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 Diagram 1

2 Use Case Descriptions 2

3 System Sequence Diagrams 21

4 Operation Contracts 24

5 Domain Model Diagram 31

6 Vision 32

7 Supplementary Specifications 33

8 Glossary 34

November 27, 2024

1 Use Case Diagram

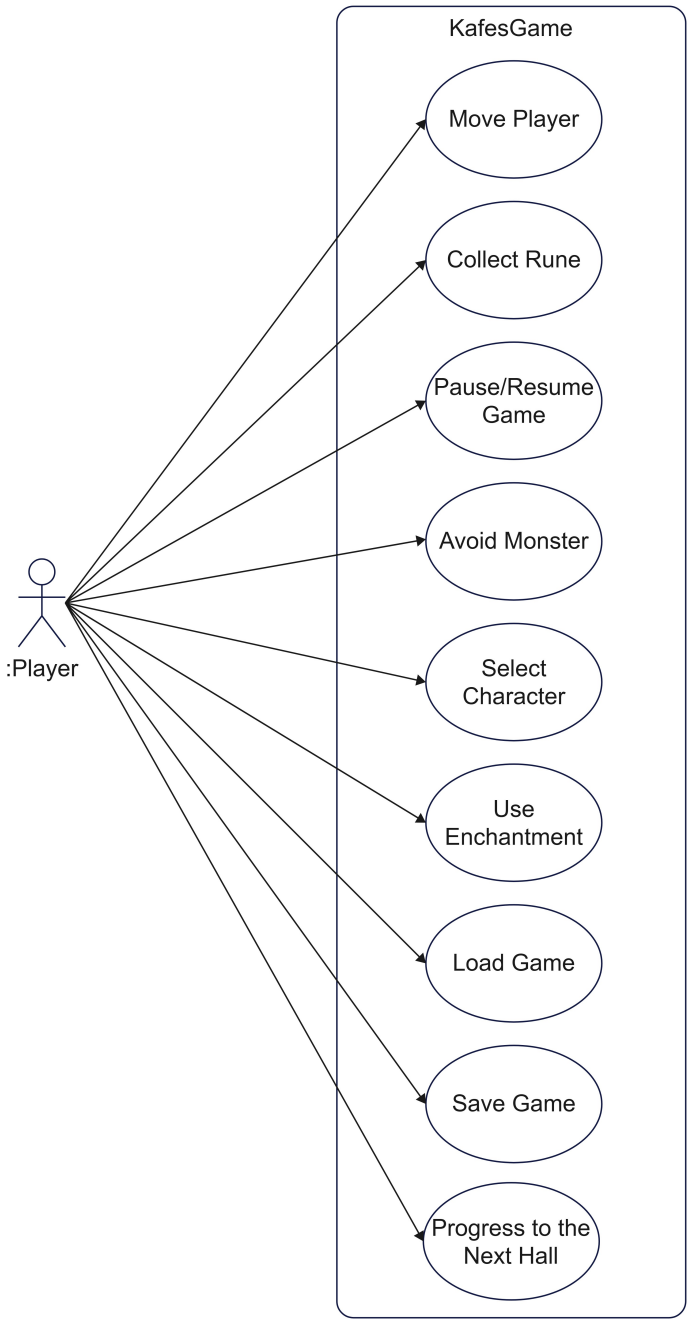

Figure 1: Use Case Diagram

2 Use Case Descriptions

Use Case ID	UC1
Use Case Name	Move Player
Scope	Play Mode
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to navigate the map effectively and complete objectives.
Preconditions	• The device must support keyboard inputs. • The player must have completed a valid map in Building Mode. • The game must be in Play Mode.
Postconditions	• The player moves their character seamlessly across the map.
Main Success Scenario	1. The player presses an arrow key to move. 2. The game processes the input and moves the character to the new position.
Extensions	1a) Obstacle encountered: The character remains in place.
Frequency of Occurrence	Any time the player presses an arrow key during Play Mode.

Table 1: Use Case: Move Player

Use Case ID	UC2
Use Case Name	Collect Rune
Scope	Play Mode
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	• **Player:** Aims to locate and collect runes in each game hall to progress to the next hall. • **Developer:** Wants to ensure the runes are hidden effectively but collectible to make the game challenging to the player.
Preconditions	• The player is next to an object containing a rune.
Postconditions	• The rune is revealed. • The door to the next hall is unlocked.
Main Success Scenario	1. The player comes next to an object. 2. The player interacts with the object by clicking left mouse on it. 3. The system checks if the object contains a rune. 4. If a rune is found, it is revealed. 5. The game unlocks the door to the next hall.
Extensions	3a) Another item is dropped from the object. Player adds the item to the inventory.
Frequency of Occurrence	Occurs once per hall.

