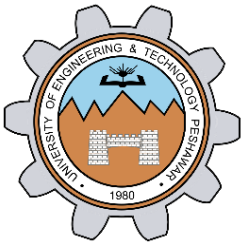
Project Report



Fall 2024

CSE-411L Intro to Game Development Lab

Submitted by:

Ali Asghar(21PWCSE2059)

Muhammad Sadeeq(21PWCSE2028)

Suleman Shah(21PWCSE1983)

Muhammad Shahab(21PWCSE2074)

Submitted to:

Engr. Abdullah Hamid

Date:

31st January 2025

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Contents

Abstract

Quantum Legacy is a cutting-edge sci-fi tower defense game that immerses players in a futuristic universe where strategic planning and quick decision-making are key to survival. Featuring 8 unique towers, each with distinct abilities and upgrade paths, players must defend their base against waves of alien invaders across 4 meticulously designed levels, each offering unique terrain and environmental challenges. The game includes a Towers Shop for strategic tower selection, a Settings Menu for customizable sound and graphics, a robust Saving System to track currency and experience, and a Username Change System for personalization. With its blend of tactical gameplay, immersive visuals, and a rewarding progression system, Quantum Legacy delivers an engaging and dynamic experience for both casual and hardcore gamers. Designed for PC and mobile, the game sets the stage for future expansions, including multiplayer modes and additional content, ensuring long-term replayability and growth.

Chapter 1

Introduction

Quantum Legacy is a fast-paced, strategic tower defense game set in a futuristic sci-fi universe. Players must defend their base from waves of alien invaders by strategically placing and upgrading towers. With a blend of tactical gameplay, immersive visuals, and a progression system, the game offers a rewarding experience for both casual and hardcore gamers.

1.1 Background

Tower defense (TD) games are a strategy subgenre where players defend a base by placing and upgrading towers to stop waves of enemies. Core mechanics include tower variety, enemy waves, resource management, and strategic level design. Popular themes range from fantasy and sci-fi to historical settings, appealing to a wide audience. TD games are known for their accessibility, replayability, and strategic depth, making them a staple on PC, consoles, and mobile devices. Originating as custom maps in games like Warcraft III, the genre has evolved with modern titles like Kingdom Rush and Bloons TD 6, incorporating advanced graphics, RPG elements, and multiplayer modes. The future of TD games may include VR/AR integration, cross-platform play, and procedural generation, ensuring the genre remains dynamic and engaging.

1.2 Unity Game Engine

The Unity Game Engine is a powerful and versatile game development platform used for creating 2D, 3D, AR, and VR experiences across multiple platforms, including PC, consoles, mobile devices, and web browsers. Known for its user-friendly interface, real-time rendering, and extensive asset store, Unity enables developers of all skill levels to build games efficiently. Unity is built around a component-based architecture, where objects in a scene, known as GameObjects, can be customized using scripts, physics, animations, and UI elements. It supports C# scripting, allowing developers to write custom game logic using the MonoBehaviour system. The engine also includes powerful tools such as NavMesh for AI pathfinding, Cinemachine for camera control, and Unity Physics for realistic interactions. One of Unity's greatest strengths is its cross-platform support, enabling developers to build once and deploy to multiple platforms with minimal adjustments. Additionally, Unity offers a robust Asset Store, where developers can access a vast library of ready-to-use assets, scripts, and plugins. Unity is widely

used across the game industry for both indie and AAA game development, as well as in fields like simulation, architecture, film production, and machine learning. Its continuous updates and growing community make it a top choice for game developers worldwide.

1.3 Unity Tower Defense Template

The Unity Tower Defense Template provides a structured framework for developing a tower defense game using Unity. It includes essential features such as main menu navigation, level selection, tower placement, enemy waves, and game progression mechanics. This template allows developers to quickly set up a game where players can select levels, build defensive structures, and defend against incoming enemies.

The template follows a modular design, allowing easy customization of tower attributes, enemy behaviors, and difficulty scaling. It integrates UI elements, animations, and sound effects to enhance user experience. Upon completing a level, players can either return to the main menu or continue playing, ensuring a seamless game loop.

This template serves as a solid foundation for developing a full-fledged tower defense game, offering flexibility for further enhancements such as upgradable towers, special abilities, and multiplayer features.

The Tower defense Template is made using two reusable frameworks.

1.3.1 Action Game Framework

The first framework is called the Action Game Framework. The Action Game Framework includes code that covers concepts needed in action games, such as taking damage and logic for ballistics and projectiles.

1.3.2 Core Framework

The second framework is called the Core Framework. The Core Framework covers concepts that are common to games of all genres, such as game saving, data management, timers, math utilities and more

Figure 1.1: Unity Tower Defense Template

Chapter 2

Methodology

In this chapter, we describe the methods for extending the Unity Tower Defense template to make it a fully functional tower defense game instead of a prototype. Section 2.1 describes the main game flow while other sections describe the additional features of our project.

