KOÇ ÜNİVERSİTESİ

Fall 2024
Software Engineering - COMP 302
Project *“gameName”*
Final Report

Group: groupName

Enes Ak – 80090

Muhammed Babelli - 84342

Yusuf Cemâl Karataş - 83639

İbrahim Cebecioğlu - 79906

Caner Kösem - 80246

Cemal Nişan – 79158

Table of Contents

REQUIREMENTS ANALYSIS - MODELING **5**

VISION **5**

INTRODUCTION 5
POSITIONING 5
Business Opportunity *5*
Problem Statement *5*
Product Position Statement *5*
Alternatives and Competition *6*
Stakeholder Descriptions *6*

USE CASE DIAGRAM **7**

USE CASE NARRATIVES **8**

UCN 1: MOVE 8
UCN 2: PLACE STRUCTURE 10
UCN 3: REMOVE STRUCTURE 11
UCN 4: START PLAY MODE 12
UCN 5: EXPLORE STRUCTURE 14
UCN 6: USE ENCHANTMENT 16
UCN 7: COLLECT ENCHANTMENT 18
UCN 8: COLLECT RUNE 19
UCN 9: SAVE GAME 21

DOMAIN MODEL **22**

SYSTEM SEQUENCE DIAGRAMS **23**

SSD 1: GETHELP 23
SSD 2: COLLECTENCHANTMENT 23
SSD 3: EXPLORESTRUCTURE 24
SSD 4: PAUSEGAME 24
SSD 5: USEENCHANTMENT 25
SSD 6: PLACESTRUCTURE 25
SSD 7: REMOVESTRUCTURE 26
SSD 8: RESUMEGAME 26
SSD 9: SAVEGAME 27
.... 27
SSD 10: LOADGAME 27

OPERATION CONTRACTS **28**

OC 1: COLLECTENCHANTMENT 28
OC 2: EXPLORESTRUCTURE 28
OC 3: MOVEPLAYER 29
OC 4: USEENCHANTMENT 29
OC 5: PLACESTRUCTURE 30

OC 6: exitGame 30
OC 7: removeStructure 31
OC 8: runeCollected 31

SUPPLEMENTARY SPECIFICATIONS 32

Introduction 32
Functionality 32
Logging and Error Handling *32*
Pluggable Rules *32*
Security *32*
Usability 33
Human Factors *33*
Performance *33*

GLOSSARY 34

EXTRA FEATURES 36

Random Map Initilization 36
New Enchantment: Speed Up! 36
Custom Pixel Arts 36
Sound Effects 37
Try Again Button 37

DESIGN 38

LOGICAL ARCHITECTURE 38

SEQUENCE DIAGRAMS 39

SD 1: Start BuildMode 39
SD 2: Move Player 40
SD 3: Explore Structure 41
SD 4: Start PlayMode 42
SD 5: Use Luring Gem 43
SD 6: Use Cloak of Protection 44
SD 7: Extra Life 45
SD 8: Use Reveal 46
SD 9: Archer vs Player 47
SD 10: Find Rune 48
SD 11: Wizard Behavior 49
SD 2: Wizard Disappear 50

COMMUNICATION DIAGRAMS 51

CD 1: Start BuildMode 51
CD 2: Move Player 52
CD 3: Explore Structure 53
CD 4: Start PlayMode 54
CD 5: Use Luring Gem 55
CD 6: Use Cloak of Protection 56

CD 7: Extra Life 57
CD 8: Use Reveal 58
CD 9: Archer vs Player 59
CD 10: Find Rune 60

CLASS DIAGRAM 61

DESIGN ALTERNATIVES 62

REQUIREMENTS ANALYSIS - MODELING

Vision

Introduction

We envision a next-generation, interactive, and immersive dungeon adventure game, *gameName*" designed to offer players an engaging experience of exploration, puzzle-solving, and strategy. The game combines dynamic gameplay mechanics, such as monster encounters and enchantment collection, with user-friendly customization features, enabling players to create their unique journey through customizable hall designs.

Positioning

Business Opportunity

The current landscape of 2D dungeon adventure games often lacks sufficient user control over the environment and gameplay mechanics. Existing games rarely provide players with the ability to design their own game environment or introduce strategic elements like enchantments and diverse monster behaviors. This creates a demand for a customizable game that balances user creativity with challenging gameplay. "*gameName*" addresses this gap by providing a unique dungeon adventure experience with a fully customizable hall-building mode, dynamic monster encounters, and strategic use of enchantments.

Problem Statement

Many adventure games lack modularity and fail to provide an experience designed to both casual and strategic players. This project aims to resolve these issues by integrating a grid-based environment that supports seamless game mechanics and strategic elements, while maintaining accessibility and visual appeal. Our approach targets both the entertainment of gaming by creating a scalable game that adheres to software engineering best practices.

Product Position Statement

"*gameName*" is a dungeon exploration game designed for adventure enthusiasts and gamers seeking an interactive, strategic, and customizable experience. Its standout features include:

- A hall-building mode that enables players to design game levels.

- Diverse monster interactions requiring strategic thinking.

- A range of enchantments that add depth to gameplay.

The game differentiates itself through its integration of design, exploration, and strategy in a single package.

Alternatives and Competition

Existing 2D dungeon games emphasize either aesthetics or gameplay complexity, often neglecting user-driven customization and modularity. "*gameName*" fills this gap by combining user creativity with an engaging, action-packed experience.

Stakeholder Descriptions

Players: Seek an immersive and customizable gaming experience.

Developers: Gain hands-on experience in implementing modular software systems.

Use Case Diagram

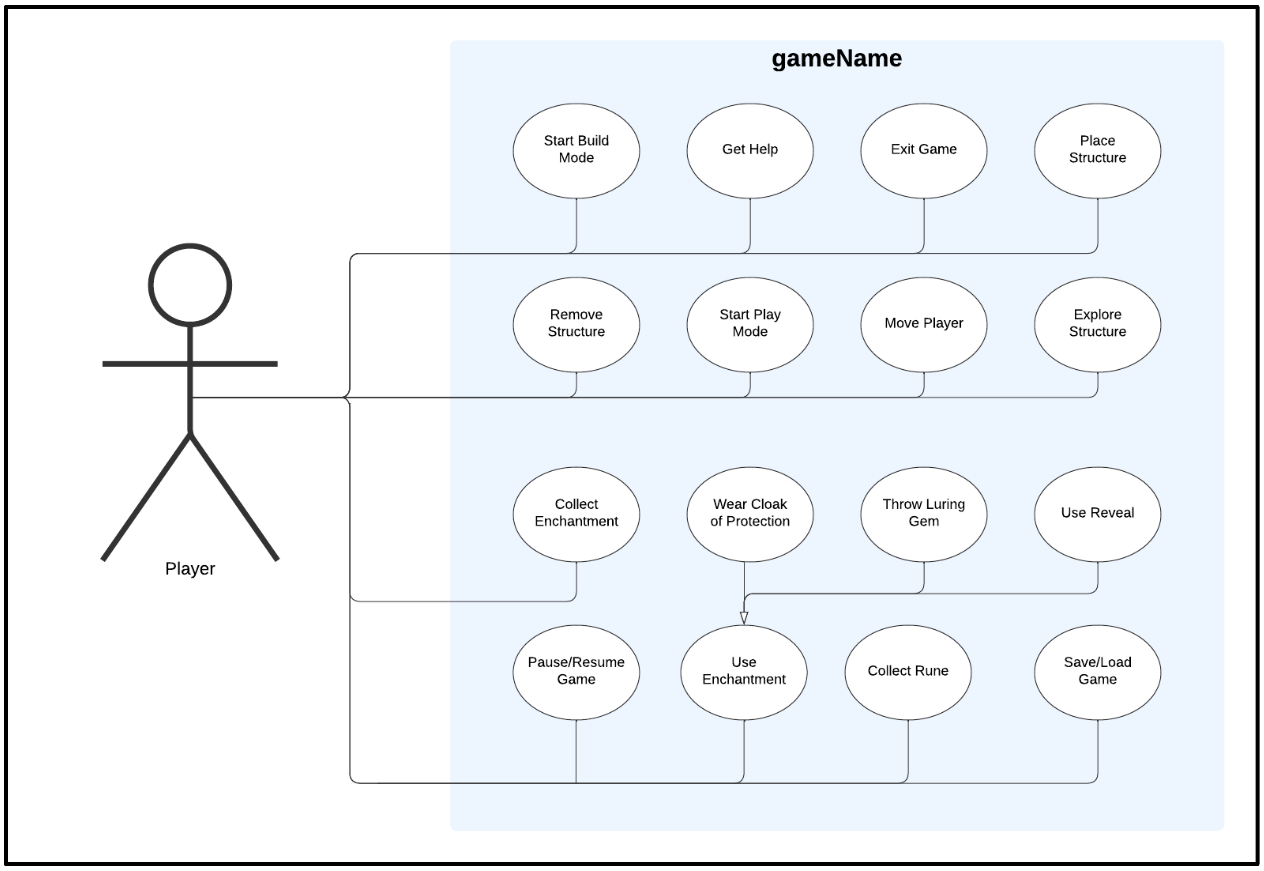


Figure 1: Use Case Diagram

Use Case Narratives

UCN 1: Move

Use Case Name	Move
Scope	Play Mode
Primary Actor	Player
Description	This use case describes the process by which the player controls the hero to navigate through the dungeon halls. The player attempts to move the hero character in one of four cardinal directions (up, down, left, or right) using the game controls. Movement is restricted by the game environment, which includes walls, structures, monsters, and other objects placed on the grid.
Pre-Condition/s	• The game must be in play mode, and the hero must be in an active hall. • The hero must have at least one valid direction available for movement.
Post-Condition/s	The hero's position is updated on the grid to reflect the intended movement if the target tile is valid.
Main Success Scenario (Basic Flow)	1. The player decides on a direction to move the hero (right, left, up, or down). 2. The game receives the input and calculates the hero's intended destination tile based on the direction pressed. 3. The game verifies that the target tile is valid for movement (i.e., not occupied by walls, monsters, or other objects). 4. If the target tile is valid, the game updates the hero's position on the grid to the new tile. 5. The screen refreshes to show the updated position of the hero.

Extensions (Alternate Flow/s)	4a) Occupied Tile: • The game checks the intended tile and finds it occupied by a structure, monster, or wall. • The game prevents the hero from moving and retains their current position.

UCN 2: Place Structure

Use Case Name	Place Structure
Scope	Build Mode
Primary Actor	Player
Description	This use case describes the process by which the player places structures in specific locations on the grid during build mode. Structures serve strategic purposes, such as hiding runes or influencing gameplay. The player is responsible for meeting the minimum structure requirements for each hall.
Pre-Condition/s	• The game must be in build mode. • The selected tile must be empty
Post-Condition/s	The chosen structure is placed on the selected tile in the grid, and the hall layout is updated.
Main Success Scenario (Basic Flow)	1. The player opens the Structure Storage to view available structures. 2. The player selects a structure from the storage. 3. The player chooses a specific tile in the hall grid to place the structure. 4. The game verifies that the selected tile is valid (i.e., not occupied by another object or outside the grid boundaries). 5. If the tile is valid, the structure is placed on the grid, and the hall's layout is updated to reflect the change. 6. A visual confirmation is displayed, showing the structure in its new position.
Extensions (Alternate Flow/s)	4a) **Tile Occupied:** • The game checks the selected tile and finds it already occupied. • The game prevents the structure from being placed.

UCN 3: Remove Structure

Use Case Name	Remove Structure
Scope	Build Mode
Primary Actor	Player
Description	This use case describes the process by which the player removes a structure previously placed on the grid during build mode. Removing structures allows the player to refine the hall's design to meet strategic goals or fulfill minimum requirements.
Pre-Condition/s	• The game must be in build mode. • The selected tile must contain a structure
Post-Condition/s	The structure is removed from the selected tile, and the grid is updated accordingly.
Main Success Scenario (Basic Flow)	1. The player identifies a structure on the grid they want to remove and selects it. 2. The game detects the structure at the selected location and confirms the interaction. 3. The game removes the structure from the grid, leaving the tile empty. 4. The hall layout is updated to reflect the removal, and the change is visually displayed to the player.
Extensions (Alternate Flow/s)	None

UCN 4: Start Play Mode

Use Case Name	Start Play Mode
Scope	Build mode
Primary Actor	Player
Description	This use case describes the process by which the player transitions the game from Build Mode to Play Mode. The transition occurs after the player completes the design of the dungeon halls and ensures all required structures are in place. Play Mode begins with the initialization of the game environment, including monsters, runes, and enchantments.
Pre-Condition/s	• The game is in build mode. • The player must satisfy all minimum structure placement requirements for each hall.
Post-Condition/s	The game transitions into Play Mode, initializing the first hall and preparing all game elements for active gameplay.
Main Success Scenario (Basic Flow)	1. The player completes the hall designs by placing the required number of structures in each hall. 2. The player clicks "Start Play Mode" to initiate the transition. 3. The game verifies that all halls meet the minimum structure placement requirements. 4. The game transitions from Build Mode to Play Mode. 5. The first hall (Hall of Earth) is initialized, with the hero randomly placed in a valid starting location. 6. The game spawns initial monsters, hides the rune under a random structure, and places enchantments. 7. The Play Mode interface is displayed, showing the current hall name, remaining time, lives, and the hero's inventory.

Extensions (Alternate Flow/s)	3a) **Requirements not satisfied:** • The game checks if the build mode requirements are met. • If the requirements are not satisfied (e.g., a hall has fewer than the minimum required structures), the game prevents the transition to Play Mode. • The player is shown an error message indicating which requirements are not met.

UCN 5: Explore Structure

Use Case Name	Explore Structure
Scope	Play Mode
Primary Actor	Player
Description	This use case describes the process by which the player interacts with a structure in the hall to determine if it contains a rune. The player must navigate the hero to a position adjacent to the structure and interact with it to uncover its contents.
Pre-Condition/s	• The game must be in play mode. • The hero must be adjacent to the structure being explored.
Post-Condition/s	If the structure contains a rune, it is revealed, and the door to the next hall is unlocked.
Main Success Scenario (Basic Flow)	1. The player navigates the hero to a tile adjacent to a structure on the grid. 2. The player clicks on the structure to initiate exploration. 3. The game verifies that the hero is in a valid position adjacent to the structure. 4. The game checks if the structure contains the rune. 5. If the rune is found, it is revealed, and the door to the next hall is unlocked. 6. A sound effect and visual animation indicate that the rune has been discovered.
Extensions (Alternate Flow/s)	3a) **Hero Not Adjacent to Structure:** • The game detects that the hero is not in a valid position to interact with the structure. • The game prevents the interaction. 4a) **Structure Does Not Contain Rune:**

	• The game checks the structure and finds no rune. • The player must continue exploring other structures to locate the rune.

UCN 6: Use Enchantment

Use Case Name	Use Enchantment
Scope	Play Mode
Primary Actor	Player
Description	This use case describes the process by which the player activates a collected enchantment to assist in gameplay. Enchantments provide strategic advantages, such as avoiding monsters or gaining extra time, and can be used immediately or stored in the hero's inventory for later.
Pre-Condition/s	• The game must be in play mode. • The desired enchantment must be present in the player's inventory.
Post-Condition/s	The selected enchantment is activated, and its effect is applied to the hero or the game environment.
Main Success Scenario (Basic Flow)	1. The player opens the hero's inventory to view available enchantments. 2. The player selects an enchantment from the inventory and activates it. 3. The game verifies that the enchantment is present in the inventory. 4. The game applies the enchantment's effect (e.g., adding time, providing protection, or distracting monsters). 5. For time-limited enchantments, the game starts a countdown to end the effect.
Extensions (Alternate Flow/s)	2a) **Enchantment Not Present:** • The game checks the inventory and finds the selected enchantment is missing. • The game prevents activation and notifies the player. 4a) **Effect Invalid in Current Context:** • The enchantment effect cannot be applied due to the current game state (e.g., no monsters for the Luring Gem).

	• The enchantment remains in the inventory.

UCN 7: Collect Enchantment

Use Case Name	Collect Enchantment
Scope	Play Mode
Primary Actor	Player
Description	This use case describes the process by which the player collects enchantments that appear on the grid. Enchantments can provide immediate benefits (e.g., extra time or lives) or be stored in the hero's inventory for later use.
Pre-Condition/s	• The game must be in play mode. • The enchantment must be present on the tile.
Post-Condition/s	The enchantment is collected, and its effect is either applied immediately or stored in the player's inventory for later use.
Main Success Scenario (Basic Flow)	1. The player notices an enchantment on a grid tile. 2. The player clicks on the tile containing the enchantment. 3. The game verifies that the clicked tile contains an enchantment. 4. The game removes the enchantment from the grid. 5. The game applies the enchantment effect immediately (for instant effects) or stores it in the hero's inventory for future use.
Extensions (Alternate Flow/s)	3a) **Tile Does Not Contain Enchantment:** • The game checks the clicked tile and finds no enchantment. • No action is performed, and the player receives feedback via sound or visual cue. 4a) **Enchantment Expires Before Collection:** • The game detects that the enchantment has expired before the player's interaction. • The enchantment disappears from the grid and cannot be collected.

