Use Cases

Use Case Name: Start Game

Use Case ID: UC-1
Scope: Rokue-Like Game
Level: User Goal
Primary Actor: Hero (player)

Stakeholders and Interests:

- **Player**: Wants to initiate a new game session and begin their adventure by designing the halls in Build Mode.
- **Developer**: Ensures the game initialises correctly and provides a seamless transition into Build Mode.

Preconditions:

- The game app is opened and running.
- The main menu comes up with the following options: "Start Game," "Help," and "Exit."

Postconditions:

- The game enters Build Mode, and the player can build the halls.

Main Success Scenario:

1. Player starts the game program, and the title menu comes up.
2. Player clicks on the "Start Game" button.
3. The system initialises the game state, setting up all required variables (e.g., hall layout, hero starting position, timer).
4. The system transitions to Build Mode, displaying an interactive grid for the player to design the halls.

Extensions:

3a. System encounters an error during initialisation (missing assets or files):

1. The system displays an error message, such as "Unable to load game assets. Please reinstall the game.".
2. The system terminates the process or allows the player to retry.

Frequency of Occurrence:
Every time the player starts a new game session.

Use Case Name: Build Map

Use Case ID: UC-2
Scope: Rokue-Like Game – Phase 1
Level: User Goal
Primary Actor: Player (Hero)

Stakeholders and Interests:

- **Player:** Wants to customise and design the layout of the halls by placing objects to improve strategy
- **Developer:** Wants the game to switch to Build Mode when "Play a New Game" is clicked. Wants the Build Mode to be user-friendly to design and to function without bugs.

Preconditions:

- Game must be in the "Build Mode".
- The player has access to objects to construct the map.

Postconditions:

- The halls are successfully designed, meeting all the specified criteria.
- The map is saved.
- The player can enter "Start Game".

Main Success Scenario:

1. The player selects "Play a New Game", and the Build Mode is entered.
2. A grid layout of the halls and the available objects and tools are presented.
3. The player designs the halls according to their strategy and desire.
4. The system validates the design according to the game criteria.
5. The player finalises the design and the system saves it.
6. The player starts the game by clicking "Start Game".

Extensions:

3a. The player tries to place an object at an invalid position (overlapping or out of grid):

1. System prevents the placement and alerts the player.

4a. The design doesn't meet the minimum criteria specified in the game rules:

1. System alerts the player (such as not enough objects).
2. Player adjusts the design accordingly.

5a. System encounters an error after the saving attempt:

1. System displays an error message and suggests retrying.

Special Requirements:

- Build Mode interface must be user-friendly and intuitive.
- The validation alert must be descriptive, including the object requirements for each hall.

Frequency of Occurrence:
At the beginning of each game.

Open Issues:

- What tools and objects are available to the player?
- What happens if the player exits without saving?
- Should the system auto-save to prevent data loss?

Use Case Name: Move Hero

Use Case ID: UC-3
Scope: Rokue-Like Game
Level: User Goal
Primary Actor: Hero (player)

Stakeholders and Interests:

- **Player:** Wants to navigate through the dungeon halls to explore, find runes, and avoid monsters, which is essential for progressing through the game.
- **Developer:** Ensures the movement system is accurately populated with the rules of the game, prevents invalid moves (for example, walking through walls), and interacts properly with game objects, monsters, and enchantments.

Preconditions:

- The game must be in Play Mode.
- The hero must have a valid starting position on the grid.

Postconditions:

- The hero's position is updated within the grid world.
- Any interactions with objects, monsters, or enchantments are processed.

Main Success Scenario:

1. The player presses an arrow key (or equivalent control) to move the hero in a direction (north, south, east, or west).
2. The system checks the hero's current position and determines if the move is valid (e.g., no walls or obstacles block the way).
3. If the move is valid, the system updates the hero's position and redraws the grid to reflect the new location.
4. If the hero encounters an interactive object (e.g., enchantment, rune, monster), the system processes the interaction based on game rules.
5. The hero continues moving as per player input, with the timer updating in real-time.