2.1 Game Flow

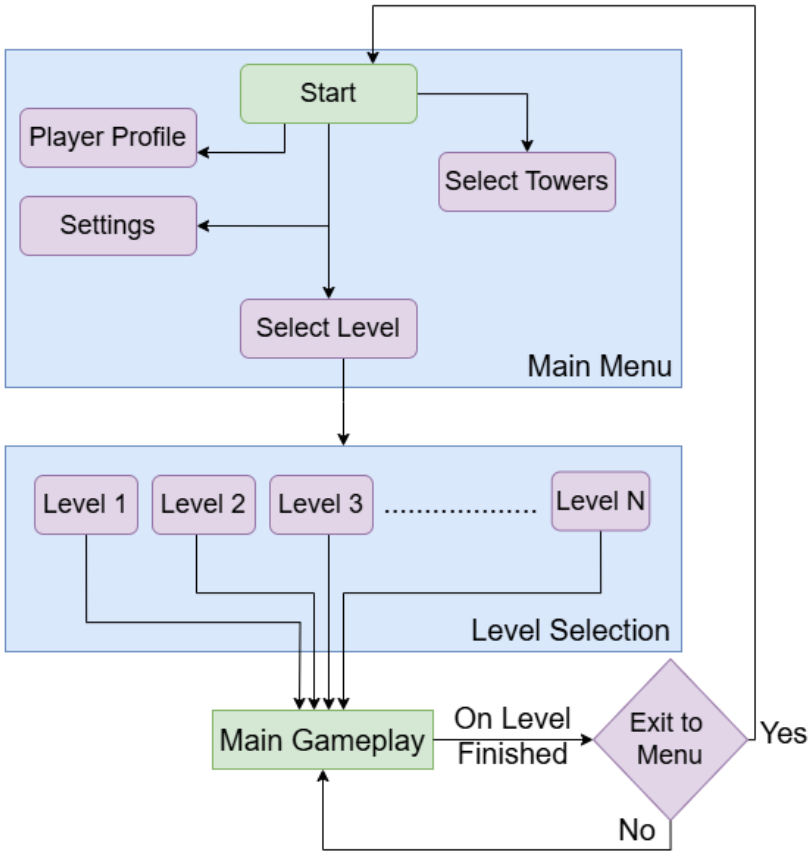


Figure 2.1: Gameplay Flowchart

Figure 2.1 outlines the user navigation flow in this game, detailing the interactions between the main menu, level selection, and gameplay. The process begins at the Start node, where players can access different features such as Player Profile,

Settings, and Select Towers Menu before proceeding to Select Level. The Level Selection section presents multiple levels, allowing players to choose one before transitioning into Main Gameplay. Once a level is completed, a decision point determines whether the player returns to the menu or continues playing

2.2 Towers

Players have access to 8 distinct towers, each with unique abilities, strengths, and upgrade paths. Examples include:

- **Machine Gun**: High damage, single-target attacks.
- **EMP**: Slows nearby enemies down.
- **FlameThrower**: Breathes Fire at enemies.
- **Laser**: Shoots laser beams with long range, high damage, slow fire rate.
- **Mega Laser**: Better version of Laser Tower
- **Mortar**: High AoE damage on ground enemies only.
- **Missile Launcher**: Fires homing missiles for heavy damage.
- **Energy Pylon**: Generates Energy

Each tower can be upgraded up to 3 levels (during gameplay), enhancing its power, range, and special abilities.

2.3 Levels

The game features 4 meticulously designed levels, each with its own terrain, enemy types, and challenges. Each level introduces new enemy types and environmental hazards, requiring players to adapt their strategies.

2.4 Towers Shop and Selection

Before each level, players can visit the **Towers Shop** to purchase and select up to 4 towers for the upcoming battle. The shop features:

- **Tower Unlocks**: New towers can be unlocked using in-game currency earned from completing levels.
- **Strategic Selection**: Choosing the right combination of towers is crucial for success. A player can select only 4 cards at a time.

2.5 Settings Menu

The game includes a comprehensive **Settings Menu** to customize the player's experience:

- **Sound Settings**: Adjust master volume, music, and sound effects.
- **Graphics Settings**: Toggle between low, medium, and high graphics presets.

2.6 Saving System

The game features a robust **Saving System** to track player progress. User data is encrypted in a file and saved in the persistant path of device:

- **Currency**: Earned by defeating enemies and completing levels. Used to purchase and upgrade towers.
- **Experience Points (XP)**: Gained by completing levels and achieving objectives. Levels up the player profile, unlocking new towers and abilities.
- **Auto-Save**: Progress is automatically saved after each level, ensuring no progress is lost.

2.7 Username Change System

Players can personalize their experience with a **Username Change System**:

- **Custom Username**: Players can set or change their in-game username at any time.
- **Profile Integration**: Usernames are displayed on leaderboards and in multiplayer modes (if added in future updates).

Chapter 3

Some Important Code Screenshots

3.1 TowerLibrary.cs

TowerLibrary.cs | TowerSelectScreen.cs M | GameManager.cs

D:\GitHUb\TowerDefense_GD\Assets\UnityTechnologies\TowerDefenseTemplate\Scripts\TowerDefense\Towers\Data\TowerLibrary.cs

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

namespace TowerDefense.Towers.Data
{
    /// <summary>
    /// The asset which holds the list of different towers
    /// </summary>
    [CreateAssetMenu(fileName = "TowerLibrary.asset", menuName = "TowerDefense/Tower Librar
    7 references
    public class TowerLibrary : ScriptableObject, IList<Tower>, IDictionary<string, Tower>
    {
        /// <summary>
        /// The list of all the towers
        /// </summary>
        21 references
        public List<Tower> configurations;

        /// <summary>
        /// The internal reference to the dictionary made from the list of towers
        /// with the name of tower as the key
        /// </summary>
        13 references
        Dictionary<string, Tower> m_ConfigurationDictionary;