UCN 8: Collect Rune

Use Case Name	Collect Rune
Scope	Search Rune Controller
Primary Actor	Player
Description	This use case describes the process by which the player interacts with a structure in the hall to determine if it contains a rune. The player must navigate the hero to a position adjacent to the structure and interact with it to uncover its contents.
Pre-Condition/s	• The game must be in play mode. • The hero must be adjacent to the structure being explored.
Post-Condition/s	If the structure contains a rune, it is revealed, and the door to the next hall is unlocked.
Main Success Scenario (Basic Flow)	7. The player navigates the hero to a tile adjacent to a structure on the grid. 8. The player clicks on the structure to initiate exploration. 9. The game verifies that the hero is in a valid position adjacent to the structure. 10. The game checks if the structure contains the rune. 11. If the rune is found, it is revealed, and the door to the next hall is unlocked. 12. A sound effect and visual animation indicate that the rune has been discovered.
Extensions (Alternate Flow/s)	3a) **Hero Not Adjacent to Structure:** • The game detects that the hero is not in a valid position to interact with the structure. • The game prevents the interaction. 4a) **Structure Does Not Contain Rune:**

	• The game checks the structure and finds no rune. • The player must continue exploring other structures to locate the rune.

UCN 9: Save Game

Use Case Name	Save Game
Scope	Game Management Controller
Primary Actor	Player
Description	This use case describes the process by which the player saves their current game progress. The player interacts with the game menu to initiate the save action, ensuring their progress is stored for future sessions.
Pre-Condition/s	• The game must be in play mode and paused.
Post-Condition/s	• The game state is saved to a specified save slot or file.
Main Success Scenario (Basic Flow)	1. The player pauses the game. 2. The player clicks the "Save Game" button. 3. The game displays a list of available save slots or prompts the player to name the save file. 4. The player selects a save slot or enters a save file name. 5. The game saves the current game state, including the hero's position, inventory, and progress.
Extensions (Alternate Flow/s)	4a) **No Available Save Slots:** - The game detects that all save slots are full. - The game prompts the player to overwrite an existing save or cancel action. 5a) **Save Action Fails:** - The game encounters an error while saving (e.g., insufficient storage). - The game displays an error message and does not save progress.

Domain Model

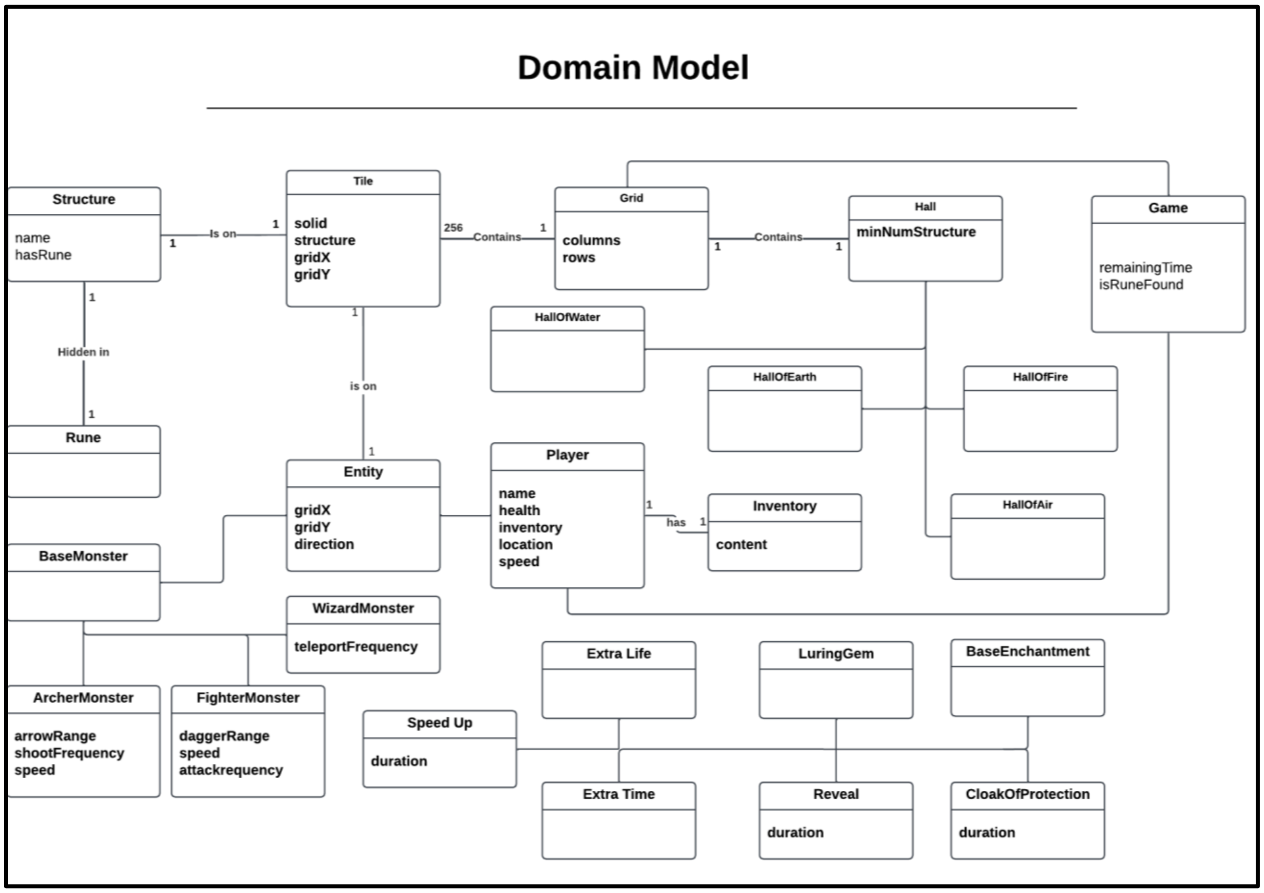


Figure 2: Domain Model

System Sequence Diagrams

SSD 1: getHelp

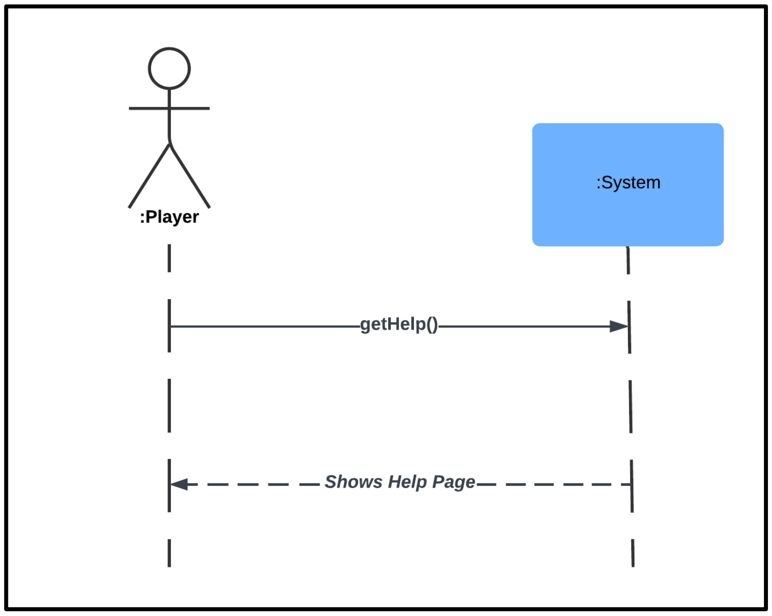


Figure 3: SSD 1 getHelp

SSD 2: collectEnchantment

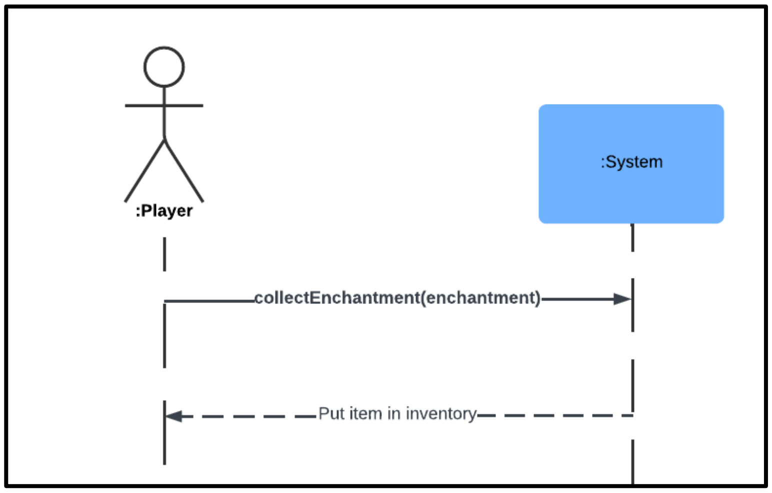


Figure 4: SSD 2 collectEnchantment

SSD 3: exploreStructure

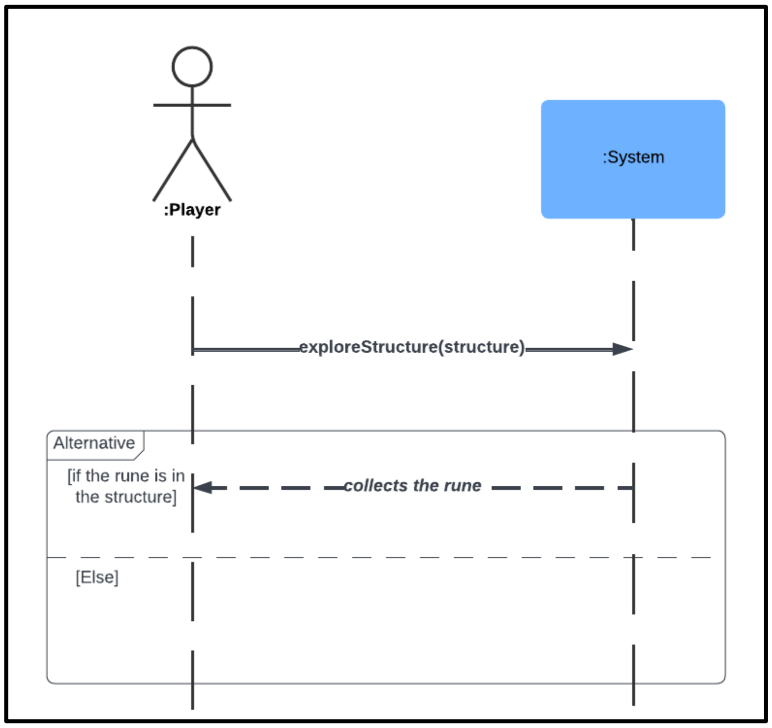

:Player
:System
exploreStructure(structure)
Alternative
[if the rune is in the structure]
collects the rune
[Else]

