Advanced Design Document (ADD)

UPB

Shooter

**Team members:**

**Ciobanu Bianca**

**Lupașcu Gelu-Codrin**

**Nicoară Andreea-Georgeta**

**Stanciu Ștefan-Daniel**

**Table of contents:**

1.Introduction

Purpose and scope of the advanced design document

Overview of the game concept, target audience, and development goals

2.Game modes and features

Weapons, characters, and maps

Progression system and other gameplay mechanics

3.Setting and plot

Characters and narrative

Presentation of the story to the player

Art Style

4.Character designs and environments

User interface and other visual elements

Art direction and constraints

Technical Design

5.Hardware and software requirements

Game engine and tools

Technical architecture

6.User interface and menus

Controls and gameplay feedback

Error reporting and customer support

7.Monetization

In-game purchases and microtransactions

Legal and business considerations

8.Marketing and Distribution

Marketing campaigns and partnerships

Distribution channels

9.Conclusion

Summary of key points

Final thoughts and considerations

**1. INTRODUCTION**

**Purpose and scope of the advanced design document:**

The purpose of this document is to provide a detailed design plan for the development of a first-person shooter game. It covers all major aspects of the game, including gameplay mechanics, art style, technical design, user experience, monetization, and marketing and distribution. The scope of the document includes the initial release of the game and any planned updates or expansions.

**Overview of the game concept, target audience, and development goals:**

The game is a fast-paced, competitive shooter set in a futuristic world. It is aimed at players aged 18-35 who enjoy action-packed multiplayer games. The development goals for the game include creating a fun and engaging gameplay experience, delivering high-quality graphics and audio, and offering a variety of content for players to enjoy.

**2. GAME MODES AND FEATURES**

The game will offer a variety of game modes, including deathmatch, capture the flag, and objective-based modes. Players will be able to choose from a range of characters, each with their own unique abilities and playstyle. The game will also feature a progression system, allowing players to unlock new weapons, characters, and other content as they play.

**Weapons, characters, and maps:**

The game will have a diverse selection of weapons, including rifles, pistols, shotguns, and more. Players will be able to customize their characters with a variety of appearance options and loadouts. The game will also feature a variety of maps, each with its own unique layout and visual style.

**Progression system and other gameplay mechanics:**

The progression system in the game will allow players to earn experience points and level up, unlocking new content as they progress. Other gameplay mechanics will include character abilities, weapon attachments, and environmental elements such as cover and destructible objects.

**3. SETTING AND PLOT**

In this shooter game, players will assume the role of a skilled soldier fighting against a zombie apocalypse in a modern-day setting. The game will feature fast-paced, tactical gameplay, with a focus on team-based play and objective-based modes. Players will be able to customize their characters and loadouts, and will have access to a wide range of weapons and equipment. The game will also feature a compelling story told through cutscenes and dialogue, as well as immersive sound effects and music.

**Characters and narrative:**

The game will feature a cast of characters representing each of the factions, each with their own motivations and personalities. The narrative will be presented through in-game cutscenes and dialogue.

**Presentation of the story to the player:**

The story will be integrated into the gameplay experience, with missions and objectives reflecting the ongoing conflict between the factions. Cutscenes and dialogue will be used to advance the plot and provide context for the player's actions.

**4. CHARACTER DESIGN AND ENVIRONMENTS**

The game will have a sleek, futuristic art style, with characters and environments that are detailed and realistic. Character designs will reflect the diverse range of factions in the game, with each having their own distinct visual identity.

**User interface and other visual elements:**

The user interface will be clean and modern, with a focus on ease of use. Other visual elements such as menu screens, loading screens, and in-game HUDs will follow the same aesthetic.

**Art direction and constraints:**

The art direction for the game will prioritize high-quality graphics and a cohesive visual style. Any constraints or considerations, such as technical limitations or budget constraints, will be taken into account when designing the game's art assets.

**5. HARDWARE AND SOFTWARE REQUIREMENTS**

The game will support a range of hardware configurations, including both desktops and laptops. It will require a minimum of Windows 7 or higher and a DirectX 11-compatible graphics card. The game will also have recommended system requirements for optimal performance.