        /// <summary>
        /// The accessor to the towers by index
        /// </summary>
```

Figure 3.1: TowerLibrary Script

The Script in Figure 3.1 is a ScriptableObject that manages a collection of Tower objects, providing both list-based and dictionary-based access. It implements IList and IDictionary to enable retrieval by index and by tower name.

3.2 TowerLevelData.cs

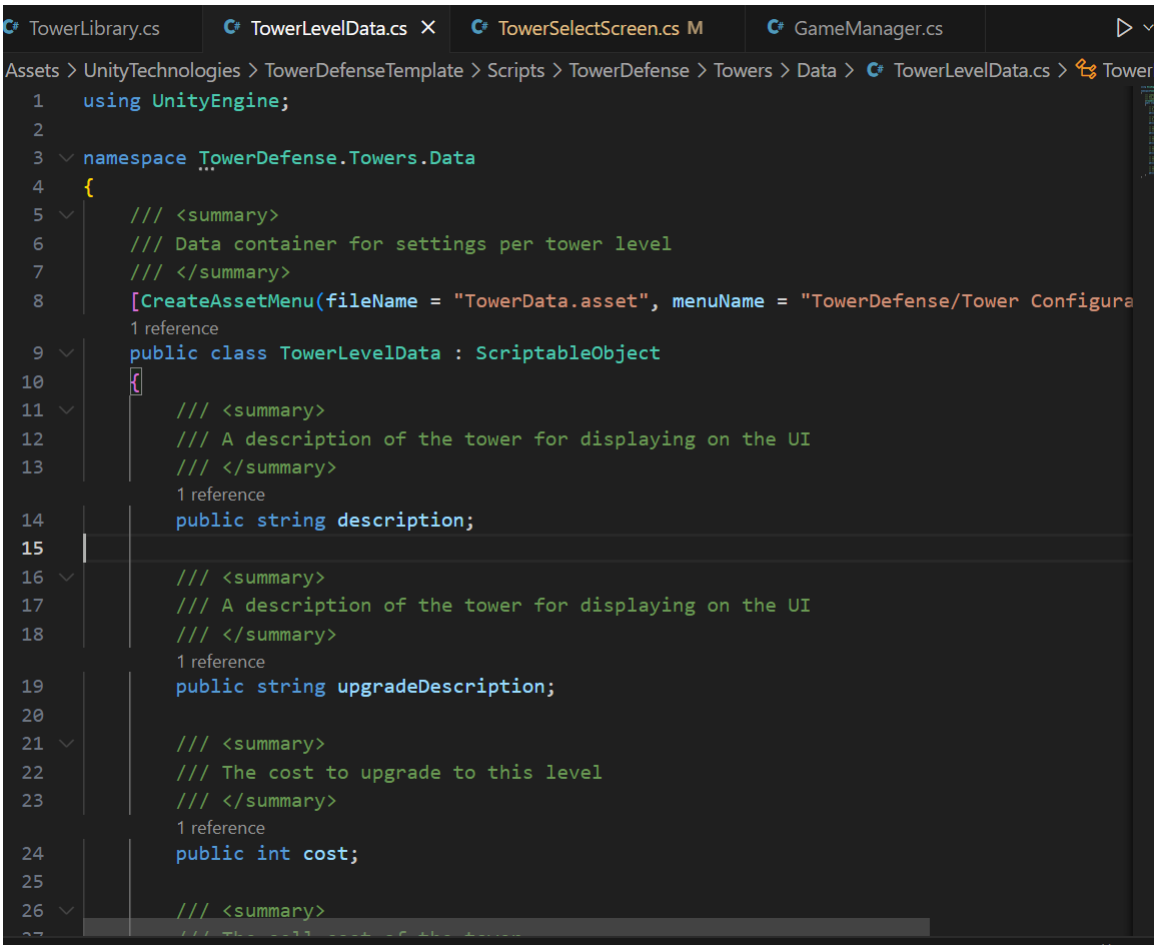


Figure 3.2: TowerLevelData Script Part 1

The script in figure [3.2,3.3] is a 'ScriptableObject' that stores settings for each tower level, including descriptions, cost, health, and an icon for UI display. It helps manage tower upgrades and attributes in a Tower Defense game.

```
            1 reference
24          public int cost;
25
26          /// <summary>
27          /// The sell cost of the tower
28          /// </summary>
            1 reference
29          public int sell;
30
31          /// <summary>
32          /// The max health
33          /// </summary>
            1 reference
34          public int maxHealth;
35
36          /// <summary>
37          /// The starting health
38          /// </summary>
            1 reference
39          public int startingHealth;
40
41          /// <summary>
42          /// The tower icon
43          /// </summary>
            2 references
44          public Sprite icon;
45      }
46  }
```

Figure 3.3: TowerLevelData Script Part 2

```
15  public class GameManager : GameManagerBase<GameManager, GameDataStore>
27      protected override void Awake(){
28          Screen.sleepTimeout = SleepTimeout.NeverSleep;
29          base.Awake();
30
31          int i;
32          for ( i = 0; i < towerlist.Count; i++){
33              Debug.Log("is Unlocked "+ IsTowerUnlocked(i));
34          }
35
36          //Ensure first 4 towers are always unlocked
37          for ( i = 0; i < 4; i++){
38              if (!IsTowerUnlocked(i)){
39                  UnlockTower(i);
40                  SelectTower(i);
41              }
42          }
43
44          if (LevelManager.instance){
45
46          LevelManager.instance.towerLibrary.Clear();
47
48          for (i = 0; i < towerlist.Count; i++){
49              if (IsTowerUnlocked(i) && IsTowerSelected(i)){
50                  selectedTowers.Add(towerlist[i]);
51                  Debug.Log($"{i} Added");
52              }
53           }
54           Debug.Log("Tower Updated");
```

Figure 3.4: GameManager Script Part 1

```
58      /// <summary>
59      /// Method used for completing the level
60      /// </summary>
61      /// <param name="levelId">The levelId to mark as complete</param>
62      /// <param name="starsEarned"></param>
        1 reference
63      public void CompleteLevel(string levelId, int starsEarned)
64      {
65          if (!levelList.ContainsKey(levelId))
66          {
67              Debug.LogWarningFormat("[GAME] Cannot complete level with id = {0}. Not in
68              return;
69          }
70
71          m_DataStore.CompleteLevel(levelId, starsEarned);
72          SaveData();
73      }
```

Figure 3.5: GameManager Script Part 2

```
76      /// <summary>
77      /// Method used for unlocking the tower
78      /// </summary>
        2 references
79      public void UnlockTower(int ind)
80      {
81          m_DataStore.UnlockTower(ind);
82          SaveData();
83      }
84
85      /// <summary>
86      /// Method used for selecting the tower
87      /// </summary>
        3 references
88      public void SelectTower(int ind)
89      {
90          m_DataStore.SelectTower(ind);
91          SaveData();
92      }
93
        1 reference
94      public void DeSelectAllTowers(){
95          m_DataStore.DeSelectAllTowers();
96          SaveData();
97      }
```

Figure 3.6: GameManager Script Part 3

```
99          /// <summary>
100         /// Gets the id for the current level
101         /// </summary>
            4 references
102         public LevelItem GetLevelForCurrentScene()
103         {
104             string sceneName = SceneManager.GetActiveScene().name;
105
106             return levelList.GetLevelByScene(sceneName);
107         }
108
109         /// <summary>
110         /// Determines if a specific level is completed
111         /// </summary>
112         /// <param name="levelId">The level ID to check</param>
113         /// <returns>true if the level is completed</returns>
            0 references
114         public bool IsLevelCompleted(string levelId)
115         {
116             if (!levelList.ContainsKey(levelId))
117             {
118                 Debug.LogWarningFormat("[GAME] Cannot check if level with id = {0} is compl
119                 return false;
120             }
121
122             return m_DataStore.IsLevelCompleted(levelId);
123         }
```

Figure 3.7: GameManager Script Part 4

