COMP-302 TERM PROJECT
FINAL REPORT

Group Name: Brogrammers
Date: 12.01.2022

Group Members:

- İsmail Ozan Kayacan
- Barış Kaplan
- Ege Seçilmiş
- Lütfü Mustafa Kemal Ato
- Hikmet Demirel

Table Of Contents

Vision

Introduction:

Need For Spear is an easy to play and enjoyable game targeting everyone. It is based on breaking blocks and gathering points. It interacts with the user by buttons on the display. There are some tricks like superpowers (gained by the user, enchanted sphere, and noble phantasm), and the obstacles in the game which make it more fun.

Product Positioning:

The game might be available on the application stores and take place in the game category. It comes forward with its basic and simple design but also various in-game scenarios by far compared to other similar games in the market. It targets all ages but especially children to ensure an enjoyable time.

Stakeholder Summary:

All the COMP302 staff including our instructor, TAs, and the other people who will play this game will be the stakeholders.

Problem Statement:

In today's world, coronavirus is obliging us to stay home and keep our distance from the outside world so people are able to spend more time with their phones and laptops. In such a case, our game can be preferable for people to have fun in these hard times.

Product overview:

The game is designed to give people a break from a boring moment. It works by the movement of a spear which is controlled by users so it is quite easy to learn and play. Overall, the Need for Spear will be a fun game and available on all digital platforms to download.

Teamwork

Building Mode: Obstacle dragging, removing, adding by mouse. Making system add obstacles: Barış
Transferring built game to running game panel: İsmail
Making the obstacles do not collide with each other: Hikmet and Barış and Lütfü

Noble Phantasm: Rotation does not work. Phantasm moving: Lütfü and İsmail.

Collisions

Wall - Enchanted Sphere Collision: Mostly Ege and group work
Obstacle - Enchanted Sphere Collision: Group work
Remaining & Gift Box - Noble Phantasm Collision: İsmail
Enchanted Sphere - Noble Phantasm Collision: Group work

Magical Abilities: Group work

Ymir Abilities: Group work

Moving Obstacles: İsmail

Pause/Resume: İsmail and Barış

Save/Load: İsmail

Scoreboard Update (score and lives): Group work

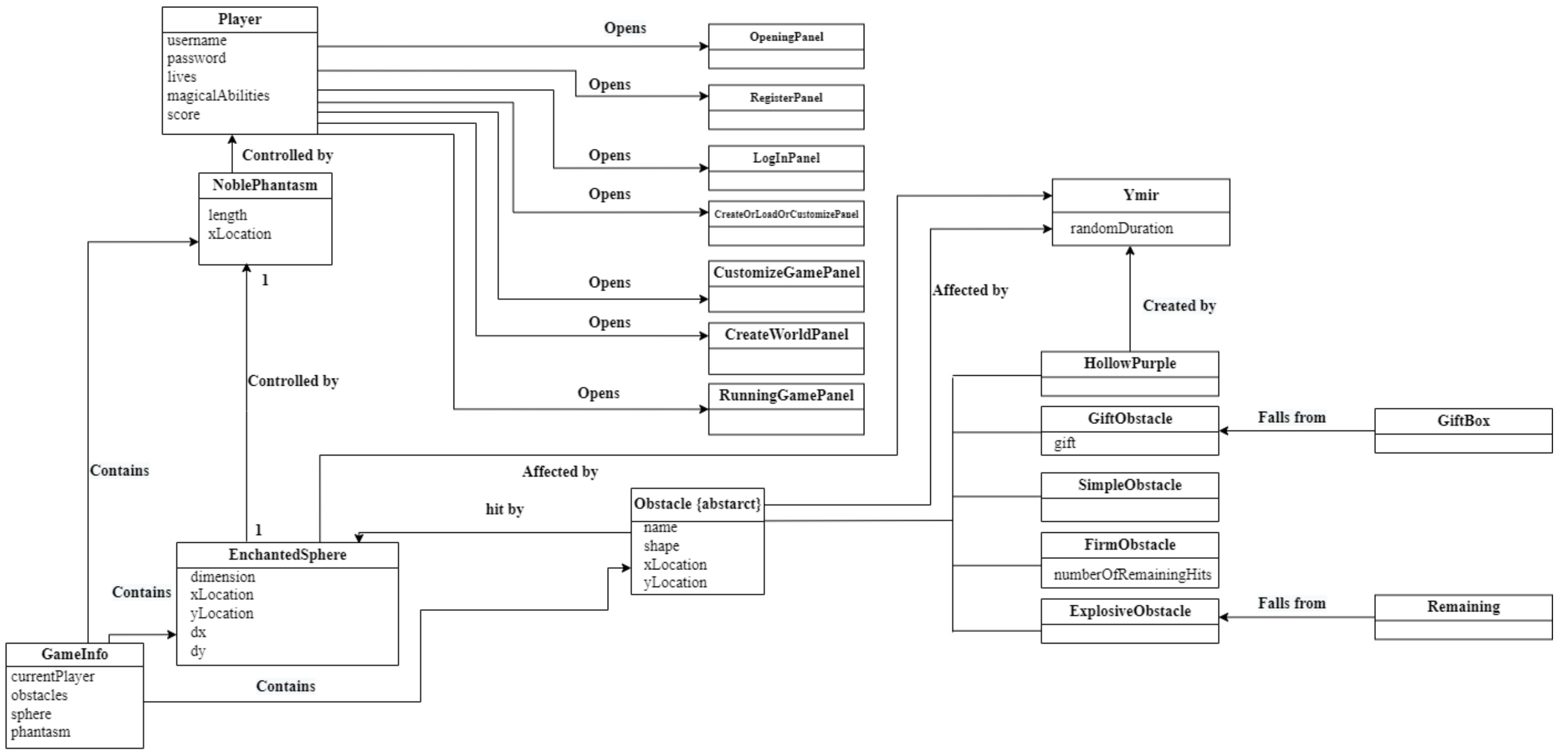
Controller: Mostly İsmail, group work

Other Patterns
Observer: İsmail and Barış
Adapter: Mostly Lütfü and Barış
Factory & Singleton: Mostly İsmail, group work

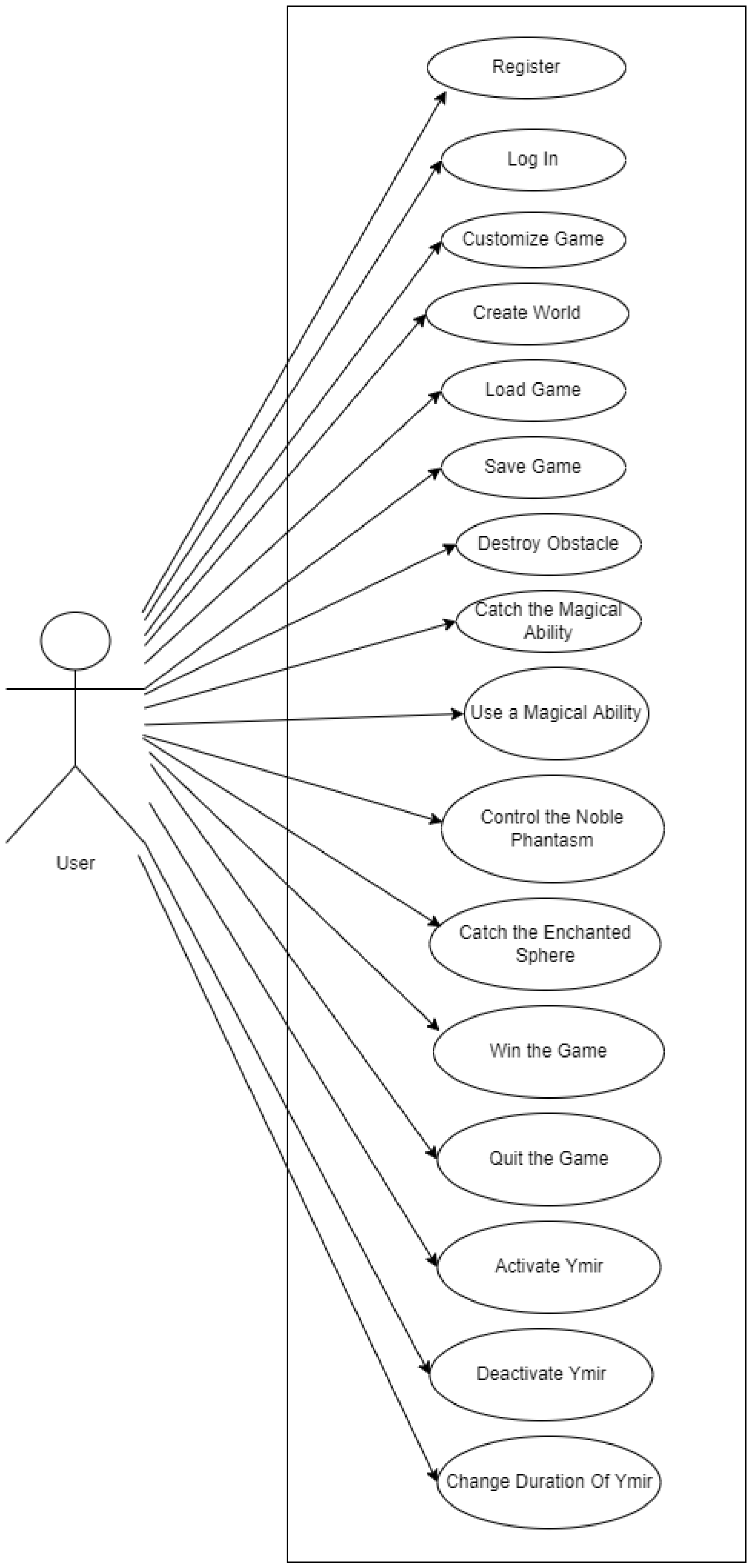
Model View Separation: Mostly İsmail, group work.