Figure 5: SSD 3 exploreStructure

SSD 4: pauseGame

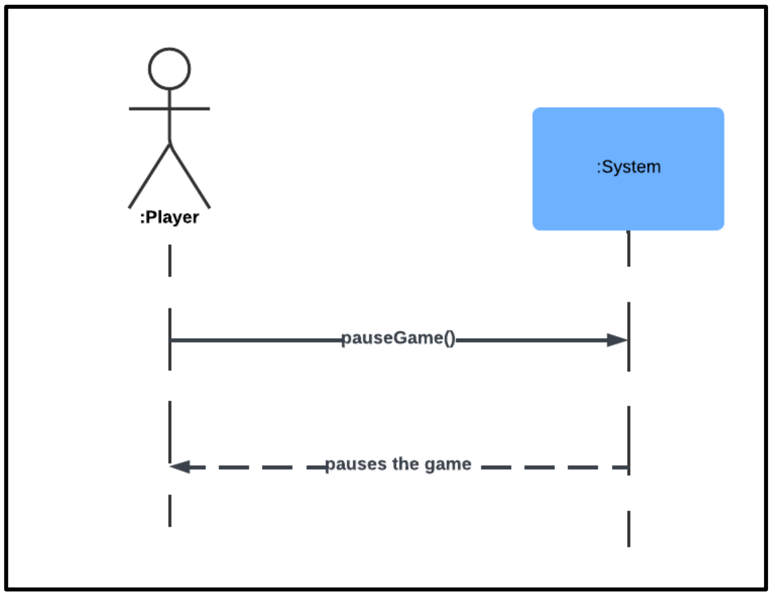


Figure 6: SSD 4 pauseGame

SSD 5: useEnchantment

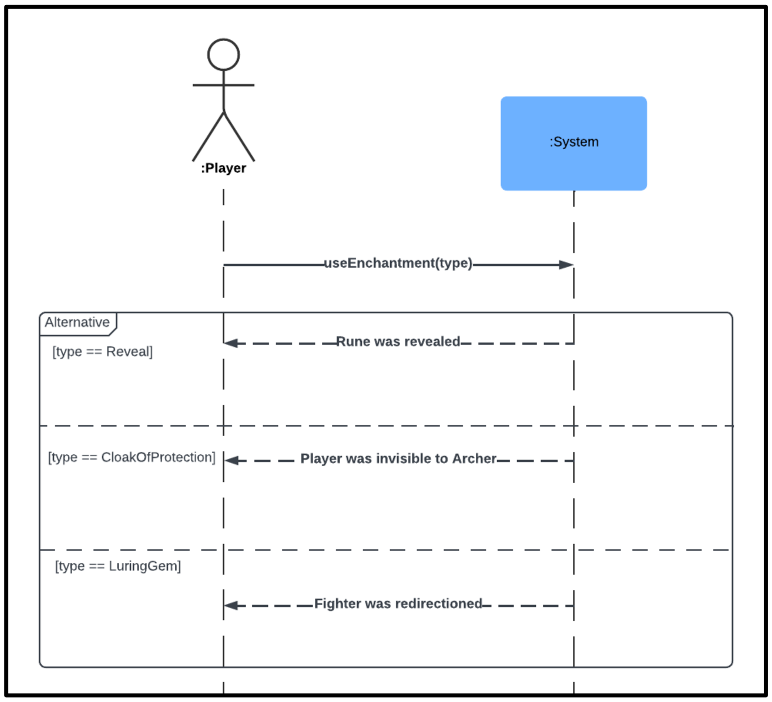


Figure 7: SSD 5 useEnchantment

SSD 6: placeStructure

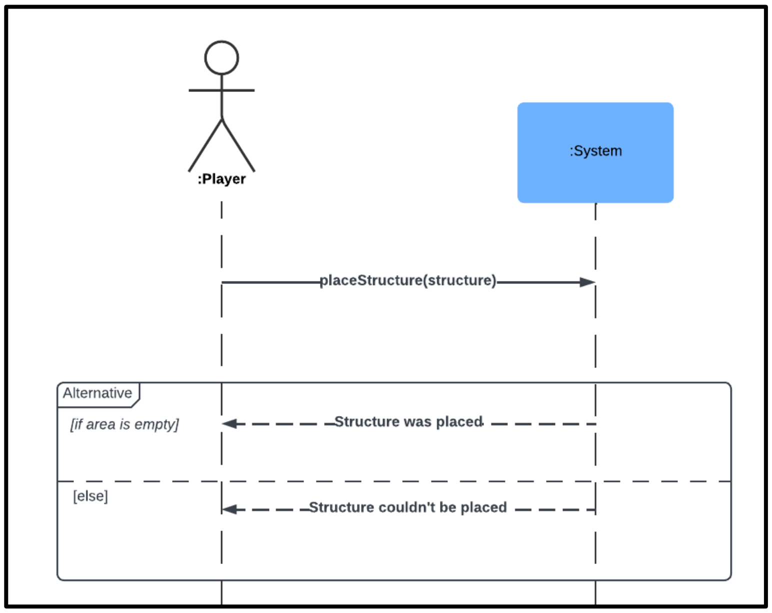


Figure 8: SSD 6 placeStructure

SSD 7: removeStructure

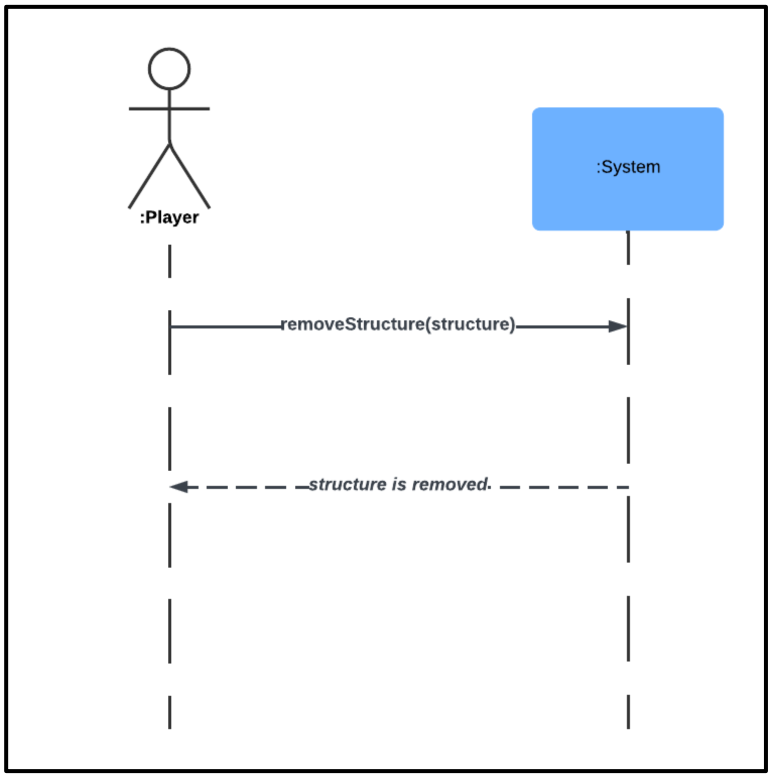


Figure 9: SSD 7 removeStructure

SSD 8: resumeGame

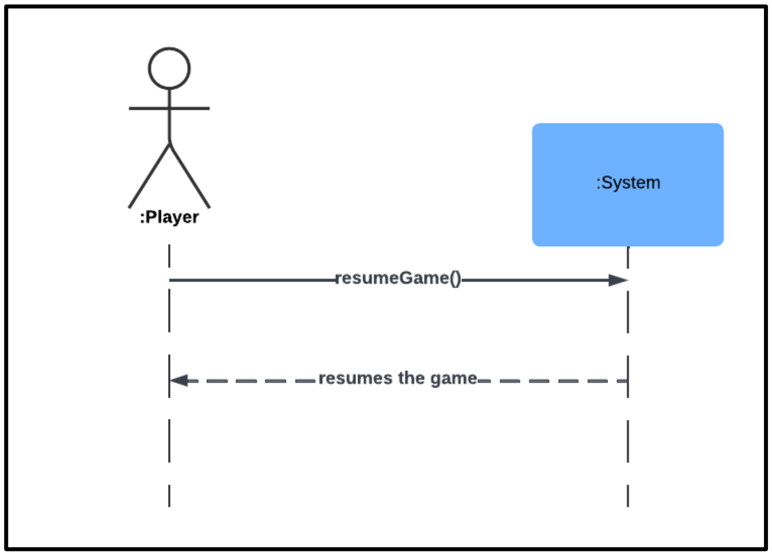


Figure 10: SSD 8 resumeGame

SSD 9: saveGame

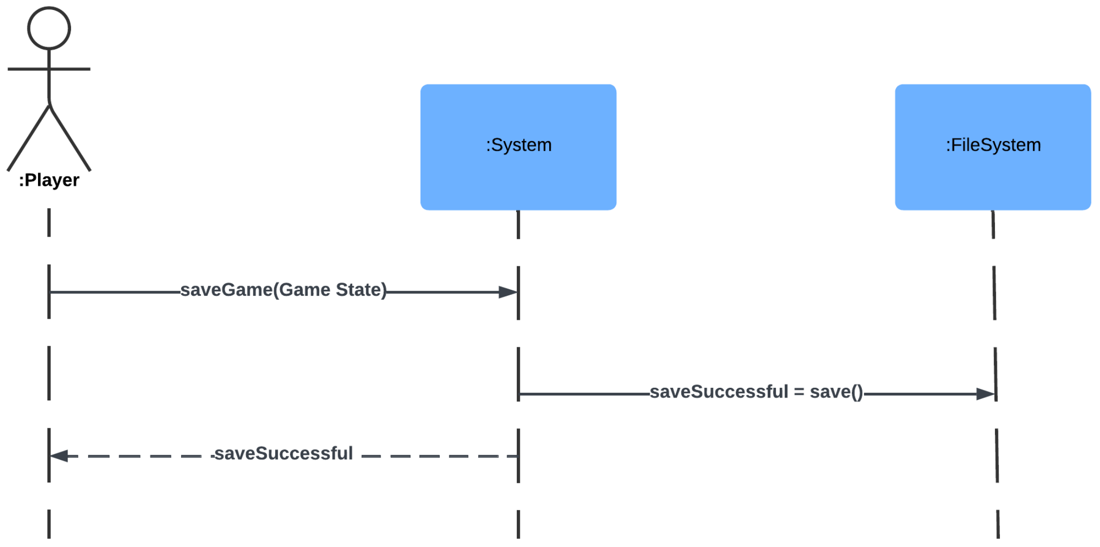


Figure 11: SSD 9 saveGame

SSD 10: loadGame

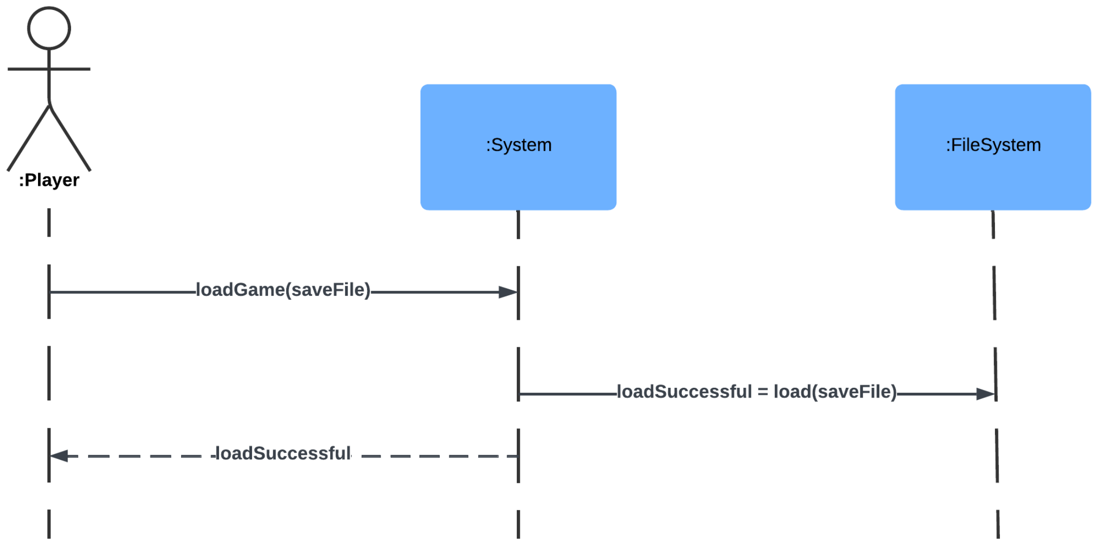


Figure 12: SSD 10 loadGame

Operation Contracts

OC 1: collectEnchantment

Operation	collectEnchantment(location: Tile)
Cross References	Use Cases: Collect Enchantment
Preconditions	• The Game is in the Play Mode. • There is an Enchantment in *location* in question.
Postconditions	• The Enchantment was added to inventory or directly used. • The Enchantment was removed from the *location*.

OC 2: exploreStructure

Operation	exploreStructure(structure: Structure)
Cross References	Use Cases: Explore Structure
Preconditions	• The Game is in the Play Mode. • There is a Structure in *location* in question. • Player must be within the one tile of the Structure.
Postconditions	• The rune was revealed if it was in this *structure.*

OC 3: movePlayer

Operation	movePlayer(direction: Enum)
Cross References	Use Cases: Move Player
Preconditions	• Game is in Play Mode
Postconditions	• Location of the player was updated.

OC 4: useEnchantment

Operation	• useEnchantment(type: Enchantment)
Cross References	• Use Cases: Use Enchantment
Preconditions	• The game is in the Play Mode. • Player has the *enchantment* of type *type* in his inventory.
Postconditions	• Player was put under the effect of the *enchantment* of type *type*. • The enchanment *in question* was removed from the inventory.

OC 5: placeStructure

Operation	placeStructure(itemID: itemID, location: Location)
Cross References	Use Cases: Place Structure
Preconditions	• Game is in build mode
Postconditions	• The structure with the itemID was placed in the location.

OC 6: exitGame

Operation	exitGame()
Cross References	Use Cases: Exit the Game
Preconditions	• You are already in the game
Postconditions	• The game was terminated

OC 7: removeStructure

Operation	RemoveStructure(itemID: itemID, location: Location)
Cross References	Use Cases: Remove the Structure
Preconditions	• Game is in build mode • There is a structure in the *location.*
Postconditions	• Structure was removed from location • Location was made available

OC 8: runeCollected

Operation	runeCollected(Tile: ClickedTile)
Cross References	Use Cases: Explore Structure
Preconditions	• The Game is in the Play Mode. • There is a Structure in *location* in question. • Player must be within the one tile of the Structure.
Postconditions	• The rune was revealed if it was in this *structure.*

Supplementary Specifications

Version	Date	Description	Author
Inception Draft	November 20, 2024.	First draft. To be refined during further development phases.	Project Team

Introduction

This document serves as the repository of all requirements for the "*gameName*" game that are not explicitly detailed in the use cases. It outlines essential functionality, usability standards, and non-functional requirements to ensure the development of a robust and engaging game. While the use cases focus on gameplay mechanics, this specification expands on the foundational elements required for a seamless user experience and scalability of the game.

Functionality

Logging and Error Handling

All errors, including invalid grid movements or overlaps, should be logged for debugging purposes. Persistent storage should record these errors for further analysis and future improvements to the game.

Pluggable Rules

The game should support the ability to customize gameplay rules, such as modifying the time limits or adjusting difficulty settings, to enhance replayability and adapt to player preferences. These rules may be defined and executed at various points within gameplay.

The game supports saving and loading, so the players can save their progress while playing the game and load a save file to continue later.

Security

The game ensures that players cannot access the source code to manipulate it.

Usability

Human Factors

The user interface should be clear and easy to navigate for players of all skill levels. Game elements such as the player's inventory, remaining time, current hall, and player's lives must be prominently displayed and easily visible during gameplay.

The use of colors should be mindful of accessibility needs, such as providing a colorblind mode. The game should use alternative visual cues or textual hints for important events (e.g., finding a rune or losing a life) to ensure clarity for all players.

The gameplay experience should be smooth and intuitive, emphasizing strategic thinking and exploration. Audio and visual feedback, such as sound effects when the rune is found or time is running out, should enhance the immersive experience. Pausing, resuming, or exiting the game should be simple and immediately effective.

Performance

The game must maintain a smooth user experience throughout, ensuring that all animations and interactions are seamless. Gameplay should run without delays, even when multiple monsters, objects, or enchantments are active in the hall. Loading times between halls or transitions should be minimal to maintain player immersion.

Glossary

Player: The main character controlled by the player. The player explores the dungeon halls, searching for runes while avoiding monsters and collecting enchantments. The player begins the game with three lives, which can be replenished by collecting specific enchantments. The player is the key figure responsible for unlocking each hall and eventually escaping the dungeon in a limited time.

Rune: A magical artifact hidden within each hall. The player must locate the rune to unlock the exit and proceed to the next hall. Runes may be teleported to new locations by Wizards, adding an extra layer of challenge to the game.

Halls: The dungeon consists of four themed halls—Earth, Air, Water, and Fire. Each hall must be completed in sequence by finding its hidden rune within the given time limit. The design and object placement in each hall differ, designed by the player and also, each hall has their own requirements.

Structure: A type of object in the game that is placed on tiles within the grid. Structures serve as obstacles or strategic elements in the game. They can hide runes or influence the player's navigation, requiring players to plan their movements and actions around their placement.

Tile: The basic unit of the grid-based map in the game. Each Tile can host a structure, the player, a monster, or an enchantment. The player can move only on empty tiles. Tiles serve as the foundation for strategic movement and structure placement in the game.

Monsters: These are enemies that obstruct the player's progress and create challenges. There are three types of monsters:

1. **Archer Monster**: Shoots arrows every second if the player comes within four squares. The player can avoid detection by using the Cloak of Protection enchantment.
2. **Fighter Monster**: Attacks the player with a dagger when adjacent. It can be distracted using the Luring Gem enchantment.
3. **Wizard Monster**: Does not attack the player but teleports the rune to a random location every 5 seconds, making it harder to locate. **Depending on the remaining time, the wizard monster will act differently, making the game easier as the remaining time runs out.**
4. **Timer Monster:** Decreases the player's remaining time by 5 seconds when nearby. It can be distracted using the Luring Gem enchantment.

Enchantments: Special items that assist the player in various ways. They appear randomly in the halls and provide advantages like extra time, extra lives, or strategic abilities. Some enchantments are used immediately upon collection, while others can be stored in the player's inventory for later use.

Inventory: The player's inventory for storing collected enchantments. Stored enchantments can be used strategically during the game to overcome specific obstacles or challenges.

Grid: The dungeon map is divided into a grid of tile. Each tile can contain one object, such as the player, a monster, a wall, or an enchantment. The grid structure ensures organized and systematic movement for all game elements.

Timer: A countdown mechanism that sets a time limit for each hall. If the timer runs out before the rune is found, the game ends. Collecting the Extra Time enchantment can add seconds to the remaining time.

Build Mode: The phase where players design the interiors of the dungeon halls before starting the game. Players must place a minimum number of objects in each hall, creating opportunities for rune hiding and strategic gameplay.

Play Mode: The active phase of the game where the player explores the halls, avoids monsters, collects enchantments, and searches for the rune. The player's progress, remaining lives, and timer are displayed during this mode.

Luring Gem: A specific enchantment used to distract Fighter monsters. The player throws the gem in a particular direction, causing nearby Fighter monsters to follow it, allowing the player to move away safely.

Cloak of Protection: An enchantment that hides the player from the Archer monster's vision, enabling close proximity without losing lives. The effect lasts for a limited duration.

Speed Up: This enchantment can be stored in the inventory and, when activated, doubles the player's speed.

Extra Life: An enchantment that immediately adds one life to the player upon collection. This helps the player continue their journey if lives are lost during encounters with monsters.

Extra Time: An enchantment that adds additional seconds to the timer when collected. This effect is applied immediately upon collection, giving the player more time to search for the rune and complete the current hall.

Reveal Enchantment: A helpful enchantment that highlights a 4x4 area on the grid where the rune is hidden. The player can use it to narrow down the search area strategically.

Pause/Resume Button: A feature in the game interface that allows the player to pause the game at any time during play mode. The game can be resumed from the same state.

Main Menu: The initial screen that appears when the game is launched. It provides options to start a new game, access the help screen, or exit the game.

Help Page: A section of the game's interface that provides players with detailed information about gameplay mechanics, controls, objectives, and descriptions of game elements such as monsters, enchantments, and runes. It is designed to guide new players and assist them in understanding how to play the game effectively.

Extra Features

Random Map Initilization

In the build mode, a dice icon has been incorporated, allowing players to initialize the hall with randomly placed structures of various types. This feature streamlines the process by enabling players to save time and effort, such as avoiding the need to place 17 structures individually. To use the feature, the player must click on the dice icon and then select a specific grid, after which the required number of structures will be randomly placed within the designated area. Since the player remains in build mode after using the dice, they retain the flexibility to edit or reposition the randomly placed structures as desired.

New Enchantment: Speed Up!

This enchantment can be stored in the inventory and, when activated, doubles the player's speed. This adds greater excitement and strategic depth to the game.

Custom Pixel Arts

We have designed a collection of pixel art specifically for our game:

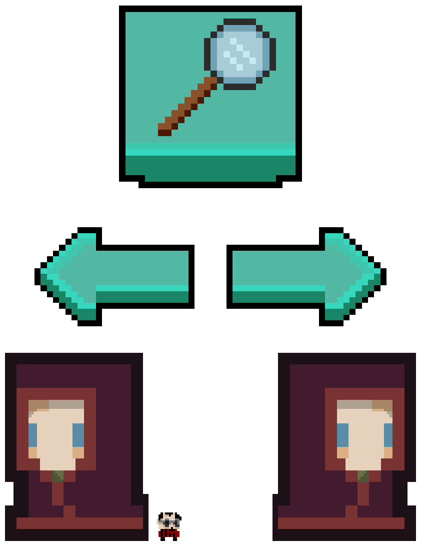


Figure 13: Extra Features

Sound Effects

We incorporated music and various sound effects into our game, some of which we created ourselves. For instance, "archer.wav" plays when the archer shoots an arrow, while "structureSounds" is triggered when the player clicks on a structure without a rune.

Try Again Button

If the player wins the game, a victory screen is displayed, whereas a game over screen appears if the player loses for any reason. Both screens include a "Try Again" button, which allows the player to restart the game quickly with the same hall setups upon being clicked.

