

Google Development Group Project



Game made by Unity:
Cube-R Run

Made by:
Altaf Kalappatt
(17BCE2235)

Index

1 Introduction

1.1 Gaming in the Field of Computer Science

1.2 About This Project

2 Methodology

2.1 Quality Function Deployment of "Cube-r Run"

2.2 Making of the Game

3 Conclusion

4 Reference

Acknowledgement

I express my deepest level of gratitude to my extra-curricular course and guild for this project, the Google Development Group VIT Vellore. I also would like to thank all my colleagues and friends who had first had experience in game development whose contribution for this project is invaluable. The blessings of my friends and family have helped me in times of peril and in times of doubt to which we would like to display our profound degree of thankfulness.

I would like to thank my esteemed institution VIT to give me the opportunity to create this project and to allow me to use the facilities that were provided by VIT itself.

I would also like to show appreciation to whomsoever that has helped me in this project and have not mentioned above.

Thank you

1 Introduction

1.1 Gaming in the Field of Computer Science

In the fast-growing field of computer science and development and even more rapidly growing sector of game development the future is hard to predict. I am working on this game as my project assignment from my 2 credit extra-curricular course and as part of my degree, I choose this type of work to work with development cycle, development period, graphics, scripting, adopting new technology, animation. In general software project is a project focusing on the creation of software. Consequently, Success can be measured by taking a look at the resulting software. In a game project, the product is a game. But and here comes the point: A game is much more than just its software. It has to provide content to become enjoyable. Just like a web server: without content the server is useless, and the quality cannot be measured. This has an important effect on the game project as a whole. The software part of the project is not the only one, and it must be considered in connection to all other parts: The environment of the game, the story, characters, game plays, the artwork, and so on.

1.2 About This Project

It is a movement-based game that developed through the Unity Engine whose rights belongs to Unity Technologies. The player of this game has to control a block (only control it's left and right movements) that is continuously moving forward on a plane with obstacles and few ramps. The goal of the player is to reach the end of the level without colliding or falling of the plane which is scaled to be 10 on the x axis. The main workings of this project are the use of the basic features of Unity Engine like levels, objects, scripts, animation, Unity based UI, input controlling.

The project has a welcome page which asks the user to read the instructions and has a button that links to the games first level, which is followed by five other levels making the full game a six-leveled game and the last screen asks if the player wants to restart the game or would the player like to exit the application.

2 Methodology

2.1 Quality Function Deployment of “Cube-r Run”

Quality Function Deployment is a technique that translates the needs of the customer into technical requirements for software/game. It concentrates on maximizing customer satisfaction from the Software engineering process. With respect to our project the following requirements are identified by a QFD

- Normal Requirements.
- Expected Requirements.
- Exciting requirements.

Normal Requirements

Normal requirements consist of objectives and goals that are stated during the meeting with the actor/gamer/relevant people. Normal requirements of my project are:

1. User friendly efficient and lucrative system.
2. Minimum maintenance cost (may be graphics definition).
3. Availability of expected requirements within the PC/mobile configuration.
4. Easy to operate.
5. The game with measured coding, professional thinking.

Expected Requirements

These requirements are implicit to the system and may be so fundamental that the actor/gamer/ relevant people do not explicitly state them. Their absence will be a cause for dissatisfaction.

1. Develop system within limited cost.
2. Maximum high definition.
3. Minimum hardware requirements which is relevant for this game.
4. Design whole system with efficient manner.

