COMP4902
Graduation Design Project

FEYZİYE SCHOOLS FOUNDATION
IŞIK UNIVERSITY

Project Title
Survival Game Design and Implementation

By Yaren Çay

SYSTEM DESIGN DOCUMENT

SYSTEM DESIGN DOCUMENT

1. Introduction

The Software Design Overview Document (SDD) for the "Big Picture" project delineates the architectural framework and design objectives for the ongoing software system development.

1.1 Purpose of the System

The primary role of the "Big Picture" SDD is to offer a comprehensive overview of the software architecture and design principles within the project. It serves as a fundamental reference for the development team, providing guidance throughout the software development process. This document primarily achieves the following objectives:

- It presents an architectural perspective that defines the structural and operational aspects of the upcoming software system.
- It offers a comprehensive representation of the system's composition, encompassing its components, interfaces, and their interactions.
- It ensures a systematic and precise comprehension of the software's architecture for all stakeholders involved in the development process.

Furthermore, through maintaining traceability with the "Big Picture" RAD, the SDD ensures a seamless integration of all identified requirements and constraints into the software architecture.

1.2 Design Goals

This document focuses on the fundamental architecture of the "Big Picture" software system. It provides a high-level overview of architectural principles without delving into

detailed code-specific information. The "Big Picture" SDD is agnostic to particular programming languages, frameworks, or development methods.

1.3 Definitions, Acronyms, and Abbreviations

Definitions:

- Crafting System: The game mechanism that allows players to combine and create items, tools, and structures for survival and progression.
- NPC (non-Player Character): Characters within the game environment controlled by the computer rather than a human player.
- A* Pathfinding Algorithm: A* pathfinding algorithm used in navigation systems to find the shortest path between two points, crucial for character movement in the game world.

Acronyms

- UI: User Interface
- MSSQL: Microsoft SQL Server

Abbreviations

- IoC: Inversion of Control
- HUD: Heads-Up Display
- GUI: Graphical User Interface

2. Current Software Architecture

Differences from other survival game projects:
1. Narrative and Emotional Depth:

- The storyline revolves around a character who lost his wife in a zombie attack, introducing a compelling personal motivation for survival. The emphasis on the emotional journey sets "Big Picture" apart in creating a more immersive and resonant player experience.

2. Crafting System Innovation:

- Introduces a unique crafting system that goes beyond the conventional. Players not only craft survival essentials but also play a pivotal role in reconstructing the protagonist's house, adding a layer of creativity and personalization to the survival experience.

3. StrangeIoC Framework Integration:

- Utilizes the StrangeIoC framework, providing a robust foundation for modular development and dependency injection. This enhances code organization, maintainability, and scalability, setting "Big Picture" apart in terms of technical architecture.

