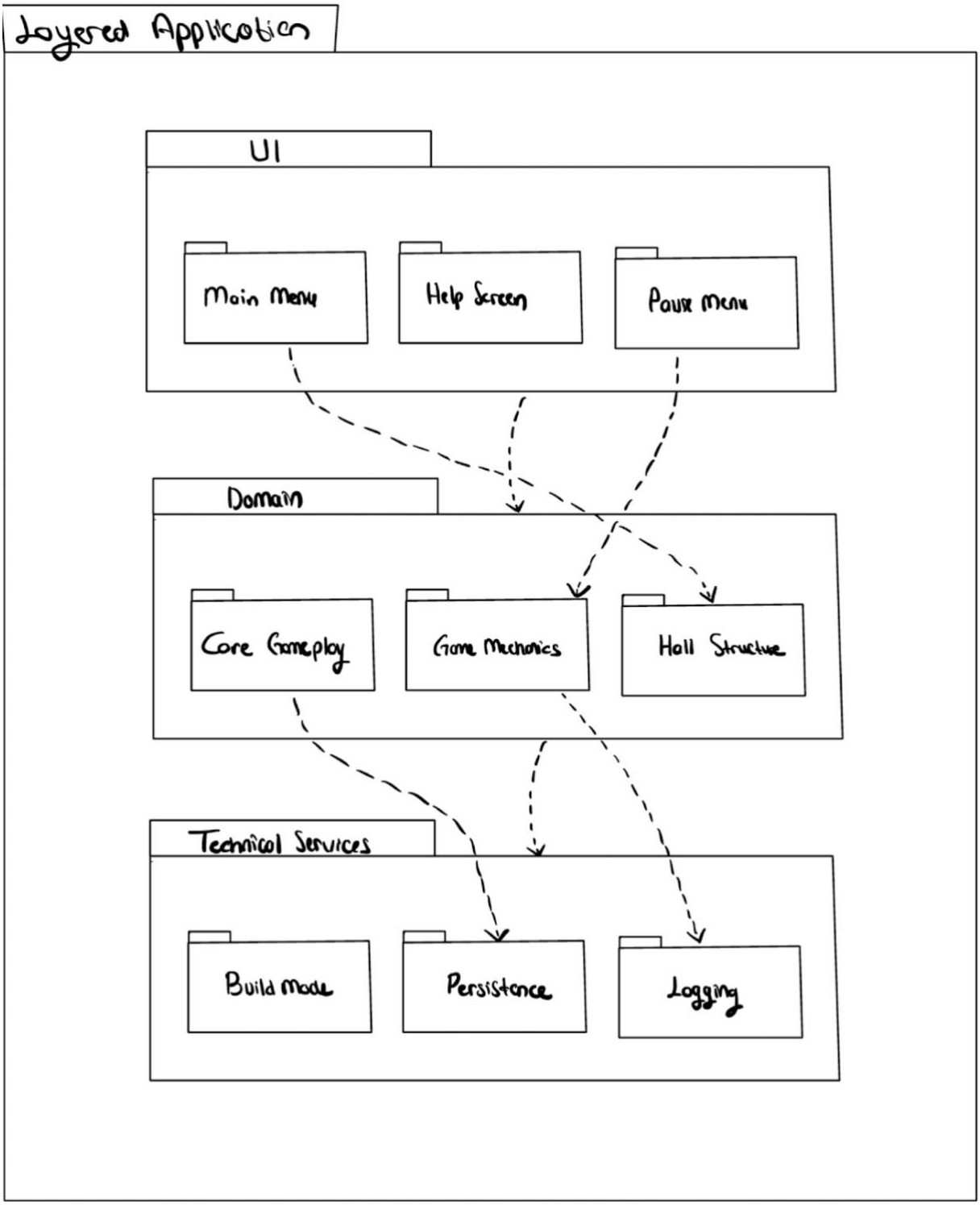
Logical Architecture



Defining Layers

1. **Presentation Layer (UI Layer):**
    - **Includes UI components like:**
        - **Main Menu: Start game, view help, exit options.**
        - **Help Screen: Explains gameplay, objects, and mechanics.**
        - **Pause Menu: Options to resume or quit the game.**
2. **Domain Layer (Middle Layer):**
    - **Represents the core game logic and gameplay elements:**
        - **Core Gameplay: Includes `Player`, `Monsters`, `Enchantments`.**
        - **Game Mechanics: Handles `Timer`, `Combat System`, `Inventory Management`.**
        - **Hall Structure: Includes `Halls` and `Grid` for game layout and object management.**
3. **Technical Services (Bottom Layer):**
    - **Handles lower-level operations:**
        - **Build Mode: For designing halls and placing objects.**
        - **Persistence: Saves game progress or configurations.**
        - **Logging: Tracks game events for debugging purposes.**

Objects in UI Layer:

· Main Menu:
- `Start Button, Help Button, Exit Button`

· Pause Menu:
- `Resume Button, Quit Button, Pause Overlay`

· Help Screen:
- `Control Guide, Game Objective Section, Monster Guide, Enchantment Guide, Navigation Buttons`

· Game HUD:
- `Timer Display, Lives Display, Current Hall Indicator, Bag Inventory Display, Pause Button`

· Build Mode UI:
- `Grid Layout, Object Palette, Confirm Button, Error Messages`

· Gameplay Interaction Elements:
- `Clickable Objects, Action Feedback`

Objects in Domain Layer:

- **Core Gameplay:**
    - `Player, Monsters, Enchantments`
- **Game Mechanics:**

    - `Timer`, `Combat System`, `Inventory`
- **Hall Structure:**
    - `Halls`, `Grid`, `Objects`

Objects in Technical Services Layer:

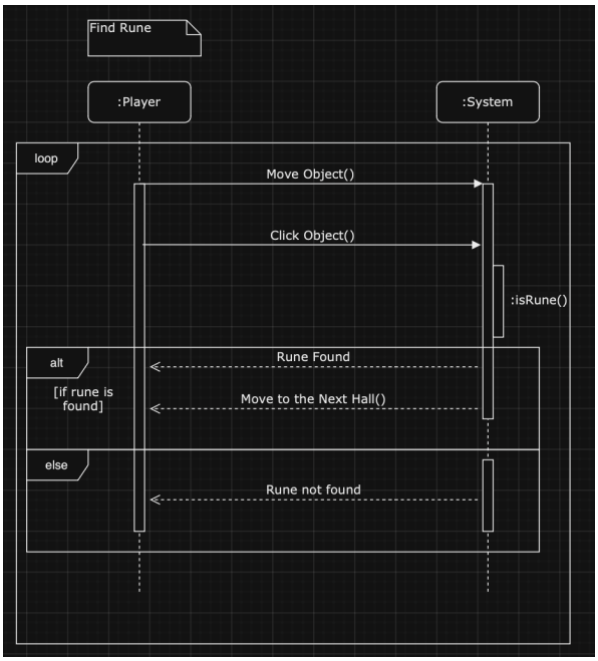
- **Build Mode:**
    - `Map Editor`, `Object Placement`
- **Persistence Services:**
    - `Save Manager`, `Load Manager`
- **Logging Services:**
    - `Event Logger`

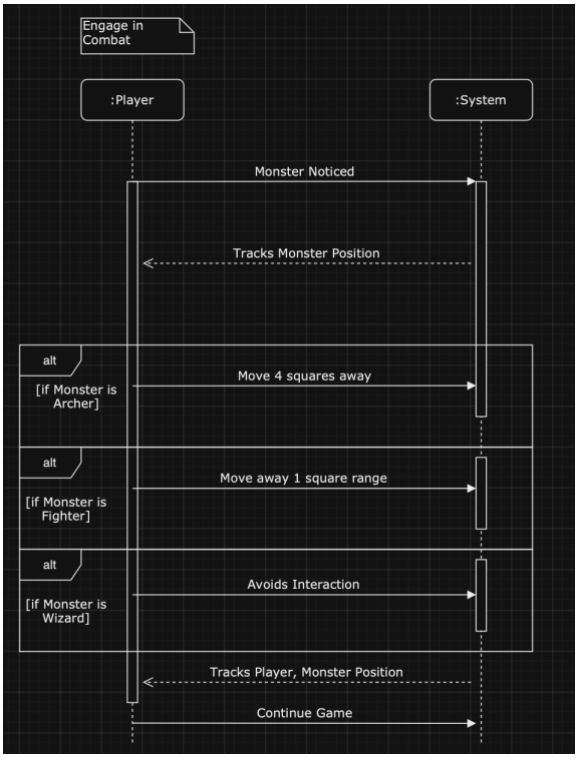
Relationships

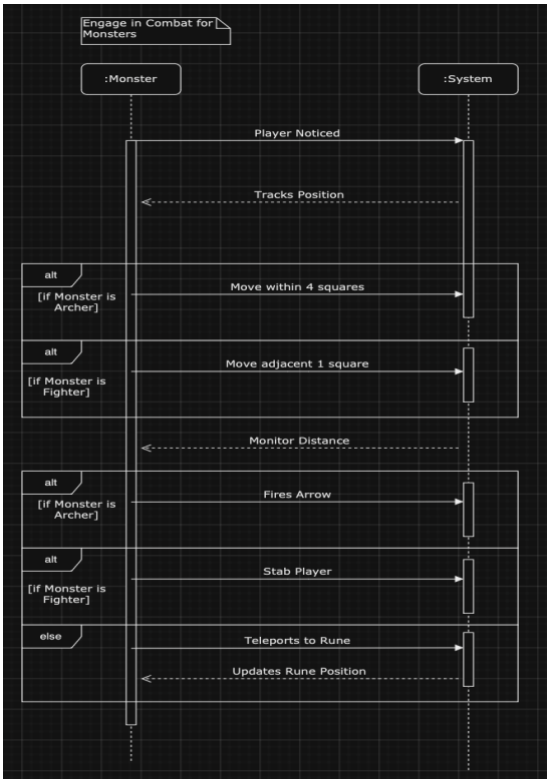
1. **UI to Domain:**
    - `Main Menu` → `Halls` **(to initialize a new game layout).**
    - `Pause Menu` → `Timer` **(to freeze gameplay).**
2. **Domain to Technical Services:**
    - `Player` → `Save Manager` **(to store player progress).**

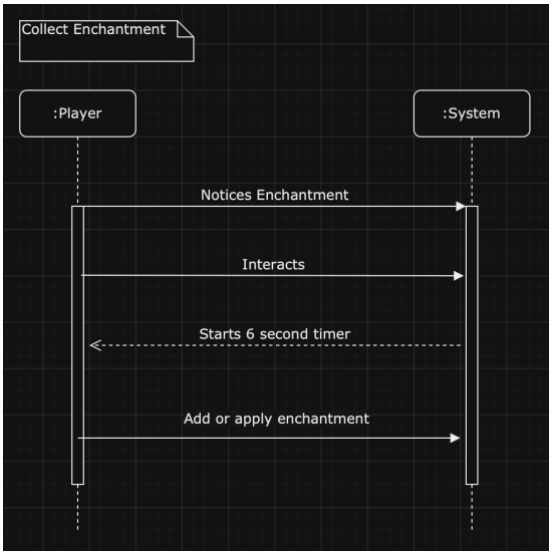
`Combat System` → `Event Logger` **(to log combat results).**

