# Need For Spear Game

COMP 302 - Fall 2021

KUse Case

Final Report

Çisem Özden

Doruk Örnekci

Mehmet Çuhadar

Meryem Karakaş

Zeynep Sıla Kaya

KOÇ UNIVERSITY

# Table of Contents

# Introduction

We envision a fun paddle brick breaker type of game, which is also our Comp302 group project, named “Need For Spear” (NFS for short). We aim for NFS to support multiple users (players), with the flexibility of both direct playing and level design phases. We also aim for both a single-player experience and a multiplayer experience, which is up to the players.

## Positioning

The NFS game is the Comp302 project, which we need to design and implement according to class requirements and lecture materials. In addition to that, we have the opportunity to achieve the best project in the semester title, which motivates us to put more than enough effort into this project.

## Stakeholder Descriptions

**User(s):** playing the game, level design, playing against other users

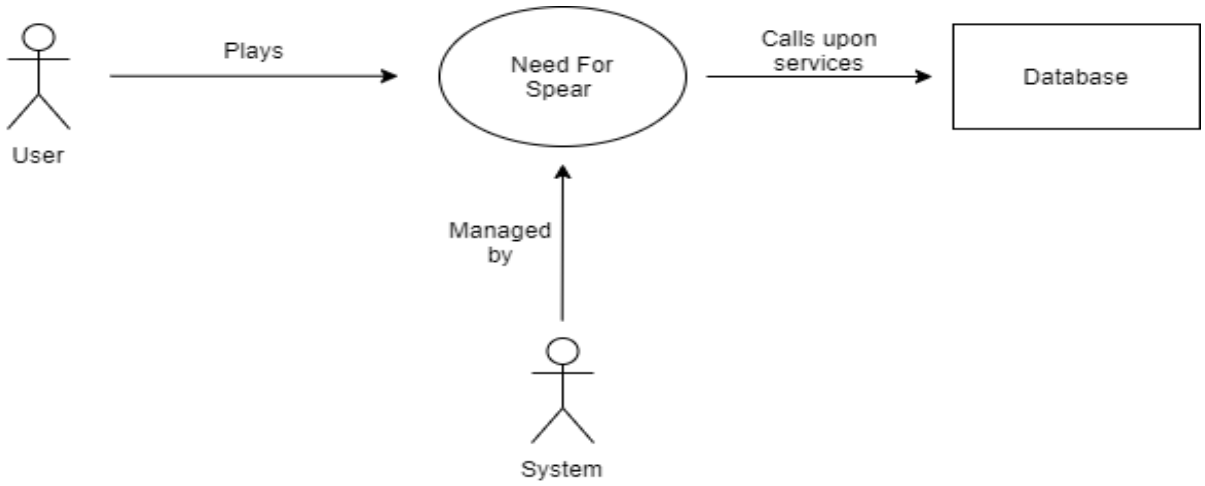
**System:** user authentication, error handling, data collection and data processing

**Developer(s):** providing a successful project

**Lecturer and TA’s:** overseeing the project phases, grading the developer(s)

## Product Overview

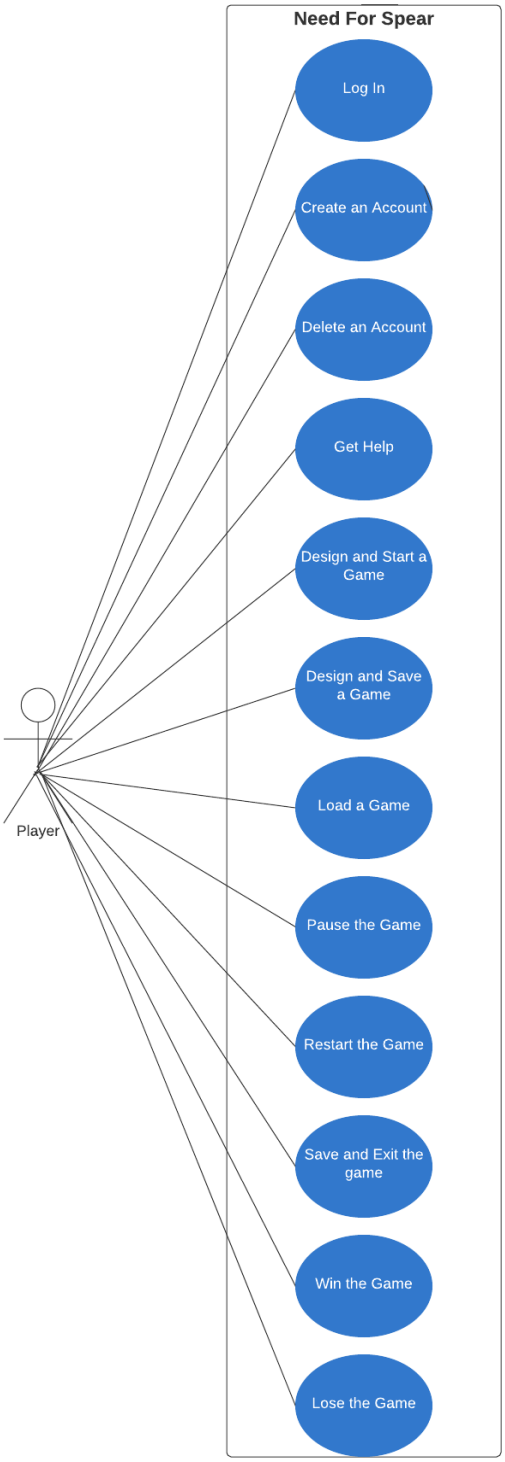
The NFS game will be available to be played through personal computers. The game will provide a fun experience to users (players) and collaborate with the database for data storing and processing.


User
Plays
Need For Spear
Calls upon services
Database
Managed by
System

# Teamwork Organization

During the project process, we often met at school or online to discuss what we should do and how we should plan it.In the parts before the code part started, we were usually divided into individual or binary groups and everyone was working on their part, and then when the deadline approached, everyone was telling what they had done and we were putting together what came out. When we moved to the code part, we were working individually or in pairs in the same way. It is not possible to tell by name who worked on which branch, as everyone was working on very different topics and branches throughout the project, but we have used GitLab effectively and all of our pushes and merges can be accessed throughout the commit date.During the periods when we were supposed to upload the Phase 1 and Phase 2 demos, we generally gathered together and looked at the project on one of our computers, and in this way, we completed the deficiencies in our code and corrected our mistakes.

# Use Case Diagram



# Use Cases

## Use Case 1: Load a Game

**Scope:** Need for Spear, Game

**Primary Actor:** The User

**Stakeholders and Interests:**

The User: Wants to play a saved game.

**Preconditions:**

The user must be logged into the game.

**Success Guarantee (Postconditions):**

The game that the user wants to play must be successfully saved in the database.