Exciting requirements

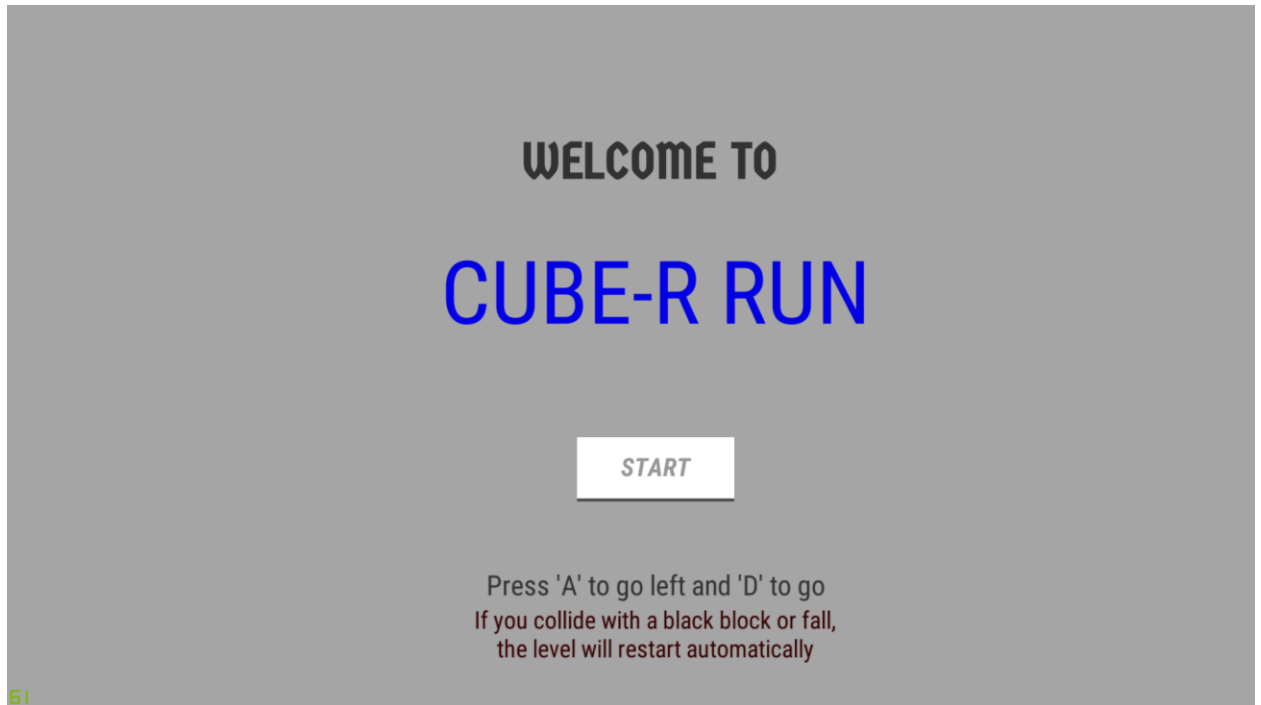
These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present:

1. The game is user-friendly and easy to use.
2. Maximum high regulation with minimum hardware.
3. I may provide an update for more levels.
4. Easy to update.

2.2 Making of the Game

Using Unity Engine, I have used the following basic concepts:

- 1 Transform (Positioning, Rotation and Scale of an object)
- 2 Rigidbody for 3D physics on blocks
- 3 Directional Lighting for fog and shadow
- 4 Scripts to assign functions for objects and buttons
- 5 Materials for coloring and identification of objects
- 6 Animations for transition of scenes and texts
- 7 Box Collider for trigger of level failed or passed

