
Palestine Technical University - Kadoorie
Faculty of Engineering and Technology
Computer Systems Engineering Department

# GAMIFYING PROGRAMMING LEARNING AND PROBLEM SOLVING

**Prepared by:**
Majd Kittaneh
Omar Shaikh Ibrahim

**Supervised by:**
Nael Salman, Ph.D.

**A graduation project submitted in partial fulfillment of the requirements for the bachelor's degree in computer systems engineering.**

Tulkarm, Palestine

January, 2024

# ACKNOWLEDGEMENT

We begin by expressing our gratitude to Allah for His blessings and guidance throughout this journey. We would like to extend our heartfelt appreciation to our parents and families for their unwavering support and encouragement that enabled us to successfully complete this work. We are also deeply grateful to our university Palestine Technical University - Kadoori and specifically the Faculty of Engineering and Technology for providing us with the opportunity to pursue our studies and engage in this project.

Our utmost appreciation goes to our supervisor, Dr. Nael Salman, for his invaluable guidance, continuous support, and dedication. His expertise and insightful feedback have greatly contributed to the success of this project.

We would also like to extend our sincere thanks to all the faculty members of the Department of Computer Systems Engineering for their valuable teachings and assistance, which have been instrumental in shaping our knowledge and skills.

Lastly, we would like to acknowledge the efforts of everyone involved in this project, whose contributions have been instrumental in its completion. We are grateful for their collaboration and support throughout this endeavor.

# ABSTRACT

This thesis presents the concept of fusing game design with programming teaching, aiming to leverage the principles of game design to create an engaging and effective environment for individuals to learn programming concepts.

The idea revolves around using game mechanics, such as levels, exploration, puzzles and progression to change the challenging process of programming into an interactive and enjoyable journey.

This conceptual project sets the stage for further exploration and refinement. It raises questions about the good integration of game mechanics, choosing languages and the design of challenges to maintain a balanced blend of fun and education.

As a conceptual project, it will represent an innovative and forward-thinking approach to address the challenges in programming education. As the concept evolves, it holds the potential to redefine how individuals approach and engage with programming learning.

# Table of Contents

# List of Figures

# List of Tables

# CHAPTER 1|INTRODUCTION

*In this chapter, we will discuss the outlines of our project.*

## 1.1 Background

With the rise of generations that mainly consume digital content especially visual, there is a shift happening in learning preferences towards interactive and dynamic approaches. The traditional lecture-based learning may not fully capture the attention and interest of those individuals.

At the same time in this digital era, the ability to code is not only valuable for dedicated software developers but is increasingly becoming important for professionals in different fields.

Video games and programming share the same core: problem solving. Recognizing this opens up opportunities to leverage the interactive nature of gaming for educational purposes.

Another thing that supports the project direction is the scene of games modding –short for modification- communities, where players who are often teenagers learn programming to modify a game and change its content, which shows that players in high numbers are motivated to learn programming because of games.

## 1.2 Problem Statement

Many programming courses focus heavily on theoretical concepts, which gives learners no opportunities to apply their knowledge to real-world problems. Creating a video game environment with its own rules and objectives simulates the situation of writing a code to solve a real problem.

Traditional education follows a one-size-fits-all approach, which makes a problem for individuals with different learning styles, this and what we mentioned in the previous section about the newer generations being digital content consumers weakens the classroom traditional setting and calls for a more interactive methods.

## 1.3 Objectives

1. To develop a conceptual project that investigates the idea of fusing game design and programming teaching, and to provide a framework for guiding the development of such platform.
2. To ensure a smooth learning curve by starting with basics and gradually advancing and giving more difficult challenges.

3. To use challenging nature of both video games and programming to create a satisfying experience that ends with a sense of accomplishment.

## 1.4 Procedures

To achieve our project goals, we focused on developing a computer game that allows users to solve puzzles and proceed in levels by writing the proper code using a custom programming language made for our project.

To be able to do this, we started learning about interpreters and how to make a programming language in addition to studying some game design basics. Thus, we gathered the information we needed to start designing.

After gathering requirements, we analyzed and organized the information using UML (Unified Modeling Language) diagrams. These diagrams represented the structure and behavior of each functionality in the system (both game and interpreter), providing a blueprint for how they would be implemented and integrated.

Once the design phase was completed, our team transitioned into the development phase. For this, we adopted the agile approach to the Software Development Life Cycle (SDLC). The agile model is widely recognized in the software development industry, as it promotes flexibility and iterative development strategies. Our workflow progressed through cycles of planning, designing, coding, and testing specific sets of features or functionality.

By adopting the agile methodology, we were able to ensure regular communication and collaboration within our development team. This facilitated efficient decision-making, effective problem-solving, and continuous feedback loops.

Throughout the development process, we prioritized regular testing to verify the functionality, performance, and usability of the system. This iterative approach allowed us to identify and resolve issues early, ensuring a smooth and reliable user experience.

## 1.5 Organization of the Study

This thesis is divided into seven chapters each one describing a part of our project.

The chapters are as the following:

- Chapter 1: Introduction: This chapter outlines the idea and the main objective of the project.
- Chapter 2: Literature review: This chapter discusses similar previous projects.
- Chapter 3: System Requirements: This chapter discusses the System Requirements (functional and non-functional).
- Chapter 4: Methodology And Technologies: This chapter discusses the methodology we used during the construction of this project.
- Chapter 5: System Workflow and Challenges: This chapter shows detailed description of the system work and the challenges we faced.
- Chapter 6: Conclusion and Future Vision: This chapter reviews the conclusion and Future Vision for this project.
- Chapter 7: References.

# CHAPTER 2|LITERATURE REVIEW

*In this chapter, we will discuss similar previous systems*

## 2.1 Similar Works and Differences

Scratch [2.1] is a visual programming language and online community that allows users to create interactive stories, games and animations using its simple lego-style programming language. Scratch is similar to our project as it focuses on the same audience of young students and newcomers to programming. However, it does not offer the code-to-win concept or adding game elements to the teaching process.

CodinGame [2.2] is an online platform that offers coding puzzles and challenges in the form of games where users can solve programming problems using various languages and compete with other programmers. CodinGame is similar to our project as it offers the code-to-win concept and uses puzzles and levels to teach programming. However, it uses established programming languages which does not give the flexibility we want to achieve in our project. In addition to that, a big difference is that it lacks the real-time feedback and interactivity as it plays the winning scene if the code is true and the losing one if it is false, it does not provide enough game elements.

Codecademy [2.3] is an online learning platform that offers coding lessons in various programming languages where users learn by writing and running code directly in the browser. It is similar to our project with its use of level and progressive learning but it lacks the interactivity and game elements.

# CHAPTER 3|SYSTEM REQUIREMENTS

*This chapter discusses the System Requirements (functional and non-functional).*

## 3.1 Functional Requirements

### 3.1.1 Interactive Environment

The game should provide levels and environments that interacts at real-time with the player-written code showing them right and wrong results in visual environment.

This can be done by using the custom interpreter-based programming language by pausing the execution for an amount of time at specific lines to show the effect of the code at the objects in the environment based on the values stored in the memory.

### 3.1.2 Progression

The player's progression in game level should reflect his progression in learning. This can be done by starting with basic and simple puzzles, then introducing more concepts ensuring a smooth learning and difficulty increase curves.

Each level should use concepts of the level before it and add to them new ones to be learned and used.

### 3.1.3 Challenge Description

Inputs, outputs and the code player will write must all be described both textually and visually to ensure the requirements of the challenge and the concept it introduces are well understood by the player.

### 3.1.4 Providing Help

The player should have access to all concepts he learnt in previous levels at anytime, supported with additional examples and explanation.

### 3.1.5 Code Editor

The system must include an integrated code editor for players to write code inside the game.

In addition to that, the code editor should provide syntax highlighting and code refactoring running at real-time to get the best understanding of the written code.

## 3.2 Non-Functional Requirements

### 3.2.1 Usability

- The game's user interface should be user-friendly and consistent across all levels.
- The action that will be performed upon a key press depending on the context must be clear to the player and changes in real-time.
- The user interface must have minimal learning curve, using it in the first level is enough to get used to it.
- The game should give a feedback upon player's actions that gives him clear response of his action and the result of it.
- The user interface is adaptive and responsive, works seamlessly with different screen sizes, aspect ratios and resolutions.

### 3.2.2 Scalability

The system will be designed with scalability in mind, adding new challenges, levels, additional systems or replacing any of them should be done with no difficulties.

# CHAPTER 4|METHODOLOGY AND TECHNOLOGIES

*This chapter discusses the methodology we used during the construction of this project representing it through Software Analysis & Design.*

## 4.1 Methodology

Methodology means the method that will be used to build this system. In addition, the methodology is the most important part of the system development. In our project, we use System Development Life Cycle (SDLC). SDLC is a series of phases in the development process. It provides a model for the development and lifecycle of an application. The SDLC process will help to produce an effective, cost-efficient, and high-quality system.

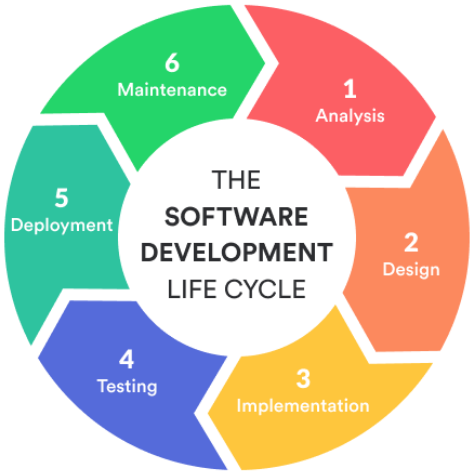


*Figure 4.1 System Development Life Cycle*

One of several types of SDLC is the Agile model and that is what we depended on when we started work on this project because requirements may vary based on the change of our understanding of the project requirements as long as we progress, using the agile technique will be easier because it is flexible and iteratively developing. The agile method can help in plan and scheduling the system development. The development

process moves from planning, development, testing then the feedback from testing is collected and fit back into the cycle.

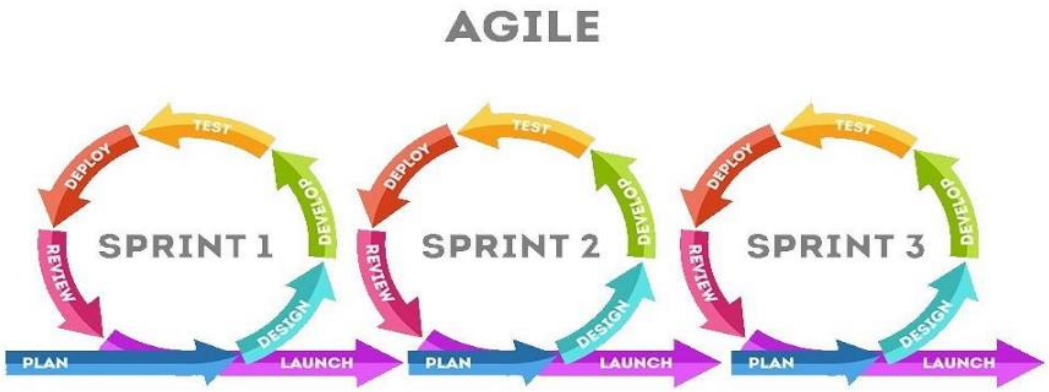


*Figure 4.2 Agile Model*

## 4.2 Technologies

### 4.2.1 Integrated Development Environment: Visual Studio

For the implementation of the system, we used Visual Studio [4.1] to write needed C# code.