```
            5 references
125         public bool IsTowerUnlocked(int ind){
126             return m_DataStore.IsTowerUnlocked(ind);
127         }
128
            2 references
129         public bool IsTowerSelected(int ind){
130             return m_DataStore.isTowerSelected(ind);
131         }
132         /// <summary>
133         /// Gets the stars earned on a given level
134         /// </summary>
135         /// <param name="levelId"></param>
136         /// <returns></returns>
            2 references
137         public int GetStarsForLevel(string levelId)
138         {
139             if (!levelList.ContainsKey(levelId))
140             {
141                 Debug.LogWarningFormat("[GAME] Cannot check if level with id = {0} is comp
142                 return 0;
143             }
144
145             return m_DataStore.GetNumberOfStarForLevel(levelId);
146         }
147     }
148 }
```

Figure 3.8: GameManager Script Part 5

3.3 GameManager.cs

3.4 GameDataStore.cs

3.5 LevelSaveData.cs

3.6 TowerSaveData.cs

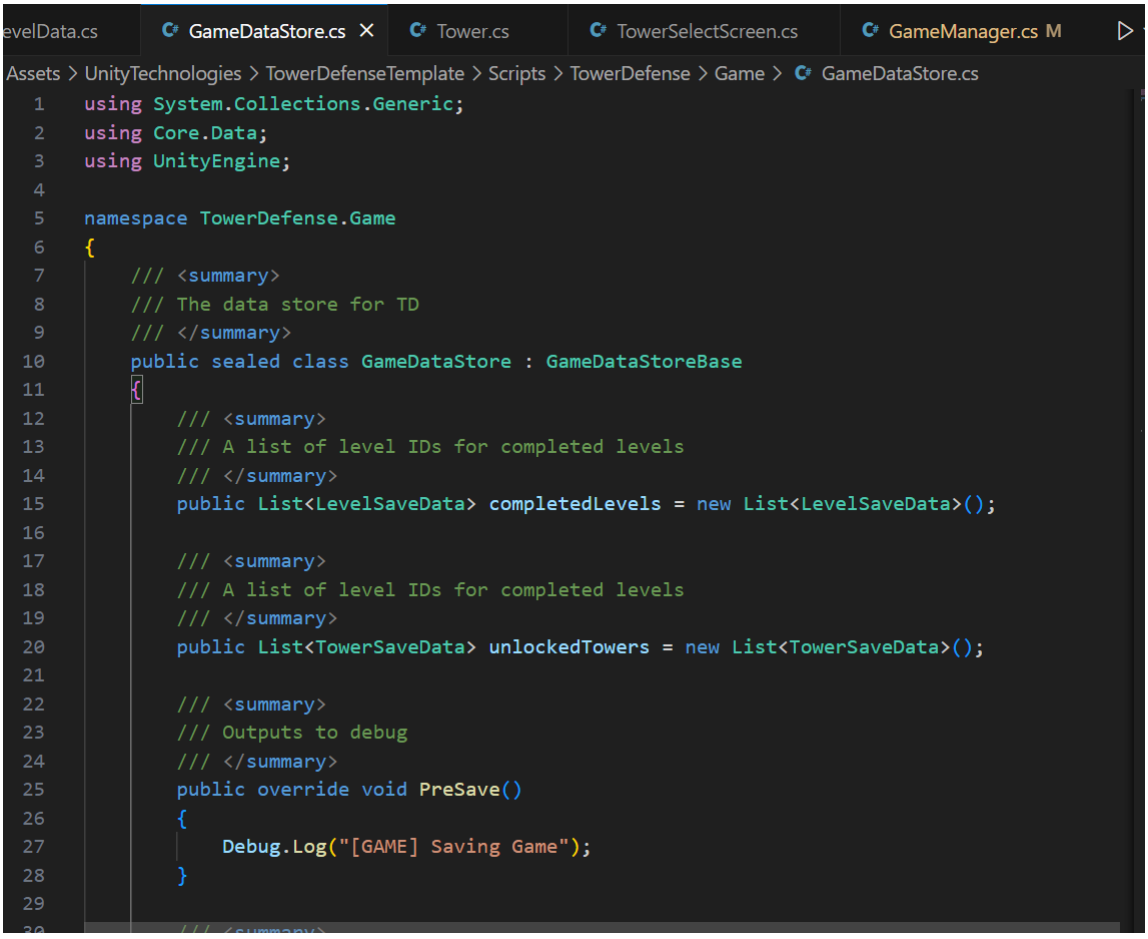


Figure 3.9: GameDataStore Script Part 1

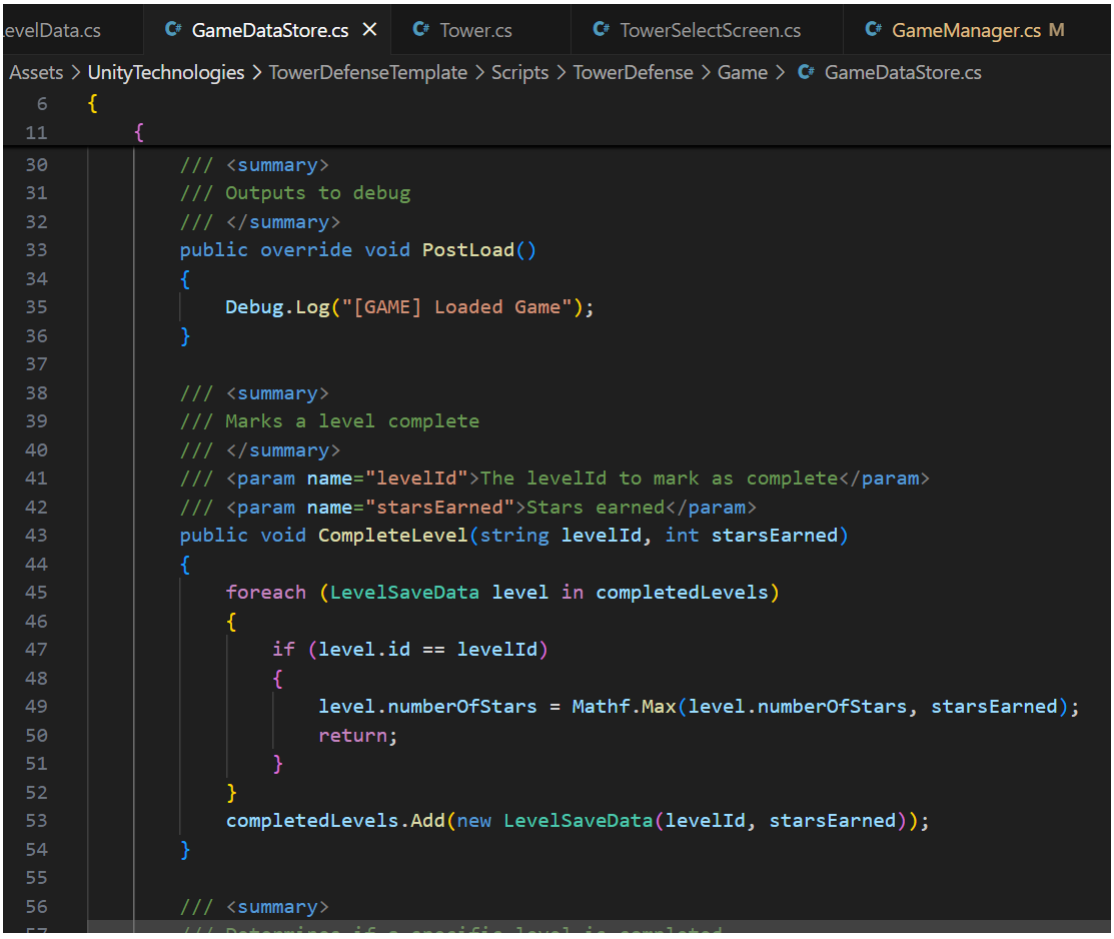


Figure 3.10: GameDataStore Script Part 2