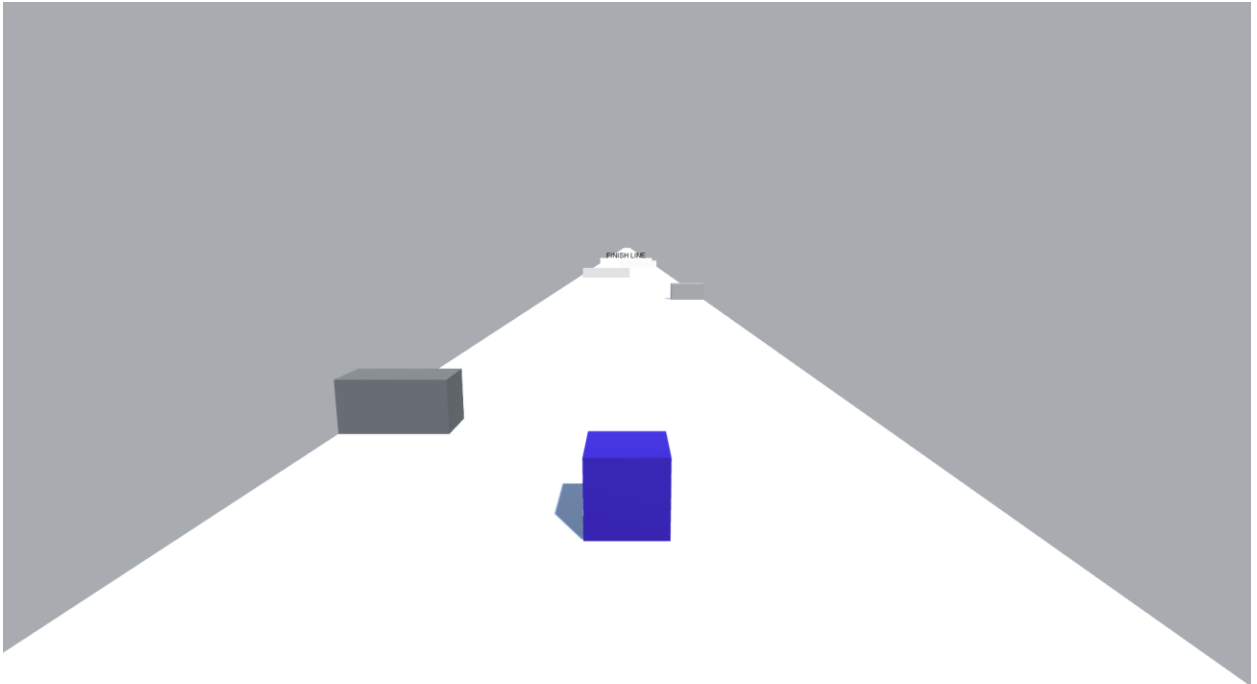


This is the welcome page which has the build index of 0.
All the texts are texts only with no specific functions.

The START button is linked to the StartGameScript.cs which loads the first level through the following function:

```
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class StartGameScript : MonoBehaviour
5  {
6      public void StartGame()
7      {
8          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
9      }
10 }
11
```

The StartGame() function loads the next scene by buildIndex which has int values starting from 0.



61

The player is the blue block and must avoid the black blocks. This is a screenshot of level 1 which has the buildIndex of 1.

The player is controlled through the PlayerMovement.cs script which is linked to the blue block

```

1  using UnityEngine;
2
3  public class Playermovement : MonoBehaviour
4  {
5
6      //Initializing Rigidbody properties to rb of player
7      public Rigidbody rb;
8
9      public Transform player_y;
10
11      public float forwardforce = 2000f;
12      public float sidewaysforce = 500f;
13
14      // Use this for initialization
15
16
17      // Update is called once per frame
18      void FixedUpdate()
19      {
20          //add a forward force
21          rb.AddForce(0, 0, forwardforce * Time.deltaTime);
22
23          if (Input.GetKey("d"))
24          {
25              rb.AddForce(sidewaysforce * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
26          }
27
28          if (Input.GetKey("a"))
29          {
30              rb.AddForce(-sidewaysforce * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
31          }
32
33          if (player_y.position.y < -1f)
34          {
35              FindObjectOfType<GameManager>().EndGame();
36          }
37      }
38  }

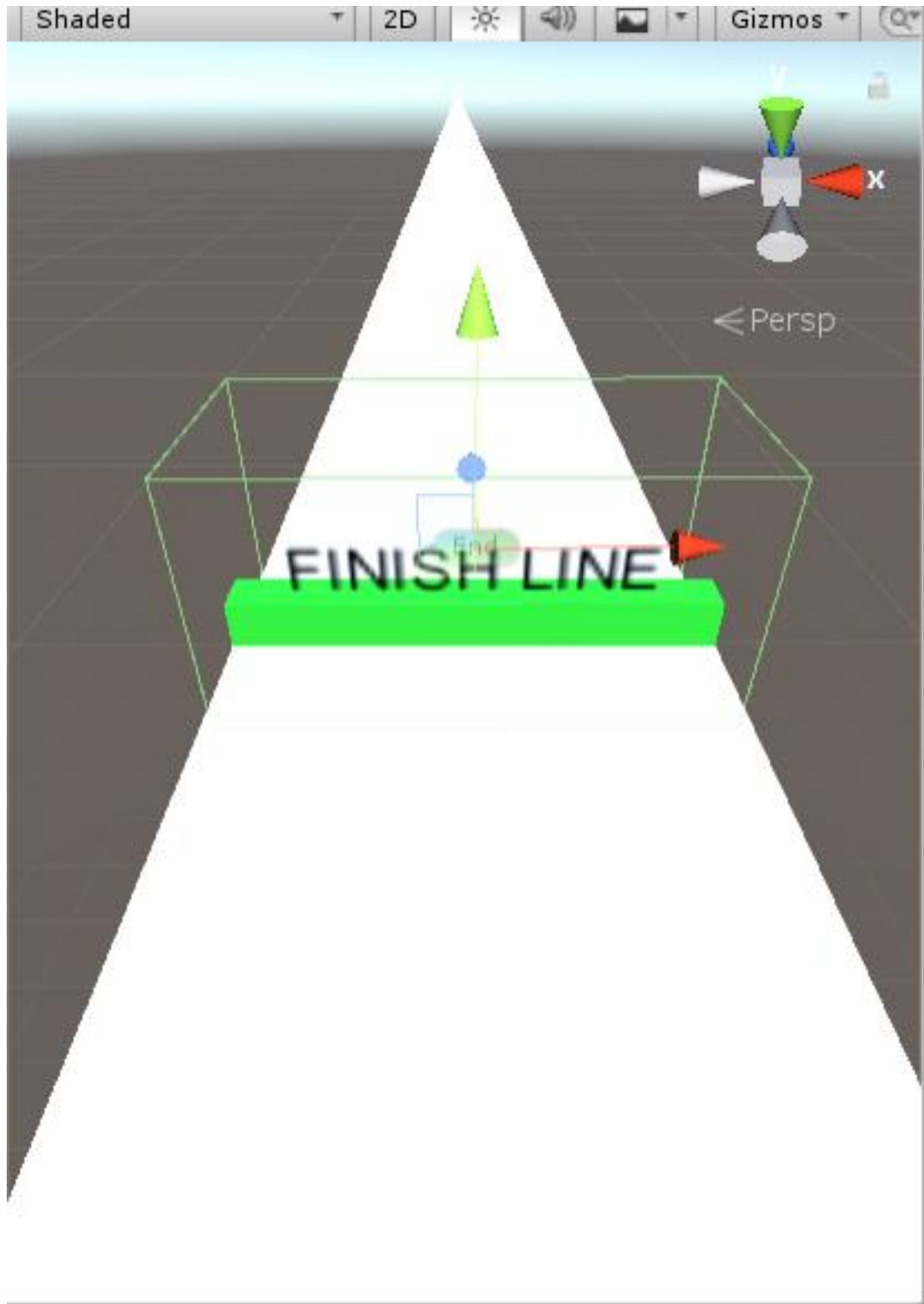
```