Other extra features (Like customize game, login, register, etc.): Customize game use case Barış and Hikmet, login use case mostly Ege and Barış, register use case mostly İsmail and Lütfü. Other extra features that are not listed above include group work.

Domain Model



Use Case Diagram



Use Cases

USE CASE 1: Register

Actor: User

Pre Condition: User successfully opened the game.
Post Condition: User is successfully directed to the main opening page.

Happy Scenario:

1. System asks the user to provide a username and password.
2. User types a username and password and submits.
3. System adds the username and password to the txt file.
4. System opens the main page.

Extensions (or Alternative Flows):

2a. User provides missing information:
    1. System displays "Please provide username and password to register to the game." and stays in the register page.

3a. Username exists in the txt file:
    1. System displays "Username already taken. Please enter another username." and stays on the register page.

USE CASE 2: Create World

Actor: User

Pre Condition: User successfully logged in.
Post Condition: Running mode is opened with the new game environment.

Happy Scenario:

1. User enters the number of each obstacle he/she wants.
2. User presses the "Create" button.
3. The system puts obstacles to arbitrary positions.
4. World is created. The user can click "Go to Game" and start playing.

Extensions (or Alternative Flows):

*a. User selects an obstacle type and clicks a point in the world:
    1a. There is an obstacle on that point:
        1. System displays "You can't put an obstacle here, there is an obstacle.".

2a. There is no obstacle on that point:
1. System puts an obstacle here.
*b. User clicks on obstacle and drags it to some point:
1a. There is an obstacle on that point:
1. System displays “Oops, you cannot move the obstacle there!”
2. System puts the obstacle on its previous location.
2a. There is no obstacle on that point:
1. System moves the obstacle here.
*c. User clicks on an obstacle in the world:
1. System removes that obstacle.
2a. One of the obstacle numbers is not adequately specified by the user:
1. System displays “To create a world there must be at least 75 simple obstacles, 5 explosive obstacles, 10 gift obstacles, 10 firm obstacles.".
2b. System cannot put obstacle to the world since there is not enough space in the editing area for the user:
1. System displays “You can add maximum 200 obstacles in total.”.

USE CASE 3: Load Game

Actor: User

Pre Condition: User successfully logged in.
Post Condition: Running mode is opened with the saved game.

Happy Scenario:

1. User presses the “Load” button.
2. System displays the games which were saved by the current user.
3. User selects one of the saved games.
4. System pulls the information of the selected saved game from the txt file.
5. System creates the world by the data it got from the txt file.

Extensions (or Alternative Flows):

2a. User does not have any saved game:
1. System displays “You do not have a saved game.”

USE CASE 4: Customize Game

Actor: User

Pre Condition: User logged in.
Post Condition: Game is customized according to the user’s choices.

Happy Scenario:

1. User selects the customize game menu’s button.
2. System opens customize game menu.
3. System shows some options: brightness level, sound level, play music, stop music, background colors, easy game mode, hard game mode, and ymir.
4. User selects a desired option.

Extensions (or Alternative Flows):

4a. User adjusts the sound:
  1. System increases or decreases the sound.

4b. User adjusts the brightness level:
  1. System increases or decreases the brightness.

4c. User selects a background color from the background color options:
  1. System changes background color to selected color.

4d. User clicks play music button:
  1. Music starts to play.

4e. User selects stop music button:
  1. Music stops.

4f. User selects the Ymir option:
  1. System activates or deactivates Ymir.

4g. User selects the hardness level:
  1. System changes the ymir activation time according to the user's choice.

<u>USE CASE 5:</u> Use a Magical Ability

Actor: User

Pre Condition: User has a magical ability in his/her packet.
Post Condition: User has one less magical ability in his/her packet.

Happy Scenario:

1. User selects an ability.

Extensions (or Alternative Flows):

1a. User selects Chance Giving Ability:
  1a. User does not have Chance Giving Ability:
    1. System displays “You don’t have Chance Giving Ability.”.
  1b. User has Chance Giving Ability:
    1. User’s chances are increased by 1.
    2. Number of Chance Giving Abilities that user have is decreased by 1.

1b. User selects Noble Phantasm Expansion by pressing the button T, or by pressing the Noble Phantasm Expansion icon on the screen:

- 1a. User does not have Noble Phantasm Expansion:
    - 1. System displays “You don’t have Noble Phantasm Expansion.”.
- 1b. User has Noble Phantasm Expansion:
    - 1. The length of the noble phantasm doubles for 30 seconds.
    - 2. Number of Noble Phantasm Expansions that user have is decreased by 1.

1c. User selects Magical Hex by pressing the button H, or by pressing the Magical Hex icon on the screen:

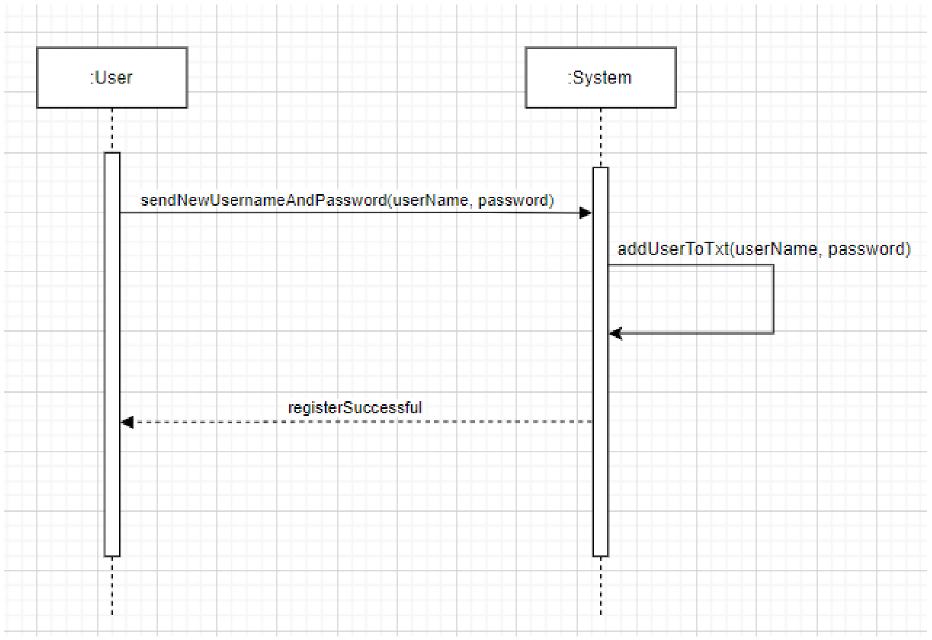
- 1a. User does not have Magical Hex:
    - 1. System displays “You don’t have Magical Hex.”.
- 1b. User has Magical Hex:
    - 1. Noble phantasm is equipped with two magical canons on both of its ends:which fires balls for 30 seconds whenever user clicks the space button.
    - 2. Number of Magical Hexes that user have is decreased by 1.