```
11      {
56          /// <summary>
57          /// Determines if a specific level is completed
58          /// </summary>
59          /// <param name="levelId">The level ID to check</param>
60          /// <returns>true if the level is completed</returns>
61          public bool IsLevelCompleted(string levelId)
62          {
63              foreach (LevelSaveData level in completedLevels)
64              {
65                  if (level.id == levelId)
66                  {
67                      return true;
68                  }
69              }
70              return false;
71          }
72
73          /// <summary>
74          /// Marks a tower unlock
75          /// </summary>
76          /// <param name="levelId">The levelId to mark as complete</param>
77          /// <param name="starsEarned">Stars earned</param>
78          public void UnlockTower(int ind)
79          {
80              foreach (TowerSaveData tower in unlockedTowers)
81              {
82                  if (tower.index == ind)
```

Figure 3.11: GameDataStore Script Part 3

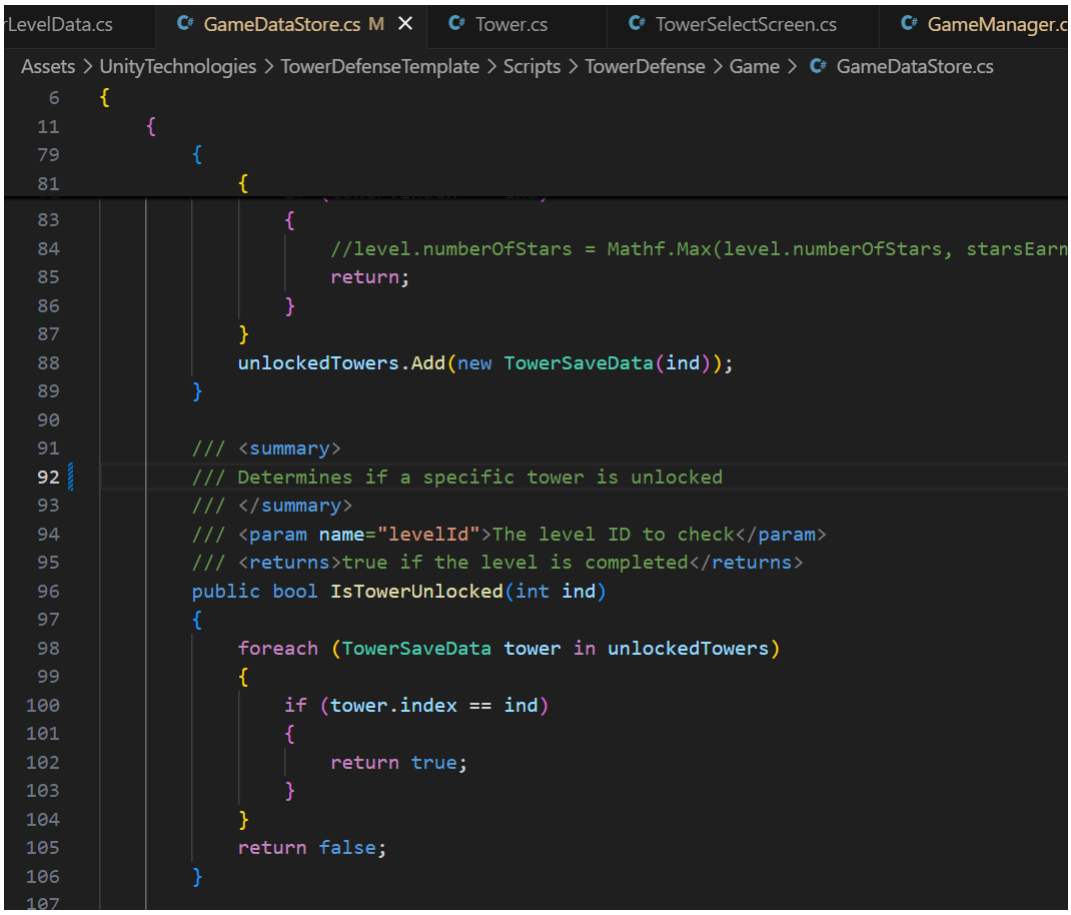


Figure 3.12: GameDataStore Script Part 4

Figure 3.13: GameDataStore Script Part 5