3. Proposed Software Architecture

3.1 Overview

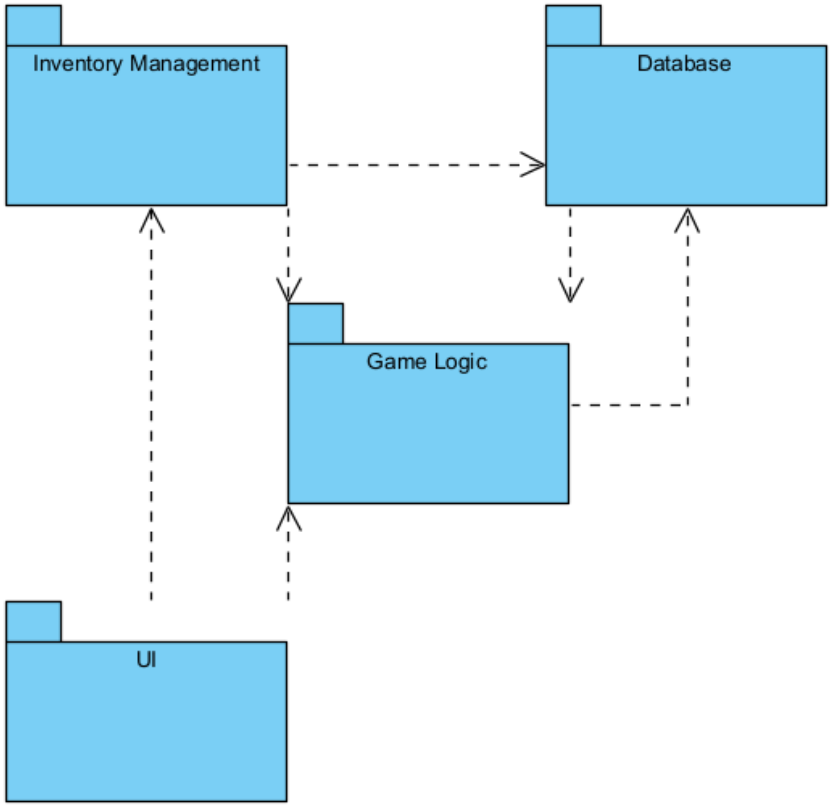

Figure 1. Package diagram

- User Interface (UI):
  It is responsible for presenting vital information to players and facilitating their interaction with the game's world. This includes various in-game menus, screens, and graphical interfaces that enhance player engagement and provide a seamless way to navigate the post-apocalyptic world, make decisions, and respond to in-game challenges.
- Game Logic:
  The core of the system, the Game Logic subsystem, manages the game rules, mechanics, and interactions. It oversees player actions, character navigation, combat, and house defense.
- Inventory Management:
  This subsystem handles the organization and utilization of collected items and resources. It is crucial for proper inventory management, which impacts survival and crafting within the game.
- Audio Control:
  The Audio Control subsystem manages in-game music and sound effects, allowing users to customize their audio experience. It handles volume adjustments and sound effect triggers.

- Database Integration:
  To support the persistence of player data and game state, the Database Integration subsystem interacts with a Microsoft SQL (MSSQL) database. This interaction ensures that player progress is stored and can be retrieved across gaming sessions.

3.2 System Decomposition

- Subsystem 1: User Interface (UI)

  Responsibilities:

  - Present the game's graphical elements, including menus, character models, and user interface controls.
  - Collect and process user input for in-game actions and menu navigation.
  - Display essential information to players, such as character stats, inventory, and game status.
  - Ensure a responsive and visually appealing gaming experience to enhance player immersion.
- Subsystem 2: Game Logic

  Responsibilities:

  - Manage the game's core rules and mechanics, including character navigation, combat, and house defense.
  - Interpret player actions and implement corresponding in-game responses.
  - Simulate the behavior of Chromozombies, their movement patterns, and interactions with the player.
  - Control the game's progression, objectives, and events to maintain player engagement.
- Subsystem 3: Inventory Management

  Responsibilities:

  - Organize and maintain the player's inventory of collected items and resources.
  - Implement crafting mechanics that allow players to create weapons, tools, and other items.
  - Ensure that inventory management is efficient and user-friendly, contributing to player survival and house defense.
- Subsystem 4: Database Integration

  Responsibilities:

  - Communicate with the Microsoft SQL (MSSQL) database to store and retrieve player data and game progress.
  - Ensure the persistence of player inventories, house defense setups, and game state.

– Enable players to continue their progress across gaming sessions and devices.

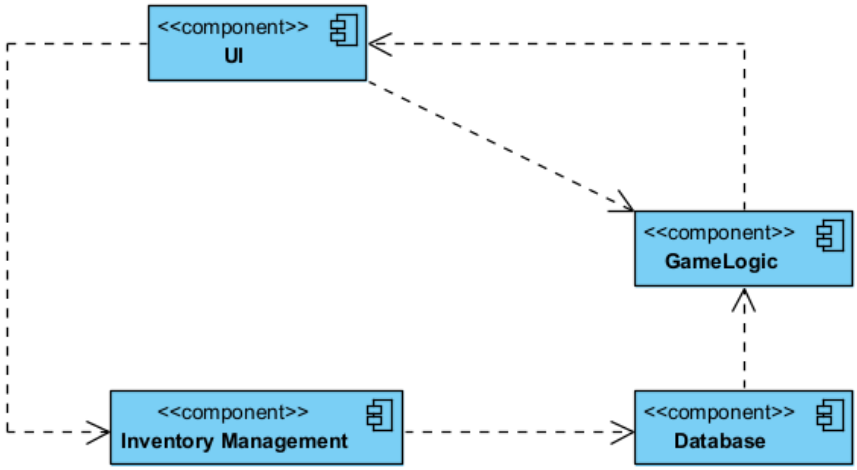

Figure 2. Component Diagram

3.3 Hardware/Software Mapping

Subsystem 1: User Interface (UI)
Mapping:

- In the context of "Big Picture" and its MVC architecture facilitated by the StrangeIoC framework, the User Interface (UI) subsystem is a crucial software component tailored to run efficiently on standard computer hardware. It is responsible for delivering an engaging and visually captivating UI experience to players. The following considerations apply:
- Software-Based: The UI subsystem primarily operates as software, seamlessly integrated into the MVC architecture. It is designed to provide players with an intuitive and visually appealing means of accessing game features, managing resources, and navigating the game's post-apocalyptic world.
- Graphics Libraries: To ensure efficient and visually pleasing UI presentation, the UI subsystem utilizes well-established graphic rendering libraries. These libraries contribute to the creation of immersive in-game menus, screens, and graphical interfaces, enhancing the overall player experience.

Subsystem 2: Game Logic
Mapping:

- The Game Logic subsystem, within the "Big Picture" project and its MVC architecture provided by StrangeIoC, is another essential software-based component. It operates on the game server, hosted on standard computing hardware. The following considerations apply:

- Software-Based: Game Logic is a software component that seamlessly integrates with the MVC architecture. It manages game mechanics and simulations, ensuring consistent and engaging gameplay experiences.
- Custom Game Engines: To handle complex game mechanics and simulations, the Game Logic subsystem utilizes custom game engines and libraries. These components are tailored to the specific needs of "Big Picture."

Subsystem 3: Inventory Management
Mapping:

- In "Big Picture," the Inventory Management subsystem functions as software integrated with the game server, making it compatible with standard computing hardware. The following considerations apply:
- Software-Based: Inventory Management operates as software that integrates seamlessly into the MVC architecture. It efficiently handles inventory data storage and retrieval.
- Database Management Systems: To ensure efficient inventory data management, the subsystem utilizes off-the-shelf database management systems. These systems are optimized for storing and retrieving inventory-related information.

Subsystem 4: Database Integration
Mapping:

- Database Integration in "Big Picture" interfaces with Microsoft SQL (MSSQL) database servers, which can be hosted on standard server hardware. The following considerations apply:
- Software-Based: Database Integration is a software component that efficiently interfaces with the MVC architecture. It manages data communication with the database, ensuring seamless access to game-related information.
- Database Management Systems: The subsystem utilizes database management systems and drivers designed to establish reliable and efficient communication with the Microsoft SQL (MSSQL) database servers.