Table 2: Use Case: Collect Rune

Use Case ID	UC3
Use Case Name	Use Enchantment
Scope	Play Mode
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	• **Player:** Use enchantments to gain advantages and overcome game challenges. • **Developer:** Wants Enchantments to be balanced, effective, and meaningful to the player.
Preconditions	• The player must have at least one enchantment in their inventory.
Postconditions	• The Enchantment effect becomes active. • The Enchantment is removed from the player's inventory.
Main Success Scenario	1. The player chooses an enchantment from the inventory. 2. The system applies the effect of the enchantment (e.g., Cloak of Protection enchantment makes the player invisible). 3. The system starts the duration of the effects (if they exist). 4. Once the time is up, the system removes the effects.

Extensions	3a) The effect of Enchantment is desired to be ended early: • If the player manually cancels the enchantment, the system removes the effects. 4a) There are no enchantments in the player's inventory: • The system displays an error message and does not take any action.
Frequency of Occurrence	It occurs multiple times depending on the number of enchantments in the player's inventory and their needs in-game.

Table 3: Use Case: Use Enchantment

Use Case ID	UC4
Use Case Name	Select Character
Scope	Character Selection Mode
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to choose a character with unique abilities and starting enhancements to fit their playstyle. • **Developer:** Wants to ensure character selection is clear, bug-free, and offers replayability by providing diverse characters.
Preconditions	• The game is in the character selection stage, before entering gameplay or the build mode. • All selectable characters are unlocked and ready for display.
Postconditions	• The game system stores the selected character's data. • The selected character's unique enhancements, passives, or special powers are stored in the gameplay environment.

Main Success Scenario	1. The player is in the character selection stage after starting the game. 2. The system displays available characters, each with predefined: • Appearance • Starting Enhancements (e.g., Extra time; Luring Gem). • Passive Abilities (e.g., increased max health, gaining more effects on some Enhancements). • Unique Special Powers (e.g., time freeze, killing an enemy). 3. The player selects a character. 4. The system confirms the selection and transitions to the build mode and gameplay with the chosen character's configuration.
Extensions	1a) Player does not select a character: • Game does not start or optionally starts with a default character.
Frequency of Occurrence	Once per game session, before entering build mode.

Table 4: Use Case: Select Character

Use Case ID	UC5
Use Case Name	Pause/Resume Game
Scope	Play Mode
Level	Subfunctional
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to pause the game during interruptions and resume it without losing progress.
Preconditions	• The game is in play mode
Postconditions	• The player will not be able to move its character. • The characters controlled by the game system will not be able to act.
Main Success Scenario	1. The player interacts with the pause button while they are in play mode. 2. The system records the interaction and restricts all the character actions. 3. Any timer in game is stopped. 4. The player interacts with the resume button and the characters are not restricted of actions and timers continue again.

Extensions	4a) The player interacts with help button: • The help screen appears. • The game is still in pause state. 4b) The player interacts with the main menu button: • Game exits to the main menu.
Frequency of Occurrence	Any time the player interacts with the resume/pause button.

Table 5: Use Case: Pause/Resume Game

Use Case ID	UC6
Use Case Name	Avoid Monster
Scope	Play Mode
Level	User-Goal
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to navigate the game world safely without encountering monsters that could harm them. • **Monsters:** Should behave according to their hard-coded patterns, creating an immersive experience. • **Game System:** Must provide a challenging environment where avoiding monsters is a viable strategy, ensuring game balance. • **Developer:** Interested in player engagement and satisfaction through well-designed game mechanics.
Preconditions	• The player is in an area where monsters can appear. • The game is in play mode and the player is in control of their character. • The monster exists in the game world and is active.
Postconditions	• The player successfully avoids the monster without being detected.

