Kafes Game

Group Name: **KAFES-A**

Group Members:

- Abdullah Tunçer
- Ali Han Kılınç
- Engin Sühan Bilgiç
- Faruk Aksoy
- Kayrahan Yüce
- Sermet Arda Topal

Contents

1 Use Case Descriptions 1

2 System Sequence Diagrams 11

3 Sequence Diagrams 13

4 Operation Contracts 15

5 Revised Domain Model Diagram 17

6 Revised Class Diagram 18

January 11, 2025

1 Use Case Descriptions

Use Case ID	UC11
Use Case Name	Save Build Mode Design
Scope	Build Mode
Level	Subfunctional
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to save the current hall design to revisit or edit it later. • **Game System:** – Must accurately save the hall design, including objects and their positions, in JSON format. – Ensure the saved file is loadable and error-free. • **Developer:** Must ensure data integrity and prevent issues like file corruption.
Preconditions	• The player is in Build Mode. • At least one object is placed in the hall. • There is sufficient storage space to create or overwrite the save file.
Postconditions	• The current hall design, including object types, positions, and the hall type, is saved in a JSON file. • The system displays a success message confirming the save. • The saved design can be loaded and edited later.

Main Success Scenario	1. The player selects the ’Save Design’ option while in Build Mode. 2. The system collects the current hall’s state: 2.1. Object types (e.g., ’box’, ’chest’). 2.2. Object positions (x, y coordinates). 2.3. Hall type (e.g., Earth, Air, Water, Fire). 3. The system converts this data into JSON format. 4. The system writes the JSON data to a file in the designated save directory. 5. The system displays a success message: ’Design saved successfully!’.
Extensions	2a. **Insufficient Storage Space:** 0.1. The system detects insufficient storage space on the device. 0.2. The system displays the following error message: ”Save failed: Not enough storage space available.” 0.3. The system does not proceed with saving and leaves the current hall design unchanged. 2b. **File Write Error (I/O Exception):** 0.1. The system encounters an error while writing the file (e.g., disk access failure). 0.2. The system displays the following error message: ”Save failed: An error occurred while saving your design.” 0.3. The system does not modify the existing hall design and allows the player to continue in Build Mode.

Special Requirements	• The saved JSON file must be free from corruption and fully loadable. • Clear and descriptive messages should be shown to the player about the save process status. • File names should be unique (e.g., including timestamps) to avoid overwriting existing files accidentally.
Frequency of Occurrence	• The player saves their hall design whenever they want to preserve progress in Build Mode.

Table 1: Use Case: Save Build Mode Design

Use Case ID	UC12
Use Case Name	Load Build Mode Design
Scope	Build Mode
Level	Subfunctional
Primary Actor	Player
Stakeholders and Interests	• **Player:** The player wants to load a previously saved hall design to continue editing or reviewing it in Build Mode. • **Game System:** – Ensure proper parsing and restoration of saved JSON files. – Handle errors gracefully when encountering missing or corrupted save files. • **Developer:** Must ensure that the load mechanism handles corrupted or missing files gracefully and provides meaningful feedback to the player.
Preconditions	• At least one valid save file must exist in the designated save directory. • The player must be in the Build Mode menu. • The game system must be compatible with the JSON save file format.
Postconditions	• The hall design, including object types, positions, and the hall type, is restored from the selected save file. • The hall design is fully restored, allowing the player to edit or continue working seamlessly from where they left off. • A confirmation message is displayed to the player indicating successful loading.

Main Success Scenario	1. The player enters Build Mode and selects the ”Load Design” option. 2. The system displays a list of available save files, including file names, hall types, and save dates. 3. The player selects a save file to load. 4. The system validates the integrity and compatibility of the selected save file. 5. The system parses the JSON data from the file and restores: 5.1. The hall type (e.g., Earth, Air, Water, Fire). 5.2. Object types (e.g., ”box”, ”chest”). 5.3. Object positions (x, y coordinates). 5.4. Any special attributes, such as whether an object contains a rune. 6. The system updates the current Build Mode state to reflect the loaded design. 7. The system displays a confirmation message: ”Design loaded successfully!” 8. The player resumes working in Build Mode with the loaded design.