**Main Success Scenario:**

1. The user opens the game.
2. The user logs into his/her account.
3. The user opens the saved games screen.
4. The user selects a game among the preexisting games.
5. Selected game is loaded.

**Extensions (or Alternative Flows):**

*a At any time, the user can close the game.

*b At any time the connection may fail.

2a. The user enters wrong account information.

1. System signals error and rejects entry.

2. The user responds to the error.

    2a. The user tries to enter the credentials again.

    2b. The user creates an account.

4a. The user could not find the saved game he/she wanted to play.

1. The user searches for another saved game.
2. The user selects a different game.

**Special Requirements:** -

**Technology and Data Variations List:** -

**Frequency of Occurrence:** Whenever a user wants to play a saved game.

**Open Issues:** -

## Use Case 2: Design and Start a Game

**Scope:** Need for Spear, Game

**Primary Actor:** The User

**Stakeholders and Interests:**

The User: Wants to start a game after he/she designs it.

**Preconditions:**

The user must be logged into the game.

**Success Guarantee (Postconditions):**

Login is successful. The user can design and then start the game he/she has just designed.

**Main Success Scenario:**

1. The user logs in.
2. The user enters the building mode.
3. The user chooses how many of each obstacle there will be.
4. Obstacles are placed in random locations.
5. The user starts the newly designed game.

**Extensions (or Alternative Flows):**

*a. At any time, the user can close the game.

*b. At any time, the connection may fail.

1a. The user enters wrong account information.

2. System signals error and rejects entry.

2. The user responds to the error.

2a. The user tries to enter the credentials again.

2b. The user creates an account.

3a. Minimum number of obstacles is not satisfied.

3. System signals error and wants a new entry.

4. The user responds to the error and enters new valid numbers.

4a. The user leaves the number of obstacles as they are.

4b. The user changes the places of obstacles.

2. Puts obstacles in positions s/he desired.

1a. If the obstacles overlap, system gives error

2. User repositions obstacles

**Special Requirements:** -

**Technology and Data Variations List:**

A database is required to store the game information.

**Frequency of Occurrence:**

Whenever the user wants to design and start a new game.

**Open Issues:** -

## Use Case 3: Save And Exit the Game

The User: Save the game, then exit.

**Preconditions:**

The game must be paused

**Success Guarantee (Postconditions):**

The game information is successfully saved in the databases and then left the game without an error.

**Main Success Scenario:**

1.The user logs in.

2.The user starts the game.

3.The user pauses the game.

4.The user saves the game.

5.The user exits the game.

**Extensions (or Alternative Flows):**

*a. At any time, the user can close the game.

*b. At any time, the connection may fail.

1a. The user enters wrong account information.

2. System signals error and rejects entry.

2. The user responds to the error.

2a. The user tries to enter the credentials again.

2b. The user creates an account.

2a. The user starts an already existing game.

2. The user resumes the game. 2b. The

user designs a new game.

4. The user chooses the building mode.
5. S/he designs the game according to her/his preferences.
6. The user starts the game.

**Special Requirements:** -

**Technology and Data Variations List:**

A database is required to store the game information

**Frequency of Occurrence:**

Whenever the user wants, but only one time.

**Open Issues:** -

## Use Case 4: Lose the Game

**Scope:** Need for Spear, Game

**Primary Actor:** The User

**Stakeholders and Interests:**

The User: Loses the game.

**Preconditions:**

The game must be started.

**Success Guarantee (Postconditions): -Main**

**Success Scenario:**

1. The user logs in.
2. The user starts the game.
3. The user loses all of his/her chances.
4. The user loses the game.
5. The game is over.

**Extensions (or Alternative Flows):**

*a. At any time, the user can close the game.

*b. At any time, the connection may fail.

1a. The user enters wrong account information.

1. System signals error and rejects entry.
2. The user responds to the error.

    2a. The user tries to enter the credentials again.

    2b. The user creates an account.

2a. The user starts an already existing game.

1. The user resumes the game. 2b. The

user designs a new game.

1. The user chooses the building mode.
2. S/he designs the game according to her/his preferences.
3. The user starts the game.

3a. The user can obtain chances from the magical boxes.

1. Extra chance is added to the current chances of the user.

4a. The user starts an already existing game.

1. He/she chooses the game he/she wants to play among the predesigned games.

4b. The user enters the building mode.

1. He/she designs a new game.
2. He/she starts playing the game he/she has just designed.

**Special Requirements:** -

**Technology and Data Variations List:** -

**Frequency of Occurrence:** When the user loses all of his/her chances.

**Open Issues:** Is there any upper limit for chances?

## Use Case 5: Face “Infinite Void” Magical Ability

**Scope:** Need for Spear, Game

**Primary Actor:** The User

**Stakeholders and Interests:**

The User: Wants to face the “Infinite Void” Magical Ability

**Preconditions:**

The user has started the game.

**Success Guarantee (Postconditions):**

The randomly chosen magical power of Ymir is Infinite Void.

**Main Success Scenario:**

1. The user logs in.
2. The user starts playing the game.
3. “Infinite Void” magical ability pops up
4. 8 objects are selected randomly and frozen

**Extensions (or Alternative Flows):**

*a. At any time, the user can close the game.

*b. At any time, the connection may fail.

1a. The user enters the wrong account information.

1. System signals error and rejects entry.
2. The user responds to the error.

    2a. The user tries to enter the credentials again.

    2b. The user creates an account.

2a. The user starts an already existing game.

1. The user resumes the game. 2b. The user designs a new game.

1. The user chooses the building mode.
2. S/he designs the game according to her/his preferences.
3. The user starts the game.

2c. The user wins the game before the magical ability pops up.

1. The game is over.

2d. The user loses the game before the magical ability pops up.

1. The game is over.

4a. If there are less than 8 objects in the game

1. All of the objects are selected and frozen

**Special Requirements:** -

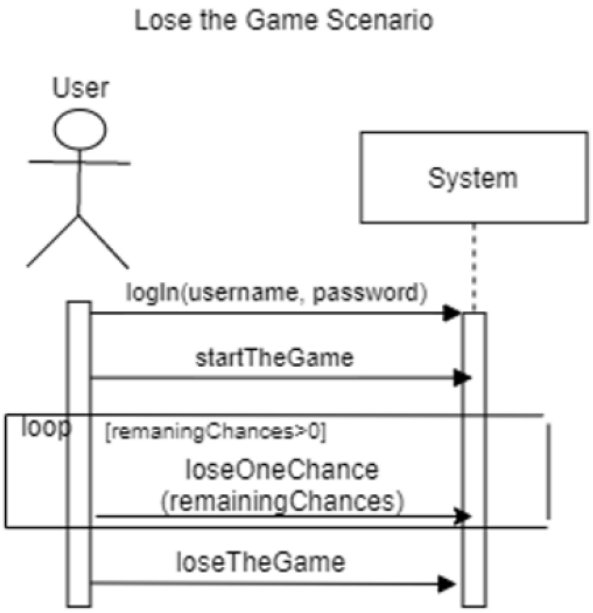
**Technology and Data Variations List:**