Main Success Scenario	1. The player detects a monster ahead (e.g., through line of sight or a mini-map indicator). 2. The player decides to avoid the monster to prevent attack from the monster. 3. The player chooses an alternative path or uses stealth mechanics. 4. The player navigates around the monster's detection zones successfully. 5. The player continues on their journey without alerting the monster.
Extensions	3a) No Alternative Path Available: (a) The player uses an item or skill to become temporarily invisible. (b) The player waits for the monster to move away. (c) Return to Step 4. 4a) Monster Changes Patrol Route Unexpectedly: (a) The monster moves towards the player's current location. (b) The player must quickly find cover or retreat. (c) If successful, return to Step 4; else, proceed to Extension 4c.
Frequency of Occurrence	Continuous throughout the game as monsters are encountered.

Table 6: Use Case: Avoid Monster

Use Case ID	UC7
Use Case Name	Progress to Next Hall
Scope	Play Mode
Level	User-Goal
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to advance through the game by moving to the next hall. • **Game System:** Needs to seamlessly load new game areas, ensuring consistency and performance without disrupting gameplay. • **Monsters:** May appear or behave differently as the player progresses, affecting game difficulty.
Preconditions	• The player is in a hall with an accessible exit door to the next hall. • The game is in play mode, and the player controls their character.
Postconditions	• The player successfully enters the next hall, and the game state updates accordingly.

Main Success Scenario	1. The player navigates to the exit point of the current hall. 2. The player performs an action to interact with the exit (presses a key/button to open the door). 3. The game system verifies if all conditions to progress are met, like completion of objectives. 4. The game system saves the current state to prevent data loss during the transition. 5. The game system initiates the loading sequence for the next hall. 6. The player's character appears at the entry point of the new hall. 7. The game system updates the player's map, objectives, and any new environmental elements. 8. The player resumes control and continues exploring the new hall.
Extensions	2a) Exit door is locked: (a) The game system displays a message: "The door is locked." (b) The player searches for the rune to unlock the door. (c) Once the exit door is unlocked, return to Step 2. 3a) Loading next hall fails: (a) The game system encounters an error during loading. (b) The game system attempts to reload the hall. (c) If successful, proceed to Step 6. (d) If reload fails, the game saves progress and returns to the main menu.

Special Requirements	• Performance: Transition between halls must be smooth with minimal loading times to maintain immersion. • Data Integrity: The game must correctly save and load all relevant data during the transition. • Dynamic Content: If halls are procedurally generated, the generation algorithm must ensure coherent and navigable layouts. • User Feedback: Provide visual or audio cues during the transition to inform the player of progress (e.g., loading screens, animations). • Accesibility: Ensure that any necessary information (e.g., "Door is locked") is clearly communicated to the player.
Data Variations	• Hall Types: Varying themes, sizes, difficulties, and enemy types.
Frequency of Occurrence	At the end of every hall after the player locates the rune and unlocks the door.

Table 7: Use Case: Progress to the Next Hall

Use Case ID	UC8
Use Case Name	Save Game Progress
Scope	Play Mode
Level	Subfunctional
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to save current game progress to continue playing later without losing achievements, items, or progress. • **Game System:** – Needs to accurately capture and store/save the game's current state to ensure consistency while loading after. – Needs to accurately identify and save the player's achievements, items and progress so that the player can reach them later. • **Developer:** The developer must be sure to prevent data corruption and ensure that the saved files do not crash the game.
Preconditions	• The game is currently in play mode and not in a critical with uninterruptible process. • The player has made progress in the game that can be saved. • There is sufficient storage space available on the device to create or overwrite a save file.

Postconditions	• The game's current state, including hall status, remaining time, inventory, lives, player position, and any active enchantments, is saved to a storage medium. • A confirmation message is displayed to the player indicating that the game has been saved successfully. • The saved game data can be loaded later, allowing the player to resume from the exact point of saving.

Main Success Scenario	1. The player pauses the game while in play mode.
	2. The player selects "Save Game" option.
	3. The game system gathers all relevant data.
	  • Current hall and its state.
	  • Player's position within the hall.
	  • Remaining time on the timer.
	  • Inventory contents (enchantments, runes collected etc.)
	  • Number of lives remaining.
	  • Active status or enchantments.
	  • Monster positions and states.
	4. The game system converts the collected data into a format that is suitable for storage.
	5. The game system writes the serialized data to a save file in the designated save directory on the device.
	6. The game system system displays a message to the player: "Your game has been saved successfully."