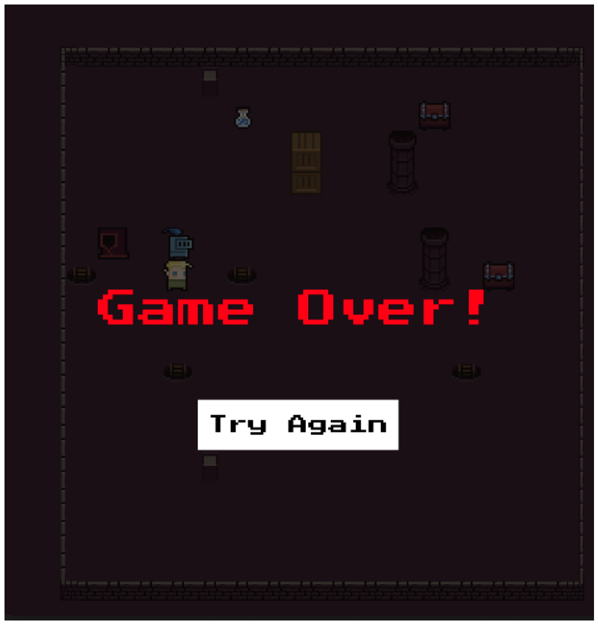
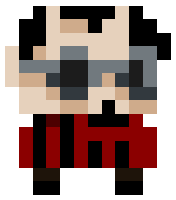


Figure 14: Extra Features

New Monster: Timer Monster

This monster steals the user's time. When in contact with the player, the player loses 5 seconds from the remaining time.



