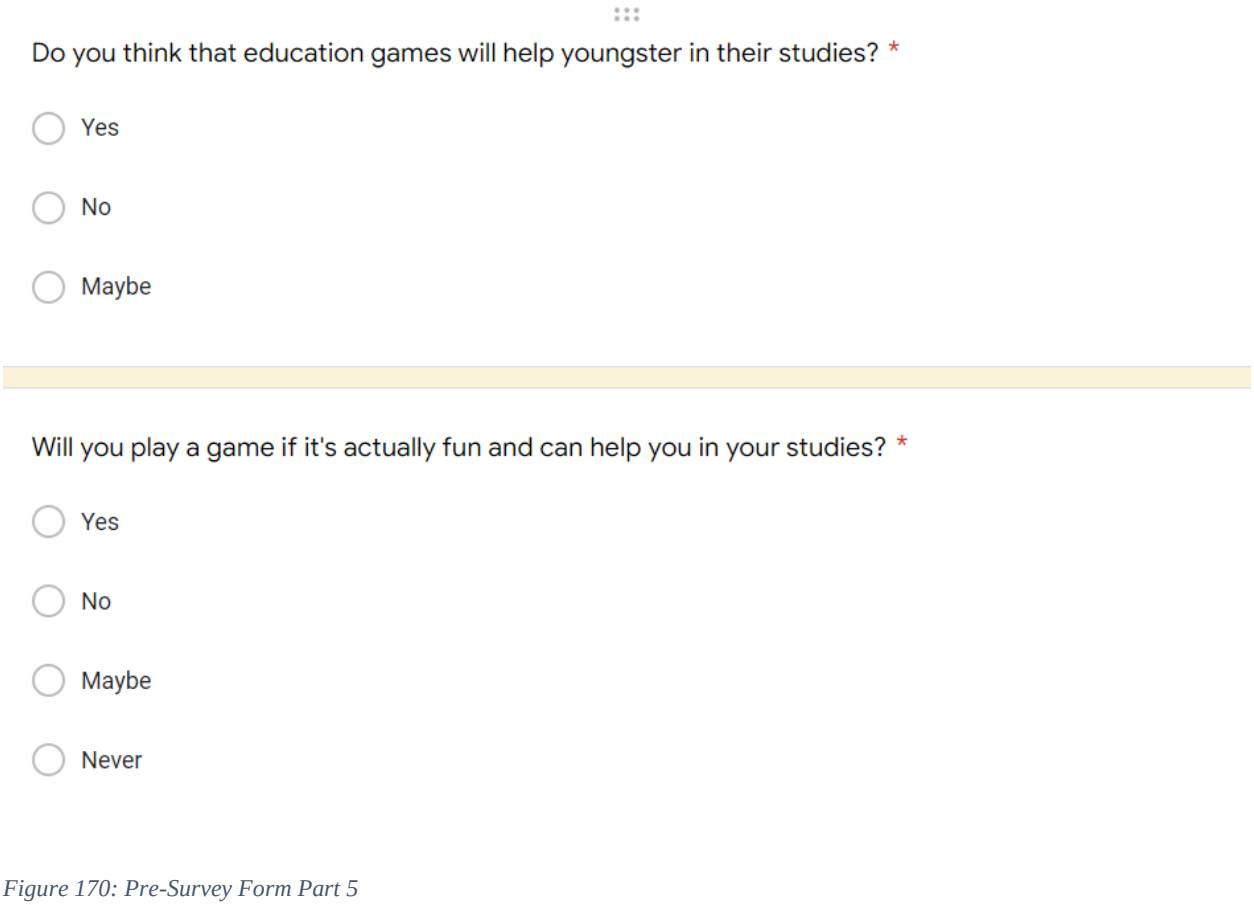


Figure 170: Pre-Survey Form Part 5

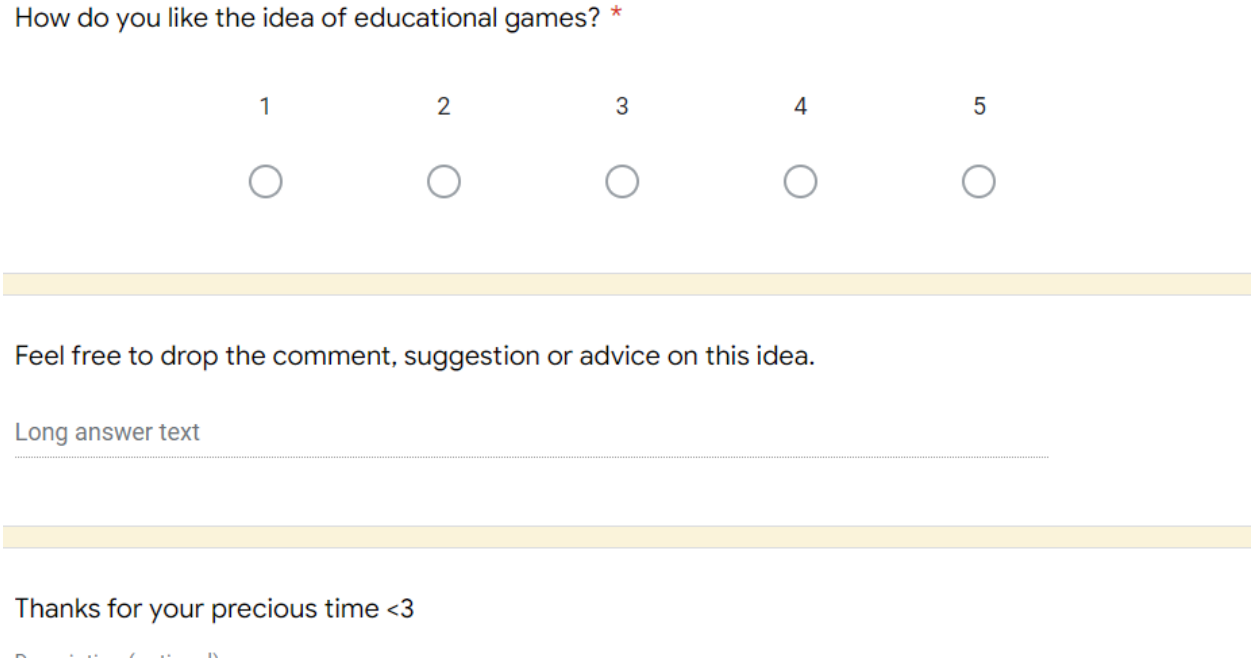


Figure 171: Pre-Survey Form Part 6

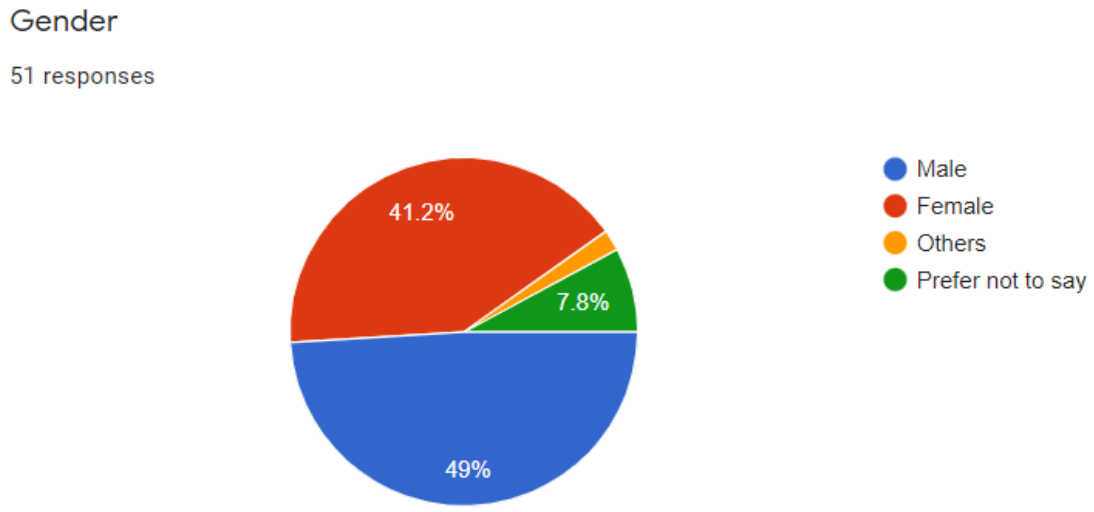


Figure 172: Pre-Survey Question 1

This survey shows the gender of people who took part in the survey.

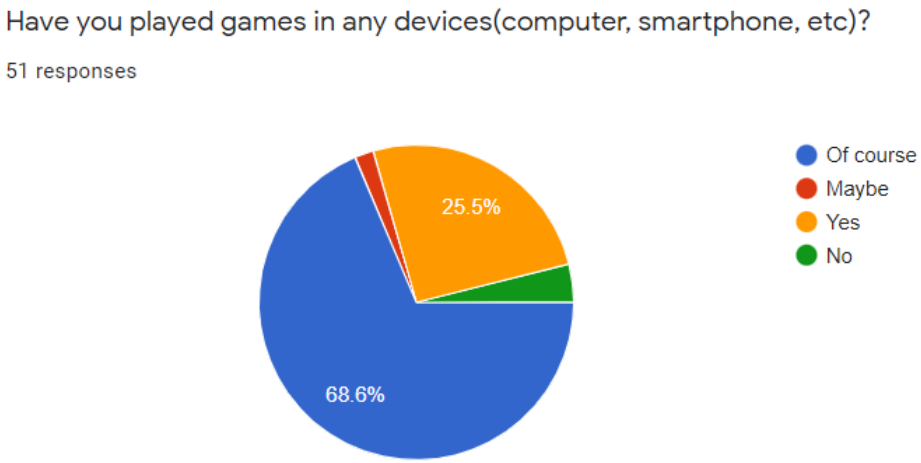


Figure 173:Pre-Survey Question 2

This survey shows if people who took part in this survey plays games.

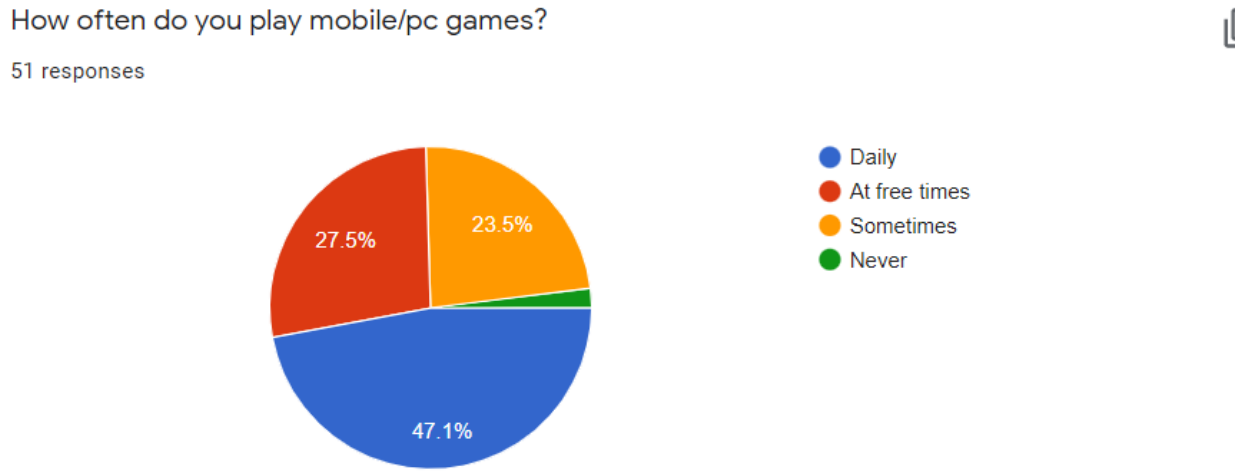


Figure 174: Pre-Survey Question 3

This survey shows how often the people who took part in this survey plays the game.

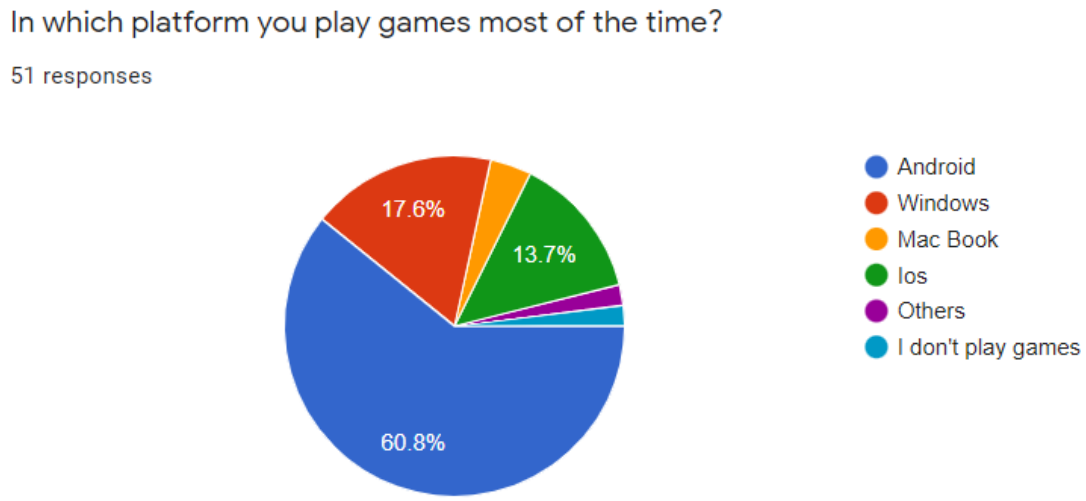


Figure 175: Pre-Survey Question 4

This survey shows the platform in which people who took part in this survey plays game in.

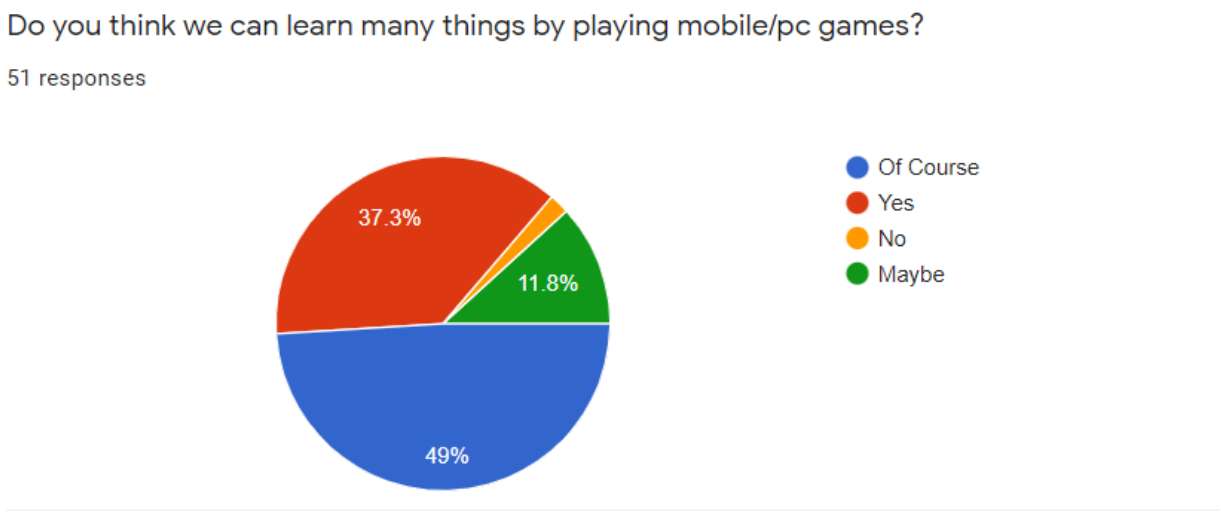


Figure 176: Pre-Survey Question 5

People giving their opinion about mobile/pc games

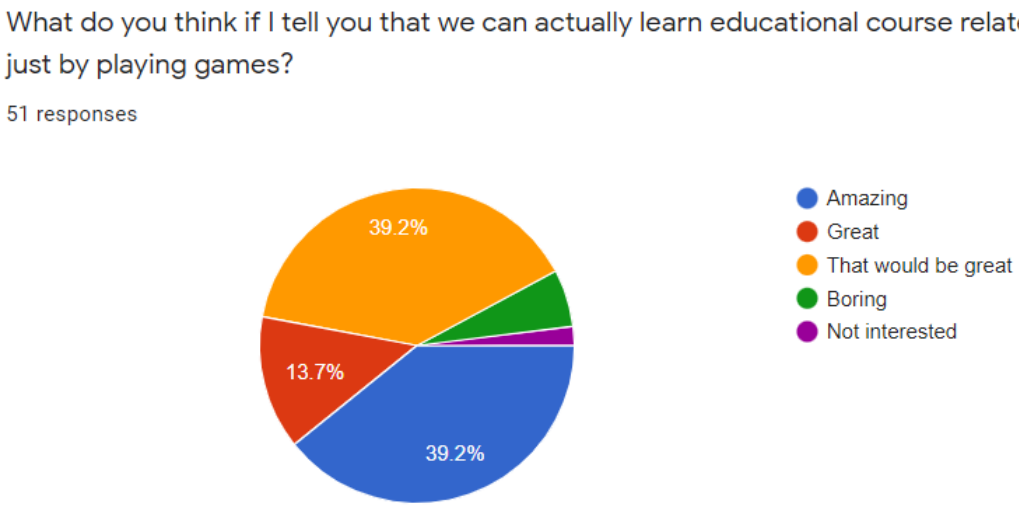


Figure 177: : Pre-Survey Question 6

People responses to educational game.