The function `FixedUpdate()` gives the keys 'A' and 'D' the control of the x axis with positive and negative values respectively. This function also checks if the player falls off from the platform by checking it's z position value and then calls the `EndGame()` function from `GameManager`.

Before I show the code of `GameManager.cs`, it has another function `Completelevel()` as well, which is to set the `completelevelUI` to true.

```
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class GameManager : MonoBehaviour
5  {
6
7      bool GameHasEnded = false;
8
9      public float restartDelay = 1f;
10
11     public GameObject completelevelUI;
12     public GameObject restartlevelUI;
13
14     public void CompleteLevel()
15     {
16         completelevelUI.SetActive(true);
17     }
18
19     public void EndGame()
20     {
21         if (GameHasEnded == false)
22         {
23             GameHasEnded = true;
24             Debug.Log("GAME OVER");
25             restartlevelUI.SetActive(true);
26         }
27     }
28 }
```

The level is completed by placing a trigger at the end of the level. In my game the trigger is another cube where the Mesh Renderer of the cube is turned off but the cube will still sense if the player goes through it.



As we can see, the trigger is invisible and the FINISH LINE is inside it.

LEVEL
COMPLETE

61

When the player goes through the trigger, Completelevel() is called and the animation of "Level Complete" is called. After two seconds of the animation the function LoadNextLevel() from LevelComplete.cs is called.

