IŞIK UNIVERSITY
COMPUTER
SCIENCE AND
ENGINEERING

Survival Game Design and Implementation

Bachelor's Thesis

Yaren Çay
20SOFT1007

Supervised by
Ahmet Feyzi Ateş

January 2020

ABSTRACT

This thesis addresses the challenge of designing and implementing an immersive survival game, named "Big Picture". The central problem revolves around creating a game that not only provides a compelling narrative and emotional depth but also introduces innovative gameplay elements, specifically in the areas of crafting and technical architecture.

In response to this challenge, the solution approach involves crafting a unique storyline where players navigate a post-apocalyptic world following a zombie attack, emphasizing a personal motivation for survival. The crafting system within the game goes beyond conventional approaches, enabling players not only to create survival essentials but also contribute to the reconstruction of the protagonist's house, adding layers of creativity and personalization. Additionally, the integration of the StrangeIoC (StrangeIoC) framework provides a robust foundation for modular development, enhancing code organization, maintainability, and scalability.

Results obtained from the implementation of "Big Picture" showcase a distinctive gaming experience with narrative depth and engaging gameplay. Through a literature survey, this thesis explores existing challenges in survival game design and related technical architectures. The comparison of the proposed solution with existing games in the genre highlights the innovation brought by "Big Picture," particularly in its crafting system and technical foundation.

ACKNOWLEDGEMENTS

A heartfelt appreciation to my dedicated advisor, Ahmet Feyzi Ateş, for providing invaluable guidance and regular support throughout the development of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The primary purpose of the 'Big Picture' survival game is to provide players with an engaging and emotionally resonant experience. The narrative unfolds in a post-apocalyptic world infested with chromozombies, genetically engineered creatures that pose a constant threat. The core objective is to survive this menace and protect the protagonist's home.

The subsequent chapters delve into specific aspects of the project. The introduction provides context and motivation for the game's development. The literature survey reviews existing survival games, emphasizing the unique features introduced in "Big Picture." The design and implementation chapter details the technical architecture, crafting system, and integration of the StrangeIoC (StrangeIoC) framework. Results present an analysis of player experiences and feedback, comparing them with industry benchmarks. The conclusion summarizes key findings, discusses implications, and outlines directions for future enhancements to "Big Picture." This document serves as a comprehensive guide, illustrating the formatting and content for each chapter, including the incorporation of tables, figures, and references.

1.1. Problem Definition

The challenge at the core of the 'Big Picture' survival game project lies in crafting an immersive and emotionally resonant gaming experience within a post-apocalyptic narrative. Envision a world haunted by chromozombies, the result of experimental genetic engineering, where players navigate for survival following a personal tragedy—the loss of a loved one in a chromozombie onslaught. The problem extends beyond technical intricacies to encompass the art of creating an engaging adventure. Balancing diverse gameplay mechanics, including resource collection, crafting, combat, and house defense, is central to providing players with a seamless and enjoyable experience. Aligning project objectives with player expectations, as reflected in the success criteria, involves not only defeating chromozombies but ensuring players thoroughly enjoy and engage with the game. The technical backbone, incorporating the Unity Engine (Unity), MySQL, and the StrageIoC (StrangeIoC) framework, presents its own set of challenges in harmonizing code and storytelling.

1.1.1. Unity Engine

The survival game project incorporates key components of the Unity Engine (Unity), including graphics rendering, physics engine, and C# scripting. This integration enables the

project to utilize Unity's advanced graphics rendering capabilities, creating a visually immersive post-apocalyptic world. The Unity physics engine contributes to realistic character movements, enhancing the overall gaming experience. Additionally, some features of the Unity Engine, such as Scriptable Objects, NavMesh, and Shader Graphs, are utilized.

The usage of Scriptable Objects (Scriptable Objects) serves as a foundational element, providing a solution for data management and memory efficiency. In essence, a Scriptable Object acts as a data container, decoupled from class instances, and excels in storing substantial amounts of data.

The incorporation of (AI NavMesh) technology further enhances the project by enabling realistic character navigation within the game environment. A NavMesh, designated within the Unity scene, specifies navigable areas, including paths for characters and obstacle locations. This functionality proves particularly useful for scenarios involving pathfinding and AI-controlled navigation.

Furthermore, (Shader Graphs), another integral part of the Unity Engine, serves as a tool for visually building shaders. Rather than writing code, users can create and connect nodes in a graph framework, receiving instant feedback that reflects their changes. This feature is especially beneficial for users who are new to shader creation, providing a user-friendly approach to enhance the visual elements of the game.

1.1.2. StrangeIoC Framework

(StrangeIoC) is a framework for Unity game development. StrangeIoC stands for "Strange Inversion of Control" and is an open-source framework designed to implement the principles of Inversion of Control (IoC) and Dependency Injection in Unity projects. The IoC design pattern aims to invert the flow of control in a system, where the framework is responsible for managing the lifecycle of objects and their dependencies.