*Figure 4.2.1 Visual Studio Logo*

Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs including websites, web apps, web services and mobile apps.

### 4.2.2 Programming Language: C#

*Figure 4.2.2 C# Logo*

- C# [4.2] is a simple, modern, general-purpose, object-oriented programming language.
- C# applications are intended to be economical with regard to memory and processing power requirements.
- C# is the language used to write scripts in the Unity engine.

### 4.2.3 Unity 3D

Unity 3D [4.3] is a powerful and widely used game development. With Unity 3D, we can create immersive and interactive experiences by combining 3D graphics, animations, physics, and more.


*Figure 4.2.3 Unity 3D Logo*

Here's an overview of how Unity works:

- Scene Setup: Developers begin by setting up their Unity project and scene. They can import 3D models, textures, and other assets into the project, and arrange them within the scene to create the desired environment for the level.

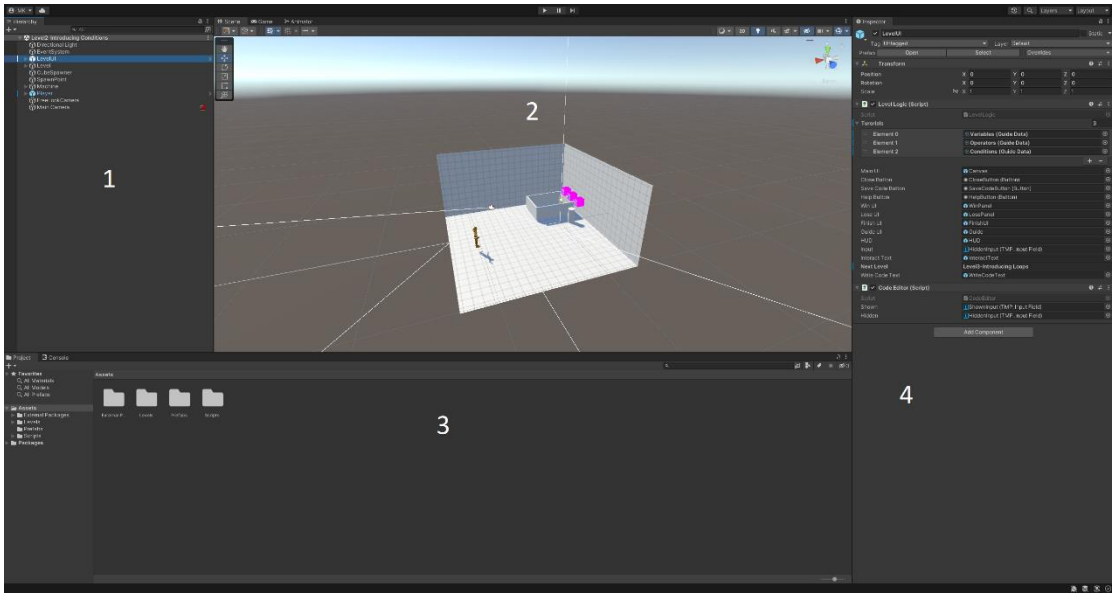


*Figure 4.2.3.1 Unity Editor*

1. Hierarchy: this window shows the scene name as the root and objects in the current scene as child nodes. Each one of these objects is called a GameObject and has at least the basic component Transform which describes the location, rotation and scale of the object in the scene.
2. Scene: the Scene view is where you visualize and interact with the world you create in the Editor. In the Scene view, you can select, manipulate, and modify GameObjects that act as scenery, characters, cameras, lights, and more.
3. Project Window: this is the window that shows all files in the project, with the main folder called "Assets". Project files and folders can be organized by the developer to access them easily.

4. Inspector: this window shows the components of selected object in Scene or Hierarchy. Components vary from ones included in unity by default and some written by user like C# scripts.

- Unity Components:

  We will mention some Unity Components that we used in our project:

  1. Mesh Renderer and Mesh Filter: these components work together to render 3D objects.

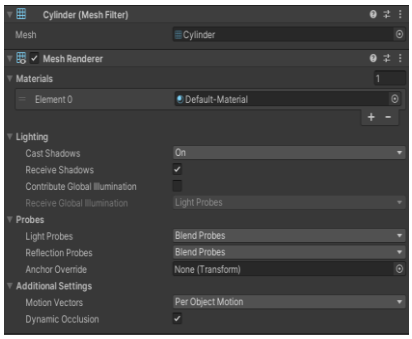


*Figure 4.2.3.2 – Mesh Renderer and Filter of a cylinder shaped object*

  2. Colliders: component that defines the shape of an object for the purposes of physical collisions. A collider, which is invisible, need not be the exact same shape as the object's mesh.

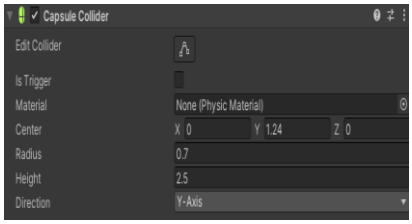


*Figure 4.2.3.3 – Collider Component of a capsule shaped object*

3. Rigidbody: the main component that enables physical behavior for a GameObject. With a Rigidbody attached, the object will immediately respond to gravity.

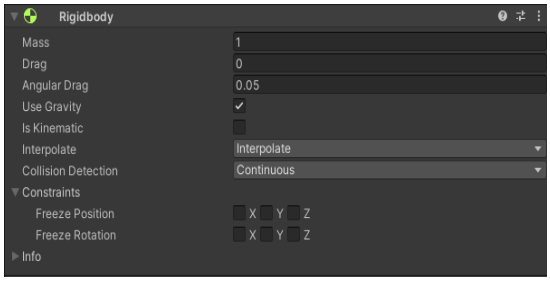


*Figure 4.2.3.4 – Rigidbody Component*

4. UI Components: like texts, input fields and buttons. A UI Component's parent must be a Canvas which is the container of UI Objects and can be set to either show UI elements in a Screen Space or World Space.