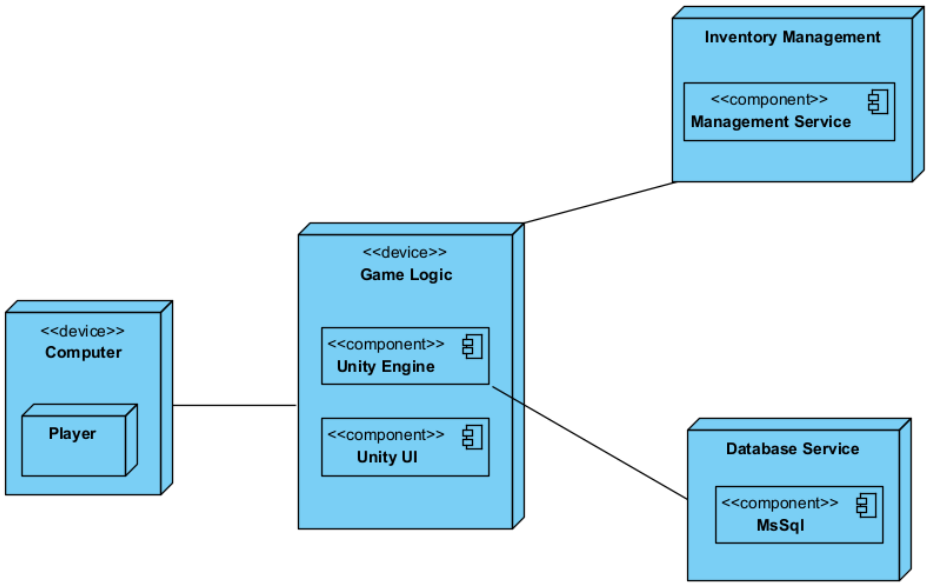

Figure 3. Deployment Diagram

3.4 Persistent Data Management

In "Big Picture," the management of persistent data is a fundamental aspect of providing an engaging and immersive gaming experience. This section outlines the data structures, database selection, and data encapsulation strategies employed in the project.

Description of Persistent Data:

- Data Schemas: The game requires the storage of various data types, including player profiles, game progress, inventory items, and achievements. These data are organized into distinct schemas to facilitate efficient data management.
  - Player Profile Schema: Contains attributes such as player username, in-game currency.
  - Inventory Schema: Manages the inventory of in-game items, including weapons, tools, and consumables.
- Data Attributes: Each data schema includes specific attributes relevant to the stored data.
  - Player Profile Schema Attributes: Username (string), In-Game Currency (integer), Unlocked Content (list).
  - Inventory Schema Attributes: Inventory Items (list), Quantity (integer).

Selection of Database:

- Database Management System (DBMS): "Big Picture" employs a relational database management system (RDBMS) for data storage. The selection of an RDBMS is based

on the structured nature of the game's data and the need for ACID (Atomicity, Consistency, Isolation, Durability) properties.

- Database Type: The RDBMS used is MySQL. MySQL offers robust support for data consistency and reliability, making it suitable for managing player data and game progress.
- Scalability: The chosen database system is designed to handle scalable data storage. This accommodates a growing player base and supports future expansions and content updates.
- Security: Robust security measures, including encryption, access controls, and data backup, are implemented to protect player data and adhere to data privacy regulations.

Data Encapsulation:

- Database Integration: Data integration is achieved through a well-defined data access layer (DAL) that abstracts the database operations from the rest of the game. The DAL ensures secure and efficient data interaction with the database.
- Data Access Layer (DAL): The DAL is a key architectural component responsible for managing database interactions. It includes a set of APIs and services for reading, writing, and updating player data. The DAL is designed to prevent SQL injection and other security vulnerabilities.
- Caching: Caching mechanisms are employed to enhance data retrieval performance. Frequently accessed data, such as player profiles, are cached to reduce database load and improve response times.
- Data Backup and Recovery: Routine data backups are performed to safeguard player data. Backup and recovery strategies are in place to ensure data persistence in the event of system failures.
- Data Migration: When introducing updates or changes to the database schema, data migration processes are executed to ensure seamless transitions and maintain data integrity.

The "Persistent Data Management" infrastructure in "Big Picture" plays a vital role in delivering a consistent and enjoyable player experience by securely managing and storing player data while providing efficient access to that data. The selection of an RDBMS, coupled with a well-structured data access layer, enables the game to handle player data effectively while adhering to stringent security and privacy standards.

3.5 Global Software Control

Request Initiation:

In the "Big Picture" game, global software control is orchestrated through a centralized Game Manager module responsible for overseeing the entire gaming experience. Requests are initiated through various triggers, including:

- User Input:
    - Player actions, such as moving characters, interacting with objects, or accessing menus, generate requests through the UI subsystem.

- Game Events:
    - Key in-game events, like the detection of a zombie attack or the completion of a crafting action, trigger requests from the Game Logic subsystem.
- External Systems:
    - Requests may also originate from external systems, such as server-side events or responses from a database update.

Subsystem Synchronization:

Synchronization between subsystems is crucial to maintain a cohesive and responsive gaming experience. Here's how synchronization is managed:

- Messaging System:
    - A messaging system facilitates communication between subsystems. For example, when the Game Logic subsystem detects a significant event, it sends a message to the UI subsystem to update the display accordingly.
- Event-driven Architecture:
    - In the "Big Picture" survival game, the Player Health System employs Event-driven Architecture for dynamic responses to key events like "PlayerDamaged." When damage occurs, this event triggers UI updates, such as the health bar adjustment. This modular approach ensures synchronized and efficient handling of in-game events, enhancing the player experience.
- Mutual Exclusion (Mutex) Mechanisms:
    - To handle potential concurrency issues, critical sections of code within subsystems, especially in the Game Logic and Inventory Management modules, are protected by mutex mechanisms. This prevents conflicts arising from simultaneous access to shared resources.