In the context of StrangeIoC, the framework embraces Inversion of Control (IoC) and Dependency Injection in Unity game development. It manages component relationships, delegates control over object creation, and supports Dependency Injection for flexible, maintainable systems. With an event-driven model and a modular approach, StrangeIoC fosters loosely coupled architectures, easing extension and modification. When using StrangeIoC in a Unity project, you define dependencies, and the framework handles object instantiation and relationships seamlessly.

2. LITERATURE REVIEW

2.1. Similar Applications

2.1.1. The Last of Us

(The Last of Us) is a third-person action-adventure with stealth and survival elements. Players scavenge for resources, craft weapons and tools, and engage in intense combat against both Infected and human enemies.

The integration of the StrangeIoC framework in "Big Picture" contributes to its technical architecture. This framework, focusing on Inversion of Control (IoC) and Dependency Injection, fosters modular development and code organization. This approach enhances the maintainability and scalability of the game, setting it apart in terms of technical architecture.

In comparison, "Big Picture" sets itself apart by prioritizing emotional depth and storytelling, intertwining a captivating narrative with diverse gameplay mechanics. The protagonist's personal tragedy, losing a loved one in a Chromozombie attack, introduces a unique and emotionally charged journey. This focus on narrative innovation distinguishes "Big Picture" from traditional survival games.

While "The Last of Us" excels in its narrative-driven approach and survival mechanics, "Big Picture" seeks to complement these aspects with innovative gameplay features and a robust technical foundation. The blend of emotional storytelling, diverse gameplay mechanics, and technical excellence positions "Big Picture" as a unique and promising project in the survival game genre.

2.1.2. Dying Light

(Dying Light) is a first-person action game where you navigate a zombie-infested city during the day and night, using parkour and crafting to survive.

"Dying Light" is celebrated for its innovative integration of parkour elements, allowing players to navigate the zombie-infested environment with fluidity and creativity. In contrast, "Big Picture" distinguishes itself by incorporating a more strategic and emotionally charged traversal system, emphasizing the protagonist's personal journey through the post-apocalyptic world.

In conclusion, this literature review highlights the unique gameplay mechanics, narrative elements, and technological innovations employed by "Dying Light" and "Big Picture." While

"Dying Light" excels in parkour-driven traversal and moral choices, "Big Picture" focuses on emotional depth and a strategic traversal system within a post-apocalyptic narrative.

2.1.3. The Forest

(The Forest) A top-down survival game with roguelike elements, where you must manage hunger, sanity, and the environment to survive in a strange and unpredictable world.

"The Forest" and "Big Picture" both belong to the survival game genre, offering players immersive experiences in post-apocalyptic settings. In "The Forest," players find themselves stranded on an island inhabited by mutants, focusing on crafting, building, and survival against both environmental challenges and hostile creatures. On the other hand, "Big Picture" introduces a unique narrative layer by incorporating emotional depth, with the protagonist navigating a world infested by chromozombies after the loss of a loved one.

In terms of gameplay mechanics, both games feature crafting systems, emphasizing resource management and construction.

In conclusion, while "The Forest" and "Big Picture" share common elements as survival games, each brings a unique flavor to the genre. "Big Picture" stands out with its emphasis on emotional storytelling, innovative crafting mechanics, and the integration of the StrangeIoC framework, promising players a distinctive and evolving gaming experience.

2.1.4. Comparison of Similar Applications

Table 2.1. Comparison of similar applications

	Big Picture	The Last of Us	Dying Light	The Forest
Crafting	✓	✓	✓	
Single Player	✓	✓		
Post-apocalyptic world	✓	✓	✓	
Combat Mechanism	✓	✓	✓	✓

2.2. A* Pathfinding Algorithm

The (A* Pathfinding Algorithm) by navigating through a network of nodes, forming a graph that represents possible paths within the given space. It relies on heuristic functions, particularly h(n), which estimates the cost from a designated node to the goal. The algorithm systematically assesses nodes based on cost functions, with f(n) representing the total cost, consisting of g(n) (the cost from the start node) and h(n). Its iterative approach involves managing Open and Closed Sets, prioritizing nodes with the lowest f(n) for exploration, and dynamically adjusting paths for optimal efficiency. A* is known for ensuring optimality under certain conditions, guaranteeing the identification of the most efficient path between two points. Its practicality and adaptability have made it a common choice in diverse applications, notably in gaming, where it facilitates intelligent character movement within virtual landscapes.