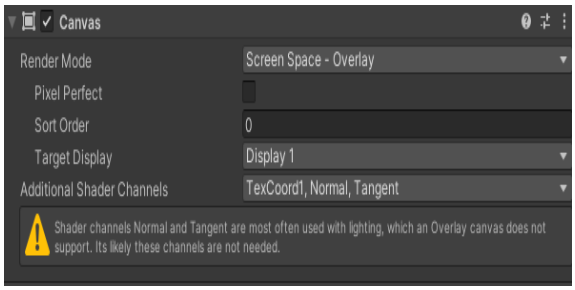


*Figure 4.2.3.5 – Canvas Component set to render in Screen Space*

5. C# Scripts: these are components that we implement and add to objects.

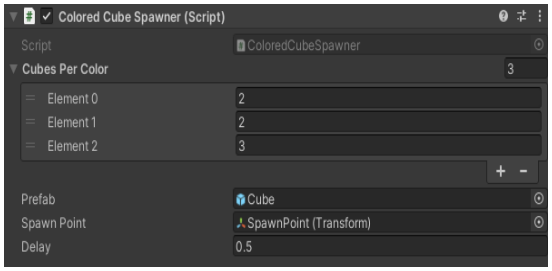


*Figure 4.2.3.6 – C# Script Component – Colored Cube Spawner*

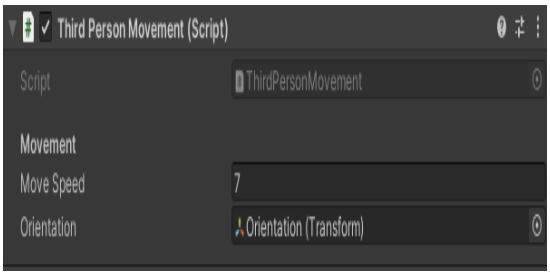


*Figure 4.2.3.7 – C# Script Component – Player Movement*

# 4.3 Software Analysis and Design

## 4.3.1 Class Diagrams

### 4.3.1.1 Interpreter Diagrams

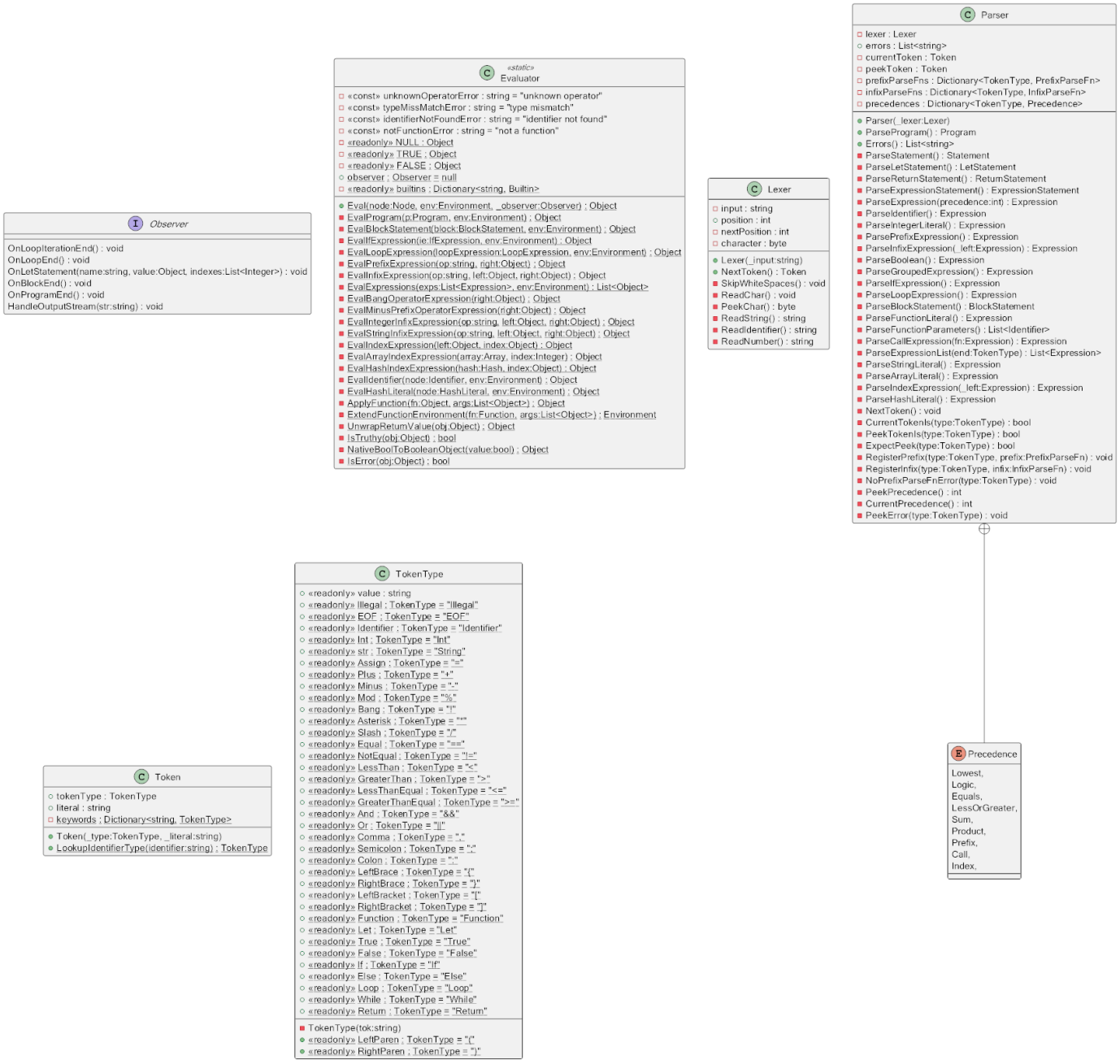


*Figure 4.3.1.1 – Class Diagram – Lexer, Parser, Evaluator, Token and Observer*

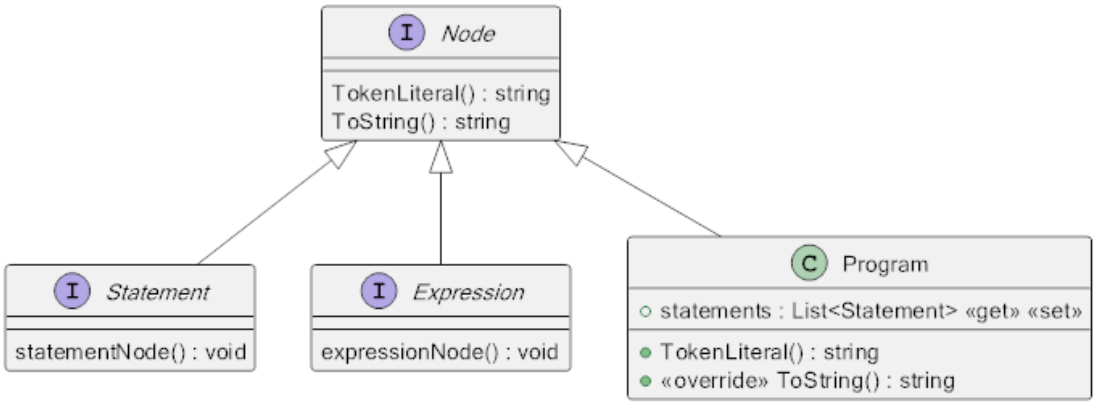


*Figure 4.3.1.2 – Class Diagram – Node, Program*

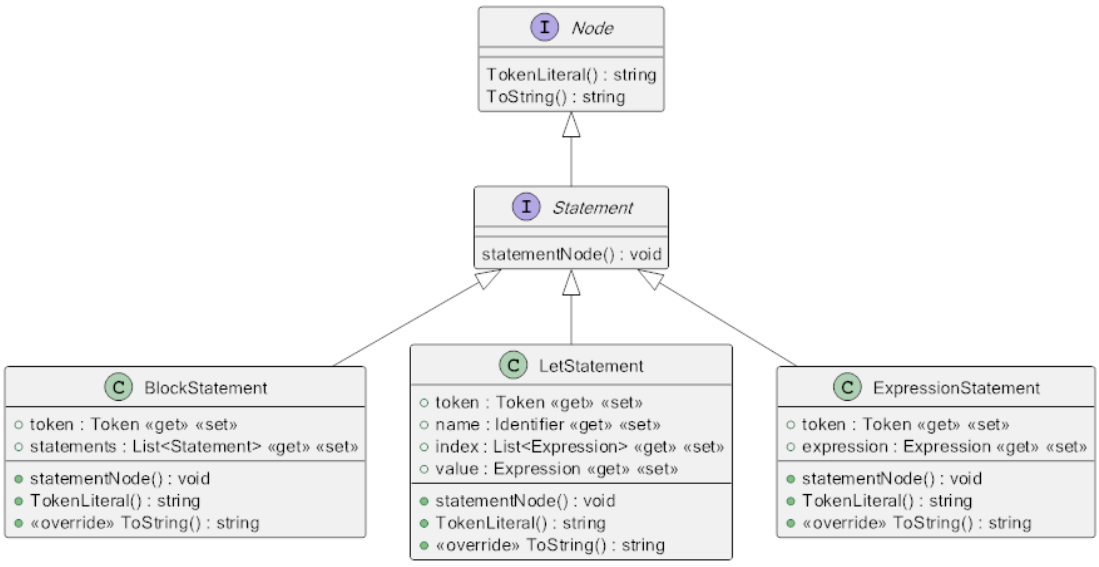


*Figure 4.3.1.3 – Class Diagram – Statements*

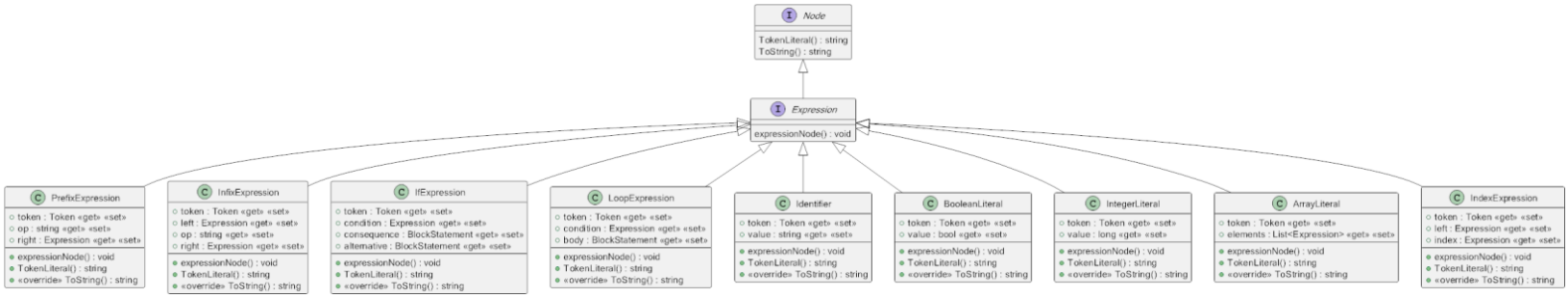


*Figure 4.3.1.4 – Class Diagram – Expressions*

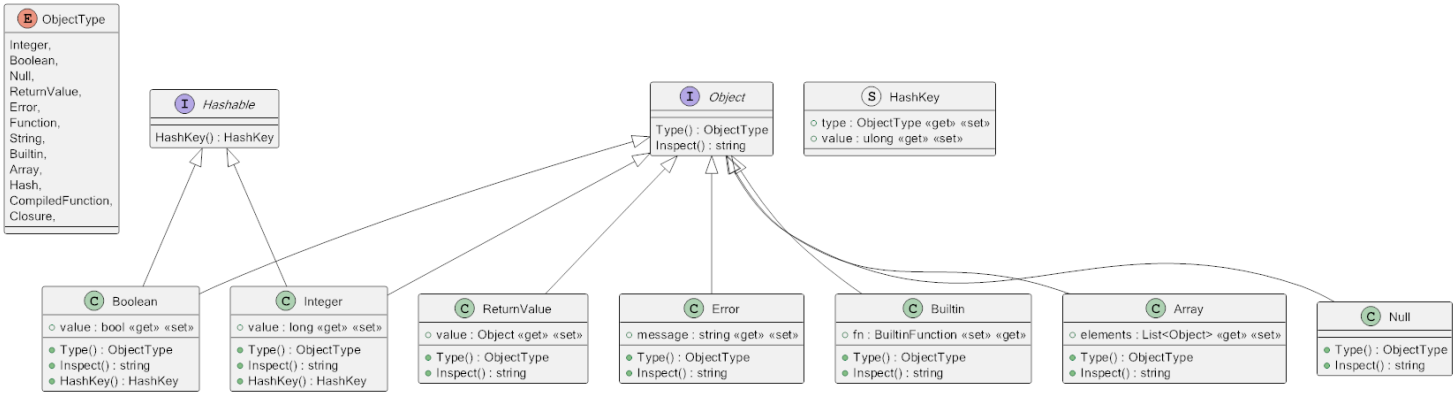


*Figure 4.3.1.5 – Class Diagram – Objects*

### 4.3.1.2 Game Diagrams – Unity Classes

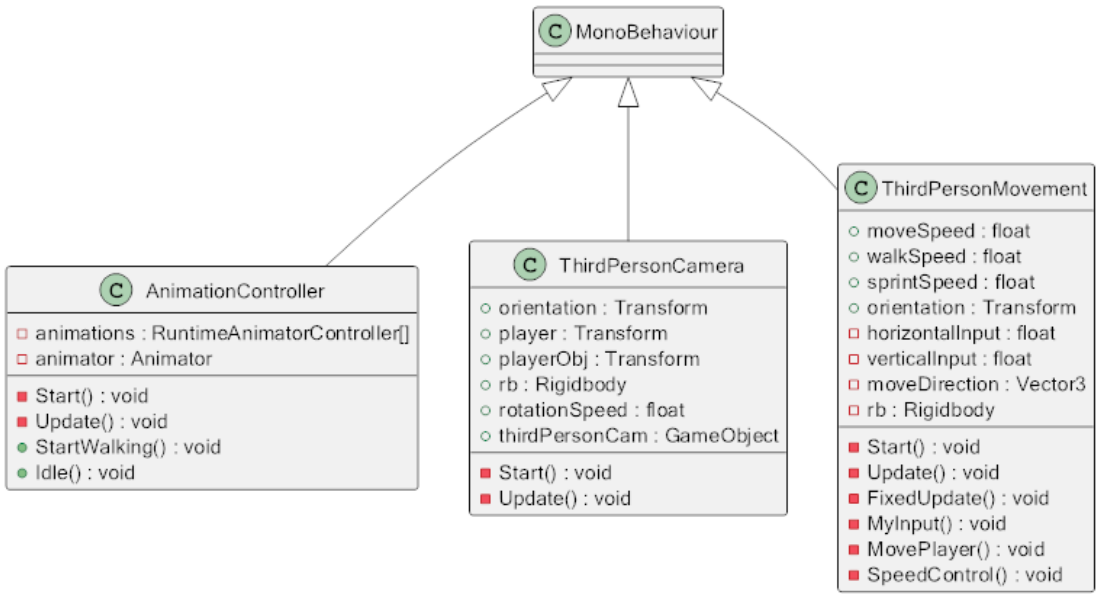


*Figure 4.3.1.6 – Class Diagram – Player Character Related Classes*

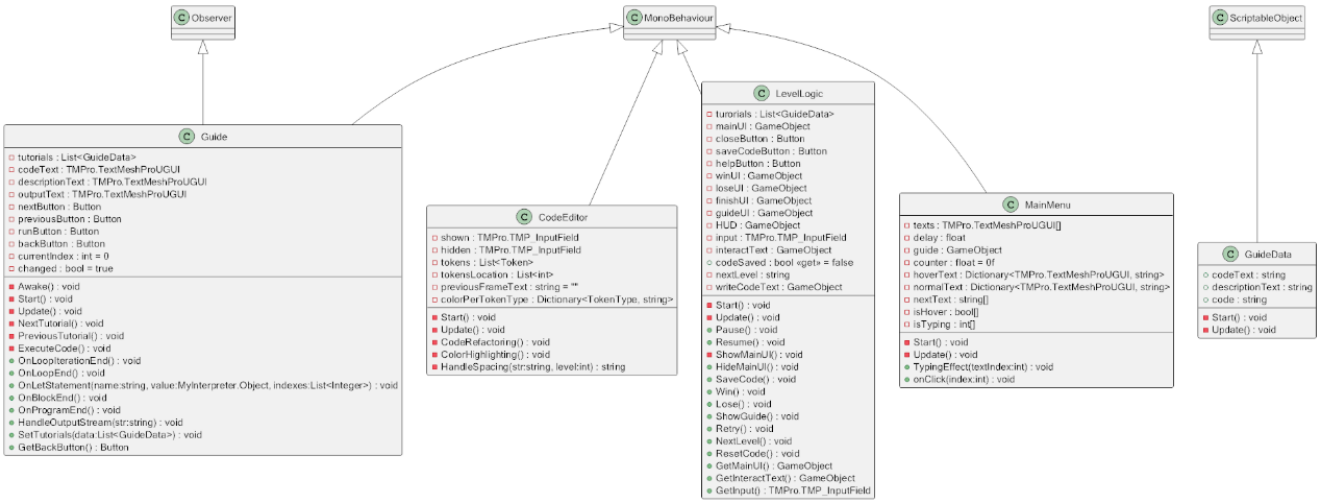


*Figure 4.3.1.7 – Class Diagram – Workflow and UI Classes*

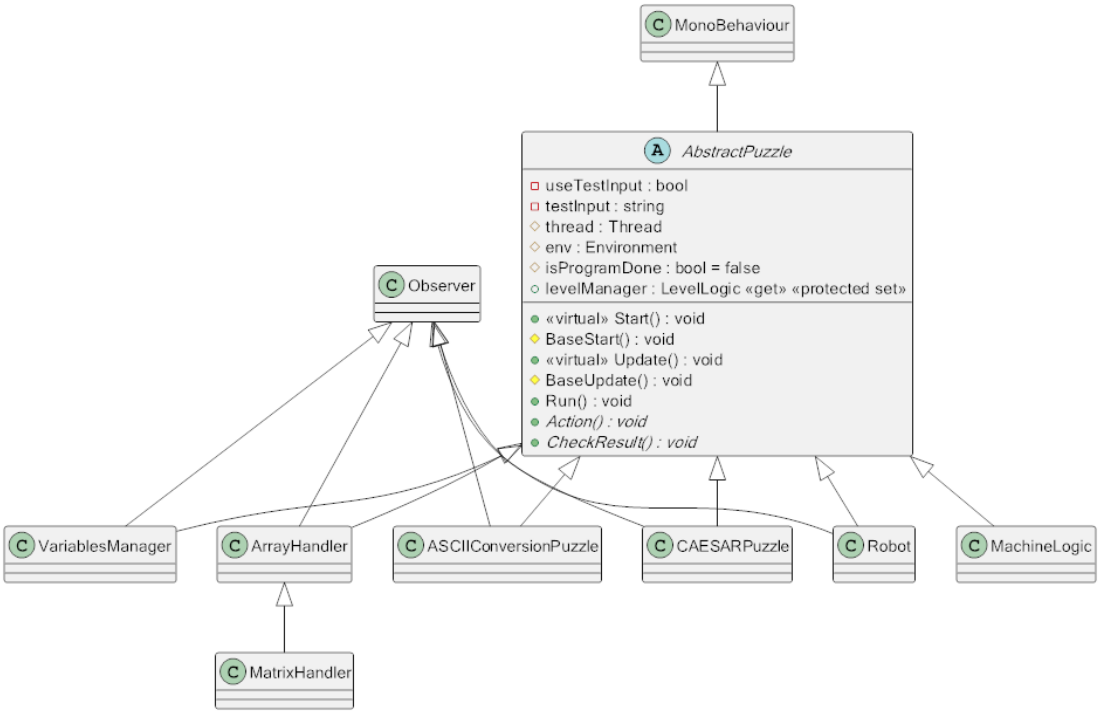


*Figure 4.3.1.8 – Class Diagram – Puzzles\Levels Classes*

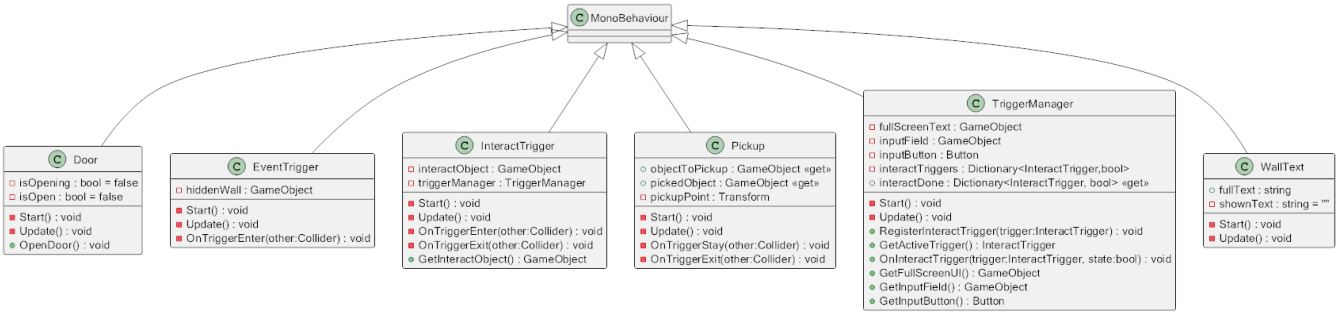


*Figure 4.3.1.9 – Class Diagram – Player Interaction Related Classes*

## 4.3.2 Use Case Diagram

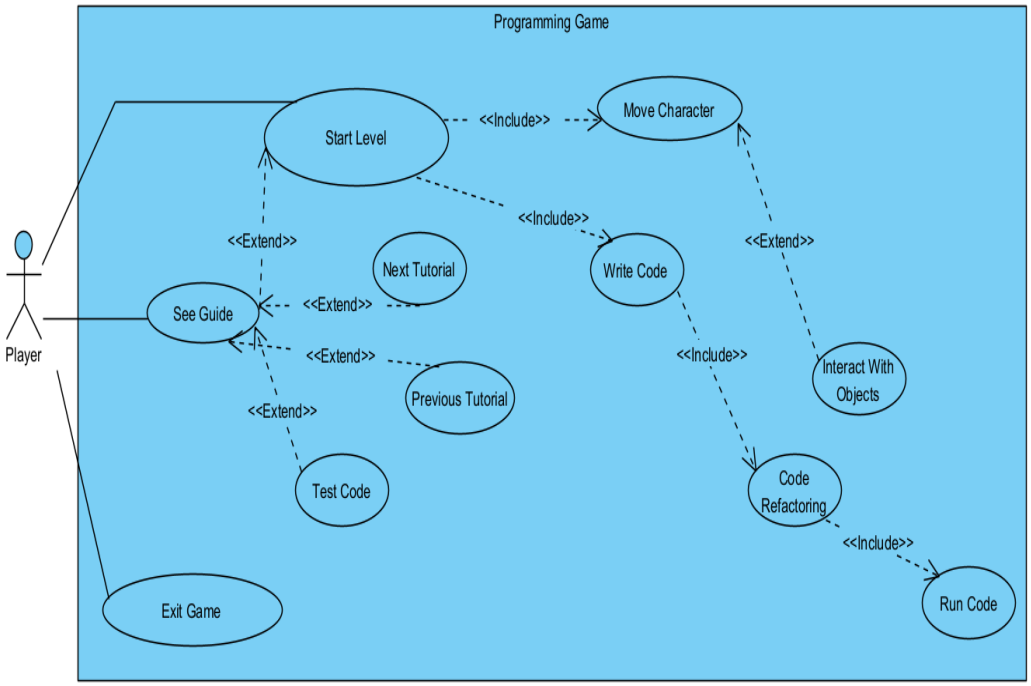


*Figure 4.3.2 – Use Case Diagram*

### 4.3.3 Use Case Description

Here we describe the workflow for some use cases that the users interact with.

<table>
<tr><td>Use case name:</td><td colspan="2">Start Level</td></tr>
<tr><td>Scenario:</td><td colspan="2">Starting new level.</td></tr>
<tr><td>Triggering event:</td><td colspan="2">Player enters new level.</td></tr>
<tr><td>Brief description:</td><td colspan="2">Player enters level by completing the one before it.<br>Player should read the description of problem and explore level to write the right code.</td></tr>
<tr><td>Actors:</td><td colspan="2">Player.</td></tr>
<tr><td>Related use cases:</td><td colspan="2">Move Character and Write Code.</td></tr>
<tr><td>Preconditions:</td><td colspan="2">Previous level done.</td></tr>
<tr><td>Postconditions:</td><td colspan="2">Player must have been written the right answer or a wrong one.<br>If the answer is right, next level will start.<br>If the answer is wrong, same level will restart.</td></tr>
<tr><td>Flow of activities:</td><td>Actor</td><td>System</td></tr>
<tr><td></td><td>1. Player starts new level</td><td></td></tr>
<tr><td></td><td></td><td>1.1 System sets up the scene with its objects.<br>1.2 System gives player control.</td></tr>
<tr><td></td><td>2. Player writes Code</td><td></td></tr>