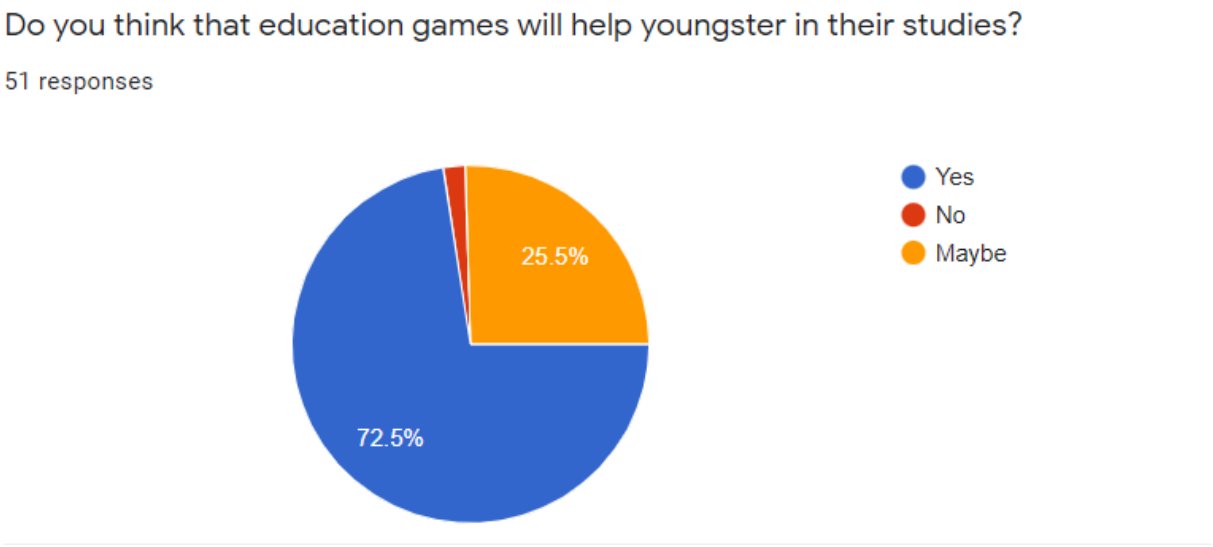


Figure 178: : Pre-Survey Question 7

People giving their opinion if education game will help youngster in their studies.

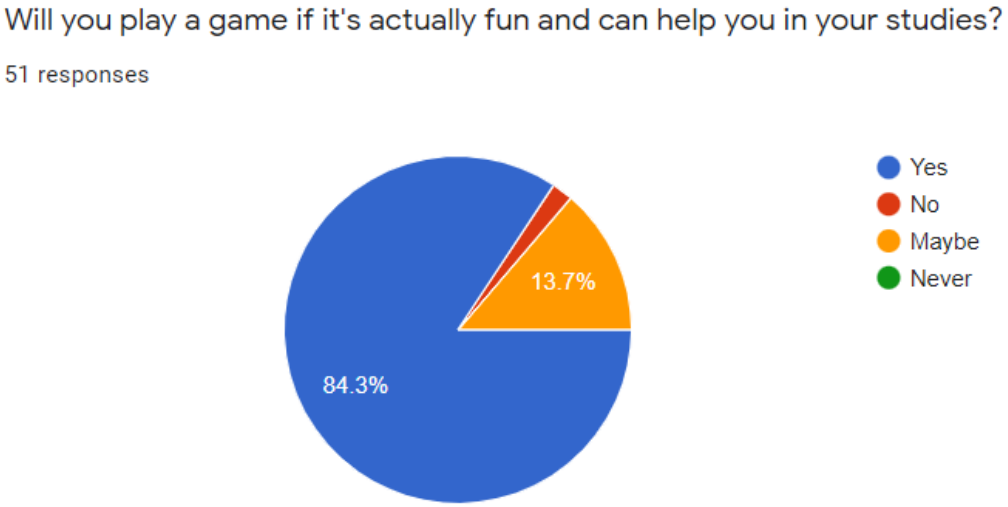


Figure 179:Pre-Survey Question 8

Result of asking people if they want to play game if it will help then in their studies.