DESIGN

Logical Architecture

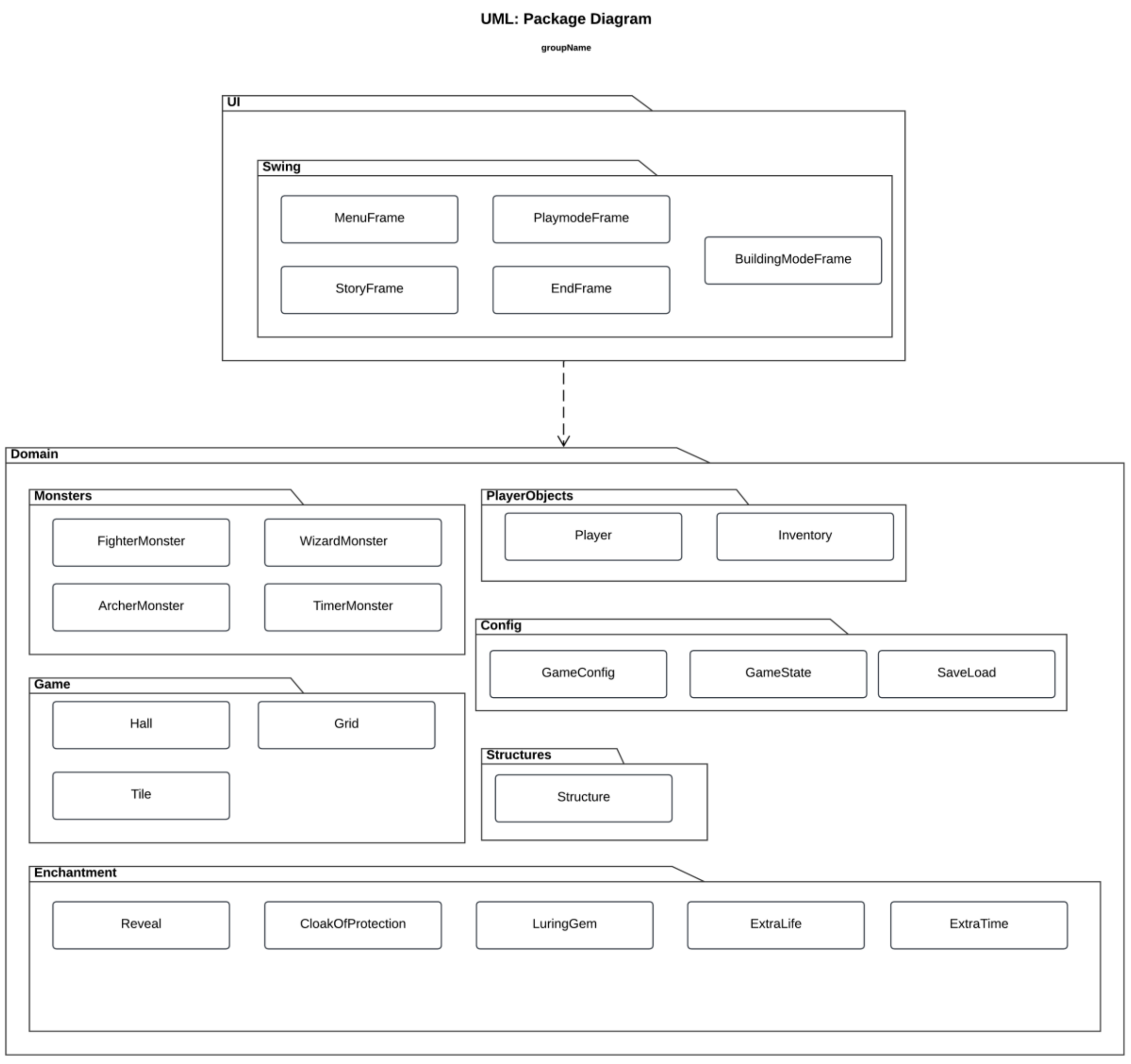


Figure 14: Package Diagram

Sequence Diagrams

SD 1: Start BuildMode

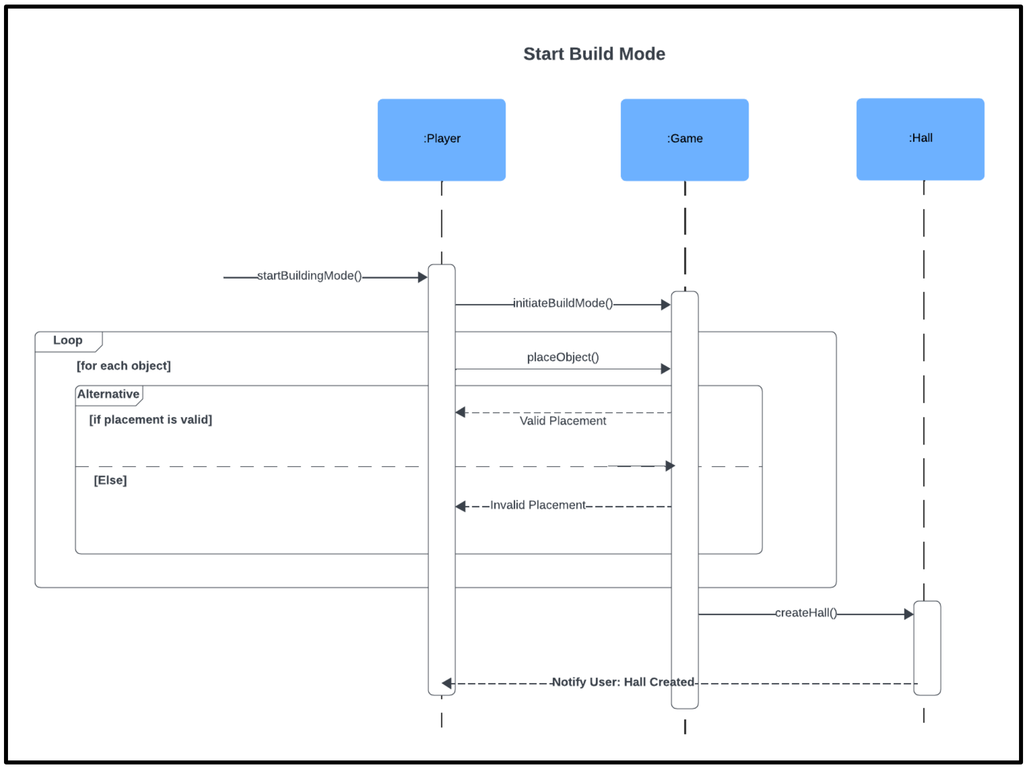


Figure 15: SD 1 – Start BuildMode

SD 2: Move Player

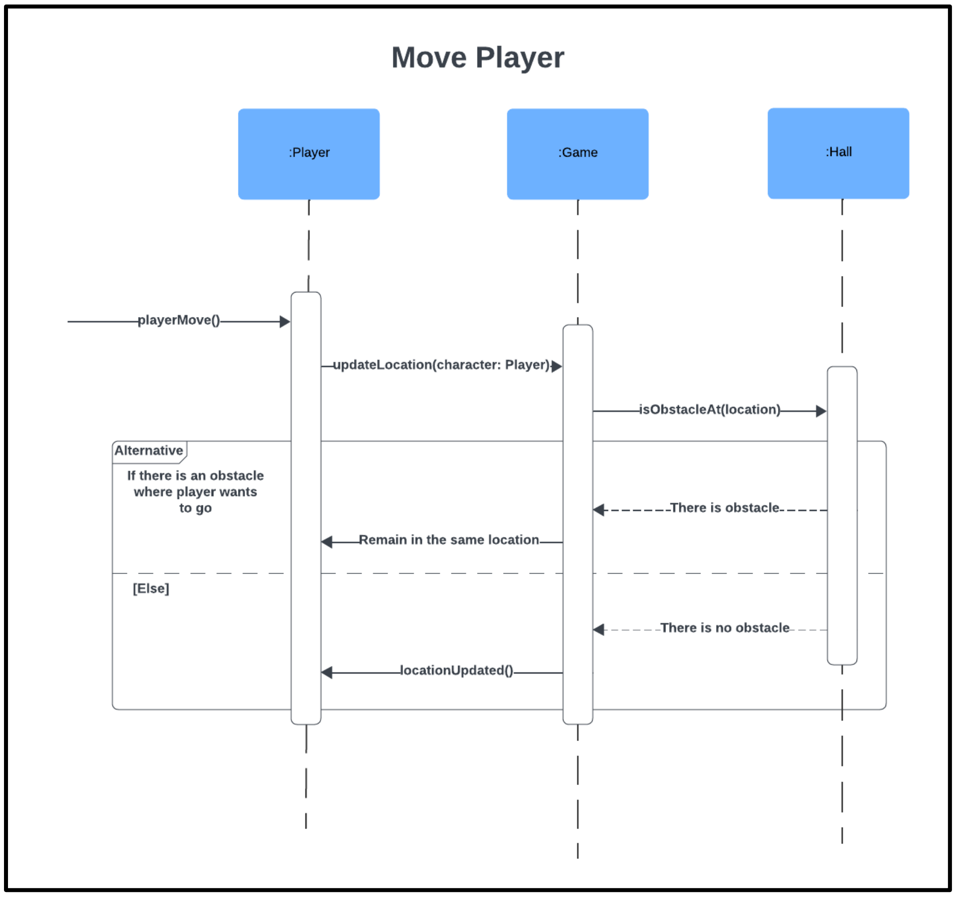


Figure 16: SD 2 - Move Player

SD 3: Explore Structure

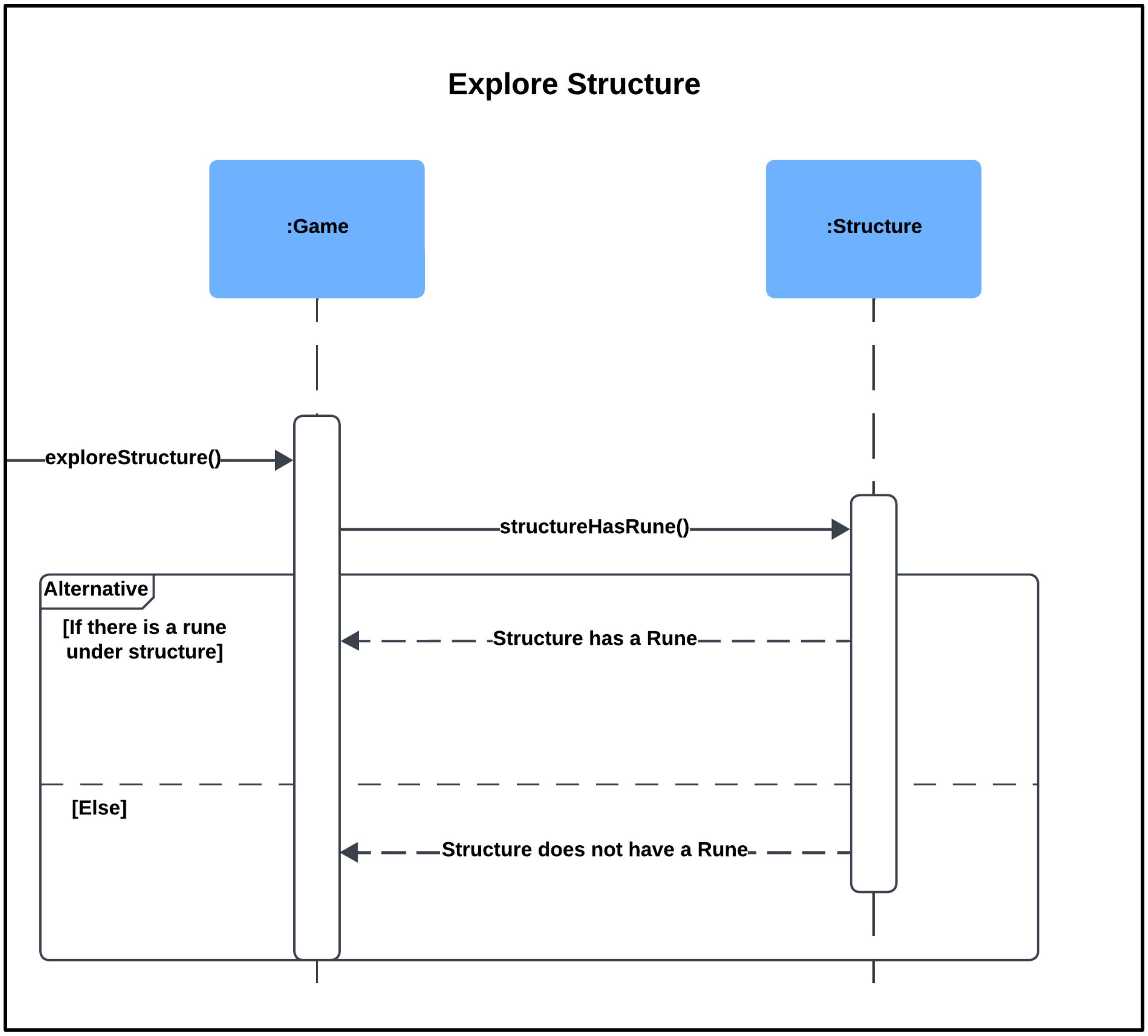


Figure 17: SD 3 - Expolore Structure

SD 4: Start PlayMode

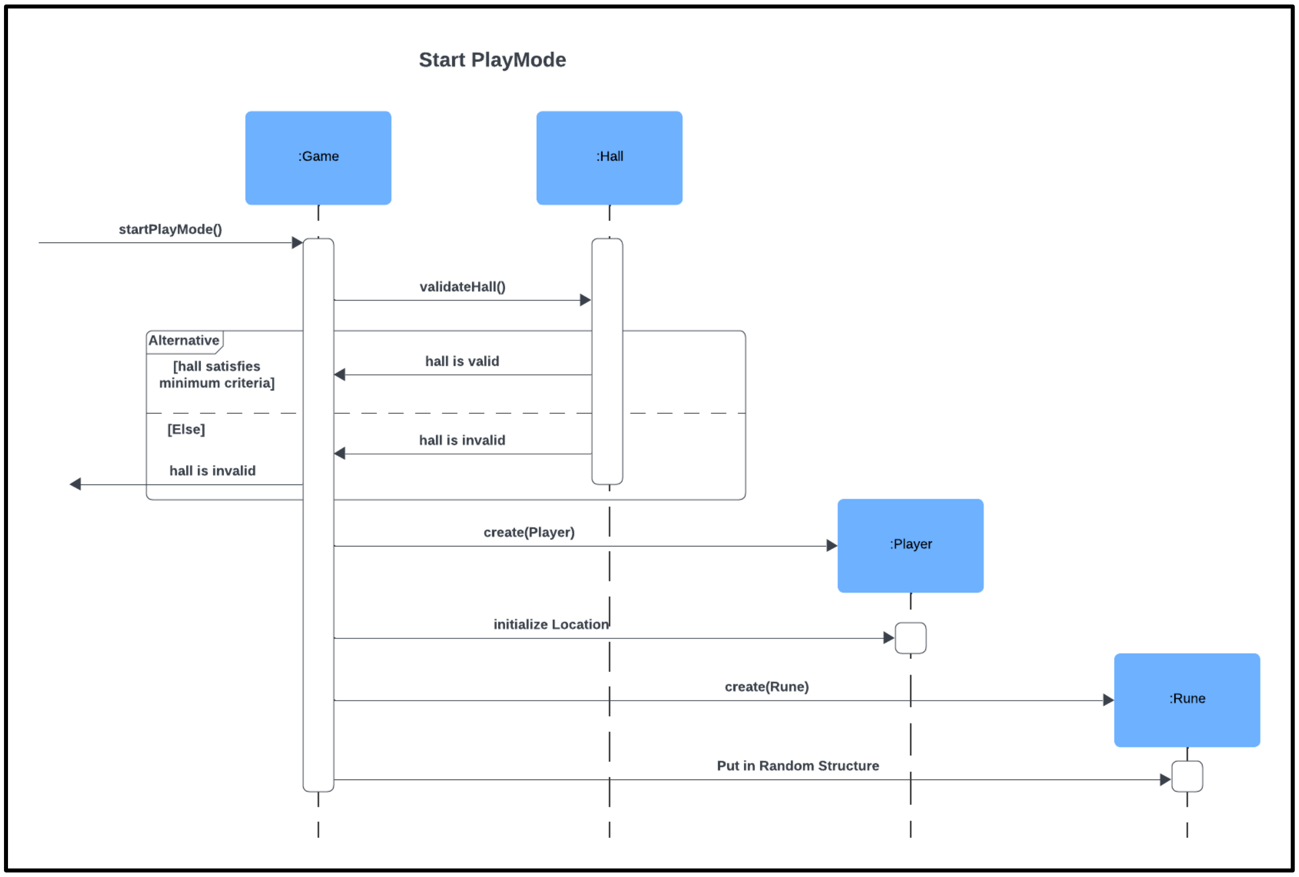


Figure 18: SD 4 - Start PlayMode

SD 5: Use Luring Gem

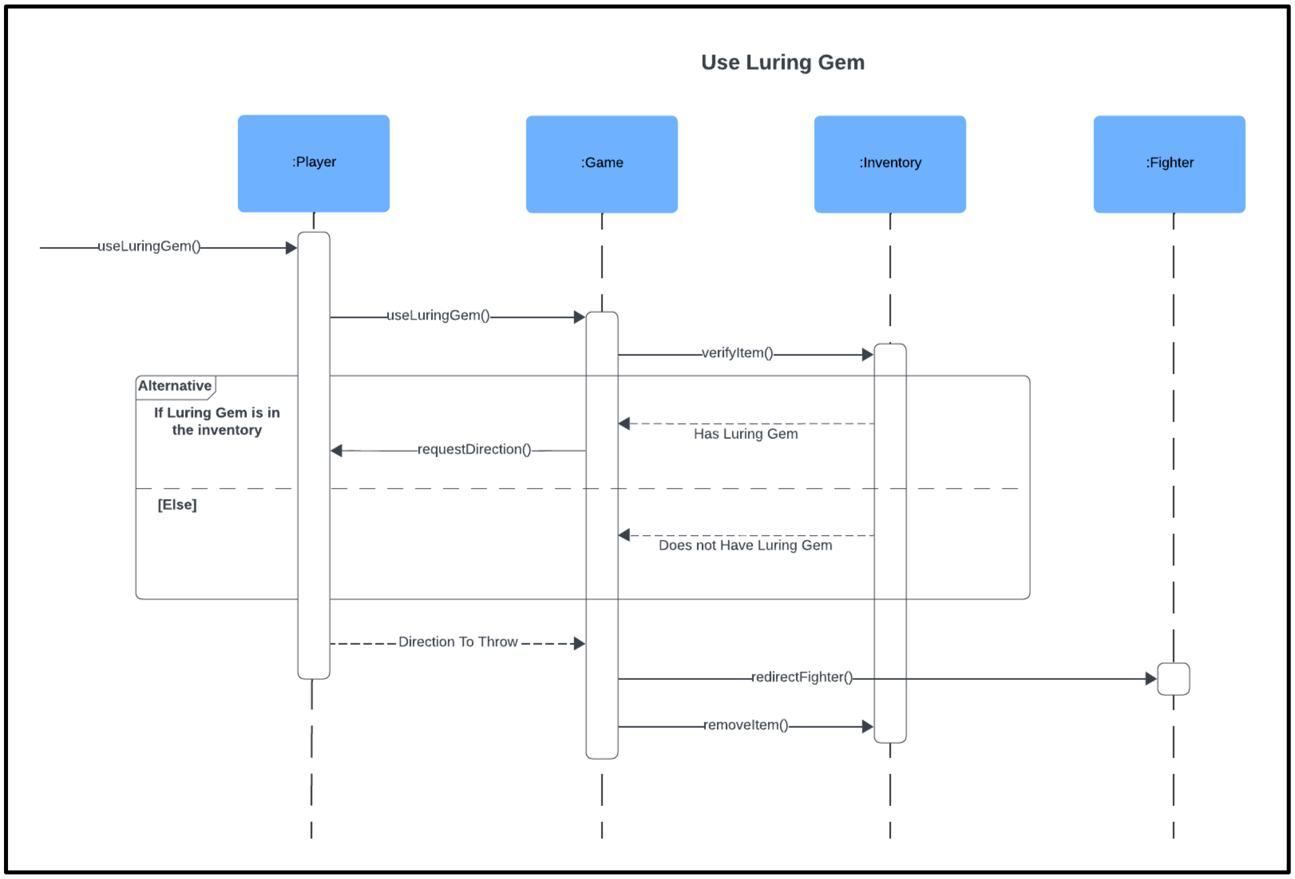


Figure 19: SD 5 - Use Luring Gem

SD 6: Use Cloak of Protection