A database is required to store the user information

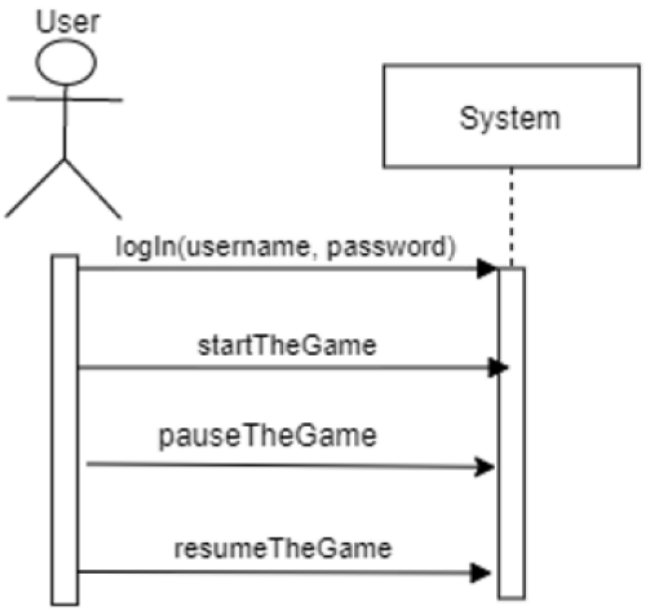
**Frequency of Occurrence:** Random

**Open Issues:** -

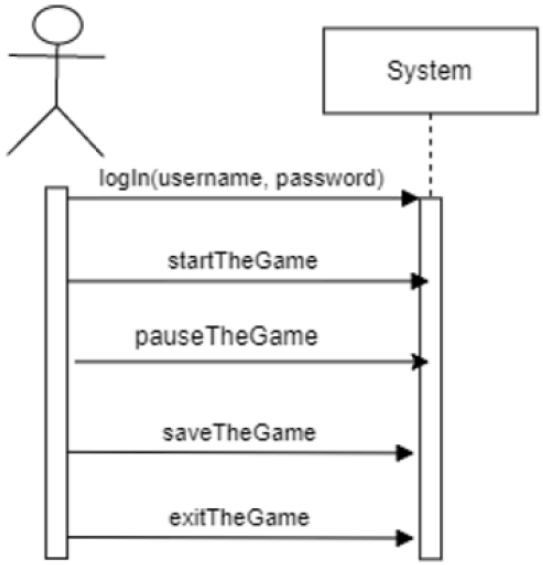
# System Sequence Diagram



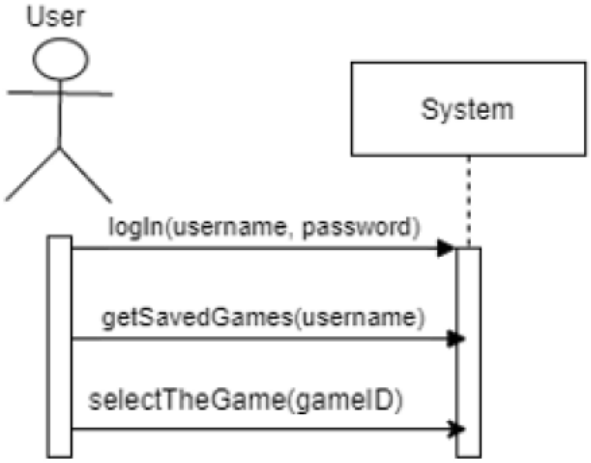
Resume the Game Scenario



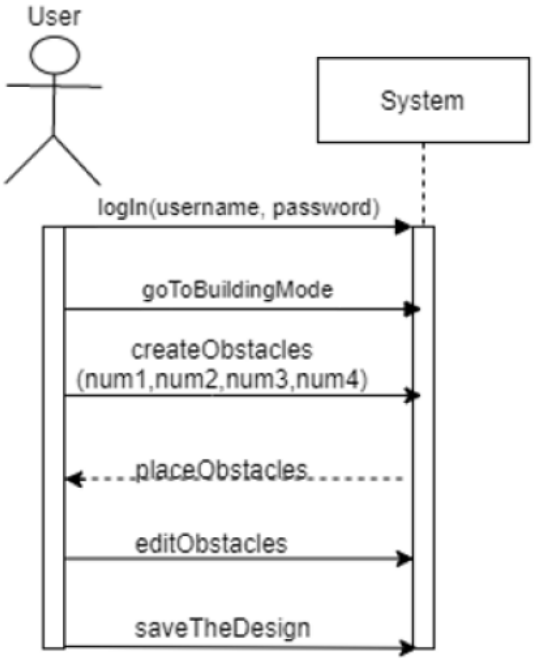

Save and Exit the Game Scenario



Load a Game Scenario




Design and Save a Game Scenario



# Operation Contracts

**Contract C01: saveTheGame**

| | |
|---|---|
| **Operation:** | **saveTheGame()** |
| **Cross References:** | **Use Cases: Save and Exit the Game Running Mode.isActive is false.** |
| **Preconditions:** | |
| **Postconditions:** | **— Attributes of the Gift Of Uranus instances were saved.**<br>**— Attributes of the Pandora's Box instances were saved.**<br>**— Attributes of the Steins Gate instances were saved.**<br>**— Attributes of the Wall Maria instances were saved.**<br>**— Attributes of the Noble Phantasm instances were saved.**<br>**— Attributes of the Enhanced Sphere were saved. — Attributes of the Game Description instances**<br>**were saved.** |

**Contract C02: editObstacles**

| | |
|---|---|
| **Operation:** | **editObstacles()** |
| **Cross References:** | **Use Cases: Design and Save a Game, Design and Start Game** |
| **Preconditions:** | **There are obstacles edited by the current player.** |
| **Postconditions:** | **— xPos, yPos attributes of the Gift Of Uranus instances were updated.**<br>**— xPos, yPos attributes of the Pandora's Box instances were updated.**<br>**— xPos, yPos attributes of the Steins Gate instances were updated.**<br>**— xPos, yPos attributes of the Wall Maria instances were updated.** |

**Contract C03: loseOneChance**

| | |
|---|---|
| **Operation:** | **loseOneChance( remainingChances: integer)** |
| **Cross References:** | **Use Cases: Losing The Game** |
| **Preconditions:** | **remainingChances is greater than zero and Running Mode.isActive is true.** |
| **Postconditions:** | **— remainingChances was decreased by 1(attribute modification).** |

