

A Project Report

ON

**“MAZE RUNNER” WITH UNITY**

BY

**S J Rajath Krishna Sayan Guha Shiva K Deviah**

**1PI13CS129 1PI13CS139 1PI13CS147**

**Guide**

**Prof. V R Badri Prasad**

**Associate Professor**

**PESIT - CSE**

**Bangalore**

**August 2016 – Dec 2016**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**PES INSTITUTE OF TECHNOLOGY (an autonomous institute under VTU)**

**100 FEET RING ROAD, BANASHANKARI III STATE**

**BANGALORE-560085**

**PES Institute of Technology,**

**(an autonomous institute under VTU)**

**100 Feet Ring Road, BSK 3rd Stage, Bangalore - 560085**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

Certified that the Special Topic: Mini Project work entitled **“MAZE RUNNER” WITH UNITY** is a bonafide work carried out by **S J Rajath Krishna (1PI13CS129), Sayan Guha (1PI13CS139), and Shiva K Deviah (1PI13CS147)** in partial fulfillment for the award of degree ofBachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic semester August 2016 to December 2016.

Signature of the Guide Signature of the HOD

**Prof. V R Badri Prasad** **Prof. Nitin V Pujari**

**S J Rajath Krishna (1PI13CS129)**

**Sayan Guha (1PI13CS139)  
Shiva K Deviah (1PI13CS147)**

**ABSTRACT**

Game development is a rapidly growing field of software engineering. In a game project, the product is undoubtedly the game. However, a game is much more than just its constituent software. It has to provide content to become enjoyable. Just like how a web-server is useless without any content, a game, too, is more than just code. Any good game has a myriad of interleaving elements: the environment, story, characters, gameplay, the artwork, and so on. These elements must seamlessly work together to provide an enjoyable experience to the user.   
  
Over the decades, numerous gaming consoles have come and gone, and every single one of them has had their fair share of games. With technology always evolving, gaming consoles are becoming more and more powerful, and gaming experience is always improving. In an extremely competitive market, designing good, profitable games that everyone can enjoy is also becoming more challenging.

There are a lot of important concepts associated with game development, especially one designed with the Unity3D engine – the game objects, the physics engine, the assets, AI, and so on. Most of these concepts have not been, nor will be covered as part of any standard syllabus. Our objective in developing *Maze Runner* is to step out of our comfort zone and turn this project into a rich learning experience, so that we may use this knowledge to develop other real world applications with aplomb.

**ACKNOWLEDGEMENT**

It gives us great pleasure to acknowledge the support, guidance and motivation rendered by our lecturer, **Prof. V R Badri Prasad,** as the project guide. We express our profound gratitude for his continuous support and invaluable guidance in turning the project into reality.

We would also like to thank our friends, classmates, and family members for taking the time to look at our project and offering their invaluable criticism, all of which resulted in the betterment of the project.

Finally, we would like to thank God for blessing us with the enthusiasm and willpower to strive towards our goal.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **SL NO.** | | **TOPIC** | **PAGE NO.** |
| 1 | | Introduction | 1 |
| 2 | | Problem definition | 2 |
| 3 | | Literature survey |  |
| 4 | | Project requirement definition |  |
| 5 | | System requirements specification |  |
| 6 | | System design |  |
| 7 | | Pesudo code |  |
| 8 | | Results discussion |  |
| 9 | | Conclusion |  |
| 10 | | Further enhancement |  |
| 11 | | Bibliography |  |
| 12 | User manual | |  |

**LIST OF TABLES AND FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **SL NO.** | | **TOPIC** | **PAGE NO.** |
|  | |  |  |
|  |  | |  |
|  | |  |  |
|  |  | |  |
|  | |  |  |
|  |  | |  |

**1. INTRODUCTION**

Before embarking on the formidable task of developing a game, it is important to look at what makes a good game. Here are some of the most important ingredients for any good game:

* **Continuous challenge**

A good game designer gives their players continuous challenges, each of which leads to another challenge, to keep them "hooked" on playing the game.

* **Flexibility**

There should be more than one way to accomplish goals. Simply plotting out a step-by-step progression through the goals can be stifling. As much as possible, let each player (or team) work out their own strategy to the endpoint while still keeping the game challenging and achieving the learning objectives.

* **Rewards/Powerups**

Instead of just points towards victory, successful players can be rewarded with new capabilities, a new part of the board to explore or even a new task. These are surprisingly motivating, as the point of the game is not just to win it, but also to keep playing.