<tr><td></td><td></td><td>2.1 System starts refactoring and highlighting code.</td></tr>
<tr><td></td><td>3. Player runs the code</td><td></td></tr>
<tr><td></td><td></td><td>3.1 System executes the code.<br>3.2 System checks for results of execution.<br>3.3 System starts level depending on the result.</td></tr>
</table>

*Table 4.3.3.1 Use Case Description – Start Level*

<table>
<tr><td>Use case name:</td><td>Interacting with Objects</td></tr>
<tr><td>Scenario:</td><td>Player character interacts with an intractable object.</td></tr>
<tr><td>Triggering event:</td><td>Player character enters interact area.</td></tr>
<tr><td>Brief description:</td><td>Player enters interact area.<br>Interact trigger activates and instructions are shown on screen.<br>Waiting for player input to interact.</td></tr>
<tr><td>Actors:</td><td>Player.</td></tr>
<tr><td>Related use cases:</td><td>None.</td></tr>
<tr><td>Preconditions:</td><td>Interact trigger active</td></tr>
<tr><td>Postconditions:</td><td>Action will be done dependent on the intractable object.</td></tr>
<tr><td>Flow of activities:</td><td>
<table>
<tr><th>Actor</th><th>System</th></tr>
<tr><td>1. Player enters interact area<br><br>2. Player input</td><td>1.1 System informs player he is ready to interact.<br>1.2 System waits for player input.<br><br>2.2 System performs action dependent on intractable object.</td></tr>
</table>
</td></tr>
</table>

*Table 4.3.3.2 Use Case Description – Interacting with Objects*

<table>
<tr><td>Use case name:</td><td>See Guide</td></tr>
<tr><td>Scenario:</td><td>Player browses guide and tutorials inside game.</td></tr>
<tr><td>Triggering event:</td><td>Player pressing on Help/Guide button.</td></tr>
<tr><td>Brief description:</td><td>Player can access guide and tutorials at the main menu before playing or in the middle of a level by pausing it.</td></tr>
<tr><td>Actors:</td><td>Player.</td></tr>
<tr><td>Related use cases:</td><td>None.</td></tr>
<tr><td>Preconditions:</td><td>Player can access the guide before writing his code.</td></tr>
<tr><td>Postconditions:</td><td>Player gets access to guide and examples useful at the current level.<br>Player can execute these example codes and see the output.<br>Player can exit the guide and continue the level.</td></tr>
<tr><td>Flow of activities:</td><td>
<table>
<tr><th>Actor</th><th>System</th></tr>
<tr><td>1. Player presses Guide button.</td><td></td></tr>
<tr><td></td><td>1.1 System shows guide UI where player can access different examples related to the current level.</td></tr>
<tr><td>2. Player presses Test Code button.</td><td></td></tr>
<tr><td></td><td>2.1 System executes example code and prints its output.</td></tr>
<tr><td>3. Player presses Back button.</td><td></td></tr>
<tr><td></td><td>3.1 System closes guide UI.<br>3.2 System returns player to the main UI to write code.</td></tr>
</table>
</td></tr>
</table>

