Chapter 3

Some Important Code Screenshots

3.1 TowerLibrary.cs

```
TowerLibrary.cs ×   TowerSelectScreen.cs M   GameManager.cs
Ass  D:\GitHUb\TowerDefense_GD\Assets\UnityTechnologies\TowerDefenseTemplate\Scripts\Tower  ata > TowerLibrary.cs > ...
     Defense\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

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

Assets > UnityTechnologies > TowerDefenseTemplate > Scripts > TowerDefense > Towers > Data > TowerLevelData.cs > Tower

```
using UnityEngine;

namespace TowerDefense.Towers.Data
{
    /// <summary>
    /// Data container for settings per tower level
    /// </summary>
    [CreateAssetMenu(fileName = "TowerData.asset", menuName = "TowerDefense/Tower Configura
    1 reference
    public class TowerLevelData : ScriptableObject
    {
        /// <summary>
        /// A description of the tower for displaying on the UI
        /// </summary>
        1 reference
        public string description;

        /// <summary>
        /// A description of the tower for displaying on the UI
        /// </summary>
        1 reference
        public string upgradeDescription;

        /// <summary>
        /// The cost to upgrade to this level
        /// </summary>
        1 reference
        public int cost;

        /// <summary>
```

Figure 3.2: TowerLevelData Script Part 1

```
        1 reference
        public int cost;

        /// <summary>
        /// The sell cost of the tower
        /// </summary>
        1 reference
        public int sell;

        /// <summary>
        /// The max health
        /// </summary>
        1 reference
        public int maxHealth;

        /// <summary>
        /// The starting health
        /// </summary>
        1 reference
        public int startingHealth;

        /// <summary>
        /// The tower icon
        /// </summary>
        2 references
        public Sprite icon;
    }
}
```

Figure 3.3: TowerLevelData Script Part 2

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.

3.3 GameManager.cs

```
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
        public bool IsTowerUnlocked(int ind){
            return m_DataStore.IsTowerUnlocked(ind);
        }

        2 references
        public bool IsTowerSelected(int ind){
            return m_DataStore.isTowerSelected(ind);
        }
        /// <summary>
        /// Gets the stars earned on a given level
        /// </summary>
        /// <param name="levelId"></param>
        /// <returns></returns>
        2 references
        public int GetStarsForLevel(string levelId)
        {
            if (!levelList.ContainsKey(levelId))
            {
                Debug.LogWarningFormat("[GAME] Cannot check if level with id = {0} is comp
                return 0;
            }

            return m_DataStore.GetNumberOfStarForLevel(levelId);
        }
    }
}
```

Figure 3.8: GameManager Script Part 5

The script in figure [3.4,3.8] implements the GameManager for whole game and every level. This class contain code wrappers for implementing the core methods for towers and levels unlocking and selecting logic.

3.4 GameDataStore.cs

```
evelData.cs   GameDataStore.cs ×   Tower.cs   TowerSelectScreen.cs   GameManager.cs M
Assets > UnityTechnologies > TowerDefenseTemplate > Scripts > TowerDefense > Game > GameDataStore.cs
1   using System.Collections.Generic;
2   using Core.Data;
3   using UnityEngine;
4
5   namespace TowerDefense.Game
6   {
7       /// <summary>
8       /// The data store for TD
9       /// </summary>
10      public sealed class GameDataStore : GameDataStoreBase
11      {
12          /// <summary>
13          /// A list of level IDs for completed levels
14          /// </summary>
15          public List<LevelSaveData> completedLevels = new List<LevelSaveData>();
16
17          /// <summary>
18          /// A list of level IDs for completed levels
19          /// </summary>
20          public List<TowerSaveData> unlockedTowers = new List<TowerSaveData>();
21
22          /// <summary>
23          /// Outputs to debug
24          /// </summary>
25          public override void PreSave()
26          {
27              Debug.Log("[GAME] Saving Game");
28          }
```

Figure 3.9: GameDataStore Script Part 1

```
evelData.cs   GameDataStore.cs ×   Tower.cs   TowerSelectScreen.cs   GameManager.cs M
Assets > UnityTechnologies > TowerDefenseTemplate > Scripts > TowerDefense > Game > GameDataStore.cs
6   {
11      {
30          /// <summary>
31          /// Outputs to debug
32          /// </summary>
33          public override void PostLoad()
34          {
35              Debug.Log("[GAME] Loaded Game");
36          }
37
38          /// <summary>
39          /// Marks a level complete
40          /// </summary>
41          /// <param name="levelId">The levelId to mark as complete</param>
42          /// <param name="starsEarned">Stars earned</param>
43          public void CompleteLevel(string levelId, int starsEarned)
44          {
45              foreach (LevelSaveData level in completedLevels)
46              {
47                  if (level.id == levelId)
48                  {
49                      level.numberOfStars = Mathf.Max(level.numberOfStars, starsEarned);
50                      return;
51                  }
52              }
53              completedLevels.Add(new LevelSaveData(levelId, starsEarned));
54          }
55
56          /// <summary>
```

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