3. PROPOSED SYSTEM

3.1. Introduction

Traditional survival games often prioritize gameplay mechanics over emotional depth and storytelling, creating a gap in player immersion. "Big Picture" aims to address this by intertwining compelling narrative elements with gameplay mechanics. The protagonist, who tragically lost his wife in a Chromozombie attack, sets the stage for a unique and emotionally charged journey, distinguishing our game from others in the genre. Engineered by scientists, Chromozombies are an invasive species that demands strategic thinking and adaptability from players.

“Big Picture” adopts a Model-View-Controller (MVC) architecture. This architecture provides a separation of concerns, enhancing code organization, maintainability, and scalability. The MVC structure is implemented through the integration of the StrangeIoC framework, fostering modular development and dependency injection.

3.2. Graphical User Interface

Figure 3.1 Main Game UI

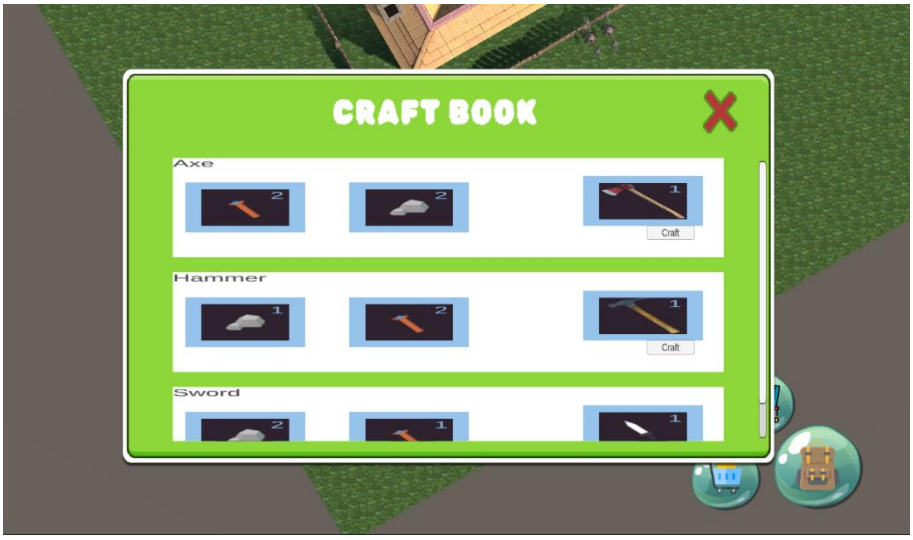

Figure 3.2 CraftBook UI

Figure 3.3 MarketPlace UI

3.3. Business Logic

The business logic encompasses a network of processes and rules, orchestrating the underlying operations that drive the immersive gaming experience. This section delves into the core components and functionalities of the Business Logic, unraveling the intricacies that contribute to the dynamic post-apocalyptic world players navigate.

3.3.1. Game Logic Subsystem

In the game, Chromozombies rely on a Navigation Mesh (NavMesh) system for their movement patterns, ensuring dynamic and realistic navigation while avoiding obstacles. On the other hand, players utilize the A* pathfinding algorithm to navigate through the post-apocalyptic world efficiently. The NavMesh guides Chromozombies by providing a structured map of navigable areas, allowing them to adapt to changes in the environment intelligently. Simultaneously, the A* algorithm serves as a powerful tool for players, calculating optimal paths based on factors like distance, terrain difficulty, and obstacles.

3.3.2. Inventory System

The subsystem excels in efficiently organizing various items, including weapons and tools. To enhance the survival aspect of the game, the Inventory System dynamically tracks the quantity and condition of resources, ensuring that players stay informed about their inventory status, influencing strategic decision-making.

3.3.3. Crafting System

The Crafting System in "Big Picture" adds depth and strategy, allowing players to shape their survival journey through item creation. This subsystem seamlessly integrates with the Inventory System, offering a practical crafting experience. A straightforward crafting interface serves as the gateway, presenting players with a range of craftable items. This interface provides details on required materials, tools, and potential outcomes. Crafting involves a thoughtful process, checking the player's inventory for necessary components. Resources are consumed upon initiation, and the crafted item integrates into the player's inventory. Crafting introduces a strategic layer to gameplay, requiring players to explore the game world, gather resources, and make decisions about what to craft. The system promotes a dynamic and adaptive approach, making crafting a crucial aspect of survival in the post-apocalyptic environment.

3.3.4. Combat System

At the center of the player's survival experience is the Combat System, a subsystem overseeing in-game combat, from encounters with Chromozombies to house defense. Orchestrating Chromozombie dynamics, it dictates their movement patterns and responses to in-game events, creating unpredictable encounters. Intuitive controls empower players to fluidly engage in combat, from evading zombie attacks to strategic house defense. The system manages game progression, introducing challenges and events for player engagement. Combat scenarios are woven into the narrative, offering evolving challenges. Together, these subsystems contribute to the gaming experience in the game, making players active participants in the post-apocalyptic world.