*Table 4.3.3.2 Use Case Description – See Guide*

### 4.3.4 Activity Diagrams

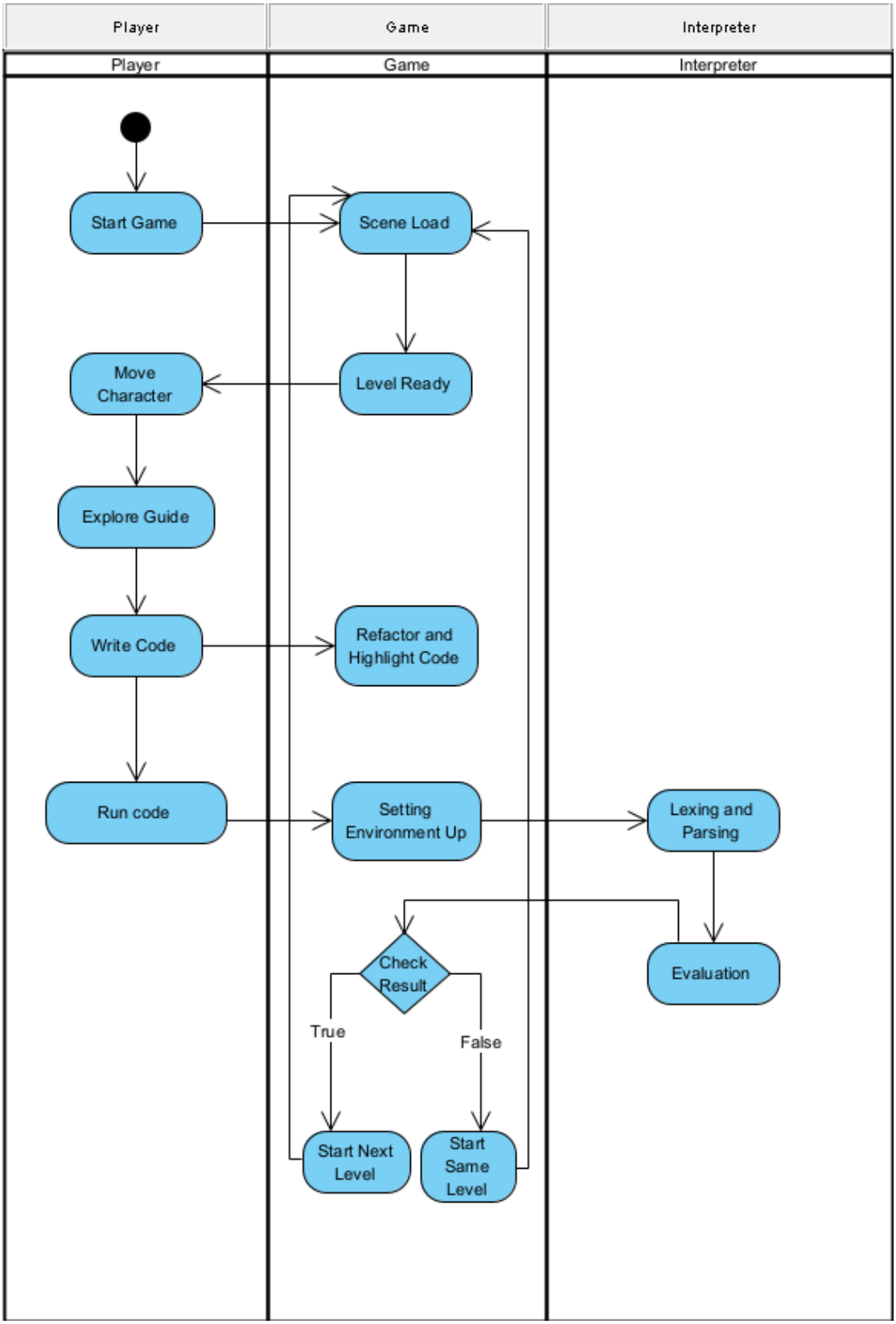


*Figure 4.3.4.1 – Activity Diagram – Solving a Puzzle*

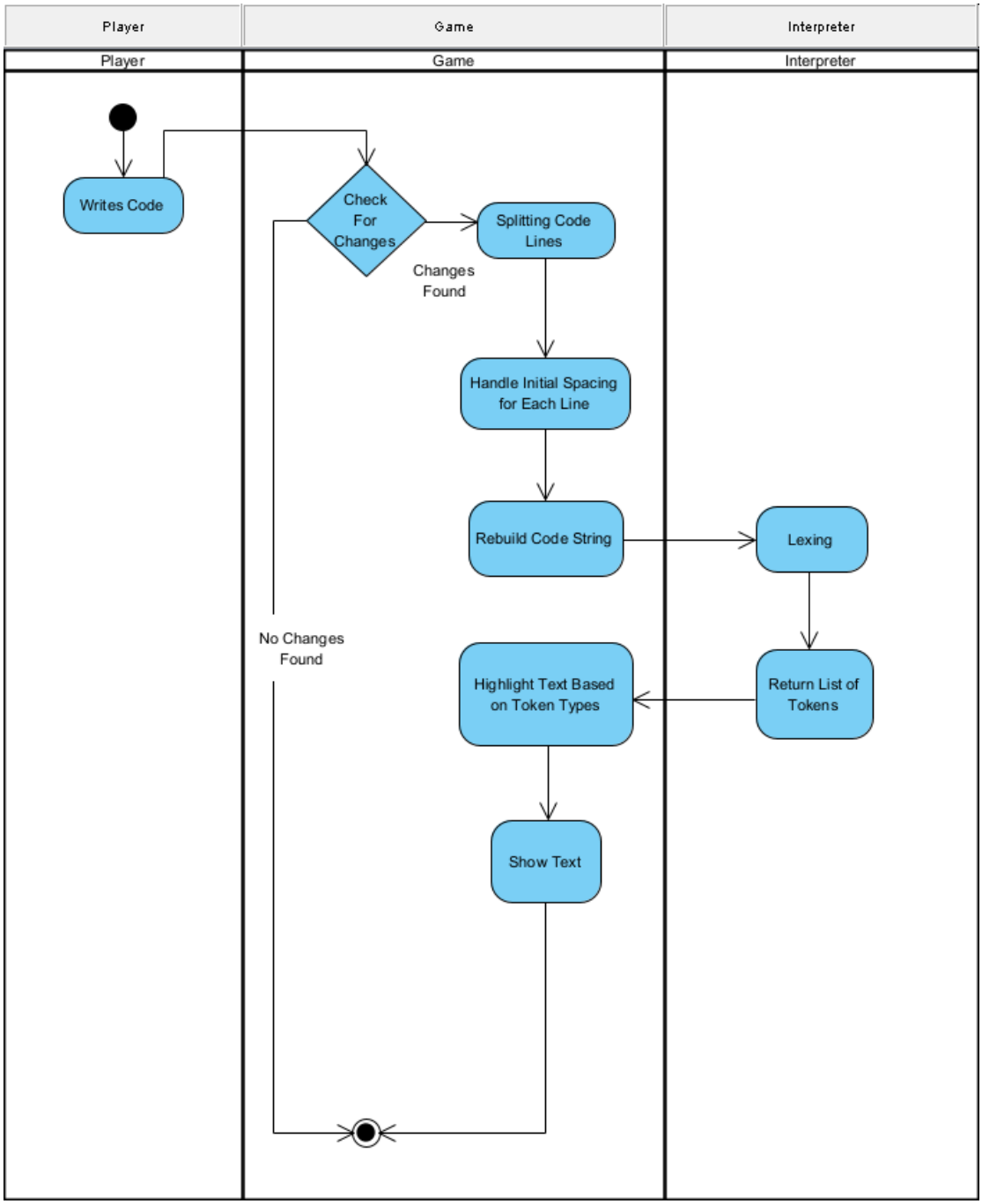


*Figure 4.3.4.2 – Activity Diagram – Code Refactoring and Highlighting*

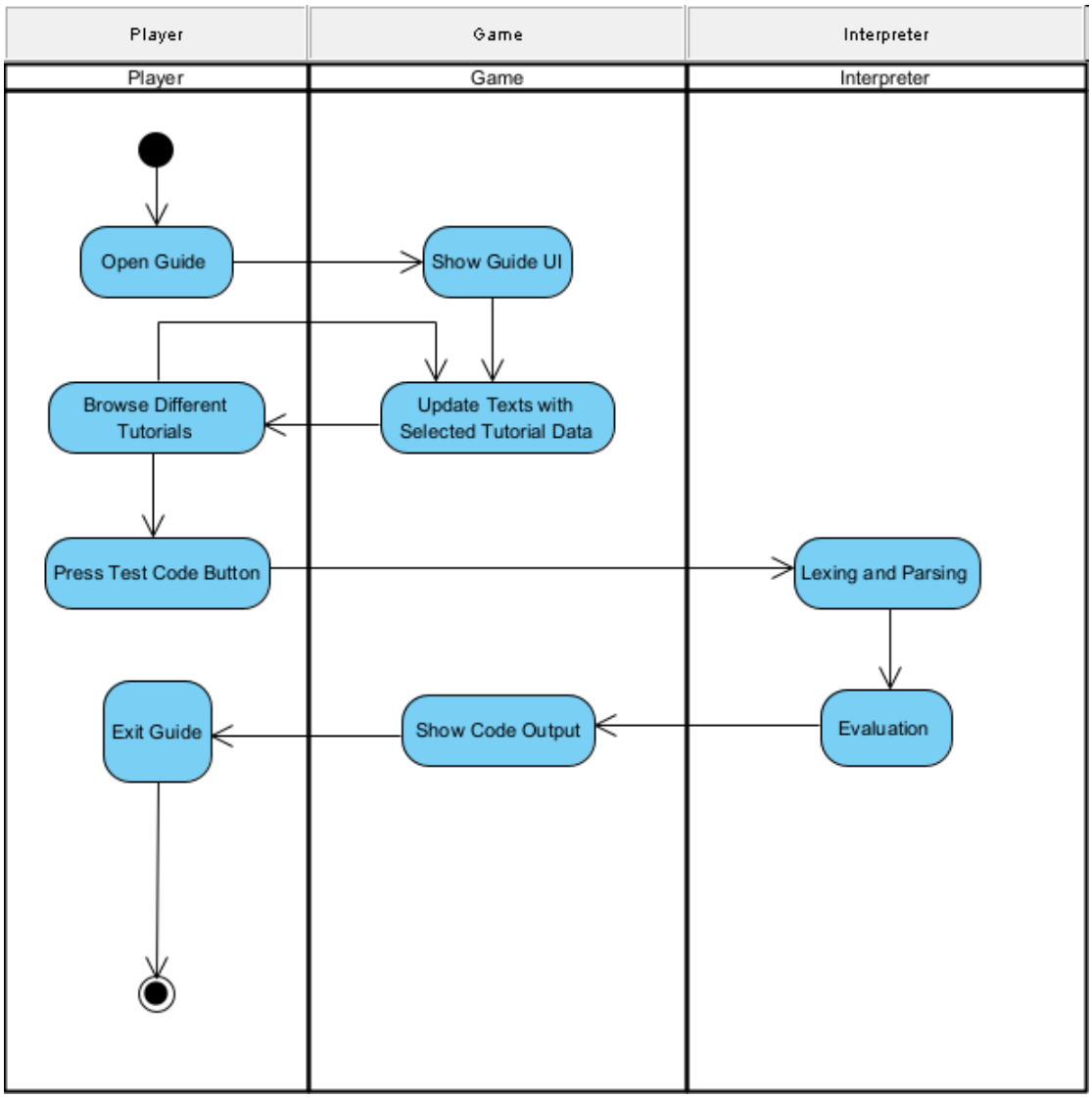


*Figure 4.3.4.3 – Activity Diagram – Browsing the Guide*

## 4.3.5 System Sequence Diagrams

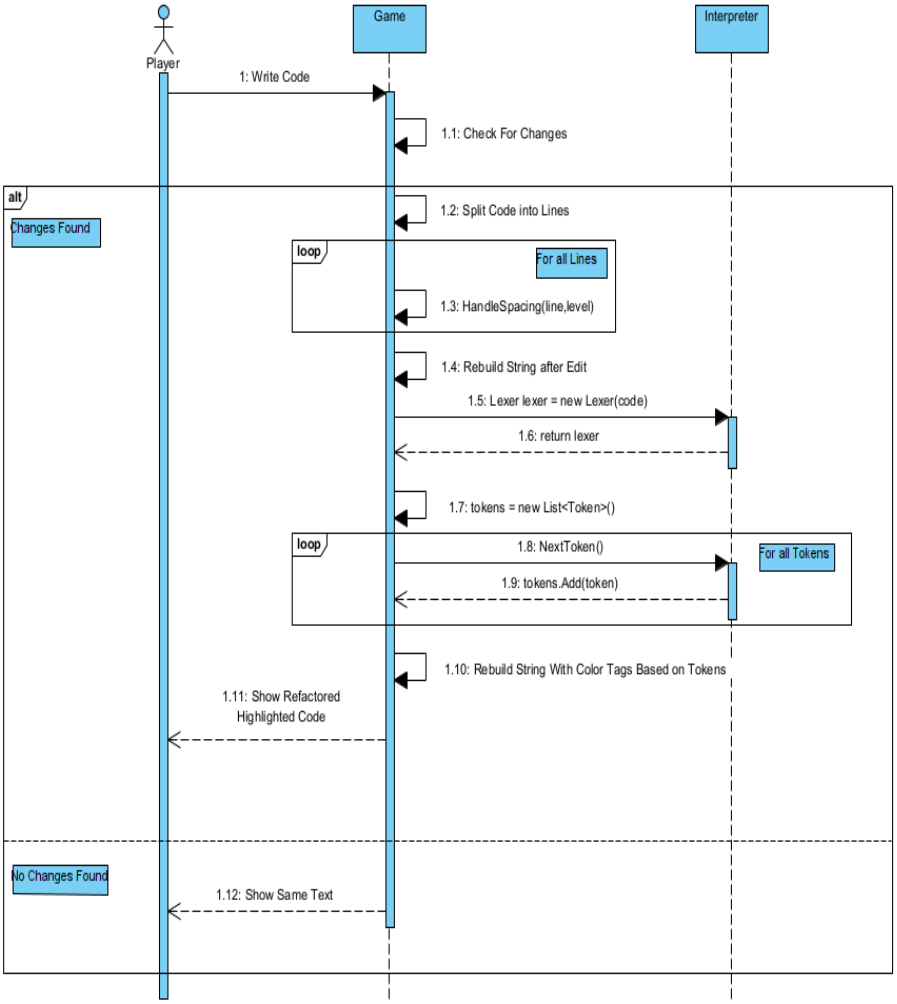


*Figure 4.3.5.1 – System Sequence Diagram – Refactoring and Highlighting Code*

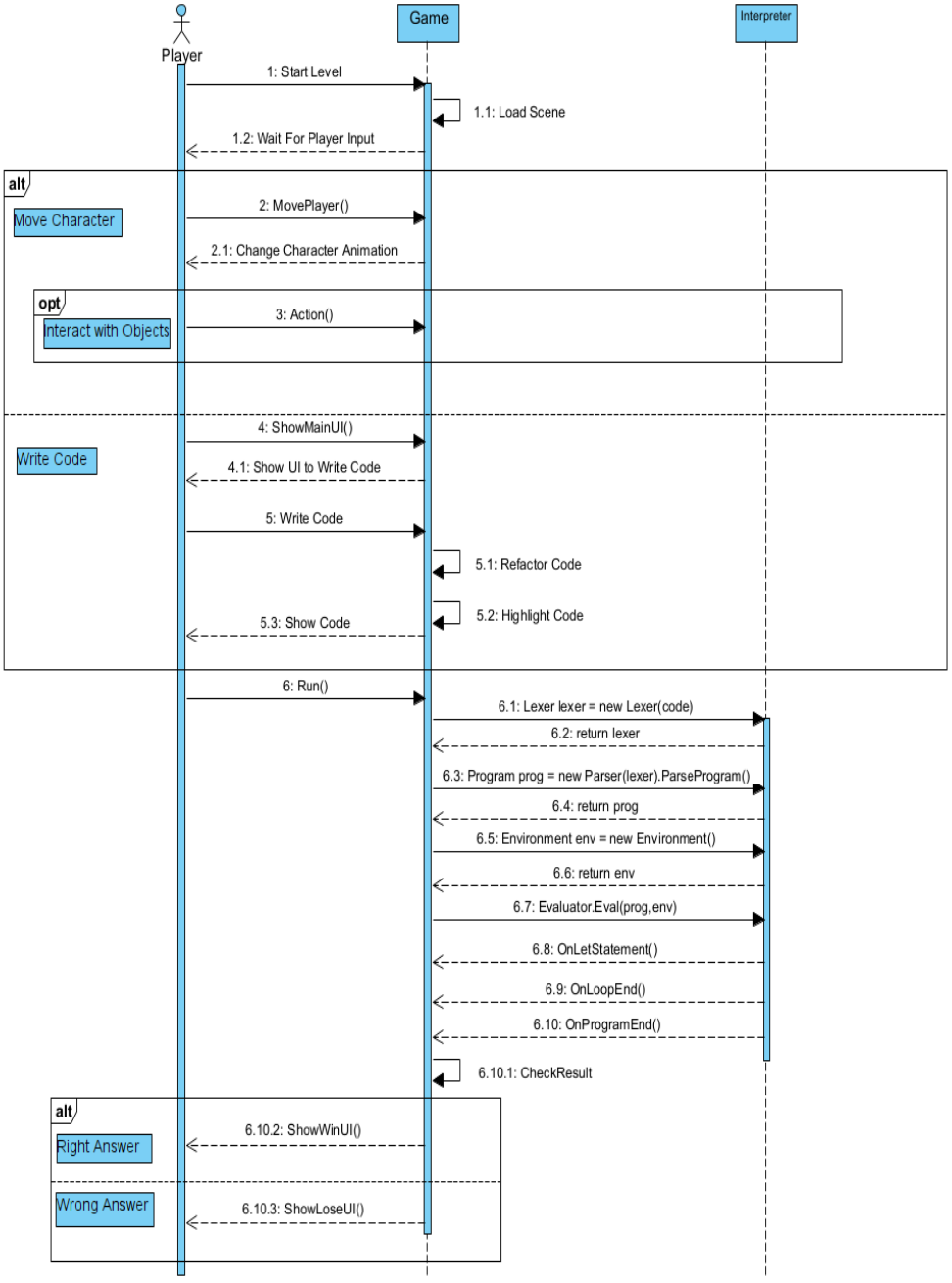


*Figure 4.3.5.2 – System Sequence Diagram – Solving a Puzzle*

# CHAPTER 5|SYSTEM WORKFLOW AND CHALLENGES

*This chapter shows detailed description of the system work and the challenges we faced.*

## 5.1 What is an Interpreter?

An interpreter is a computer program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.

Using an interpreter, a program goes through three main stages:

1. Lexer: The lexer is responsible for breaking the source code into a stream of tokens. Tokens are the smallest units of meaning in a programming language, such as keywords, identifiers, operators, and literals.
2. Parser: The parser takes the stream of tokens generated by the lexer and organizes them into a hierarchical structure, a syntax tree.
3. Evaluator: It traverses the tree and executes the corresponding actions to produce the final result of the program. The evaluator enforces the meaning of the code and handles tasks such as variable assignments, function calls, and other language-specific operations.

## 5.2 Why not using an established programming language?

- Making our own interpreter will give us the flexibility we need in creating interactive environments.