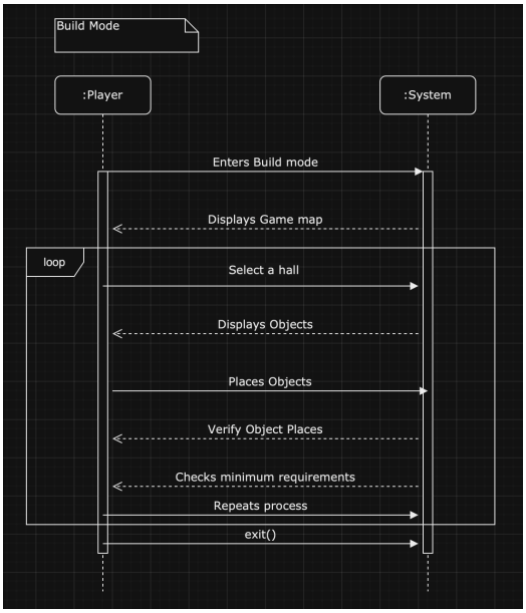
UML SSDs (10)

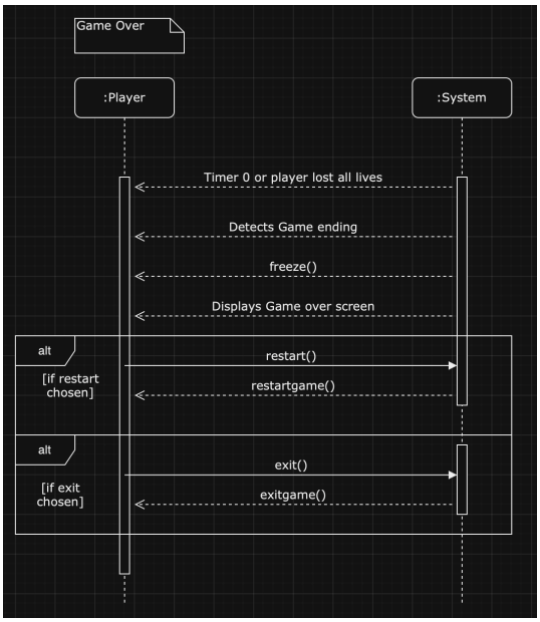

Find Rune
:Player
:System
loop
Move Object()
Click Object()
:isRune()
alt
[if rune is found]
Rune Found
Move to the Next Hall()
else
Rune not found

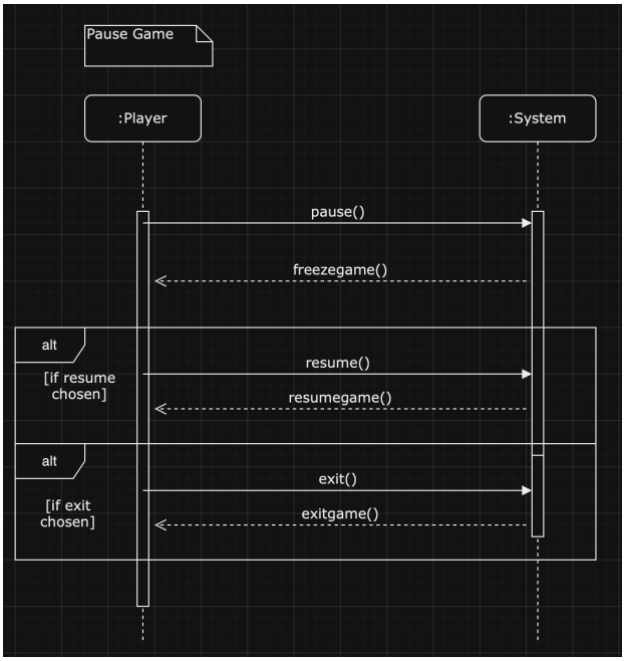

Engage in Combat
:Player
:System
Monster Noticed
Tracks Monster Position
alt
[if Monster is Archer]
Move 4 squares away
alt
[if Monster is Fighter]
Move away 1 square range
alt
[if Monster is Wizard]
Avoids Interaction
Tracks Player, Monster Position
Continue Game

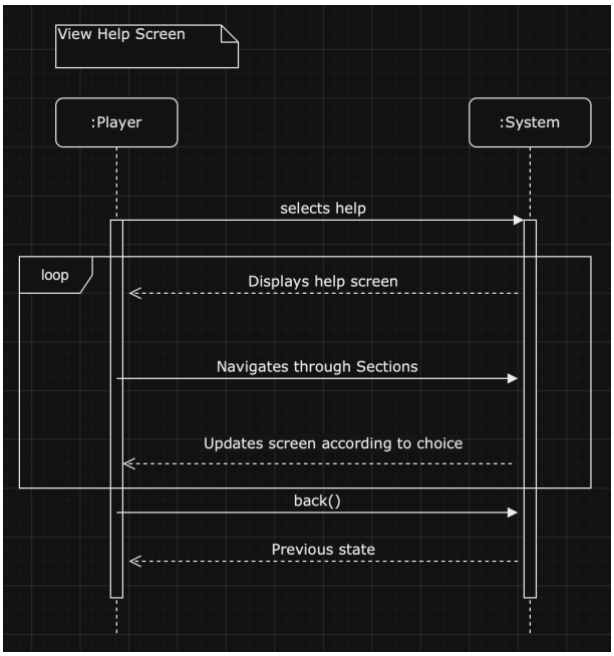

Engage in Combat for Monsters
:Monster
:System
Player Noticed
Tracks Position
alt
[if Monster is Archer]
Move within 4 squares
alt
[if Monster is Fighter]
Move adjacent 1 square
Monitor Distance
alt
[if Monster is Archer]
Fires Arrow
alt
[if Monster is Fighter]
Stab Player
else
Teleports to Rune
Updates Rune Position