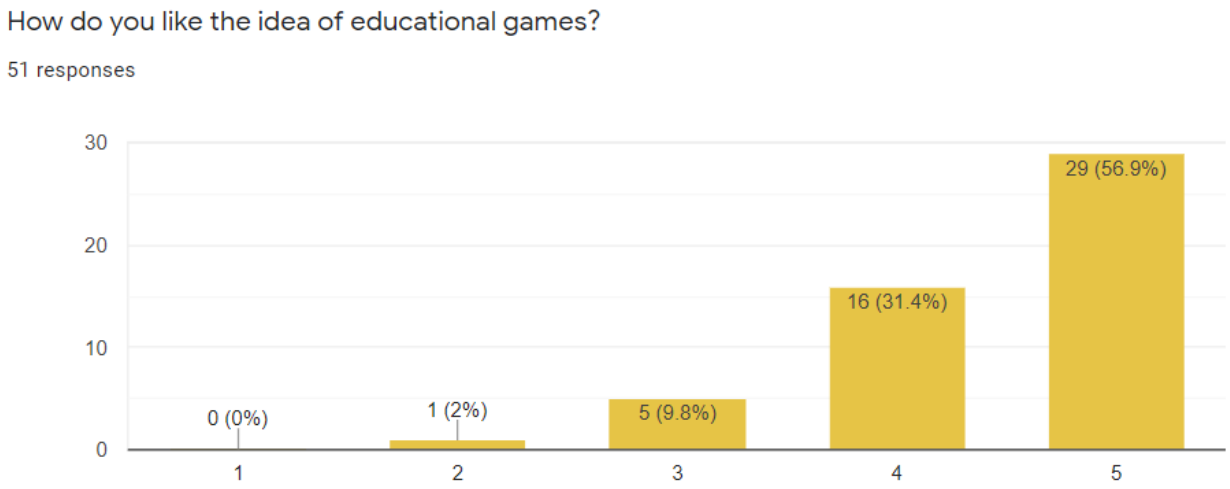


Figure 180: Pre-Survey Question 9

8.3. Appendix C: Pre-Survey Form

Link: https://forms.gle/8xzgVRzzWJyMQkhz8

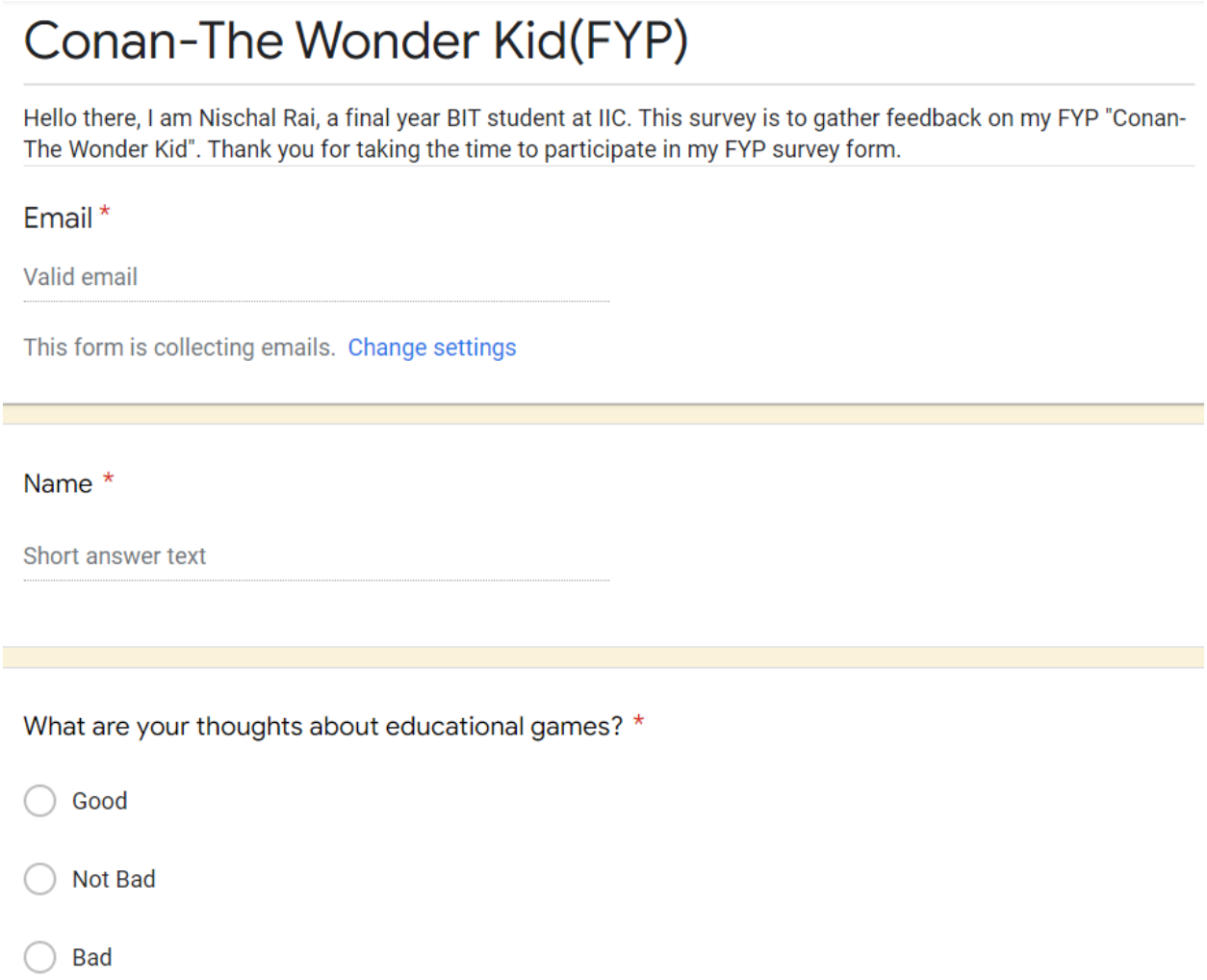


Figure 181: Post-Survey Part 1

Which type of games do you prefer playing? *

- ◯ Online
- ◯ Offline
- ◯ Both

On which device do you play games usually? *

1. Mobile
2. PC
3. Both

Figure 182: Post-Survey Part 2

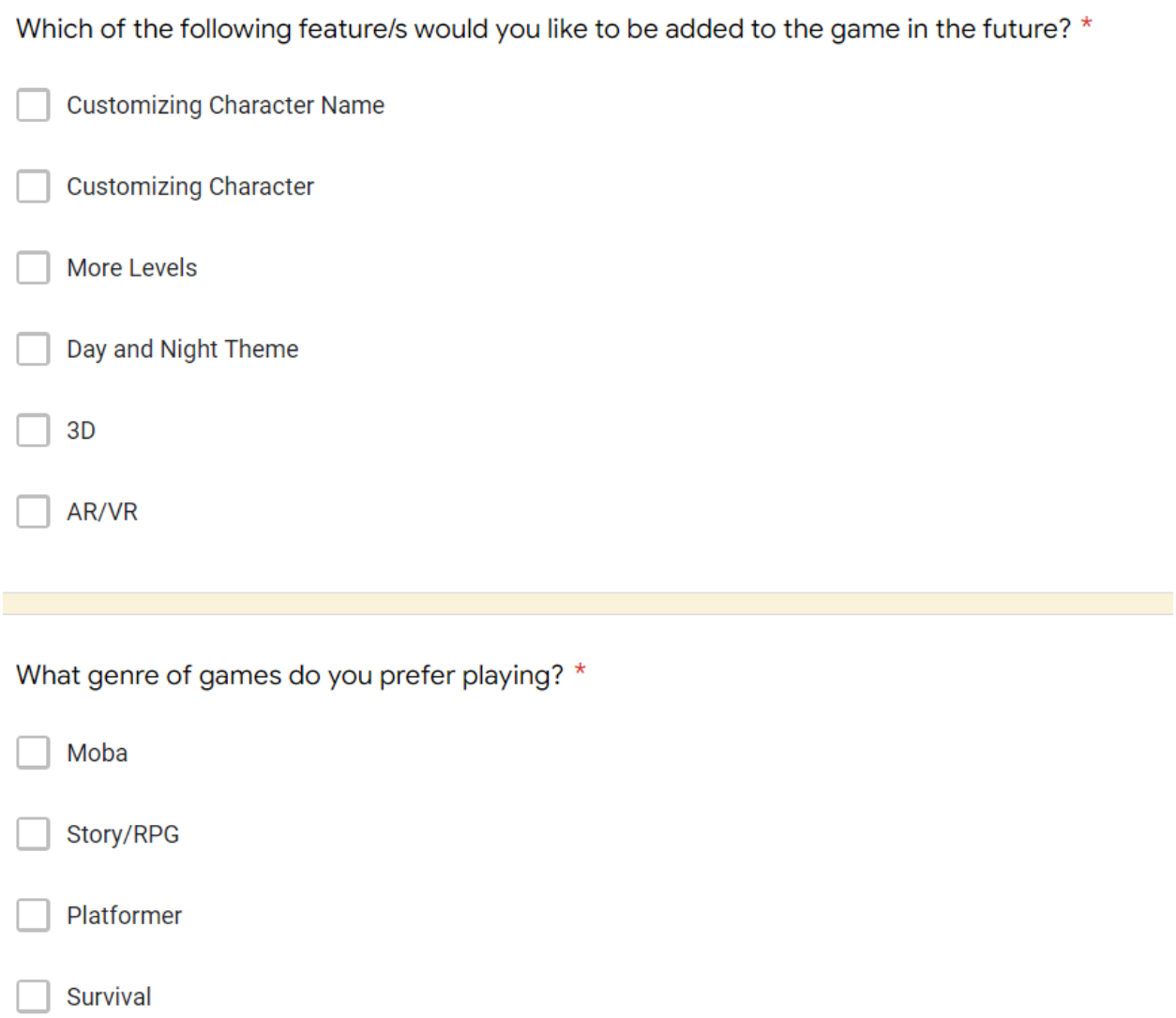


Figure 183: Post-Survey Part 3

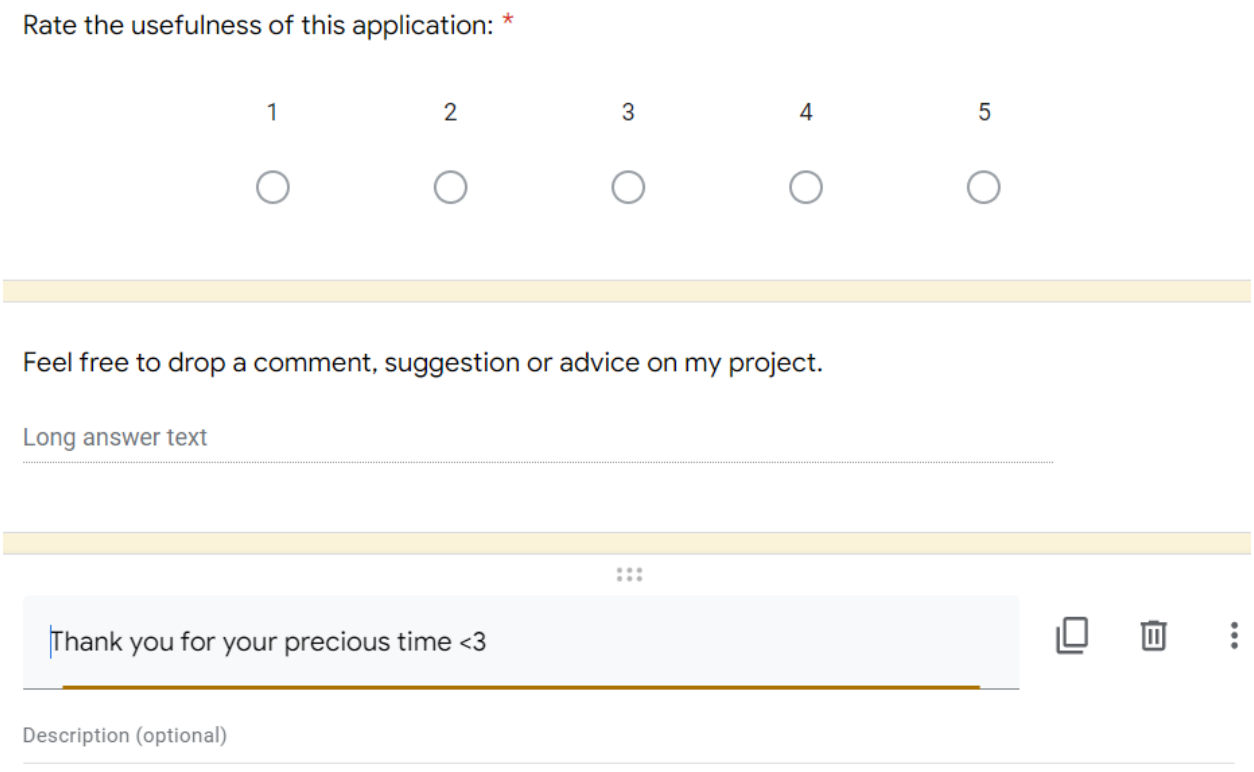


Figure 184:: Post-Survey Part 4

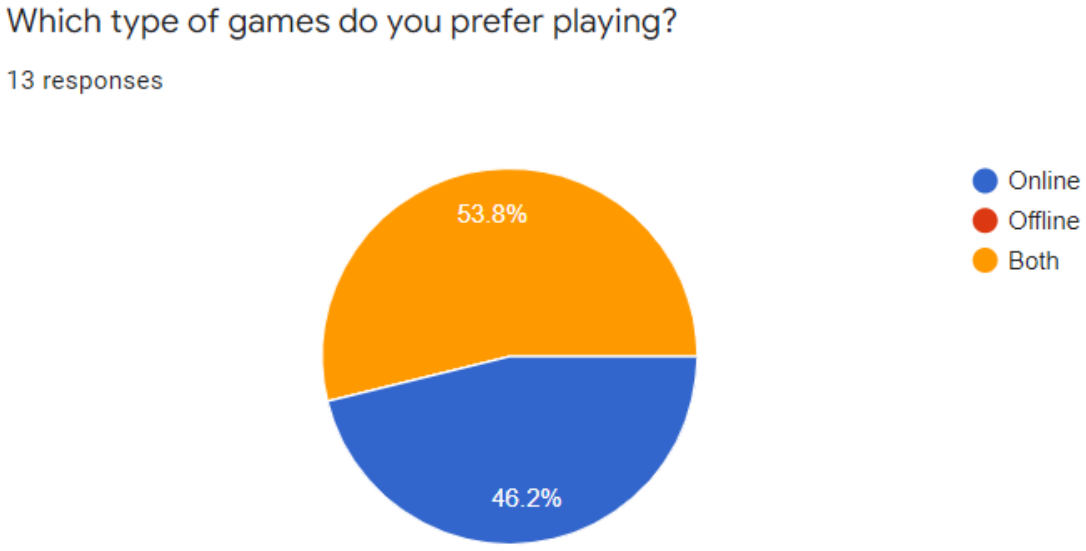


Figure 185: Post-Survey Question 1

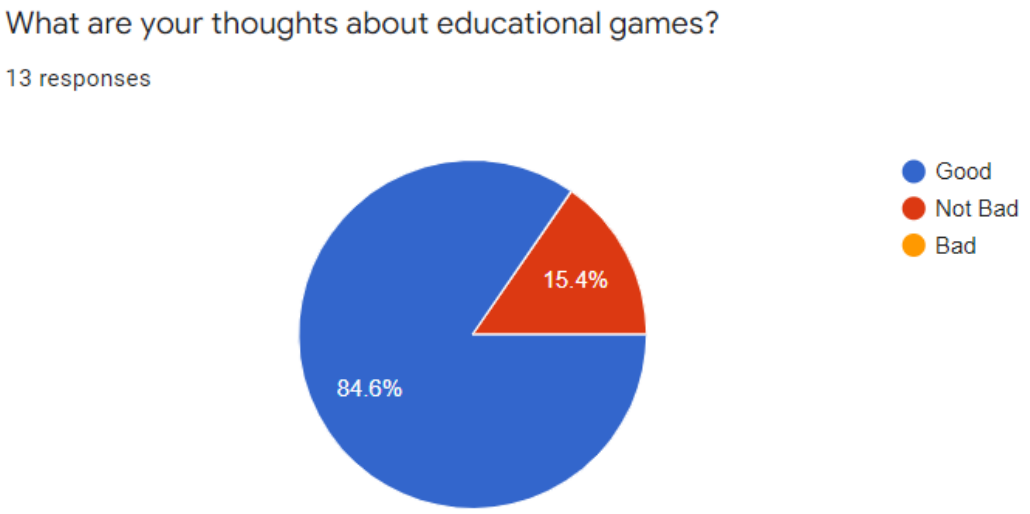


Figure 186: Post-Survey Question 2

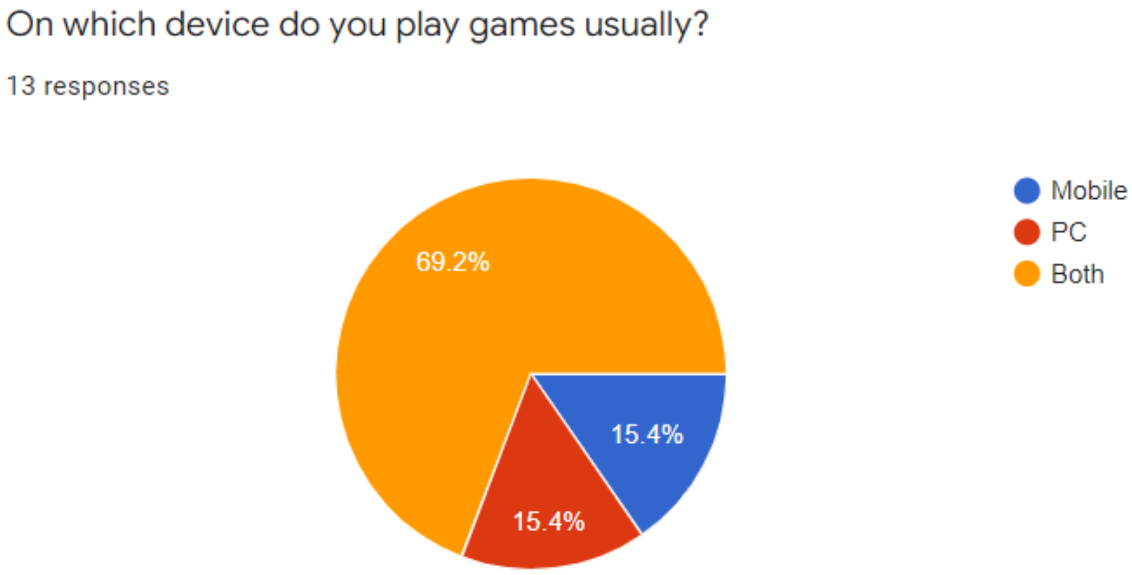


Figure 187: Post-Survey Question 3

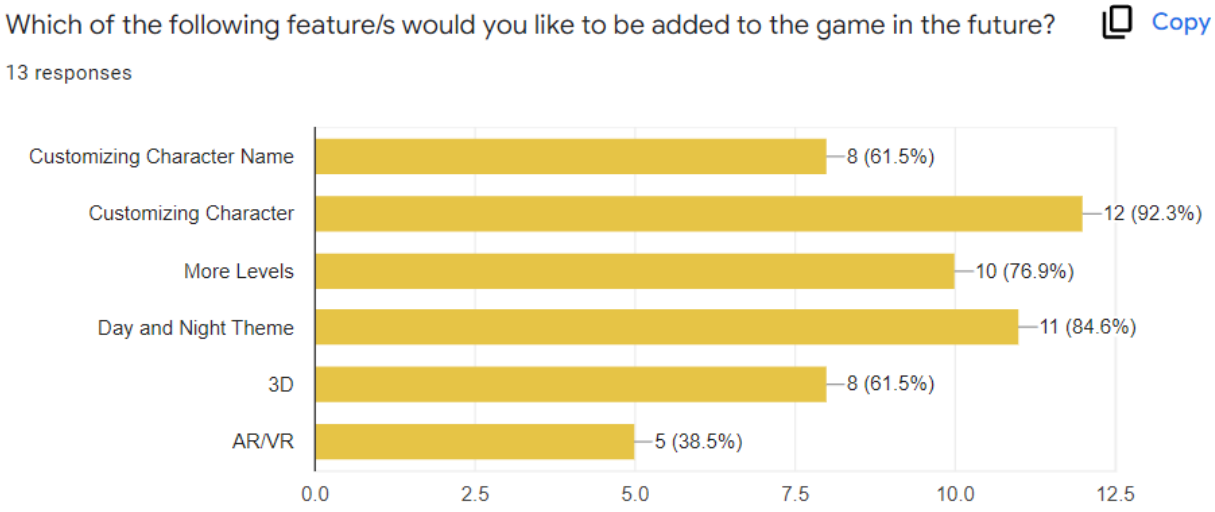


Figure 188: Post-Survey Question 4

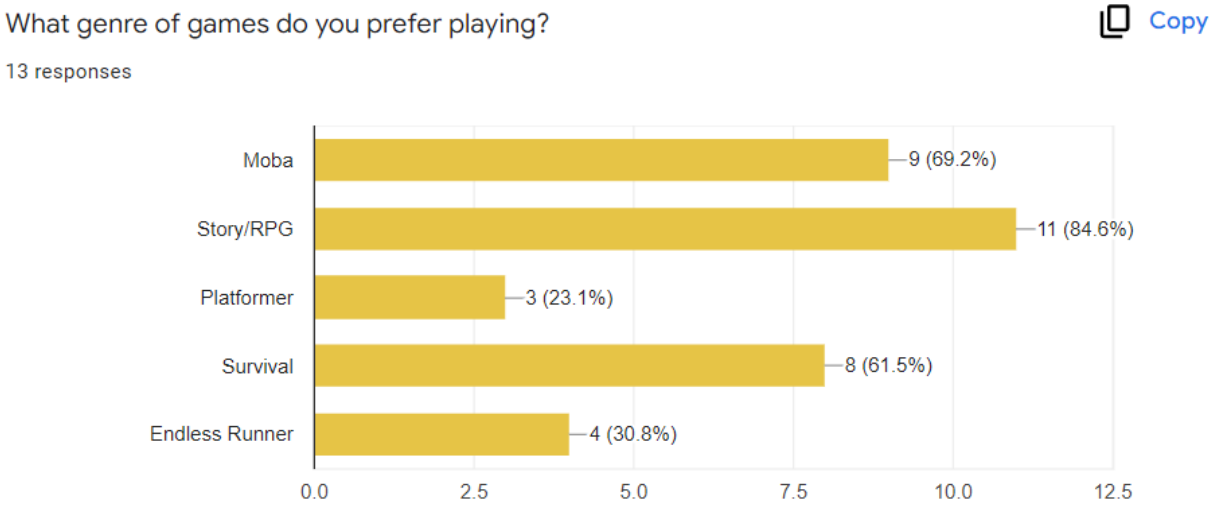


Figure 189: Post-Survey Question 5

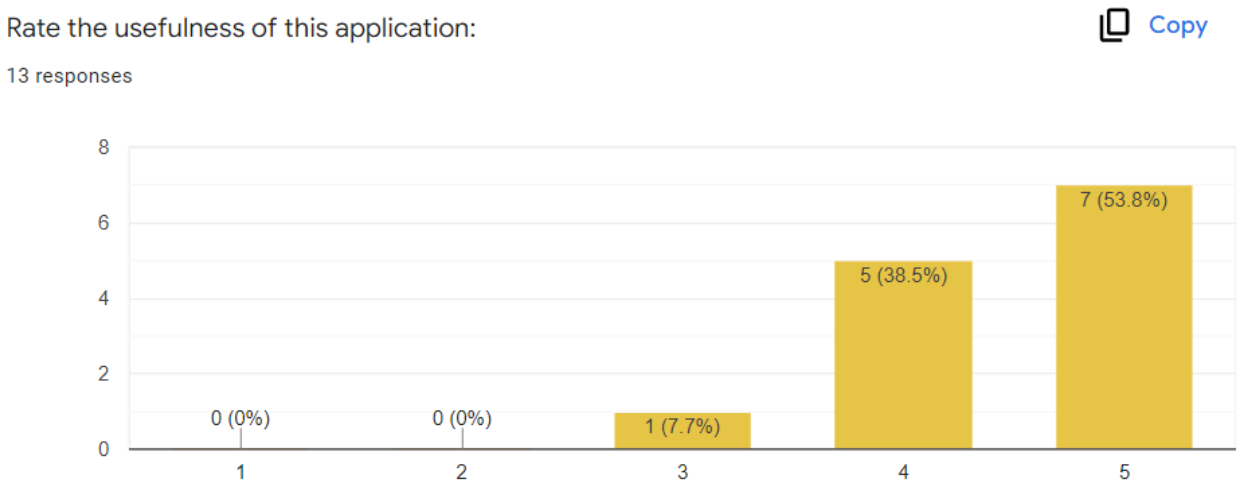


Figure 190: Post-Survey Question 6

8.4. Appendix D: Sample Codes

8.4.1. Code of Settings script

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.UI;
5.
6.  public class Settings : MonoBehaviour
7.  {
8.      //Taking UI
9.      [SerializeField] GameObject UI;
10.
11.     //For Sliders
12.     [SerializeField] Slider brightnessSlider;
13.     [SerializeField] Slider contrastSlider;
14.     [SerializeField] Slider saturationSlider;
15.
16.     //Get UI
17.     public GameObject GetUI()
18.     {
19.         return UI;
20.     }
21.
22.     //BrightnesSlider
23.     public Slider GetBrightness()
24.     {
25.
26.         return brightnessSlider;
27.     }
28.
29.     //ContrastSlider
30.     public Slider GetContrast()
31.     {
32.         return contrastSlider;
33.     }
34.
35.     //SaturationSlider
36.     public Slider GetSaturation()
37.     {
38.         return saturationSlider;
39.     }
40. }
41.
```

8.4.2. Code of ColorSettings script

```
1.  using UnityEngine;
2.  using UnityEngine.Rendering;
3.  using UnityEngine.Rendering.Universal;
4.  using UnityEngine.UI;
5.
6.  public class ColorSettings : MonoBehaviour
7.  {
8.      //Sliders
9.      Slider brightnessSlider;
10.     Slider contrastSlider;
11.     Slider saturationSlider;
12.
13.     //Volume
14.     private Volume v;
15.
16.     //For color adjustment
17.     private ColorAdjustments setting;
18.
19.     //Color Setting
20.     Settings colourSettings;
21.     private void Awake()
22.     {
23.         //Taking object type Settings
24.         colourSettings = FindObjectOfType<Settings>();
25.
26.         //Finding Sliders
27.         brightnessSlider = colourSettings.GetBrightness();
28.         contrastSlider = colourSettings.GetContrast();
29.         saturationSlider = colourSettings.GetSaturation();
30.     }
31.     private void Start()
32.     {
33.         v = GetComponent<Volume>();
34.         v.profile.TryGet(out setting);
35.
36.         //Default Value for first time
37.         brightnessSlider.maxValue = 1f;
38.         brightnessSlider.minValue = -2f;
39.
40.         //Default Value for first time
41.         contrastSlider.maxValue = 100f;
42.         contrastSlider.minValue = -50f;
43.
44.         //Default Value for first time
45.         saturationSlider.maxValue = 100f;
46.         saturationSlider.minValue = -100f;
47.
48.
49.         //PlayerPrefs for value
50.         if (PlayerPrefs.HasKey("Brightness"))
51.         {
52.             setting.postExposure.value = brightnessSlider.value =
    PlayerPrefs.GetFloat("Brightness");
53.             setting.contrast.value = contrastSlider.value =
    PlayerPrefs.GetFloat("Contrast");
54.             setting.saturation.value = saturationSlider.value =
    PlayerPrefs.GetFloat("Saturation");
55.         }
56.         else
```

```
57.         {
58.             setting.postExposure.value = brightnessSlider.value = 0f;
59.             setting.contrast.value = contrastSlider.value = 0f;
60.             setting.saturation.value = saturationSlider.value = 0f;
61.         }
62.     }
63.
64.     //For color reset
65.     public void resetColour()
66.     {
67.         setting.postExposure.value = brightnessSlider.value = 0.01194698f;
68.         setting.contrast.value = contrastSlider.value = 0.83559f;
69.         setting.saturation.value = saturationSlider.value = 35.4568f;
70.     }
71.
72.     public void Update()
73.     {
74.         setting.postExposure.value = brightnessSlider.value;
75.         setting.contrast.value = contrastSlider.value;
76.         setting.saturation.value = saturationSlider.value;
77.     }
78.
79.     //Saving value on destroy (Cache memory)
80.     private void OnDestroy()
81.     {
82.         PlayerPrefs.SetFloat("Brightness", setting.postExposure.value);
83.         PlayerPrefs.SetFloat("Saturation", setting.saturation.value);
84.         PlayerPrefs.SetFloat("Contrast", setting.contrast.value);
85.     }
86.
87.     //Saving value while going back to setting (Cache memory)
88.     private void Save()
89.     {
90.         PlayerPrefs.SetFloat("Brightness", setting.postExposure.value);
91.         PlayerPrefs.SetFloat("Saturation", setting.saturation.value);
92.         PlayerPrefs.SetFloat("Contrast", setting.contrast.value);
93.     }
94. }
95.
```

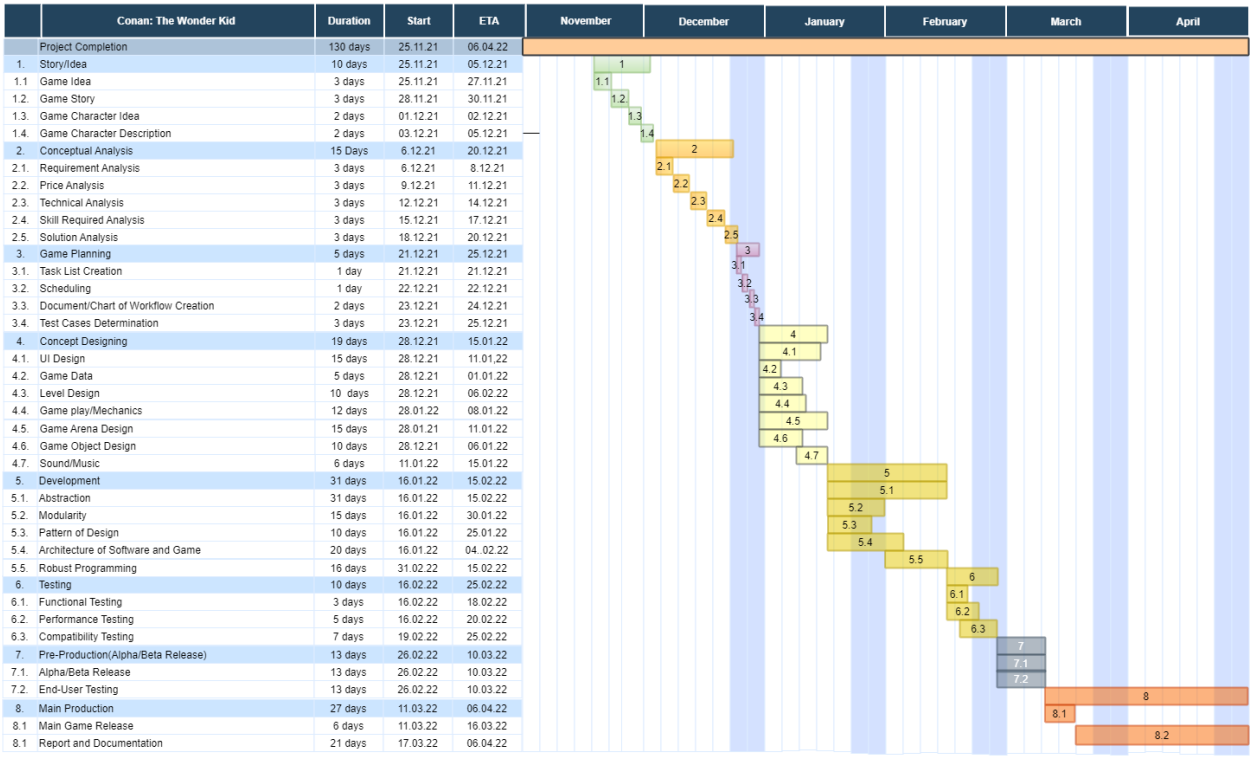
8.4.2. Code of AudioManager script