Extensions	2a) Insufficient storage space: (a) The game system displays an error message: "Save failed: Not enough storage space available." (b) The game system returns the player to the in-game menu without altering the current game state. 2b) Save file corruption or write error: (a) The game system encounters an I/O exception during the save process. (b) The game system displays an error message: "Save failed: An error occurred while saving your progress. Error: xxx ".
Special Requirements	• Clear and immediate messages should inform the player for the current status of the save process.
Frequency of Occurrence	Whenever a player wants to save or exists from the game.

Table 8: Use Case: Save Game Progress

Use Case ID	UC9
Use Case Name	Load Saved Game
Scope	Main Menu
Level	Subfunctional
Primary Actor	Player
Stakeholders and Interests	• **Player:** Wants to resume gameplay from a previously saved state without losing achievements, items, or progress. • **Game System:** – Needs to accurately restore the game state from the saved data to ensure continuity. – Must handle loading operations efficiently without causing performance issues or crashes. • **Developer:** The developer must be sure to prevent data corruption and ensure that the loading files do not crash.
Preconditions	• At least one valid saved game file must exist on the device in the designated save directory. • The game should be at the main menu. • The game system is compatible with the version of the saved game file.

Postconditions	• The game's state is restored to match the data from the selected saved game file. • The player resumes gameplay from the exact point where the game was saved. • Any necessary game components are loaded appropriately. • A confirmation is provided to the player indicating that the game has been loaded successfully.

Main Success Scenario	1. The player launches the game and reaches the main menu.
	2. The player selects "Load Game" option.
	3. The game system displays a list of saved files including date/time of save, player progress, hall name details.
	4. The player selects the game to load.
	5. The game system checks the integrity and compatibility of the selected save file.
	6. The game system reads the saved data from the file.
	7. The game system restores all game state elements:
	  • Current hall and its state.
	  • Player's position within the hall.
	  • Remaining time on the timer.
	  • Inventory contents (enchantments, runes collected etc.)
	  • Number of lives remaining.
	  • Active status or enchantments.
	  • Monster positions and states.
	8. The game transitions to the gameplay screen.
	9. A message is displayed to the player: "Game loaded successfully".

Extensions	3a) No save files available: (a) The game system displays a message: "No saved games available." (b) The player is returned to the main menu.
Special Requirements	• Clear and immediate messages should inform the player for the current status of the load process.
Frequency of Occurrence	Whenever a player wants to load a saved game.