* **Interesting storyline**

A good storyline may not be required in every game. However, in event driven/story based games, the quality of the storyline can either make or break a game.

* **Enjoyableness**

Good games often incorporate interesting hidden features, Easter Eggs, or reward unrealistic behavior. These can all go a long way toward enhancing the user’s gaming experience.

In summary, the hallmark of a good game is one that provides a long-lasting source of challenge and enjoyment and leaves the player wanting more.

**2. PROBLEM/PROJECT DEFINITION**

To create an adventure-based game where a player maneuvers through randomly generated mazes filled with traps, collecting coins and advancing levels.

**3. LITERATURE SURVEY**

When it comes to game development, there is no dirth of tools, both free and commercial. For *Maze Runner,* we have decided to use Unity3D to develop the game. Unity3D consists of the following parts:

* **Game engine**   
  System designed to help create video games, fast and easy to develop with.
* **Visual editor**   
  See changes in real-time. For interactive and rapid prototyping.
* **Components**   
  Components are functionality built out of smaller pieces, for modularity and extensibility

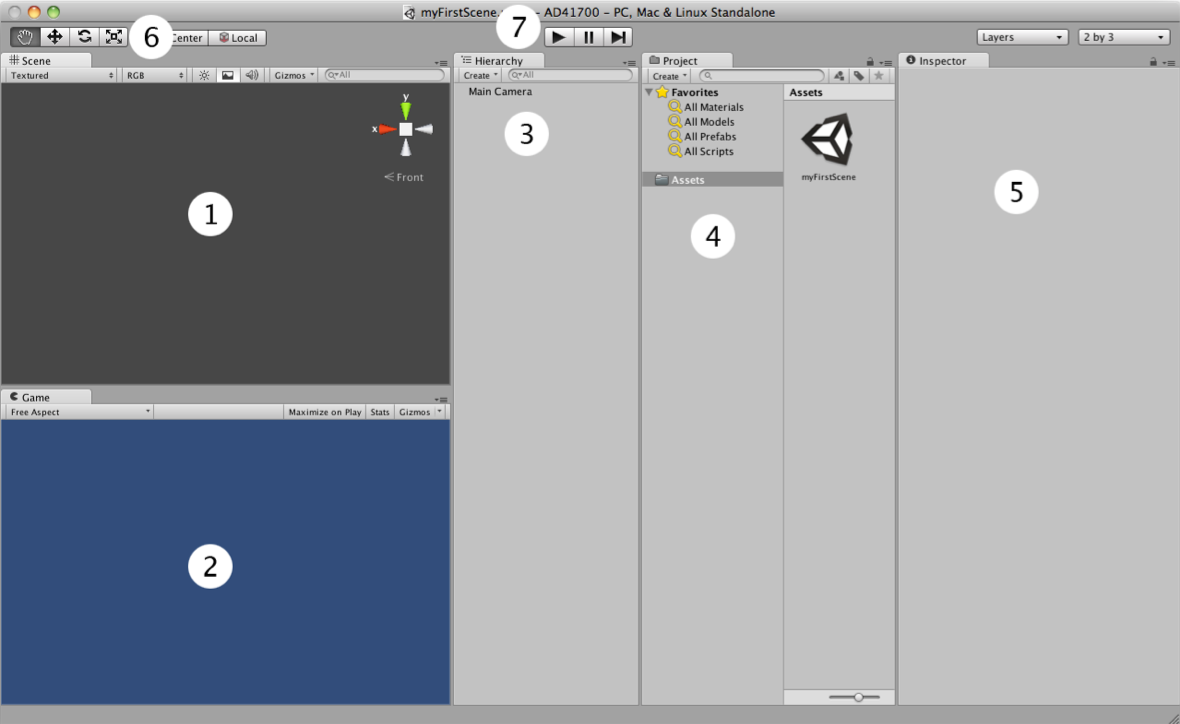
This section will take a brief look at both of these tools.

**3.1 Unity3D Engine**

Unity3D is a “game development ecosystem”, it includes an environment for the development of interactive 2D and 3D content including a rendering and physics engine, a scripting interface to program interactive content, a content exporter for many platforms (desktop, web, mobile) and a growing knowledge sharing community.

Unity3D comes in two flavors – a free version and a pro version ($1,500).