```
1.  using UnityEngine;
2.  using UnityEngine.UI;
3.
4.  public class AudioManager : MonoBehaviour
5.  {
6.      [SerializeField] AudioSource gameMusic;
7.      [SerializeField] Slider musicSlider;
8.
9.      private void Start()
10.     {
11.
12.         if (PlayerPrefs.GetInt("FIRSTTIMEOPENING", 1) == 1)
13.         {
14.             //Debug.Log("First Time Opening");
15.             //Set first time opening to false
16.             PlayerPrefs.SetInt("FIRSTTIMEOPENING", 0);
17.             gameMusic.volume = musicSlider.value = 0.5f;
18.
19.         }
20.         else
21.         {
22.             gameMusic.volume = musicSlider.value = PlayerPrefs.GetFloat("Music");
23.
24.         }
25.     }
26.
27.     private void Update()
28.     {
29.         SetVolume();
30.     }
31.
32.     private void SetVolume()
33.     {
34.         gameMusic.volume = musicSlider.value;
35.     }
36.
37.
38.     public void OnDestroy()
39.     {
40.         PlayerPrefs.SetFloat("Music", musicSlider.value);
41.     }
42.
43.
44. }
45.
```

8.5. Appendix E: Designs

8.5.1. Gantt Chart

A Gantt chart is a project management tool used to plan and schedule projects of all sizes. Project management tasks are converted into a horizontal bar chart in the Gantt chart, with start and end dates, dependencies, scheduling, and deadlines. This project's updated Gantt chart is shown below:



	Conan: The Wonder Kid	Duration	Start	ETA
	Project Completion	130 days	25.11.21	06.04.22
1.	Story/Idea	10 days	25.11.21	05.12.21
1.1	Game Idea	3 days	25.11.21	27.11.21
1.2.	Game Story	3 days	28.11.21	30.11.21
1.3.	Game Character Idea	2 days	01.12.21	02.12.21
1.4.	Game Character Description	2 days	03.12.21	05.12.21
2.	Conceptual Analysis	15 Days	6.12.21	20.12.21
2.1.	Requirement Analysis	3 days	6.12.21	8.12.21
2.2.	Price Analysis	3 days	9.12.21	11.12.21
2.3.	Technical Analysis	3 days	12.12.21	14.12.21
2.4.	Skill Required Analysis	3 days	15.12.21	17.12.21
2.5.	Solution Analysis	3 days	18.12.21	20.12.21
3.	Game Planning	5 days	21.12.21	25.12.21
3.1.	Task List Creation	1 day	21.12.21	21.12.21
3.2.	Scheduling	1 day	22.12.21	22.12.21
3.3.	Document/Chart of Workflow Creation	2 days	23.12.21	24.12.21
3.4.	Test Cases Determination	3 days	23.12.21	25.12.21
4.	Concept Designing	19 days	28.12.21	15.01.22
4.1.	UI Design	15 days	28.12.21	11.01,22
4.2.	Game Data	5 days	28.12.21	01.01.22
4.3.	Level Design	10 days	28.12.21	06.02.22
4.4.	Game play/Mechanics	12 days	28.01.22	08.01.22
4.5.	Game Arena Design	15 days	28.01.21	11.01.22
4.6.	Game Object Design	10 days	28.12.21	06.01.22
4.7.	Sound/Music	6 days	11.01.22	15.01.22
5.	Development	31 days	16.01.22	15.02.22
5.1.	Abstraction	31 days	16.01.22	15.02.22
5.2.	Modularity	15 days	16.01.22	30.01.22
5.3.	Pattern of Design	10 days	16.01.22	25.01.22
5.4.	Architecture of Software and Game	20 days	16.01.22	04..02.22
5.5.	Robust Programming	16 days	31.02.22	15.02.22
6.	Testing	10 days	16.02.22	25.02.22
6.1.	Functional Testing	3 days	16.02.22	18.02.22
6.2.	Performance Testing	5 days	16.02.22	20.02.22
6.3.	Compatibility Testing	7 days	19.02.22	25.02.22
7.	Pre-Production(Alpha/Beta Release)	13 days	26.02.22	10.03.22
7.1.	Alpha/Beta Release	13 days	26.02.22	10.03.22
7.2.	End-User Testing	13 days	26.02.22	10.03.22
8.	Main Production	27 days	11.03.22	06.04.22
8.1	Main Game Release	6 days	11.03.22	16.03.22
8.1	Report and Documentation	21 days	17.03.22	06.04.22

Figure 191: Initial Gantt Chart

This is the first Gannt Chart created while making interim report.

	Conan: The Wonder Kid	Duration	Start	ETA
	Project Completion	130 days	25.11.21	06.04.22
1.	Story/Idea	10 days	25.11.21	05.12.21
1.1	Game Idea	3 days	25.11.21	27.11.21
1.2.	Game Story	3 days	28.11.21	30.11.21
1.3.	Game Character Idea	2 days	01.12.21	02.12.21
1.4.	Game Character Description	2 days	03.12.21	05.12.21
2.	Conceptual Analysis	15 Days	6.12.21	20.12.21
2.1.	Requirement Analysis	3 days	6.12.21	8.12.21
2.2.	Price Analysis	3 days	9.12.21	11.12.21
2.3.	Technical Analysis	3 days	12.12.21	14.12.21
2.4.	Skill Required Analysis	3 days	15.12.21	17.12.21
2.5.	Solution Analysis	3 days	18.12.21	20.12.21
3.	Game Planning	5 days	21.12.21	25.12.21
3.1.	Task List Creation	1 day	21.12.21	21.12.21
3.2.	Scheduling	1 day	22.12.21	22.12.21
3.3.	Document/Chart of Workflow Creation	2 days	23.12.21	24.12.21
3.4.	Test Cases Determination	3 days	23.12.21	25.12.21
4.	Concept Designing	19 days	28.12.21	15.01.22
4.1.	UI Design	15 days	28.12.21	11.01,22
4.2.	Game Data	5 days	28.12.21	01.01.22
4.3.	Level Design	10 days	28.12.21	06.02.22
4.4.	Game play/Mechanics	12 days	28.01.22	08.01.22
4.5.	Game Arena Design	15 days	28.01.21	11.01.22
4.6.	Game Object Design	10 days	28.12.21	06.01.22
4.7.	Sound/Music	6 days	11.01.22	15.01.22
5.	Development	64 days	16.01.22	20.03.22
5.1.	Abstraction	64 days	16.01.22	15.02.22
5.2.	Modularity	15 days	16.01.22	30.01.22
5.3.	Pattern of Design	10 days	16.01.22	25.01.22
5.4.	Architecture of Software and Game	20 days	16.01.22	04..02.22
5.5.	Robust Programming	16 days	31.02.22	15.02.22
6.	Testing	10 days	16.02.22	25.02.22
6.1.	Functional Testing	3 days	16.02.22	18.02.22
6.2.	Performance Testing	5 days	16.02.22	20.02.22
6.3.	Compatibility Testing	7 days	19.02.22	25.02.22
7.	Pre-Production(Alpha/Beta Release)	13 days	26.02.22	10.03.22
7.1.	Alpha/Beta Release	13 days	26.02.22	10.03.22
7.2.	End-User Testing	13 days	26.02.22	10.03.22
8.	Main Production	29 days	11.03.22	08.04.22
8.1	Main Game Release	10 days	11.03.22	20.03.22
8.2	Report and Documentation	23 days	17.03.22	08.04.22

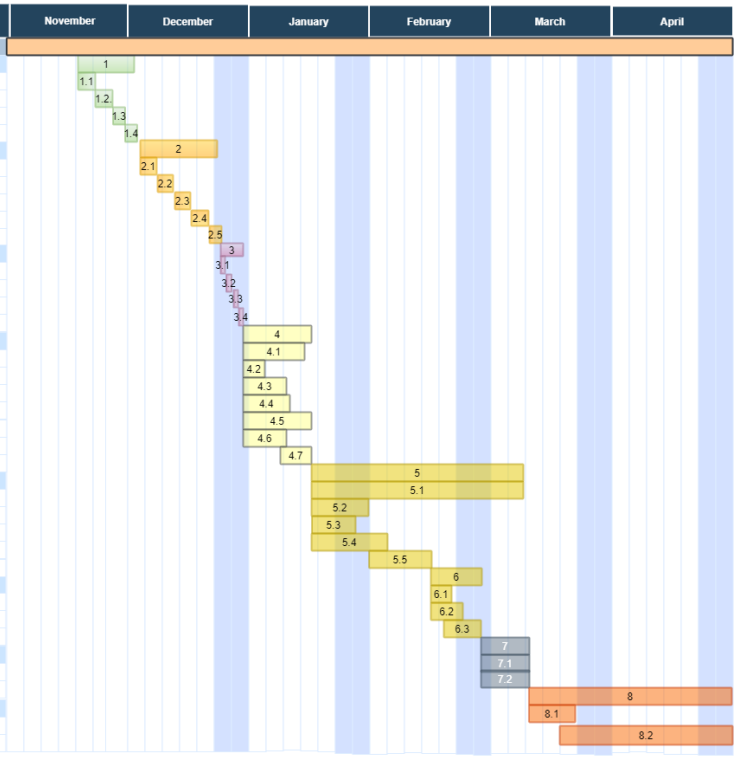


Figure 192: Final Gantt Chart

This the final Gannt Chart arranged according to the logbook and work done.

8.5.2. Work Breakdown Structure

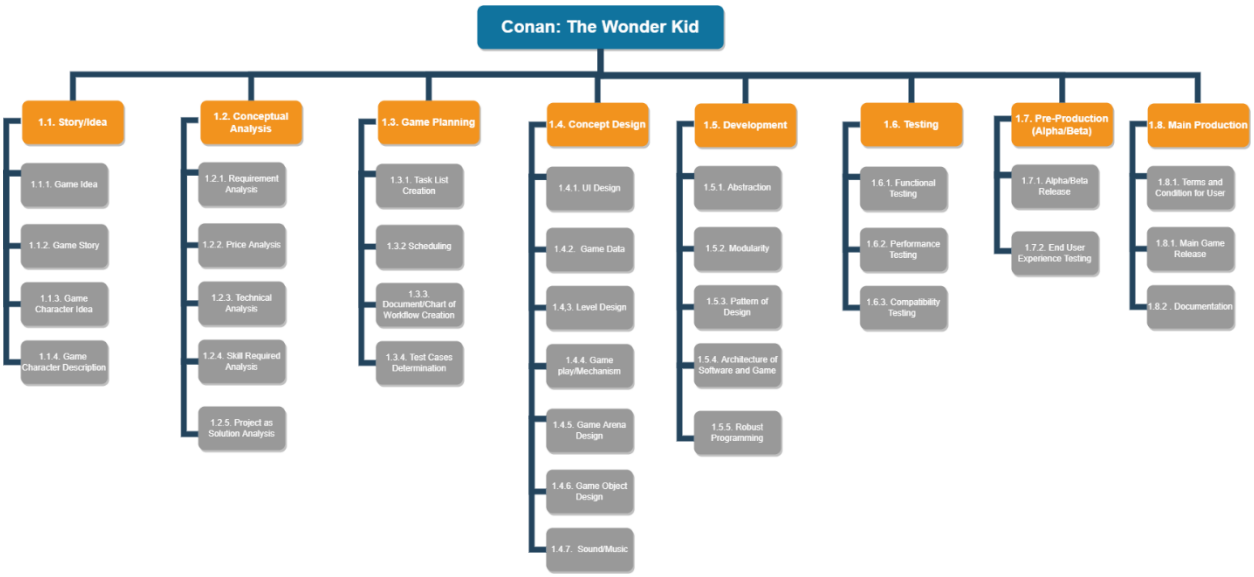


Figure 193: Work Breakdown Structure

8.5.3. Use Case Diagram

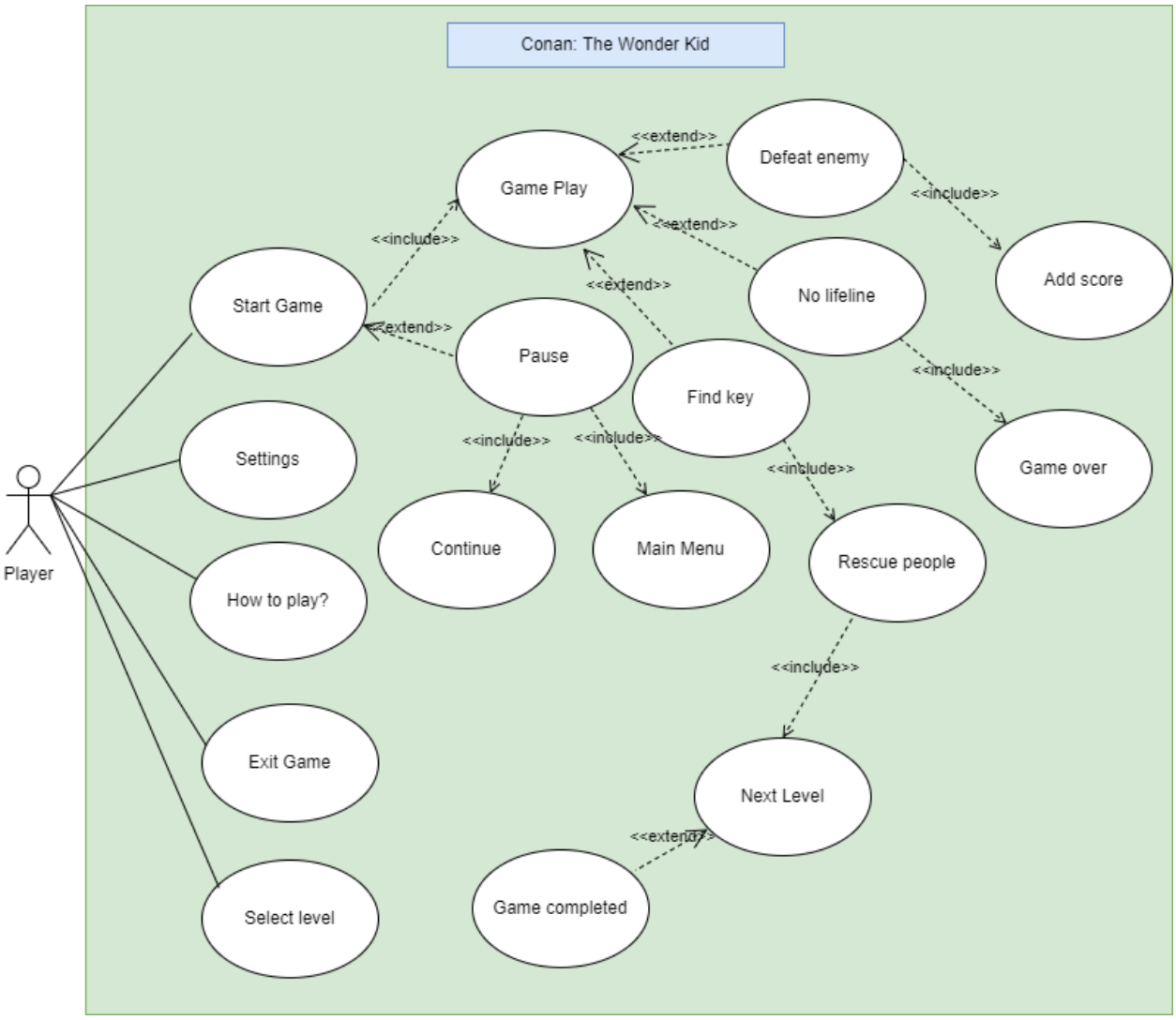


Figure 194: Use Case Diagram of player and the system

8.5.4. DFD Diagram

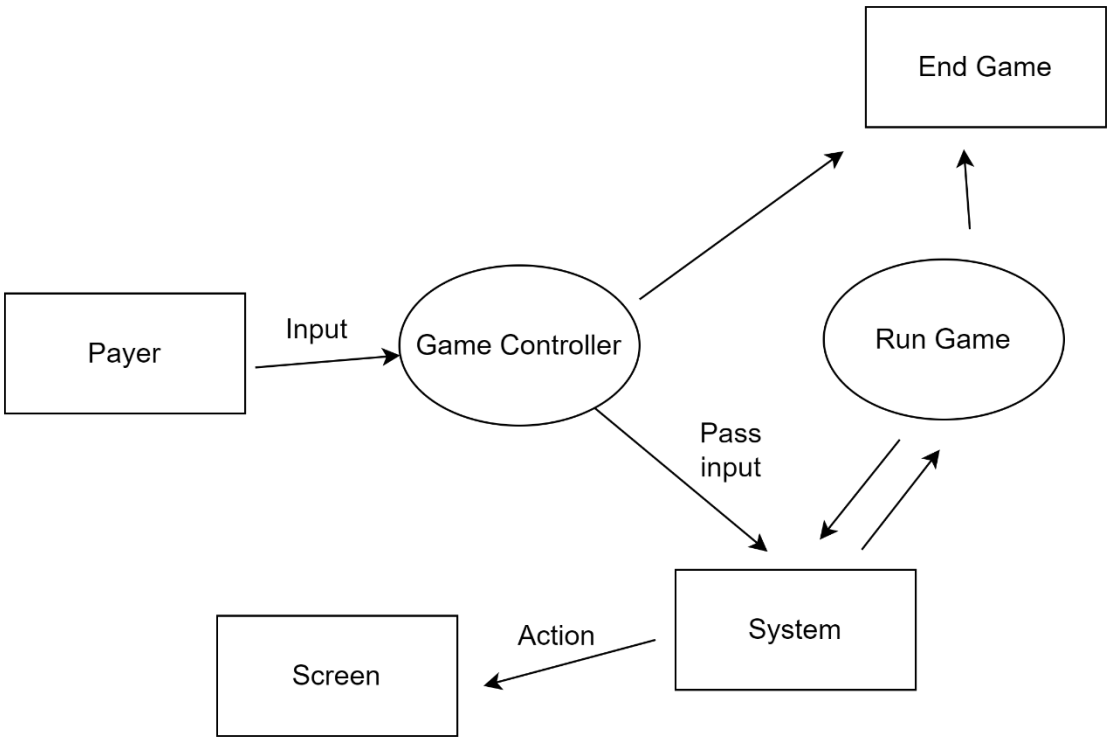


Figure 195: DFD Diagram

8.5.5. Hardware Architecture Diagram

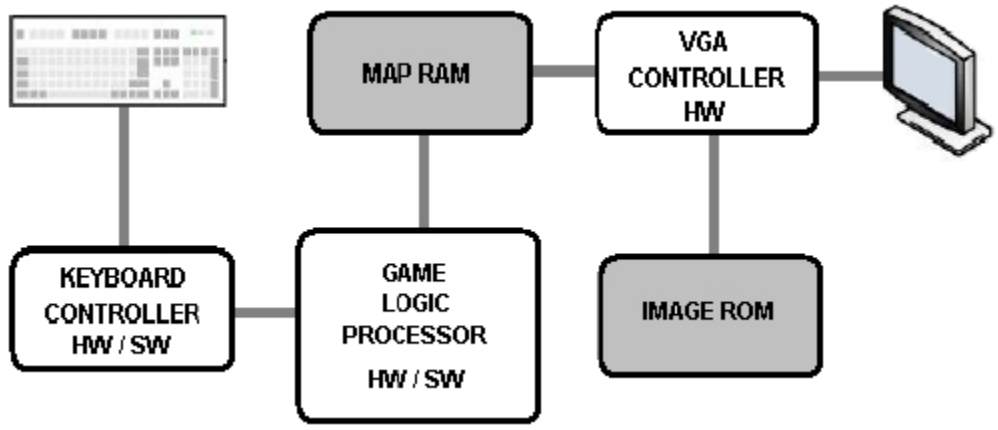


Figure 196: Hardware Architecture Diagram

8.6. Appendix F: Screenshot of the system

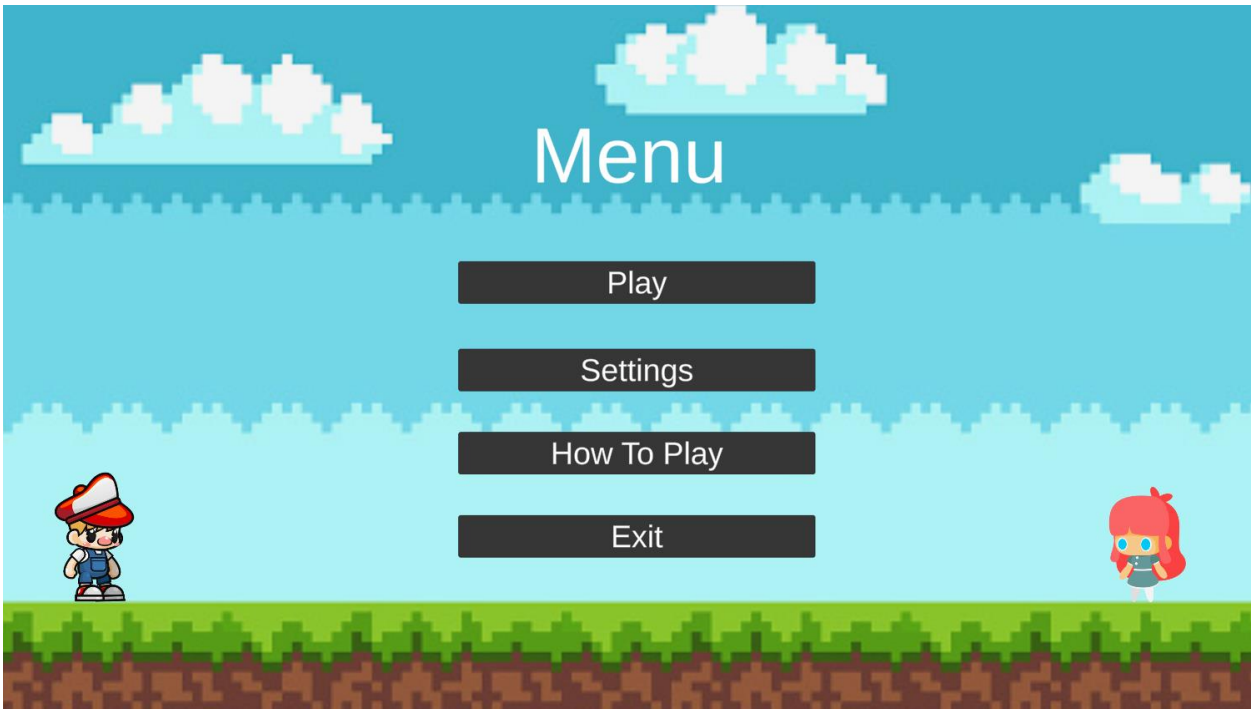


Figure 197: Main Menu

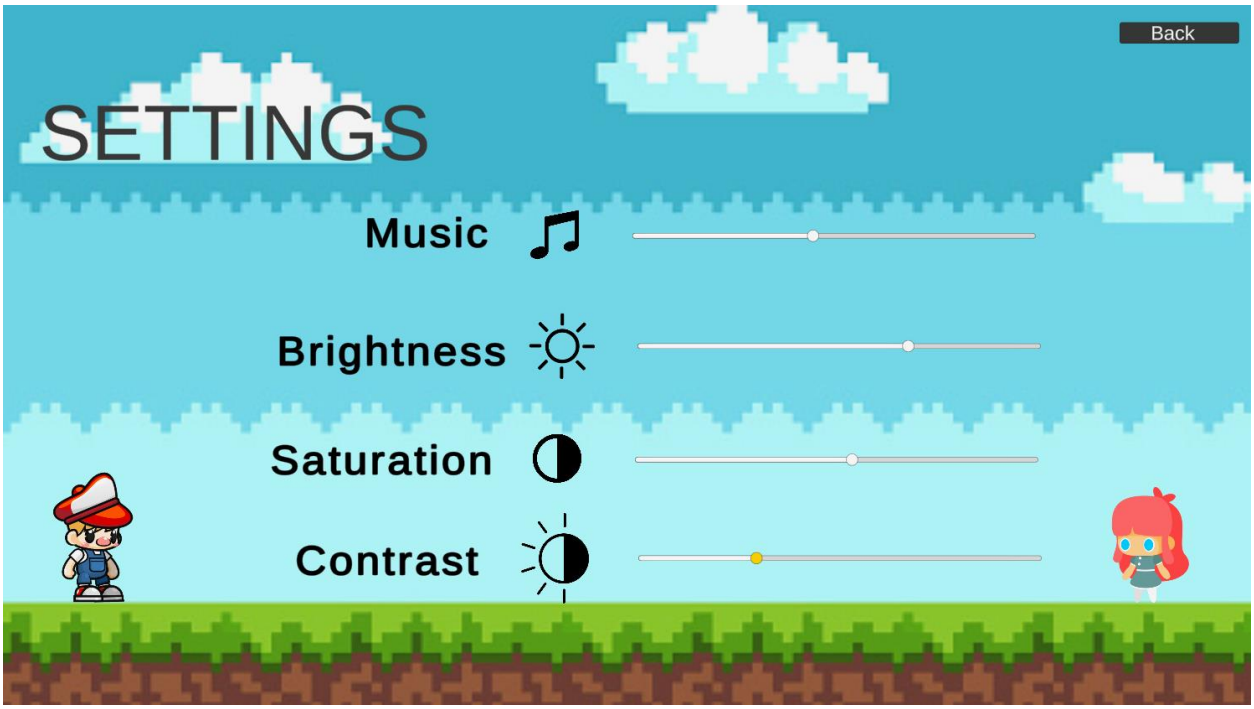


Figure 198: Settings Panel

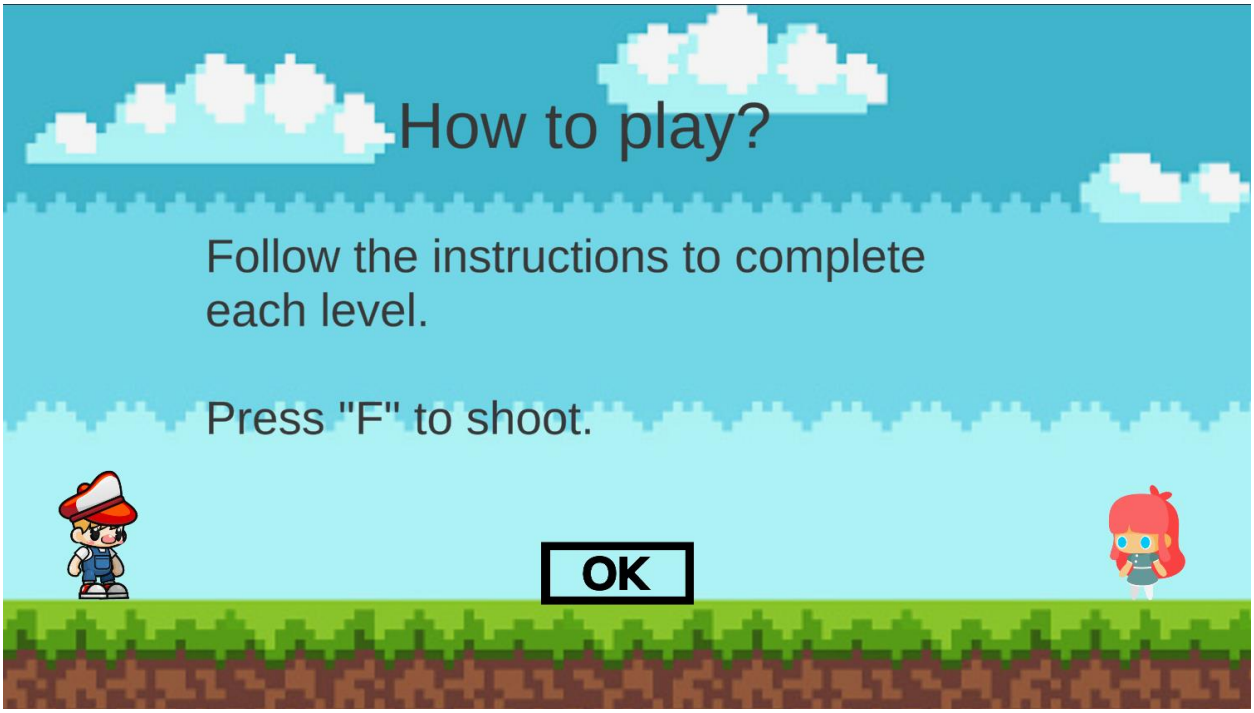


Figure 199: How to Play Panel

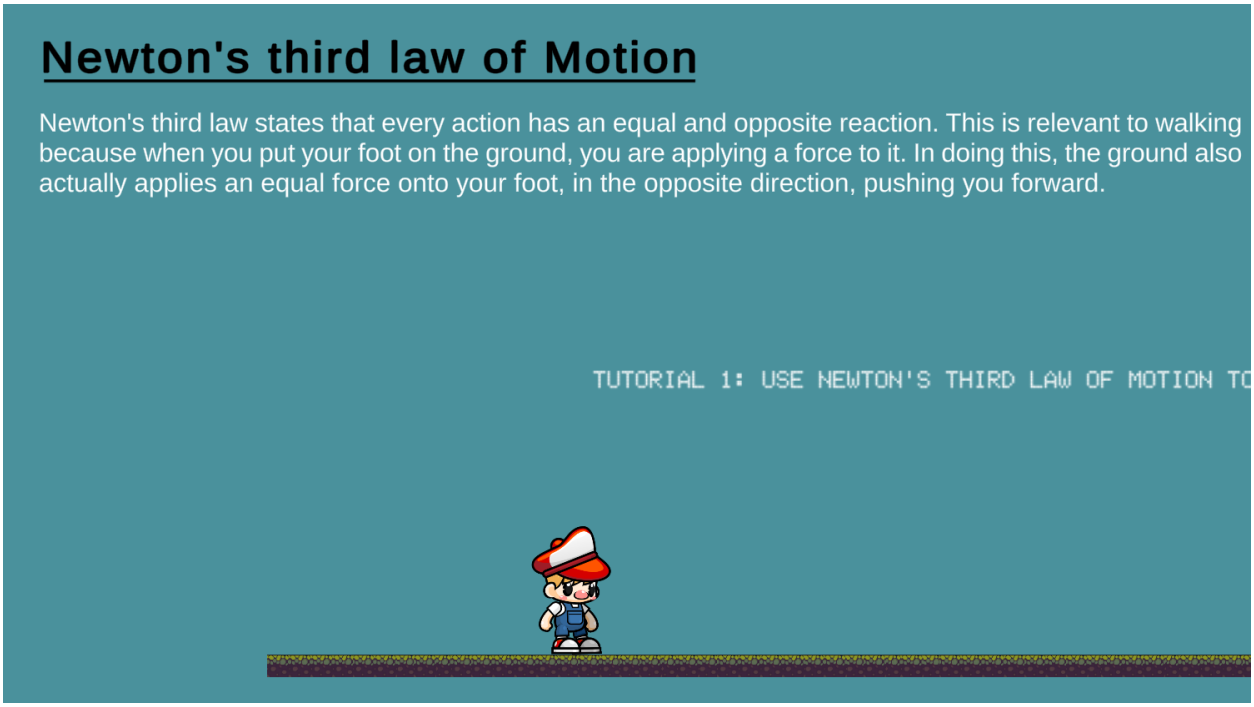


Figure 200: Tutorial 1

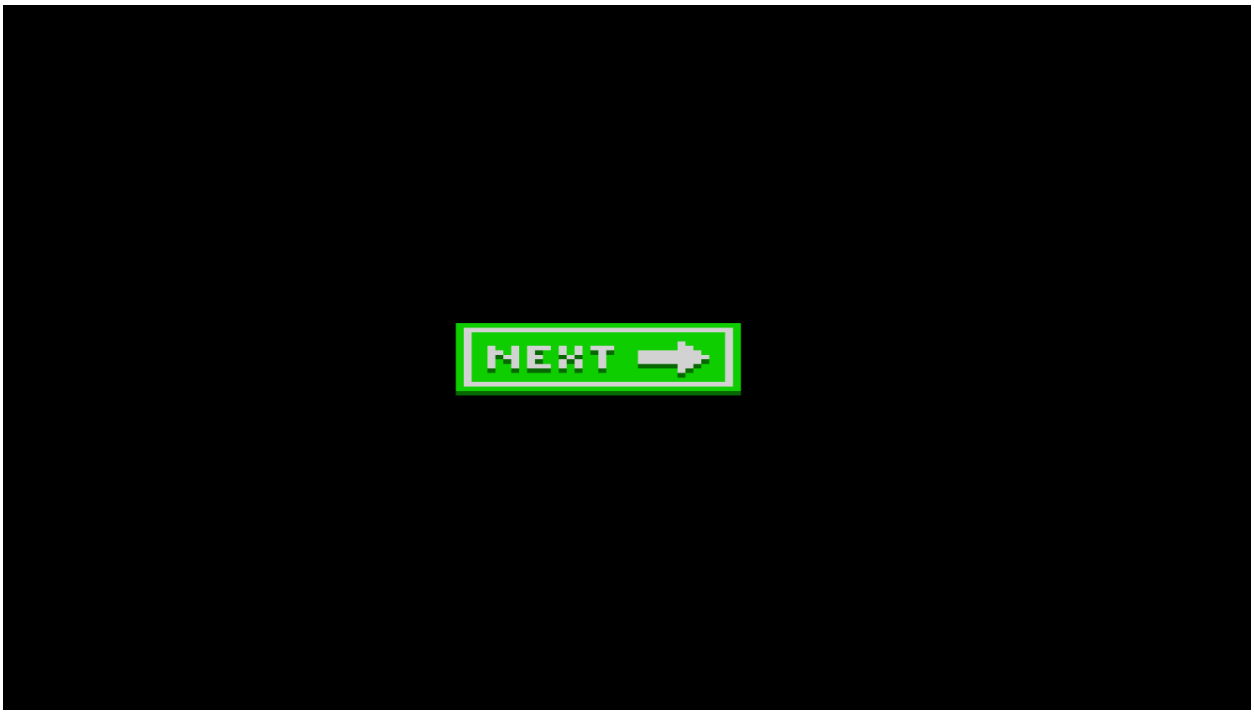


Figure 201: Next Panel

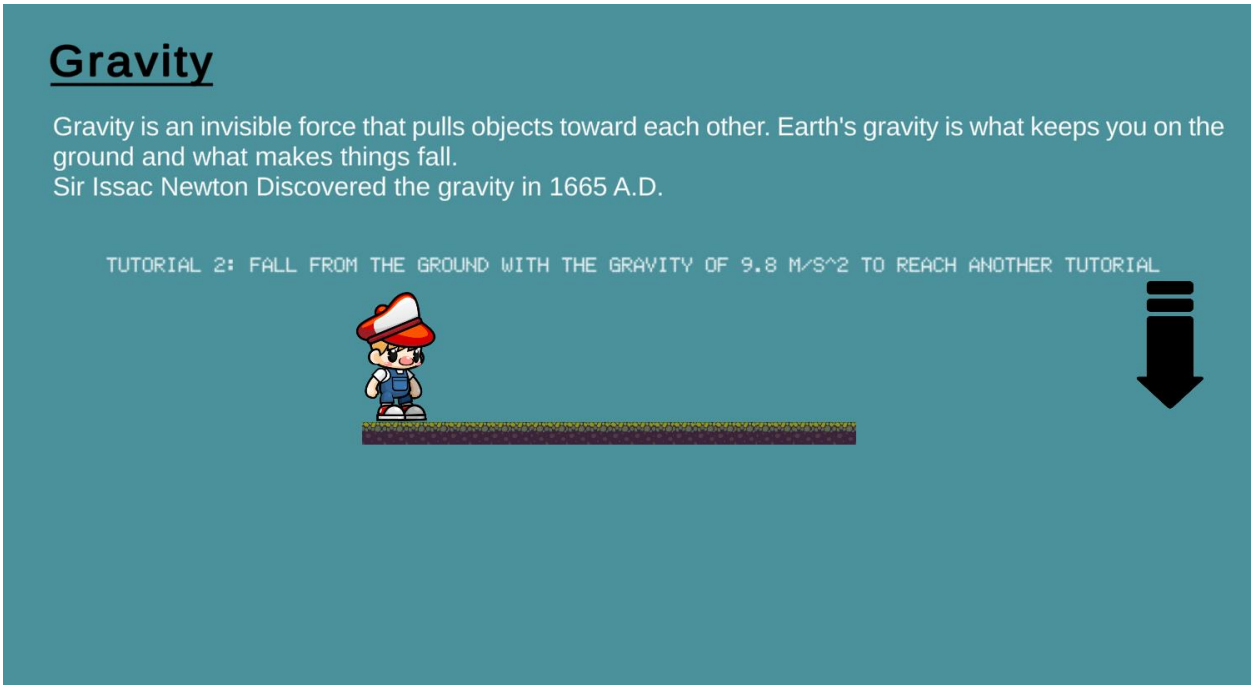


Figure 202: Tutorial 2

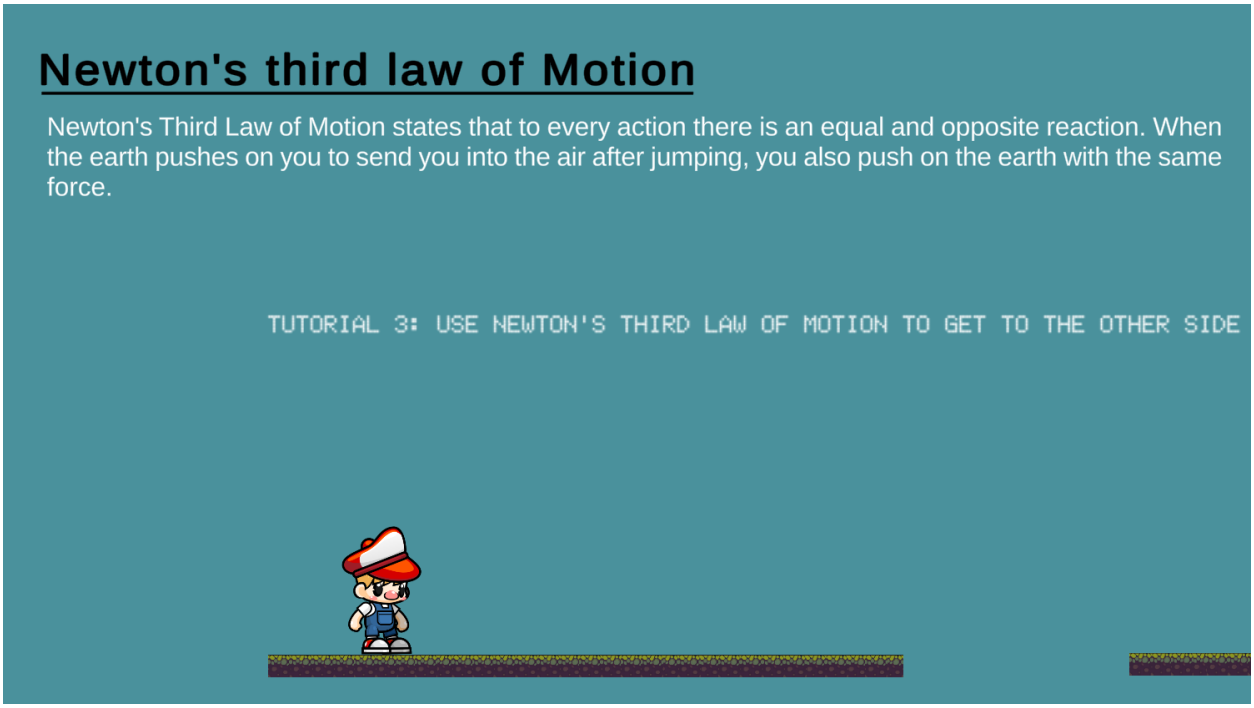


Figure 203: Tutorial 3

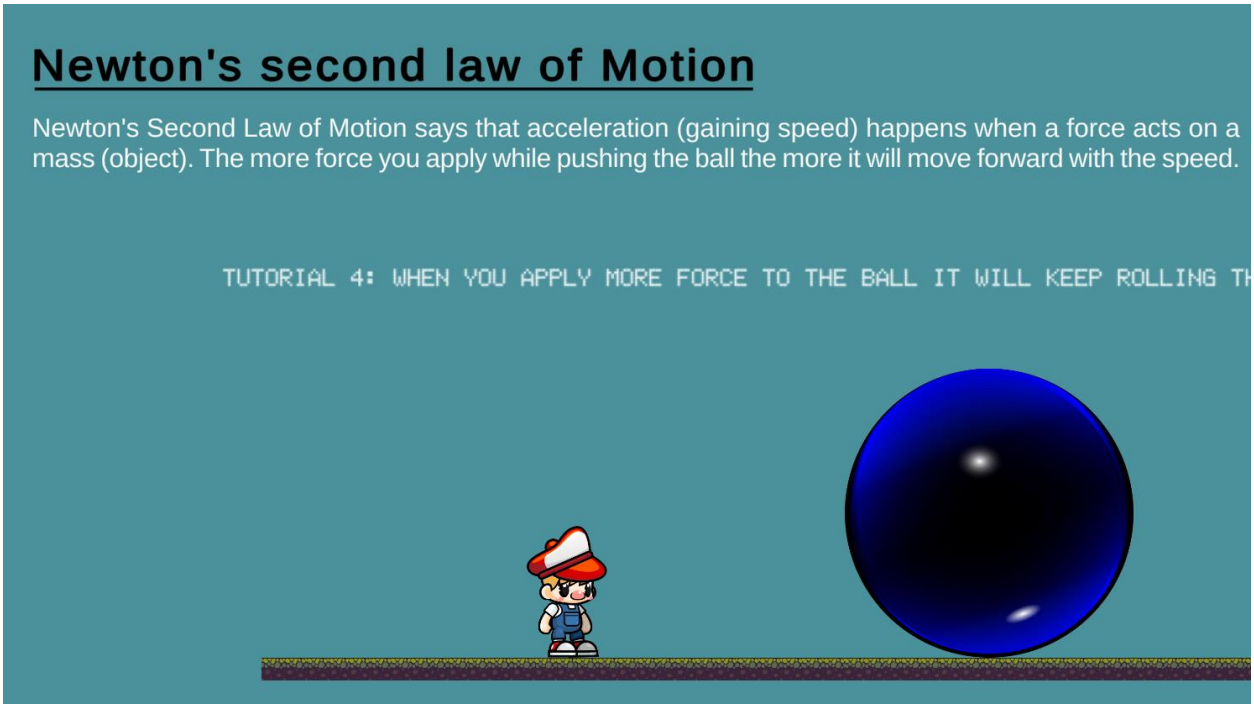


Figure 204: Tutorial 4

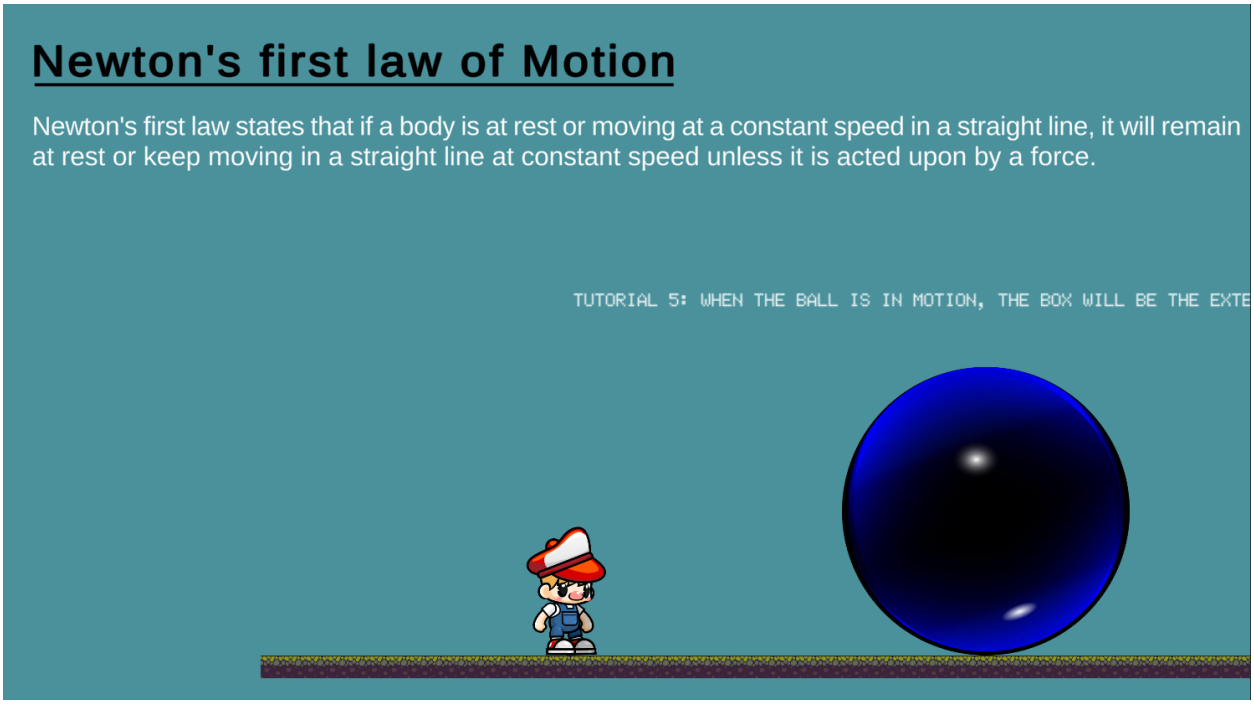


Figure 205: Tutorial 5

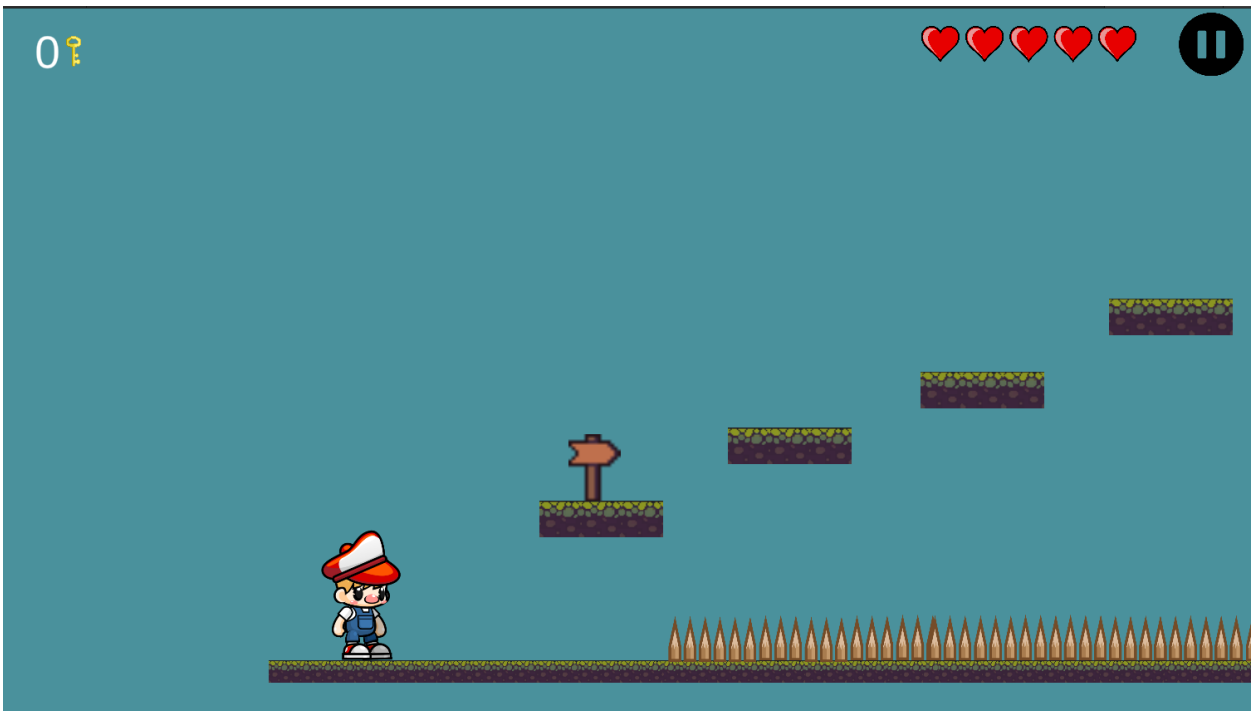


Figure 206: Level 1

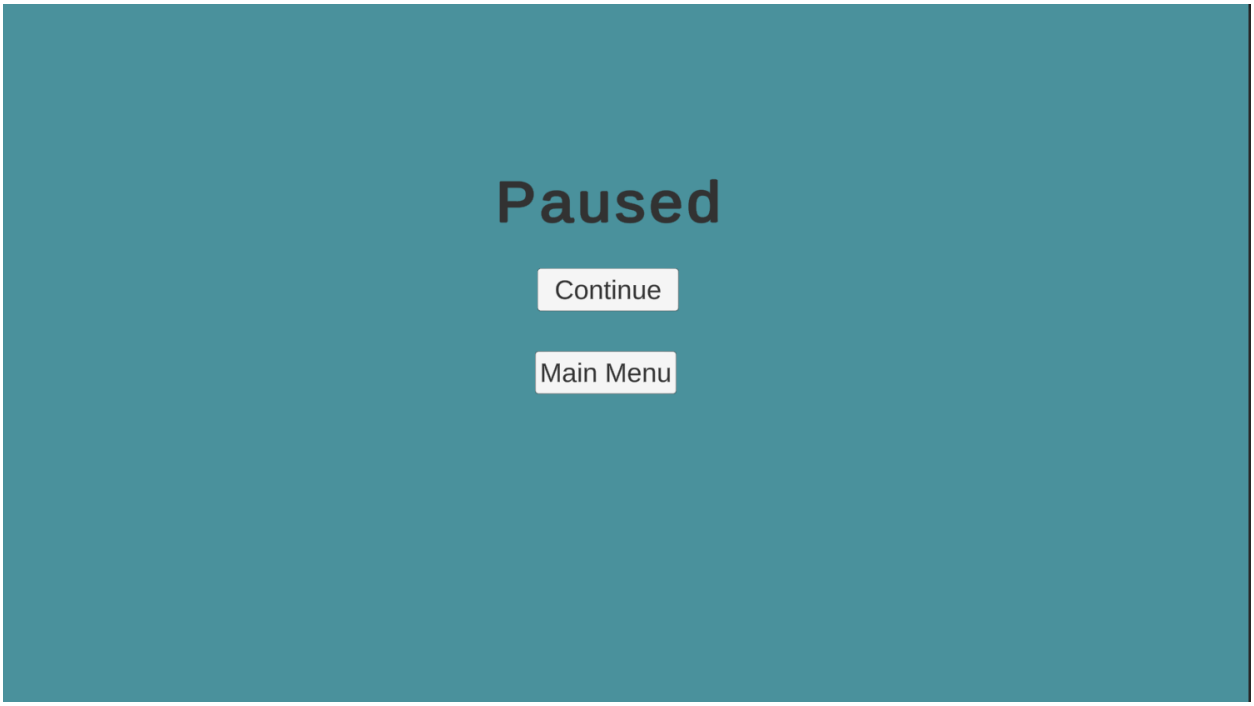


Figure 207: Paused Panel

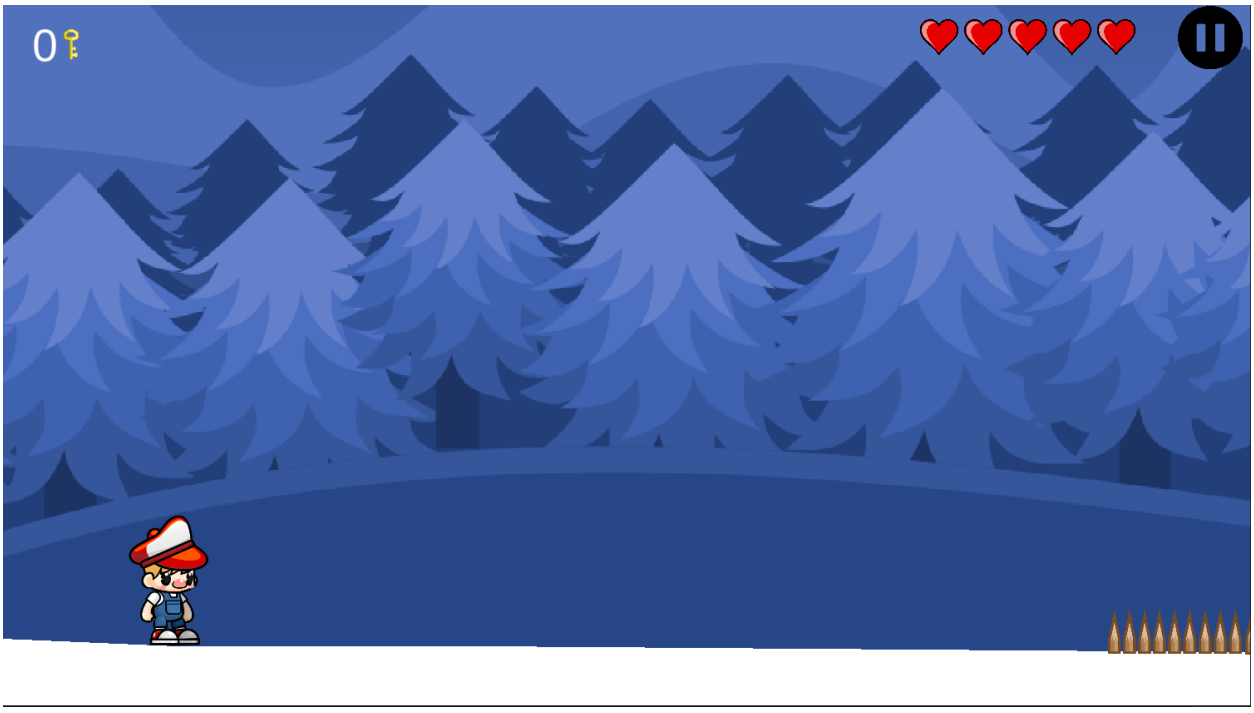


Figure 208: Level 2

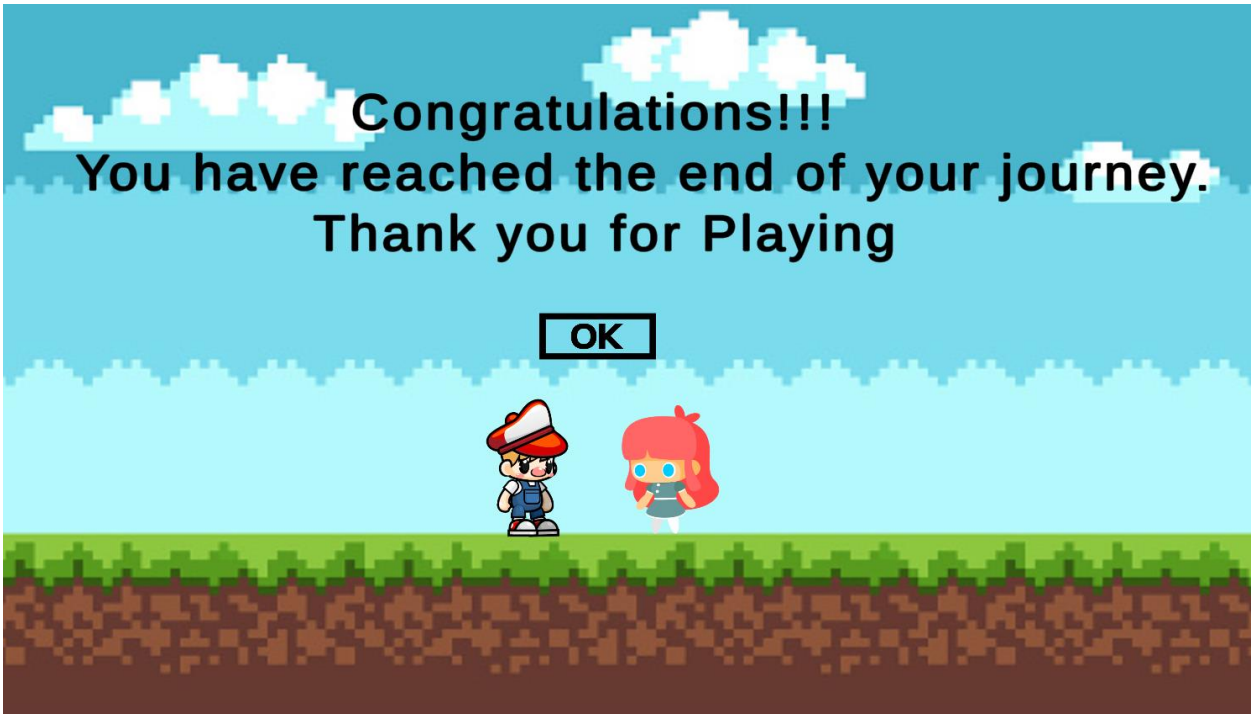


Figure 209: Congrats Panel

8.7. Appendix G: User Feedback

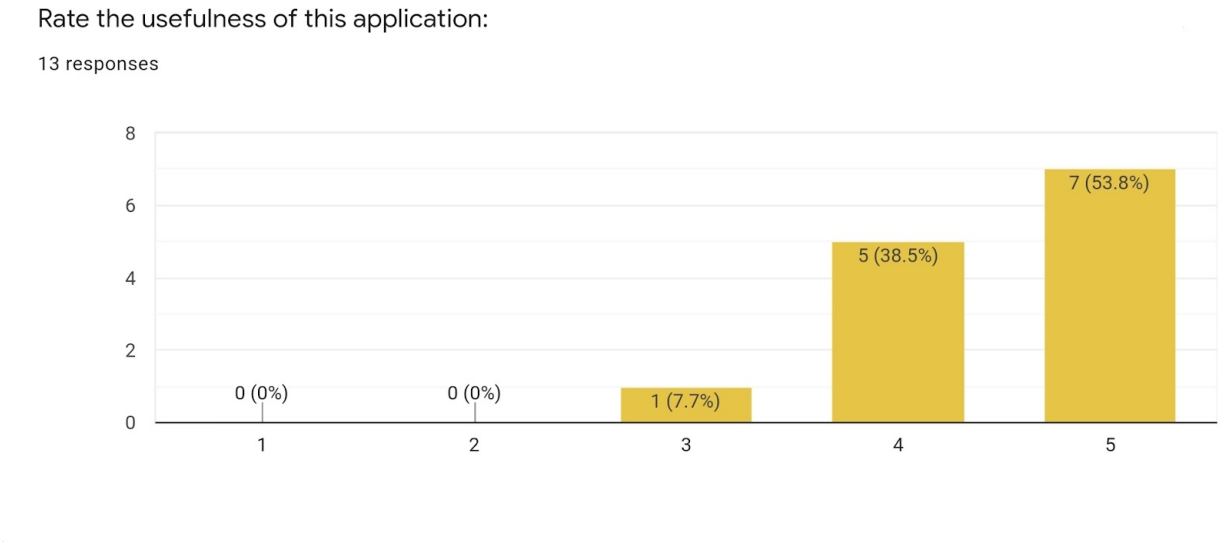


Figure 210: User Rating

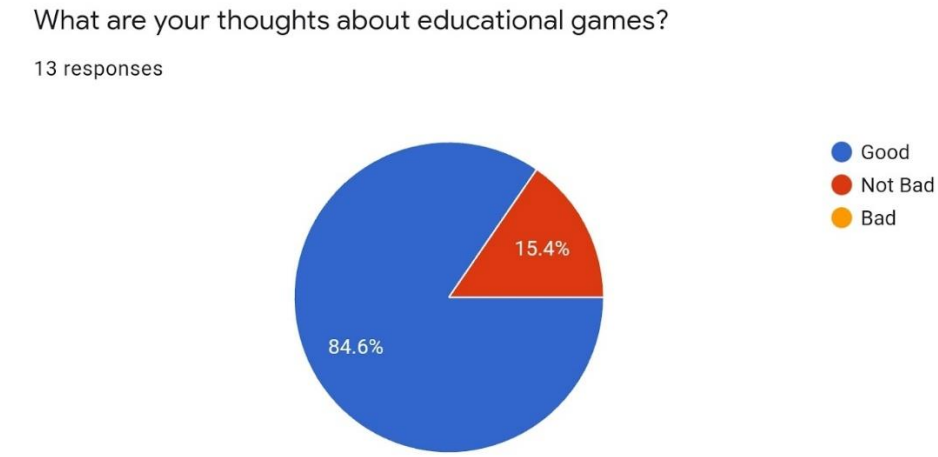


Figure 211: User thoughts

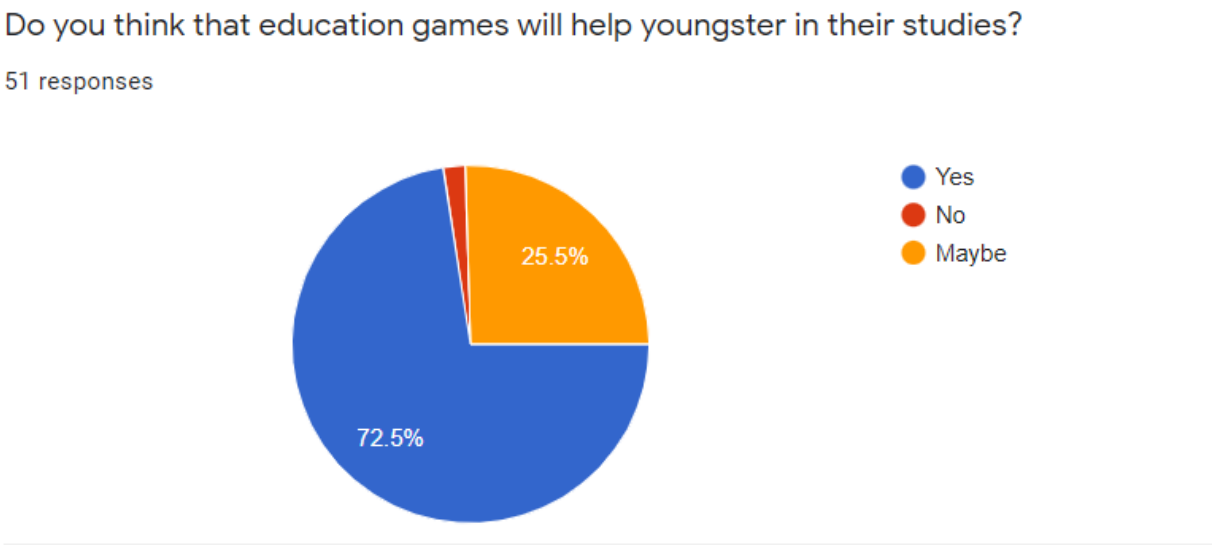


Figure 212: User thought of this app for youngsters

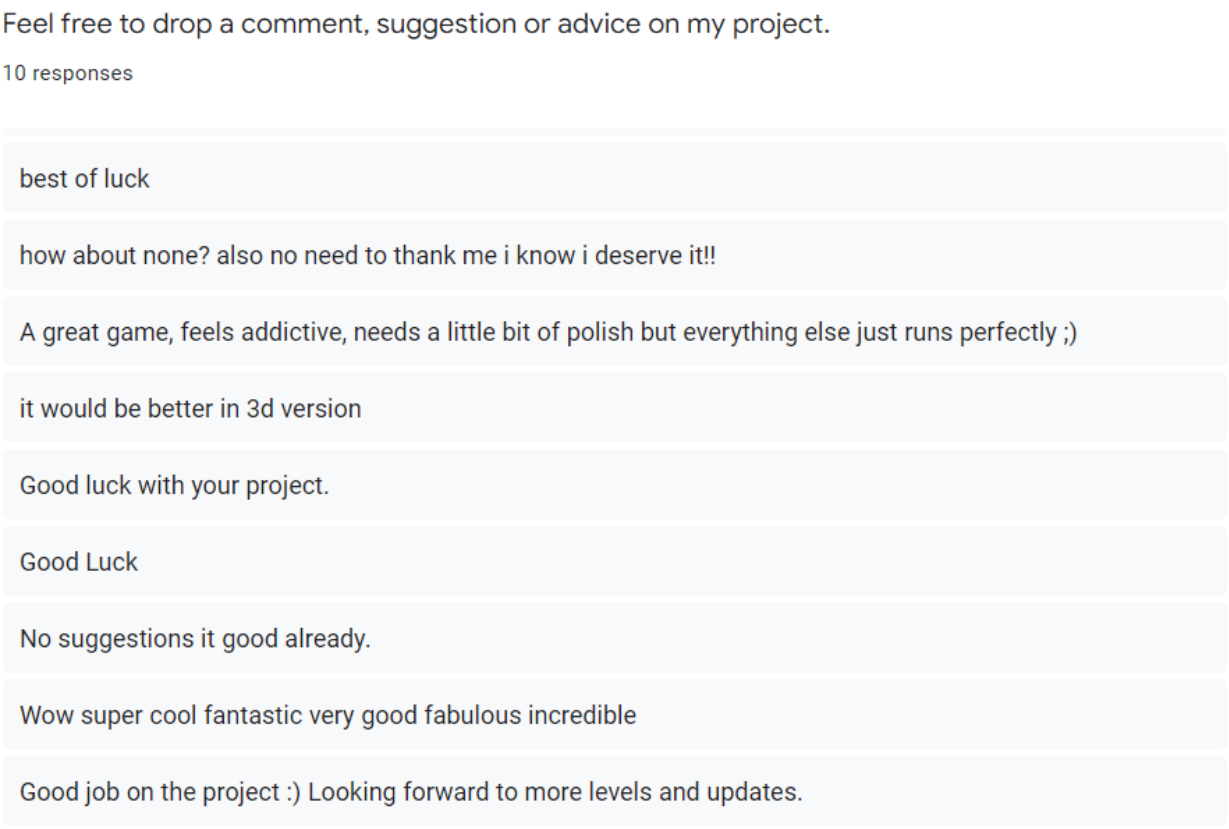


Figure 213: User comments

8.8. Appendix H: Development code

PlayerMovement.cs