1d. User selects Unstoppable Enchanted Sphere:

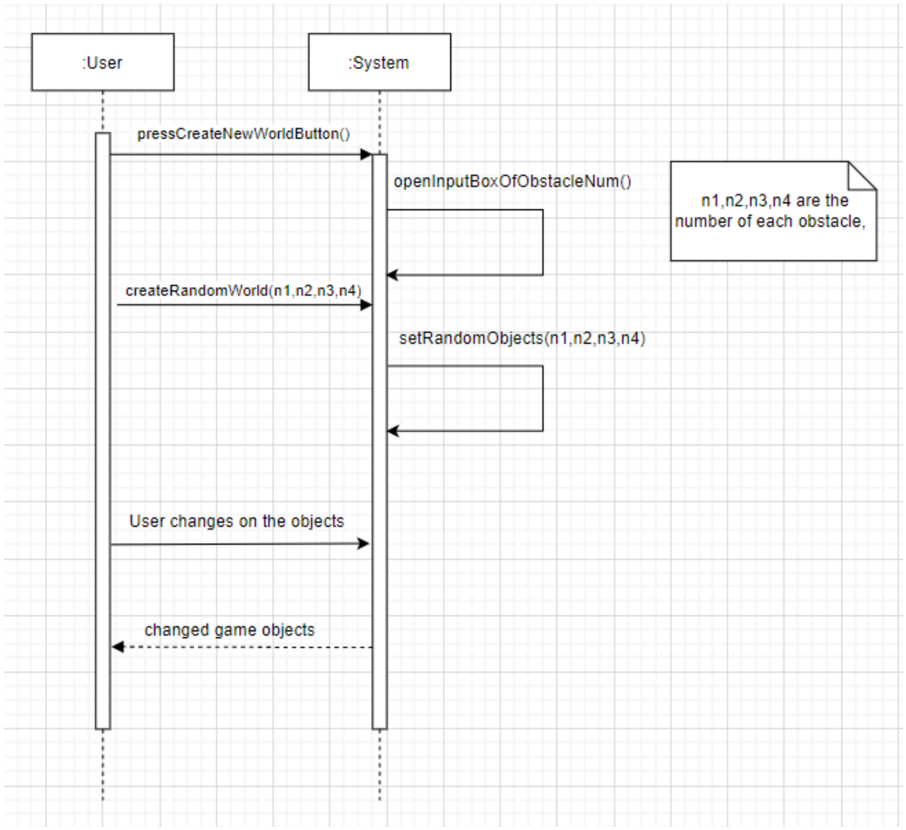
- 1a. User does not have Unstoppable Enchanted Sphere:
    - 1. System displays “You don’t have an Unstoppable Enchanted Sphere.”.
- 1b. User has Unstoppable Enchanted Sphere:
    - 1a. Infinite void was activated by Ymir:
        - 1. Enchanted sphere works with its usual power for frozen obstacles.
        - 2. Number of Unstoppable Enchanted Spheres that user have is decreased by 1.
    - 1b. Infinite void was not activated by Ymir:
        - 1. Enchanted sphere becomes much more powerful for 30 seconds.
        - 2. Number of Unstoppable Enchanted Spheres that user have is decreased by 1.

System Sequence Diagrams

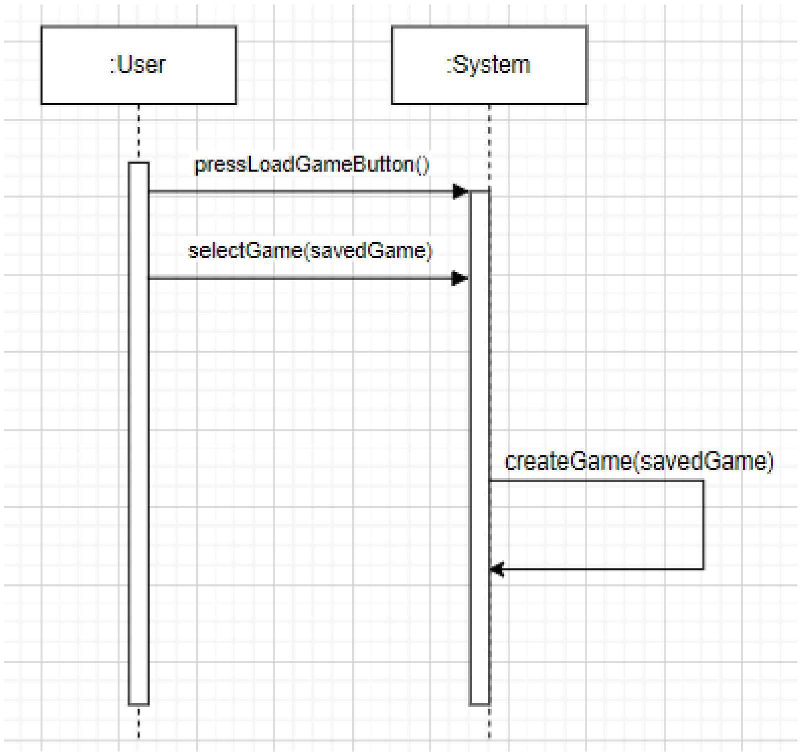
SSD-1 (Register Scenario):


:User
:System
sendNewUsernameAndPassword(userName, password)
addUserToTxt(userName, password)
registerSuccessful

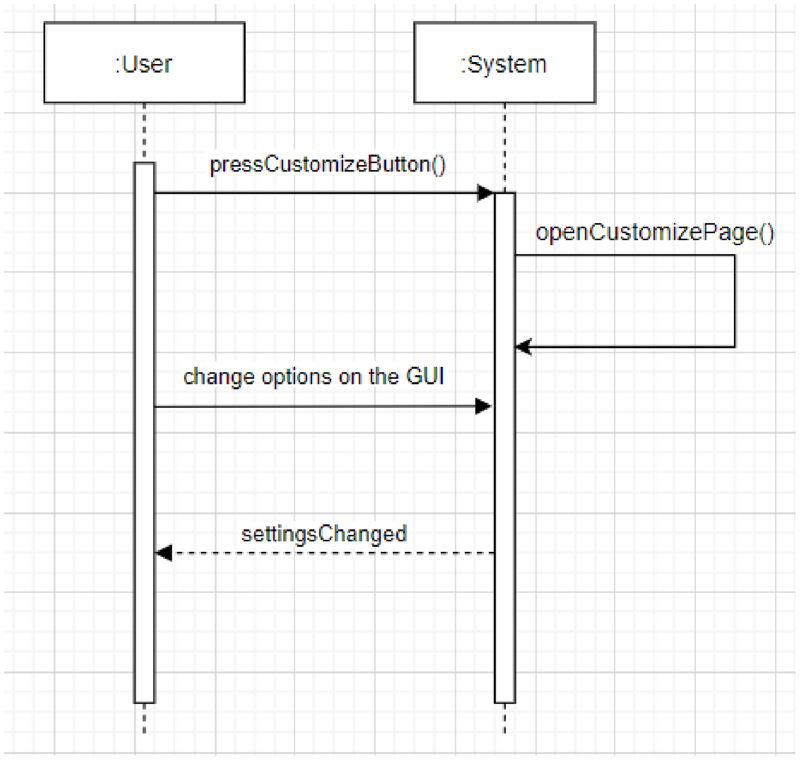
SSD-2 (Create World):


:User
:System
pressCreateNewWorldButton()
openInputBoxOfObstacleNum()
n1,n2,n3,n4 are the number of each obstacle,
createRandomWorld(n1,n2,n3,n4)
setRandomObjects(n1,n2,n3,n4)
User changes on the objects
changed game objects

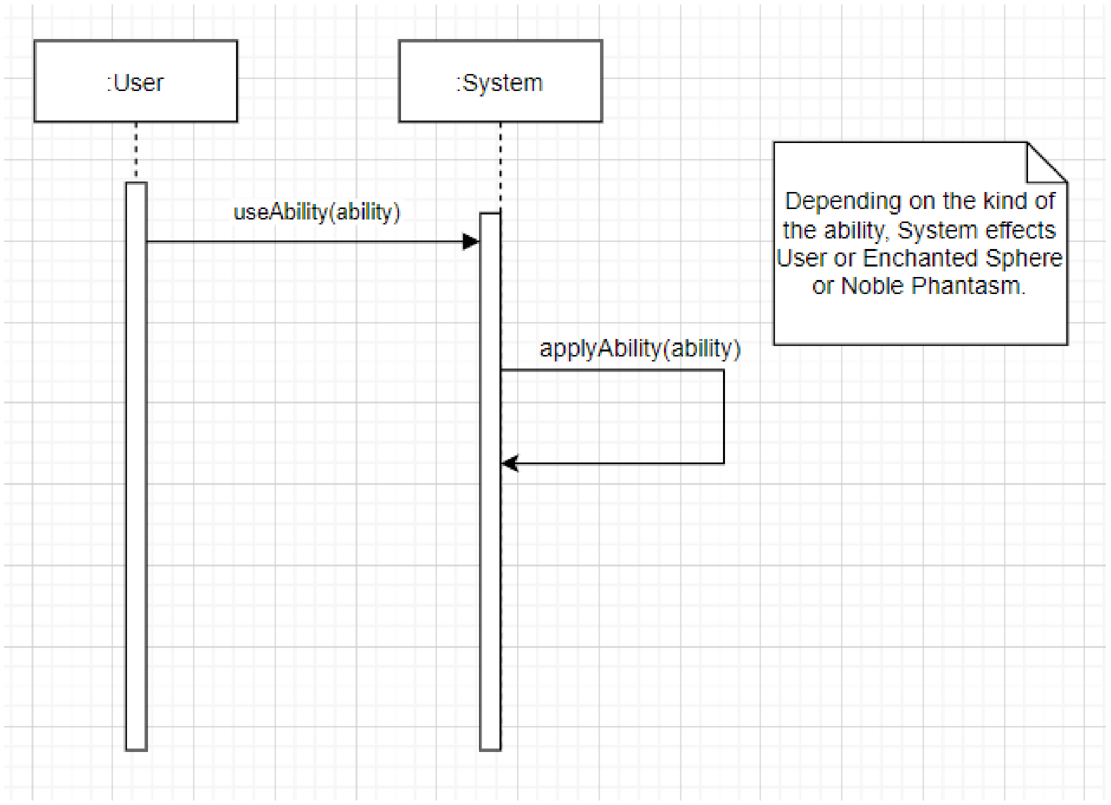
SSD-3 (Load Game):



SSD-4 (Customize Game):



SSD-5 (Use a Magical Ability):


:User
:System
useAbility(ability)
applyAbility(ability)
Depending on the kind of the ability, System effects User or Enchanted Sphere or Noble Phantasm.