Physics engine, positional audio, web browser and standalone deployment, shaders, terrain editor



User Interface Components (see screenshot on previous page):

1. **Scene:** This is where you will place any visual assets in your Unity environment. It will update in real-time when you are previewing the game. Note the manipulator on the top right; this allows you to switch between a number of standard views. We are currently in the perspective view (toggle between isometric (2D) and perspective (3D)). Although this doesn't matter too much, it allows us to view our scene with a vanishing point, which is the standard way Unity games will display.
2. **Game:** When you're not actively running the game, it will show a rendering of how the game will look, ignoring graphical effects that need to be computed at run-time, from the point of view of the main camera. When you're previewing the game, you'll be playing through this window. Since our scene is currently empty, all this window is showing is the background color.
3. **Hierarchy:** This lists all the objects in the currently loaded scene, and any children they may have. **Children** are objects that can be thought of as subordinate to the parent object; wherever the top object moves, they'll follow, keeping the current offset they have to this object. This is an important concept for Unity beginners to understand; we'll cover it more in detail later and in the workshops.
4. **Project/Assets view:** This is a list of all custom assets for our game, including graphical assets, sound, scripts (more on these later), prefabs (pre- assembled game objects), and much more. Our current game is currently using only one empty scene (titled “myFirstScene”).
5. **Inspector:** Since we currently don't have any objects selected in the Hierarchy or the Project/Assets view, it's completely blank. The inspector allows us to look at and tweak individual settings of various game objects and assets, as well as adjust some global settings. The Inspector is content-sensitive and changes its parameters based on which game object/asset is selected. This is also a place to show you your project settings and preferences by choosing them from the Edit menu.
6. **Graphical icons for moving the scene and its contents**. The hand allows us to pan around the scene; when combined with other scene camera controls, Unity becomes very easy to navigate (see below). The icon on its right, which looks like four arrows, allows you to move a selected object around. We call this transforming the object. The next icon allows for rotation of the object, and the final one allows for uniform scaling of the object.
7. **Playback bar**. This allows us to play, pause, and stop running our game in the Unity editor. This is the quickest and easiest way to test and tweak the game.

Unity Languages

* C# (most common)
* Javascript
* Boo

**3.2 MonoDevelop**

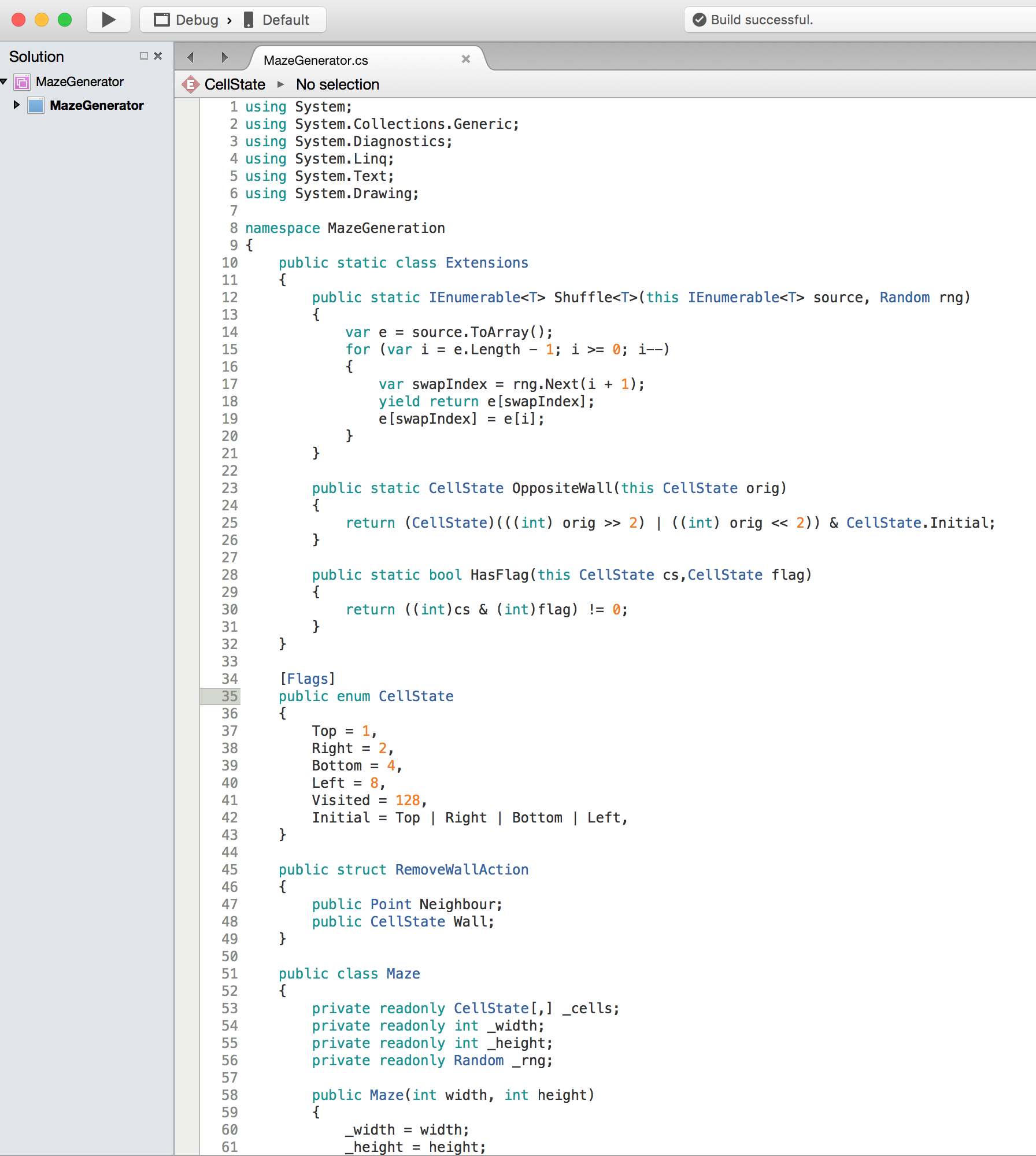
C# scripting editor to implement the game’s functionality