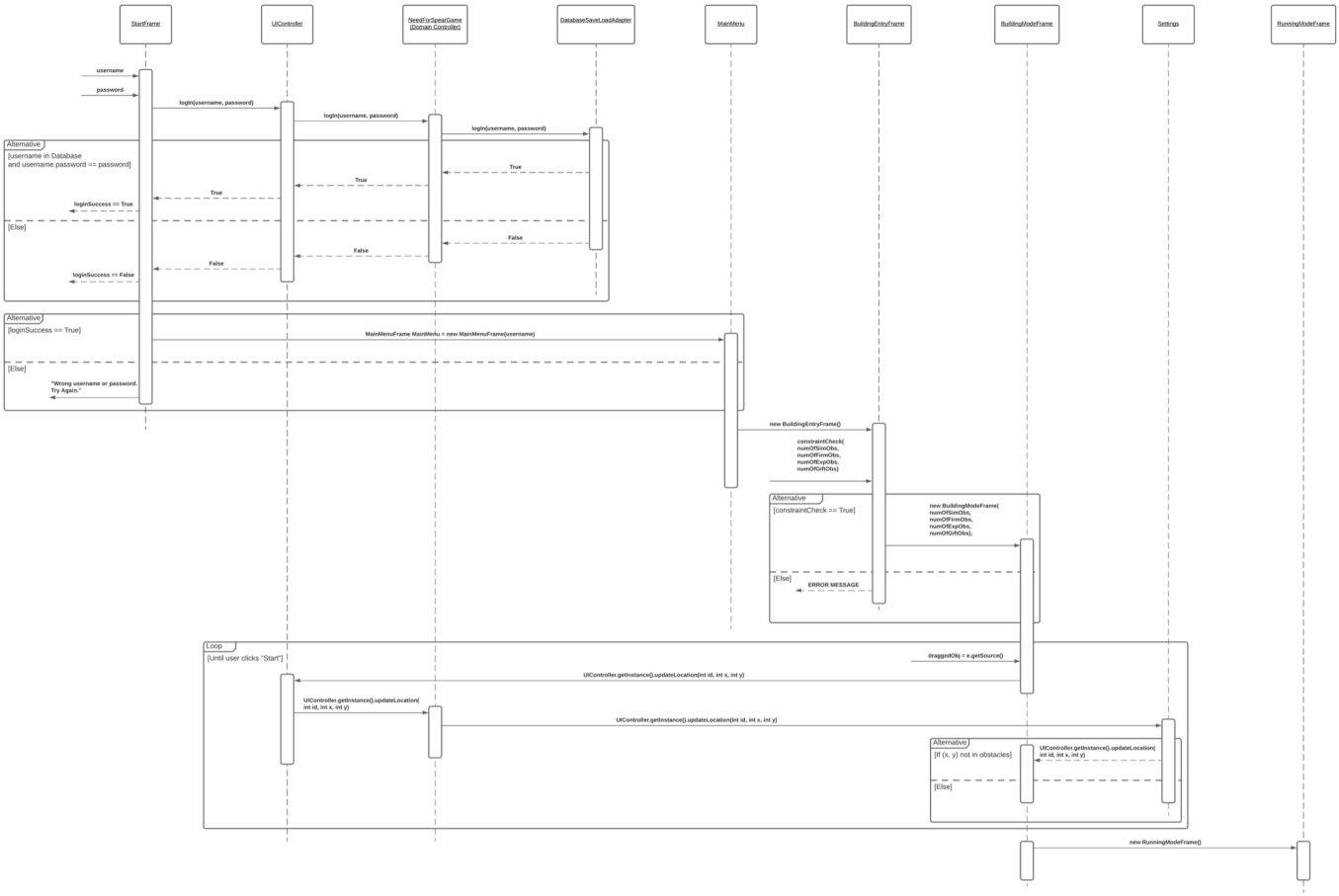
**Contract CO4: SelectRandomMagicalAbility**

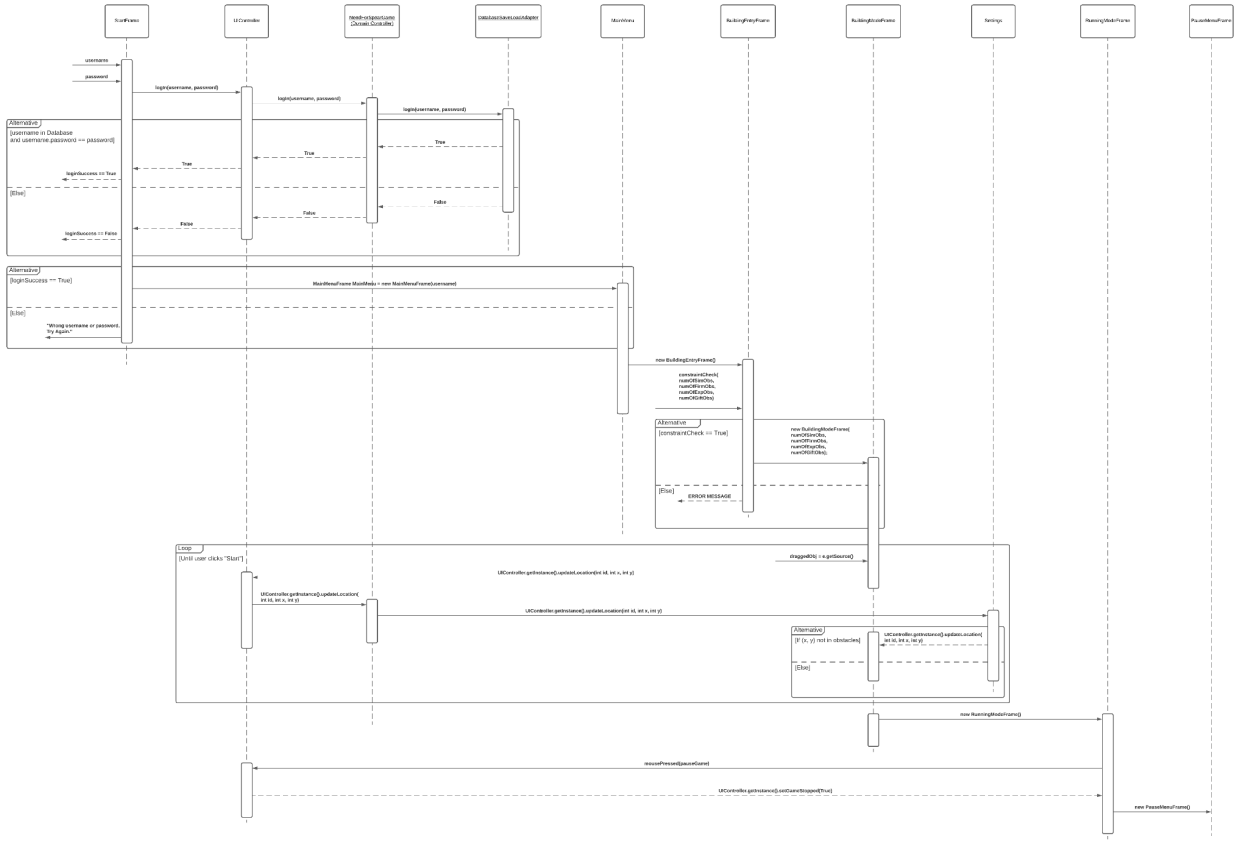
| | |
|---|---|
| **Operation:** | **SelectRandomMagicalAbility()** |
| **Cross References:** | **Use Cases: Design and Start the Game, Load a Game, Resume the Game** |
| **Preconditions:** | **Coin flip is successful.** |
| **Postconditions:** | **— Ymir made his choice.**<br>**— A Hollow Purple, Infinite Void or Double Accel instance was created according to Ymir's choice.(instance creation).** |