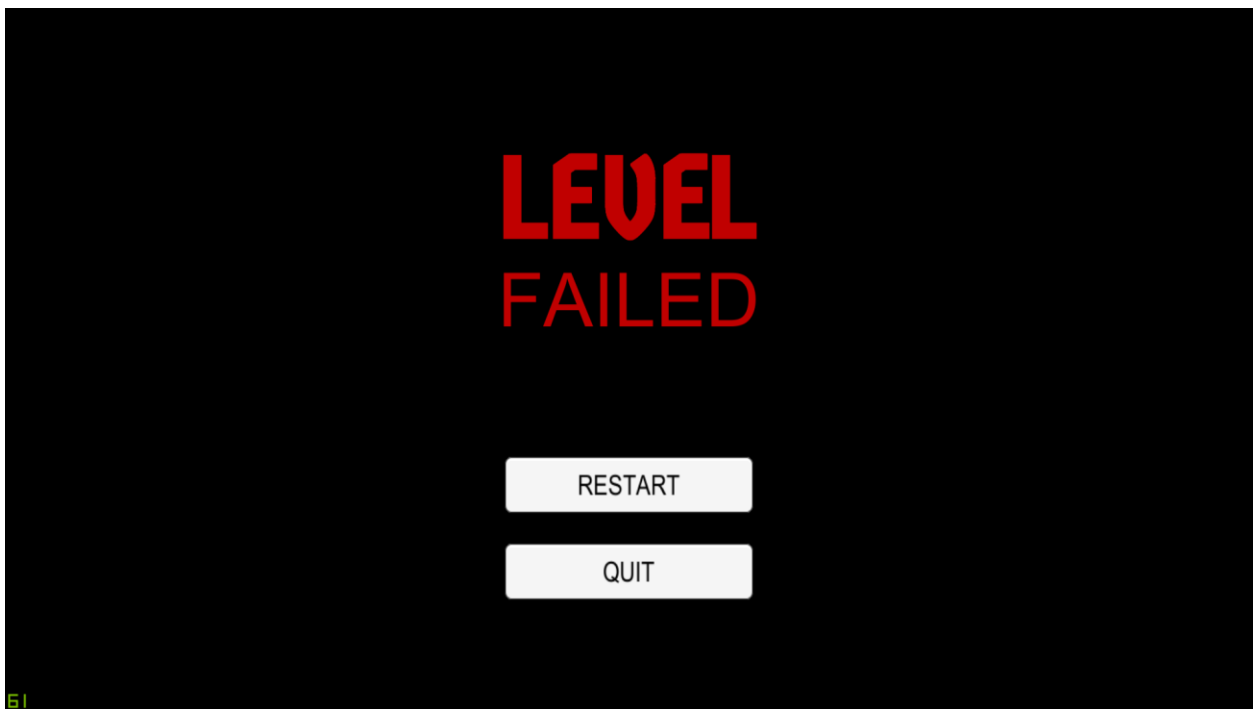
```
Assembly-CSharp - LevelComplete - LoadNextLevel()
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class LevelComplete : MonoBehaviour
5  {
6      public void LoadNextLevel()
7      {
8          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
9      }
10 }
11
```

This function calls the next buildIndex.

But if the player fails by colliding into a black block or falling off this screen is animated as restartlevelUI is set to 'true'.

This is PlayerCollision.cs which has the function OnCollisionEnter() which is called if the player collides with a black cube all of which has the tag "Obstacle".

```
1  using UnityEngine;
2
3  public class PlayerCollision : MonoBehaviour
4  {
5
6      public Playermovement movement;
7      public GameObject restartlevel;
8
9      private void OnCollisionEnter(Collision collision)
10     {
11         if (collision.collider.tag == "Obstacle")
12         {
13             movement.enabled = false;
14             FindObjectOfType<GameManager>().EndGame();
15             //Debug.Log("You lost :(");
16         }
17     }
18 }
19
```



This Canvas has two texts and two buttons all of which are animated for two seconds after EndGame() is called.

The RESTART button will load the scene with the same buildIndex again.