Table 9: Use Case: Load Saved Game

3 System Sequence Diagrams

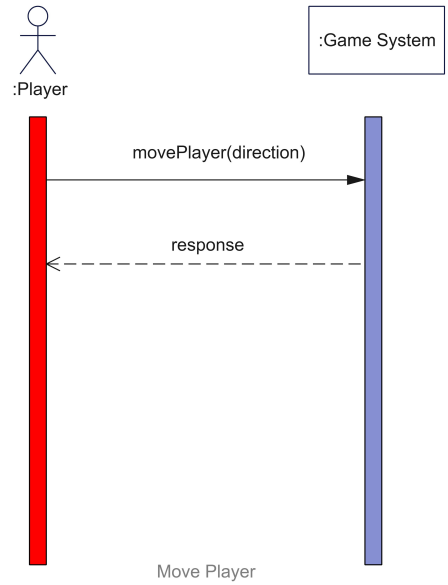

Figure 2: Move Player

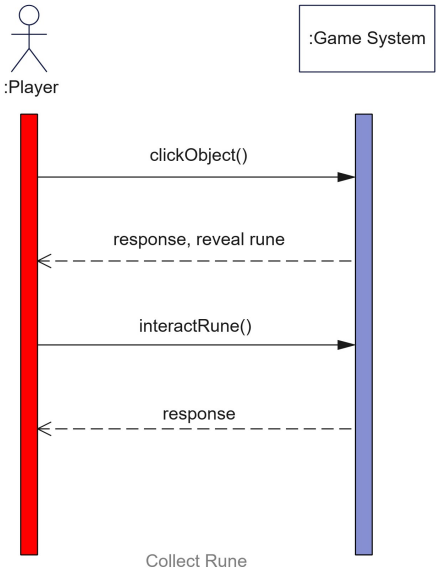

Figure 3: Collect Rune

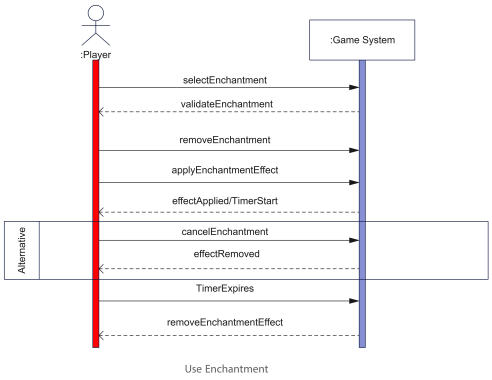

Figure 4: Use Enchantment

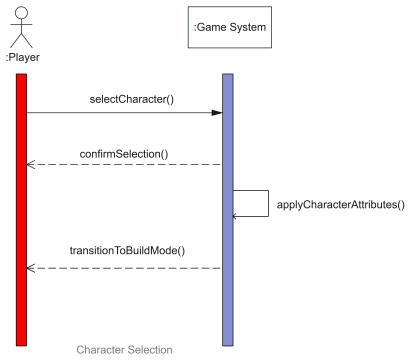

Figure 5: Change Character

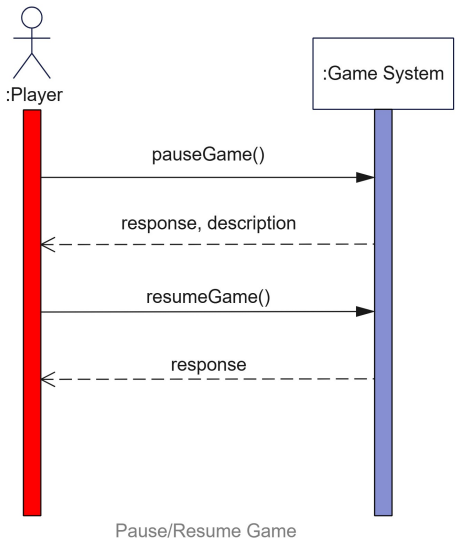

Figure 6: Pause/Resume Game

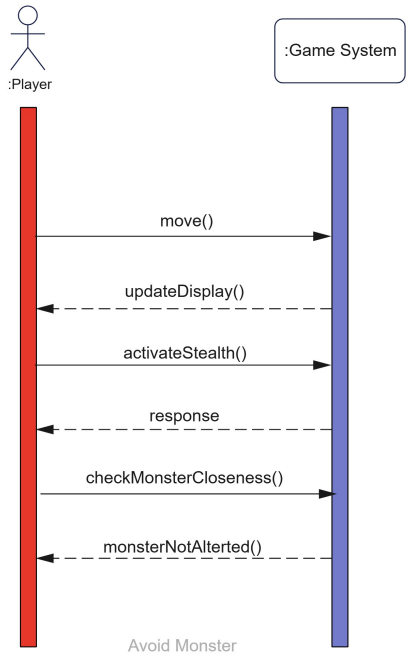

Figure 7: Avoid Monster

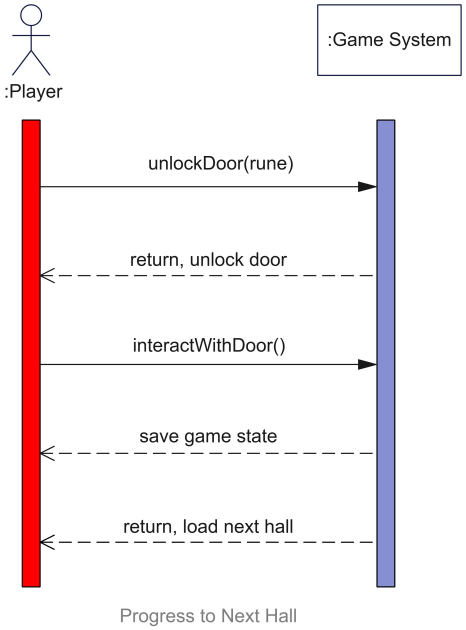

Figure 8: Progress To Next Hall

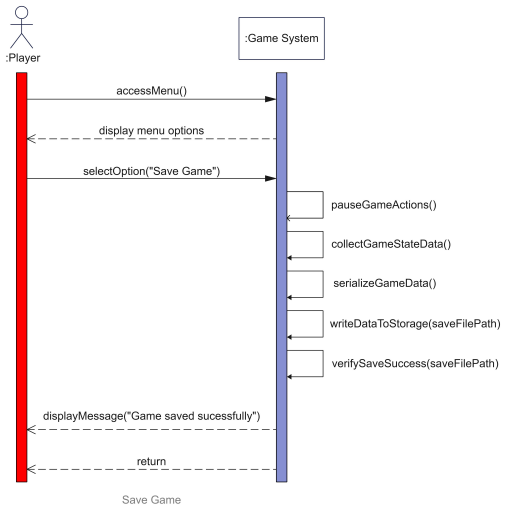

Figure 9: SaveGame

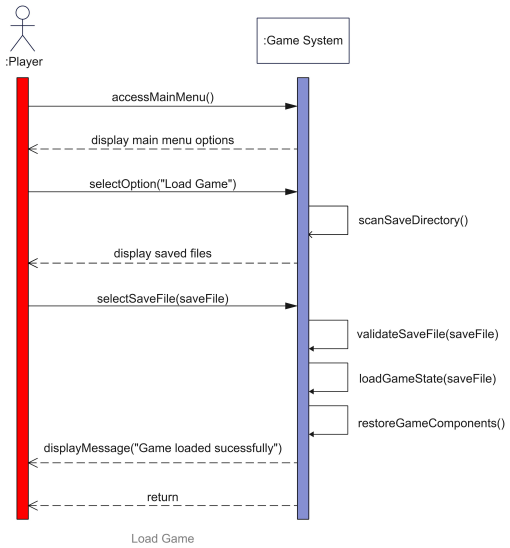

Figure 10: Load Game

4 Operation Contracts

Operation Contract	**pauseGame()**
References	Use Cases: Pause/Resume Game
Preconditions	• The game is running.
Postconditions	• The game state is paused; all characters stop acting and the timer stops.

Table 10: Operation Contract: pauseGame()

Operation Contract	**resumeGame()**
References	Use Cases: Pause/Resume Game
Preconditions	• The game is stopped.
Postconditions	• The game state is resumed; all characters start acting again and the timer starts where it's left.

Table 11: Operation Contract: resumeGame()

Operation Contract	**revealRune(player: Player, object: Object)**
References	Use Cases: Collect Rune
Preconditions	• The player is next to an object. • The object has not yet been checked.
Postconditions	• If the object contains a rune: – The rune is revealed, and status of the object is updated. – The door to the next hall is unlocked.

Table 12: Operation Contract: revealRune()

Operation Contract	**movePlayer(direction: Direction)**
References	Use Cases: Move Player
Preconditions	• The next grid to move does not contain an object or a monster.
Postconditions	• The player moves to the next grid depending on the direction: – Arrow Up -¿ North – Arrow Down -¿ South – Arrow Left -¿ West – Arrow Right -¿ East

Table 13: Operation Contract: movePlayer()

Operation Contract	**useItem(itemID: ItemID)**
References	Use Cases: Avoid Monster