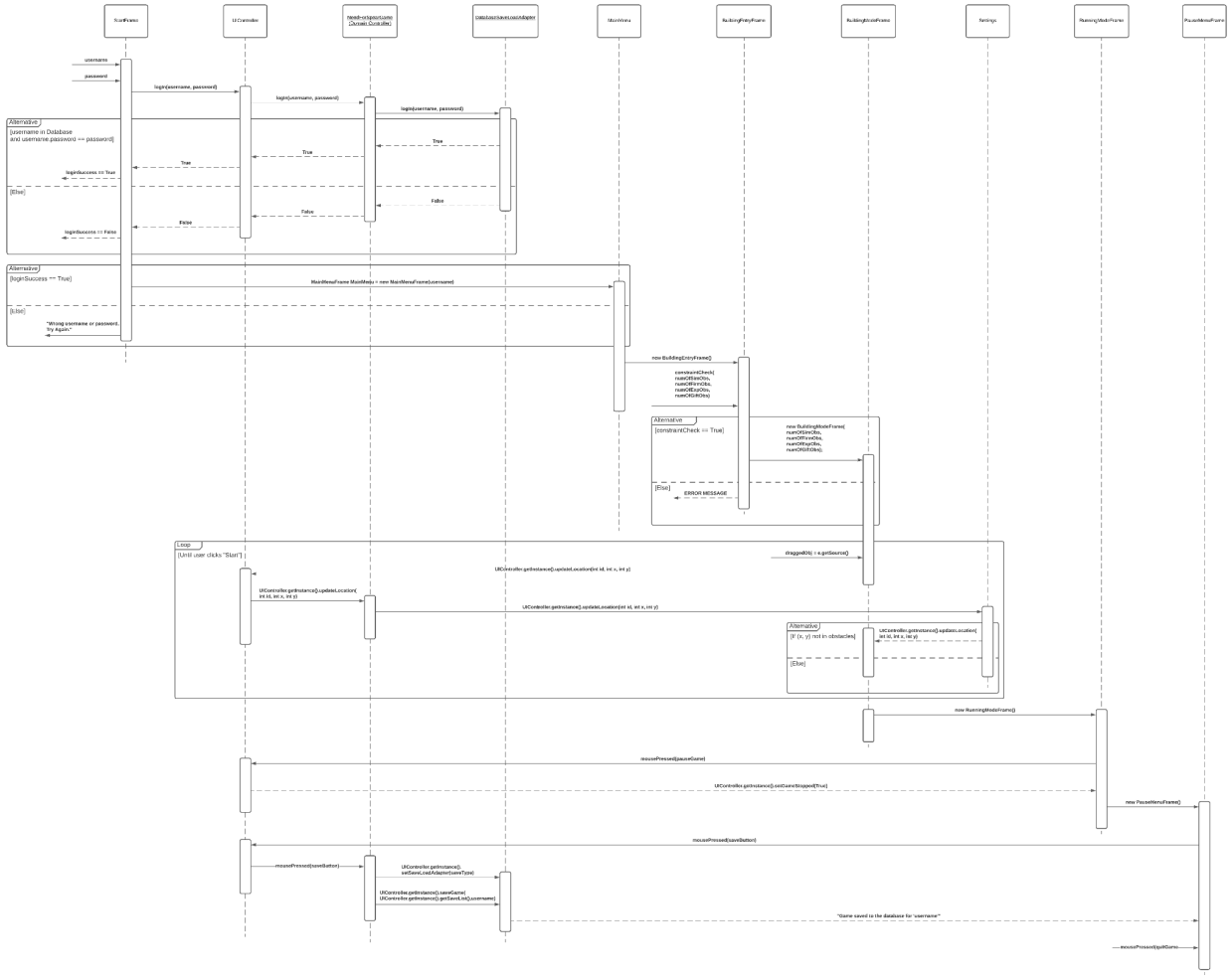
**Contract CO5: freezeObstacles**

| | |
|---|---|
| **Operation:** | **freezeObstacles()** |
| **Cross References:** | **Use Cases: Face “Infinite Void” Magical Ability** |
| **Preconditions:** | **Coin flip is successful.**<br><br>**Ymir chooses Infinite Void Ability.** |
| **Postconditions:** | **— 8 Obstacles are randomly selected.**<br><br>**— Obstacle.isInfiniteVoidActive became true.** |

