KOÇ ÜNİVERSİTESİ
COLLEGE OF ENGINEERING

COMP 302: SOFTWARE ENGINEERING

Fall 2021

Final Report

İrem Demir

Atakan Kara

Eren Yenigül

Asu Tutku Gökçek

Farrin Marouf Sofian

Table of Contents

Introduction

In this report, the finalised version of the team SyntacticSugar's game Need for Spear is explained in terms of planning, design and implementation. Need for Spear is an improved version of the famous game Brick Breaker. This report is going to start going over the game by explaining the vision of the project and the contribution of the team-members. Then, it will go over the Use Case Diagram, use case narratives, System Sequence Diagrams, operation contracts, Sequence and Communication Diagrams, Class Diagrams, Class Diagrams and Package Diagrams. After those, there is going to be the discussion of design alternatives and design patterns and principles. Finally, the report will be finalised after supplementary specifications and glossary.

Vision

Business Opportunity:

Different versions of the game are already available in the market. Different versions present the same brick breaking idea with different features. Since it is a well-known game, people will be interested if better graphics and different features are introduced.

Problem Statement:

Versions that are available on the market are popular and widely played. However, playing the same game with the same features becomes dull. Thus, a new version of the game with new changes are requested.

Product Position Statement:

Need for Spear is for people who like to play games in the range of all ages following the logic of an Atari game with modern graphics and features like 2-player compete.

Alternatives and Competition:

There are many versions of this game in the market (especially for Android& iOS). However it is not as common for computers.

User Goals

Player is the only stakeholder of the game Need for Spear.

Product Perspective

This game project exists on the user's computer without any special requirements. The player will be able to run the game and start the fun immediately.

Teamwork Organization

İrem Demir

Implemented ball movement and collision for basic requirements in the ball branch. perpCollision branch handles the case of how the ball will be reflected when colliding with the paddle perpendicularly. Initialized AbilityEngine and abilities in the AbilityEngine branch. Added the magical hex rotation while the paddle is rotated in the magicalHexRotation branch. Worked in the hollowPurpleColor branch in which locations of hollow purple obstacles are updated following the new gameplay area size. Made final updates in UI frames regarding the structure and layout. Changed the structure of the title screen to resolve bugs in the titleScreenFix branch. Created information panel in the InfoPanel, informationScreenUpdate, YmirtoUI branches. Added the methods to get needed information. Active useful abilities and Ymir's abilities added to the information panel. Connections with other screens like are done. Added deleting obstacles in the build mode. Created a button in BuildDelete branch, if deleting mode is active, it will delete selected obstacles both in UI and send a message to the domain to delete obstacles. Created inventory of BuildScreen in BuildScreenInventory branch. Added a function that checks if added obstacles meet the minimum requirements to the gameboard and game. Created a screen that pops when minimum requirements are not met. Game grid logic is added to the build screen and domain. Using the game grid, controlled if there exists an already added obstacle and did not allow adding if there is one. The random game option is added to the domain. Gameboard creates a random game that meets minimum requirements. Changed the calculations for adding an obstacle in a clicked point and creating its location are updated. Tested ball collision in BallCollisionTests.

Atakan Kara

Created obstacle related classes: Obstacle superclass, ExplosiveObstacle, FirmObstacle, GiftObstacle, HollowObstacle, ObtacleFactory and Simple Obstacle. Added health and events happens when obstacles are broken. Connected collision behaivor with obstacles to be able to handle those events. Implemented remainings of explosive obstacle. Used observer pattern to tell the UI when an explosive obstacle is broken. Created initial structure and logic of the game screen. Added a mathematical trick (using modulo

operator) to add obstacles on a grid like structure. Created ymir with a thread added underlying logic. Connected Ymirr to domain using controller. Implemented related functions on game, gameboard, hollowPurpleAbility, InfiniteVoidAbilitiy, DoubleAccelAbility, abilityFactory, on the domain using controller pattern. Implemented Ymir abilities inheriting from Abilities superclass. classes to use Ymir's abilities. Created wonlost events using Observer pattern and added conditions to check this events. Added time and score variables to game and implemented needed logic to calculate those. Improved abilities by debugging them and adding functionality for edge cases. Made ball stick to the paddle on the beginning of the game and each time user loses a chance. Made final improvements on abilities and made them satisfy requirements. Updated ball movementBehavior and collisionBehavior accordingly. Worked on those branches: obstacleHealth, fixHollow, remainingExplosiveObstacle, wonlost, savetime, fixAbilities, stickyBall, ymir, circlerObstacle, test-addObstacle, obstacles, buildGameScreen.

Eren Yenigül

Added following features, utilities and classes to the game:

Implemented collision related features such as Collision Mechanism, Polygon and Sphere Bounding Boxes, (which heavily uses functions of our custom Vector function), Collision Engine and lastly Collision Dataclass which stores collision information for Collision Behaviors. Bounding Boxes provide functions that can be used to detect if two bounding boxes are colliding. These functions produce information on how the collision is occurred. For example, the normal vector of the collision, or the references to the physical objects that are colliding. Collision Engine uses these functions to detect the collisions and then calls the Collision Behavior of the related Physical Objects.

I also implemented the Physics Engine and Movement Behaviors for Physical Objects which is responsible for moving the Physical Objects according to their Movement Behaviors. I also worked on the drawing of sprites, the rotation of the paddle, the movement of the paddle, the movement of the ball and its reflection. I implemented MagicalHexAbility, Paddle Expansion Ability, Double Accel Ability, Ability Factory, and fixed some bugs on the Ability Engine. I created the Vector class and its really crucial functions such as: add, cross, dot, rotate etc. I created the PhysicalObject superclass which is the basis of anything that the players interact with during the game. I also added these classes to the game: Explosive Obstacle Fragments, Gift Obstacle Fragments, Service superclass (which allows some classes to gain some functionalities that the GameBoard provides), Walls, GameBoard Services, Listeners(which the UI heavily uses to add or remove images from the screen whenever an obstacle is deleted etc.)

My branches: collison, ball, abilityengine, objectLabel, magical-hex, gift-obstacle, ability-display, imageToUI, eyenigul-test, rotatePaddle

Asu Tutku Gökçek

Worked on all the UI elements that would be the first thing the user sees after the game starts. All the necessary UI elements like buttons and labels were implemented and connected to the necessary functions with listeners. A JPanel was created which covered the screen to look as the title screen which had all the UI elements attached to. Buttons were added that allowed the player to start the game, enter the build mode, save and load. There was also another button which was named “Exit” that would let the player to end the game instantly which was removed later during the process. A pause screen was implemented which appeared when the “Pause” button was clicked. On the pause screen there were buttons to allow the player to save the game and resume etc. All of these implementations were done in the pausetitle branch. needForSpearUI branch was created while trying to solve some title screen related errors. In the gameUI branch a test class has been added with some ui additions. agokcek19_testBallMovement branch was implemented for the test submission of the project which had test cases for enchanted sphere’s movement. During the process, there have been a couple branches that had to be deleted due to some merge problems. However, all the necessary implementations were later added within other branches. Tried fixing the screen related problems like the screen not fitting the computer’s screen.Worked on fixing some build screen and enchanted sphere movement errors for example: the invisible obstacles the sphere was hitting. Added test cases in the testVector branch which was the test branch for Vector class which was chosen by the whole group. Also, was in charge of the arrangement and the submission of the Final report of the SyntacticSugar’s project.