Operation Contracts

Operation-1: pressRegisterButton(username, password)
Cross Reference: Use case 1 (Register)
Pre Condition: *RegisterPanel.isVisible()* was *true.*
Post Condition: *RegisterPanel.isVisible()* was changed to false. *CreateOrLoadOrCustomizePanel.isVisible()* was changed to *true. GameInfo.currentPlayer* was changed to *new Player(username).*

Operation-2: createRandomWorld(n1, n2, n3, n4)
Cross Reference: Use case 2 (Create World)
Pre Condition: *CreateWorldPanel.isVisible()* was *true.*
Post Condition: n1 *SimpleObstacle* objects were randomly created. n2 *FirmObstacle* objects were randomly created. n3 *ExplosiveObstacle* objects were randomly created. n4 *GiftObstacle* objects were randomly created. These obstacles were added to *GameInfo.obstacles.*

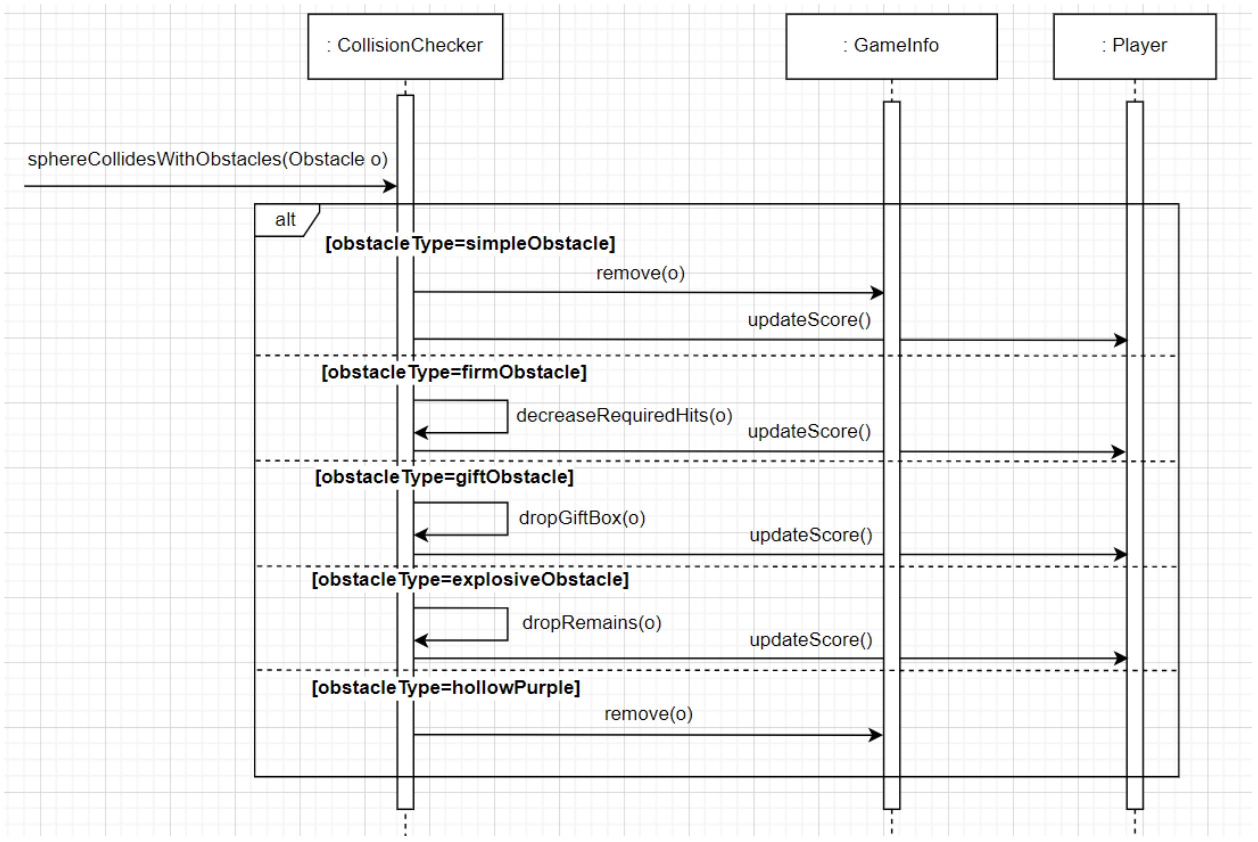
Operation-3: selectGame(savedGame)
Cross Reference: Use case 3 (Load Game)
Pre Condition: *CreateOrLoadOrCustomizePanel.isVisible() was true OR RunningGamePanel.isVisible* was *true.*
Post Condition: The enchanted sphere was changed to the new loaded sphere. The noble phantasm was changed to the loaded noble phantasm. The obstacles were changed to the loaded obstacles. *RunningGamePanel.isVisible* was still true OR *RunningGamePanel.isVisible* was changed to true.

Operation-4: openCustomizeMenu()
Cross Reference: Use Case 4 (Customize Game)
Pre Condition: *OpeningPanel.isVisible()* was *true* or *RunningGamePanel.isVisible()* was *true. CustomizePanel.isVisible() was false.*
Post Condition: *CustomizePanel.isVisible()* was changed to true.

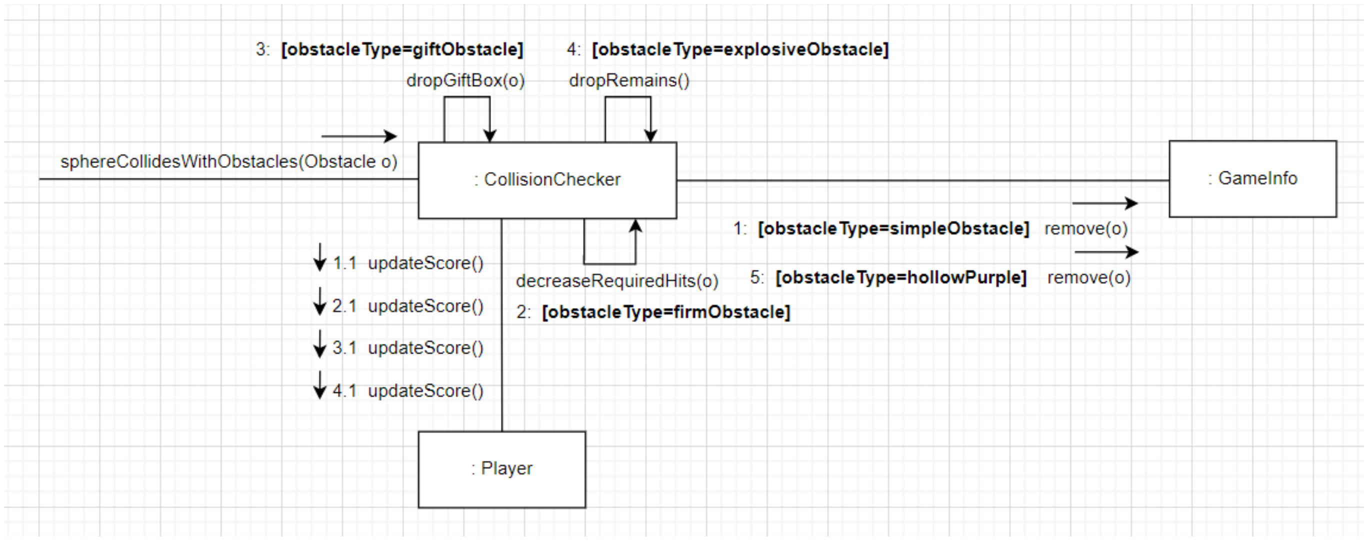
Operation-5: useAbility(ability)
Cross Reference: Use Case 5 (Use a Magical Ability)
Pre Condition: *user.magicalAbilities* != [0,0,0,0] (which means the user had an ability.) and *RunningGamePanel.isVisible()* was *true.*
Post Condition: One value in (depending on the magical ability contained in the box) *user.magicalAbilities* was decreased by one. Magical ability was added to Enchanted Sphere or Noble Phantasm or user depending on the type of ability. (For example *NoblePhantasm.length* could be doubled.)

