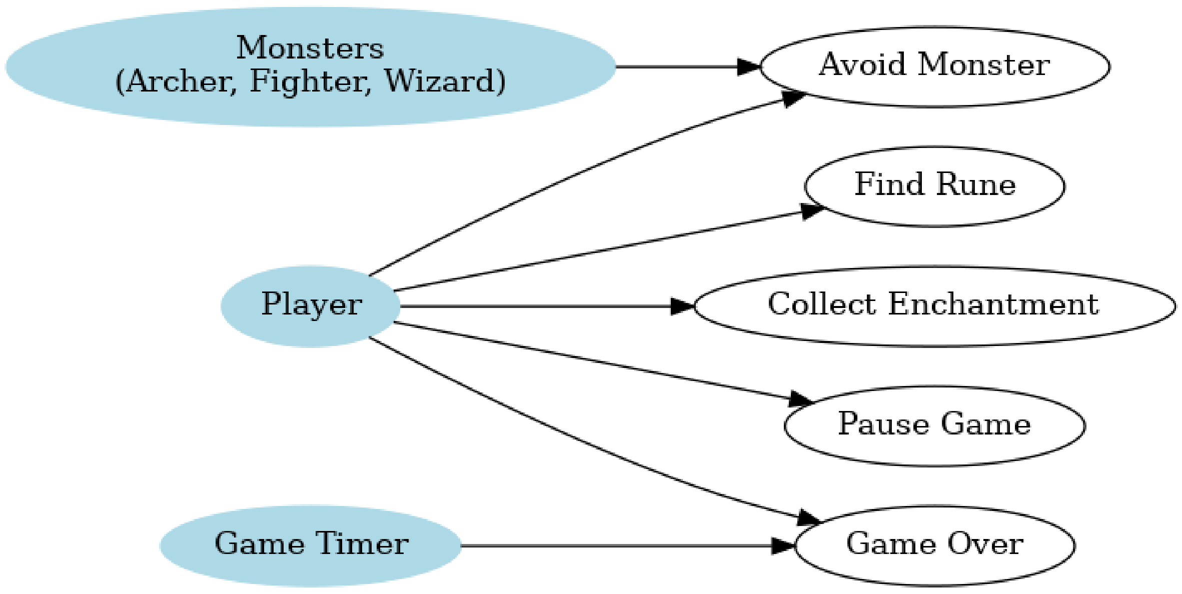
–UML Use Case Diagram



Use Case Narratives

Use Case 1: Find Rune
Actors: Player, System

Preconditions:

1. The player is inside a specific hall (e.g., Hall of Earth).
2. At least one rune is hidden under objects within the hall.
3. The hall contains a minimum number of objects as required for that level.
4. The player has remaining time and lives.

Main Success Scenario:

1. The player navigates using arrow keys to move next to an object.
2. The player clicks the object to search for the rune.
3. The system checks if the rune is hidden under the selected object.

- If the rune is found:
  - A. The system reveals the rune with a visual effect.
  - B. The door to the next hall unlocks with a sound effect.
  - C. The player progresses to the next hall.
- If the rune is not found:
  - D. The system does not reveal anything.
  - E. The player continues searching for other objects.

Alternative Scenarios:

- Time runs out, then the system triggers the "Game Over" use case.
- The player loses all lives the game ends.
- Objects do not contain the rune, the system provides no feedback, and the player continues searching.

Postconditions:

If the rune is found then the player progresses to the next hall, which is updated accordingly.
If the player fails to find the rune (time runs out or lives are lost), then the game ends.

Use Case 2: Engage in Combat for Player
Actors: Player, Monsters (Archer, Fighter, Wizard)

Preconditions:

1. At least one monster has appeared in the current hall.
2. The player has at least one life remaining.

Main Success Scenario:

1. The player notices a monster in the hall.
2. The player navigates using arrow keys to stay out of the monster's attack range:
    - **Archer Monster:** The player moves at least 4 squares away from the monster to avoid being hit by arrows.
    - **Fighter Monster:** The player moves out of the monster's immediate vicinity (1 square range) to avoid being stabbed.
    - **Wizard Monster:** The player avoids interaction as the wizard does not directly attack.
3. The system continuously tracks the monster's position relative to the player.
4. The player successfully avoids losing any lives by staying out of attack range.