MonoDevelop (also known as Xamarin Studio) is an open source integrated development environment for Linux, OS X, and Windows. Its primary focus is development of projects that use Mono and .NET frameworks. MonoDevelop integrates features similar to those of NetBeans and Microsoft Visual Studio, such as automatic code completion, source control, a graphical user interface (GUI) and Web designer. MonoDevelop integrates a Gtk# GUI designer called Stetic. It supports Boo, C, C++, C#, CIL, D, F#, Java, Oxygene, Vala, and Visual Basic.NET.

MonoDevelop can be used on Windows, OS X and Linux. Officially supported Linux distributions include CentOS, Debian, Fedora, openSUSE, SUSE Linux Enterprise, Red Hat Enterprise Linux and Ubuntu, with many other distributions providing their own unofficial builds of MonoDevelop in their repositories. OS X and Windows have been officially supported since version 2.2.

MonoDevelop has included a C# compiler (an alternative to MSBuild and CSC) since its earliest versions. It currently includes a compiler that supports C# 1.0, C# 2.0, C# 3.0, C# 4.0, C# 5.0 and C# 6.0.

A customized version of MonoDevelop ships with Unity, the game engine by Unity Technologies. It enables advanced C# scripting, which is used to compile cross-platform video games by the Unity compiler.



(add stuff here)