Extensions:

3a. The hero encounters a wall:

1. The system prevents movement and plays an error sound to indicate the wall is blocking the path.

4a. The hero encounters a monster:

1. If unprotected, the hero loses a life and is forced to retreat to the previous position.
2. If the hero has "Cloak of Protection," the monster does not react to the hero.

4b. The hero encounters an enchantment:

1. The system triggers the appropriate effect (e.g., extra life, extra time, or item added to inventory).

4c. The hero steps onto a square adjacent to a rune:

1. The system allows the hero to interact with the object to reveal the rune.

Frequency of Occurrence:
Hero movement occurs frequently throughout the game as it is a core mechanic.

Use Case Name: Check for Rune

Use Case ID: UC-4
Scope: Rokue-Like Game – Phase 1
Level: User Goal
Primary Actor: Player (Hero)

Stakeholders and Interests:

- **Player:** Wants to find a rune to unlock the next hall and to win or progress in the game.
- **Developer:** Ensures the game rules and wants to provide a challenging environment.

Preconditions:

- The player has entered the hall.
- The hall contains several objects, under which a rune could be hidden.

Postconditions:

- The player selects the object under which a rune is hidden.
- The rune is found, allowing the gate for the next hall to be unlocked.

Main Success Scenario:

1. Player approaches an object and clicks on it.
2. System checks whether the selected object contains the rune.
3. Rune is found underneath the object.
4. The doors of the next hall are unlocked.

Extensions:

2a. Selected object does not contain the rune:

1. System provides negative feedback.
2. Player continues to search for the rune.

Special Requirements:

- Interaction should be through click or key press.
- There should be a clear feedback for both success and failure.

Frequency of Occurrence:
Frequent. Each hall includes multiple occurrences.

Use Case Narrative: Collect Enchantment

Use Case ID: UC-5
Scope: Rokue-Like Game
Level: User Goal
Primary Actor: Player (Hero)

Stakeholders and Interests:
- **Player (Hero)**: Aims to improve gameplay and enhance survivability via enchantments.
- **Game System**: Maintains the availability and effects of enchantments to better balance the gaming experience.

Preconditions:
- Enchantments appear around the game environment at random.

Postconditions:
- The player picks up an enchantment, which automatically activates or stores for later use.

Main Success Scenario (MSS):
1. An enchantment spawns at a random position in the game.
2. The player moves to the enchantment's location.
3. The player picks up and collects the enchantment.
4. The system identifies and applies the enchantment's effect:
   Extra Time: Adds extra time to the level timer.
   Extra Life: Increases the hero's life count.
   Storable Enchantment: Adds the item to the hero's inventory for later use.
5. The system confirms the collection with audio and visual cues.

Extensions (Alternative Paths):

3a. Enchantment disappears before collection:
1. The enchantment vanishes if not collected in time, and the player misses the opportunity.

5a. Inventory is full (storable enchantments only):
1. The system notifies the player that item management is required to make space.

Special Requirements:
- Enchantments should be visually distinct and easily recognizable.
- Collection timing should be balanced to maintain gameplay fairness.

Frequency of Occurrence:
Frequently, integral to gameplay dynamics.

Open Issues:

- How does the game ensure balanced enchantment distribution?
- Details on the mechanics of activating stored enchantments from inventory.

Use Case Name: Cast Inventory Enchantment

Use Case ID: UC-6
Scope: Rokue-Like Game – Phase 1
Level: User Goal
Primary Actor: Player (Hero)

Stakeholders and Interests:

- **Player:** Wants to use the collected enchantments to gain advantages in the game (revealing the location of the rune, hiding self from archer or fool the fighter monster)
- **Developer:** Wants to make the enchantments functional, bug-free and provide the expected rewards.

Preconditions:

- Player has stored at least one corresponding enchantment in the bag.

Postconditions:

- The selected enchantment is activated, rewarding the player.