**Game engine and tools:**

The game will be developed using the Unreal Engine 4 game engine. Other tools and software used in the development process will include version control software, project management software, and design and prototyping software.

**Technical architecture:**

The game will have a client-server architecture, with the client running on the player's device and the server handling gameplay logic and networking. The game will use a variety of technologies and protocols, including TCP/IP, HTTP, and websockets, to communicate between the client and server. The server will be hosted in a cloud-based infrastructure, allowing for scalability and reliability.

**6. USER INTERFACE AND MENU**

The game's user interface will be intuitive and easy to navigate, with a focus on providing players with quick access to the features and content they need. Menus will be organized clearly and concisely, and will use visual elements such as icons and graphics to aid in navigation.

**Controls and gameplay feedback:**

The game's controls will be customizable and responsive, allowing players to tailor their setup to their preference. In-game feedback such as hit markers and damage numbers will provide players with important information about their actions and the state of the game.

**Error reporting and customer support:**

The game will have an in-game error reporting system to allow players to easily report any bugs or issues they encounter. Customer support will be available through a variety of channels, including email and social media, to assist players with any issues or questions they may have.

**7. MONETIZATION**

**In-game purchases and microtransactions:**

The game will offer a variety of in-game purchases and microtransactions, including cosmetic items, weapon skins, and other vanity items. It will also offer a "battle pass" system, allowing players to purchase a season-long progression track with exclusive rewards.

**Legal and business considerations:**

The game will follow all relevant laws and regulations regarding monetization, including age restrictions and consumer protection laws. Business considerations such as pricing and payment processing will also be taken into account when designing the game's monetization strategy.

**8. MARKETING AND DISTRIBUTION**

**Marketing campaigns and partnerships:**

The game will have a comprehensive marketing plan in place, including advertising campaigns on social media and other platforms, partnerships with streamers and content creators, and other promotional activities.

**Distribution channels:**

The game will be distributed digitally through platforms such as Steam, GOG, and the Epic Games Store. It will also be available for purchase on the game's official website.

**9. CONCLUSION**

**Summary of key points:**

The advanced design document has outlined the design and development plan for a first-person shooter game, covering gameplay mechanics, art style, technical design, user experience, monetization, and marketing and distribution.

**Final thoughts and considerations:**

The game's development team will continue to review and refine the design document as the project progresses, making any necessary adjustments to ensure the best possible player experience.

**Architectural design**

**Type or architecture used:**

The Entity-Component-System (ECS) architecture is a design pattern that is commonly used in game development to manage and organize game objects and their behaviors. It is based on the idea of separating game objects into three types: entities, components, and systems.

**Entities:**

An entity is a unique identifier for a game object. It does not have any data or behavior of its own, and serves as a placeholder for attaching components. An entity is usually represented by a simple integer or string ID, and can be thought of as a blank canvas that is given meaning and functionality through the components it is associated with.

For example, in a first-person shooter game, an entity might represent a player character, a weapon, or an enemy. A player character entity might have components such as a transform (position, rotation, and scale), a mesh (3D model), and a health component. A weapon entity might have components such as a transform, a mesh, and a damage component. An enemy entity might have components such as a transform, a mesh, an AI component, and a health component.

**Components:**

A component is a small, self-contained piece of data or behavior that is associated with an entity. A component should be as simple and focused as possible, and should only contain the data or logic that is directly related to its purpose.

For example, in a first-person shooter game, some possible components might include:

Transform: Stores the position, rotation, and scale of an entity.

Mesh: Stores the 3D model and material of an entity.

Health: Stores the current health and maximum health of an entity.

Damage: Stores the amount of damage that an entity can deal.

AI: Stores the AI logic and behavior of an entity.

Components are usually implemented as simple structs or classes, and are designed to be easily serialized and deserialized for saving and loading. They should not contain any complex logic or dependencies, and should be easy to add, remove, or modify at runtime.

**Systems:**

A system is a piece of code that operates on one or more types of components, and is responsible for a specific aspect of the game's behavior. Systems are usually implemented as classes, and are designed to be reusable and flexible.

For example, in a first-person shooter game, some possible systems might include:

Physics System: Handles the simulation of physics forces and collisions, and updates the transform components of entities accordingly.