Farrin Marouf Sofian

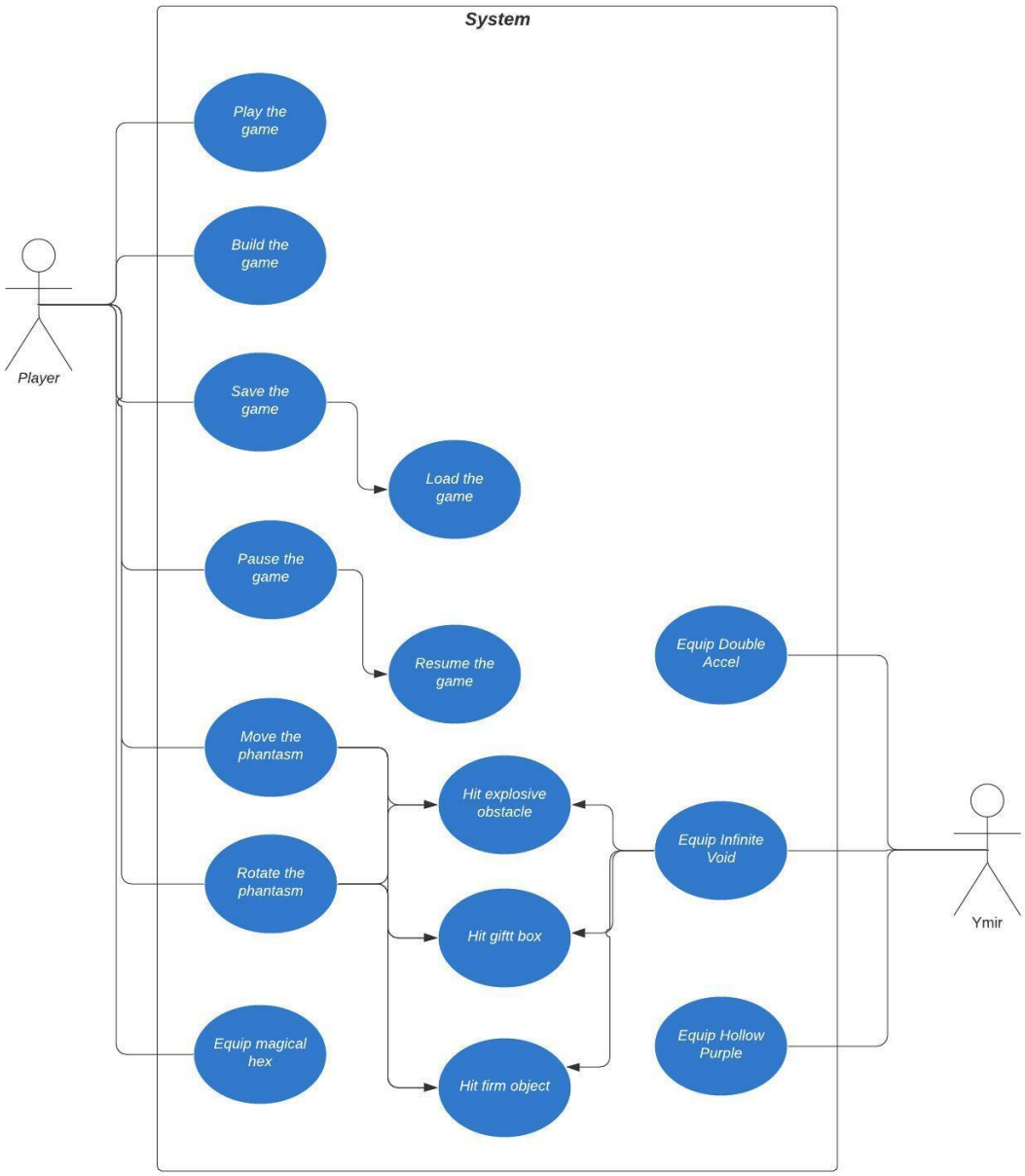
Responsible for loading and saving the game as well as the load game and help screens. The branches where the aforementioned sections are implemented are as follows: SaveGame, save-ability, save-magicalAbilities, loadGameScreen, helpScreen, load-game-update and updateTitleScreen.
The game starts with a title screen in which the user enters his/her name, this code is implemented in the updateTitleScreen. From the title screen, the player can gain information about the game from the help screen which was developed in the helpScreen branch. If the player creates or randomly chooses a game from the build screen, he/she can save the game at any time in build mode and running mode. At run time

the game can only be saved when it is paused. Player's unused abilities, health, score, username, time and obstacle types and their positions are saved and stored in a json file. The branches containing "save" in them each maintain the code for different parts of the saving process. If the player chooses load game button from the title screen, the game checks for his/her previously saved games and lists them so that he/she can choose among them. The latter section is created in the loadGameScreen.

However, if the player has not saved any games yet, he/she can return back to the title screen and create a new game from build mode.

Different tests were written to ensure that the classes/methods serve their purpose such as the paddle movement (farrin-test-paddleMovement branch) and load game class test (test-load branch). The Vector class was chosen as the group test class where all group members implemented a test case for important methods of the Vector class (test-vector branch).

Use Case Diagram



Use Case Narratives

Use Case 1: Play the Game

Use Case Name: Play the game
Scope: Game
Level: user-goal
Primary Actor: Player
Stakeholders and Interests: Player / wants to play Need for Spear's game
Preconditions: Player has a computer in which required softwares is installed.
Success Guarantee: Game opened and processed without errors.
Main Success Scenario:

1. Player opens the game.
2. Build mode shown to the user.
3. Player performs Build(setup) Game.
4. Game starts with the selected layout.
5. Player controls a noble phantasm to protect the enchanted sphere.
6. Enchanted sphere interacts with obstacles in the game.
7. Game state updated according to the result of the interaction.
8. System repeats between 5-7 until all obstacles are cleaned.
9. Player wins the game.

Extensions:

*a At anytime, player performs Pause the Game
    1. Player performs Resume the Game
    2. Player performs Save the Game
    3. Player goes back to start
    4. Player exits the game

5.a Player can't protect the enchanted sphere.
5.b Player loses live.
5c. If the player is out of lives, the game's lost screen appears.

Special Requirements:Game is installed on the machine.
Technology and Data Variations List: A database
Frequency of Occurrence: Every time player wants to play a game.
Miscellaneous:

Use Case 2: Build(Setup) the Game

Use Case Name: Build(Setup) the Game
Scope: Game*
Level: user-goal
Primary Actor: Player
Stakeholders and Interests: Player / wants to load or create a layout
Preconditions: Game has been started.
Success Guarantee: A functional layout is shown and the game is ready to start.
Main Success Scenario:

1. "Create a new layout" and "Load a layout" buttons are shown to the user.
2. User clicks on the "create a new layout" button.
3. Obstacles are given to the user to edit on an empty screen.
4. User creates the layout with provided obstacles.
5. User saves the layout.
6. Relevant information is saved to the database.
7. Game has started with the created layout.

Extensions:

2.a. Users click on the "Load a layout" button.
2.b. A pop-up window for file selection shown to the user.
2.c User selects a layout.
2.d Layout is successfully loaded.
2.e Game has started with the loaded layout.
4.a User didn't satisfy the minimum requirements of the layout.
4.b A pop-up shown to warn the user about the requirements.

Special Requirements:
Technology and Data Variations List: There should be a database for either saving or loading a layout.
Frequency of Occurrence: Every time player opens the game.
Miscellaneous:

Use Case 3: Hit the Gift Box (Gift of Uranus)

Use Case Name: Hit the Gift Box (Gift of Uranus)
Scope: Game
Level: Subfunction
Primary Actor: Enchanted Sphere which bounces off the Phantasm
Stakeholders and Interests:**Preconditions**: The game is running and the Enchanted Sphere is moving towards the obstacle.
Success Guarantee: The Enchanted Sphere hits Gift Box

Main Success Scenario:

1. Enchanted Sphere bounces off of the Phantasm.
2. Enchanted Sphere hits the obstacle
3. Obstacle gets destroyed
4. The destroyed obstacle drops a box downwards towards the noble phantasm.
5. The Noble Phantasm touches the box
6. 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.

Extensions:

5a. The Noble Phantasm does not touch the box.