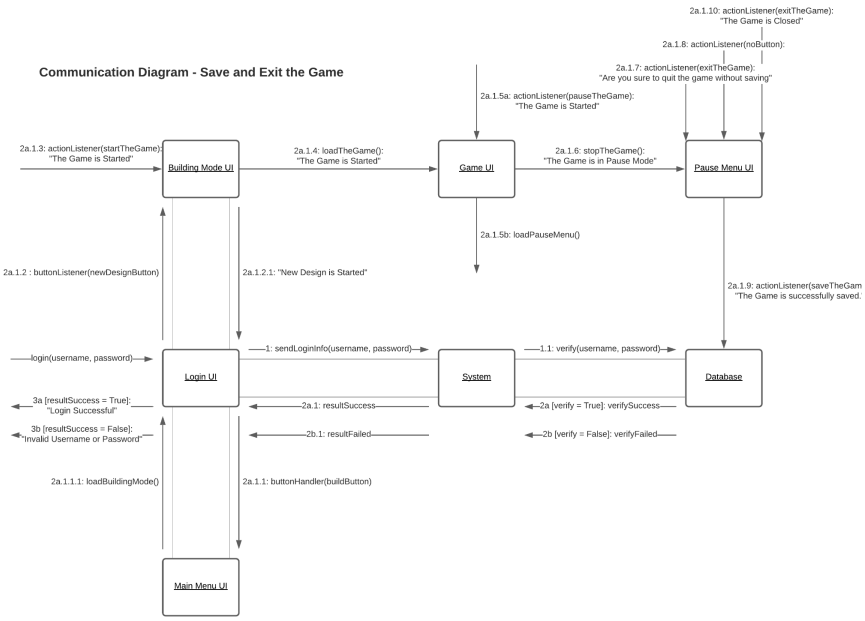
# Sequence Diagrams

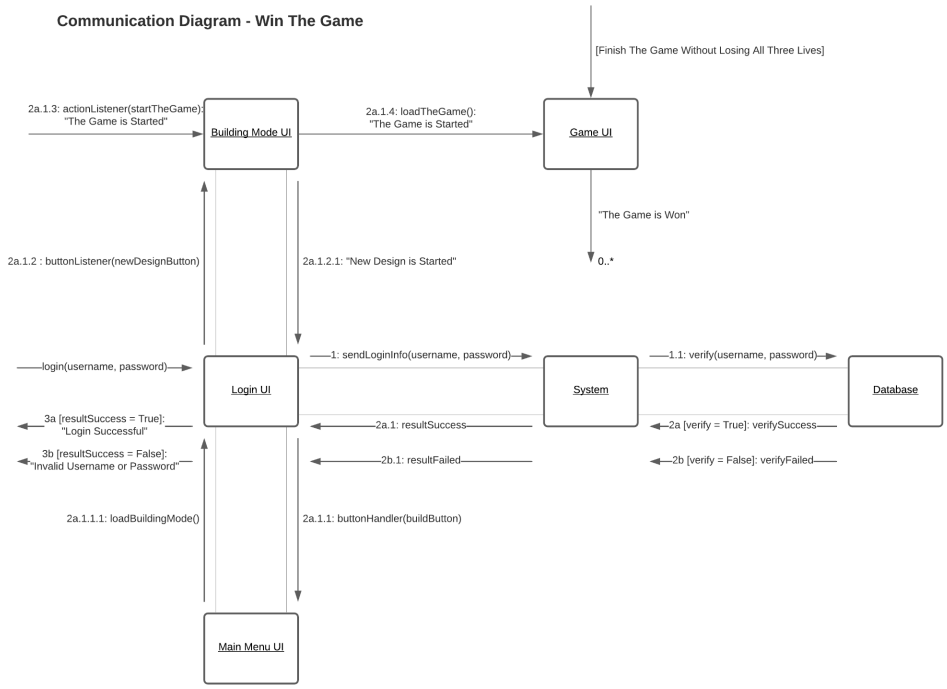



StartMenu
UI Controller
NeedHelpSupportUser (Support Controller)
DataUserAssetLoadAdapter
MainMenu
BuildingEntryFrame
BuildingModeFrame
Settings
RunningModeFrame
PauseMenuFrame
username
password
login(username, password)
login(username, password)
login(username, password)
Alternative
[username in Database
and username.password == password]
True
True
True
loginSuccess == True
[Else]
False
False
False
loginSuccess == False
Alternative
[loginSuccess == True]
MainMenuFrame MainMenu = new MainMenuFrame(username)
[Else]
"Wrong username or password,
try Again."
new BuildingEntryFrame()
constraintCheck(
numOfBeeDos,
numOfFirmDBs,
numOfRepDBs,
numOfGRIDIos)
Alternative
[constraintCheck == True]
new BuildingModeFrame(
numOfBeeDos,
numOfFirmDBs,
numOfRepDBs,
numOfGRIDIos);
[Else]
ERROR MESSAGE
Loop
[Until user clicks "Start"]
draggedIOs = e.getSource()
UIController.getInstance().updateLocation(int id, int x, int y)
UIController.getInstance().updateLocation(
int id, int x, int y)
UIController.getInstance().updateLocation(int id, int x, int y)
Alternative
[If (x, y) not in obstacles]
UIController.getInstance().updateLocation(
int id, int x, int y)
[Else]
new RunningModeFrame()
mousePressed(pauseGame)
UIController.getInstance().setGameStopped(True)
new PauseMenuFrame()