Alternative Scenarios:

1. **Partial Damage:**
    - The player gets hit by an archer's arrow or stabbed by a fighter monster.
    - The player's remaining lives decrease by one.
    - The player continues attempting to avoid further damage or decides to engage strategically.
2. **Player Retreats:**
    - The player chooses to move to another area of the hall to avoid the monster entirely.
    - The player delays searching for the rune until the monster is no longer a threat.
3. **Player Defeated:**
    - The player fails to avoid damage and loses all lives.
    - The system triggers the "Game Over" use case.

Postconditions:

1. **If the player survives:** The player continues searching for the rune or engaging in combat with monsters.
2. **If the player is defeated:** The game ends.

Use Case 3: Engage in Combat for Monsters:

Actors: Player, Monsters (Archer, Fighter, Wizard)
Preconditions:

- Player has entered a hall.

Main Success Scenario:

- The monster notices the player in the hall.
- The monster closes the distance between itself and the player..
- The system tracks the player's position relative to the monster.
- The monster successfully gets close enough(Archer monster: 4 squares, Fighter monster: 1 square) to attack the player.
- The monster successfully inflicts damages to the player.

Alternative Scenarios:

1. **Archer Monster:**
    a. If the player is within 4 squares of the Archer Monster:
        i. The monster fires an arrow.
        ii. The player doesn’t lose a life because it is wearing a Cloak of Protection.
2. **Fighter Monster**:
    a. If the player is adjacent to the Fighter Monster:
        i. The monster attempts to stab the player.
        ii. The player doesn’t lose a life because it used a distraction (e.g., Luring Gem) on the monster.
3. **Wizard Monster**:
    a. If the Wizard Monster teleports the rune:
        i. The player is noticed by the monster.
        ii. The monster casts a spell to change the position of the rune.
        iii. The system updates the rune’s new position.

Postconditions:

- The monster succeeds on inflicting damage to the player and tries to attack again.

Use Case 4: Collect Enchantment

Actors: Player
Preconditions:

- The player notices the enchantment visual on his screen.

Main Success Scenario:

1. The system generates a random enchantment and places it in the hall.
2. The player notices the enchantment and moves toward it using arrow keys.
3. The player interacts with the enchantment to gather it before 6 seconds elapses.
4. The system adds the enchantment to the player's inventory or applies its effect immediately depending on the enchantment's type(e.g., Extra Time or Extra Life).

Alternative Scenarios:

- The player fails to collect the enchantment within 6 seconds:
    - The enchantment disappears.
- The player already has maximum inventory capacity:
    - The system rejects the enchantment.

Postconditions:

- The enchantment is successfully added to the player's inventory or applied.

Use Case 5: Build Mode

Actors: Player
Preconditions:

- The player entered Build Mode from the main menu.
- The game map is displayed as a grid, showing the four halls: Earth, Air, Water, and Fire.
- A minimum number of objects required for each hall is defined:

  Earth Hall: 6 objects

  Air Hall: 9 objects

  Water Hall: 13 objects

  Fire Hall: 17 objects

Main Success Scenario:

1. The player selects a hall from the grid view to start designing.
2. The system displays the available objects for placement in the selected hall.
3. The player places objects on the grid by clicking or dragging them to desired locations.
4. The system verifies that object placement does not overlap or exceed the grid boundaries.
5. The player completes the object placement for the selected hall, meeting or exceeding the minimum object requirements.
6. The player repeats the process for all four halls.
7. Once all halls are designed, the player confirms the design and exits Build Mode to start gameplay.

Alternative Scenarios:

1. Player Places Fewer than the Minimum Objects:

   A. If the player attempts to exit a hall without placing the required number of objects, the system prevents the exit and the message prompts the player to add more objects to meet the minimum criteria.

2. Object Placement Conflicts:

   If the player tries to place an object on an occupied grid cell or outside the hall's boundaries:

   The system prevents the action and displays an error message (e.g., "Invalid Placement").

3. Object Removal/Replacement:

   - The player decides to remove or replace an already placed object:
   - The player selects the object, and the system removes it, freeing the grid cell for new placement.

4. Exceeding Grid Capacity:

- If the player tries to place more objects than the grid can accommodate:'
- The system prevents additional placements and prompts the player to rearrange existing objects.

Postconditions:

- The map design meets the minimum object requirements for all halls designed
- The maps designed using build mode are ready-to-play for player

Use Case 6: Game Over

Actors: Player, Game Timer
Preconditions:

- The timer reaches zero, or the player loses all lives.
- The game is actively running when the end condition is triggered.