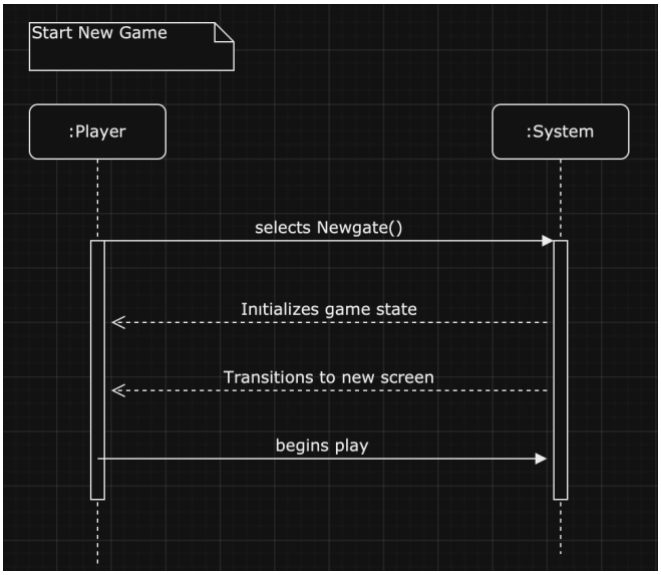

Collect Enchantment
:Player
:System
Notices Enchantment
Interacts
Starts 6 second timer
Add or apply enchantment

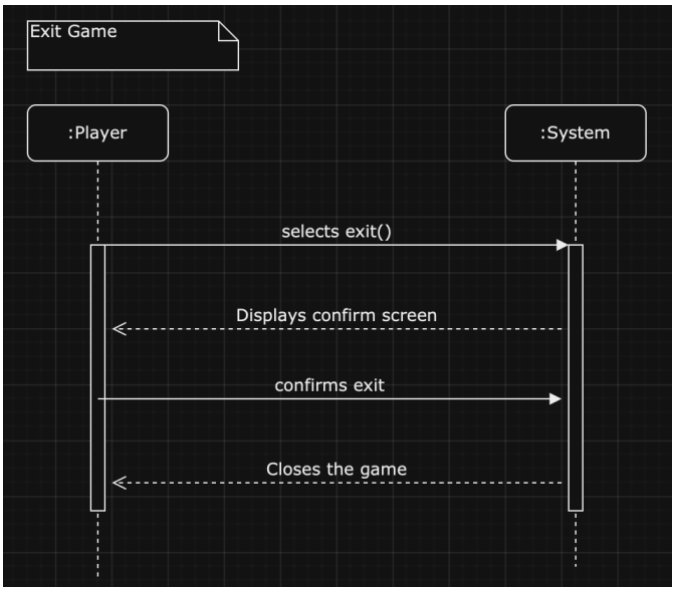

Build Mode
:Player
:System
Enters Build mode
Displays Game map
loop
Select a hall
Displays Objects
Places Objects
Verify Object Places
Checks minimum requirements
Repeats process
exit()


Game Over
:Player
:System
Timer 0 or player lost all lives
Detects Game ending
freeze()
Displays Game over screen
alt
[if restart chosen]
restart()
restartgame()
alt
[if exit chosen]
exit()
exitgame()

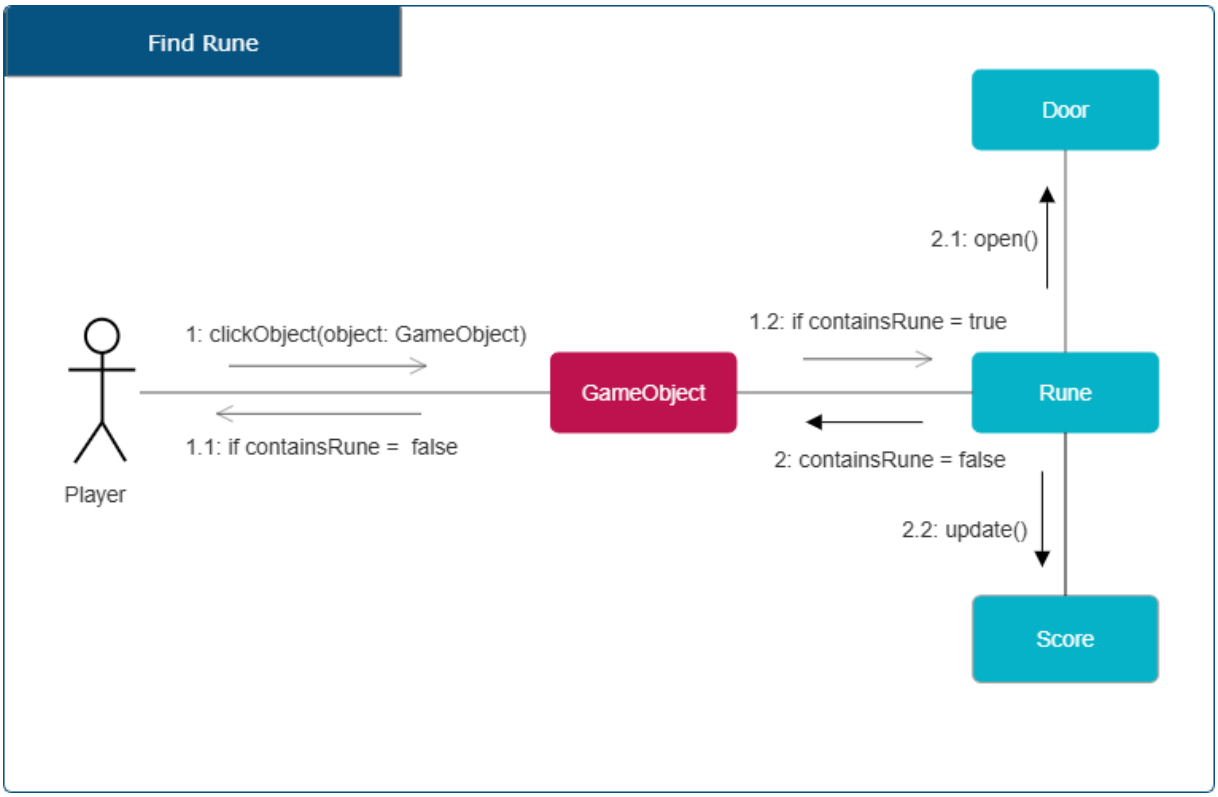

Pause Game
:Player
:System
pause()
freezegame()
alt
[if resume chosen]
resume()
resumegame()
alt
[if exit chosen]
exit()
exitgame()