5a.1 Nothing happens

Special Requirements: None
Technology and Data Variations List: None
Frequency of Occurrence: Sometimes (depends on the number of the gift box)
Miscellaneous: None

Use Case 4: Equip Magical Hex

Use Case Name: Equip Magical Hex
Scope: Game
Level: Subfunction
Primary Actor: Player
Stakeholders and Interests: Player wants to fire magical hexes that hit obstacles
Preconditions: The existence of noble phantasm
Success Guarantee: The noble phantasm is equipped with the magical hex to be able to fire magical hexes
Main Success Scenario:

1. The magical hex is equipped when the player presses H or clicks on the icon on the screen
2. The player shoots the hexes by pressing S on keyboard and if the noble phantasm rotates, the magical canons also rotate
3. The magical hex is functioning for 30 seconds before it disappears

Extensions:
Special Requirements: Non
Technology and Data Variations List: The player should press H or use a mouse to click the icon on the screen
Frequency of Occurrence: Rarely
Miscellaneous:

Use Case 5: Equip Infinite Void

Use Case Name: Equip Infinite Void
Scope: Game
Level: Subfunction
Primary Actor: Ymir
Stakeholders and Interests: Ymir / wants to make game difficult for player
Preconditions: The game is running and Ymir had a successful toss and infinite void is the randomly chosen magical ability.
Success Guarantee: Chosen obstacles are freezed for 15 seconds and cannot be affected by an enchanted sphere nor the magical hex ability. They can be only destroyed by the unstoppable enchanted sphere.
Main Success Scenario:

1. The infinite void equipped by the game.
2. Randomly chosen 8 obstacles are freezed
3. Ball hits the frozen obstacles it reflected but obstacles are not destroyed
4. Magical hex used and no effect on the obstacles
5. Infinite void stays active for 15 seconds.

Extensions:

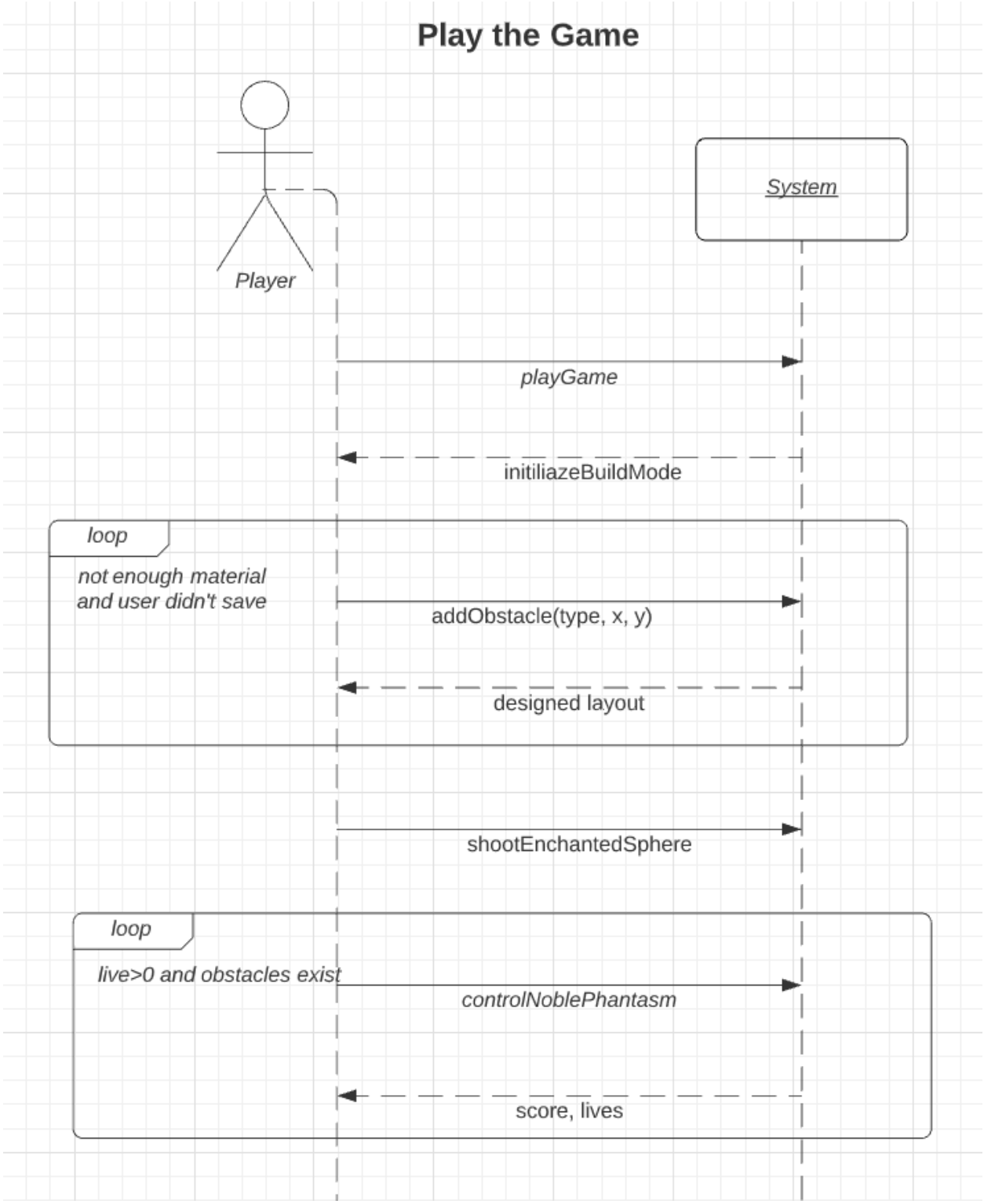
*a At anytime, player performs Pause the Game

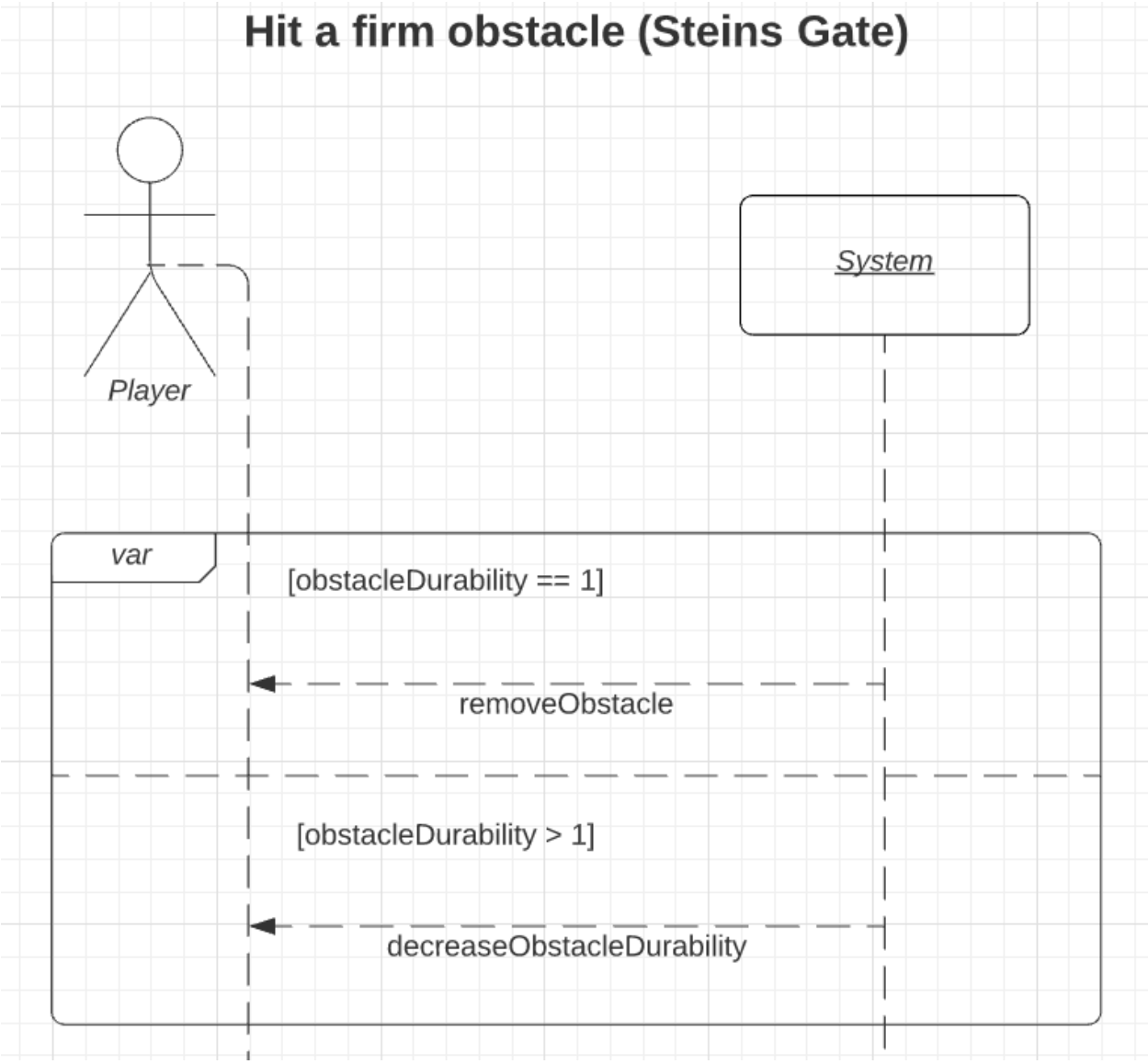
1.Player performs Resume the Game
2.Player performs Save the Game
3.Player goes back to start
4.Player exits the game