Rendering System: Renders the game world and its entities, using the transform and mesh components as input.

AI System: Controls the behavior of AI-controlled entities, using the AI and transform components as input.

Input System: Processes user input and updates the transform and other components of the player character entity accordingly.

Systems are usually organized into a hierarchy or a graph, and can be composed of multiple smaller systems or sub-systems. They should be designed to be extensible and modular, allowing for easy customization and expansion.

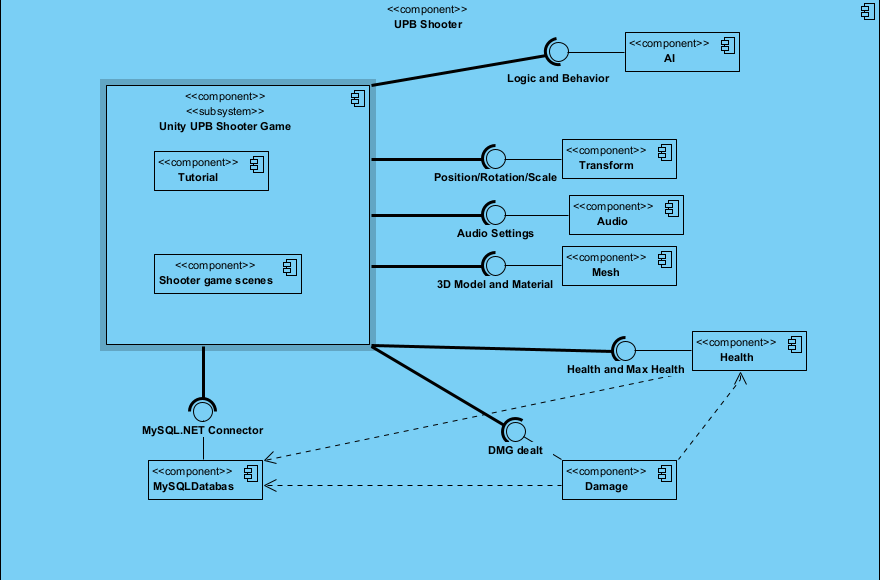
**Benefits of ECS:**

The Entity-Component-System (ECS) architecture has several benefits for game development:

It allows for a clean separation of concerns, with each component and system focused on a specific aspect of the game's behavior.

It promotes modularity and reusability, making it easy to add, remove, or modify game.

**For our game, we chose the Component Diagram:**



**Data Design/ Software Components**

**Data Design:** Entity-Component-System (ECS)

**Overview:**

This game uses an Entity-Component-System (ECS) architecture, which separates game objects into three types: entities, components, and systems. The data design for this game reflects this structure, and is designed to be flexible, scalable, and easy to maintain.

**Entities:**

An entity is represented by a unique identifier, such as an integer or a string. The entity ID is used to associate the entity with its components and to identify it in the game world.

**Components:**

Components are represented by structured data types, such as structs or classes. Each component type has a specific set of fields that store the data relevant to its purpose.

For example, a Transform component might have fields such as "Position", "Rotation", and "Scale". A Mesh component might have fields such as "Model", "Material", and "Animation". A Health component might have fields such as "CurrentHealth" and "MaxHealth".

Components are usually stored in a flat data structure, such as an array or a map, indexed by their entity ID. This allows for fast access and iteration, but also requires that the component data be compact and memory-efficient.

**Systems:**

Systems are represented by code modules, such as classes or functions. Each system is responsible for a specific aspect of the game's behavior, and operates on one or more types of components.

For example, a Physics System might operate on Transform and Rigidbody components, and update their values based on the simulation of physics forces. A Rendering System might operate on Transform and Mesh components, and render the entities in the game world. An AI System might operate on AI and Transform components, and control the behavior of AI-controlled entities.

Systems are usually organized into a hierarchy or a graph, and can be composed of multiple smaller systems or sub-systems. They should be designed to be extensible and modular, allowing for easy customization and expansion.

**Data Persistence:**

 The game's data, including entities, components, and systems, should be designed to be easily serialized and deserialized for saving and loading. This can be achieved through the use of serialization libraries or custom serialization code.