3.4. Data Management

(Scriptable Objects) serve as a practical tool for managing data within the Unity framework. They offer advantages in handling specific game-related information by allowing the creation of custom data structures. Scriptable Objects provide flexibility, easy serialization, and storage as assets within the Unity Editor, facilitating convenient modification and version control.

I utilized scriptable objects to store essential data such as market items, craft recipes, and inventory items in the game.

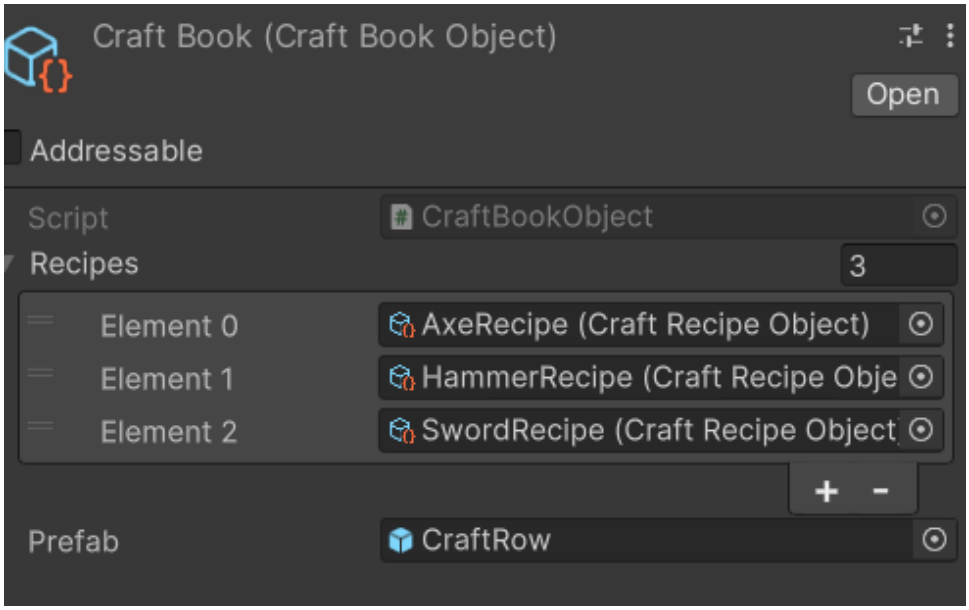

Figure 3.4 Craft Book Scriptable Object

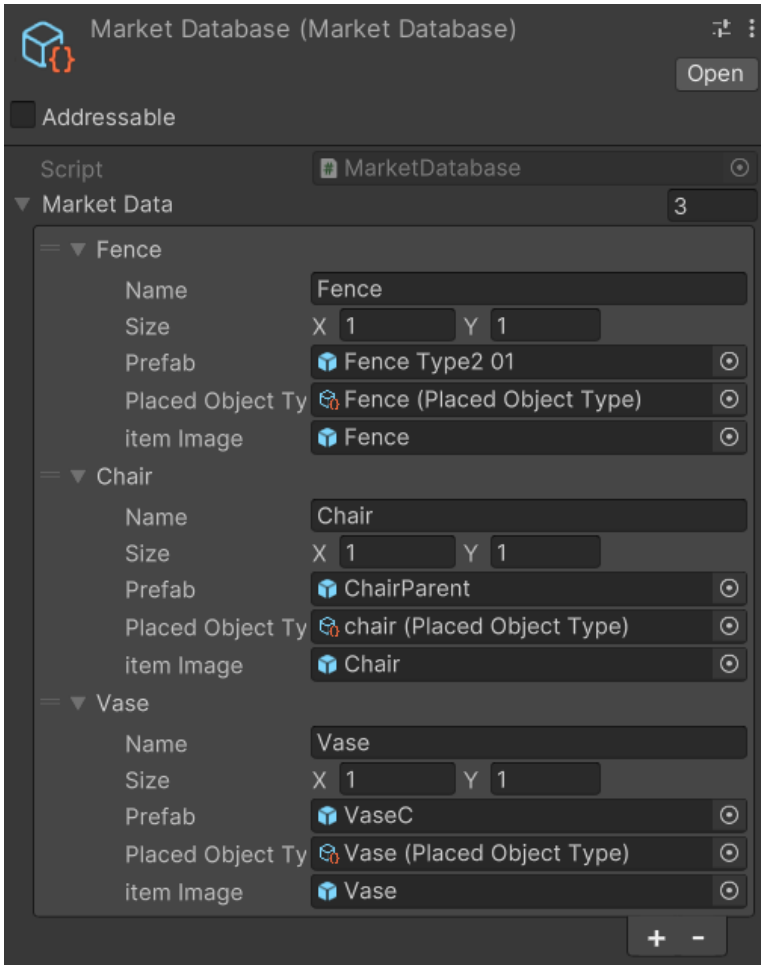

Figure 3.5 Market Database Scriptable Object

Scriptable objects prove useful for static or frequently accessed data, contributing to reduced memory overhead. However, it's essential to note that they are not a direct replacement for traditional databases, especially in scenarios involving dynamic, server-managed information. For relational data models or complex queries, traditional databases.