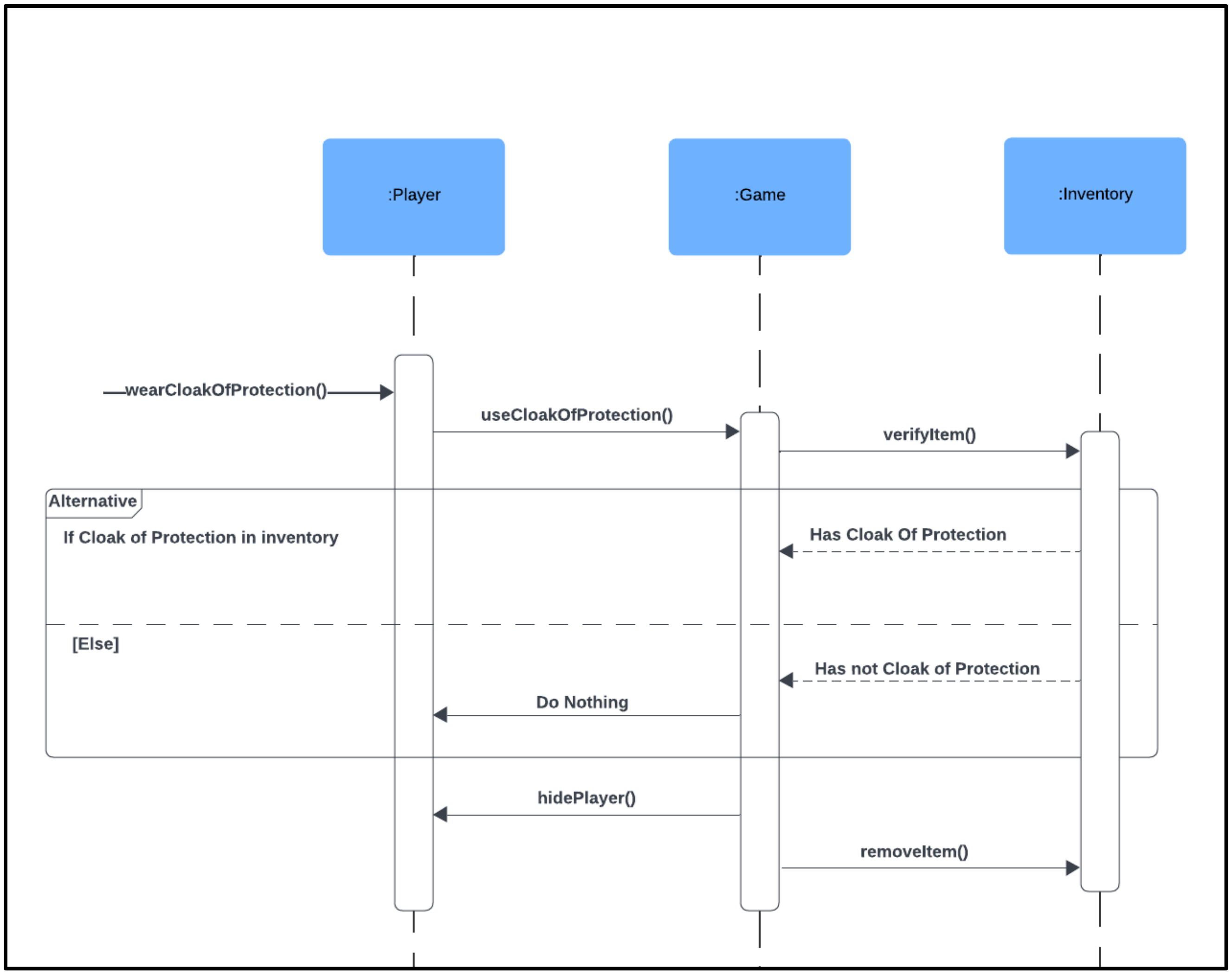


Figure 20: SD 6 - Use Cloak of Protection

SD 7: Extra Life

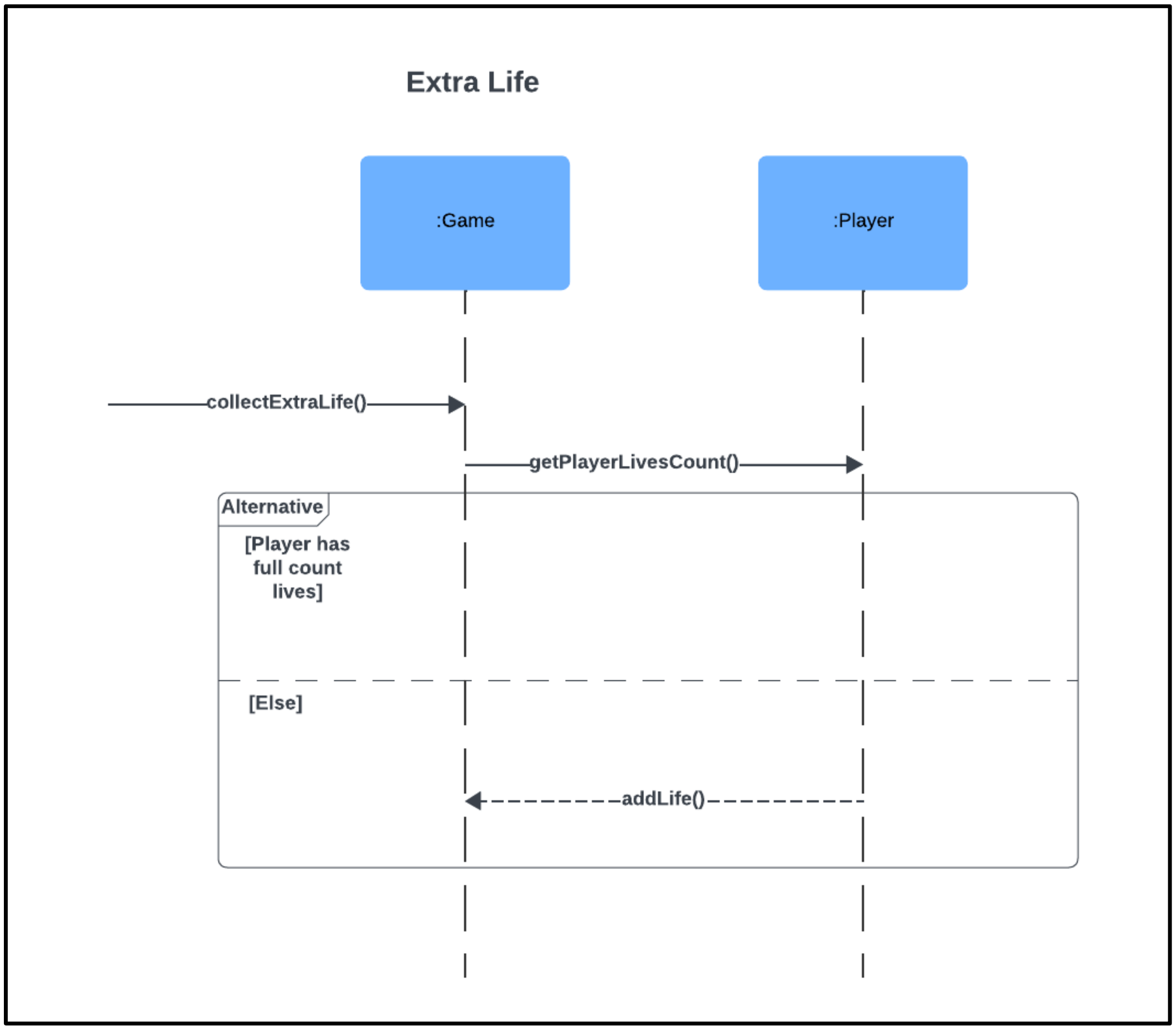


Figure 21: SD 7 - Extra Life

SD 8: Use Reveal

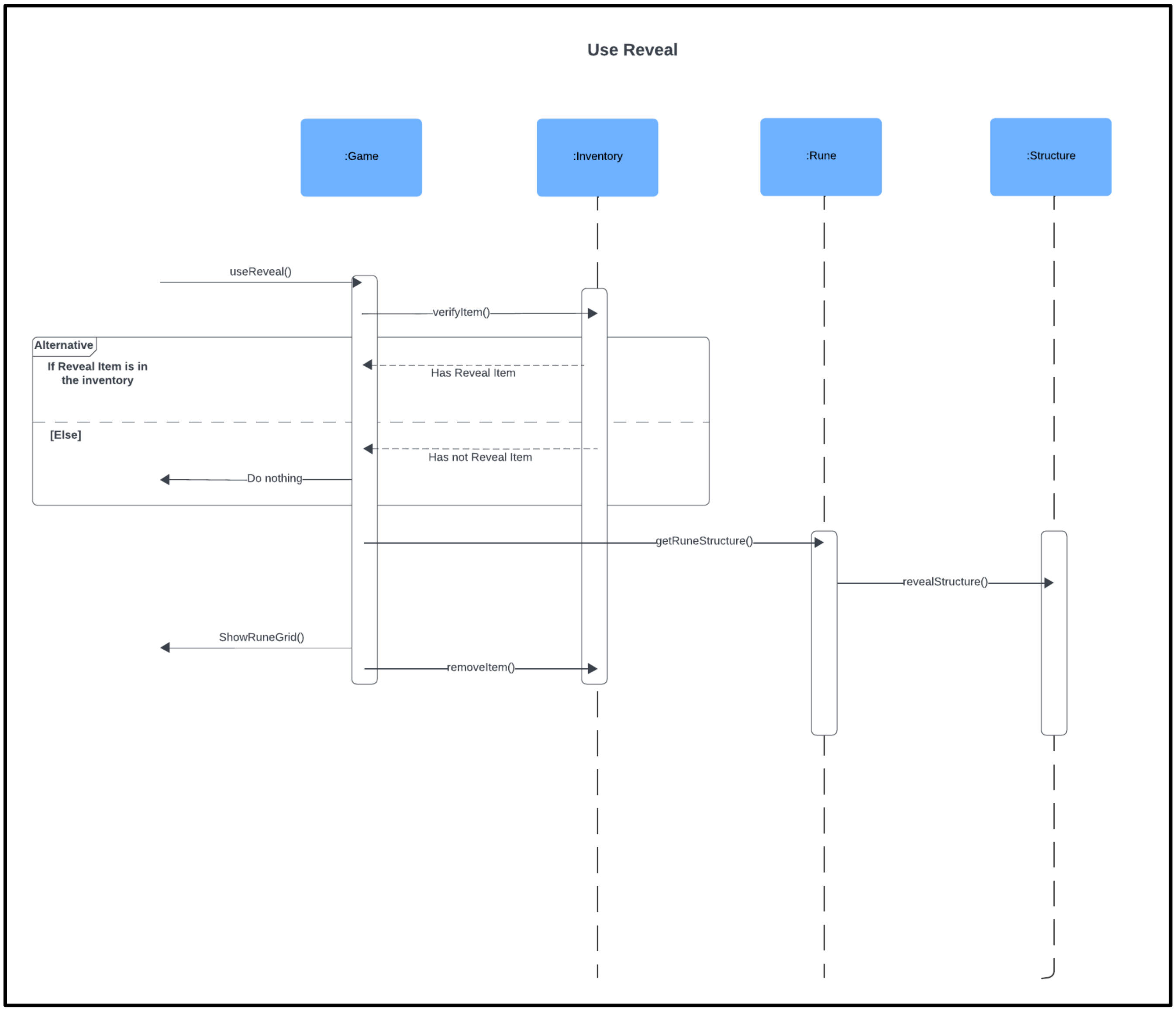


Figure 22: SD 8 - Use Reveal

SD 9: Archer vs Player

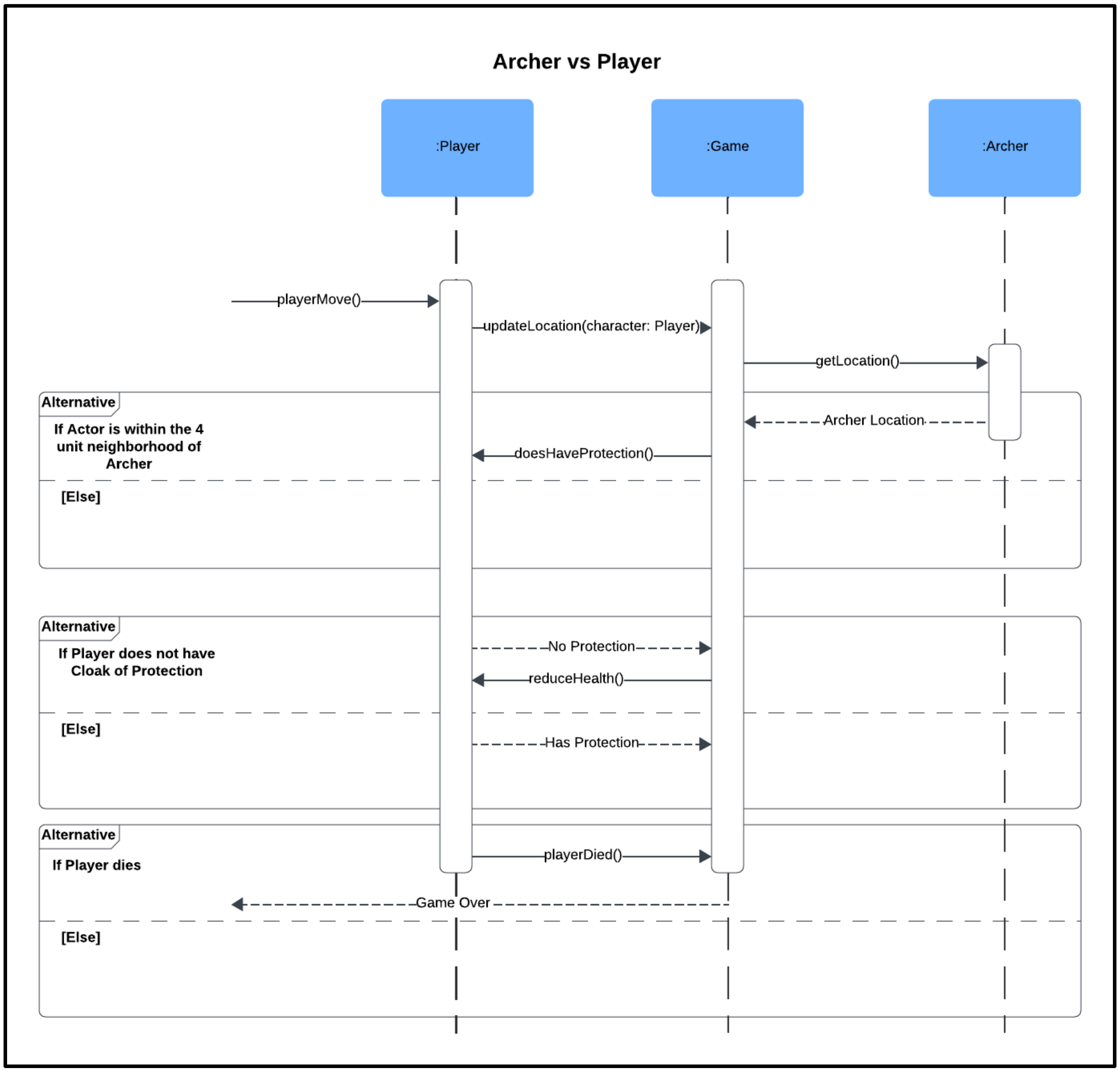


Figure 23: SD 9 – Archer vs Player

SD 10: Find Rune

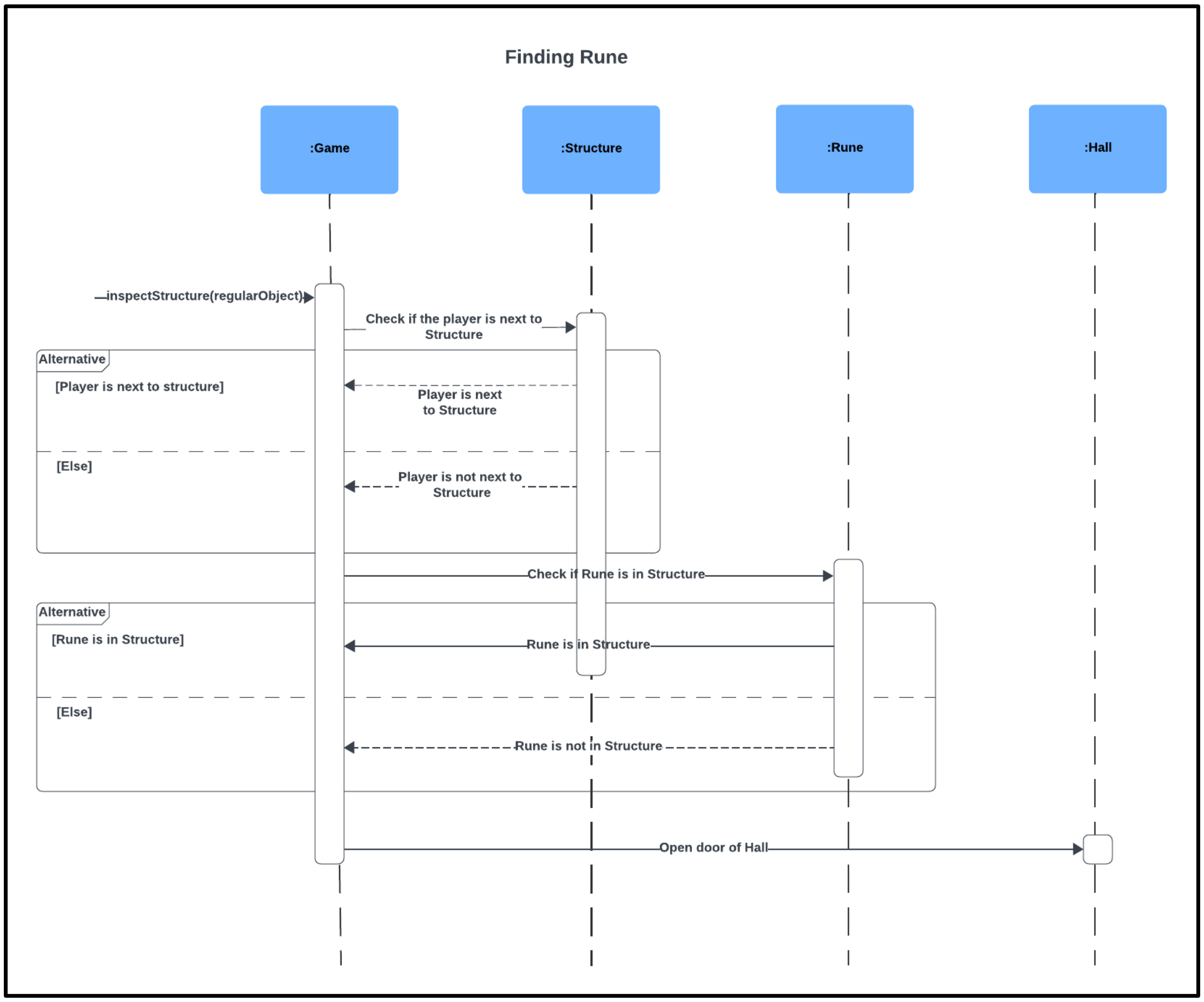

Finding Rune
:Game
:Structure
:Rune
:Hall
inspectStructure(regularObject)
Check if the player is next to Structure
Alternative
[Player is next to structure]
Player is next to Structure
[Else]
Player is not next to Structure
Check if Rune is in Structure