```
11      {
137         public bool isTowerSelected(int ind)
138         {
139             foreach (TowerSaveData tower in unlockedTowers)
140             {
141                 if (tower.index == ind && tower.isSelected == true)
142                 {
143                     return true;
144                 }
145             }
146             return false;
147         }
148
149         /// <summary>
150         /// Retrieves the star count for a given level
151         /// </summary>
152         public int GetNumberOfStarForLevel(string levelId)
153         {
154             foreach (LevelSaveData level in completedLevels)
155             {
156                 if (level.id == levelId)
157                 {
158                     return level.numberOfStars;
159                 }
160             }
161             return 0;
162         }
163     }
164 }
```

Figure 3.14: GameDataStore Script Part 6

```
Data.cs ×  GameDataStore.cs M  LevelSaveData.cs ×  Tower.cs  Tow
ets > UnityTechnologies > TowerDefenseTemplate > Scripts > TowerDefense > Game > LevelSaveD
using System;

namespace TowerDefense.Game
{
    /// <summary>
    /// A calss to save level data
    /// </summary>
    [Serializable]
    public class LevelSaveData
    {
        public string id;
        public int numberOfStars;

        public LevelSaveData(string levelId, int numberOfStarsEarned)
        {
            id = levelId;
            numberOfStars = numberOfStarsEarned;
        }
    }
}
```

Figure 3.15: LevelSaveData Script

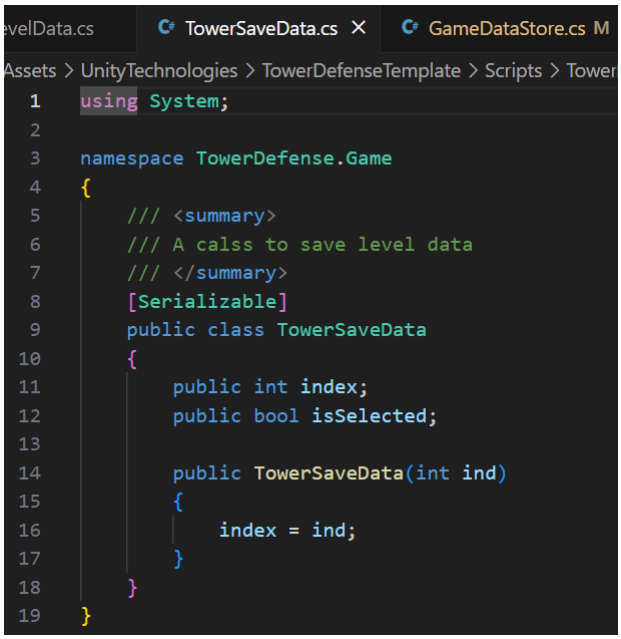


Figure 3.16: TowerSaveData Script

Figure 3.17: GameDataStore Script Part 6

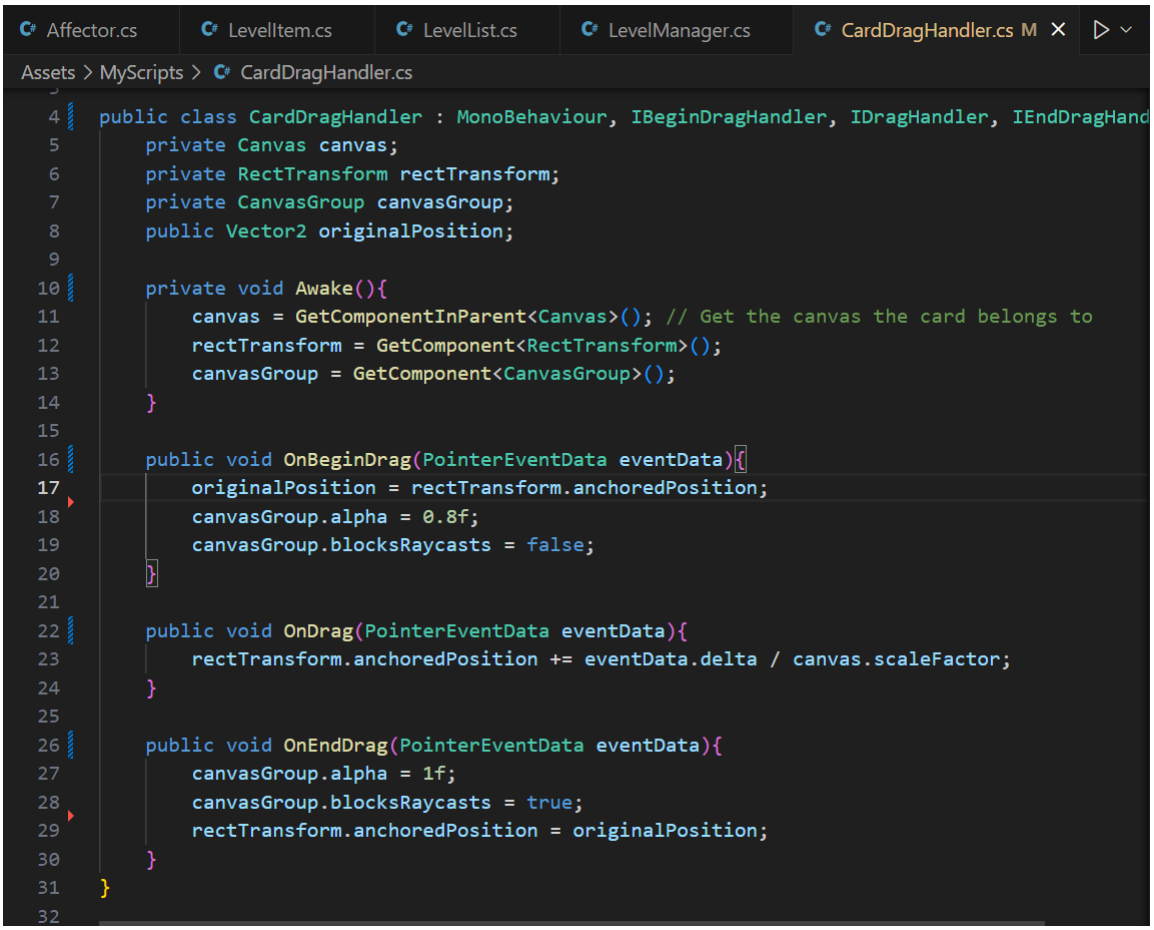


Figure 3.18: CardDragHandler Script

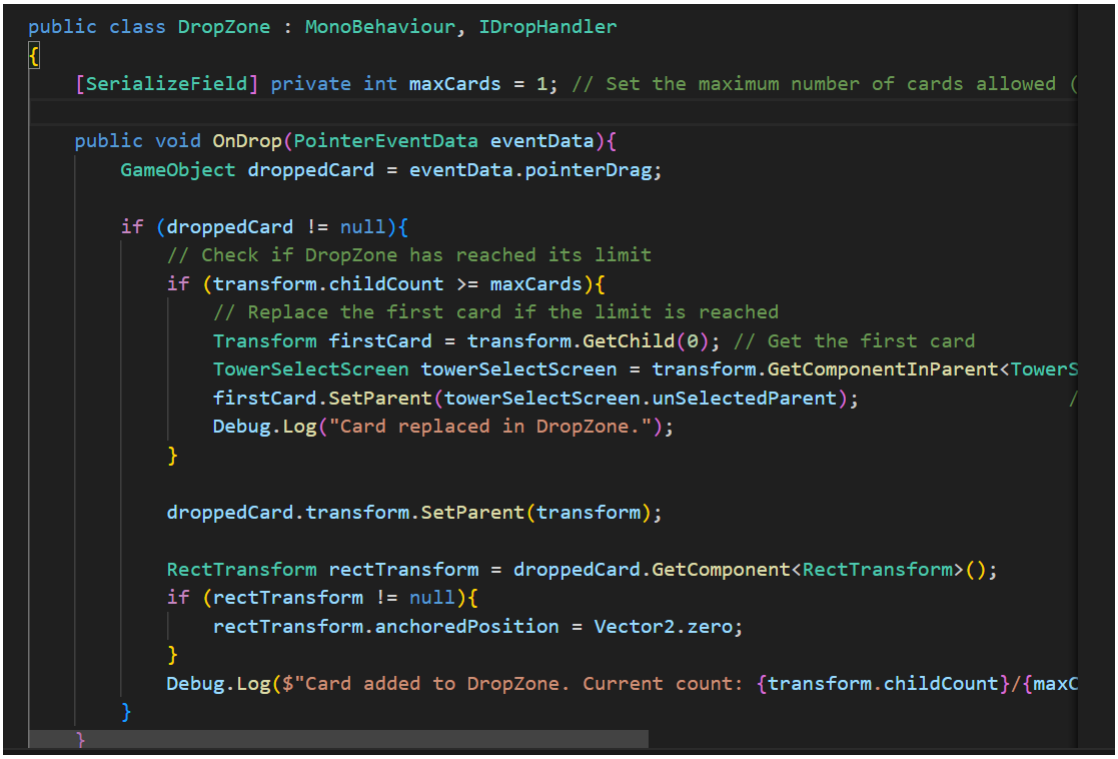


Figure 3.19: DropZone Script