- Since we have the access to the 3 components of the interpreter: Lexer, Parser and Evaluator, we will be able to control all steps of code execution. This will help us do things we can't if we used an established language like:
    - Check if the code contains specific statements.
    - Dealing with different statements in different ways.
    - Making delays at specific parts of code for the player to see their effect in-game.
    - Managing variables stored in memory without changing code written by player.

## 5.3 Challenges

- To create an interpreter, we followed the book "Writing an Interpreter in Go" By Thorsten Ball. As the name implies, the book uses Go language to craft its interpreter, but we want it in C# which has differences in syntax, so it was a bit bothersome to convert the code to C#.
- Creating interactive puzzles was not an easy job, finding a puzzle that suits a programming concept and applying it took many tries for each new level.
- Lack of knowledge about this new field, the game design. We had to learn a little about game design and how video games work.
- Programming a video game is different, where we don't only wait for player input, but also we must think about every frame, where 200+ frames are generated per second and some code is being executed every frame.
- Another different aspect in programming we dealt with is multithreading which we used to pause code and resume it at specific statements. Unity restricts access for some methods to the main thread only, which forced us to divide working between threads and think of synchronization and transitions between them.

## 5.4 Detailed Example

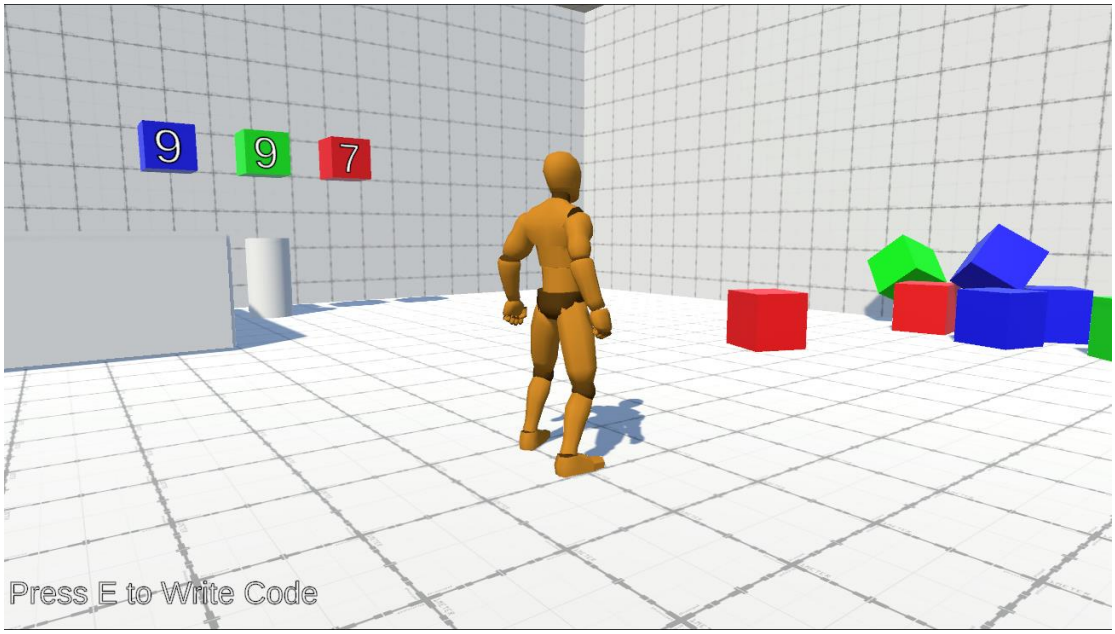


*Figure 5.4.1 – Slicing Cubes Puzzle*

For this example, we will show the second level where the player is being introduced to the "Conditions" concept.