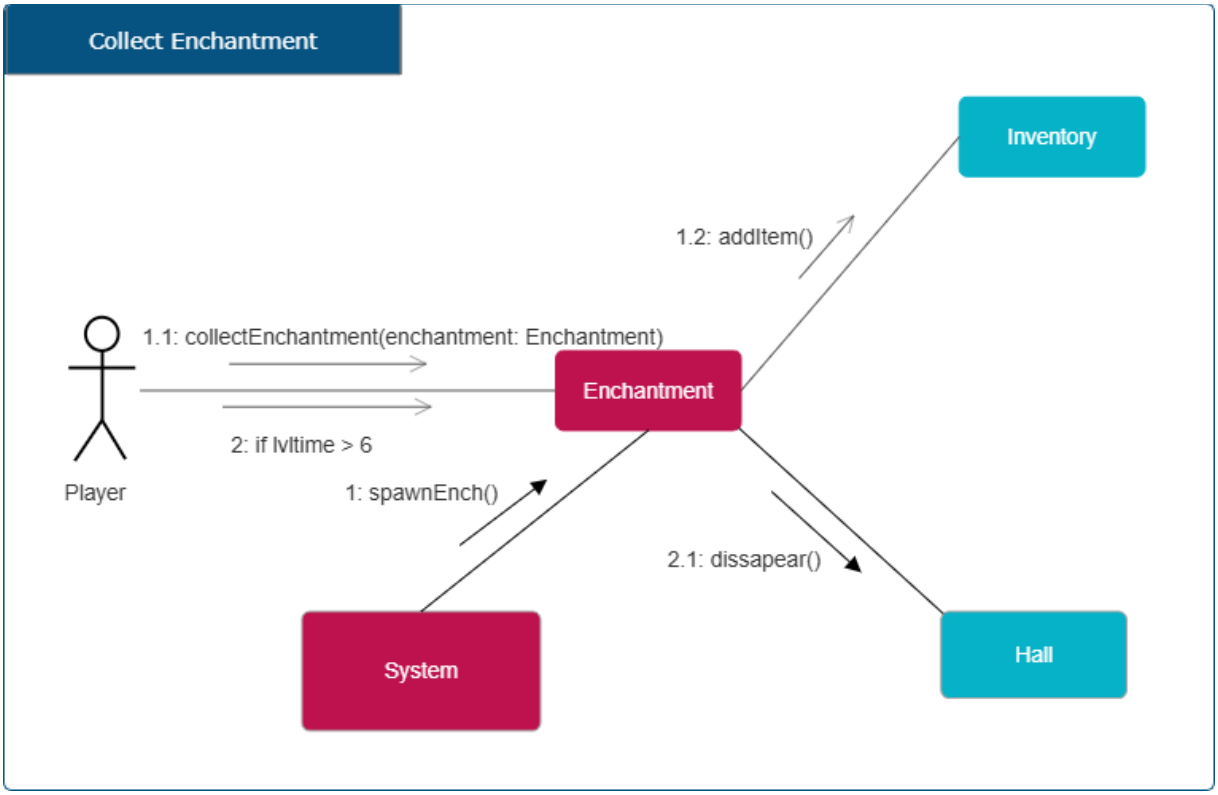

View Help Screen
:Player
:System
selects help
loop
Displays help screen
Navigates through Sections
Updates screen according to choice
back()
Previous state


Start New Game
:Player
:System
selects Newgate()
Initializes game state
Transitions to new screen
begins play

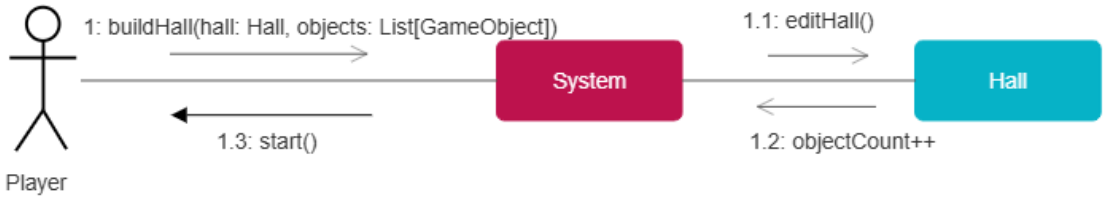

Exit Game
:Player
:System
selects exit()
Displays confirm screen
confirms exit
Closes the game

Communication Diagrams (10)