Extensions	2a. **No Save Files Available:** 0.1. The system finds no valid save files in the designated directory. 0.2. The system displays the following message: "No saved designs available." 0.3. The player is returned to the Build Mode menu. 4a. **File Validation Failed:** 0.1. The selected save file is corrupted or incompatible with the current game version. 0.2. The system displays an error message: "Failed to load: Save file is corrupted or incompatible." 0.3. The player is returned to the file selection screen.
Special Requirements	• The JSON file format must be standardized and version-controlled to avoid compatibility issues. • The system should provide clear error messages for missing, corrupted, or incompatible save files.
Frequency of Occurrence	• Whenever the player wants to load a previously saved hall design in Build Mode.

Table 2: Use Case: Load Build Mode Design

Use Case ID	UC13
Use Case Name	Wizard Monster Dynamic Behavior Based on Timer
Scope	Play Mode
Level	Subfunctional
Primary Actor	Game System
Stakeholders and Interests	• **Player:** – Expects the Wizard monster to behave dynamically and influence the gameplay based on the timer. – Should have a clear understanding of the monster's actions (e.g., teleportation or rune movement). • **Game System:** – Must correctly evaluate the remaining time and dynamically alter the Wizard monster's behavior using the Strategy pattern. – Must dynamically adjust the Wizard monster's behavior based on the timer without causing errors or unexpected outcomes. • **Developer:** – Must implement the Strategy pattern for clean, maintainable, and extendable code. – Ensure that the monster's behavior integrates seamlessly with the timer and other game components.
Preconditions	• A Wizard monster must exist on the game map. • The game timer must be active and correctly tracking the remaining time. • The Wizard monster must be within a hall where it can influence the player or the rune.

Postconditions	• The Wizard monster dynamically reacts to the remaining time: – Less than 30% remaining: The player is teleported to a random location, and the monster disappears. – More than 70% remaining: The rune is teleported to a random location every 3 seconds until the timer changes state or the monster is removed. – Between 30% and 70% remaining: The monster does nothing and disappears after 2 seconds. • The monster's actions are applied without errors, and the Strategy pattern dynamically selects the appropriate behavior.

Main Success Scenario	1. The Wizard monster is initialized in the game world. 2. The game timer updates, and the remaining time is evaluated dynamically by the game system. 3. Based on the remaining time: • **Case 1 (Less than 30%):** 3.1. The monster teleports the player to a random location on the map. 3.2. The monster disappears immediately after teleportation. • **Case 2 (More than 70%):** 3.1. The monster teleports the rune to a random location every 3 seconds. 3.2. This behavior continues until the remaining time drops below 70% or the monster is removed. • **Case 3 (Between 30%–70%):** 3.1. The monster remains stationary for 2 seconds. 3.2. The monster disappears after the 2-second duration. 4. The player's position, the rune's location, or the monster's state is updated accordingly. 5. The system ensures smooth gameplay transitions without interruptions.

Extensions	2a. **Timer Fluctuation:** 0.1. If time changes dynamically due to bonus time or penalties, the monster should smoothly transition to the new behavior without causing sudden interruptions in gameplay. 0.2. For example: • If time increases to more than 70%, the monster begins teleporting the rune every 3 seconds. • If time decreases to less than 30%, the monster immediately teleports the player to a random location and disappears. 3a. **Player Interaction:** 0.1. If the player interacts with the monster (e.g., uses an enchantment), the monster's behavior may be interrupted or overridden based on the game rules.
Special Requirements	• The Strategy pattern must be used to implement the Wizard monster's behavior. • The system must ensure that the timer updates and monster behaviors are synchronized dynamically. • Teleportation effects (player or rune) must avoid invalid positions or collisions. • Clear visual feedback should indicate the monster's actions.
Frequency of Occurrence	• This behavior is triggered dynamically whenever a Wizard monster is present, ensuring adaptability to time-based game states.