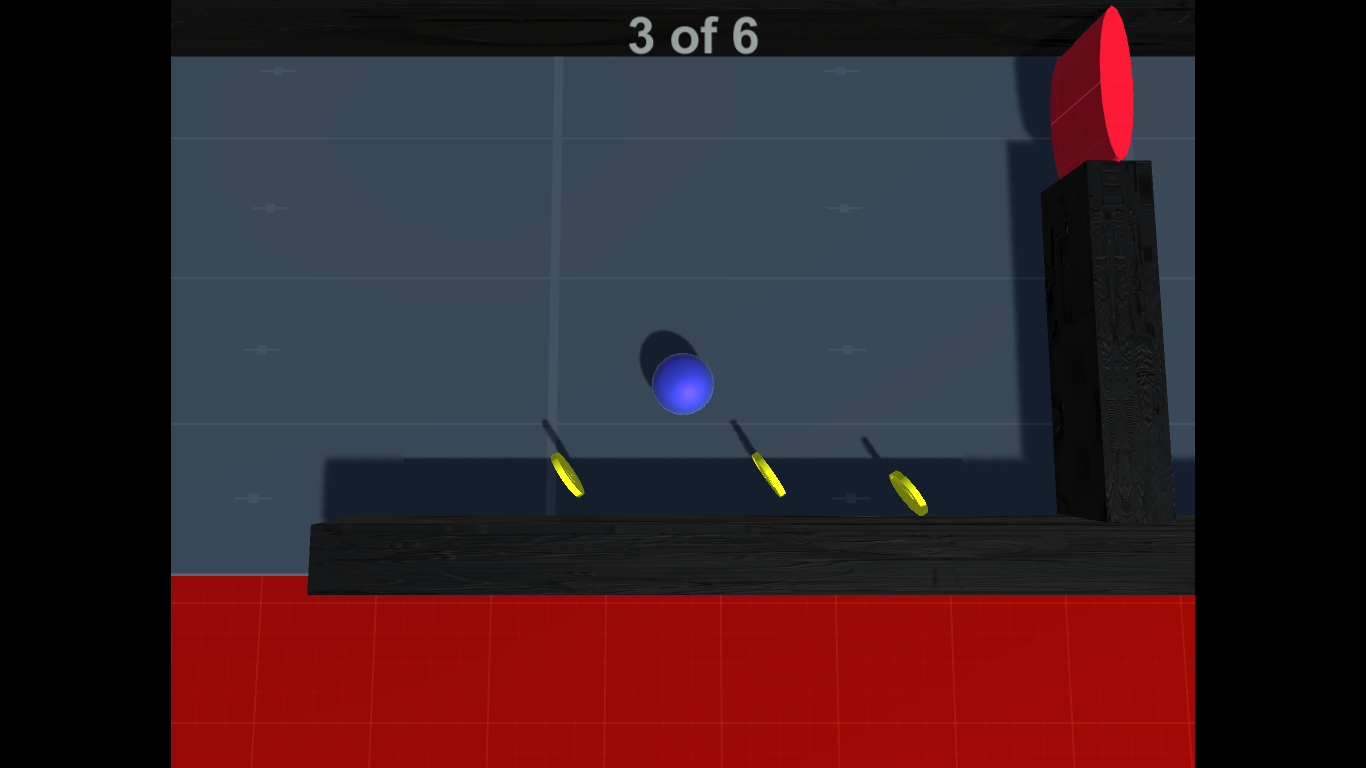
(add stuff here)**4. SYSTEM REQUIREMENT DEFINITION**

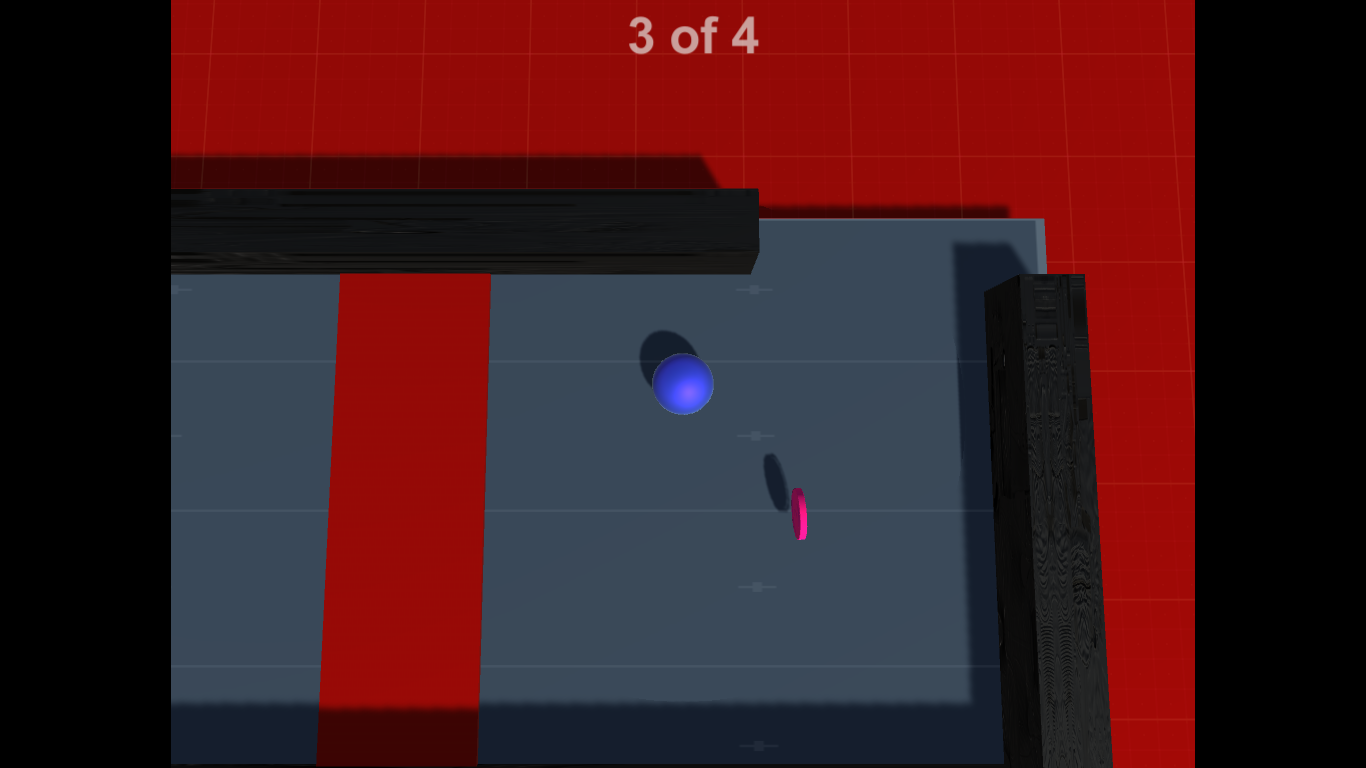
* + **Actual Game**
  + **Maze for the game**
  + **AI**