```
1.  using UnityEngine;
2.  using UnityEngine.SceneManagement;
3.
4.  public class PlayerMovement : MonoBehaviour
5.  {
6.      //SerializedFeild to protect the value
7.      //Setting the data type for these variables
8.      //For Speed
9.      [SerializeField] float speed;
10.
11.     //For jumpspeed
12.     [SerializeField] float jumpspeed;
13.
14.     //For movement
15.     [SerializeField] float movement;
16.
17.     //For jump
18.     [SerializeField] float jump;
19.
20.     //Rigidbody2D for Conan as Conan is Ridgibody2D GameObject
21.     [SerializeField] Rigidbody2D Conan;
22.
23.     //Animator for animator as animator is Animator used for adding animation frame by frame
24.     [SerializeField] Animator animator;
25.
26.     //To check if the player is touching the ground
27.     [SerializeField] bool isTouchingGround;
28.
29.     //To take the transform postion of the empty GameObject inside the player
30.     [SerializeField] Transform groundCheck;
31.
32.     //To check the layer
33.     [SerializeField] LayerMask Layer;
34.
35.     //Adding radius to the empty GameObject to check the LayerMask
36.     [SerializeField] float R;
37.
38.     //To check if player is alive
39.     [SerializeField] bool isAlive;
40.
41.     //To take the location of the player
42.     Vector2 location;
43.
44.     //To keep Game World as GameObject
45.     [SerializeField] GameObject Gameworld;
46.
47.     //To check if Double Jump Buff is available
48.     public bool DoubleJump;
49.
50.     //To check if the Double Jump buff is available
51.     public bool DJBuff;
52.
53.     //Taking DjBuff as the DjumpBuff object
54.     DjumpBuff djBuff;
55.
56.     // Start is called before the first frame update
```