```
12  public class TowerSelector : MonoBehaviour{
15
16      [Header("Buttons")]
17      public Button nextButton;
18      public Button prevButton;
19      public Button buyButton;
20
21
22      [Header("Texts")]
23      public TMP_Text nameText, maxHealthText, searchRateText,  fireRateText,  radiusText;
24      public TMP_Text IdleWaitTimeText, priceText, descText;
25
26      public TowerLibrary towerLib; // Reference to the library containing towers
27      private int index = 0; // Current tower index
28
29      public List<TowerItem> towers; // List of tower GameObjects in the scene
30      public float rotationSpeed = 1f;
31      // Start is called before the first frame update
32      void Start(){
33          nextButton.onClick.AddListener(NextTower);
34          prevButton.onClick.AddListener(PreviousTower);
35          buyButton.onClick.AddListener(Purchase);
36
37          if (towerLib == null || towerLib.Count == 0){
38              Debug.LogError("Tower Library is empty or not assigned.");
39              return;
40          }
41          DisplayTower(0);
42      }
43
```

Figure 3.20: TowerSelector Script Part 1

```
12  public class TowerSelector : MonoBehaviour{
44      // Update is called once per frame
45      void Update()
46      {
47          transform.Rotate(new Vector3(0f, 1f*Time.deltaTime*rotationSpeed, 0f));
48          if (Input.GetKeyDown(KeyCode.LeftArrow))
49          {
50              PreviousTower();
51          }
52          else if (Input.GetKeyDown(KeyCode.RightArrow))
53          {
54              NextTower();
55          }
56
57          if (Input.GetKeyDown(KeyCode.Return))
58          {
59              //SelectTower();
60          }
61      }
62
63      // Enable the tower at the given index and disable others
64      public void DisplayTower(int newIndex)
65      {
66          for (int i = 0; i < towers.Count; i++){
67              towers[i].towerPrefab.SetActive(i == newIndex);
68          }
69          UpdateUI();
70      }
```

Figure 3.21: TowerSelector Script Part 2

```
72      public void NextTower(){
73          index = (index + 1) % towers.Count;
74          DisplayTower(index);
75
76      }
77
78      public void PreviousTower(){
79          index = (index - 1 + towers.Count) % towers.Count;
80          DisplayTower(index);
81
82      }
83
84      public void Purchase(){
85          int currentCurrency;
86          GameManager.instance.GetCurrency(out currentCurrency);
87
88          if (currentCurrency >= towers[index].price){
89              GameManager.instance.UnlockTower(index);
90              GameManager.instance.SetCurrency(currentCurrency - towers[index].price, true);
91              Debug.Log("Purchased");
92          }
93
94          else{
95              Debug.Log("NOT ENOUGH MONEY");
96          }
97          UpdateUI();
98      }
```

Figure 3.22: TowerSelector Script Part 3