The level starts and the player sees cubes at his right another numbered ones in front of him. On-screen instructions tells the player to press 'E' key to write his code.

This is the visual description part of the level.

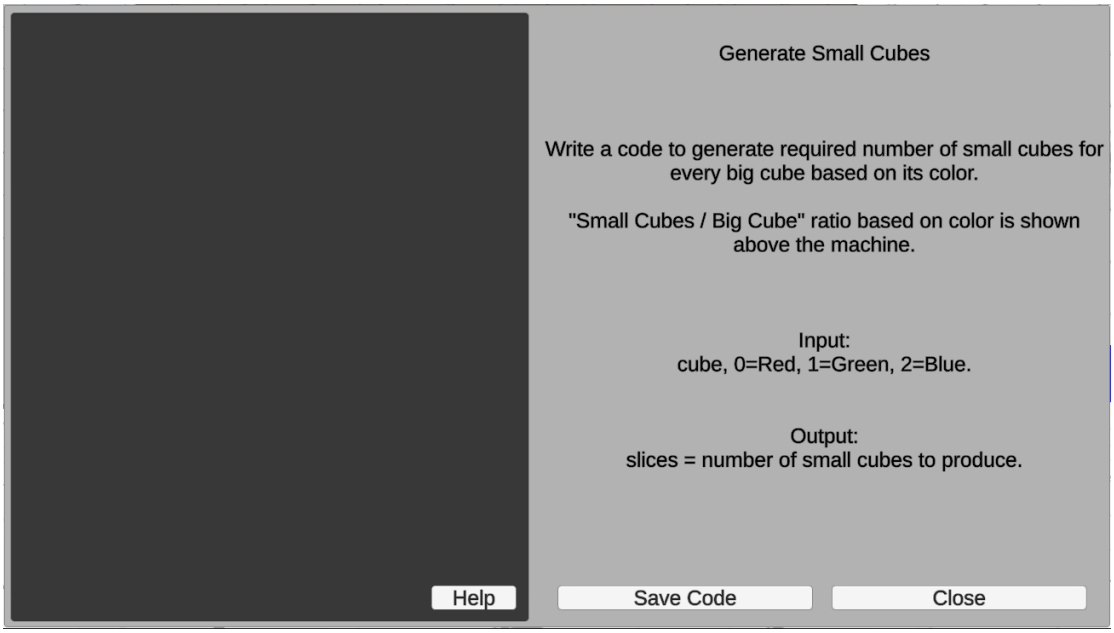


*Figure 5.4.2 – Write Code UI*

By pressing 'E' key, this UI shows where the player can read the textual description of the problem, use it along with the visual description in the player-view to write the code which solves the Problem.

Pressing "Close" button, the game will return to the player-view.

Pressing "Save Code" button, the game will save the code written by player and return to the player-view.

The player can access guide and examples by pressing the "Help" button.

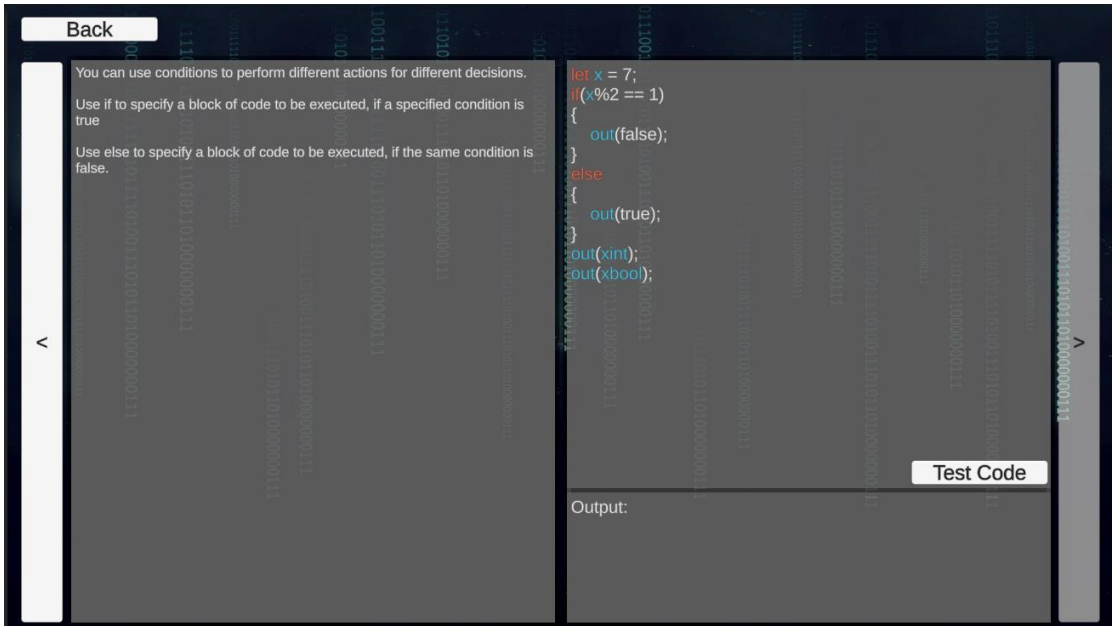


*Figure 5.4.3 – Guide UI*

Pressing "Test Code" button, output of the code will be shown at the output section.

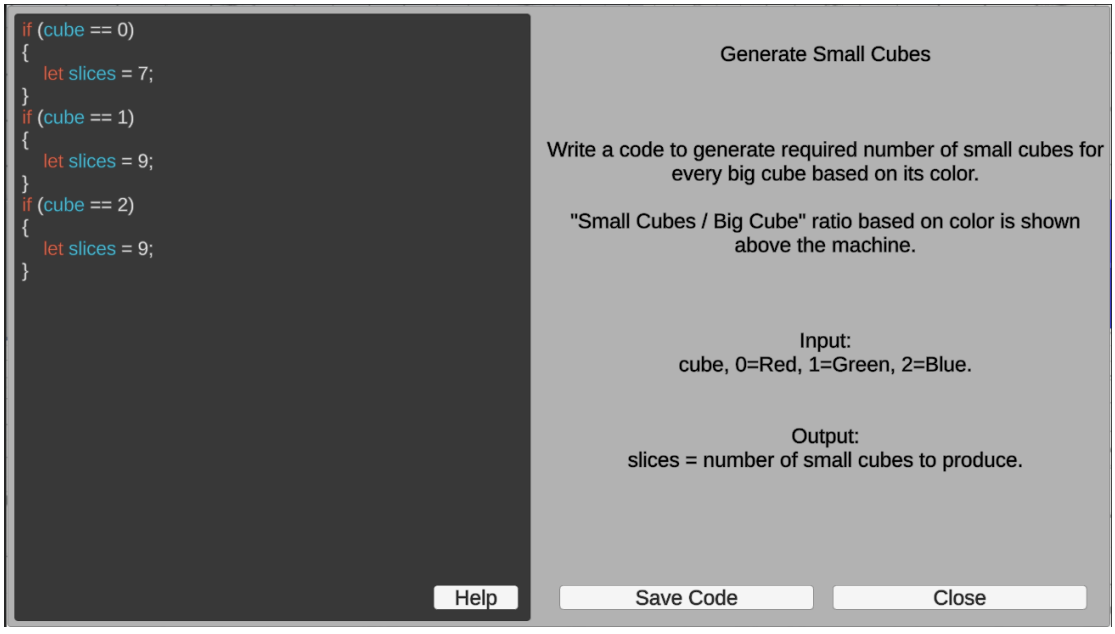


*Figure 5.4.4 – Solver Code*

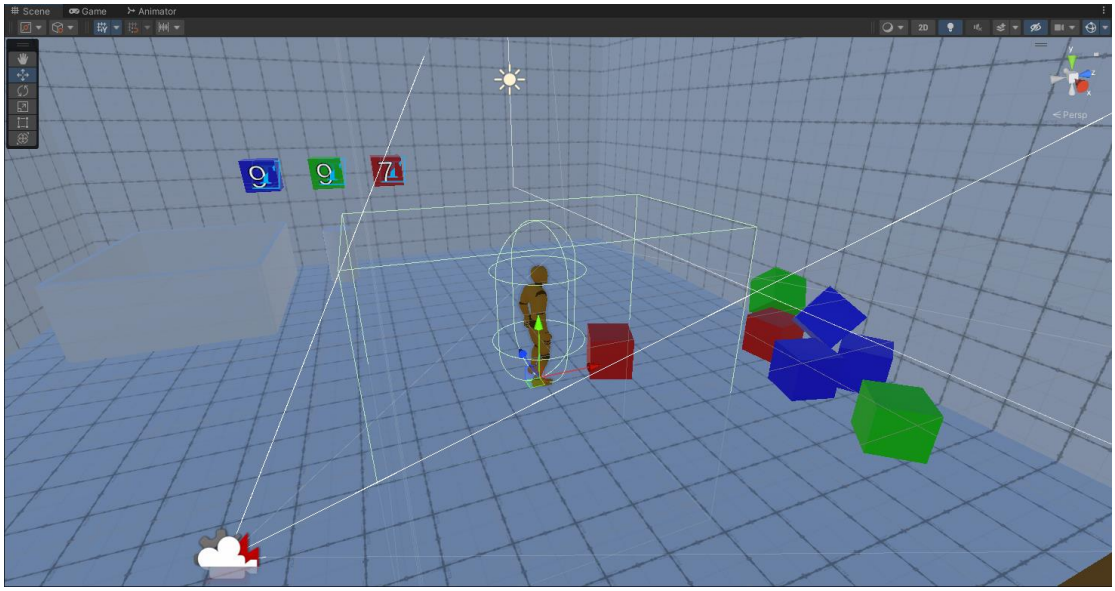


*Figure 5.4.5 – In-Engine View of Triggers*

This screen is in-engine where debugging lines are visible. The box-shaped area around the player is called a trigger, in this case the trigger checks for objects that can be picked by the player, if the object is in the area, player can pick it by pressing the interact key 'F'.

The capsule-shaped area around the player is a collider which handles player body's collisions with other objects like the ground and prevents the body from passing through it.

The player and the colored cubes has rigidbody component which applies gravity and physics on them.