The QUIT will exit the application.

```

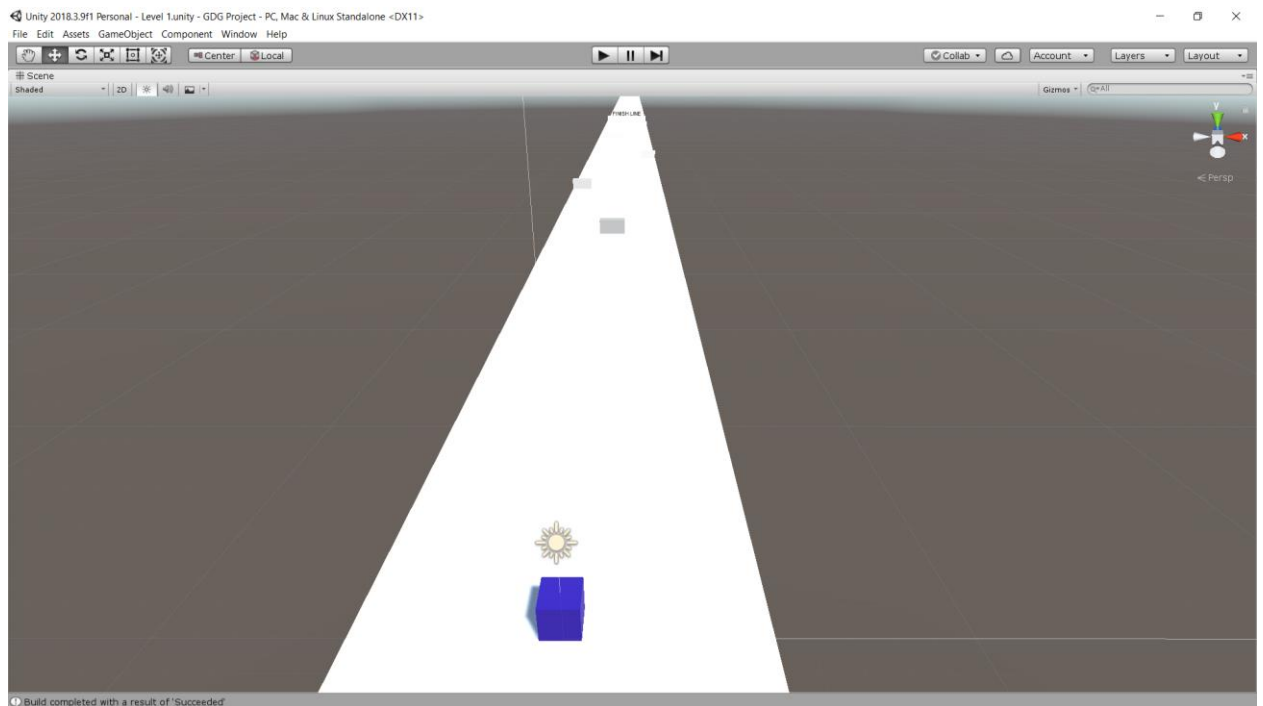
1  using UnityEngine.SceneManagement;
2  using UnityEngine;
3
4  public class Restart : MonoBehaviour
5  {
6      public void Reload()
7      {
8          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
9      }
10
11     public void Quit()
12     {
13         Debug.Log("WHYYYYYYYYYYYYYYYY?");
14         Application.Quit();
15     }

```

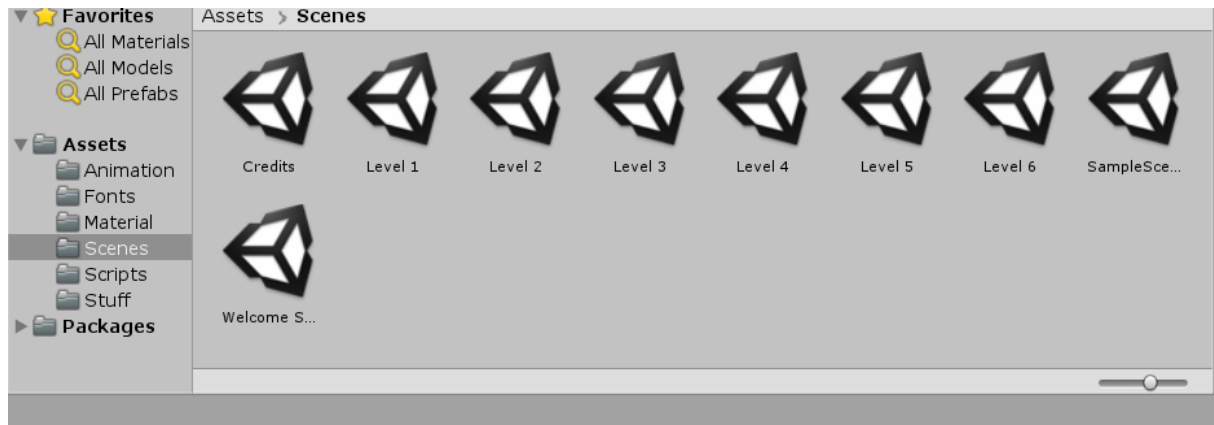
The game has 6 levels in total each increases in difficulty after another.

They were manually designed and were designed through the scene viewer.