```
57.     void Start()
58.     {
59.         //Setting is alive true
60.         isAlive = true;
61.
62.         //Taking the current position
63.         location = this.transform.position;
64.
65.         //To get component that have Rigidbody2D component i.e. Conan
66.         Conan = GetComponent<Rigidbody2D>();
67.
68.         //Finding object type DjumpBuff
69.         djBuff = FindObjectOfType<DjumpBuff>();
70.     }
71.
72.     // Update is called once per frame
73.     void Update()
74.     {
75.         //Stopping Move of Player when he is dead
76.         if (isAlive == false)
77.         {
78.             Conan.velocity = new Vector2(0, Conan.velocity.y);
79.             return;
80.         }
81.
82.         //Making circle to check the Layer
83.         isTouchingGround = Physics2D.OverlapCircle(groundCheck.position, R, Layer);
84.
85.         //Calling
86.         Movement();
87.         Jump();
88.     }
89.
90.     private void Movement()
91.     {
92.         //Getting axis as "Horizontal" and multiplying with speed everytime a or d or side
    arrows key
93.         float movement = Input.GetAxis("Horizontal") * speed;
94.
95.         //Animator movement
96.         animator.SetFloat("Speed", Mathf.Abs(movement));
97.
98.         //Local positon
99.         if (movement > 0)
100.          {
101.              transform.localRotation = Quaternion.Euler(0, 0, 0);
102.          }
103.
104.          //Turning
105.          else if (movement < 0)
106.          {
107.              transform.localRotation = Quaternion.Euler(0, 180, 0);
108.          }
109.          Conan.velocity = new Vector2(movement, Conan.velocity.y);
110.
111.      }
112.
113.      private void Jump()
114.      {
115.          //Getting Axis as "Jump" and multiplying every time with jumpspeed everytime we
    use jump button
116.          float jump = Input.GetAxis("Jump") * jumpspeed;
117.
118.          //Setting animation for jump
```