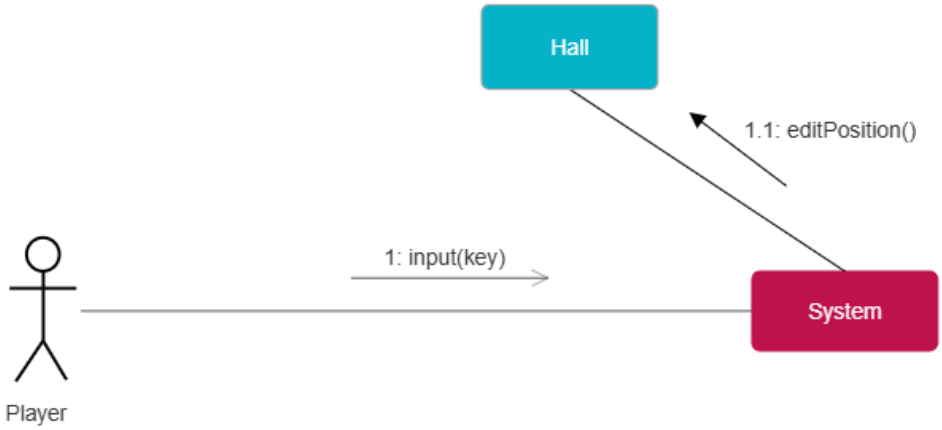




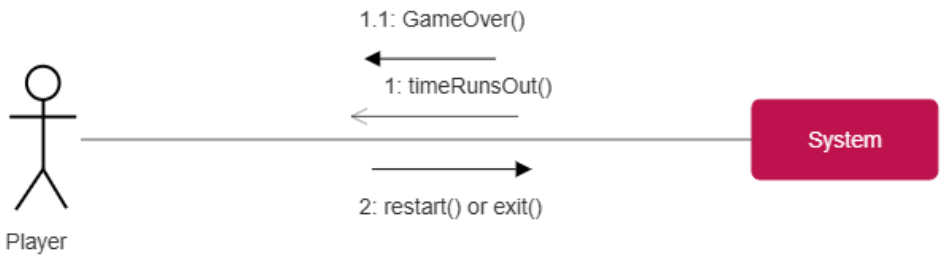
Build Mode



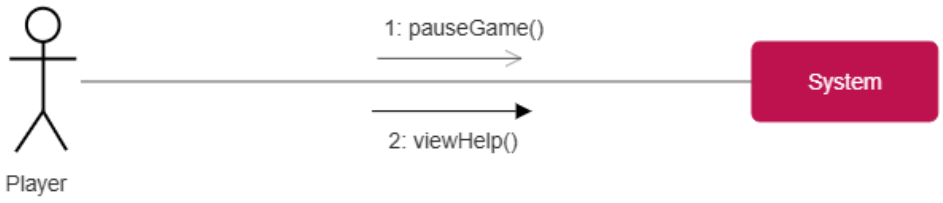
Player Movement



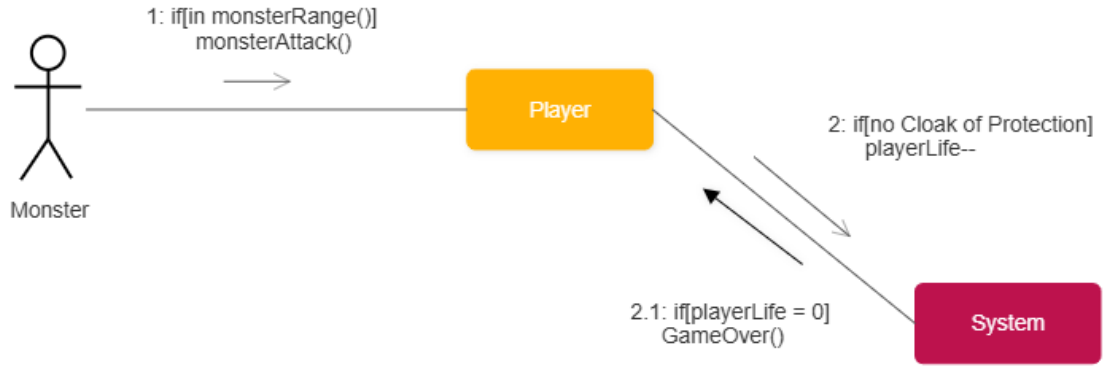
Time Runs Out



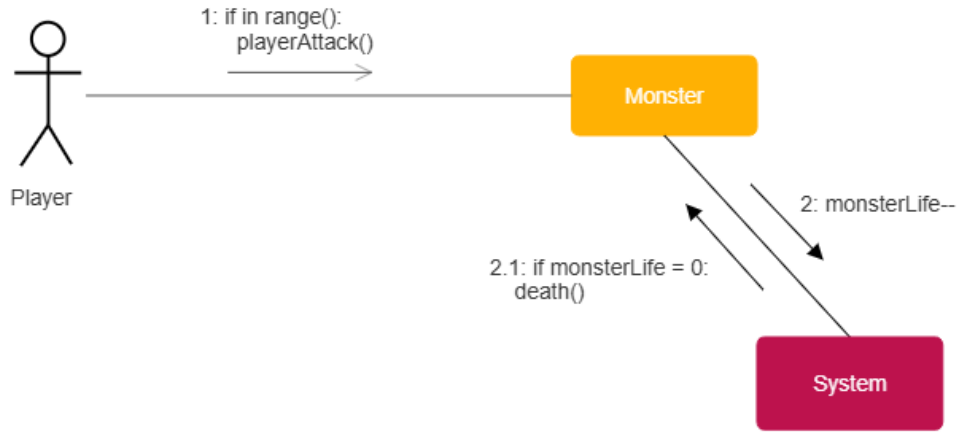
Help Screen Access

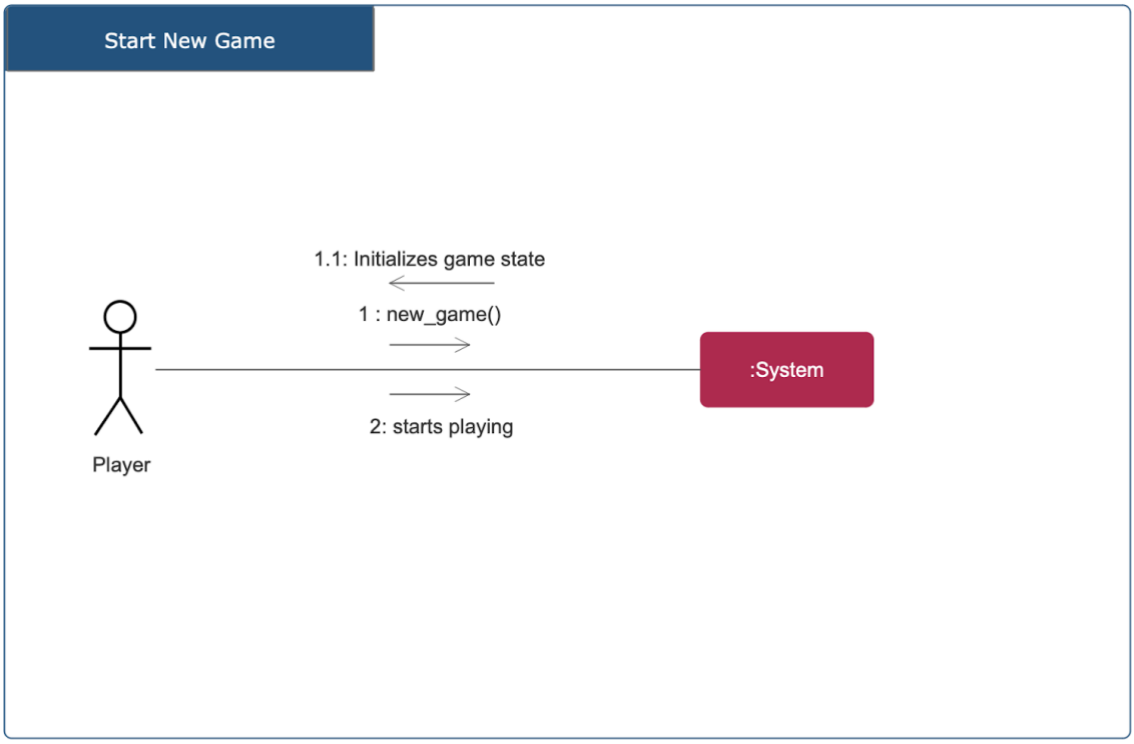


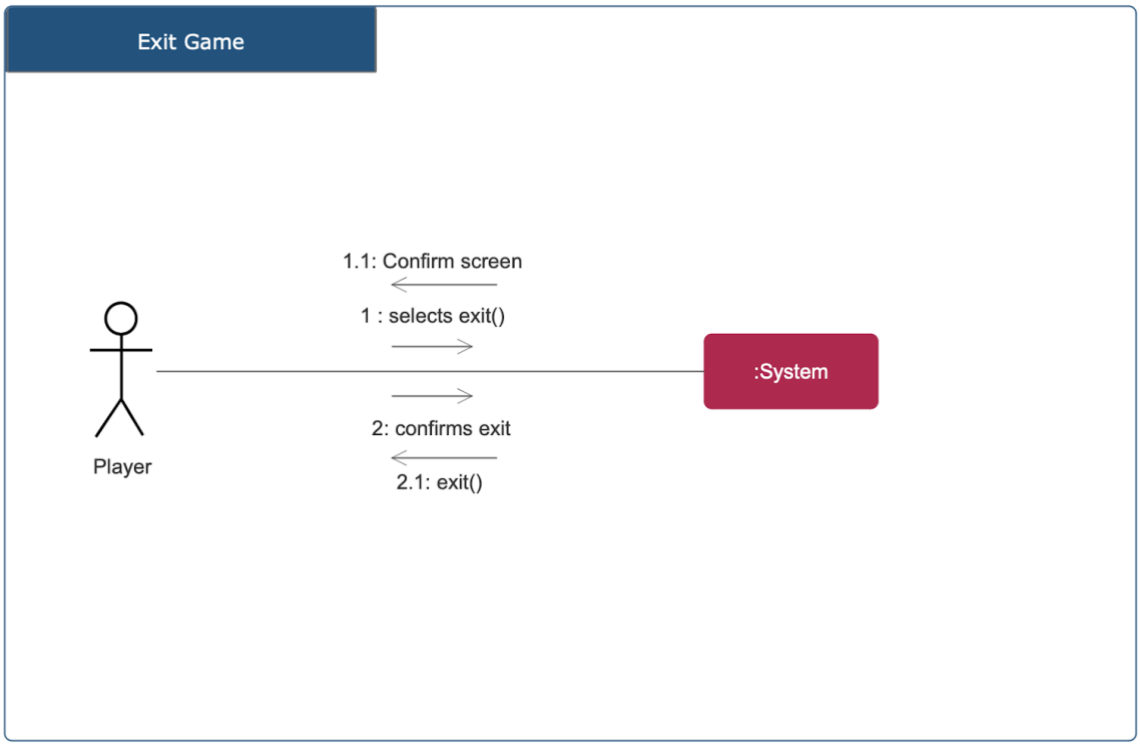
Monster Attack Pattern



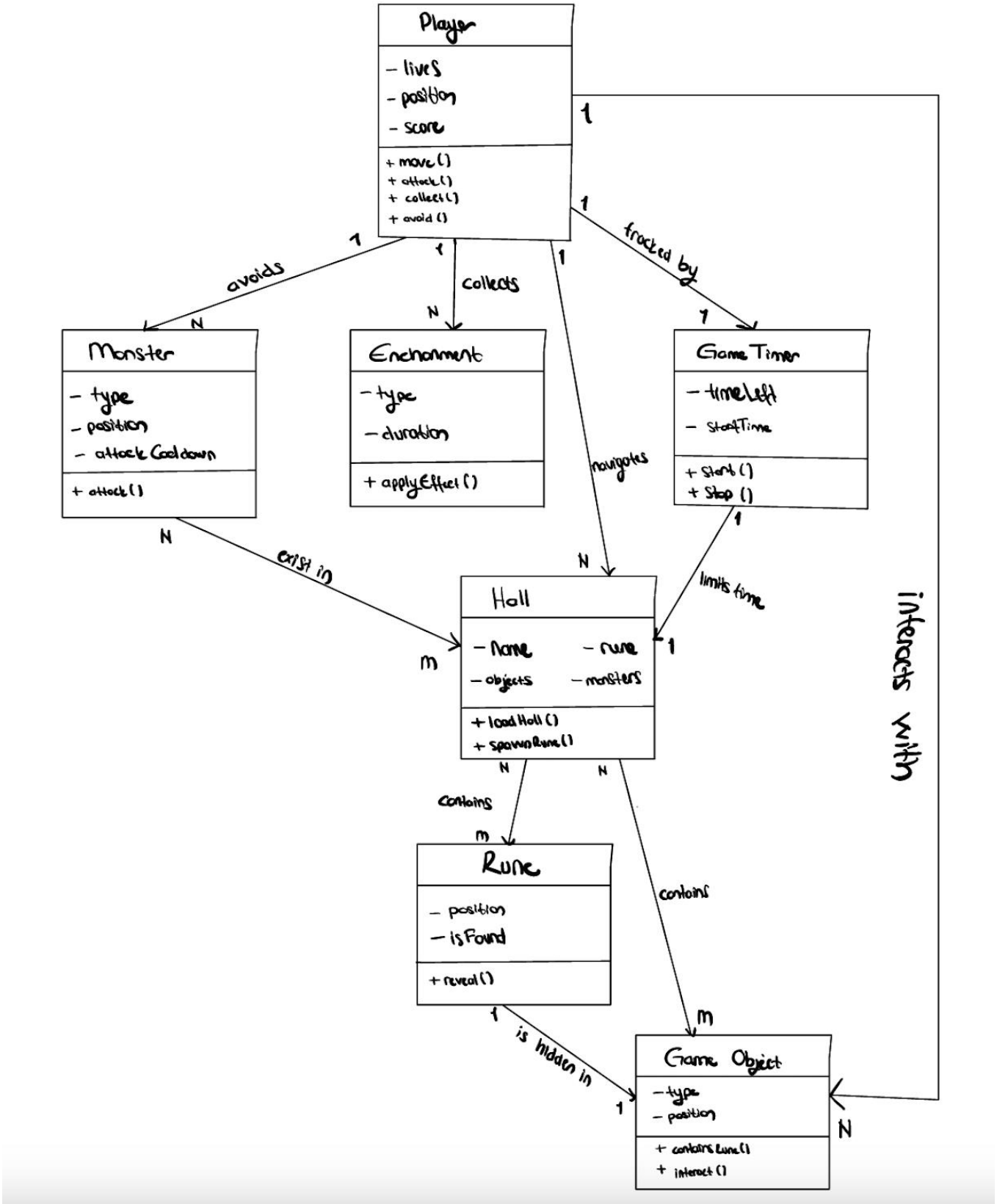
Player Attack Pattern




Start New Game
1.1: Initializes game state
1 : new_game()
:System
2: starts playing
Player


Exit Game
1.1: Confirm screen
1 : selects exit()
:System
2: confirms exit
2.1: exit()
Player

UML Class Diagrams



1) Design Alternative : Event-Driven Architecture

In an **event-driven architecture**, the game would rely heavily on event-based communication, where actions such as "player moves," "monster attacks," or "object clicked" would generate events. These events would be captured by listeners that process and handle the consequences asynchronously.

Comparison with Your Design:

- Your layered architecture uses direct method calls to perform operations (clickObject, playerAttacks), whereas an event-driven approach decouples these interactions by broadcasting events to listeners.

Pros of Event-Driven Architecture:

1. **Decoupling**: Components are less tightly connected, making it easier to modify or add new features without affecting existing ones.
2. **Scalability**: Event-driven systems can handle complex game logic by having multiple listeners handle different aspects of a single event (e.g., triggering animations, sound effects, and score updates simultaneously).