For database processes, MySQL is employed in "Big Picture." Two tables are dedicated to inventory storage: the "itemtypes" table, encompassing item names and their corresponding IDs, and the "inventoryitems" table, featuring columns for ID, itemId, and amount. Updates primarily occur in the "inventoryitems" table. To streamline the adjustment of amounts, a join operation is performed between these two tables, linking them based on the "itemtypes.id" and "inventoryitems.itemId." This approach ensures a simplified and efficient process for updating item quantities.

id	name
0	Stone
1	Stick
2	Sword
3	Axe
4	Hammer

Figure 3.6 itemtypes database table

id	itemId	amount
0	0	13
1	1	8
2	2	4
3	3	7
4	4	8

Figure 3.7 inventoryitems database table

There is one additional table for storing player money data. This table is named "moneyData" and features columns for ID and amount. It is specifically designed to store player's current money balance.

id	amount
0	470

Figure 3.8 moneyData database table

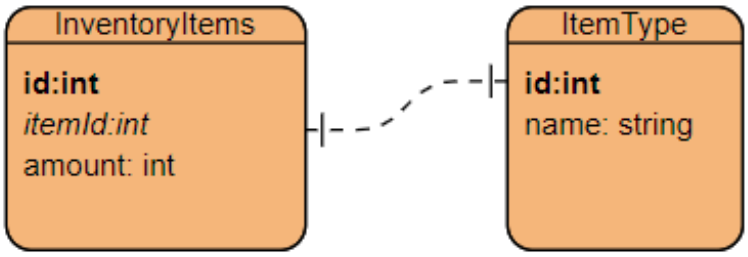

Figure 3.9 Inventory management ER Diagram

4. IMPLEMENTATION, TESTS, EXPERIMENTS

4.1. Implementation

The implementation of the survival game involved a meticulous selection of technologies, libraries, and frameworks to bring the envisioned software architecture to life. Throughout the development process, several essential libraries were integrated to optimize functionality and enhance workflow.

TextMeshPro (TextMeshPro) (TMP) proved to be a valuable addition, significantly enhancing text rendering capabilities and contributing to an improved user interface, fostering a more engaging experience for players. Addressables played a pivotal role in efficient asset management, especially during critical gameplay transitions, ensuring a seamless experience for players.

The Promise (Promises) library was strategically incorporated to manage asynchronous tasks, thereby enhancing project efficiency and simplifying intricate workflows.

In addition, the integration of the StrangeIoC framework played a crucial role in facilitating the Model-View-Controller (MVC) architecture. This framework ensured a clear separation of concerns, promoting efficient management of data, presentation, and user interactions.

In summary, the deliberate integration of TextMeshPro, Addressables (Addressables), Promise, and StrangeIoC collectively contributes to a more responsive, resource-efficient, and visually immersive gaming experience within the survival game. Each library, thoughtfully chosen and implemented, plays a crucial role in shaping the game's development, ensuring a seamless experience for players navigating the post-apocalyptic world.