```
119.            animator.SetFloat("Jump", Mathf.Abs(jump));
120.
121.            //Condition for double jump
122.            if (Input.GetKeyDown(KeyCode.Space) && isTouchingGround == true)
123.            {
124.
125.                Conan.velocity = new Vector2(Conan.velocity.x, jump);
126.                DoubleJump = DJBuff;
127.            }
128.            else if (Input.GetKeyDown(KeyCode.Space) && DoubleJump)
129.            {
130.                Conan.velocity = new Vector2(Conan.velocity.x, jump);
131.                DoubleJump = false;
132.
133.            }
134.        }
135.
136.        //For Parent and Child
137.        private void OnCollisionEnter2D(Collision2D col)
138.        {
139.            if (col.gameObject.tag=="RightToLeft")
140.                this.transform.parent = col.transform;
141.        }
142.
143.        //For removing parent and child relation of "RightToLeft" GameObject and player
144.        private void OnCollisionExit2D(Collision2D col)
145.        {
146.            if (col.gameObject.tag == "RightToLeft")
147.                this.transform.parent = Gameworld.transform;
148.        }
149.
150.        //For Trigger
151.        private void OnTriggerEnter2D(Collider2D collison)
152.        {
153.            //For PlayerDeath Animation
154.            if (collison.gameObject.tag == "Dmg")
155.            {
156.                PlayerDeath();
157.            }
158.
159.            //For PlayerDeath Animation
160.            if (collison.gameObject.tag == "Enemy")
161.            {
162.                PlayerDeath();
163.            }
164.
165.            //To go to the next scene
166.            if (collison.gameObject.tag == "Next")
167.            {
168.                //Loading next scene
169.                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
170.            }
171.
172.            //Destorying Key after getting triggered by player
173.            if (collison.gameObject.tag == "Key")
174.            {
175.                //Destorying Key
176.                Destroy(collison.gameObject);
177.            }
178.        }
179.
180.
181.        //For PlayerDeath
182.        public void PlayerDeath()
```