Main Success Scenario:

1. Player selects the enchantment they want to use by clicking on the corresponding button (R,P or B).
2. Enchantment is activated:
   Cast Reveal: Hints the location of the rune to the player, if the player presses R button. For 10 seconds, a 16 square region containing the rune becomes highlighted.
   Cast Cloak of Protection: Hides the player from the archer monster, if the player presses P button. For 20 seconds, player becomes invisible from archer by wearing the coat.
   Cast Luring Gem: If the player presses B button, the gem distracts the fighter monster towards a selected location. The player throws the luring gem by pressing A, D, W, or S.
3. System applies the effects to the environment during the designated time.

Extensions:

1a. There is no enchantment in the bag for the selected key:

1. Display error “No enchantment available for this action”.

Special Requirements:

- Enchantments should be visually distinct and easily recognizable.

Frequency of Occurrence:
Frequent, integral to gameplay dynamics.

Open Issues:

- Should the player be able to use multiple enchantments at the same time?
- Should the system notify when an enchantment is about to expire?
- Should there be a cooldown time before the player can use the same enchantment again?

Use Case Narrative: Exit Hall

Use Case ID: UC-7
Scope: Rokue-Like Game
Level: User Goal
Primary Actor: Player (Hero)

Stakeholders and Interests:

- **Player (Hero):** Seeks to progress through the game by exiting halls after completing objectives.
- **Game System:** Handles the transition between game levels and verifies if all the criteria for completion have been met.

Preconditions:

- In the hall, the hero has found the needed rune.
- The exit door is unlocked, opening up to proceed to the next level.

Postconditions:

- The hero successfully exits the hall.
- The game progresses to the next hall or concludes if it is the last level.

Main Success Scenario (MSS):

1. The player moves the hero to the exit door.
2. The player attempts to leave by interacting with the door.
3. The system acknowledges that the door is unlocked.
4. A transition sequence is activated, leading to the next hall.
5. The hero is positioned at the start, and the next environment loads.
6. The game updates all related displays and states for the new hall.

Extensions (Alternative Paths):

3a. If the door is locked:

1. The system alerts the player, and exit is denied.

4a. Exiting the final hall:

1. The game initiates the completion sequence and shows end-game options.

Special Requirements:

- Ensure seamless and visually engaging transition effects.
- Enhance environment loading times to enable smooth gameplay transitions.

Frequency of Occurrence:

Regularly at the end of each hall.

Open Issues:

- How does one handle premature exit attempts by the player?
- What other requirements may be necessary for hall exits?

Use Case Name: Pause Game

Use Case ID: UC-8
Scope: Rokue-Like Game
Level: User Goal
Primary Actor: Hero (player)

Stakeholders and Interests:

- **Player:** Wants to temporarily stop gameplay and resume it later without losing progress, providing flexibility in how they interact with the game.
- **Developer:** Ensures transition between pause and resume states is seamless and no progress or game state is lost during this interaction.

Preconditions:

- The game must be running in Play Mode.
- The "Pause" button must be accessible in the user interface.

Postconditions:

- **On Pause:** Gameplay is halted entirely, including animations, timer countdowns, and actions of both the hero and monsters.
- **On Resume:** Gameplay continues from the exact point it was paused, with all activities restored seamlessly.

Main Success Scenario:

1. During gameplay, the player clicks the "Pause" button available on the screen.
2. The system halts all ongoing game activities, including timer countdowns, hero movement, monster actions, and any animations.
3. The system displays a pause screen or updates the "Pause" button to a "Resume" button to indicate the paused state.
4. While paused, the player is unable to interact with game elements but may choose to exit to the main menu or adjust settings (if available).
5. The player clicks the "Resume" button.
6. The system restores all gameplay activities to their previous state and updates the button back to "Pause."

Extensions:

2a. The player exits the game while paused:

1. The system saves the current state automatically (if supported).
2. The system transitions back to the main menu.

4a. The player tries to interact with gameplay elements (e.g., move the hero, collect enchantments):

1. The system prevents the action and displays a message, such as "Game is paused. Resume to continue."

Frequency of Occurrence:
Players may pause the game at any time during Play Mode, potentially multiple times in a single session.