Preconditions	• The game is running. • The player possesses the item specified by `itemID` in their inventory.
Postconditions	• The item's effect is applied immediately (e.g., granting invisibility or creating a distraction). • The item is consumed or its usage count is decreased. • The player's inventory is updated accordingly.

Table 14: Operation Contract: useItem()

Operation Contract	**interactWithDoor()**
References	Use Cases: Progress to Next Hall
Preconditions	• The game is running. • The player is at the exit point of the current hall. • The player controls their character.
Postconditions	• The game checks progression conditions. • If conditions are met, the game state is saved, and the next hall begins loading. • If conditions are not met, a message is displayed to indicate unmet conditions.

Table 15: Operation Contract: interactWithDoor()

Operation Contract	**unlockDoor(itemID: ItemID)**
References	Use Cases: Progress to Next Hall
Preconditions	• The game is running. • The exit point is locked. • The player possesses the required item in their inventory.
Postconditions	• The exit is unlocked by changing its status. • The required item is consumed or removed from the inventory. • Visual indicators are updated to reflect the unlocked state.

Table 16: Operation Contract: unlockDoor()

Operation Contract	**selectEnchantment(enchantment: Enchantment)**
References	Use Cases: Use Enchantment
Preconditions	• The player's inventory must contain at least one enchantment. • The selected enchantment must be available for use.
Postconditions	• The selected enchantment is removed from the inventory. • The system prepares the selected enchantment for application.

Table 17: Operation Contract: selectEnchantment()

Operation Contract	**applyEnchantmentEffect(enchantment: Enchantment)**
References	Use Cases: Use Enchantment
Preconditions	• The `selectEnchantment` operation must have been successfully completed. • The selected enchantment's effect must be applicable.
Postconditions	• The enchantment's effect is applied to the player or the game world. • If the enchantment has a timed effect, the timer is started.