```
183.      {
184.          //Setting player as dead
185.          isAlive = false;
186.
187.          //Play death animation
188.          animator.SetFloat("Jump", 0);
189.          animator.SetBool("IsDead", !isAlive);
190.
191.          //Wait for 2 sec
192.          //Respawn player
193.          Invoke("RespawnPlayer", 2f);
194.
195.      }
196.
197.      //For RespawPlayer
198.      public void RespawnPlayer()
199.      {
200.          //Setting Player as alive
201.          isAlive = true;
202.          animator.SetBool("IsDead", !isAlive);
203.
204.          //Respawing at the first position and in player facing the same direction
205.          transform.position = location;
206.          transform.rotation = Quaternion.identity;
207.
208.          //Enabling SpriteRenderer of the double jump buff
209.          djBuff.GetComponent<SpriteRenderer>().enabled = true;
210.
211.          //Enabling CircleCollider2D of the double jump buff
212.          djBuff.GetComponent<CircleCollider2D>().enabled = true;
213.      }
214.
215.
216.  }
217.
```

Camera.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class Camera : MonoBehaviour
6.  {
7.      [SerializeField] GameObject Conan;
8.
9.      //Starting Point for horizonatal
10.     [SerializeField] float startingPoint;
11.
12.     //Starting point for vertical
13.     [SerializeField] float sPoint;
14.
15.     //Ending point for horizontal
16.     [SerializeField] float endingPoint;
17.
18.     //Ending point for vertical
19.     [SerializeField] float ePoint;
20.
21.     // Update is called once per frame
22.     void Update()
23.     {
24.        //Claming camera for horizontal Movement of player
25.        float newpoint= Mathf.Clamp(Conan.transform.position.x, startingPoint, endingPoint);
26.
27.         //Claming camera for vertical Movement of player
28.         float npoint = Mathf.Clamp(Conan.transform.position.y, sPoint, ePoint);
29.
30.         //For freezing Z axis
31.        transform.position = new Vector3(newpoint, npoint,transform.position.z);
32.     }
33. }
34.
```

bulletScript.cs

```
1.  using UnityEngine;
2.
3.  public class bulletScript : MonoBehaviour
4.  {
5.      //SerizlizeFeild to protect value
6.      //Taking BulletRB as RigidBody2D
7.      public Rigidbody2D bulletRB;
8.
9.      //Bullet Speed
10.     [SerializeField] float bulletSpeed;
11.
12.     //For Damage Value
13.     public float dmg;
14.
15.     //For impactEffect
16.     [SerializeField] GameObject impactEffect;
17.     void Start()
18.     {
19.         //getting GameObject that have Rigidbody2D component i.e. bulltetRB
20.         bulletRB = GetComponent<Rigidbody2D>();
21.     }
22.
23.     // Update is called once per frame
24.     void Update()
25.     {
26.         //Adding Velocty in bullet
27.         bulletRB.velocity = transform.right * bulletSpeed;
28.
29.         //Destorying bullet after 2 second
30.         Destroy(gameObject, 2f);
31.
32.     }
33.
34.     //On Collison
35.     private void OnCollisionEnter2D(Collision2D collision)
36.     {
37.         //Condition if the bullet collides with Ground
38.         if (collision.gameObject.tag == "Ground" )
39.         {
40.             //Destroy bullet
41.             Destroy(gameObject);
42.
43.             //ImpactEffect got instantiate where the bullet hit the ground
44.             Instantiate(impactEffect, transform.position, transform.rotation);
45.         }
46.         if (collision.gameObject.tag == "Enemy")
47.         {
48.             //Destroy Bullet
49.             Destroy(gameObject);
50.
51.             //ImpactEffect got instantiate where the bullet hit the Enemy
52.             Instantiate(impactEffect, transform.position, transform.rotation);
53.
54.             //Finding the Vllain Script and dealing dmg
55.             FindObjectOfType<Villain>().health -= dmg;
56.
57.             //Finding EnemyFollow Script and dealing dmg
58.             FindObjectOfType<EnemyFollow>().health -= dmg;
59.         }
60.     }
```

```
61.
62. }
63.
```

NextLevel.cs

```
1.  using UnityEngine;
2.  using UnityEngine.SceneManagement;
3.
4.  public class NextLevel : MonoBehaviour
5.  {
6.      //This work on collison with Player
7.      private void OnCollisionEnter2D(Collision2D collision)
8.      {
9.          //Sending to the next scene
10.         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
11.     }
12. }
13.
```

Next.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.SceneManagement;
5.  public class Next : MonoBehaviour
6.  {
7.      private void Start()
8.      {
9.          //Time scale to 1
10.         Time.timeScale = 1;
11.     }
12.
13.     //For Play
14.     public void PlayGame()
15.     {
16.         //Next Scene Loading
17.         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
18.     }
19.
20.     //For Exit
21.     public void ExitGame()
22.     {
23.         //Quiting application
24.         Application.Quit();
25.     }
26.
27.     //For MainMent
28.      public void MainMenu()
29.     {
30.         //Loading Scene 0
31.         SceneManager.LoadScene(0);
32.     }
33. }
34.
```

KeyManager.cs

```
1.  using UnityEngine;
2.  using TMPro;
3.
4.  public class KeyManager : MonoBehaviour
5.  {
6.      //Setting KeyManger as static
7.      public static KeyManager instance;
8.
9.      //Using TMPROGUI text
10.     public TextMeshProUGUI text;
11.
12.     //KeyScore
13.     public int keyScore;
14.
15.
16.     // Start is called before the first frame update
17.     void Start()
18.     {
19.         //Conditon for instance
20.         if(instance == null)
21.         {
22.             instance = this;
23.         }
24.     }
25.
26.     //or ChangeKeyValue
27.     public void ChangeKeyValue(int keyValue)
28.     {
29.         //KeyScore added as keyValue
30.         keyScore += keyValue;
31.
32.         //KeyScore to string to show in message
33.         text.text = keyScore.ToString();
34.     }
35. }
36.
```

Destroy.cs

```
1.  using UnityEngine;
2.
3.  public class Destroy : MonoBehaviour
4.  {
5.      // Start is called before the first frame update
6.      void Start()
7.      {
8.          Destroy(gameObject, 0.5f);
9.      }
10.
11.
12. }
```

Key.cs

```
1.  using UnityEngine;
2.
3.  public class Key : MonoBehaviour
4.  {
5.      //Setting Key Value to 1
6.      public int keyValue = 1;
7.
8.      //On Trigger
9.      private void OnTriggerEnter2D(Collider2D other)
10.     {
11.         //Contion if it gets trigger by Player
12.         if (other.gameObject.tag =="Player")
13.         {
14.             //Changing the KeyValue
15.             KeyManager.instance.ChangeKeyValue(keyValue);
16.         }
17.     }
18. }
19.
```

DjBuff.cs

```
1.  using System.Collections;
2.  using UnityEngine;
3.  using TMPro;
4.
5.  public class DjumpBuff : MonoBehaviour
6.  {
7.      //Message as TMPROGUI
8.      [SerializeField] TextMeshProUGUI Msg;
9.
10.     //Setting TextArea
11.     [TextArea(10, 4)]
12.
13.     //Taking Message
14.     [SerializeField] string textMessage;
15.
16.     //Player Collider
17.     public Collider2D Player;
18.
19.     //Duration
20.     public float duration = 1;
21.
22.     //Random
23.     public float random = 0;
24.
25.    //Taking PlayerMovement Object as J
26.     public PlayerMovement J;
27.
28.
29.     // Start is called before the first frame update
30.     void Start()
31.     {
32.         //Finding PlayerMovement Copmponent
33.         PlayerMovement J = Player.GetComponent<PlayerMovement>();
34.     }
35.
36.     // Update is called once per frame
37.     void Update()
38.     {
39.         //Conditon for random
40.         if (random == 2)
41.         {
42.             //Duration value goes down with real time by -0.2
43.             duration = duration - 0.2f * Time.deltaTime;
44.         }
45.         if (duration <= 0)
46.         {
47.             random = 0;
48.             PlayerMovement m = Player.GetComponent<PlayerMovement>();
49.
50.             //Double jump buff off
51.             J.DJBuff= false;
52.             duration = 1;
53.         }
54.     }
55.
56.     //On Trigger
57.     public void OnTriggerEnter2D(Collider2D other)
58.     {
59.         //If triggered by Player
60.         if (other.gameObject.tag == "Player")
```

```
61.         {
62.             //JumpTwice Coroutine started
63.             StartCoroutine(JumpTwice(other));
64.
65.             //Value of random set to 2
66.             random = 2;
67.
68.             //MessageTimer Coroutine started
69.             StartCoroutine(MessageTimer());
70.         }
71.     }
72.
73.     //Inenumerater for JumpTwice
74.     IEnumerator JumpTwice(Collider2D player)
75.     {
76.         //Setting double jump buff on
77.         J.DJBuff = true;
78.
79.         //Turning off the components
80.         GetComponent<SpriteRenderer>().enabled = false;
81.         GetComponent<CircleCollider2D>().enabled = false;
82.         yield return new WaitForSeconds(2);
83.     }
84.
85.     //Inenumerater for JMessageTimer
86.     IEnumerator MessageTimer()
87.     {
88.         Msg.text = textMessage;
89.         yield return new WaitForSeconds(5);
90.         Msg.text = " ";
91.     }
92. }
93.
```

Door.cs

```
1.  using UnityEngine;
2.
3.  public class Door : MonoBehaviour
4.  {
5.      private KeyManager kValue;
6.      private void Start()
7.      {
8.          kValue = FindObjectOfType<KeyManager>();
9.      }
10.     private void OnCollisionEnter2D(Collision2D collision)
11.     {
12.         if (collision.gameObject.tag == "Player")
13.         {
14.             if (kValue.keyScore==1)
15.             {
16.                 Destroy(gameObject);
17.             }
18.
19.         }
20.     }
21. }
22.
```

Message.cs

```
1.  using System.Collections;
2.  using UnityEngine;
3.  using TMPro;
4.
5.  public class Message : MonoBehaviour
6.  {
7.      [SerializeField] TextMeshProUGUI Msg;
8.      [TextArea(10, 4)]
9.      [SerializeField] string textMessage;
10.     [SerializeField] float messagetimer = 1;
11.
12.     private void OnTriggerEnter2D(Collider2D collision)
13.     {
14.         StartCoroutine(MessageTimer());
15.     }
16.
17.     IEnumerator MessageTimer()
18.     {
19.         Msg.text = textMessage;
20.         yield return new WaitForSeconds(messagetimer);
21.         Msg.text = " ";
22.     }
23. }
24.
```

HealthDecrease.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class HealthDecrease : MonoBehaviour
6.  {
7.      LifeLine life;
8.      // Start is called before the first frame update
9.      void Start()
10.     {
11.         life = FindObjectOfType<LifeLine>();
12.     }
13.
14.     // Update is called once per frame
15.     void Update()
16.     {
17.
18.     }
19.
20.     private void OnTriggerEnter2D(Collider2D collision)
21.     {
22.         if (collision.gameObject.tag=="Dmg")
23.         {
24.             life.decreaseHealth(1);
25.
26.         }
27.         if (collision.gameObject.tag == "Enemy")
28.         {
29.             life.decreaseHealth(1);
30.
31.         }
32.     }
33. }
34.
```

LifeLine.cs

```
1.  using System.Collections;
2.  using UnityEngine;
3.  using UnityEngine.SceneManagement;
4.
5.  public class LifeLine : MonoBehaviour
6.  {
7.      //Setting public for varibale so the we can change it values using other script
8.      //Taking GameObjets
9.      public GameObject h1, h2, h3, h4, h5, over;
10.
11.     //Health of the Player
12.     public int health;
13.
14.     //End Time
15.     public float endtime;
16.
17.     //Value
18.     public float Value;
19.
20.     //To check if the Game is Over
21.     [SerializeField] bool isGameOver;
22.
23.
24.     // Start is called before the first frame update
25.     void Start()
26.     {
27.         health = 5;
28.         h1.gameObject.SetActive(true);
29.         h2.gameObject.SetActive(true);
30.         h3.gameObject.SetActive(true);
31.         h4.gameObject.SetActive(true);
32.         h5.gameObject.SetActive(true);
33.         over.gameObject.SetActive(false);
34.         Value = 2;
35.         endtime = 1;
36.     }
37.
38.     //For GameOver
39.     private void GameOver()
40.     {
41.         //Condition for GameOver
42.         if (isGameOver ==true)
43.         {
44.             //Starting Coroutine For Message
45.             StartCoroutine(MessageTimer());
46.
47.         }
48.     }
49.      IEnumerator MessageTimer()
50.     {
51.        //Stopping time
52.         Time.timeScale = 0;
53.         yield return new WaitForSeconds(3);
54.         Time.timeScale = 1;
55.         SceneManager.LoadScene(0);
56.     }
57.
58.     //For Decreasing Health
59.     public void decreaseHealth(int dmg)
60.     {
61.         //Switch Conditons
```

```
62.         health -= dmg;
63.         if (health > 5)
64.             health = 5;
65.
66.         switch (health)
67.         {
68.             case 5:
69.                 break;
70.             case 4:
71.                 h5.gameObject.SetActive(false);
72.                 break;
73.             case 3:
74.                 h4.gameObject.SetActive(false);
75.                 h5.gameObject.SetActive(false);
76.                 break;
77.             case 2:
78.                 h3.gameObject.SetActive(false);
79.                 h4.gameObject.SetActive(false);
80.                 h5.gameObject.SetActive(false);
81.                 break;
82.             case 1:
83.                 h2.gameObject.SetActive(false);
84.                 h3.gameObject.SetActive(false);
85.                 h4.gameObject.SetActive(false);
86.                 h5.gameObject.SetActive(false);
87.                 break;
88.             case 0:
89.                 h1.gameObject.SetActive(false);
90.                 h2.gameObject.SetActive(false);
91.                 h3.gameObject.SetActive(false);
92.                 h4.gameObject.SetActive(false);
93.                 h5.gameObject.SetActive(false);
94.                 over.gameObject.SetActive(true);
95.                 isGameOver = true;
96.                 GameOver();
97.                 break;
98.         }
99.     }
100.  }
101.
```

Topdown.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class Topdown : MonoBehaviour
6.  {
7.      //Starting point
8.      [SerializeField] float StartPoint;
9.
10.     //Ending Point
11.     [SerializeField] float EndPoint;
12.
13.     //Speed
14.     [SerializeField] float speed;
15.
16.     //Moving top check
17.     [SerializeField] bool moveTop = true;
18.     void Update()
19.     {
20.         //Conditions
21.         if (transform.position.y >= EndPoint)
22.         {
23.             moveTop = false;
24.         }
25.         if (transform.position.y <= StartPoint)
26.         {
27.             moveTop = true;
28.         }
29.
30.         if (moveTop)
31.             transform.position = new Vector2(transform.position.x, transform.position.y +
    speed * Time.deltaTime);
32.         else
33.             transform.position = new Vector2(transform.position.x, transform.position.y -
    speed * Time.deltaTime);
34.
35.     }
36. }
37.
```

Villain.cs

```
1.  using UnityEngine;
2.
3.  public class Villain : MonoBehaviour
4.  {
5.      //Starting Point
6.      [SerializeField] float StartPoint;
7.
8.      //Ending Point
9.      [SerializeField] float EndPoint;
10.
11.     //Speed
12.     [SerializeField] float speed;
13.
14.     //Check Move right
15.     [SerializeField] bool moveRight = true;
16.
17.     //Health
18.     public float health;
19.     void Update()
20.     {
21.         if (health == 0)
22.         {
23.             //Destroying Villain
24.             Destroy(gameObject);
25.         }
26.
27.         //Condition for rotation
28.         if (transform.position.x >= EndPoint)
29.         {
30.             transform.localRotation = Quaternion.Euler(0, 180, 0);
31.             moveRight = false;
32.         }
33.         if (transform.position.x <= StartPoint)
34.         {
35.            transform.localRotation = Quaternion.Euler(0, 0, 0);
36.             moveRight = true;
37.         }
38.
39.         //Condition for Move right
40.         if (moveRight)
41.             transform.position = new Vector2(transform.position.x + speed * Time.deltaTime,
    transform.position.y);
42.         else
43.             transform.position = new Vector2(transform.position.x - speed * Time.deltaTime,
    transform.position.y);
44.
45.     }
46.
47.     //On Collison
48.     private void OnCollisionEnter2D(Collision2D collision)
49.     {
50.         //If collides with Player
51.         if (collision.gameObject.tag=="Player")
52.         {
53.             //Self destruction
54.             Destroy(gameObject);
55.         }
56.     }
57. }
58.
```

Spawner.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class Spawner : MonoBehaviour
6.  {
7.      [SerializeField] GameObject ObjectToBeSpawn;
8.      BoxCollider2D Spw;
9.      // Start is called before the first frame update
10.     void Start()
11.     {
12.         //Get Object Having BoxCollider2D i.e. Spw
13.         Spw = GetComponent<BoxCollider2D>();
14.     }
15.
16.     public BoxCollider2D GetBox(bool tempBool)
17.     {
18.         Spw.enabled = tempBool;
19.         return Spw;
20.     }
21.
22.    private void SpawnObject()
23.     {
24.         //Instantiating object
25.         Instantiate(ObjectToBeSpawn, transform.position, transform.rotation);
26.     }
27.
28.     //On Trigger
29.     private void OnTriggerEnter2D(Collider2D collision)
30.     {
31.         if (collision.gameObject.tag == "Player")
32.         {
33.             SpawnObject();
34.             Spw.enabled = false;
35.         }
36.     }
37. }
38.
```

Sidetoside.cs

```
1.  using UnityEngine;
2.
3.  public class Sidetoside : MonoBehaviour
4.  {
5.      [SerializeField] float StartPoint;
6.      [SerializeField] float EndPoint;
7.      [SerializeField] float speed;
8.      [SerializeField] bool moveRight = true;
9.      void Update()
10.     {
11.         if (transform.position.x >= EndPoint)
12.         {
13.             moveRight = false;
14.         }
15.         if (transform.position.x <= StartPoint)
16.         {
17.             moveRight = true;
18.         }
19.
20.         if (moveRight)
21.             transform.position = new Vector2(transform.position.x + speed * Time.deltaTime,
    transform.position.y );
22.         else
23.             transform.position = new Vector2(transform.position.x - speed * Time.deltaTime,
    transform.position.y);
24.
25.     }
26. }
27.
```

Enemyscript.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class enemyScript : MonoBehaviour
6.  {
7.
8.      public float health;
9.
10.     // Update is called once per frame
11.     void Update()
12.     {
13.         if(health == 0)
14.         {
15.             Destroy(gameObject);
16.         }
17.     }
18. }
19.
```

EnemyFollow.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class EnemyFollow : MonoBehaviour
6.  {
7.      //Speed
8.      [SerializeField] float speed;
9.
10.     //Target that this AI going to follow
11.     private Transform target;
12.
13.     //Health
14.     public float health = 100;
15.
16.
17.     // Start is called before the first frame update
18.     void Start()
19.     {
20.         //Taking Player as target to follow
21.         target = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
22.     }
23.
24.     // Update is called once per frame
25.     void Update()
26.     {
27.         //Taking the position of the playre and moving towards it
28.         transform.position = Vector2.MoveTowards(transform.position, target.position, speed
    * Time.deltaTime);
29.
30.         //Condition
31.         if (health == 0)
32.         {
33.             //Self Destroy
34.             Destroy(gameObject);
35.         }
36.     }
37.
38.     //On collision
39.     private void OnCollisionEnter2D(Collision2D collision)
40.     {
41.         //if collides with Player
42.         if (collision.gameObject.tag == "Player")
43.         {
44.             //Self Destroy
45.             Destroy(gameObject);
46.         }
47.     }
48. }
49.
```