Interaction Diagrams

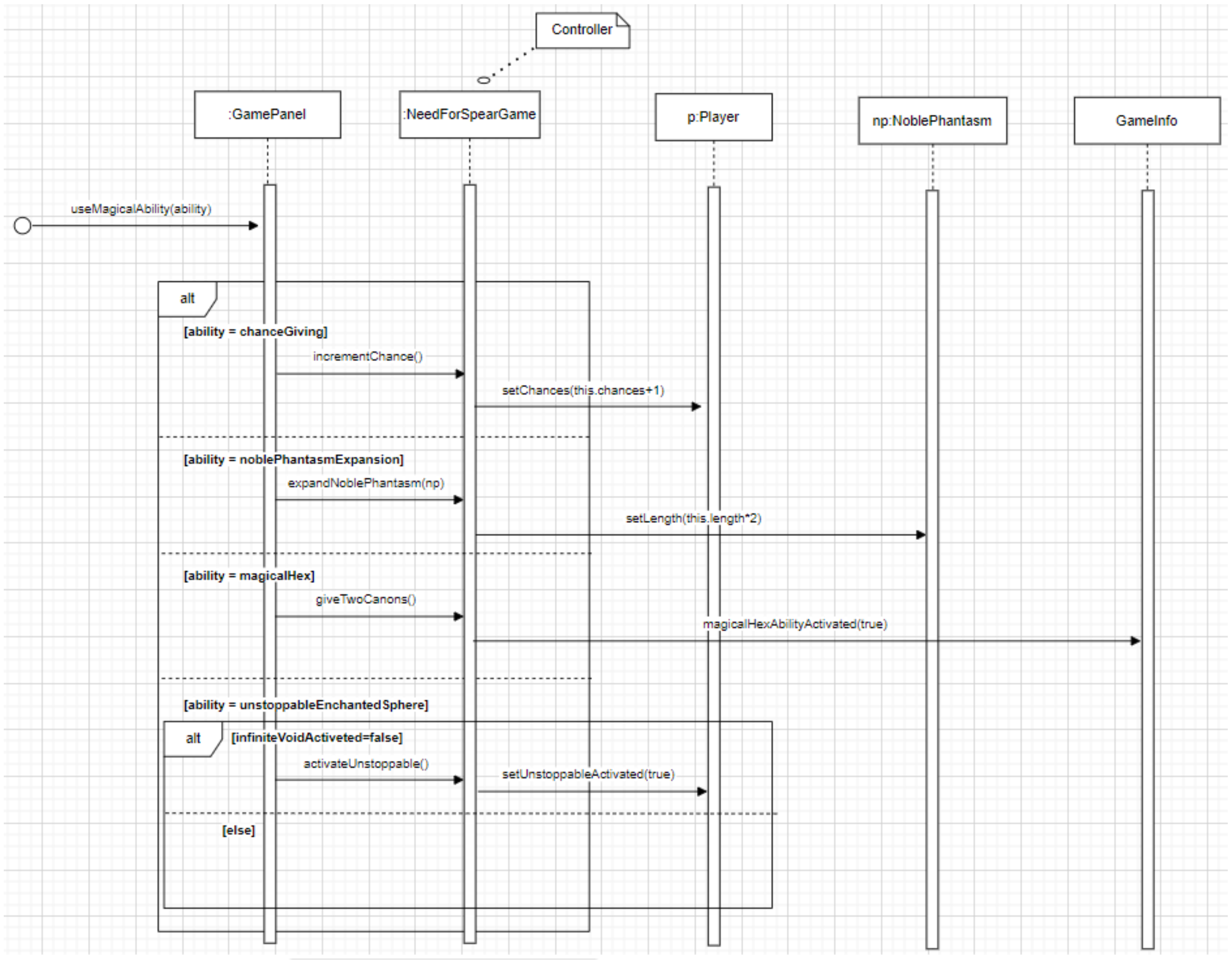
Sequence Diagram-1:


: CollisionChecker
: GameInfo
: Player
sphereCollidesWithObstacles(Obstacle o)
alt
[obstacleType=simpleObstacle]
remove(o)
updateScore()
[obstacleType=firmObstacle]
decreaseRequiredHits(o)
updateScore()
[obstacleType=giftObstacle]
dropGiftBox(o)
updateScore()
[obstacleType=explosiveObstacle]
dropRemains(o)
updateScore()
[obstacleType=hollowPurple]
remove(o)

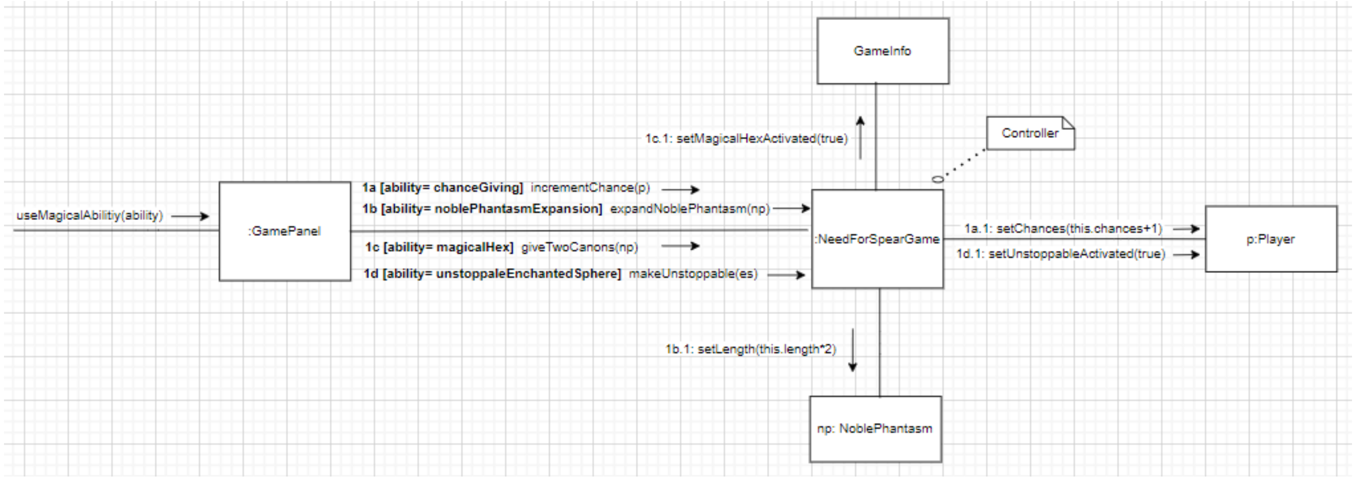
Communicate Diagram-1:


3: [obstacleType=giftObstacle]
dropGiftBox(o)
4: [obstacleType=explosiveObstacle]
dropRemains()
sphereCollidesWithObstacles(Obstacle o)
: CollisionChecker
: GameInfo
1: [obstacleType=simpleObstacle] remove(o)
5: [obstacleType=hollowPurple] remove(o)
decreaseRequiredHits(o)
2: [obstacleType=firmObstacle]
1.1 updateScore()
2.1 updateScore()
3.1 updateScore()
4.1 updateScore()
: Player

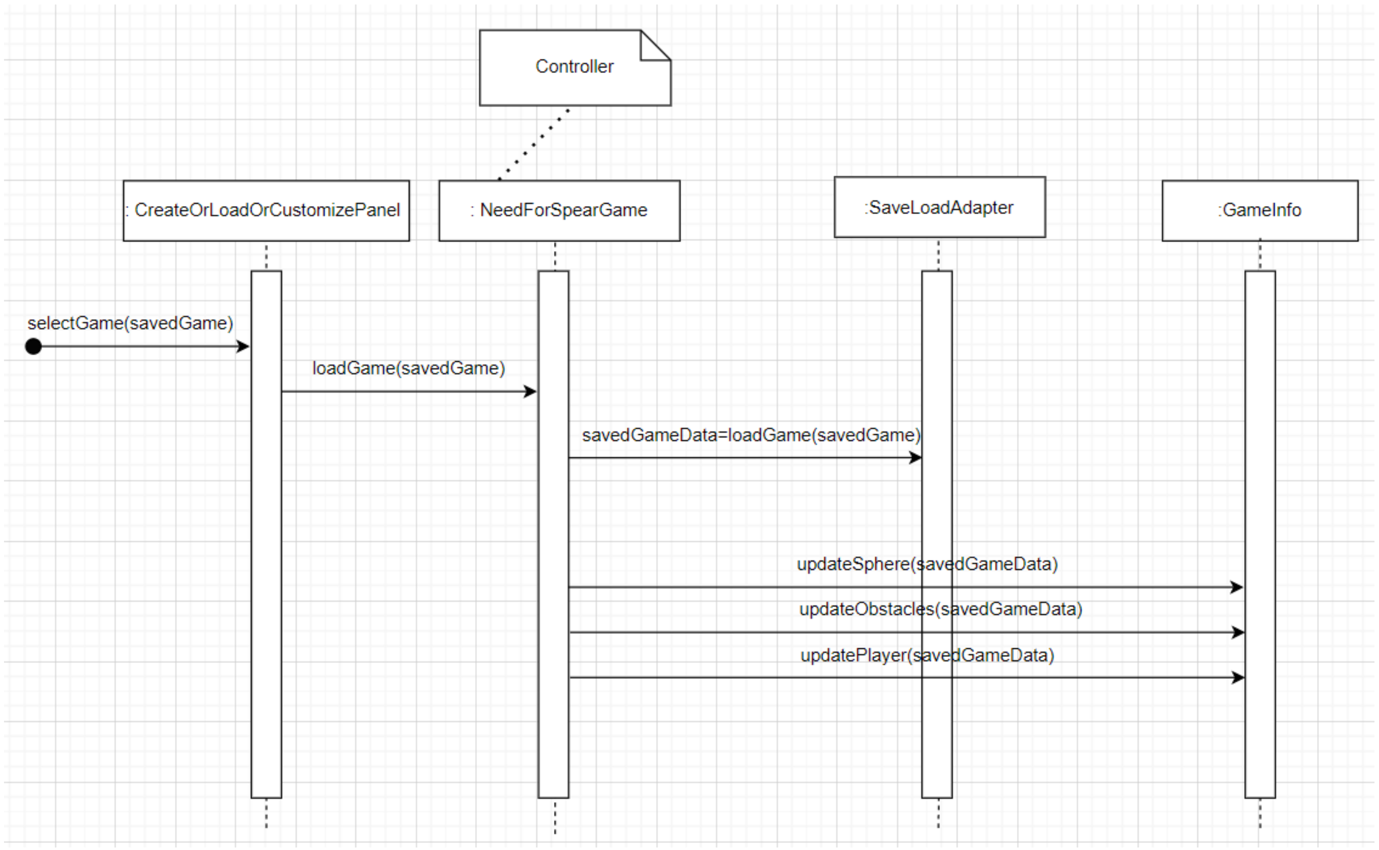
Sequence Diagram-2:



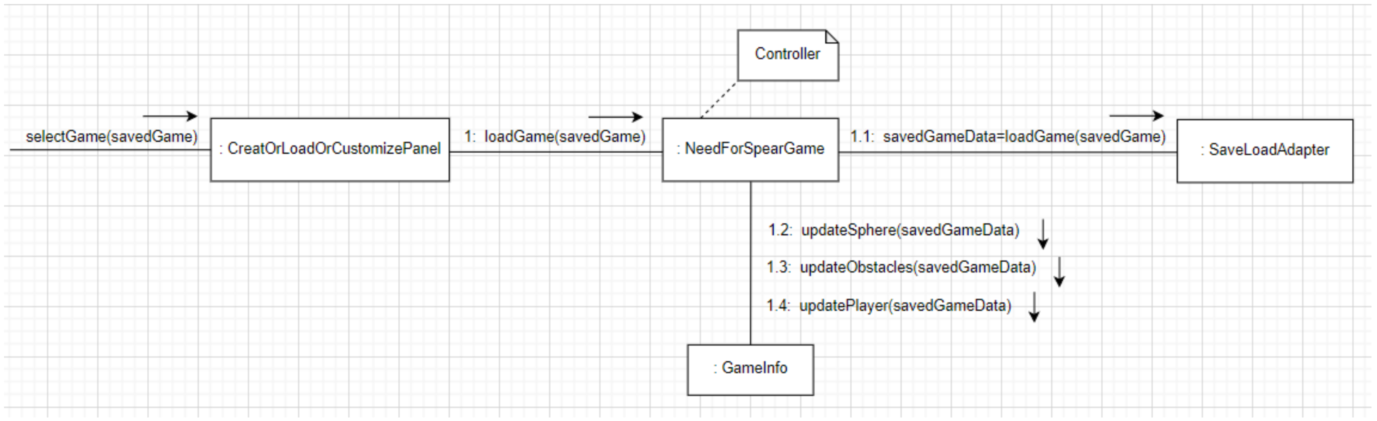
Communicate Diagram-2:


GameInfo
1c.1: setMagicalHexActivated(true)
Controller
useMagicalAbilitiy(ability)
:GamePanel
1a [ability= chanceGiving] incrementChance(p)
1b [ability= noblePhantasmExpansion] expandNoblePhantasm(np)
1c [ability= magicalHex] giveTwoCanons(np)
1d [ability= unstoppaleEnchantedSphere] makeUnstoppable(es)
:NeedForSpearGame
1a.1: setChances(this.chances+1)
1d.1: setUnstoppableActivated(true)
p:Player
1b.1: setLength(this.length*2)
np: NoblePhantasm

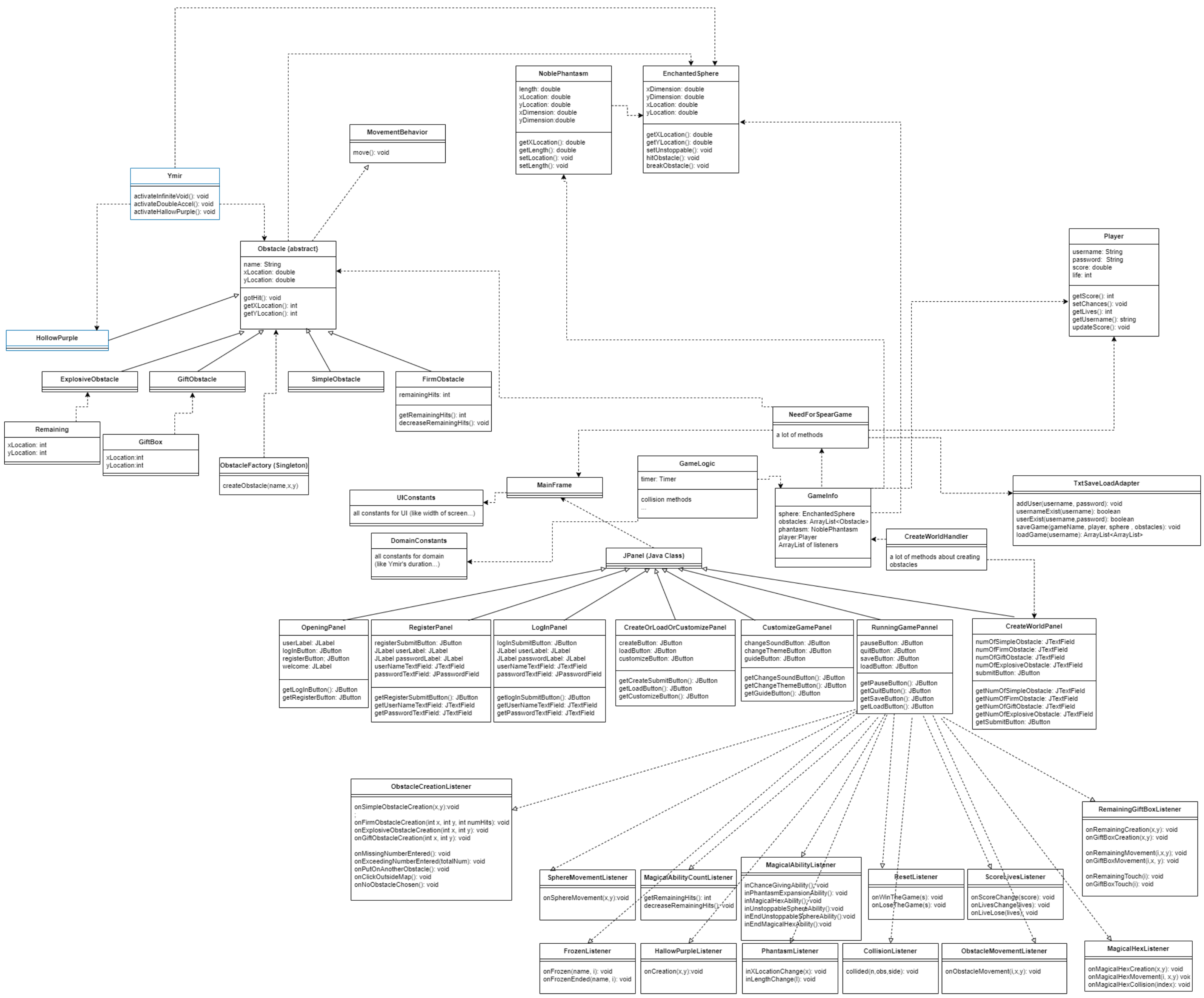
Sequence Diagram-3:


Controller
: CreateOrLoadOrCustomizePanel
: NeedForSpearGame
:SaveLoadAdapter
:GameInfo
selectGame(savedGame)
loadGame(savedGame)
savedGameData=loadGame(savedGame)
updateSphere(savedGameData)
updateObstacles(savedGameData)
updatePlayer(savedGameData)