**(elaborate)**

**5. SYSTEM DESIGN**

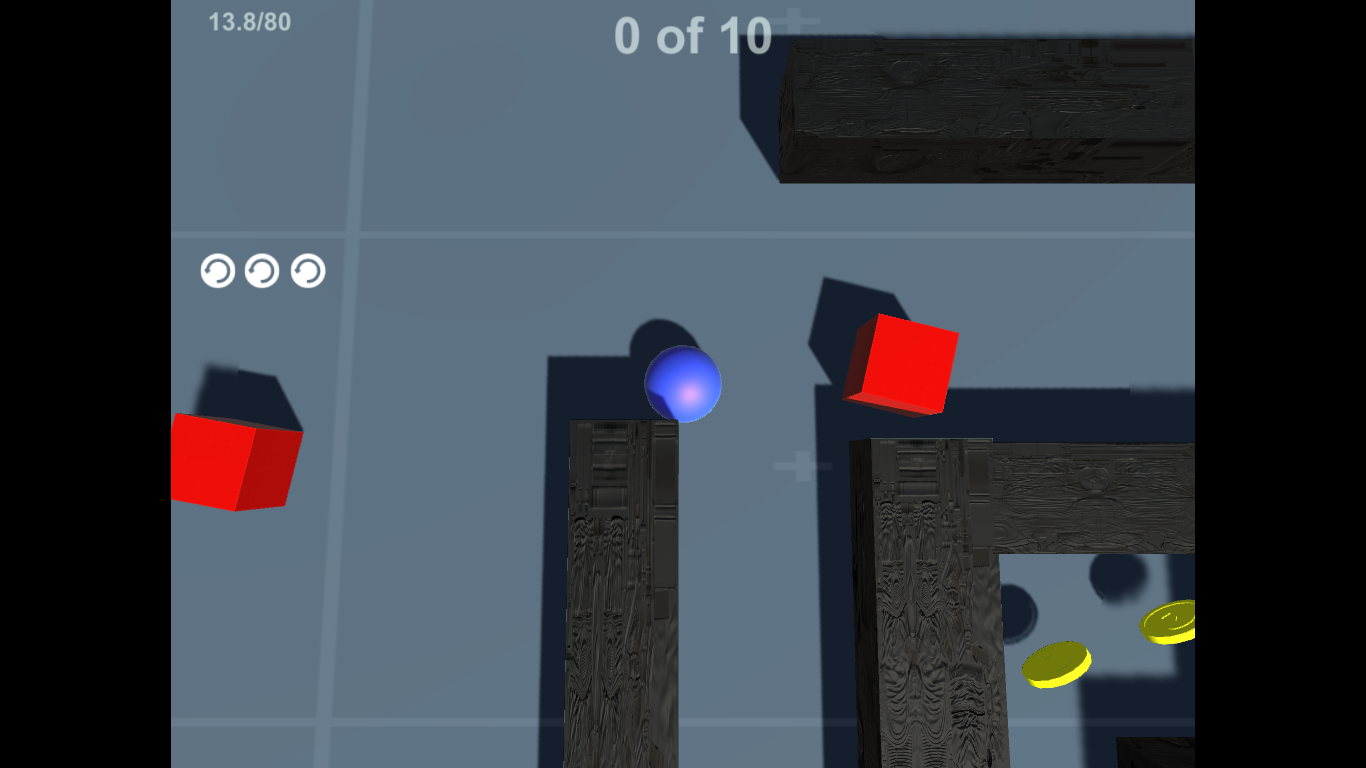
**5.1 Graphics**

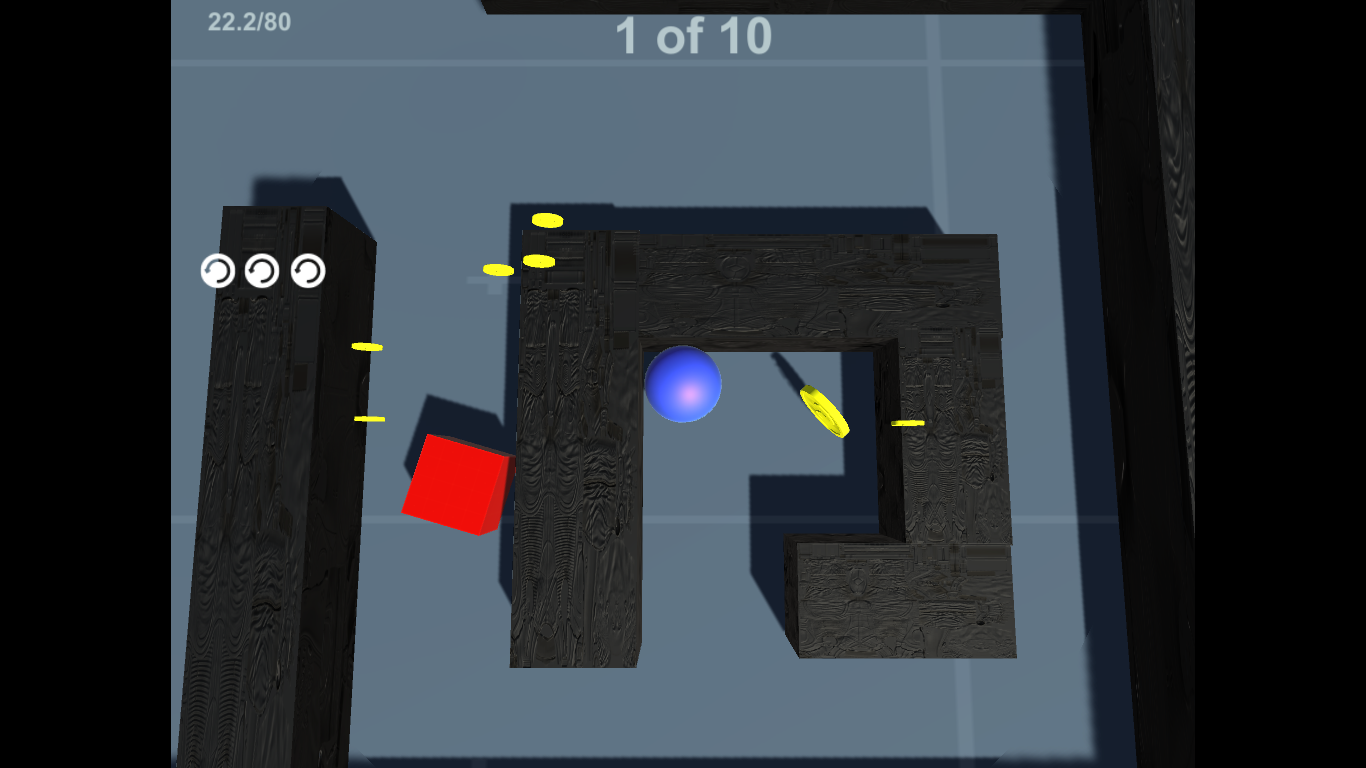
****











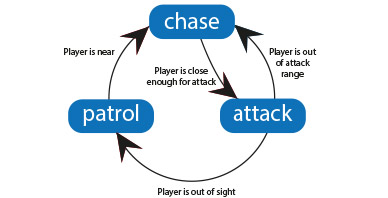
**5.2 Maze**

**Why the need for a maze?**   
The player begins each new level in a **randomly generated maze**. The player must navigate through the maze until they find the exit that leads to the next level.   
  