Table 3: Use Case: Wizard Monster Dynamic Behavior Based on Timer

2 System Sequence Diagrams

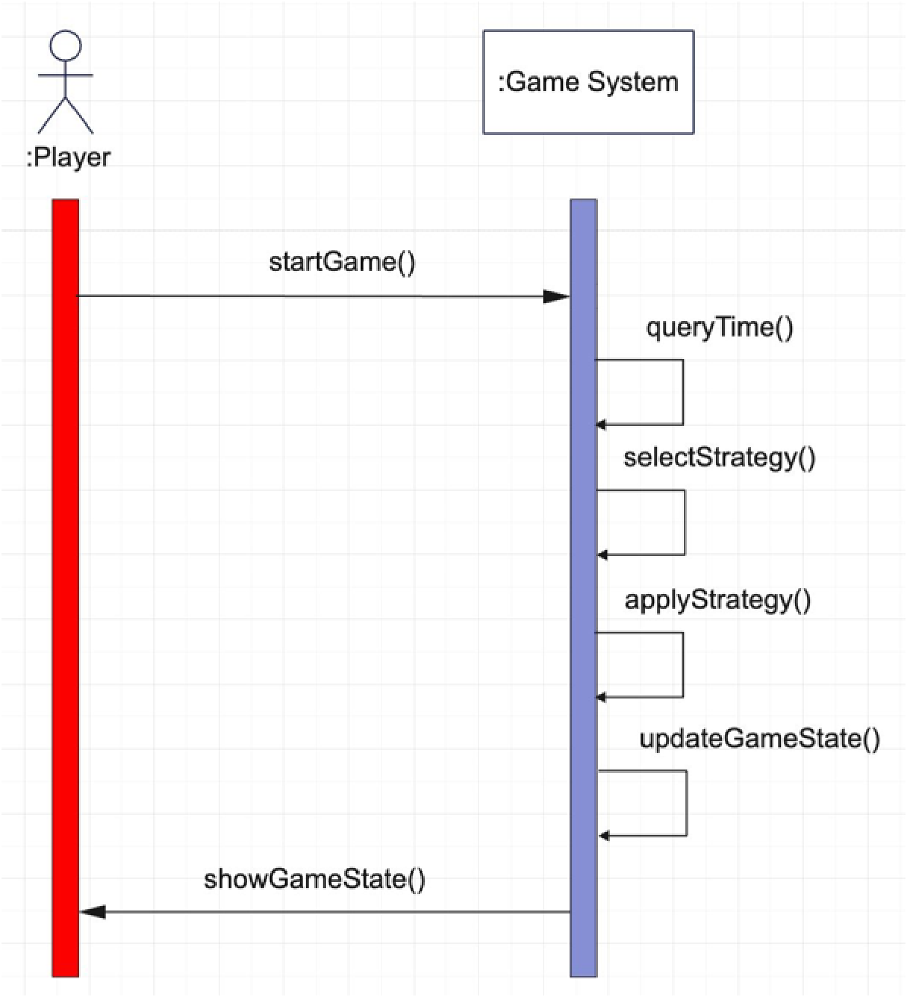

Figure 1: Update Wizard Behavior System Sequence Diagram

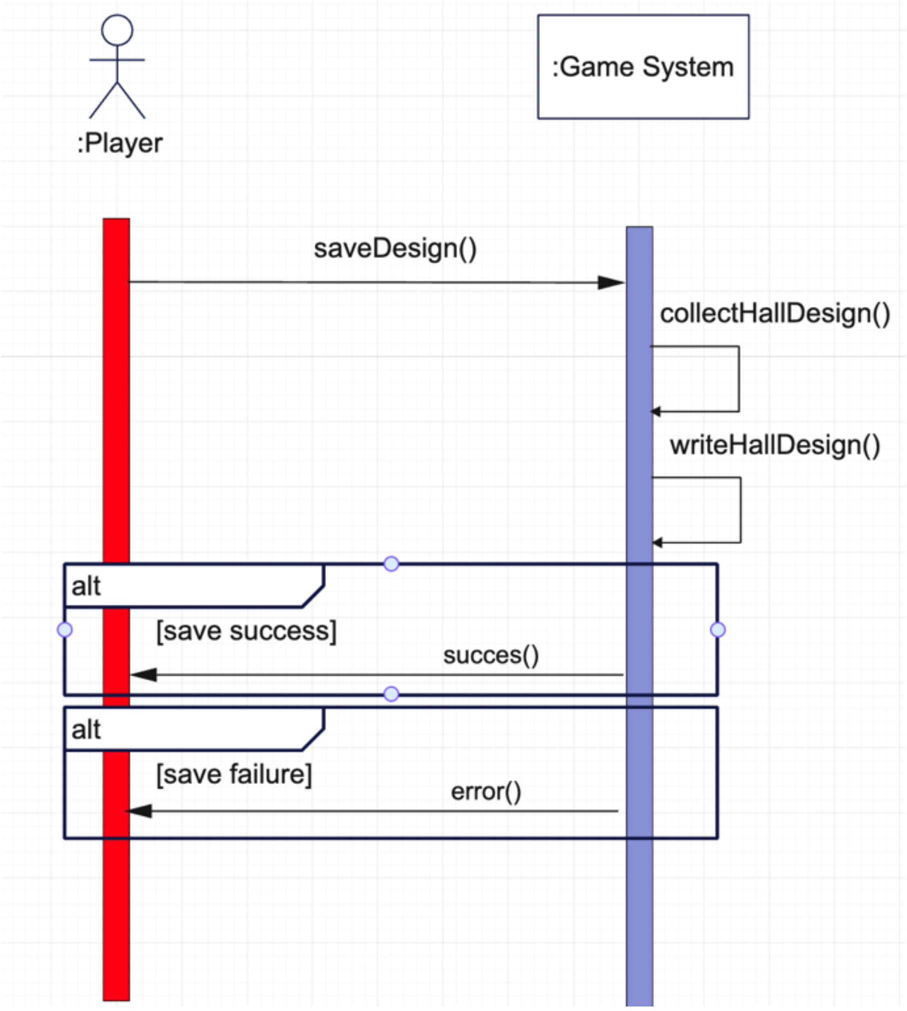

Figure 2: Save Build Mode Design System Sequence Diagram

3 Sequence Diagrams

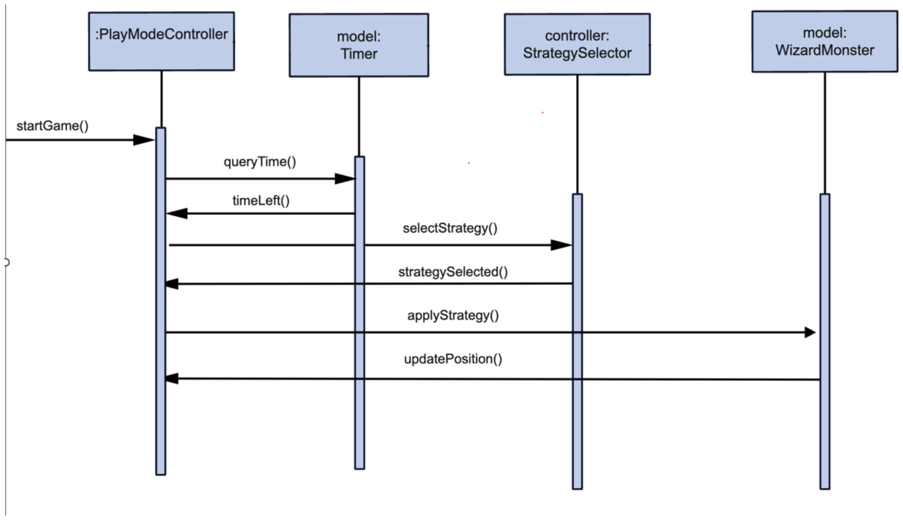

Figure 3: Update Wizard Behavior Sequence Diagram

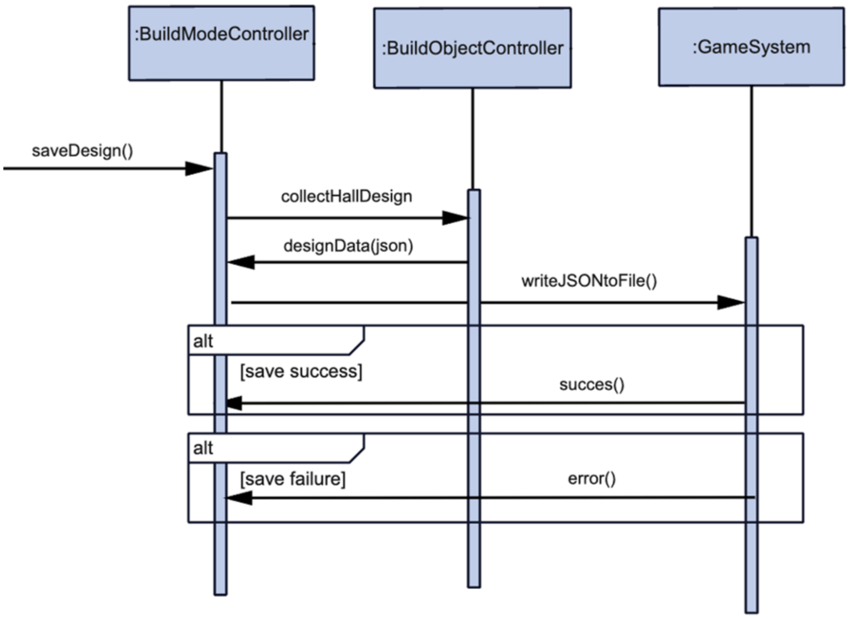

Figure 4: Save Build Mode Design Sequence Diagram

4 Operation Contracts

Operation Contract	**updateWizardBehavior()**
References	Use Cases: Wizard Monster Dynamic Behavior Based on Timer