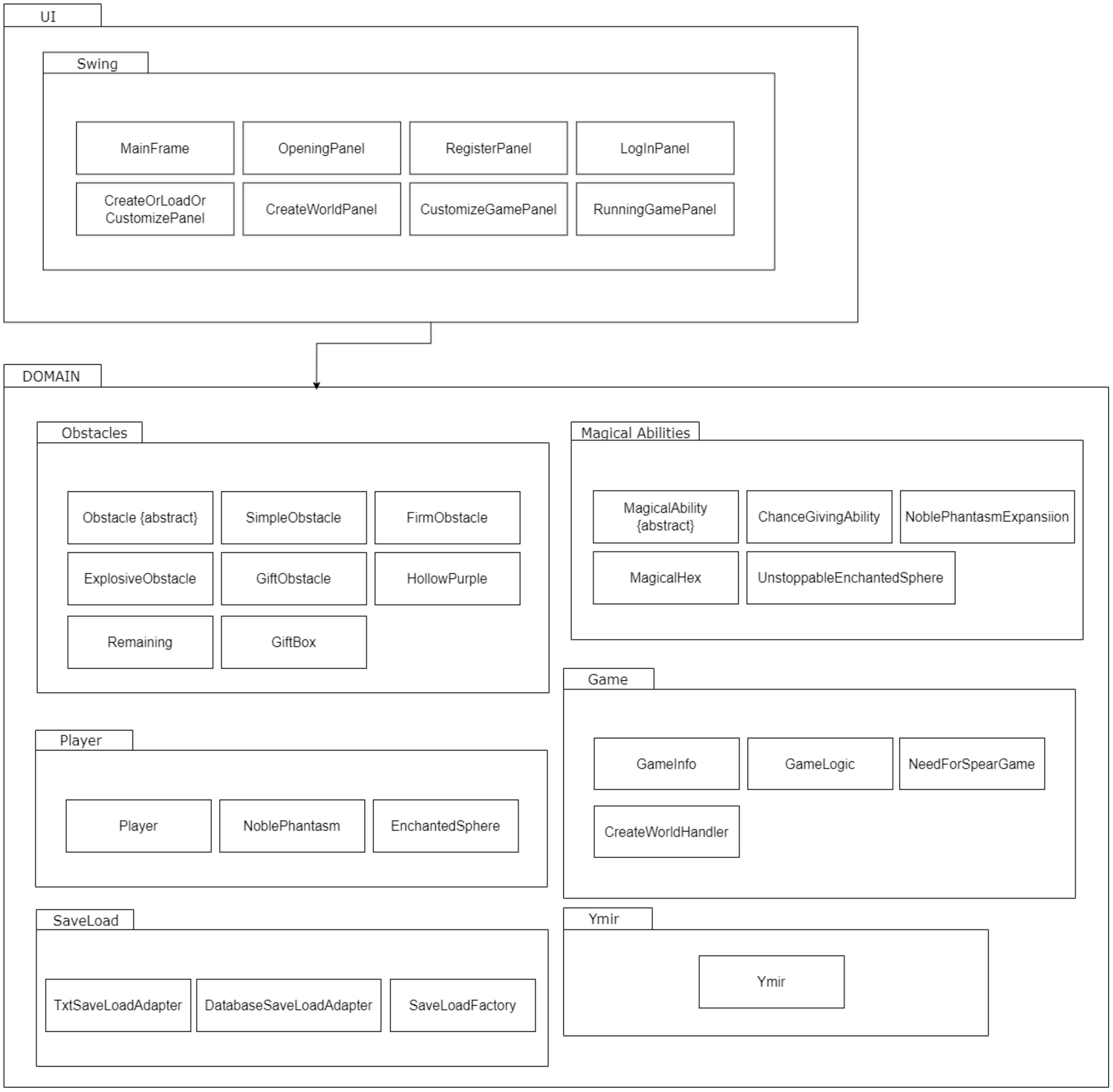
Communicate Diagram-3:


Controller
selectGame(savedGame)
: CreatOrLoadOrCustomizePanel
1: loadGame(savedGame)
: NeedForSpearGame
1.1: savedGameData=loadGame(savedGame)
: SaveLoadAdapter
1.2: updateSphere(savedGameData)
1.3: updateObstacles(savedGameData)
1.4: updatePlayer(savedGameData)
: GameInfo

Class Diagram


NoblePhantasm
length: double
xLocation: double
yLocation: double
xDimension: double
yDimension:double
getXLocation(): double
getLength(): double
setLocation(): void
setLength(): void
EnchantedSphere
xDimension: double
yDimension: double
xLocation: double
yLocation: double
getXLocation(): double
getYLocation(): double
setUnstoppable(): void
hitObstacle(): void
breakObstacle(): void
MovementBehavior
move(): void
Ymir
activateInfiniteVoid(): void
activateDoubleAccel(): void
activateHallowPurple(): void
Obstacle {abstract}
name: String
xLocation: double
yLocation: double
gotHit(): void
getXLocation(): int
getYLocation(): int
HollowPurple
ExplosiveObstacle
GiftObstacle
SimpleObstacle
FirmObstacle
remainingHits: int
getRemainingHits(): int
decreaseRemainingHits(): void
Remaining
xLocation: int
yLocation: int
GiftBox
xLocation:int
yLocation:int
ObstacleFactory (Singleton)
createObstacle(name,x,y)
Player
username: String
password: String
score: double
life: int
getScore(): int
setChances(): void
getLives(): int
getUsername(): string
updateScore(): void
NeedForSpearGame
a lot of methods
GameLogic
timer: Timer
collision methods
...
MainFrame
UIConstants
all constants for UI (like width of screen...)
DomainConstants
all constants for domain
(like Ymir's duration...)
GameInfo
sphere: EnchantedSphere
obstacles: ArrayList<Obstacle>
phantasm: NoblePhantasm
player:Player
ArrayList of listeners
TxtSaveLoadAdapter
addUser(username, password): void
usernameExist(username): boolean
userExist(username,password): boolean
saveGame(gameName, player, sphere , obstacles): void
loadGame(username): ArrayList<ArrayList>
CreateWorldHandler
a lot of methods about creating obstacles
JPanel (Java Class)
OpeningPanel
userLabel: JLabel
logInButton: JButton
registerButton; JButton
welcome: JLabel
getLogInButton(): JButton
getRegisterButton: JButton
RegisterPanel
registerSubmitButton: JButton
JLabel userLabel: JLabel
JLabel passwordLabel: JLabel
userNameTextField: JTextField
passwordTextField: JPasswordField
getRegisterSubmitButton(): JButton
getUserNameTextField: JTextField
getPasswordTextField: JTextField
LogInPanel
logInSubmitButton: JButton
JLabel userLabel: JLabel
JLabel passwordLabel: JLabel
userNameTextField: JTextField
passwordTextField: JPasswordField
getlogInSubmitButton(): JButton
getUserNameTextField: JTextField
getPasswordTextField: JTextField
CreateOrLoadOrCustomizePanel
createButton: JButton
loadButton: JButton
customizeButton: JButton
getCreateSubmitButton(): JButton
getLoadButton(): JButton
getCustomizeButton(): JButton
CustomizeGamePanel
changeSoundButton: JButton
changeThemeButton: JButton
guideButton: JButton
getChangeSoundButton(): JButton
getChangeThemeButton(): JButton
getGuideButton(): JButton
RunningGamePannel
pauseButton: JButton
quitButton: JButton
saveButton: JButton
loadButton: JButton
getPauseButton(): JButton
getQuitButton(): JButton
getSaveButton(): JButton
getLoadButton(): JButton
CreateWorldPanel
numOfSimpleObstacle: JTextField
numOfFirmObstacle: JTextField
numOfGiftObstacle: JTextField
numOfExplosiveObstacle: JTextField
submitButton: JButton
getNumOfSimpleObstacle: JTextField
getNumOfFirmObstacle: JTextField
getNumOfGiftObstacle: JTextField
getNumOfExplosiveObstacle: JTextField
getSubmitButton: JButton
ObstacleCreationListener
onSimpleObstacleCreation(x,y):void
;
onFirmObstacleCreation(int x, int y, int numHits): void
onExplosiveObstacleCreation(int x, int y): void
onGiftObstacleCreation(int x, int y): void
onMissingNumberEntered(): void
onExceedingNumberEntered(totalNum): void
onPutOnAnotherObstacle(): void
onClickOutsideMap(): void
onNoObstacleChosen(): void
SphereMovementListener
onSphereMovement(x,y):void
MagicalAbilityCountListener
getRemainingHits(): int
decreaseRemainingHits(): void
MagicalAbilityListener
inChanceGivingAbility(); void
inPhantasmExpansionAbility(): void
inMagicalHexAbility(); void
inUnstoppableSphereAbility():void
inEndUnstoppableSphereAbility():void
inEndMagicalHexAbility():void
ResetListener
onWinTheGame(s): void
onLoseTheGame(s): void
ScoreLivesListener
onScoreChange(score): void
onLivesChange(lives): void
onLiveLose(lives): void
RemainingGiftBoxListener
onRemainingCreation(x,y): void
onGiftBoxCreation(x,y): void
onRemainingMovement(i,x,y); void
onGiftBoxMovement(i,x, y): void
onRemainingTouch(i): void
onGiftBoxTouch(i): void
FrozenListener
onFrozen(name, i): void
onFrozenEnded(name, i): void
HallowPurpleListener
onCreation(x,y):void
PhantasmListener
inXLocationChange(x): void
inLengthChange(l): void
CollisionListener
collided(n,obs,side): void
ObstacleMovementListener
onObstacleMovement(i,x,y): void
MagicalHexListener
onMagicalHexCreation(x,y): void
onMagicalHexMovement(i, x,y) voi
onMagicalHexCollision(index): void

Package Diagram



Design Patterns

Controller: We created a controller class that is responsible for the whole game except creating the world called NeedForSpearGame. It receives inputs from the UI layer, then distributes them to the related domain layer objects. Therefore, it is also helpful for Model-View Separation. In short, it provides the information flow from the UI to the domain layer. We also created another controller class called CreateWorldHandler which takes inputs about creating the world from UI and sends them to the domain.