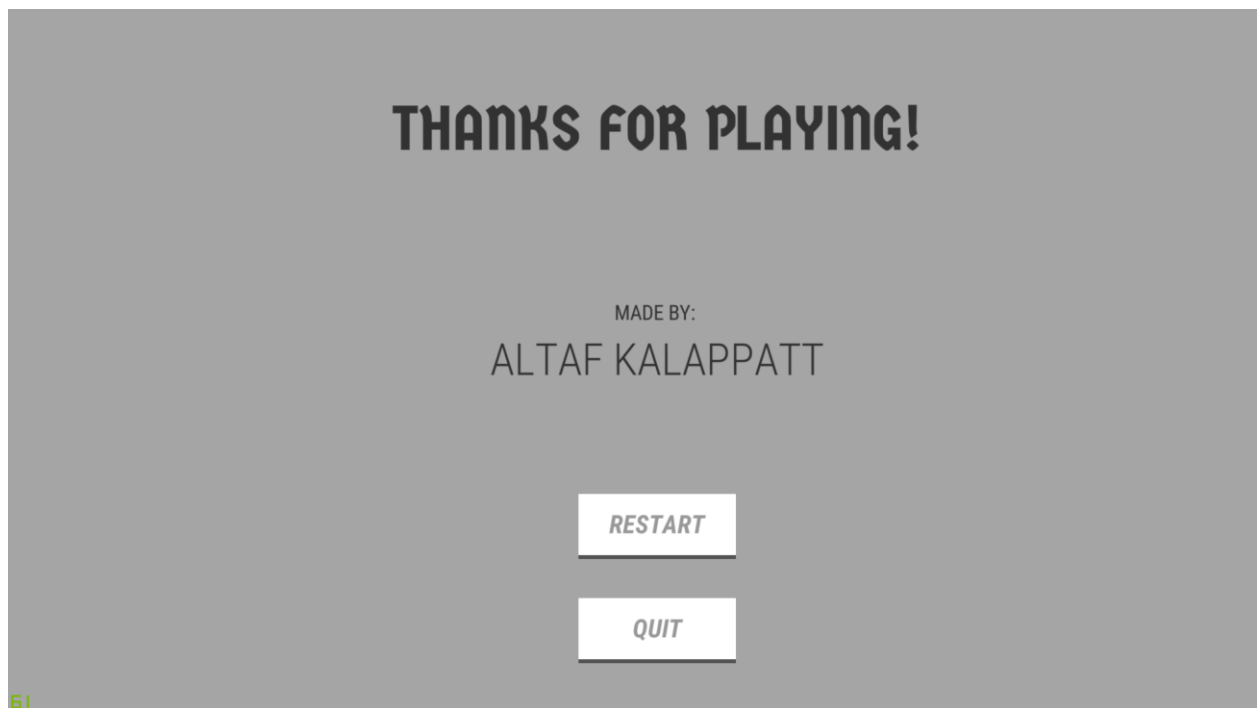
I used Level 1 as a template for all the remaining levels. This is a screenshot of Level 1 in scene viewer



All the black cubes cannot be seen in this shot because of the fog lighting.



After the completion of all six levels, the player will be taken to the last buildIndex and to the last scene of the game which is Credits.



This scene is similar to the EndGame() called page.

But the difference here is that when RESTART is clicked, the game takes the player to buildIndex 0 which is the start of the game by calling Restart() in Credits.cs script.

```

1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class Credits : MonoBehaviour
5  {
6      public void Quit()
7      {
8          Debug.Log("WHYYYYYYYYYYYYYYYY?");
9          Application.Quit();
10     }
11
12     public void Restart()
13     {
14         Debug.Log("Thank Lord!");
15         SceneManager.LoadScene(0);
16     }
17 }







```

This concludes all the basics of the making of this game and designing it.