Preconditions	• The game timer is active and tracking the remaining time. • A `WizardMonster` instance exists on the game map. • The player and rune are valid objects in the game world. • The `WizardMonster` is within a hall where it can influence the player or the rune.
Postconditions	• **If remaining time ¡ 30%:** – The player is teleported to a random, valid location. – The `WizardMonster` disappears from the map. • **If remaining time ¿ 70%:** – The rune is teleported to a random, valid location every 3 seconds until the timer drops below 70% or the monster is removed. • **If 30% ¡= remaining time ¡= 70%:** – The `WizardMonster` remains stationary for 2 seconds. – The `WizardMonster` disappears after the 2-second duration. • If the game timer fluctuates, the `WizardMonster` dynamically behaves accordingly.

Table 4: Operation Contract: updateWizardBehavior()

Operation Contract	**saveBuildModeDesign()**
References	Use Cases: Save Build Mode Design
Preconditions	• The player is in Build Mode. • At least one object is placed in the hall. • There is sufficient storage space to create or overwrite the save file.
Postconditions	• The current hall design, including object types, positions, and the hall type, is saved in a JSON file in the designated save directory. • A success message is displayed: ”Design saved successfully!”. • The saved design can be loaded and edited later. • If an error occurs (e.g., insufficient storage space or file write error), the system displays an appropriate error message.

Table 5: Operation Contract: saveBuildModeDesign()