We will use AI to strategically place powerups as well as enemies in such a way that it is not too hard or too easy to beat the game.

**5.3 Smart AI**

**Proposed Implementation**We develop an enemy AI script that models a finite state machine that consists of patrol, alert and chase states along with simple behavior for each of those states.

**Finite State Machine**



**6. PSEUDOCODE**

* 1. **Unity3D Boilerplate** 
     1. **Skeleton Code**

using UnityEngine;

using System.Collections;

public class BasicTargetMover : MonoBehaviour {

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}  
}

* MonoBehavior is the base class that all Unity scripts are derived from
* public variables are exposed in the editor
* public variables and functions can be accessed when you have a reference to the component
* you can make an Object global by creating a public static variable
* **Referencing GameObjects**
* You can reference a gameObject in code in the following ways:  If the script is attached to the gameObject, you can reference the gameObject as:  this.gameObject or simply gameObject If we want to reference a gameObject that the script is NOT  attached to, we can do this in a number of ways:  We can have a public variable of type GameObject that we attach to another gameObject or prefab in the Unity Editor.  public GameObject target;
* We can search for the game Object using GameObject.Find to search for a GameObject by name:    
  GameObject target = GameObject.Find(“Enemy”);
* We can search for the game Object using GameObject.FindWithTag to search for a GameObject with a particular tag:    
  GameObject target = GameObject.FindWithTag(“Player”);

**Referencing Components**

You can get a reference to a component in the following ways:

Setup a public variable that holds the reference that you set in the Editor, such as:

public AudioSource backgroundMusic;

Get the component through a reference to a gameObect, such as:

gameObject.GetComponent<AudioSource>();

**Instantiating Prefabs**

You can dynamically create, or instantiate gameObjects in a scene from other gameObjects or Prefabs:

GameObject spawnedObject = Instantiate

(prefabObject, spawnPosition, spawnRotation) as

GameObject;

**Common Game Events**

Awake & Start - called once. This is where you can set things up.

Update - is called every game cycle before rendering a frame. This is where most game behaviour code goes, except physics code.

FixedUpdate - is called once every physics time step. This is the place to do physics-based game behaviour.  OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.

 OnTriggerEnter is called when this collider has touched another collider that is tagged as a trigger.

**Useful API Classes**

GameObject, Time, Transform, RigidBody, AudioSource

**Algorithm**

1. Generate a grid **G** of dimension **m** x **n**
2. Start with the top-left square on **G**, mark it as visited
3. **while** there are still unvisited squares **do**
   * **if** the **current** cell has unvisited neighbours:
     1. choose a random square **next**
     2. break the wall between **current** and **next**
     3. push **current** onto the stack
     4. set **current ← next** and mark it visited
   * **else** 
     1. pop a square **prev** from the stack
     2. set **current** ← **prev**
4. Finally, break the wall for the top-left and bottom-right squares to complete the maze

**Algorithm**

1. Initialize the values for static data like speed of movement, speed of rotation and span of patrol area.
2. Target player to be attacked.
3. If the player lies within a specific range (line of sight, Ray Casting in Unity) of the enemy's area of interest, then rotate the enemy to “look” at the player. Chase or follow the player till he collides or is out of sight, else continue patrolling.

**7. RESULTS DISCUSSION**

We hope that the results of our work can be evaluated through the quality of your gaming experience.

**8. CONCLUSION**

In conclusion, game development is a vast, hot field in today’s market. Games have come and gone. Some have left their mark. Others have been less lucky. Any good game is more than just the sum of its parts. There is a lot of thought, design, logic, and programming that must be pumped into a game before results can be seen.

Developing *Maze Runner* has provided us with great insight into the new and fascinating world of game development. It is our hope that our experience working on this project will enable us to go on to even greater endeavours in the future.

**9. FURTHER ENHANCEMENTS (Sayan)**

**10. BIBLIOGRAPHY**

[1] Introduction to Unity3D  
<https://www.cla.purdue.edu/vpa/etb/resources/AD41700_Unity3D_workshop01_F13.pdf>

[2] Unity3D tutorials  
<https://unity3d.com/learn/tutorials>  
  
[3] Maze generation   
<http://rosettacode.org/wiki/Maze_generation>

[4] Maze generation with Unity3D  
<http://catlikecoding.com/unity/tutorials/maze/>

[5] Implementing AI in Unity3D  
<http://answers.unity3d.com/questions/938221/basic-enemy-ai-in-c.html>