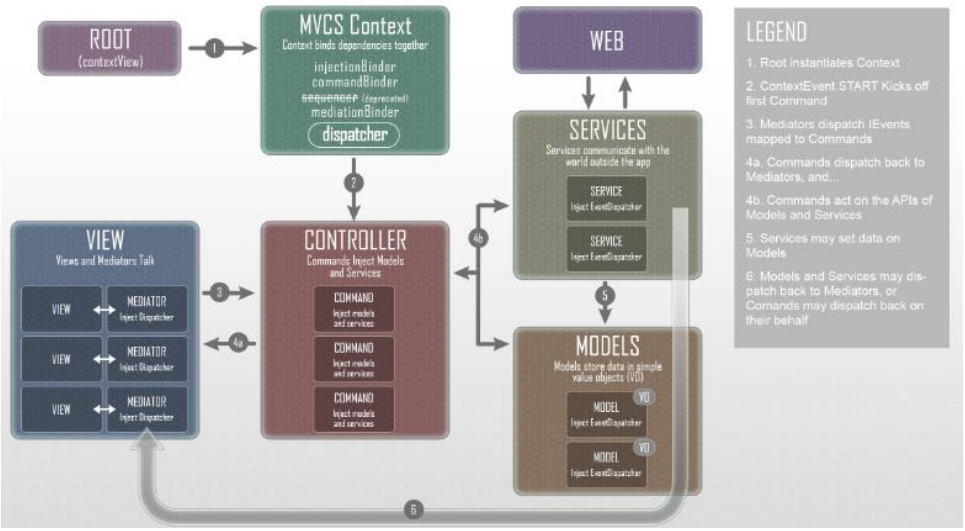

Figure 4.1 StrangeIoC MVCS Context Architecture

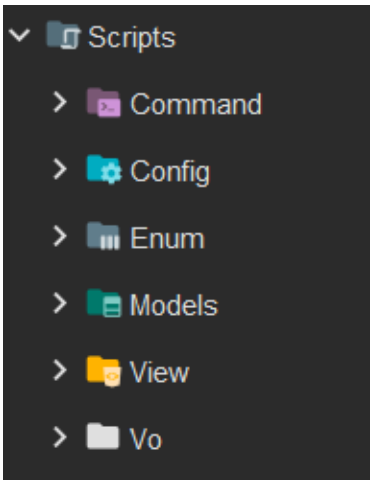

Figure 4.2 Scripts Folder Organization

In the commands folder, InitGameCommand class is placed. It derived from EventCommand. An EventCommand is a specific type of command that is executed in response to an event. When an event is dispatched, the associated EventCommand is automatically triggered.

In the config folder, GameBootstrap and GameContext is placed. GameContext class represents the main context for the game, where we define and map the bindings between interfaces and concrete implementations. It extends MVCSContext, which is part of the StrangeIoC framework.

Figure 4.3 GameBootstrap Class

The GameBootstrap class serves as the entry point for the game, extending ContextView, a component integral to StrangeIoC. Within the 'enum' folder, you'll find classes defining enums for game layers and panels, with frequently used ones centralized in this directory.

As the name implies, model classes and interfaces reside in the 'model' folder, while views and mediators are organized under the 'View' folder. View represents the user interface in software, displaying data and handling user input. In MVC architecture, it focuses on presenting information and communicates with the Model and Controller. Mediator is a behavioral pattern that centralizes communication between components, promoting loose coupling. The Mediator object manages interactions between components without them needing direct awareness of each other.

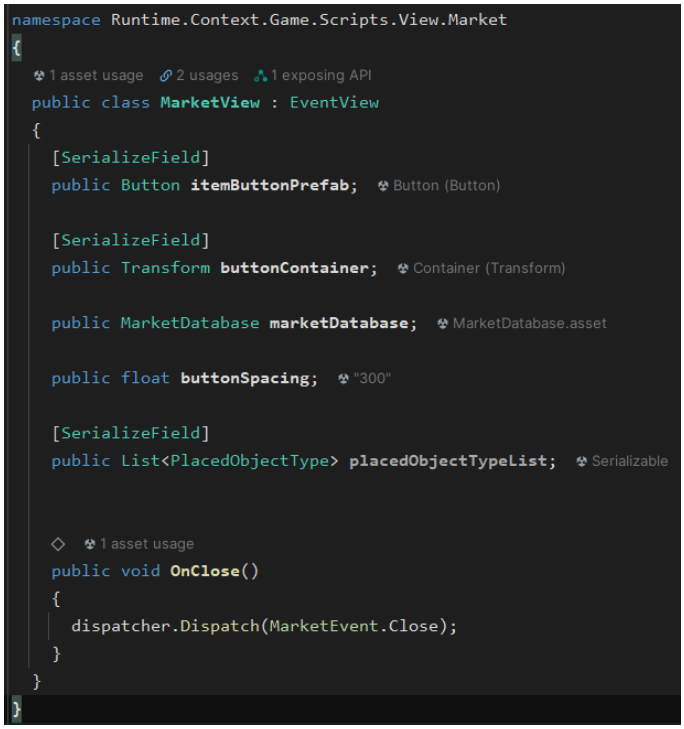

Figure 4.4 MarketView class

```
public class MarketMediator : EventMediator
{
  [Inject]
  9 usages
  public MarketView view { get; set; }

  private List<Button> _itemButtons;
  public PlayerData player;

  private PlacedObjectType currentSelectedType;

  [Inject]
  Frequently called  3 usages
  public IGridModel gridModel { get; set; }

  [Inject]
  1 usage
  public ICameraModel cameraModel { get; set; }

  private PlacedObjectType.Dir _dir = PlacedObjectType.Dir.Down;

  public Camera sceneCamera;  Unchanged

  ◇  0+1 usages
  public override void OnRegister()
  {
    player = new PlayerData();
    player.AddMoney(amount: 200f);
    view.dispatcher.AddListener(evt: MarketEvent.Close, OnClose);
    _itemButtons = new List<Button>();
    PopulateMarketButtons();
    sceneCamera = cameraModel.GetCameraByKey("1");
  }
```