3. **Concurrency**: Events can be processed asynchronously, improving performance in certain scenarios.

Cons of Event-Driven Architecture:

1. **Debugging Complexity**: Tracing the flow of events and finding the source of bugs can be challenging.
2. **Overhead**: Maintaining an event system introduces additional overhead, such as ensuring listeners are registered/unregistered correctly.
3. **Potential Latency**: Actions might take slightly longer to execute due to the asynchronous nature of event handling.

Why Your Design Was Chosen:

Your layered approach prioritizes **simplicity and predictability**, ensuring direct and immediate execution of operations. This reduces the complexity of debugging and is suitable for a game with a smaller scale or fewer simultaneous interactions.

2) Design Alternative: Entity-Component-System

What It Is:

In an **ECS architecture**, game elements (e.g., player, monsters, objects) are entities composed of reusable components. Each component handles a specific piece of functionality (e.g., movement, health, AI), and systems process components based on their types.

Pros of ECS:

1. **Reusability**: Components can be reused across different entities, reducing code duplication.
2. **Flexibility**: New behaviors can be added by creating new components without altering existing entities.
3. **Performance**: ECS often leverages data-oriented design, leading to improved performance by optimizing memory access patterns.

Cons of ECS:

1. **Steep Learning Curve**: ECS requires a shift in mindset from traditional object-oriented programming, which may slow development initially.
2. **Overengineering**: For small-scale projects, the complexity of ECS may outweigh its benefits.
3. **Debugging Challenges**: Decoupling functionality into components and systems can make tracking down issues more difficult.

3) Design Alternative: Procedural Generation (Like Minecraft and Vampire Survivors)

Instead of relying on **Build Mode**, where players manually design hall layouts, the game could use **procedural generation** to automatically create halls based on predefined algorithms and rules.

Comparison with Your Design:

- Build Mode focuses on user creativity (if the player chooses to design his/her own map) or game creators vision of map creation, giving players direct control over hall design. Procedural generation would shift this responsibility to the game engine.

Pros of Procedural Generation:

1. **Replayability**: Procedurally generated content ensures each playthrough is unique, increasing replay value.
2. **Reduced Player/Creator Effort**: Players who are less interested in designing can jump directly into gameplay without needing to create layouts.
3. **Scalability**: Large numbers of halls can be generated automatically, reducing development time for level design.

Cons of Procedural Generation:

1. Players lose the ability to design their own levels, which could decrease creativity of players
2. **Quality Control**: Procedural generation might produce nonsensical or unbalanced layouts, requiring additional checks and algorithms.

3. Not being able to express a story easily if we choose to, since