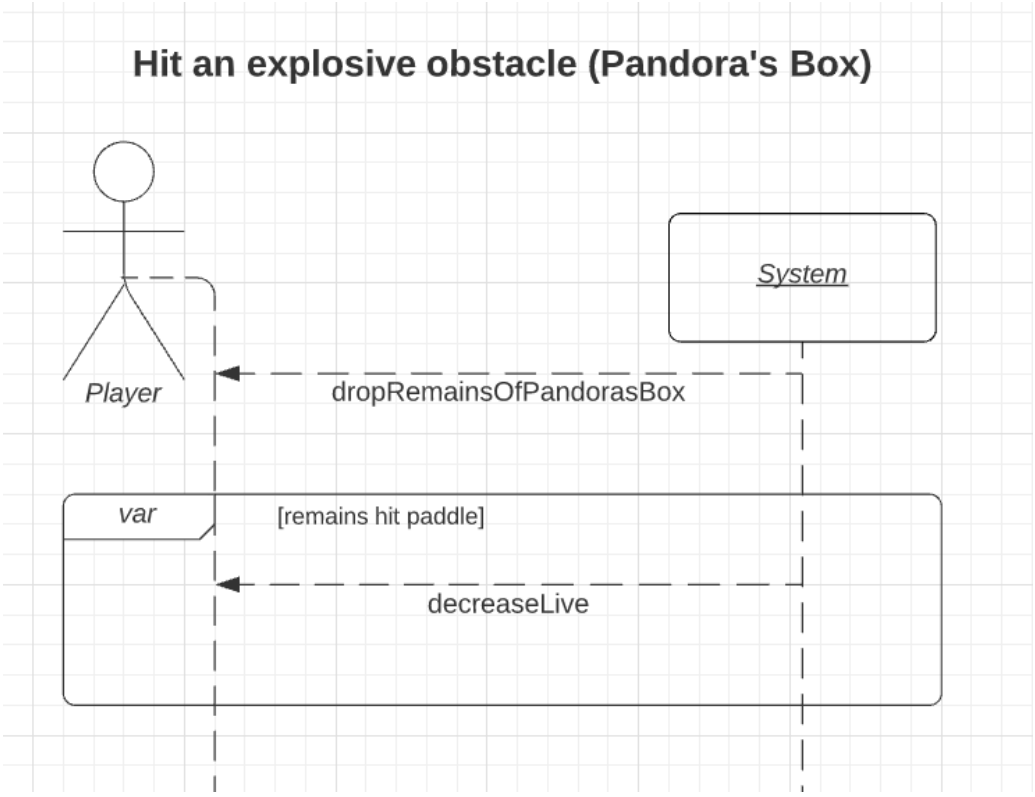
2.a. There are less than 8 obstacles in the game
2.b. All of the obstacles are chosen
3.a. Ball is in the unstoppable mode
3.b. Ball acts like a normal ball on the frozen obstacles

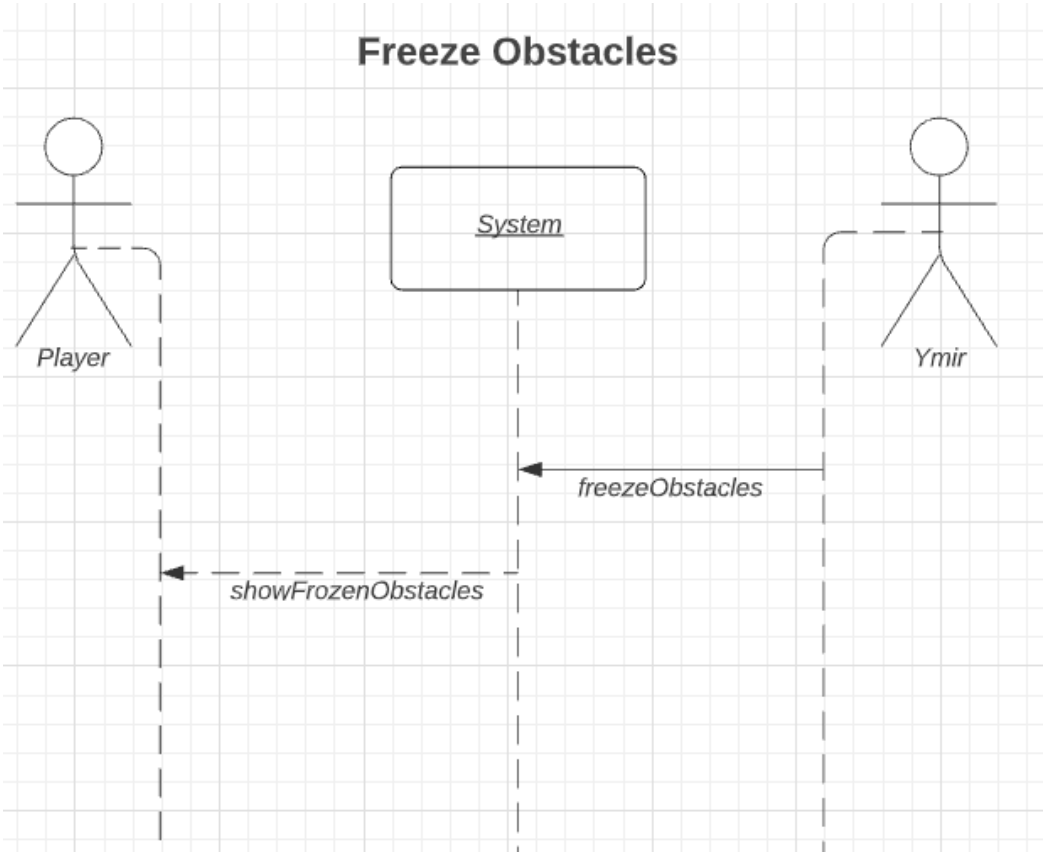
Special Requirements:Game is installed on the machine.
Technology and Data Variations List:
Frequency of Occurrence: On every 30 seconds Ymir wins coin toss(50%) and choses infinite void among 3 options.
Miscellaneous:

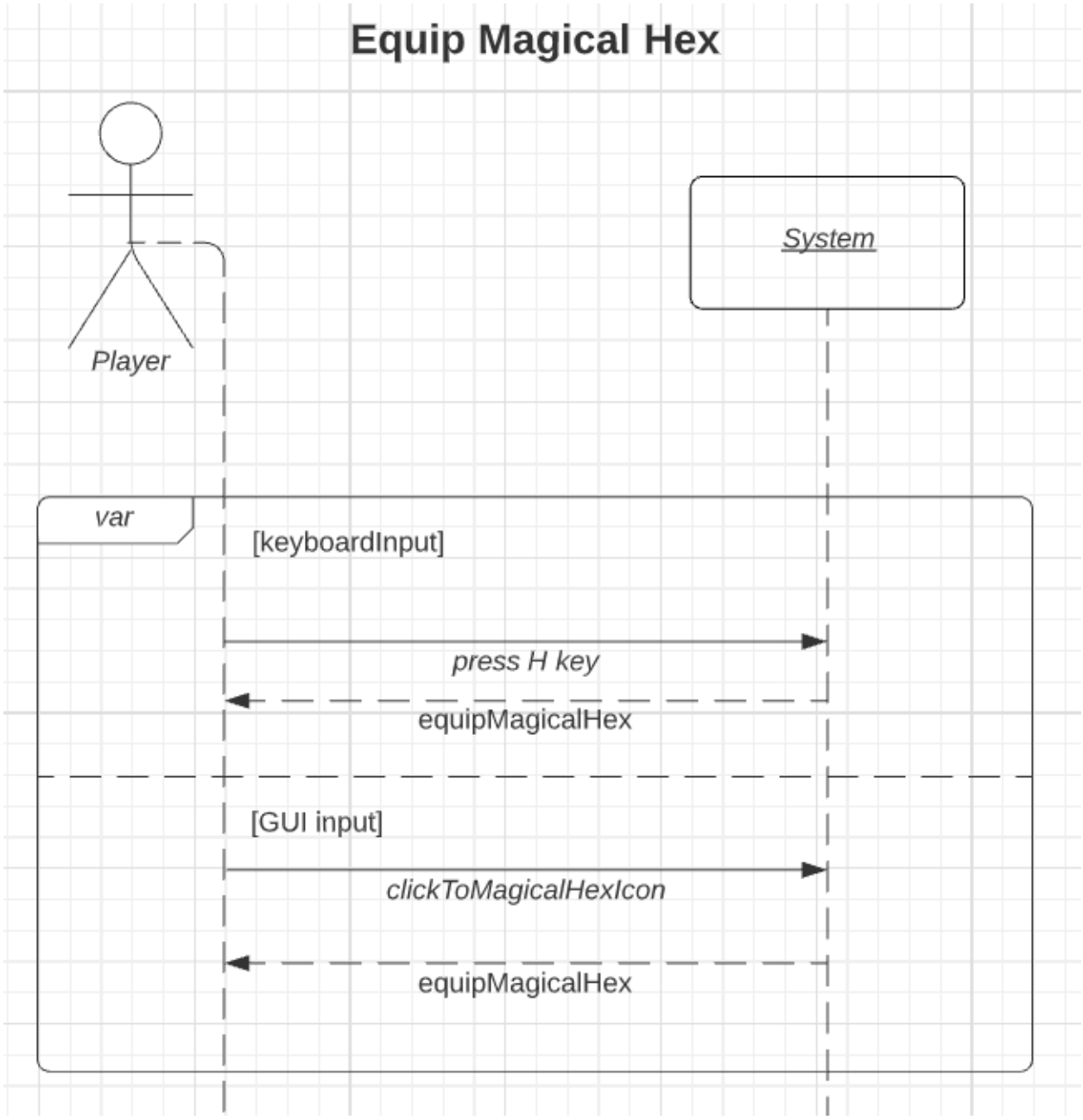
System Sequence Diagrams




Hit a firm obstacle (Steins Gate)
Player
System
var
[obstacleDurability == 1]
removeObstacle
[obstacleDurability > 1]
decreaseObstacleDurability


Hit an explosive obstacle (Pandora's Box)
System
Player
dropRemainsOfPandorasBox
var
[remains hit paddle]
decreaseLive


Freeze Obstacles
System
Player
Ymir
freezeObstacles
showFrozenObstacles


Equip Magical Hex
Player
System
var
[keyboardInput]
press H key
equipMagicalHex
[GUI input]
clickToMagicalHexIcon
equipMagicalHex

Operation Contracts

Contract CO1: Shoot:

Operation: shoot()

Cross References: Use Case 7: Moving the Phantasm
Use Case 8: Rotating the Phantasm
Use Case 9: Hitting the Simple Obstacle (Wall Maria)
Use Case 10: Hitting the Firm Obstacle (Steins Gate)
Use Case 11: Hitting the Explosive Obstacle (Pandora's Box)
Use Case 12: Hitting the Gift Box (Gift of Uranus)

Preconditions: Noble Phantasm and Enchanted Sphere are present in the Game Window
Game is in the running mode

Postconditions: enchantedSphere.location = location

Contract CO2: Pick Magical Ability:

Operation: pickMagicalAbility(type)

Cross References: Use Case 9: Hitting the Simple Obstacle (Wall Maria)
Use Case 10: Hitting the Firm Obstacle (Steins Gate)
Use Case 11: Hitting the Explosive Obstacle (Pandora's Box)
Use Case 12: Hitting the Gift Box (Gift of Uranus)

Preconditions: A list of Magical Abilities are available on the game window

Postconditions: Depending on the type of the magical ability, an instance of that type was created and the type of the object was set to type
magicalAbility.type = type

Contract CO3: Move Paddle:

Operation: movePaddle(direction)

Cross References: Use Case 1: Play the Game
Use Case 7: Move the Phantasm
Use Case 9: Hit the Simple Obstacle
Use Case 10: Hit the Firm Obstacle

Use Case 11: Hit the Explosive Obstacle
Use Case 12: Hit the Gift Box
Use Case 13: Equip Magical Hex

Preconditions: Game is started
Paddle can be moved in the specified direction

Postconditions: paddle.location was updated in accordance with paddle.speed and direction given

Contract CO4: Rotate Paddle:

Operation: rotatePaddle(direction)

Cross References: Use Case 1: Play the Game
Use Case 8: Rotate the Phantasm
Use Case 9: Hit the Simple Obstacle
Use Case 10: Hit the Firm Obstacle
Use Case 11: Hit the Explosive Obstacle
Use Case 12: Hit the Gift Box

Preconditions: Game is started

Postconditions: paddle.angle was updated in accordance with paddle.rotationSpeed and rotation given

Contract CO5: Freeze Obstacles:

Operation: freezeObstacles(freezedObstaclesList)

Cross References: Use Case 14: Equip Infinite Void

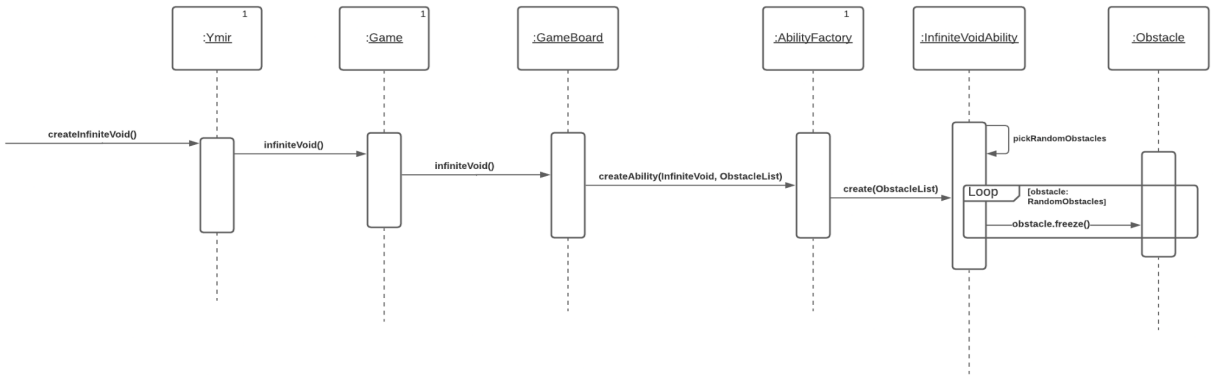
Preconditions: The obstacles in the obstaclesList exists in the game

Postconditions: freeze method of the obstacles in the freezedObstaclesList was called. The obstacles were set to be unbreakable by the enchanted sphere. The freezed obstacles were set to be destroyable by the unstoppable ball. if hit by the unstoppable, the health was set to be decremented by one.