Main Success Scenario:

1. The **System** detects the game-ending condition:
   - The timer reaches zero.
   - OR the player's lives are exhausted.
2. The **System** freezes all ongoing gameplay actions (e.g., monster movements, player interactions).
3. The **System** displays a **Game Over screen** with the following options:
   - Restart the game from the beginning.
   - Exit to the main menu.
4. The **Player** selects an option:
   - **Restart**: The system resets all progress (e.g., timer, player position, lives, and inventory) and starts a new game.
   - **Exit**: The system ends the current game session and navigates to the main menu.

Postconditions:

- If the player selects "Restart":
  - The game resets all progress and begins a new session.
- If the player selects "Exit":
  - The game session ends, and the player is returned to the main menu.

Use Case 7: Pause Game

Actors: Player
Preconditions:

- The game is running.

Main Success Scenario:

1. The player clicks the "Pause" button.
2. The system freezes all game actions:
    - Timer stops.
    - Monsters stop moving.
    - Player cannot interact with objects or enchantments.
3. The system displays a "Paused" screen with options to resume or exit.

Alternative Scenarios:

- **Player resumes the game**:
    - The system unfreezes all actions and resumes the timer.
- **Player exits the game**:
    - The system closes the game and returns to the main menu.

Postconditions:

- The game either resumes or exits based on the player’s choice.

Use Case 8: View Help Screen

Actors: Player

Goal: To provide the player with an understanding of the game's objects, features, controls, and overall gameplay.

Preconditions:

- The game has been launched, and the player is on the main menu or has paused the game during play mode.

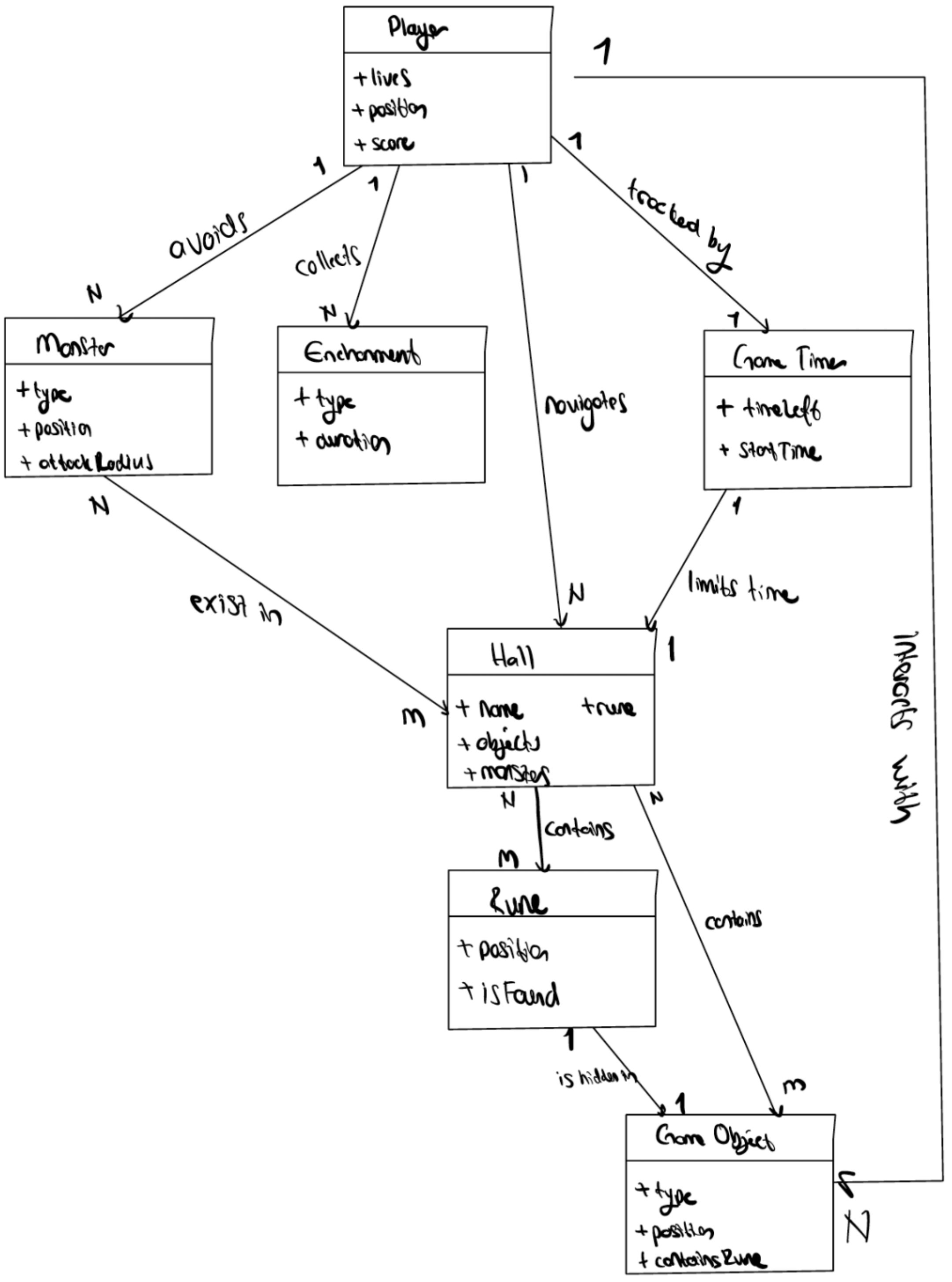
Main Success Scenario:

1. The player selects the "Help" option from the main menu or the pause menu during gameplay.
2. The system displays the help screen, which provides detailed information about the game. This screen is divided into sections for easy navigation, including:
    - **Control Scheme:** Explains key controls, such as arrow keys for movement, mouse clicks for interactions, and keyboard shortcuts for enchantment usage (e.g., "R" for Reveal, "P" for Cloak of Protection).
    - **Game Objective:** Describes the player's goal of finding runes in each hall, avoiding monsters, and progressing through all halls to win the game.
    - **Game Objects:** Details the key objects in the game, such as runes, walls, and enchantments, and explains how to interact with them.
    - **Monsters:** Provides an overview of the monster types (archer, fighter, wizard) and their behaviors, strengths, and weaknesses.
    - **Enchantments:** Describes each enchantment's function, such as extending time, revealing rune locations, or protecting the hero from monsters.
3. The player can navigate through sections using buttons or tabs, ensuring they can focus on specific topics of interest.
4. Once the player has read the required information, they select the "Back" or "Exit" button to leave the help screen.
5. The system returns the player to the previous menu or the game state (e.g., the main menu or the paused game).

Postconditions:

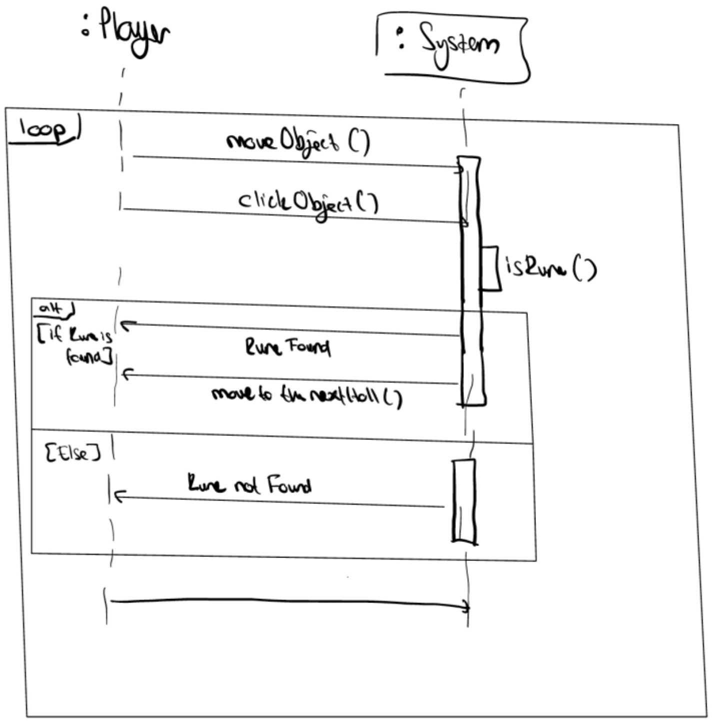
- The player gains clarity about how to play the game, including controls, mechanics, and strategies for interacting with game objects and avoiding threats.
- The player is better equipped to progress in the game, making the gameplay experience more intuitive and enjoyable.