Table 18: Operation Contract: applyEnchantmentEffect()

Operation Contract	**cancelEnchantment(enchantment: Enchantment)**
References	Use Cases: Use Enchantment
Preconditions	• The enchantment's effect must be active.
Postconditions	• The enchantment's effect is cancelled. • The system updates the player's inventory state if necessary.

Table 19: Operation Contract: cancelEnchantment()

Operation Contract	**savеProgress(player: Player)**
References	Use Cases: Save Progress
Preconditions	• The game is running and not in a critical, uninterruptible process. • The player has made progress that can be saved. • Sufficient storage space is available on the device.
Postconditions	• The game's current state is saved to a storage medium, including: – A confirmation message is displayed to the player: "Your game has been saved successfully." • If saving fails due to insufficient storage or write errors: – The system displays an appropriate error message. – The game state remains unchanged.

Table 20: Operation Contract: saveProgress()

Operation Contract	**loadSavedGame(player: Player, saveFile: File)**
References	Use Cases: Load Saved Game
Preconditions	• At least one valid saved game file exists in the designated save directory. • The game is at the main menu or starting page. • The selected save file is compatible with the game version.
Postconditions	• The game state is restored to match the data from the selected saved game file, including: – The game transitions from the main menu to gameplay. – A confirmation message is displayed to the player: "Game loaded successfully." • If no valid save files are available: – The system displays a message: "No saved games available." – The player remains at the main menu.

Table 21: Operation Contract: loadSavedGame()

5 Domain Model Diagram

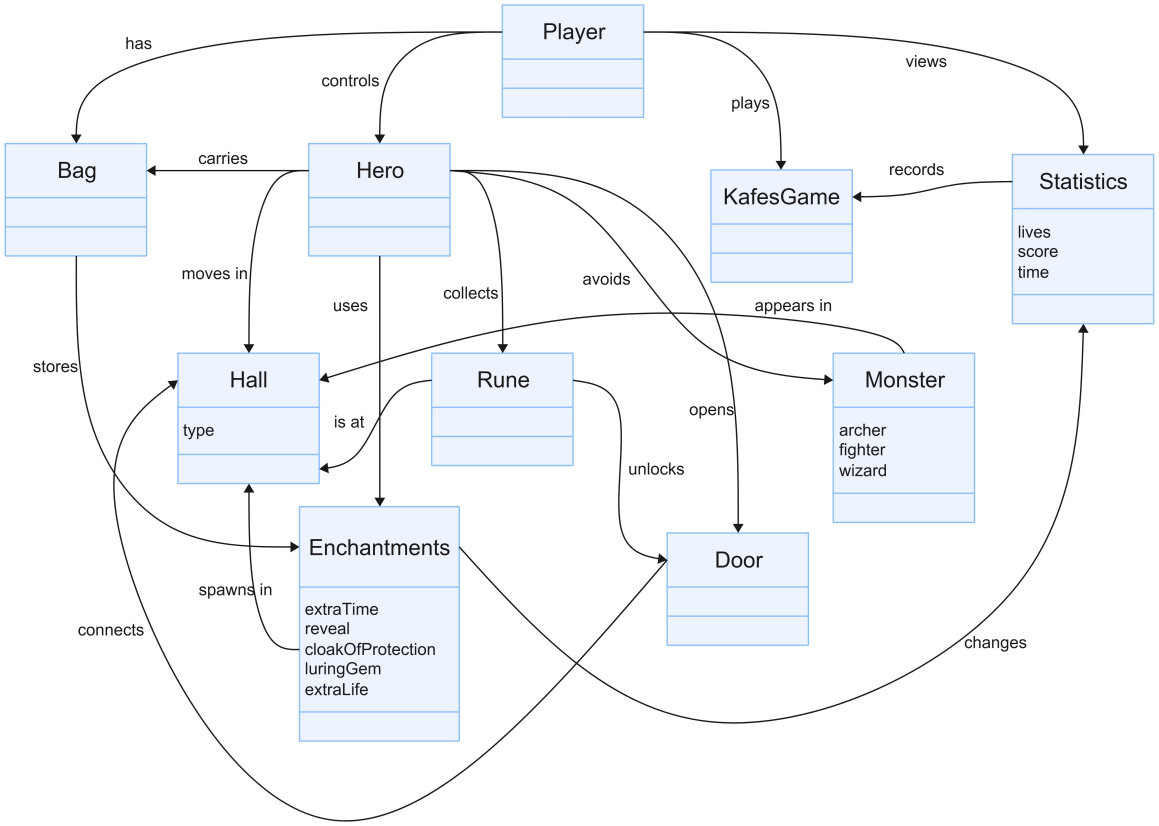

Figure 11: Domain Model Diagram

6 Vision

Introduction

This report aims to show the general vision of the team and the project.