Alternative
[Rune is in Structure]
Rune is in Structure
[Else]
Rune is not in Structure
Open door of Hall

Figure 24: SD 10 – Find Rune

SD 11: Wizard Behavior

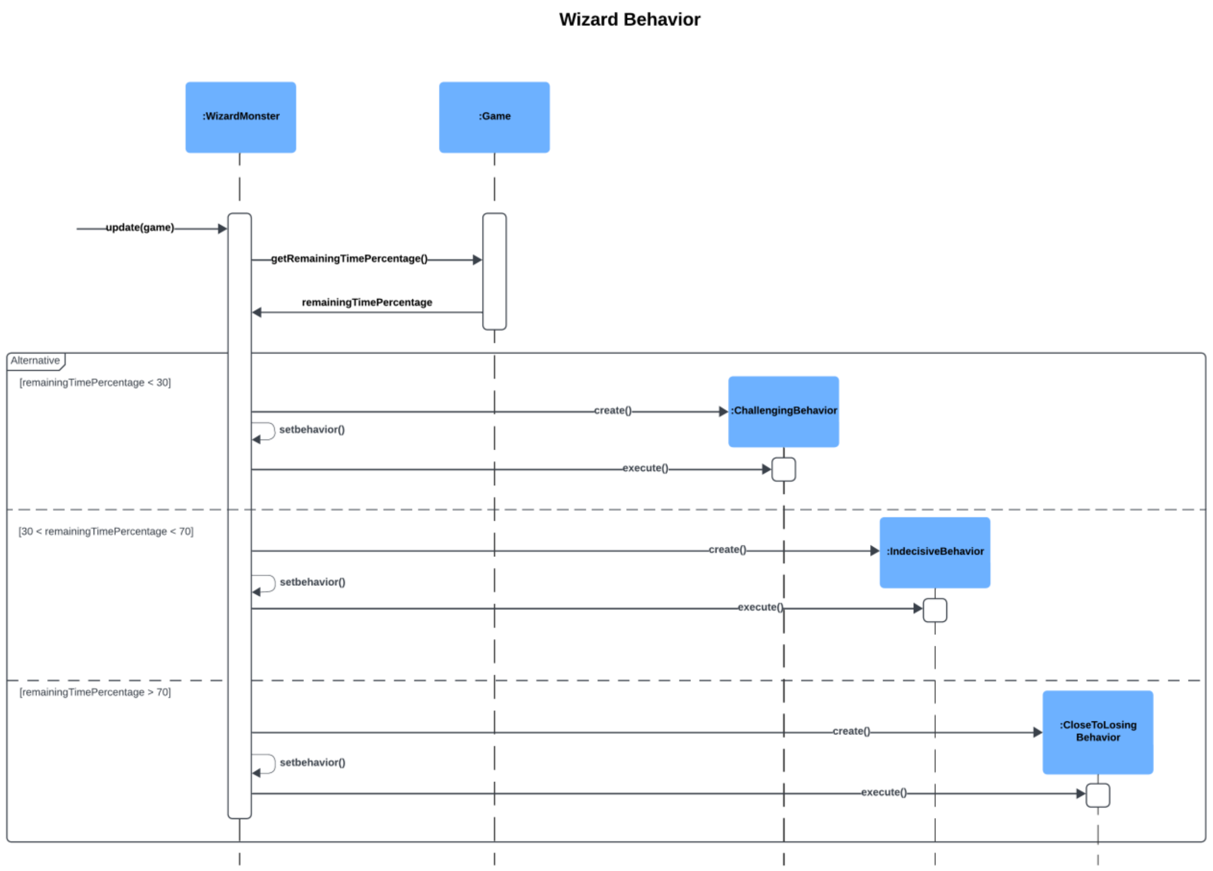


Figure 25: SD 11 – Wizard Behavior

SD 2: Wizard Disappear

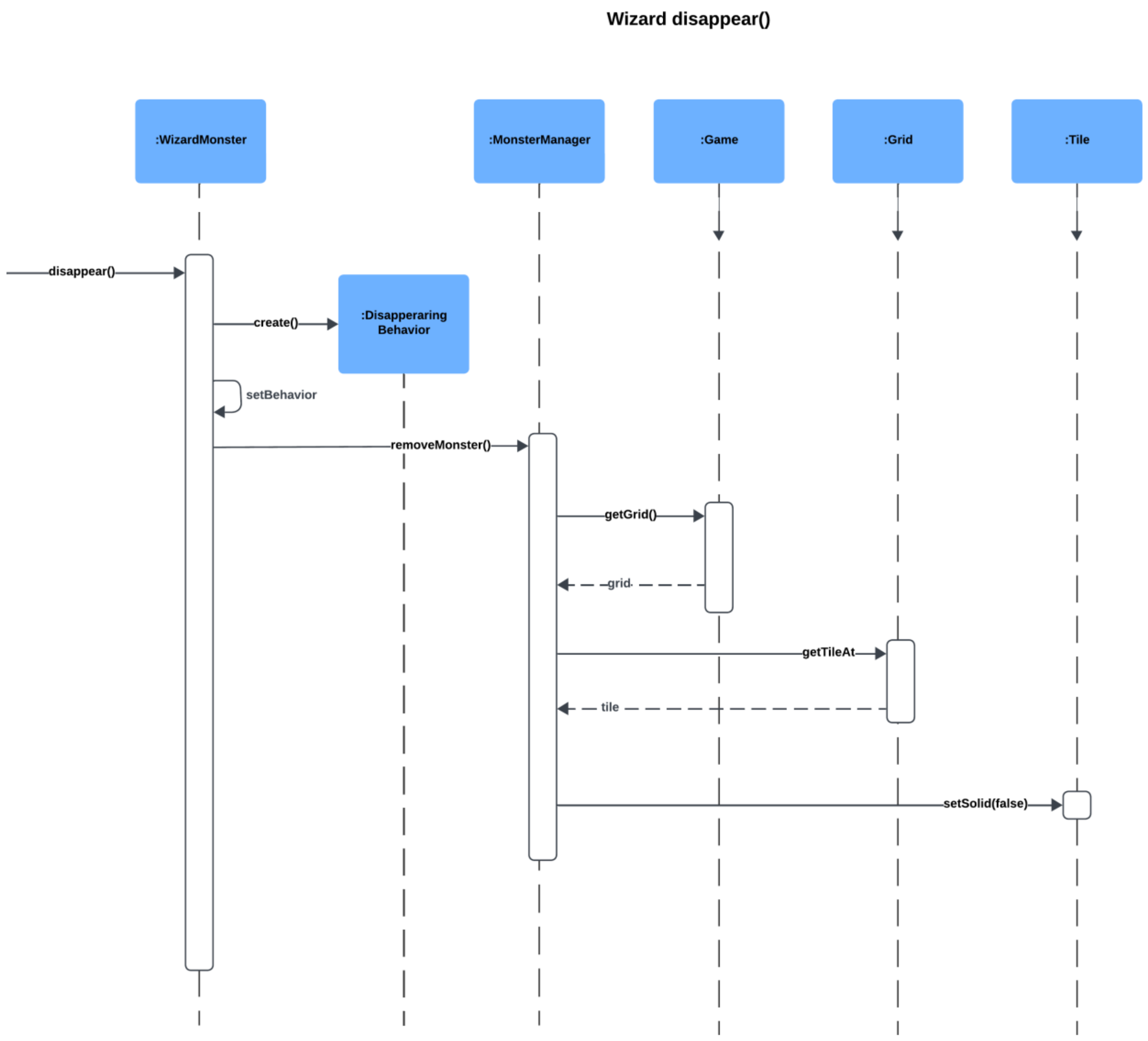


Figure 126: SD 12 – Wizard Disappear

Communication Diagrams

CD 1: Start BuildMode

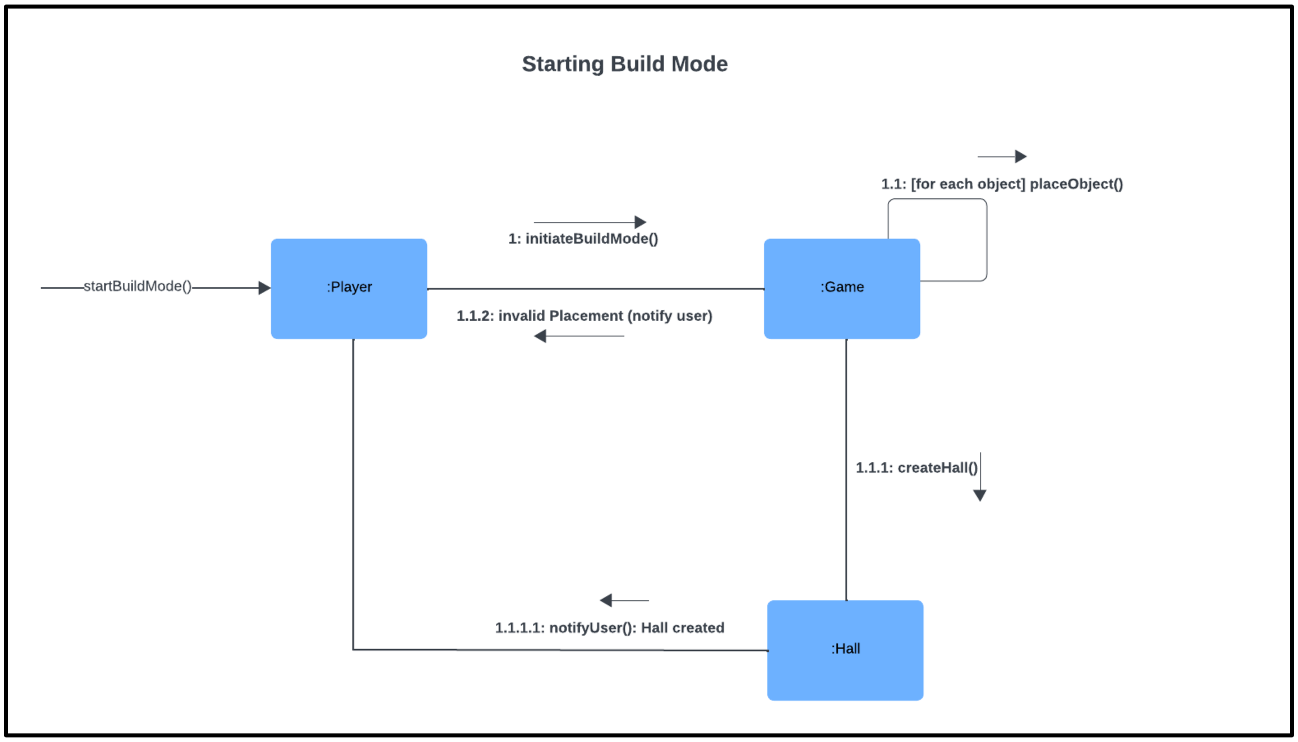

Starting Build Mode
1.1: [for each object] placeObject()
1: initiateBuildMode()
startBuildMode()
:Player
:Game
1.1.2: invalid Placement (notify user)
1.1.1: createHall()
1.1.1.1: notifyUser(): Hall created
:Hall