Domain Model

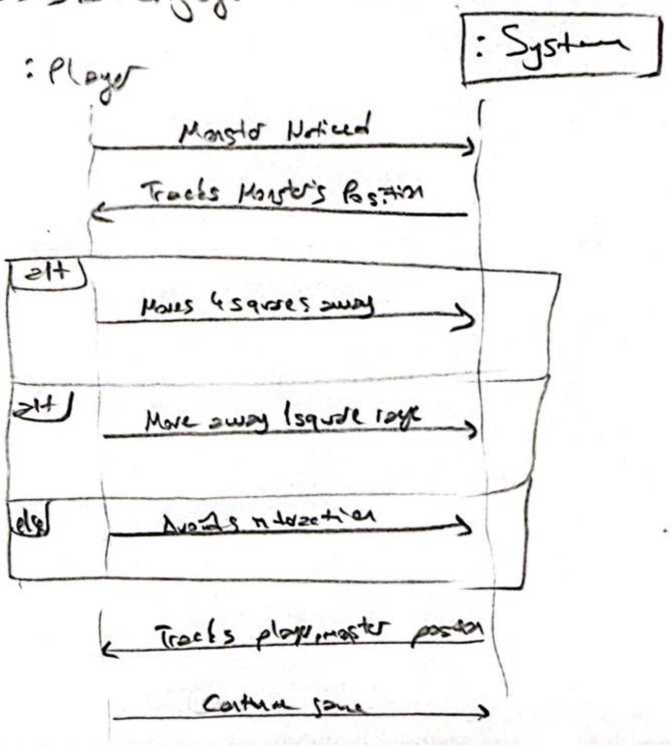


SSD’s

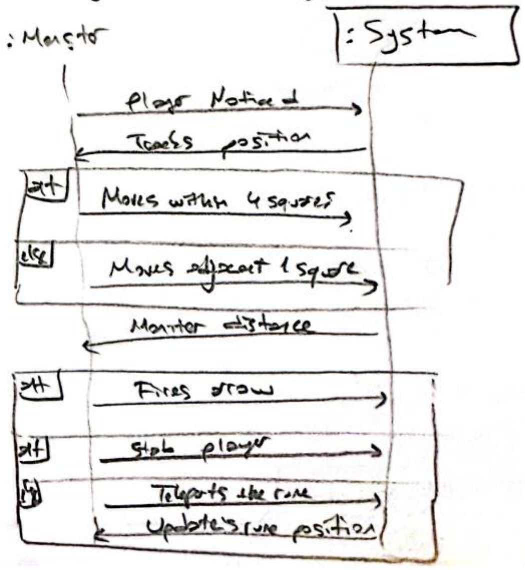
SSD 1 : Find Rune



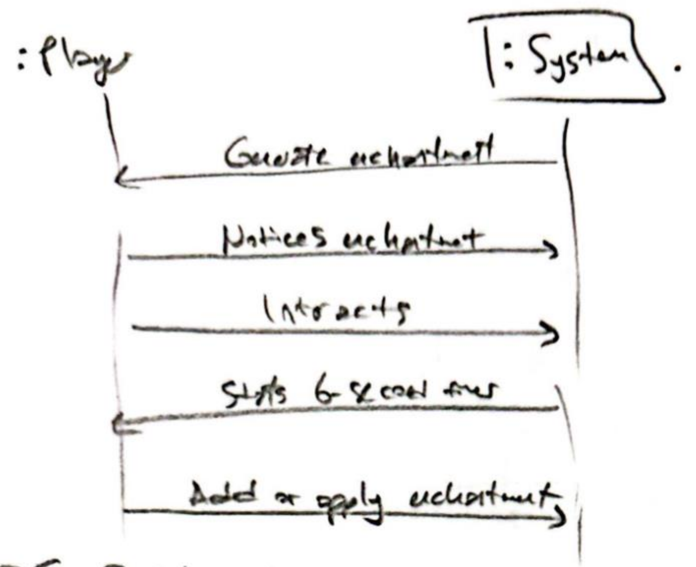
SSD2: Engage in Combat



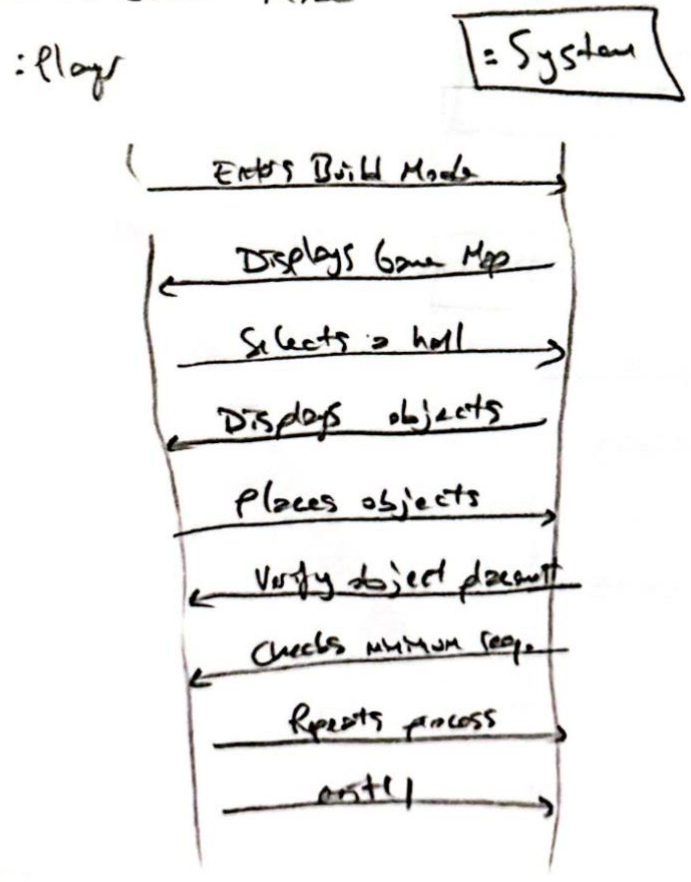
SSD3: Engage in Combat for Monsters



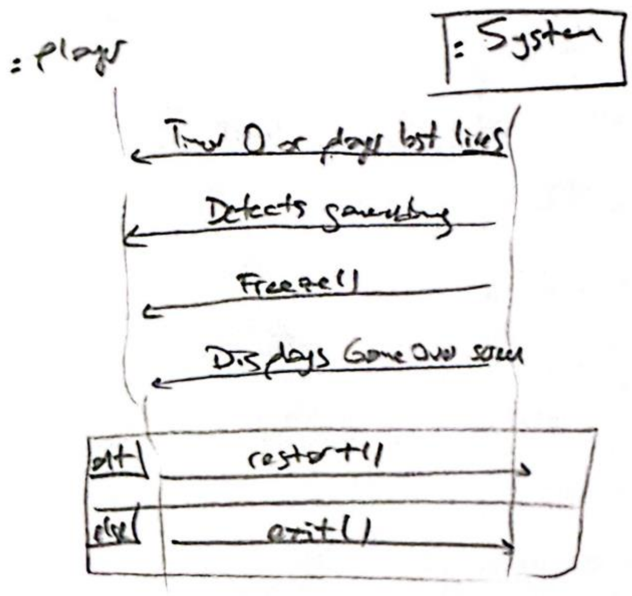
SSD4: Collect Enchantment



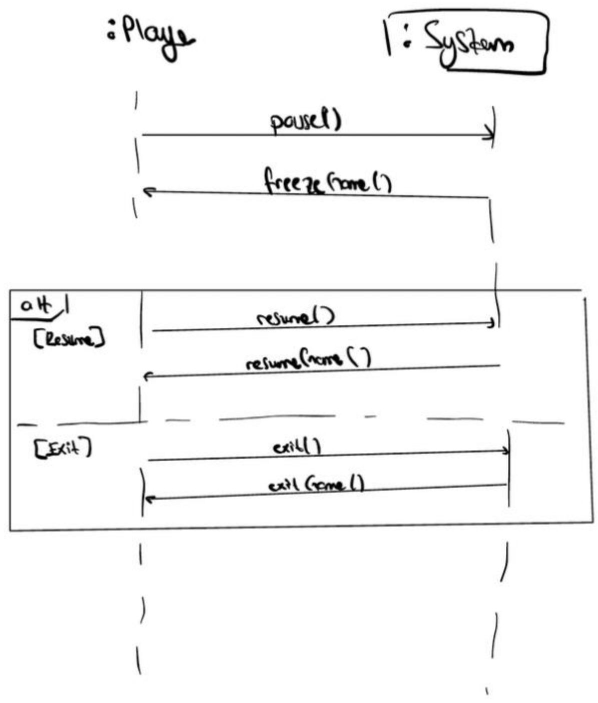
SSD5: Build Mode



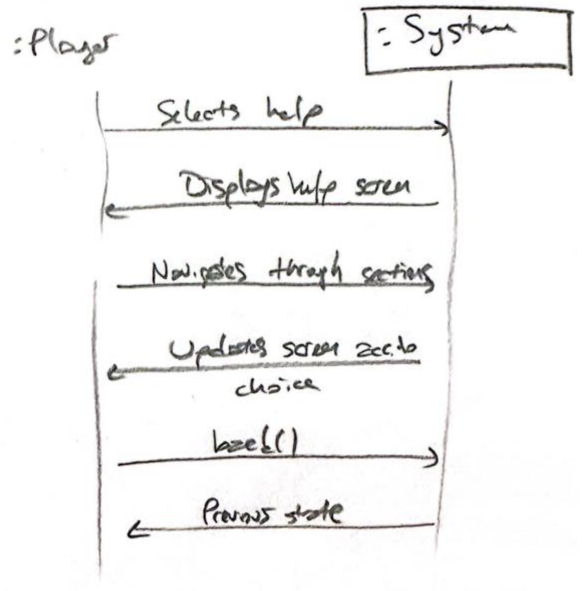
SSD6: Game Over



SSD 4 : Pause Game



SSD 8: View Help Screen



Operation Contracts

Operation Contract for `clickObject, SSD1`

Operation: `clickObject(object: GameObject)`

Cross-Reference: Find Rune (Use Case)

Preconditions:

1. The player is adjacent to the object.
2. The object exists within the current hall.
3. The game is not paused.

Postconditions:

1. **If the object contains a rune**:
   - The object's `containsRune` attribute is set to `false`.
   - The rune is revealed to the player.
   - The door to the next hall is unlocked.
   - A sound effect indicating success is triggered.
   - The system updates the player's score or progress.
2. **If the object does not contain a rune**:
   - No change is made to the object or game state.