Concurrency Issues:

Addressing concurrency issues is essential for a seamless gaming experience. Here are specific measures taken:

- Thread Management:
    - Each subsystem operates in its own thread, ensuring parallel execution without interference. For example, the UI subsystem runs independently of the Game Logic subsystem to maintain responsiveness.
- Atomic Operations:
    - Operations involving shared data, such as inventory updates, are designed as atomic operations to avoid race conditions. This ensures that data consistency is maintained even in a concurrent environment.
- Transaction Management:
    - Database transactions are used to manage data consistency in the Database subsystem. For instance, when saving player progress, a transaction ensures that either all changes are applied or none, preventing partial updates.

3.6 Boundary Conditions

Start-up Procedure:

- Initialization: When a player launches the game, the system initializes essential components, including the game client, rendering engine, and input systems.
- Subsystem Activation: The necessary subsystems, including User Interface (UI), Game Logic, and Database Integration, are activated. Subsystems synchronize to provide a seamless experience.
- Loading Game Assets: Game assets, such as graphics and game data, are loaded to prepare the game environment.
- Player Session Creation: A player session is created, associating the player with an instance of the game world. This session is maintained throughout the gaming session.

Shutdown Procedure:

- Session Termination: The player session is terminated, releasing resources associated with the player's presence in the game world.
- Subsystem Deactivation: Unnecessary subsystems are deactivated, reducing resource consumption. For example, if the player is not actively interacting with the game, certain UI components may be deactivated.
- Clean-Up: Temporary data and resources are cleaned up to maintain system efficiency.

Error Behavior:

- Graceful Degradation: In the event of non-critical errors, the system employs graceful degradation. For example, if a non-essential UI component encounters an error, the game continues to run smoothly.
- Error Logging: All errors, whether critical or non-critical, are logged for review by system administrators. Error logs include detailed information to facilitate efficient troubleshooting.
- User Notification: When errors impact the player experience, users are notified with user-friendly messages that provide guidance on resolving the issue or contacting support.

4. Subsystem Services

1. UI Subsystem:

- Show Crafting Menu:
    - Initiates the crafting menu, allowing the player to interact with the crafting system and create items.
- Show Inventory:
    - Description: Displays the player's inventory on the UI, providing a visual representation of collected items and resources.

2. Game Logic Subsystem:

- Navigate Character:
    - Guides the player-controlled character to a specified destination using the A* pathfinding algorithm.
- Handle Combat:
    - Manages combat interactions, calculating damage, and updating the game state based on player and enemy actions.
- Manage Player Inventory:
    - Facilitates inventory management, handling item pickups, usage, and storage.

3. Database Subsystem:

- Save Player Progress:
    - Persists player progress data, including inventory, and current game state, to the database.
- Load Player Data:
    - Retrieves player data from the database, allowing for seamless continuation of the game from a previous session.

4. Inventory Management Subsystem:

- Craft Item:
    - Implements the crafting system, allowing the creation of items based on collected resources and blueprints.
- Use Item:
    - Manages the usage of items from the player's inventory, applying effects or changes to the game state.

5. References

1. StrangeIoC Documentation (https://strangeioc.github.io/strangeioc/faq.html)
2. Unity With MVC **(https://www.toptal.com/unity-unity3d/unity-with-mvc-how-to-level-up-your-game-development#:~:text=The%20Model%2DView%2DController%20pattern,Controllers%20(Decision%2FAction))**
3. Unity Database Integration **(https://www.alirookie.com/post/connect-your-unity-project-to-the-ms-sql-server)**
4. Unity User Manual **(https://docs.unity.com/)**
5. Understanding Game Architecture **(https://www.studytonight.com/3d-game-engineering-with-unity/game-development-architecture)**