High Cohesion: Our controller class gets input from UI and sends them to the most related domain objects, so it helps us achieve high cohesion.

Observer: For the passing information from domain model to UI layer, we have implemented observer pattern via listeners to sustain model-view separation. For example, we have created a listener for the score. This listener is implemented by RunningGamePanel. Whenever the score is changed, Player class calls the onScoreEvent() of each class that implements the listener of the score, so the listener notifies the UI layer. Then, onScoreEvent() which belongs to the RunningGamePanel.java class changes the score displayed on the screen. For example, we have created another listener for the location of the sphere. This listener is also implemented by RunningGamePanel. When the location of the sphere changes in the domain, GameLogic class calls onSphereMovement() method of all sphere location listeners, so the change of the location of the sphere is displayed in the running game panel. While keeping track of hitting the obstacles, there is a need for an Observer pattern (when the enchanted sphere hits an obstacle, with the help of the observer pattern; UI layer is notified and the obstacles remain unchanged or disappear in UI). In this hitting process, we need the Observer pattern also to update the number written on the FirmObstacle. Whenever a firm obstacle is hit, with the help of the Observer pattern, the UI is notified and the number on the Firm obstacle decreases. We have used the observer pattern excessively for informing UI about several more other events that occur on domain.

Polymorphism: We used the polymorphism pattern for Obstacles since we have different kinds of them. Thanks to polymorphism, we implemented common methods of obstacles (like *move()*) for each obstacle separately. For example, when we call *move()* for an obstacle object, it behaves depending on the type of the obstacle. This saves us from using if-else statements for checking the type and allows us to add new obstacles if the product owner wants in the future.

Factory Pattern: We used factory pattern to create obstacles. We have used the factory pattern for saving the game. The saving type is received by using the adapter pattern, then this input is taken and given to the factory pattern. The factory pattern is used with singleton.

Pure Fabrication: Since we have given the duty of the creation of object to a new class (factory) that does not exist in our domain model instead of existing classes, we have also applied pure fabrication and this increased the cohesion.

Low Coupling: We used sub-classes to do low coupling. For example; we have a GameInfo.java class to store the game instance variables, so there is minimum coupling between the classes and inside the classes.

Singleton: We applied the Singleton pattern for factory classes. Instead of creating objects for factories, we directly reach them by calling their static functions. This enables us to save memory.

Information Expert: We have a class named GameLogic to detect collisions. We also have a class called GameInfo which contains all information about obstacles, the enchanted sphere, etc. (like their locations). Also, in general, we have given importance to assigning the responsibilities to the classes which are most related and have information to fulfill that responsibility.

In general, these patterns increase the complexity of our codes (by increasing the number of packages, classes, etc.). But it helps us have a better design. As another con, following the design patterns may prevent our creativity. However, it is clear that following these patterns would provide better design than our creativity.

These patterns and strategies are powerful, and they can be used the wrong way easily, creating way more complex code to read and execute for the programmers. In short, they can be easily misused and it would be very harmful rather than being beneficial.

However, despite the cons, in general, these patterns, especially controller and observer, were really useful for designing and implementing our project.

Supplementary Specifications

Introduction

Supplementary specification document involves all the requirements which are not mentioned in the use cases.

Usability

The customers (users) will see the menu including the “Login” and “Register” parts, the game’s main screen (including “Save” “Load” “Pause” “Quit”), and the menu including the “create game”, “load game”, and “customize game” parts. So, there are some important human factors that affect game play.

Human Factors

The system messages, the writings in the buttons, and the writings on the pages should be readable and understandable enough so that users cannot encounter a problem with playing the game.

Very shiny (also very dark colors and very light colors) colors should not be used in the game so that users and people having some defects of vision cannot face a problem with seeing the images and writings in the game.

Moreover, to provide a better gameplay experience to the customers; the game should be easy to understand and play. The transaction from the login screen to the main screen of the game should be fast enough. Furthermore, to make the gameplay experience better, the process of hitting the obstacles and destroying the obstacles should be fast enough. To provide a smooth and fast gameplay experience to the customers (users), the game should include almost no errors (preferably no errors).

Reliability

If the user has a problem with his PC, such as frozen screen or broken keyboard and broken mouse due to rage quit, the game saves itself if the user does not do any IO event for some time - the time is specified in the game menu - so that the progress is not lost.

If the user wants to delete a load game, it is asked for confirmation of the deletion, so that there is no trouble of a misclick.

If the user wants to delete his/her account, he/she can do it at any time.

No data of the user is shared with third-party companies, so the user's privacy is %100 safe. Also, we don’t use cookies.

Performance

We do not want users to spend a lot of time trying to start a game, so there is only a log-in menu, and the user can start the game with just 2 clicks.

If the user wants to create a new level, it is implemented simply.

When the user types save, there is no failure in saving with %99.9.

If the user has a low-performance PC, then he can change the theme to a lower standard to get better performance in the game.

Saved games are stored in the PC with the minimum storage capacity.

Supportability

This game runs on all operating systems.

Users can use a mouse or a touchpad.

The user can change the theme so that it needs lower performance to display the screen which yields to better performance.

Implementation Constraints

We should be developing with Java by having a point of view that Java will support several Java-based tools like Maven (Java-based build automation tool). Moreover, it will be easy to develop with Java by using concepts including modularity, inheritance, polymorphism, and other object-oriented programming concepts.

Free Open Source Components

To make the process of game development faster, we should try to find more Java-based tools which are freely available.

Legal Issues

In the process of sales of this game, the taxations of the sales should be properly applied depending on the law regulations.

Glossary

Terms	**Definitions & Information**
Score	The number which shows the user's success and increases when the obstacles are broken.
Noble Phantasm	The game object which allows the user to catch the enchanted sphere and reflect it.
Lives	The number which shows how many times the enchanted sphere can fall below the noble phantasm / the remains (remainings) of the explosive obstacle can touch the noble phantasm without losing the game.
Magical abilities	The abilities that can be used/collected by the player. They are included in gift boxes which are dropped from gift obstacles when they are broken. When the noble phantasm catches a gift box, a random magical ability is given to the player. The types of the magical abilities are “chance giving ability”, “noble phantasm expansion”, “magical hex”, “unstoppable enchanted sphere”.
Chance Giving Ability	The magical ability that increments the chances of the users by 1.
Noble Phantasm Expansion	The magical ability that doubles the length of the noble phantasm and lasts 30 seconds after the activation of it by the user.
Magical Hex	The magical ability which adds 2 magical canons to the both ends of the noble phantasm and which lasts 30 seconds after the activation. The magical canons can create magical hexes which hit the obstacles.
Unstoppable Enchanted Sphere	The magical ability which makes the enchanted sphere destroy each type of obstacle after hitting them, and this magical ability lasts 30 seconds after the activation.
Obstacles	The game objects that are destroyed by the enchanted sphere. The obstacles have different types, and every obstacle has a different effect in the game.
Simple obstacle	The obstacle that is destroyed with just one hit.

Firm obstacle	The obstacle that is destroyed with the number of hits written on the obstacle.
Explosive obstacle	The obstacle which has a sphere-like shape, and which explodes and creates remains (remainings) when it gets destroyed.
Gift obstacles	The obstacle that breaks with just one hit and drops a reward falling down. If the user catches it, the user gets the reward.
Building mode	The game mode in which a player can create a player map by removing, dragging, adding obstacles, and by making the system create obstacles.
Running Mode	The game mode in which the player controls the noble phantasm, and plays the game.
Ymir	It is the spirit of an old sorcerer that was brought to life again in order to prevent any warrior from obtaining the spear of power.
Infinite Void	The magical ability of the Ymir which lasts 15 seconds. After Ymir uses this magical ability, if there are less than 8 obstacles in the player's map; all the obstacles are chosen to be frozen for 15 seconds. If there are more than 8 obstacles in the player's map, 8 obstacles (not destroyed yet) are randomly chosen from the player's map.
Double Accel	The magical ability of the Ymir that lasts 15 seconds, and that reduces the speed of the Enchanted Sphere by half.
Hollow Purple	The magical ability of the Ymir that randomly adds 8 hollow purple obstacles in the player's map. Hollow Purple obstacle behaves in the same way with the Simple Obstacle (Wall Maria). The only difference is that breaking the Hollow Purple obstacle does not contribute to the score calculation.