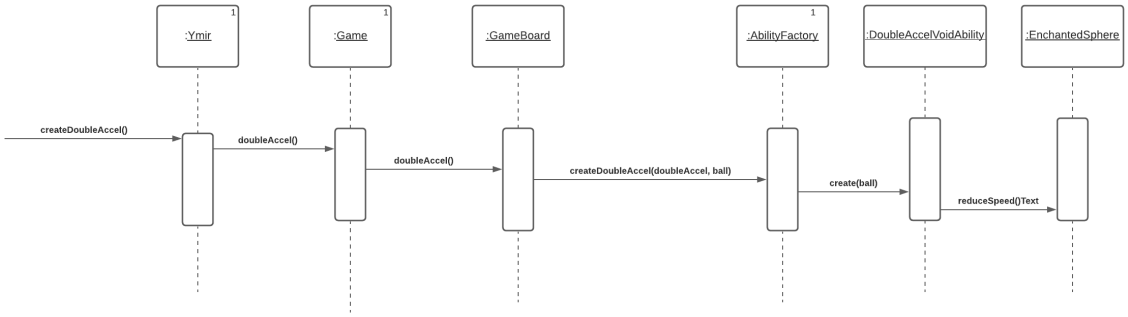
Sequence/Communication Diagrams

Sequence Diagrams

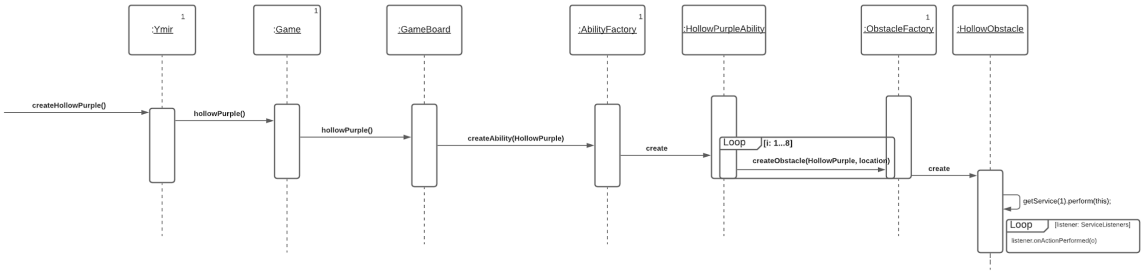
SD.1: Equip Infinite Void



SD.2: Equip Double Accel

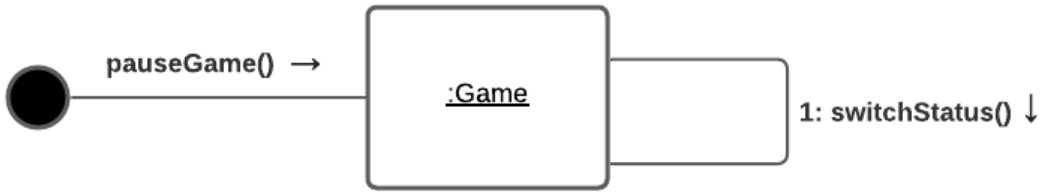


SD.3: Equip Hollow Purple

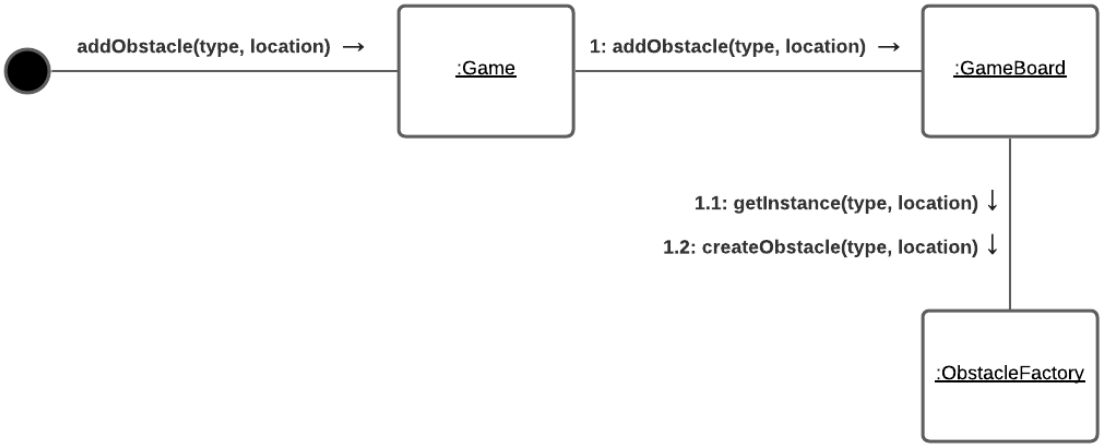


Communication Diagrams

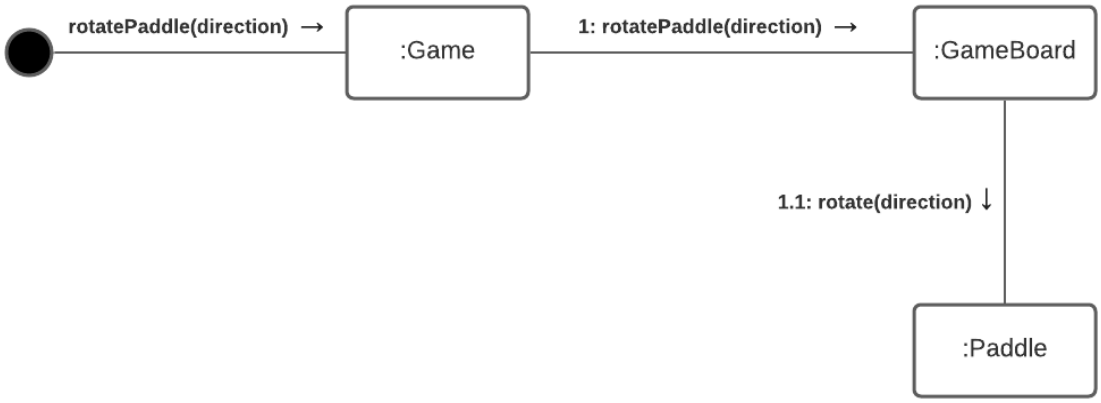
CD.1 : Pause Game



CD.2: Add Obstacle

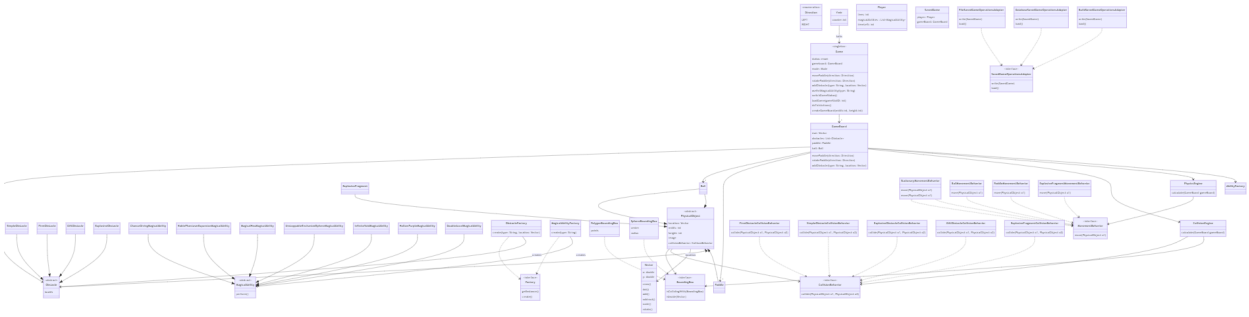


CD.3: Rotate Paddle

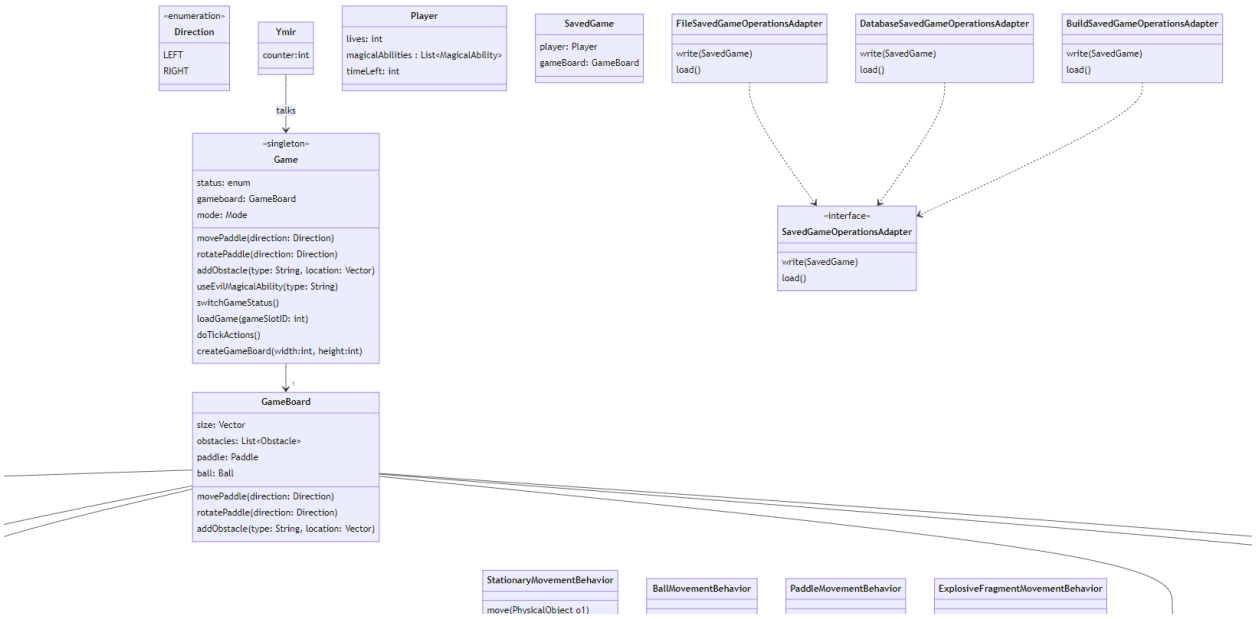


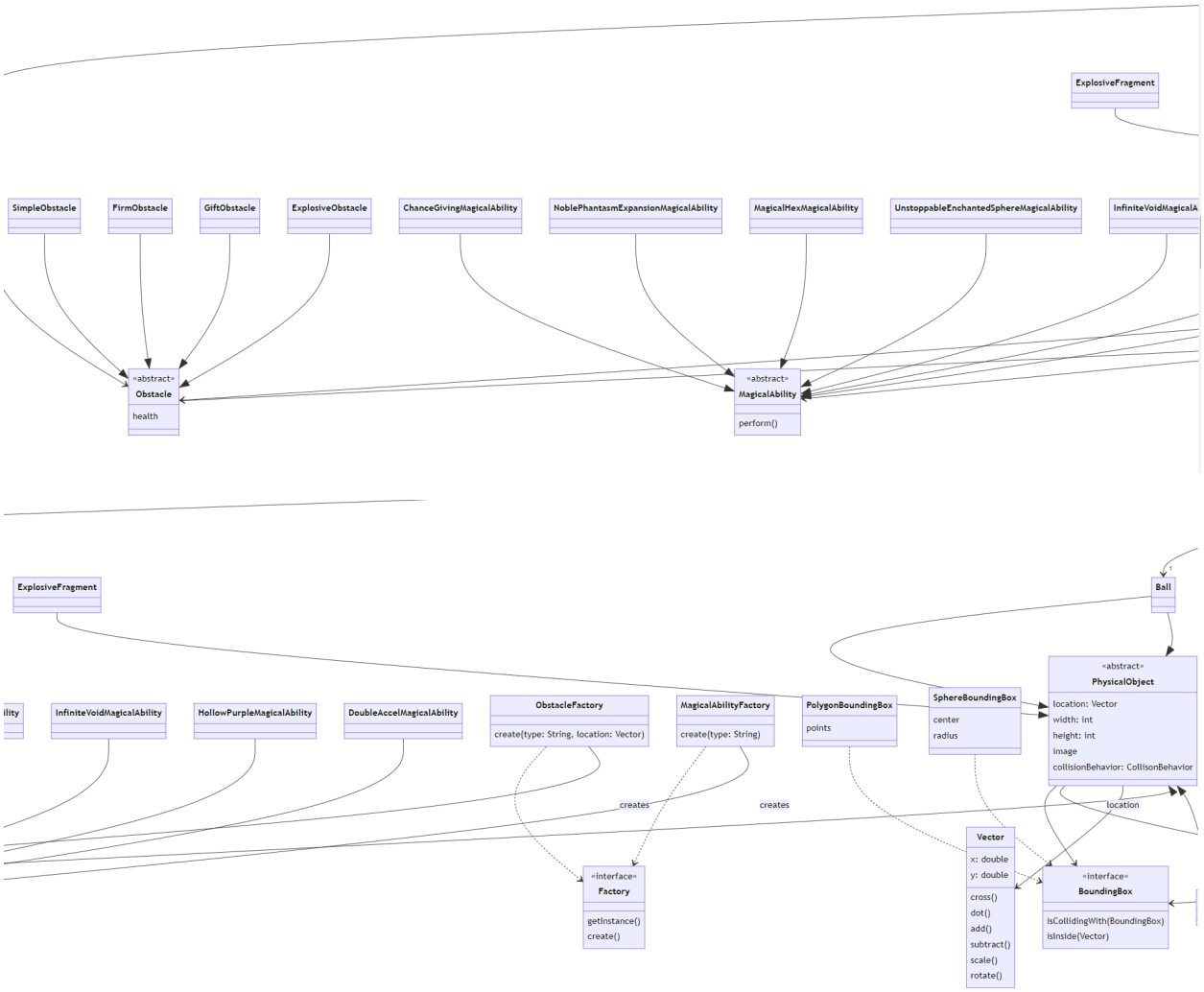
Class Diagrams

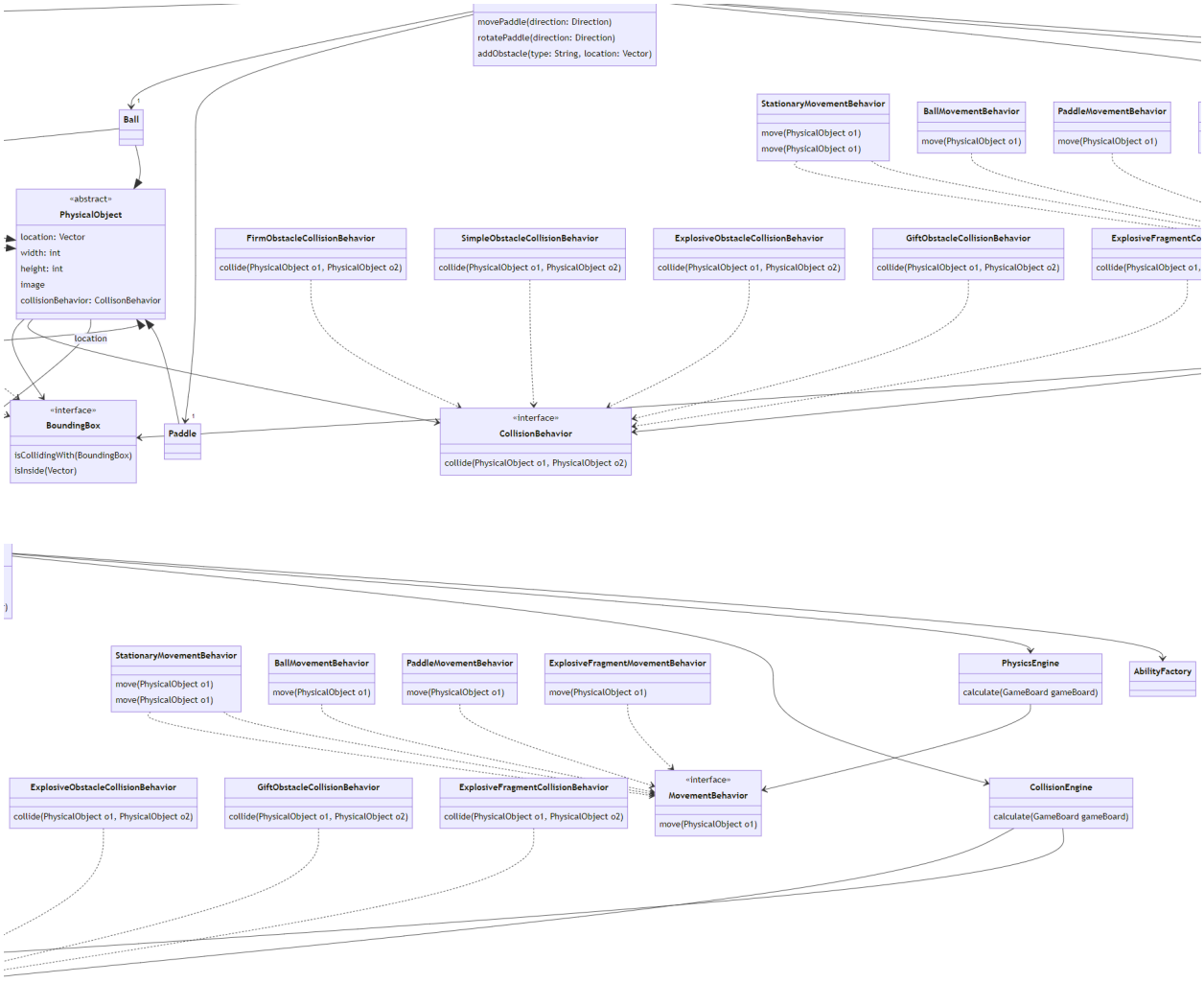
The Diagram



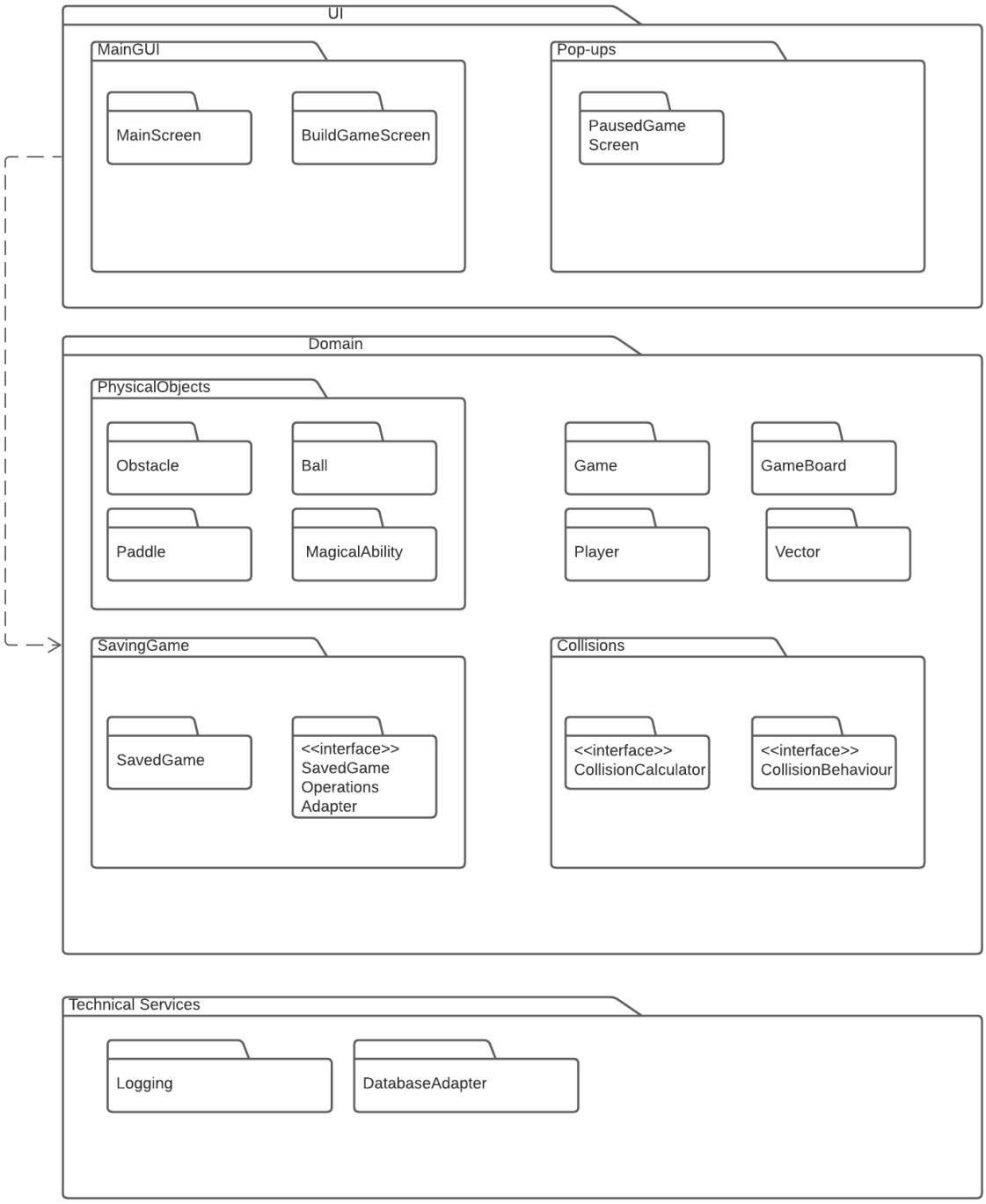
Closer Look




ExplosiveFragment
SimpleObstacle
FirmObstacle
GiftObstacle
ExplosiveObstacle
ChanceGivingMagicalAbility
NoblePhantasmExpansionMagicalAbility
MagicalHexMagicalAbility