Figure 27: CD 1 – Start BuildMode

CD 2: Move Player

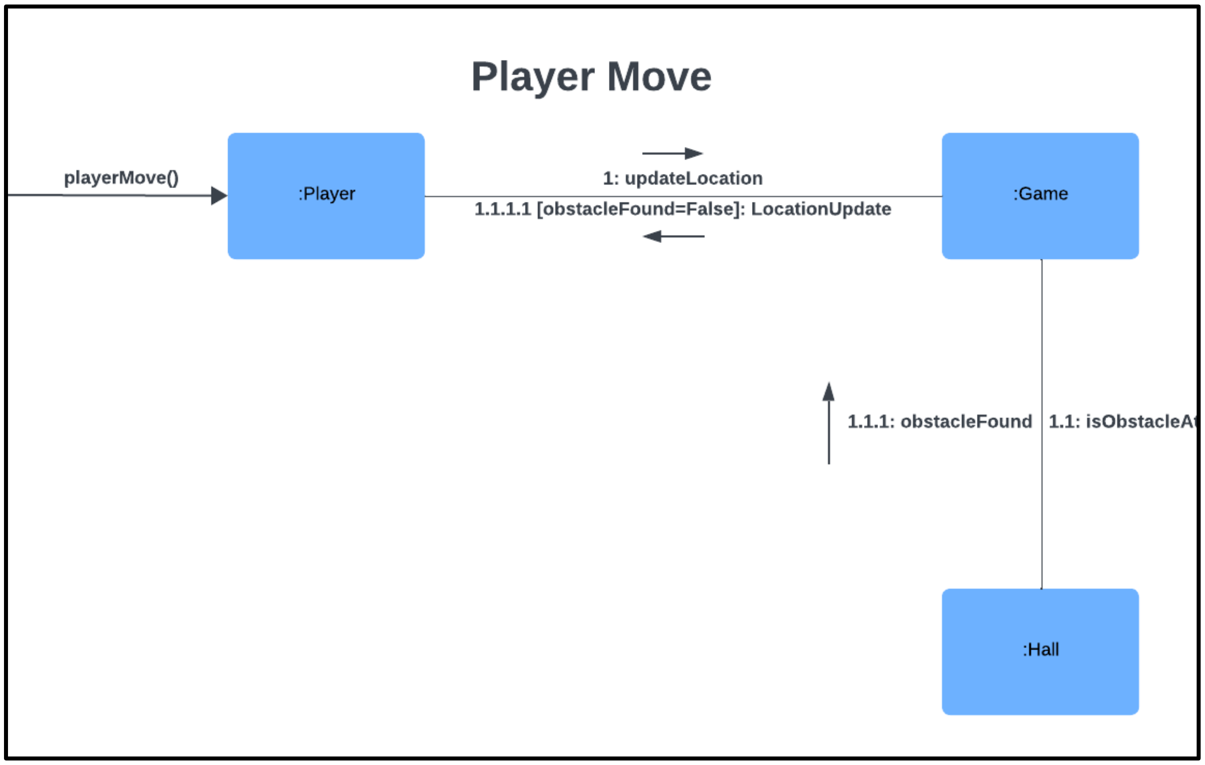


Figure 28: SD 2 – Move Player

CD 3: Explore Structure

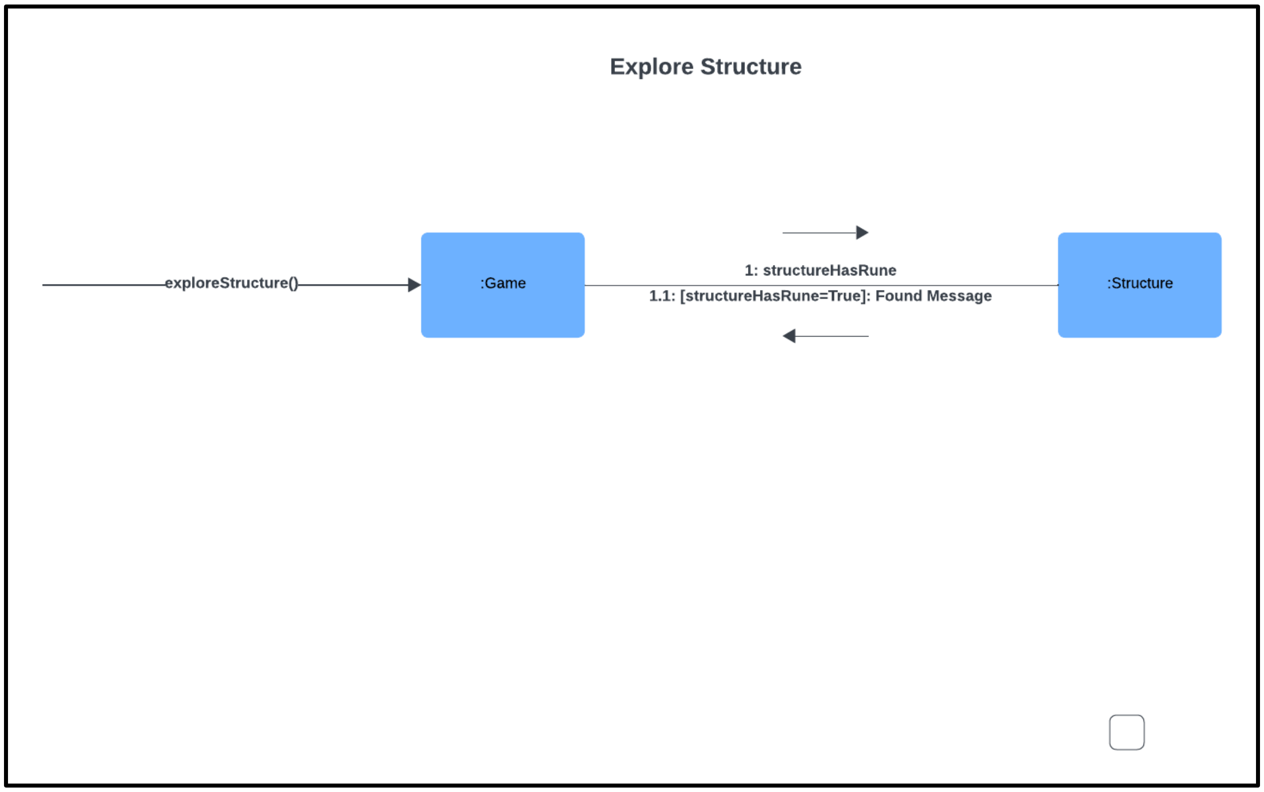


Figure 29: CD 3 - Explore Structure

CD 4: Start PlayMode

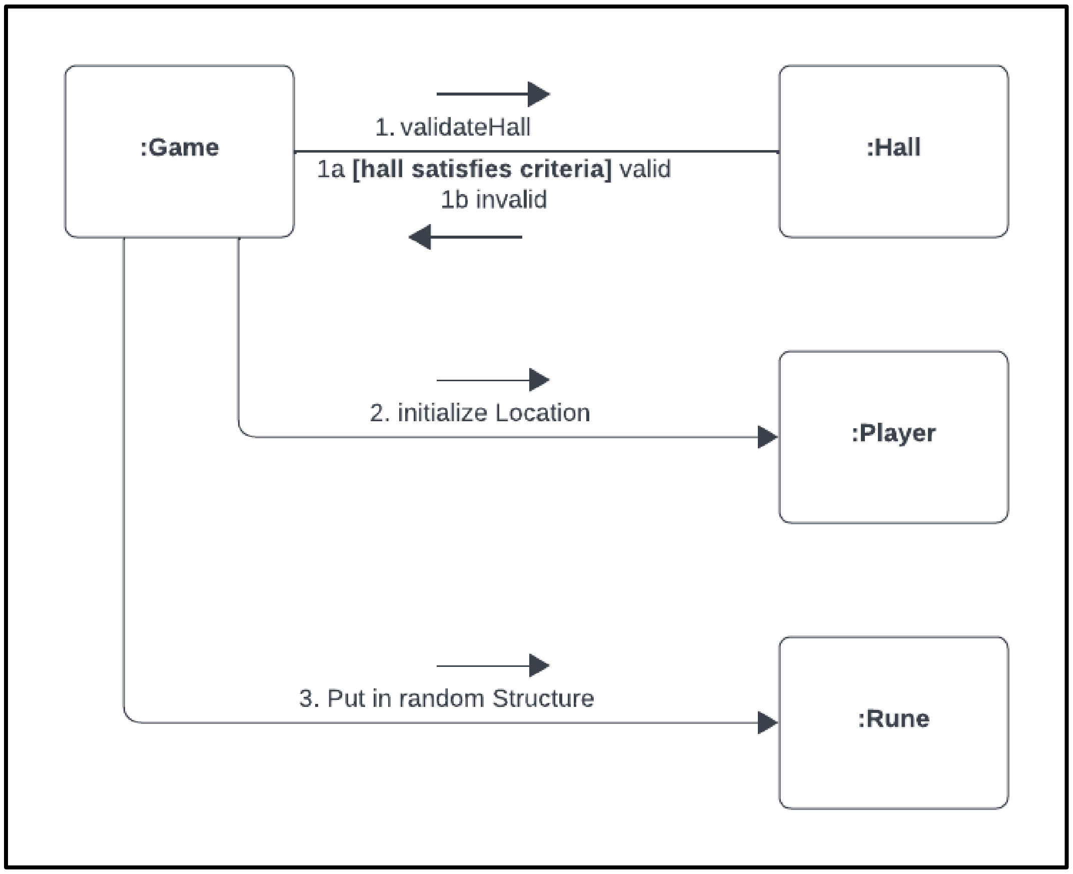


Figure 30: CD 4 - Start PlayMode

CD 5: Use Luring Gem

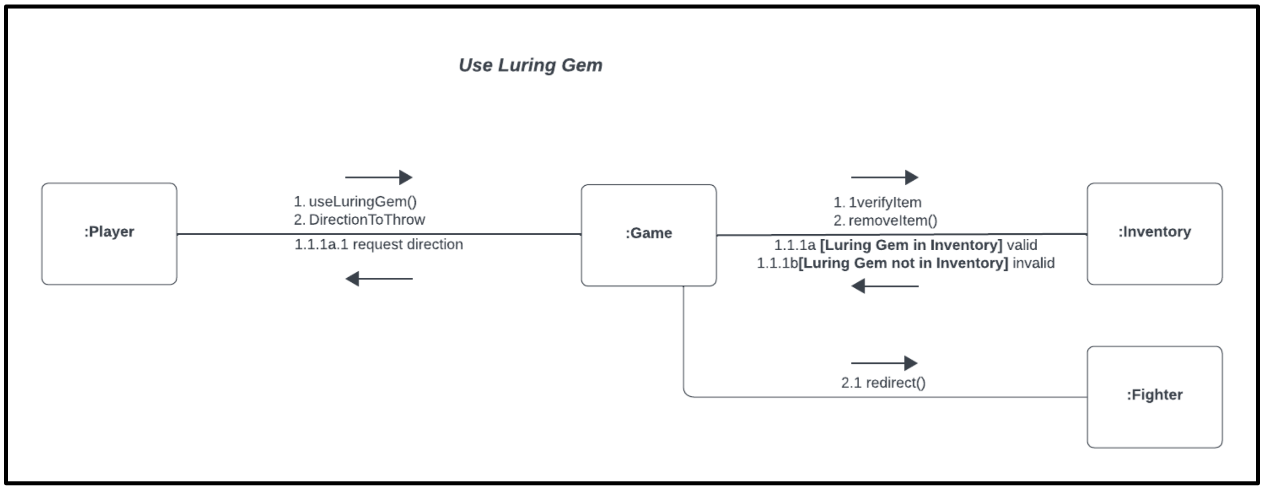


Figure 31: CD 5 - Use Luring Gem

CD 6: Use Cloak of Protection

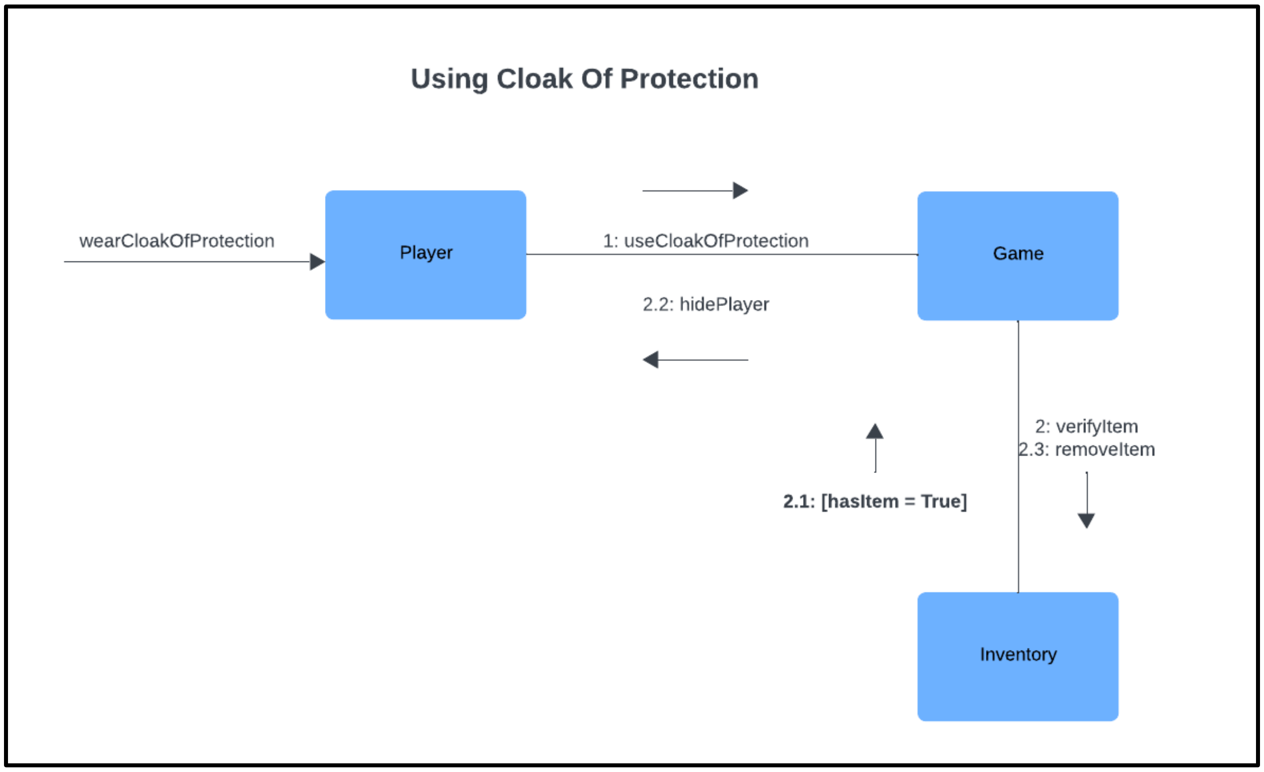


Figure 32: CD 6 - Use Cloak of Protection

CD 7: Extra Life

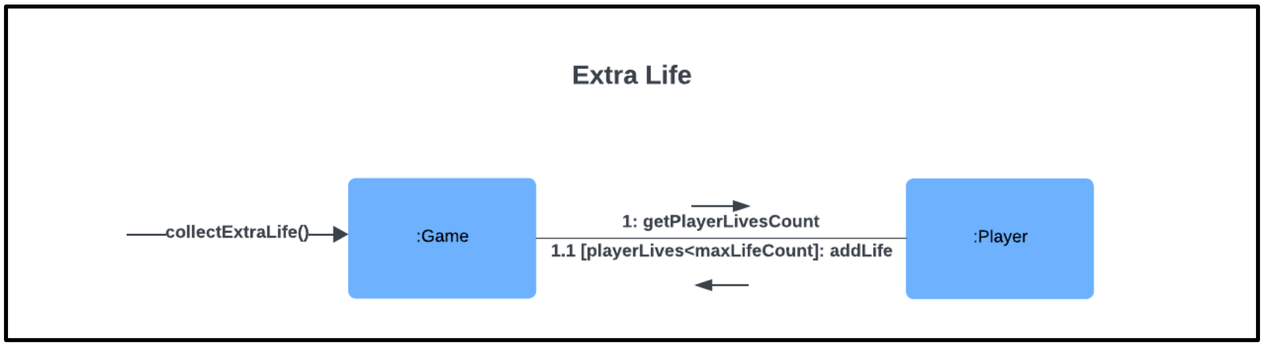


Figure 33: CD 7 - Extra Life

CD 8: Use Reveal

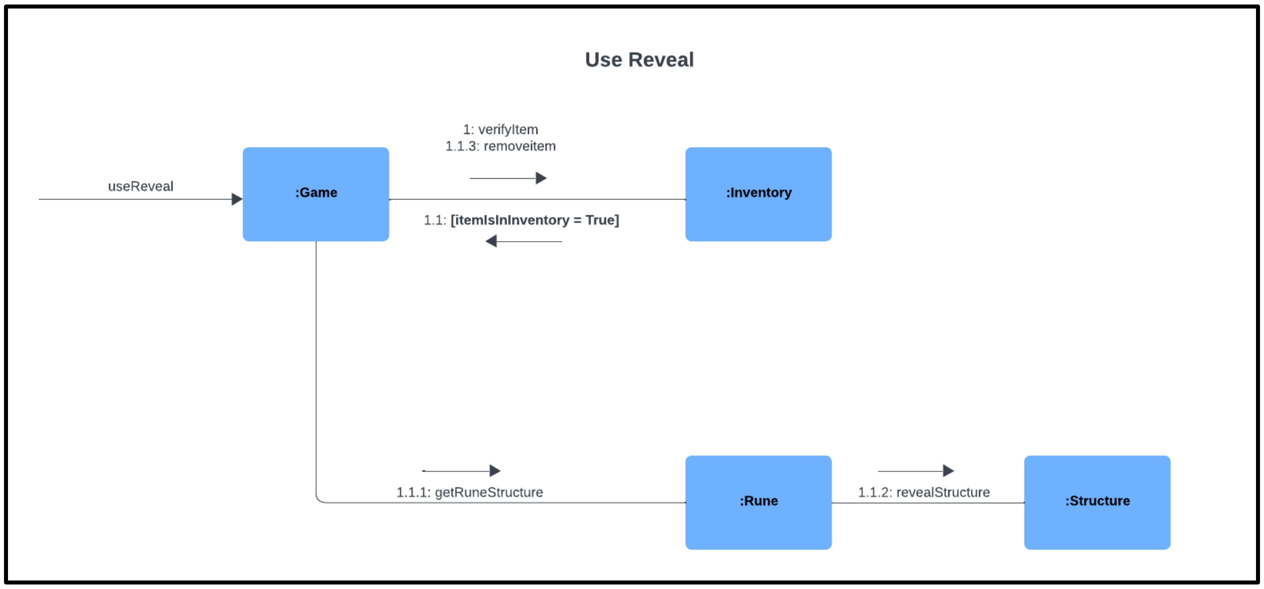


Figure 34: CD 8 - Use Reveal

CD 9: Archer vs Player

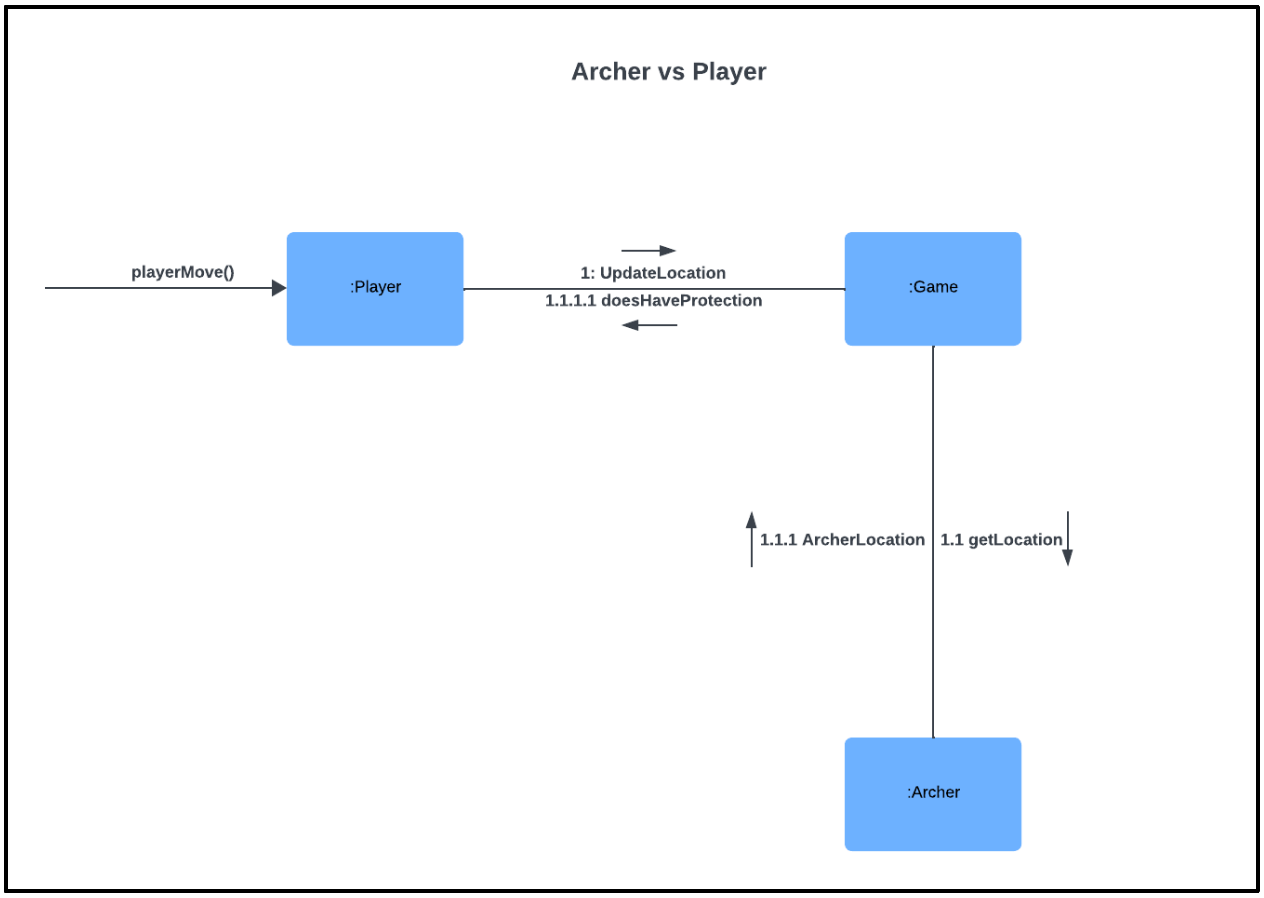


Figure 35: CD 9 – Archer vs Player

CD 10: Find Rune

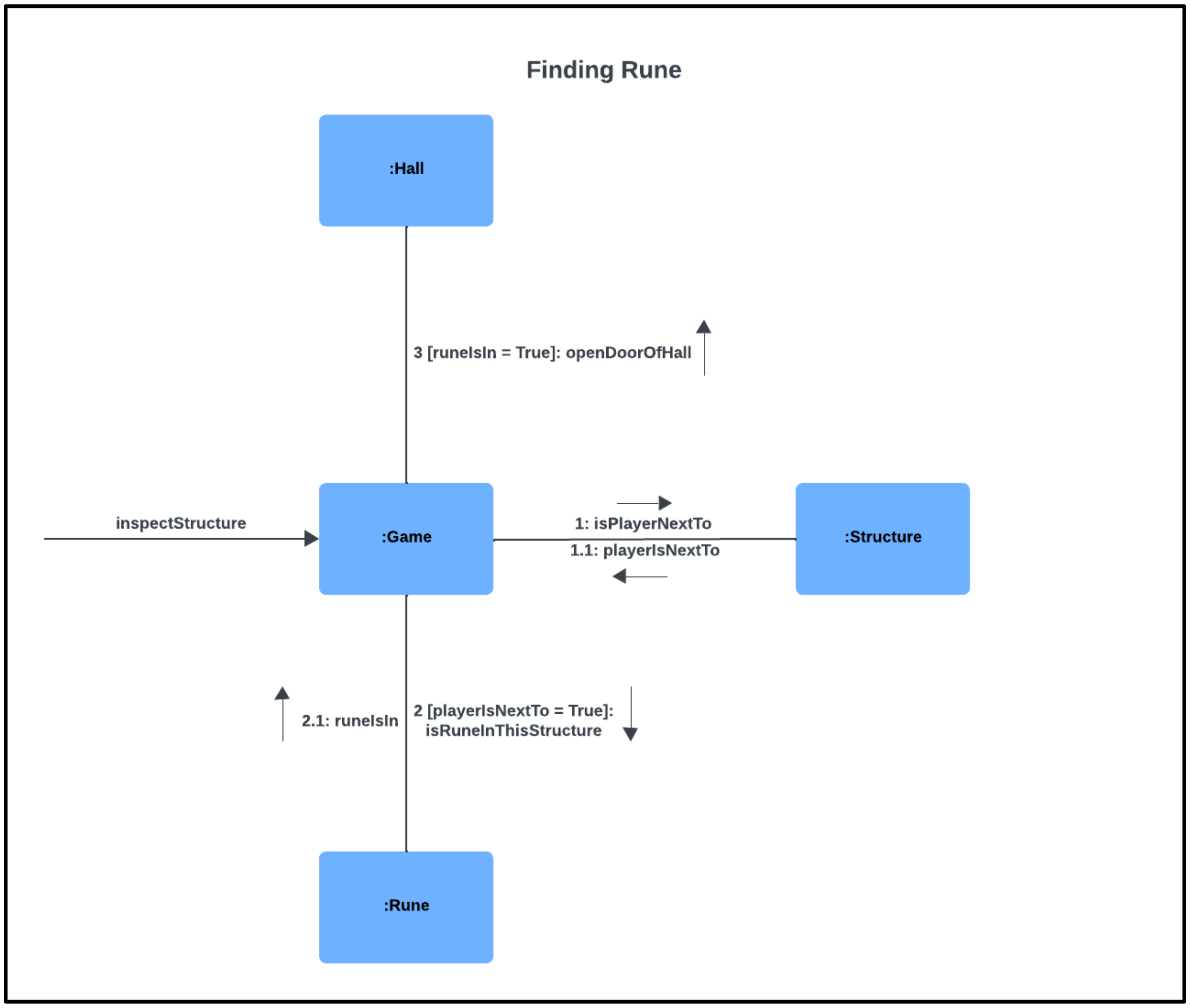


Figure 36: CD 10 – Find Rune

Class Diagram

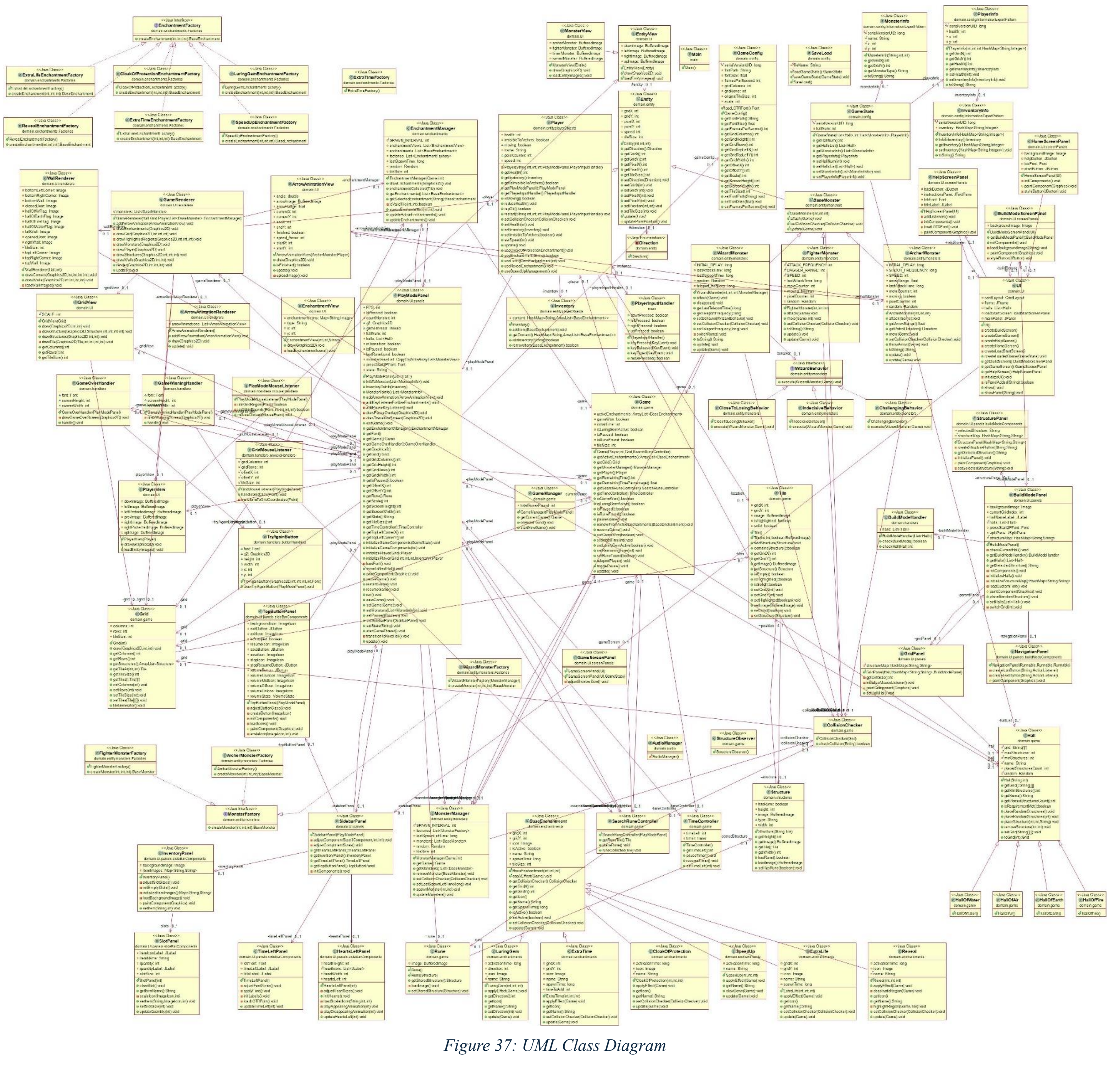


Figure 37: UML Class Diagram

Design Alternatives

- **Controller Pattern:** In our design, we used the controller pattern to design the `Game` class. The UI elements delegate the inputs to the classes at the domain level. The `Game` class controls the flow of the game, delegates the work of performing checks to the appropriate classes and then controls the game accordingly by calling certain methods of the classes. As a drawback, using the `Game` class to control too many things may make it bloated so we may add other controller classes to our design.

- **Creator Pattern:** We will use the `Game` class to create the halls and the player. Because the `Game` class closely utilizes the halls and the Player object, it is responsible for the creation of these classes. The `Game` class also handles the creation of `Monster` and `Enchantment` objects since it holds the initializing data for them such as the spawning frequency.
  Another class in which we applied the creator pattern is the `Hall` class. Each `Hall` will create its own grid based on its size. Since every `Grid` is contained in a `Hall`, the `Hall` has the responsibility of creating the Grid according to the creator pattern.

- **Strategy Pattern:** We used the Strategy Pattern to implement the Wizard monster's dynamically changing behavior. Different behavior classes (ChallengingBehavior, IndecisiveBehavior, …) implement the IWizardBehavior interface, and the Wizard monster uses this interface to change its behavior according to the time left.

- **Information Expert Pattern:** For collecting and using enchantments, we will use the Inventory class as an Information Expert. As the information expert, the Inventory class holds the number of each item in its inventory and other classes ask the Inventory for item counts and usability.

- **Low Coupling:** The `Game` class does not shoot the arrows of the Archer monster directly, instead it triggers the Archer monster's `shootArrow()` method. Therefore the changes in the arrow shooting behavior do not directly affect the `Game` class. If we had the `Game` class be responsible for shooting the arrow, it would result in high coupling.

- **Factory:** We used the Factory pattern to create Monsters and Enchantments. This way we can create similar types of objects efficiently.