Figure 4.5 MarketMediator class

```
public List<NodeVo> FindPath(int startX, int startY, int endX, int endY)
{
  NodeVo startNode = gridModel.GetGridObject(startX, startY);
  NodeVo endNode = gridModel.GetGridObject(endX, endY);
  _openList = new List<NodeVo> {startNode};
  _closedList = new List<NodeVo>();
  for (int x = 0; x < gridModel.GetWidth(); x++)
  {
    for (int y = 0; y < gridModel.GetHeight(); y++)
    {
      NodeVo pathNode = gridModel.GetGridObject(x, y);
      pathNode.gCost = int.MaxValue;
      pathNode.cameFromNode = null;
    }
  }

  startNode.gCost = 0;
  startNode.hCost = CalculateDistanceCost(a: startNode, b: endNode);
  Debug.Log(message: "hcost: " + startNode.hCost);

  while (_openList.Count > 0)
  {
    NodeVo currentNode = _openList[0];
    if (currentNode == endNode)
    {
      return CalculatePath(endNode);
    }
```

Figure 4.6 Pathfinding Service (Model) Find Path Function

```
namespace Runtime.Context.Game.Scripts.Models.Pathfinding
{
    4 usages  1 inheritor  2 exposing APIs
    public interface IPathfindingService
    {
        2 usages  1 implementation
        List<NodeVo> FindPath(int startX, int startY, int endX, int endY);
    }
}
```

Figure 4.7 Pathfinding Service (Model) Interface

```
using UnityEngine;

namespace Runtime.Context.Game.Scripts.Models.ItemObjects
{
    15 usages  8 exposing APIs
    public enum ItemType
    {
        Default,
        Wrench,
        Stone,
        Stick,
        Axe,
        Hammer,
        Sword
    }

    No asset usages  40 usages  7 inheritors  8 exposing APIs
    public abstract class ItemObject : ScriptableObject
    {
        public GameObject prefab;  Changed in 0+ assets
        public ItemType type;  Changed in 0+ assets
        public string itemName;  Changed in 0+ assets
        public int damage;  Changed in 0+ assets

        [TextArea(15, 20)]
        public string description;  Changed in 0+ assets

        public bool isWeapon;  Changed in 0+ assets
        public GameObject weaponButton;  Changed in 0+ assets
    }
}
```

Figure 4.8 ItemObject Scriptable Object

"Vo" typically stands for "Value Object." Value Objects (ValueObject) are a concept commonly used in software development to represent objects whose equality is based on the value of their attributes rather than their identity.

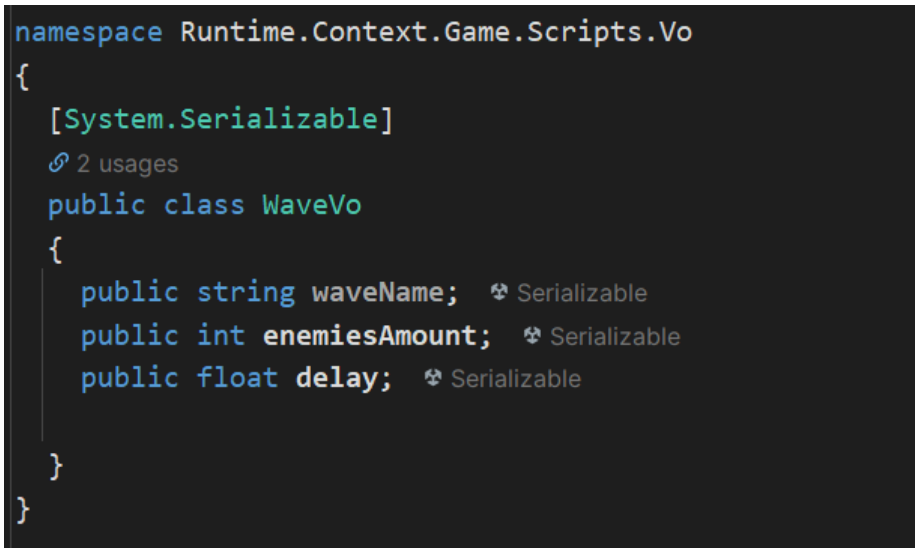

Figure 4.9 WaveVo class

4.2. Tests

Testing is a critical phase in the software development lifecycle, integral to ensuring the robustness and reliability of the "Big Picture" survival game. This dynamic process evolves alongside development, with additional scenarios identified as the system matures, prompting an expansion of the testing suite for comprehensive coverage. Adopting an iterative and comprehensive approach, the developer personally conducted tests in alignment with industry best practices, fostering continuous improvement in the game's stability and functionality.

The software testing phase encompassed a meticulous evaluation of key components to guarantee a seamless and immersive gaming experience. Specific test cases were designed for various modules, including the User Interface (UI), Game Logic, Inventory Management, Database Integration, Crafting System, and Combat System.