3 Conclusion

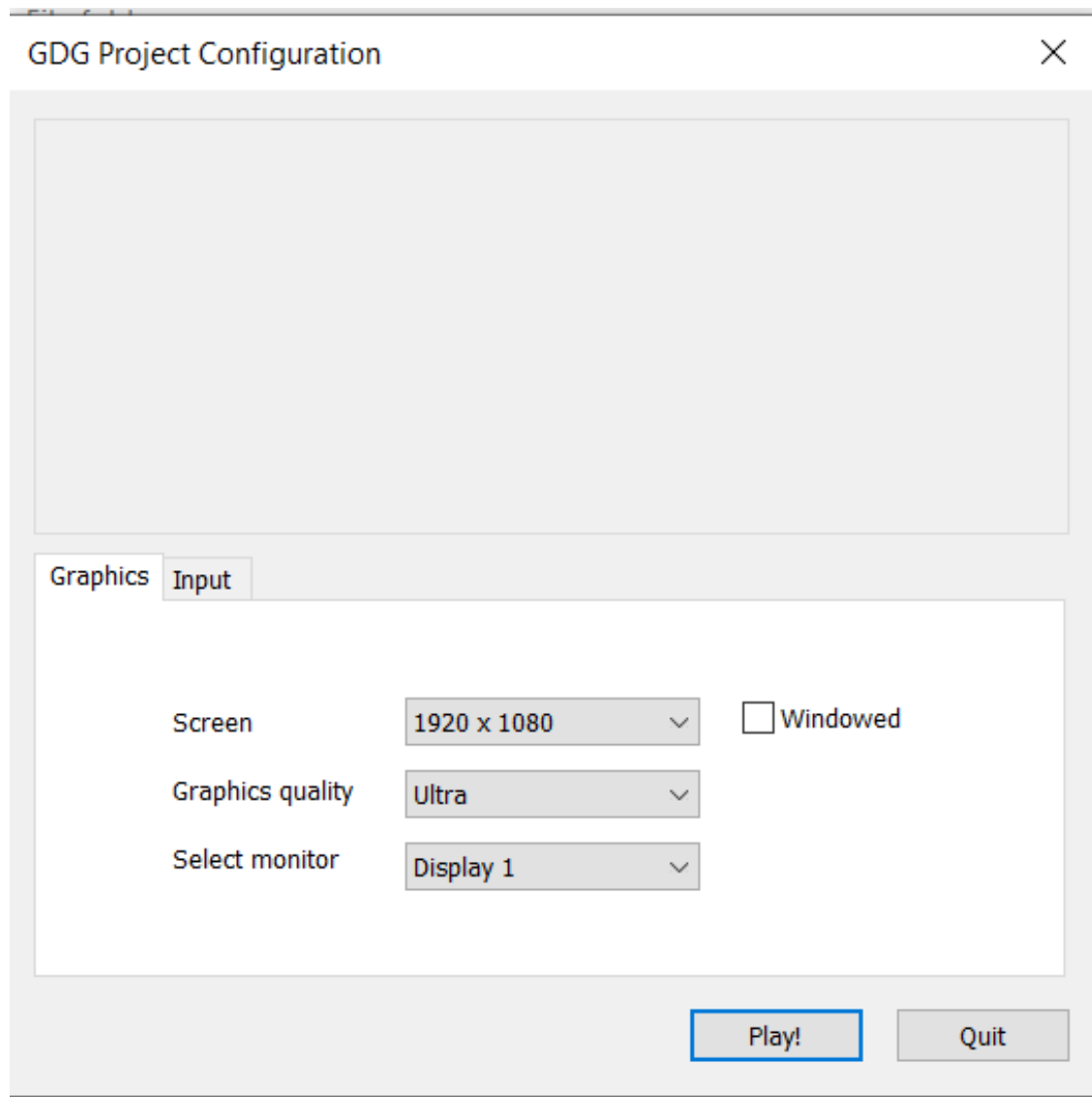
Using Unity Engine I have successfully created a game with basic physics and player input that contains 6 levels and 8 scenes in total.

The game is saved under the name GDGProject as I was unaware this will be the final name shown in the File Explorer.

	GDG Project_Data	3/24/2019 5:58 PM	File folder	
	Mono	3/24/2019 5:58 PM	File folder	
	GDG Project	3/15/2019 5:01 PM	Application	636 KB
	UnityCrashHandler64	3/15/2019 5:03 PM	Application	1,424 KB
	UnityPlayer.dll	3/15/2019 5:03 PM	Application extens...	22,364 KB
	WinPixEventRuntime.dll	3/15/2019 4:55 PM	Application extens...	42 KB

The game can be build in Android as well with the only change needed to be done in the player input in PlayerMovement.cs.

The game starts with the Unity Engine asking for the resolution and FPS to be played at.



The works just the way it can be viewed when using the Unity Engine.

4 Reference

The Engine was downloaded from:

<https://unity3d.com/get-unity/download>

YouTube channel Brackeys has many tutorials for Unity

https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA

Unity's tutorial on how to start a game

<https://unity3d.com/learn/tutorials/topics/developer-advice/how-start-your-game-development>

Game Development Tutorial using Unity 3D

<https://www.studytonight.com/game-development-in-2D/>