3. **The game timer continues to count down unless paused.**

Operation Contract for `monsterAttacks, SSD2`
Operation: `monsterAttacks(player: Player)`
Cross-Reference: Engage in Combat for Player (Use Case)

Preconditions:

1. A monster exists in the current hall.
2. The monster's position is within its attack radius relative to the player.
3. The game is not paused.

Postconditions:

1. **If the player is within attack range**:
   - The player's `life` attribute is decremented by 1.
   - The monster's attack animation or effect is triggered.
   - If the player's `lives` reach 0:
     - The system triggers the "Game Over" use case.
2. **If the player is wearing a Cloak of Protection**:
   - The attack does not reduce the player's lives.
   - The monster attack is ignored for the duration of the cloak effect.

Operation Contract for `playerAttacks, SSD3`
Operation: `playerAttacks(monster: Monster)`
Cross-Reference: Engage in Combat for Monster (Use Case)

Preconditions:

1. The game is currently running.

2. A monster exists in the current hall.
3. The player’s position is within its attack radius relative to the monster.

Postconditions:

- **If the monster is within attack range**:
  - The monster's `life` attribute is decremented by 1.
  - The player's attack animation or effect is triggered.

    - If the monster's `lives` reach 0:
        - The monster dies.

Operation Contract for `freezeGame`
Operation: `freezeGame()`
Cross-Reference: Pause Game (Use Case)

Preconditions:

4. The game is currently running.
5. The player has clicked the "Pause" button.

Postconditions:

- **All game actions are frozen**:
    - The game timer stops decrementing.
    - Monster movements stop.
    - Player interactions (e.g., moving, clicking objects) are disabled.
    - Any ongoing animations or actions are paused.
- **The game state is updated**:
    - A flag (e.g., `isPaused`) is set to `true`.
- **The pause menu or overlay is displayed**:
    - A "Paused" screen appears, providing options to resume or exit.

Operation Contract for `collectEnchantment,SSD4`

Operation: `collectEnchantment(enchantment: Enchantment)`
Cross-Reference: Collect Enchantment (Use Case)

Preconditions:

1. The player notices the enchantment on the screen.

2. The enchantment is still available within the 6-second collection window.
3. The player's inventory is not full (if the enchantment requires inventory space).