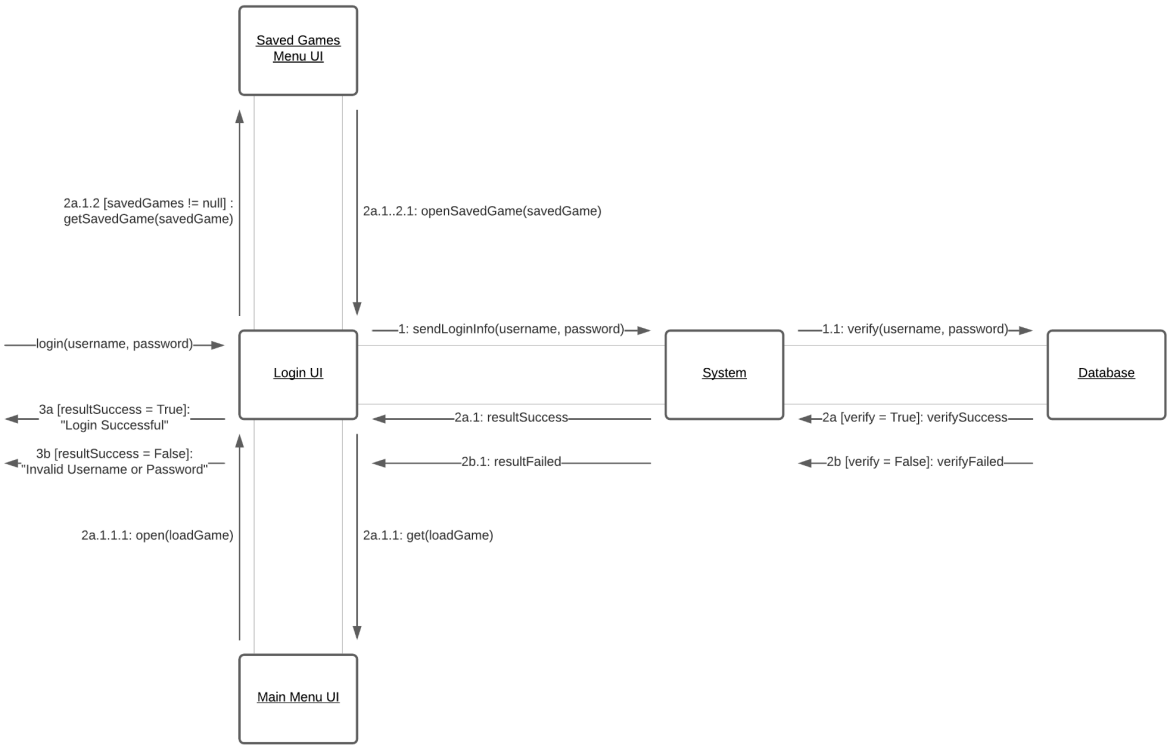

StartMenu
UI Controller
MainMenu
RuckingPrintyFrame
RuckingMainFrame
Settings
RunningModeFrame
PauseMenuFrame
username
password
login(username, password)
Alternative
[username in Database
and username.password == password]
True
loginSuccess == True
[Else]
False
loginSuccess == False
[loginSuccess == True]
MainMenuFrame MainMenu = new MainMenuFrame(username)
"Wrong username or password,
try Again."
new BuildingEntryFrame()
constraintCheck(
numOfBeeNrs,
numOfFirmOBs,
numOfKeyOBs,
numOfAGVBOts)
[constraintCheck == True]
new BuildingModeFrame(
numOfBeeNrs,
numOfFirmOBs,
numOfKeyOBs,
numOfAGVBOts);
ERROR MESSAGE
Loop
[Until user clicks "Start"]
draggedObj = e.getSource()
UIController.getInstance().updateLocation(int id, int x, int y)
[IF (x, y) not in obstacles]
new RunningModeFrame()
mousePressed(pauseGame)
UIController.getInstance().setGameStopped(True)
new PauseMenuFrame()
mousePressed(saveButton)
mousePressed(saveButton)
UIController.getInstance().
setSaveLoadAdapter(saveType)
UIController.getInstance().saveGame
UIController.getInstance().getSaveLink(username)
"Game saved to the database for 'username'"
mousePressed(quitGame)

# Communication Diagrams

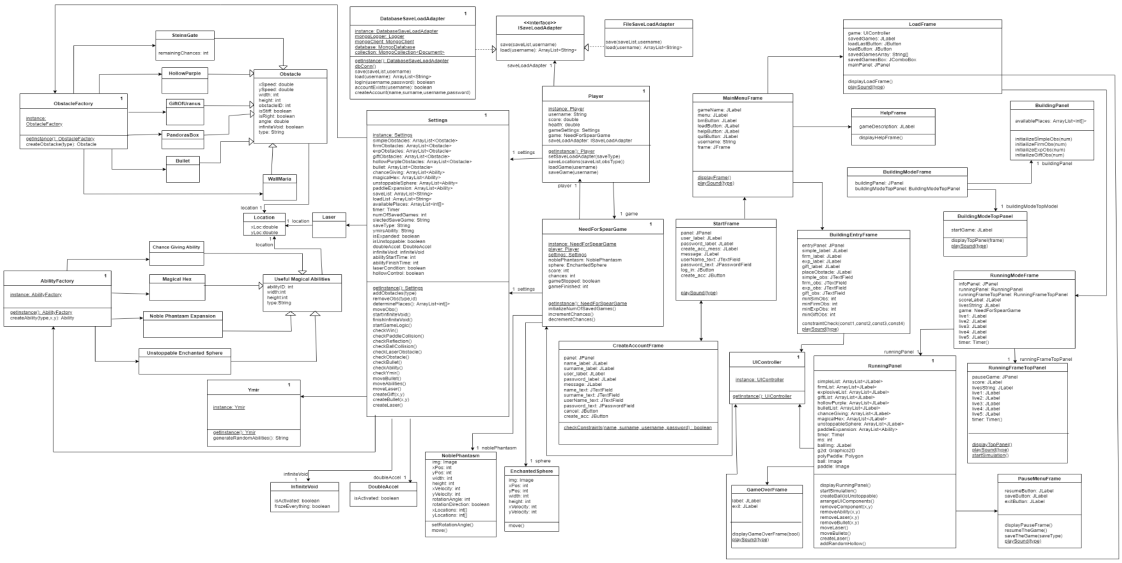


**Communication Diagram - Load a Game**

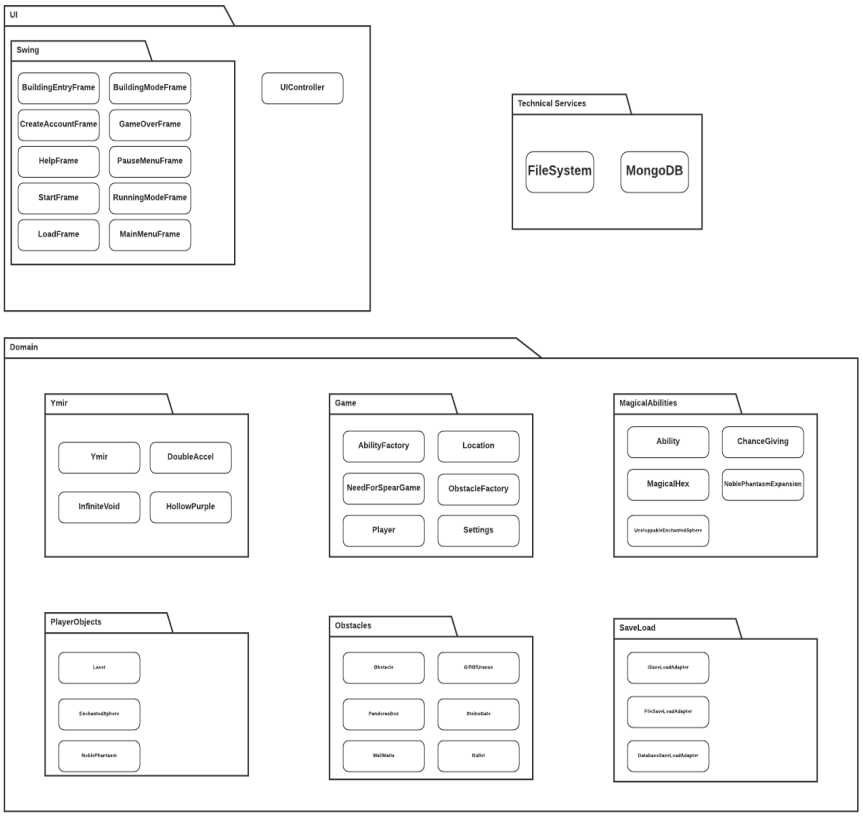


# UML Class Diagram



# Package diagram

UML package diagram



# Design Patterns

**Adapter:** We apply Adapter pattern while implementing Save and Load part.If a player voluntarily saves the game he played, this game is saved in the database. If the game exits without saving, it is saved to the file and when a game is loaded, two options are presented: loading the last game played or loading one of the saved games.With the Adapter Model, the signatures, post-conditions, and prerequisites of both the save-to-file and database method were the same.

**Singleton:** Singleton was one of the patterns that we used the most and worked for us the most. Without the Singleton pattern we would need to pass the instance of those classes as parameters each time we needed them. We used it in our domain and ui controllers, and also in ability factory and obstacle factory classes that we created with the factory pattern.