```
LevelData.cs   GameDataStore.cs M ×   Tower.cs   TowerSelectScreen.cs   GameManager.c
Assets > UnityTechnologies > TowerDefenseTemplate > Scripts > TowerDefense > Game > GameDataStore.cs
 6  {
11      {
79          {
81              {
83                  {
84                      //level.numberOfStars = Mathf.Max(level.numberOfStars, starsEarn
85                      return;
86                  }
87              }
88              unlockedTowers.Add(new TowerSaveData(ind));
89          }
90
91          /// <summary>
92          /// Determines if a specific tower is unlocked
93          /// </summary>
94          /// <param name="levelId">The level ID to check</param>
95          /// <returns>true if the level is completed</returns>
96          public bool IsTowerUnlocked(int ind)
97          {
98              foreach (TowerSaveData tower in unlockedTowers)
99              {
100                 if (tower.index == ind)
101                 {
102                     return true;
103                 }
104             }
105             return false;
106         }
107
```

Figure 3.12: GameDataStore Script Part 4

```
109         /// Select a tower
110         /// </summary>
111         public void SelectTower(int ind)
112         {
113             foreach (TowerSaveData tower in unlockedTowers)
114             {
115                 if (tower.index == ind)
116                 {
117                     tower.isSelected = true;
118                     return;
119                 }
120             }
121         }
122
123         /// <summary>
124         /// Marks all towers as unselected
125         /// </summary>
126         public void DeSelectAllTowers()
127         {
128         foreach (TowerSaveData tower in unlockedTowers)
129             {
130                 tower.isSelected = false;
131             }
132
133         }
134
```

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     }
```

Figure 3.14: GameDataStore Script Part 6

The script in figure [3.9 - 3.14] is a Data Storage Container and contains the implementation of the levels and towers unlocking and selecting. The data is saved in a file in persistent path of the device.

3.5 LevelSaveData.cs

```
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

The script in figure [3.15] is a serialized class for saving level data.

3.6 TowerSaveData.cs

```
evelData.cs    TowerSaveData.cs ×    GameDataStore.cs M
Assets > UnityTechnologies > TowerDefenseTemplate > Scripts > Tower
using System;

namespace TowerDefense.Game
{
    /// <summary>
    /// A calss to save level data
    /// </summary>
    [Serializable]
    public class TowerSaveData
    {
        public int index;
        public bool isSelected;

        public TowerSaveData(int ind)
        {
            index = ind;
        }
    }
}
```

Figure 3.16: TowerSaveData Script

The script in figure [3.16] is a serialized class for saving tower data.

3.7 CardDragHandler.cs

```
Affector.cs   LevelItem.cs   LevelList.cs   LevelManager.cs   CardDragHandler.cs M
Assets > MyScripts > CardDragHandler.cs

4   public class CardDragHandler : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHand
5       private Canvas canvas;
6       private RectTransform rectTransform;
7       private CanvasGroup canvasGroup;
8       public Vector2 originalPosition;
9
10      private void Awake(){
11          canvas = GetComponentInParent<Canvas>(); // Get the canvas the card belongs to
12          rectTransform = GetComponent<RectTransform>();
13          canvasGroup = GetComponent<CanvasGroup>();
14      }
15
16      public void OnBeginDrag(PointerEventData eventData){
17          originalPosition = rectTransform.anchoredPosition;
18          canvasGroup.alpha = 0.8f;
19          canvasGroup.blocksRaycasts = false;
20      }
21
22      public void OnDrag(PointerEventData eventData){
23          rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;
24      }
25
26      public void OnEndDrag(PointerEventData eventData){
27          canvasGroup.alpha = 1f;
28          canvasGroup.blocksRaycasts = true;
29          rectTransform.anchoredPosition = originalPosition;
30      }
31  }
32
```

Figure 3.18: CardDragHandler Script

The script in figure [3.18] contains the implementation of unity built-in methods for drag and drop.

3.8 DropZone.cs

```
public class DropZone : MonoBehaviour, IDropHandler
{
    [SerializeField] private int maxCards = 1; // Set the maximum number of cards allowed (

    public void OnDrop(PointerEventData eventData){
        GameObject droppedCard = eventData.pointerDrag;

        if (droppedCard != null){
            // Check if DropZone has reached its limit
            if (transform.childCount >= maxCards){
                // Replace the first card if the limit is reached
                Transform firstCard = transform.GetChild(0); // Get the first card
                TowerSelectScreen towerSelectScreen = transform.GetComponentInParent<TowerS
                firstCard.SetParent(towerSelectScreen.unSelectedParent);                    /
                Debug.Log("Card replaced in DropZone.");
            }

            droppedCard.transform.SetParent(transform);

            RectTransform rectTransform = droppedCard.GetComponent<RectTransform>();
            if (rectTransform != null){
                rectTransform.anchoredPosition = Vector2.zero;
            }
            Debug.Log($"Card added to DropZone. Current count: {transform.childCount}/{maxC
        }
    }
```

Figure 3.19: DropZone Script

The script in figure [3.19] creates a drop zone for the draggable object.

3.9 TowerSelector.cs

```
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      }
```

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

The script in figure [3.20 – 3.23] implements the tower shop. It contain methods for selecting and buying different towers.