Postconditions:

1. If the enchantment is successfully collected:
    - The enchantment is added to the player's inventory or applied immediately (e.g., extra life or time).
    - The enchantment is removed from the game world.
2. If the player's inventory is full:
    - The system rejects the enchantment, and no change occurs.
3. If the enchantment disappears before collection:
    - The enchantment is removed from the game world without effect.

Operation Contract for `buildHall, SSD5`

Operation: `buildHall(hall: Hall, objects: List[GameObject])`
Cross-Reference: Build Mode (Use Case)

Preconditions:

1. The player has entered Build Mode.
2. The hall grid is empty or partially filled.
3. The player has not placed more objects than the grid allows.

Postconditions:

1. If objects are placed successfully:
    - Each object is added to the hall grid without overlap or boundary violations.
    - The hall's `objectCount` is updated.
    - The hall's design is marked as "complete" if it meets the minimum object requirements.
2. If the player violates placement rules:
    - The system prevents the placement and displays an error message.
3. If the player removes or replaces an object:
    - The object is removed or replaced, and the grid is updated.

Operation Contract for `playerMoves`

Operation: `playerMoves(direction: String)`
Cross-Reference: Find Rune (Use Case)

Preconditions:

1. The player inputs a movement command (e.g., arrow keys).
2. The target tile is within the hall's boundaries.
3. The game is not paused.

Postconditions:

1. If the movement is valid:
   - The player's position is updated on the grid.
   - The game checks for interactions (e.g., traps, enchantments, monsters).
   - The system updates the fog of war to reveal newly visible tiles.
2. If the movement is invalid (e.g., wall or out-of-bounds):
   - The system prevents the movement and retains the player's current position.

Operation Contract for `timeRunsOut`

Operation: `timeRunsOut()`**, SSD6**
Cross-Reference: Game Over (Use Case)

Preconditions:

1. The game timer reaches zero.
2. The game is actively running.

Postconditions:

1. The game ends:
   - The "Game Over" screen is displayed with options to restart or exit.
   - All ongoing game actions (e.g., movements, animations) are frozen.
2. If the player selects "Restart":
   - The game resets all progress and begins a new session.
3. If the player selects "Exit":
   - The game session ends, and the player returns to the main menu.

Operation Contract for `viewHelp, SSD8`

Operation: `viewHelp()`
Cross-Reference: View Help Screen (Use Case)

Preconditions:

1. The player is on the main menu or has paused the game.

Postconditions:

1. The system displays the help screen with gameplay instructions, controls, and tips.
2. The player can navigate through different sections of the help screen.
3. The game resumes its previous state when the player exits the help screen.

Other Artifacts

1. Vision

Rokue-Like is a fun and exciting dungeon game where players explore rooms, solve challenges, and find hidden treasures while avoiding clever monsters. You'll need to beat the clock to uncover secrets and move to the next level. The game is designed to be simple to play but still super engaging, so whether you're a casual gamer or someone who loves a challenge, there's something for you. It's all about having a good time with creative gameplay and cool designs!"

2. Supplementary Specification (Non-Functional Requirements)

These are critical constraints and qualities that the system must satisfy.

Requirement Type	Description
Performance	The game must respond to player actions (e.g., movement, clicks) within 100ms.
Scalability	The game should support different screen sizes and resolutions.
Portability	The game should run on Windows, MacOS, and Linux platforms.
Usability	The interface should be intuitive for users with no prior gaming experience.
Reliability	The game should handle unexpected inputs or errors gracefully without crashing.
Security	Protect saved game progress and player data.
Accessibility	Include options for sound controls and colorblind-friendly designs.

3. Glossary

This defines key terms to ensure consistent understanding across the team.

Term	Definition
Player	The main character controlled by the user in the game.
Rune	A hidden artifact in the hall that the player must find to progress.
Hall	A room in the dungeon where the player searches for the rune while avoiding monsters.
Monster	Non-player entities in the hall that try to hinder the player's progress.
Enchantment	Collectible items that grant the player special abilities (e.g., Cloak of Protection).
Game Timer	A countdown mechanism that limits the time the player has to complete a hall.
Pause Menu	A screen that appears when the game is paused, allowing players to resume or exit the game.

Term	Definition
Grid	The structured layout of the hall, represented as rows and columns for positioning.