**Factory:** We used the Factory pattern to create obstacles, abilities, and Ymir's hollow purple which was named as AbilityFactory and ObstacleFactory. It helped us to create game object instances in an efficient way.

**Controller:** In our game, we used two controllers such as NeedForSpearGame which is the domain controller and UIController which is the UI controller to ensure the flow of information between UI layers and Domain layers.Controller pattern allows us to delegate the inputs from the UI to the various classes in the domain layer without violating Model-View Separation.We cau it is the main pattern of our game.

**Creator**: Creator guides the assigning of responsibilities related to the creation of objects, we should find creators that also support low coupling. According to our design, the BuildingMode class should be the creator of all types of obstacles, Noble Phantasm, and Enchanted Sphere. BuildingMode has the

initializing data for these objects, so BuildingMode is an Expert with respect to creating these objects. As we have learned, Expert supports Low Coupling. Therefore, we also support Low Coupling by choosing BuildingMode as creator. Gift Of Uranus should be the creator for GiftBox, because when explode method is called, dropGiftBox method will be called which closely uses GiftBox.

**Information Expert:** In our design, to further reduce coupling, we will be using the Information Expert patterns. For example, the RunningMode class can be an expert in the sense that it will be the only class in our design that knows the current time. We can reach the current time with building mode and use this time to calculate the total score in the game description class.Other classes will not have access to the current time directly. We can also have more than one information expert.

**High Cohesion & Low Coupling:** Also, while implementing the code, we take the advantage of low coupling and high cohesion. In our design, high cohesion increases the reusability since the components with the closest relationships stick together. By doing so, we decrease the complexity. Because of the fact that the aim of the low coupling is to decrease the interconnections among modules, we can easily make internal changes in a specific module. Lastly, when we successfully bring together the low coupling and high cohesion, it will definitely increase the readability and sustainability of the project.

**Polymorphism:** It was a big project which consist of same type of game objects therefore we used Polymorphism pattern a lot during the project.It was  really beneficial to use polymorphism while using obstacles and abilities.We always tried to use interfaces or abstract classes in our project to prevent code repetitions.

# Supplementary Specifications

**Introduction**

This document is the repository of all “Need For Spear” Game, the Comp302 Project, requirements not captured in the use cases.

**Functionality**

Storing account username and password information in the database. Handling login process, validating the entered username and password from the database.All inputs from the user are stored in the database, and processed in order to respond to the user as system responses.

**Usability**

The user(player) will be able to see the game from their computer screen. Therefore:

- The text and graphics should be easily visible from 0.5 meters.
- The motions of game objects should be slow enough for the user to respond.
- The buttons and other game mechanics should be accessible on the game screen.
- Colors and game objects should not tire the user’s eye

**Reliability**

The saved games should be stored properly, so that the user does not face a problem, such as not being able to continue their %95 completed game.

The system errors should be solved with minimum damages, so that small errors do not cause the whole game to crash and result in a lost game.

All game-related operations should be done after the user authentication. There should be no security problems that cause a difference in user's progress in the game.

**Performance**

The response time of the system to user actions should be minimum in order to eliminate any bad consequences that may occur because of late response from the system.

The login process should be as fast as possible. The system-database connection should be as fast as possible to eliminate the waiting time for authentication.

**Supportability**

The game should be adaptable to different controls. In any case of a different keyboard, the user should be able to manually change the controls or the game controls should be automatically adjusted to different types of keyboards.

The game CPU requirements should be adjustable in the game. In any case of a computer not providing enough CPU for the game to run properly, the game requirements (such as graphics) should be adjustable in order for the game to run properly and without lagging.

**Implementation Constraints**

The "Need For Spear" game must be developed in Java language. Since this is a class project, we must abide by the coding language selection by the course instructor.

The project must be planned, structured, and implemented according to the course layout and the book "Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and Iterative Development" by Craig Larman.

# Glossary

## Game-Specific Terms and Names:

**Need For Spear:** Name of the game

**Spear of Power:** The objective of the game. Two brave warriors are racing to obtain the Spear of Power that will enable its owner to rule the world given its powerful capabilities.

**Shazam:** The legendary wise magician, creator of the Spear of Power and the obstacles that protect the Spear.

**Enchanted Sphere:** The magical ball that is sent around to destroy obstacles, but it is affected by gravity, therefore the noble phantasm (paddle) is used to set its track towards the target obstacles to destroy them.

**Noble Phantasm:** The player-controlled, paddle-like object that is used to deflect the enchanted sphere (ball) from falling to the ground

## Obstacles

**Wall Maria (Simple Obstacle):** Can be broken in one hit. When broken, it disappears

**Steins Gate (Firm Obstacle):** These obstacles are more difficult to destroy. Each one contains a number written on it, which corresponds to the number of hits it requires to be destroyed. After every hit it receives, the number decreases by 1, and the obstacle disappears once the number reaches zero.

**Pandora's Box (Explosive Obstacle):** This obstacle has a circular shape and it explodes once it is hit. Once exploded, its remains fall downwards towards the noble phantasm. If the remains touch the noble phantasm, the player loses a chance.

**Gift of Uranus (Gift Obstacle):** This obstacle can be destroyed in one hit like the simple one. Once destroyed, it drops a box downwards towards the noble phantasm. If the noble phantasm touches the box,

then the box opens and rewards the warrior with a magical ability that can be either used to support the warrior or to create more challenges and obstacles for the other player.

## Magical Abilities

**Chance Giving Ability:** This ability increases the player's chances by 1.

**Noble Phantasm Expansion:** This ability doubles the length of the noble phantasm. It is not necessarily activated once it is received. The player can choose to activate it whenever they want by either pressing the button T or pressing its icon on the screen. Once activated, it lasts for only 30 seconds, after which the noble phantasm returns to its original state.

**Magical Hex:** This ability equips the noble phantasm with two magical canons on both of its ends. The canons should point upwards and they rotate as the noble phantasm rotates. They can fire magical hexes that can hit the obstacles. A hex hit has the same effect as the hit of an enchanted sphere. It does not activate immediately, but can be activated by pressing H or pressing its icon on the screen. Once activated it remains active for only 30 seconds and then disappears afterward.

**Unstoppable Enchanted Sphere:** This ability upgrades the enchanted sphere and makes it much more powerful, such that if it hits any obstacles, it destroys it and passes through it regardless of its type (even for the firm obstacles). This upgrade only lasts 30 seconds after it is activated.

**Building Mode:** The mode which is opened before every game. In this mode, the user can either design a new game or directly start an already designed game.

**Running Mode:** The main mode, which is for playing the game.

## Ymir

**Infinite Void:**The 8 obstacles selected from the screen become indestructible for 15 seconds.

**Double Accel:** This ability halves the enchanted sphere's speed for 15 seconds

**Hollow Purple:** Can be broken in one hit.When broken, it disappears but does not affect score.