The data should be stored in a persistent format, such as a database or a file system, and should be structured in a way that allows for fast and efficient access. The data should also be designed to be modifiable and expandable, allowing for the addition of new components, systems, and features over time.

**Tables:**

Players: Stores data about the game's players, such as their username, password, and profile data.

Fields:

PlayerID (integer, primary key): A unique identifier for the player.

Username (string): The player's username.

Password (string): The player's hashed password.

DisplayName (string): The player's display name.

ProfileImage (blob): The player's profile image.

XP (integer): The player's experience points.

Level (integer): The player's level.

Characters: Stores data about the game's characters, such as their appearance and loadout.

Fields:

CharacterID (integer, primary key): A unique identifier for the character.

PlayerID (integer, foreign key): The ID of the player that owns the character.

Name (string): The character's name.

Model (string): The character's 3D model file path.

Material (string): The character's material file path.

PrimaryWeapon (integer): The ID of the character's primary weapon.

SecondaryWeapon (integer): The ID of the character's secondary weapon.

Equipment (string): A serialized list of the character's equipment items.

Weapons: Stores data about the game's weapons, such as their stats and attributes.

Fields:

WeaponID (integer, primary key): A unique identifier for the weapon.

Name (string): The weapon's name.

Model (string): The weapon's 3D model file path.

Material (string): The weapon's material file path.

Type (string): The weapon's type (e.g. "rifle", "pistol", "shotgun").

Damage (integer): The weapon's base damage.

Accuracy (float): The weapon's accuracy.

FireRate (float): The weapon's fire rate (in shots per second).

ReloadTime (float): The weapon's reload time (in seconds).

Equipment: Stores data about the game's equipment items, such as their stats and effects.

Fields:

EquipmentID (integer, primary key): A unique identifier for the equipment item.

Name (string): The equipment item's name.

Model (string): The equipment item's 3D model file path.

Material (string): The equipment item's material file path.

Type (string): The equipment item's type (e.g. "helmet", "vest", "goggles").

Effect (string): A description of the equipment item's effect on the character.

StatModifiers (string): A serialized list of the equipment item's stat modifiers.

Matches: Stores data about the game's matches, such as the map played, the players that participated, and the match results.

Fields:

MatchID (integer, primary key): A unique identifier for the match.

MapID (integer, foreign key): The ID of the map played in the match.

StartTime (datetime): The start time of the match.

EndTime (datetime): The end time of the match.

Results (string): A serialized list of the match results, including the player IDs, character IDs, and scores of each player.

Leaderboards: Stores data about the game's leaderboards, such as the players' ranks, scores, and playtime.

Fields:

LeaderboardID (integer, primary key): A unique identifier for the leaderboard.

PlayerID (integer, foreign key): The ID of the player on the leaderboard.

Rank (integer): The player's rank on the leaderboard.

Score (integer): The player's score on the leaderboard.

Playtime (integer): The player's total playtime in minutes.

Inventory: Stores data about the players' inventories, such as the items and currency they possess.

Fields:

InventoryID (integer, primary key): A unique identifier for the inventory.

PlayerID (integer, foreign key): The ID of the player whose inventory this is.

Items (string): A serialized list of the player's items.

Currency (integer): The player's total currency.

Shop: Stores data about the items available for purchase in the game's shop, such as their prices and descriptions.

Fields:

ItemID (integer, primary key): A unique identifier for the item.

Name (string): The item's name.

Description (string): A description of the item.

Price (integer): The item's price in currency.

Image (blob): An image of the item.

Maps: Stores data about the game's maps, such as their layout and spawn points.

- Fields:

- MapID (integer, primary key): A unique identifier for the map.

- Name (string): The map's name.

- Description (string): A description of the map.

- Image (blob): An image of the map.

- Layout (string): A serialized representation of the map's layout, including the positions of walls, cover, and other objects.

- SpawnPoints (string): A serialized list of the map's spawn points.

News: Stores data about the game's news articles and announcements, such as their titles, content, and dates.

- Fields:

- NewsID (integer, primary key): A unique identifier for the news article.

- Title (string): The title of the news article.

- Content (string): The content of the news article.

- Date (datetime): The date that the news article was published.

- Image (blob): An image associated with the news article.

- Link (string): A link to the full news article (if applicable).

**Database Schema:**