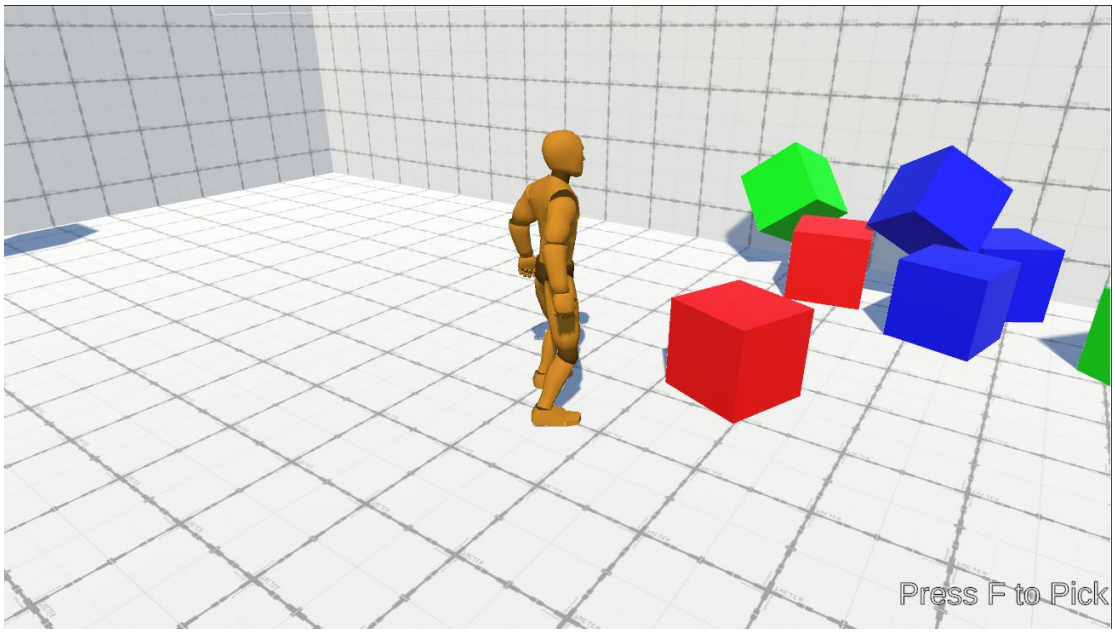


*Figure 5.4.6 – Player Ready to Pick a Cube*

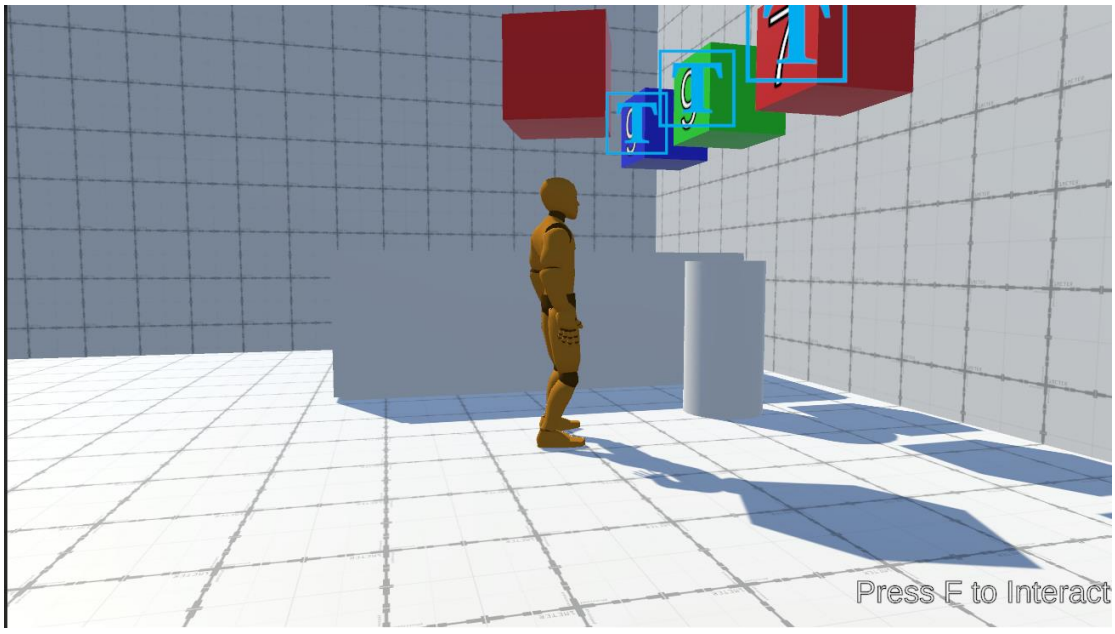


*Figure 5.4.7 – Player Picking a Cube*

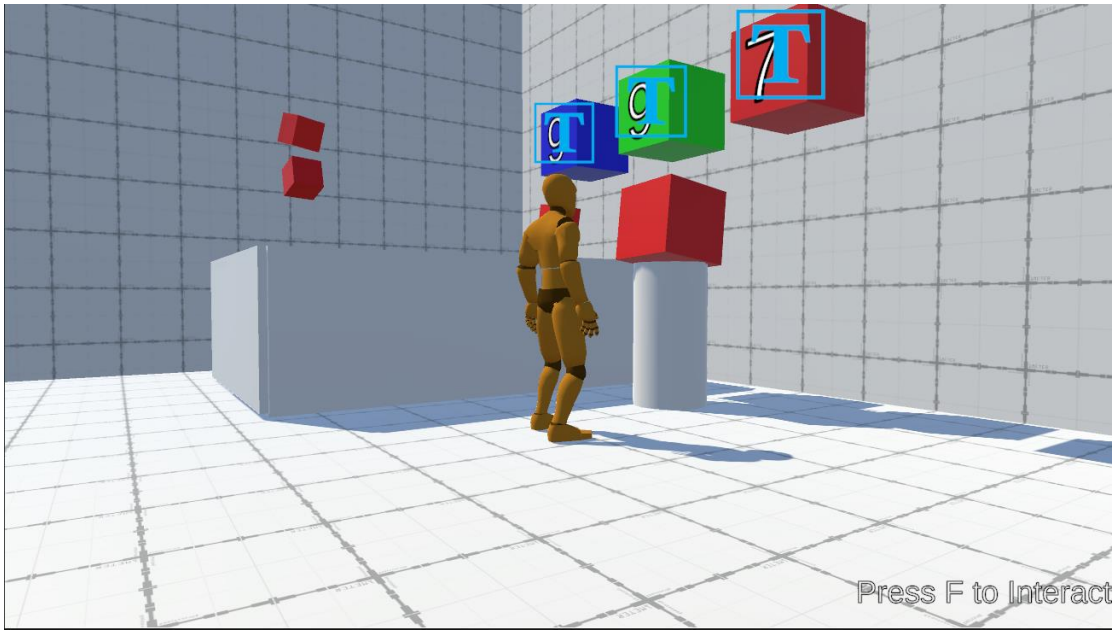


*Figure 5.4.8 – Code Executing*

By using same components like triggers and our C# scripts, when the cube is put in the machine area, it detects it and starts executing the written code, as seen, small cubes start to generate depending on the written code.

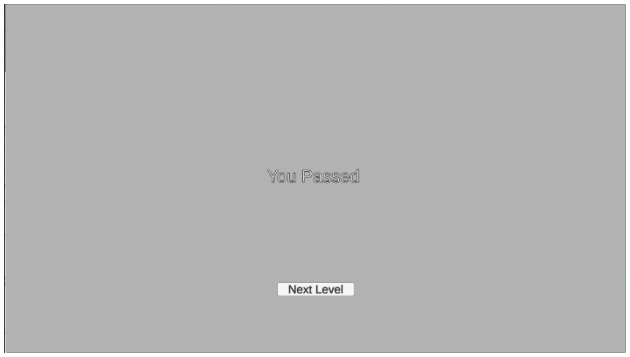


*Figure 5.4.9.A – Win UI*


*Figure 5.4.9.B – Lose UI*

Finishing all cubes, will call the CheckResult method which shows one of above screens based on the result.

Thus, we raise the difficulty, the complexity of level design and adding new concepts each level. When the player reaches the last level numbered 6, he will solve 2 puzzles in one level which will be like a test for the concepts he learnt in previous levels as long as these puzzles uses all of them and has a cryptography concept in a maze level.

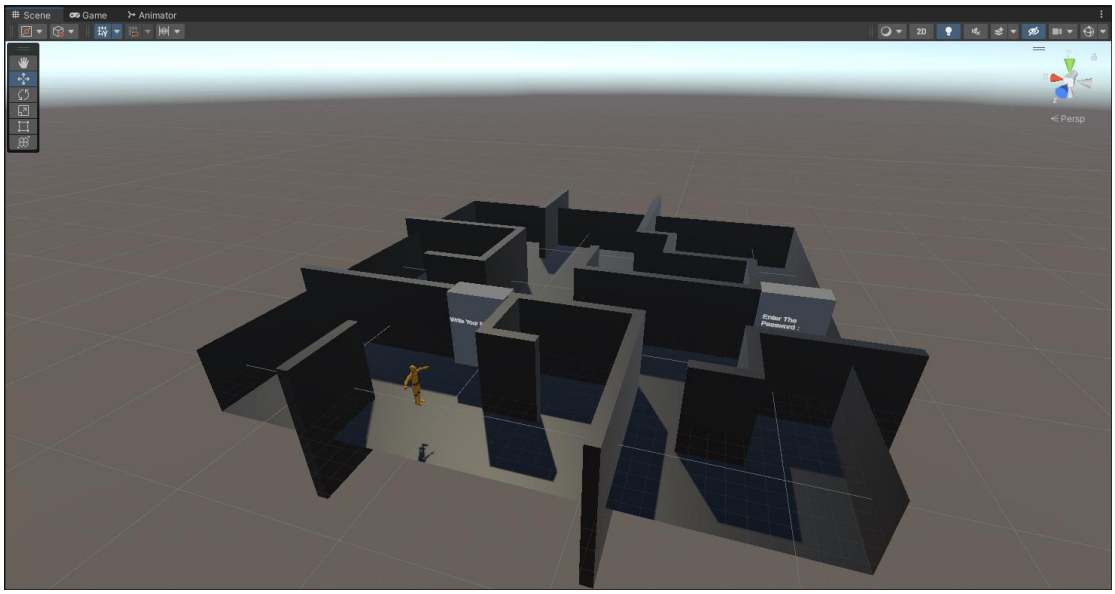


*Figure 5.4.10 – In-Engine View of the Last Level*

# CHAPTER 6|CONCLUSION & FUTURE VISION

*This chapter reviews the conclusion and Future Vision for this project.*

## 6.1 Conclusion

In conclusion, our project strives to bridge the gap between traditional education models and the expectations of modern learners. It envisions a dynamic, interactive and enjoyable learning environment where individuals of diverse backgrounds and learning styles can actively participate in the programming education journey. While the project remains conceptual at this stage, we see that we succeeded in building the basis that will have an impact on reshaping programming education. Future iterations will refine and validate the concept, opening the way for a more engaging experiences.

## 6.2 Future Vision

1. Leveraging more game design concepts like: multiplayer cooperation and competition, scores based on the efficiency of the written code and the speed of writing, more complex level design and using aesthetics like: stories, worlds design and arts to create more connection between players and the game.
2. Using the basis we got of the project to build an online problem solving contests with interactive puzzles.
3. Writing custom compilers for established programming languages should give more options to learn these languages in an interactive

environment and using the benefits we got by writing our interpreter.

4. Adding register and login with databases to track players' progression and save data across different devices.
5. Using more advanced game engine or creating a custom one should give more options and flexibility.

# CHAPTER 7|REFERENCES

**[2.1]** Scratch. "Scratch - Imagine, Program, Share". https://scratch.mit.edu/

**[2.2]** CodinGame. "Coding Games and Programming Challenges to Code Better". https://www.codingame.com/

**[2.3]** Codecademy. "Codecademy: Learn to Code - for Free". https://www.codecademy.com/

**[4.1]** Visual Studio. "Visual Studio: IDE and Code Editor for Software Developers". https://visualstudio.microsoft.com/

**[4.2]** C#. "C# docs - get started, tutorials, reference.". https://learn.microsoft.com/en-us/dotnet/csharp/

**[4.3]** Unity 3D. "Unity Real-Time Development Platform | 3D, 2D, VR & AR". https://unity.com/