Positioning

Business Opportunity

The gaming industry is getting bigger and bigger everyday. Since the game project "Kafes Game" is set in Koç University as team's COMP302 project, it is not feasable to deploy as a business product in the market. However, it can be continued to develop and turned into an indie-game, that is suitable as a commercial product.

Target Audience

Target audience is anyone who likes to play games. Specifically people who enjoy rogue-like action adventure games.

Scope

The game is a single-player rogue-like game with finite amount of levels(halls) with enchantments, runes and monsters.

Stakeholders and Users

Stakeholders

- **Project Member**: The project members aim to create a basic 2D rogue-like action game. Agile methods are applied in this project to overcome the complexity of teamwork.

Users

- **Player**: The player wants an entertaining rogue-like action adventure game.

References

- COMP302 Lecture Slides from Koç University
- *Applying UML and Patterns*, 3rd Edition, by Larman, Pearson Education

7 Supplementary Specifications

The following supplementary specifications outline the key requirements and constraints for the "Escape From Koç" game project:

System Requirements

- **Platform**: The game will be developed for Windows platform.
- **Input Devices**: The game will support mouse and keyboard for PC.

Functional Requirements

- **Gameplay**: The game will feature various levels with varying difficulties.
- **Save/Load Progress**: Players will be able to save their progress at any point, and resume from the last saved state.
- **Interactive Objects**: Players will interact with objects within the environment, such as doors, objects, and items, to progress through the game.

Non-Functional Requirements

- **Performance**: The game must run smoothly at a minimum of 30 frames per second (FPS) on standard PC hardware.
- **Usability**: The game should be intuitive to play, with easy-to-use controls and an engaging interface.
- **Scalability**: Future updates or additional levels should be easily integrated into the existing game system.

Constraints

- **Deadline**: The game must be developed and ready for release by the end of the academic semester.
- **Budget**: The project has a limited budget, and all assets (such as graphics and sound) should be either created by the team or sourced from free/open libraries.

Assumptions

- **Team Skills**: The development team is proficient in Java and basic game development principles.
- **Player Knowledge**: The target audience is familiar with basic rogue-like action adventure games.

8 Glossary

Term	Description
Player	The character controlled by the user to navigate the dungeon, collect runes, and avoid monsters.
Rune	An item that unlocks new dungeon halls and allows progression.
Monster	Enemies that behave differently; contact causes health loss.
Enchantment	Items stored in the inventory to assist the player, like invisibility or revealing rune locations.
Hall	A dungeon room containing objects, monsters, runes, and the player.
Timer	A countdown that forces the player to find runes and escape before time runs out.
Build Mode	The mode where the player sets up the hall before gameplay.
Play Mode	The phase where the player explores the dungeon and tries to escape.
Chests	Items that restore health; losing all health results in death.
Pause/Resume	A feature that lets the player pause and resume the game without losing progress.
Help Screen	A screen explaining the game rules, enhancements, and monsters.
Main Menu Screen	The first screen showing options to start, view help, or exit.
Character Selection	A screen where the player chooses a character with unique abilities.
Git	A version control system for tracking code changes.
GitHub	A platform for hosting Git repositories and collaboration.
Google Drive	Cloud storage for file sharing and collaboration.
IDEs	Software tools for coding, debugging, and compiling.
VSCode	A lightweight, open-source code editor.

Term	Description
LaTeX	A system for creating high-quality academic documents.
Overleaf	An online LaTeX editor for collaborative writing.
Java	A widely-used programming language for building applications.
Edraw.ai	A diagramming tool for creating flowcharts and UML diagrams.
Use Cases	Descriptions of system behavior in response to user actions.
System Sequence Diagram	A diagram showing the sequence of operations in response to user actions.
Operation Contract	A formal description of an operation's preconditions, postconditions, and behavior.
Domain Model Diagram	A diagram showing key objects in the system and their relationships.