- UI Testing:
    - Scenario 1: Verify responsiveness and functionality of in-game menus.
    - Scenario 2: Ensure seamless navigation through different UI screens.
    - Scenario 3: Validate user interactions for smooth gameplay experience.

- Game Logic Testing:
    - Scenario 1: Validate core rules governing player actions.
    - Scenario 2: Test Chromozombie behavior and movement patterns.
    - Scenario 3: Verify game progression mechanics and events.
- Inventory Management Testing:
    - Scenario 1: Ensure efficient organization of collected items and resources.
    - Scenario 2: Validate crafting mechanics for creating weapons and tools.
    - Scenario 3: Test inventory management for its impact on survival and house defense.
- Database Integration Testing:
    - Scenario 1: Verify seamless communication with the MySQL database.
    - Scenario 2: Test persistence of player progress and inventory data.
    - Scenario 3: Validate data retrieval across gaming sessions.
- Crafting System Testing:
    - Scenario 1: Validate the crafting interface for clarity and usability.
    - Scenario 2: Test crafting process for required materials and outcomes.
    - Scenario 3: Ensure dynamic and adaptive crafting approach.
- Combat System Testing:
    - Scenario 1: Validate intuitive controls for engaging in combat.
    - Scenario 2: Test Chromozombie encounters for unpredictability.
    - Scenario 3: Verify integration of combat scenarios into the evolving narrative.

It's important to note that, throughout this testing phase, no specific testing frameworks were employed. Each test scenario was manually executed, ensuring alignment with the envisioned gameplay and providing valuable insights for further refinement.

5. CONCLUSIONS AND FUTURE WORK

This survival game project represents a noteworthy achievement, successfully delivering an immersive post-apocalyptic gaming adventure. The project effectively realized its design goals, seamlessly integrating diverse gameplay mechanics, compelling storytelling, and achieving optimal technical performance. Key technologies, such as Unity Engine, Scriptable Objects, NavMesh, and Shader Graphs, played crucial roles in bringing the envisioned gaming experience to life.

Despite the overall success, the project faced challenges, particularly in managing technical dependencies on third-party software and the necessity for rigorous testing procedures. Proactive identification and mitigation of these challenges through alternative software solutions and comprehensive testing strategies have significantly contributed to the project's overall resilience and robustness.

Looking toward the future, the 'Big Picture' survival game project serves as a foundation for potential extensions and enhancements. Future iterations may explore the integration of multiplayer capabilities, enabling players to collaborate or compete in the expansive post-apocalyptic world. The addition of new narrative arcs, characters, and environments stands as an opportunity to further enrich the gaming experience, offering players an expanded and evolving storyline to engage with.

The crafting and customization aspects of the game open doors for future enhancements, with possibilities including more intricate crafting systems, a broader range of items, and personalized base-building options. Exploring advancements in graphics technology and incorporating more sophisticated visual effects can contribute to elevating the immersive qualities of the game, providing players with a visually stunning and captivating experience.

In conclusion, the 'Big Picture' survival game project not only achieves its initial objectives but also establishes a foundation for a dynamic and evolving gaming experience. The seamless integration of technical excellence and creative vision positions the project for future innovations, ensuring that players can continue to explore and engage with the captivating post-apocalyptic world in ever-evolving ways. The project's success and adaptability make it well-poised for continued growth and enhancement in the dynamic landscape of gaming.

6. REFERENCES

Unity Engine. n.d. 12 January 2024.< https://unity.com/>.

Promises. n.d. 12 January 2024.< https://www.the-data-wrangler.com/promises-for-game-development/ >.

TextMeshPro . n.d. 12 January 2024.< https://docs.unity3d.com/Manual/com.unity.textmeshpro.html>.

Adressables. N.d. 12 January 2024. < https://docs.unity3d.com/Manual/com.unity.addressables.html >.

Value Object. n.d. 12 January 2024.< https://martinfowler.com/bliki/ValueObject.html >.

Unity Documentation AI NavMesh. n.d. 12 January 2024.<https://docs.unity3d.com/ScriptReference/AI.NavMesh.html >.

Unity Documentation Shader Graphs. n.d. 12 January 2024.< https://docs.unity3d.com/Manual/shader-graph.html >.

Unity Documentation Scriptable Objects. n.d. 12 January 2024.<https://docs.unity3d.com/Manual/class-ScriptableObject.html>.

StrangeIoC Details. n.d. 12 January 2024.< https://strangeioc.github.io/strangeioc/>.

Dead By Daylight. n.d. 12 January 2024.< https://deadbydaylight.com/>.

Dying Light. N.d. 12 January 2024. <https://dyinglightgame.com/ >.

The Last of Us. n.d. 12 January 2024.<https://en.wikipedia.org/wiki/The_Last_of_Us>.

The Forest. n.d. 12 January 2024.<https://endnightgames.com/games/the-forest>.