5 Revised Domain Model Diagram

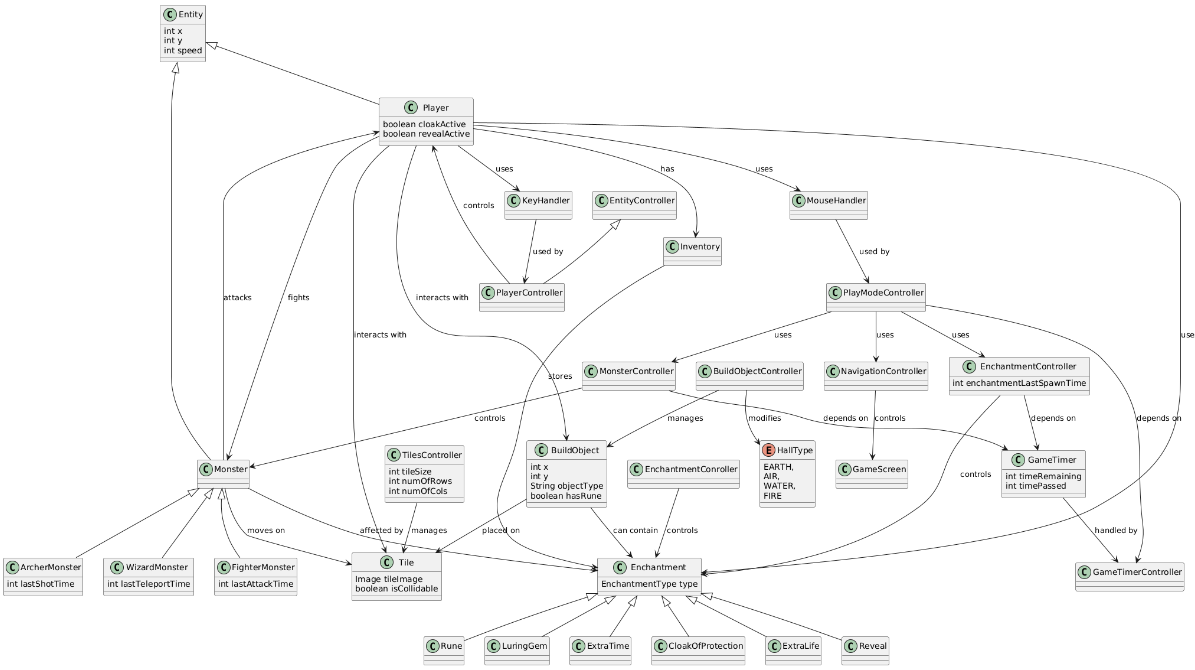

Figure 5: Revised Domain Model Diagram

6 Revised Class Diagram

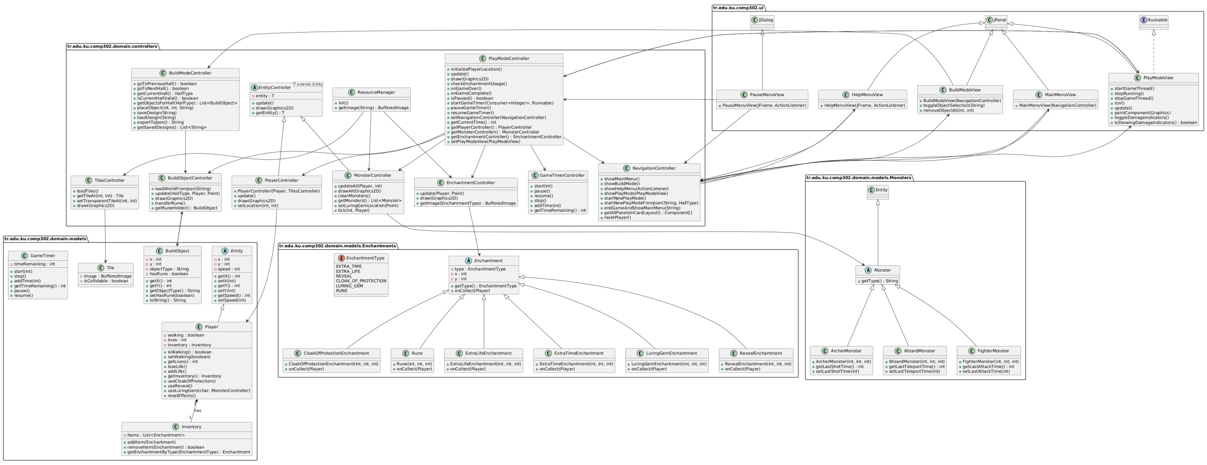

Figure 6: Revised Class Diagram