```
100     void UpdateUI(){
101
102         AttackAffector affector = towerLib.configurations[index].levels[0].GetComponentInCh
103         Targetter targetter = towerLib.configurations[index].levels[0].GetComponentInChildr
104
105         nameText.text = towerLib[index].towerName;
106         descText.text = towerLib[index].levels[0].description;
107         priceText.text = "Price:" + towers[index].price.ToString();
108         maxHealthText.text = towerLib[index].levels[0].maxHealth.ToString();
109
110         radiusText.text = targetter != null ? targetter.effectRadius.ToString() : "N/A";
111         searchRateText.text = affector != null ? affector.fireRate.ToString() : "N/A";
112         fireRateText.text = affector != null ? affector.fireRate.ToString() : "N/A";
113         IdleWaitTimeText.text = targetter != null ? targetter.idleWaitTime.ToString() : "N/
114
115         bool isUnlocked = GameManager.instance.IsTowerUnlocked(index);
116         if (isUnlocked){
117             lockImage.SetActive(false);
118             buyButton.gameObject.SetActive(false);
119             priceText.gameObject.SetActive(false);
120         }
121         else{
122             lockImage.SetActive(true);
123             buyButton.gameObject.SetActive(true);
124             priceText.gameObject.SetActive(true);
125         }
126
127     }
```

Figure 3.23: TowerSelector Script Part 4

Chapter 4

Game Outputs

4.1 Menu and UIs

Figure 4.1: Splash Screen

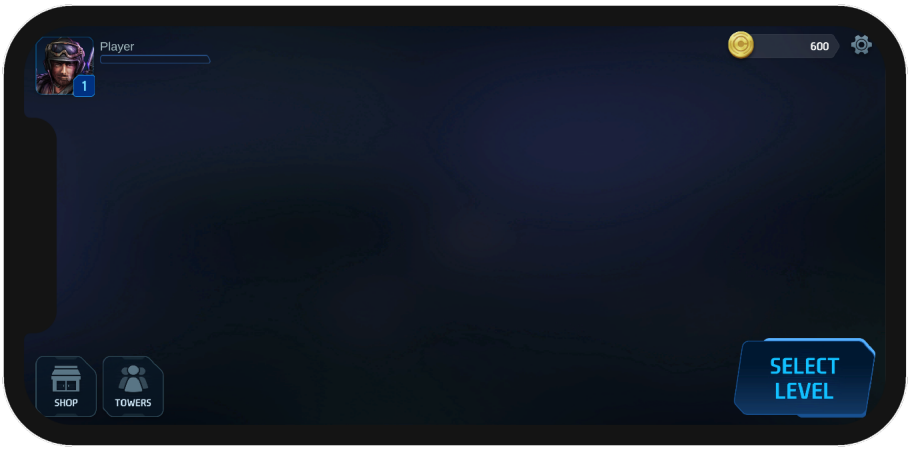


Figure 4.2: Main Menu Screen

4.2 Gameplay

Figure 4.3: Level Select Screen

Figure 4.4: Select Tower Screen

Figure 4.5: Settings Screen

Figure 4.6: Tower Shop Screen

Figure 4.7: Gameplay Screenshot 1

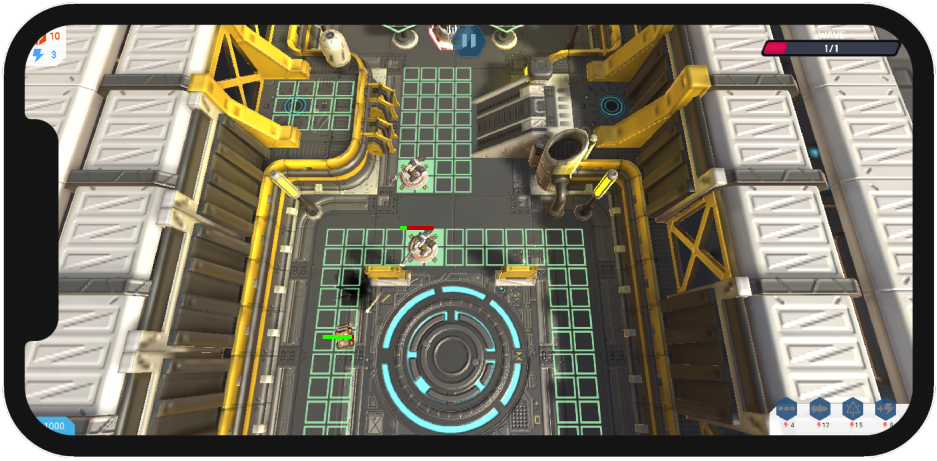


Figure 4.8: Gameplay Screenshot 2

Chapter 5

Technical Specifications

5.1 General Info

- **Engine:** Unity
- **Art Style:** Futuristic, sci-fi aesthetic with vibrant colors and sleek designs.
- **Audio:** Immersive sound effects and an original soundtrack to enhance the sci-fi atmosphere.
- **Platforms:** PC (Windows, macOS), Mobile (iOS, Android).

5.2 Topics Covered from subject

- C# Classes
- GameObject Instantiation
- Built-In Methods Like Update, Start, Awake etc
- Level Desig

5.3 Role of Each Member

- **Ali Asghar:** Main Lead Developer, responsible for designing the logic and game architecture.
- **Muhammad Sadeeq:** Co-Lead Developer, responsible for 3D environment and level designer.
- **Suleman Shah:** Level Designer, responsible for testing and making levels interesting.
- **Muhammad Shahab:** UI Designer, responsible for making interactive and eye-catching UI.

Chapter 6

Conclusion

This chapter gives the ending remarks and future work for our project.

6.1 Future Updates

- **Multiplayer Mode:** Co-op or competitive tower defense.
- **New Levels and Towers:** Expand the game with additional content.
- **Story Mode:** Introduce a narrative-driven campaign.

6.2 Final Remarks

Quantum Legacy is a visually stunning and strategically engaging tower defense game that combines sci-fi elements with addictive gameplay. With its unique towers, challenging levels, and robust progression system, it promises to captivate players and stand out in the tower defense genre.