UnstoppableEnchantedSphereMagicalAbility
InfiniteVoidMagicalA
«abstract»
Obstacle
health
«abstract»
MagicalAbility
perform()
ExplosiveFragment
Ball
InfiniteVoidMagicalAbility
HollowPurpleMagicalAbility
DoubleAccelMagicalAbility
ObstacleFactory
create(type: String, location: Vector)
MagicalAbilityFactory
create(type: String)
PolygonBoundingBox
points
SphereBoundingBox
center
radius
«abstract»
PhysicalObject
location: Vector
width: int
height: int
image
collisionBehavior: CollisonBehavior
creates
creates
location
«interface»
Factory
getInstance()
create()
Vector
x: double
y: double
cross()
dot()
add()
subtract()
scale()
rotate()
«interface»
BoundingBox
isCollidingWith(BoundingBox)
isInside(Vector)


movePaddle(direction: Direction)
rotatePaddle(direction: Direction)
addObstacle(type: String, location: Vector)
Ball
StationaryMovementBehavior
move(PhysicalObject o1)
move(PhysicalObject o1)
BallMovementBehavior
move(PhysicalObject o1)
PaddleMovementBehavior
move(PhysicalObject o1)
«abstract»
PhysicalObject
location: Vector
width: int
height: int
image
collisionBehavior: CollisonBehavior
FirmObstacleCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
SimpleObstacleCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
ExplosiveObstacleCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
GiftObstacleCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
location
«interface»
BoundingBox
isCollidingWith(BoundingBox)
isInside(Vector)
Paddle
«interface»
CollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
StationaryMovementBehavior
move(PhysicalObject o1)
move(PhysicalObject o1)
BallMovementBehavior
move(PhysicalObject o1)
PaddleMovementBehavior
move(PhysicalObject o1)
ExplosiveFragmentMovementBehavior
move(PhysicalObject o1)
PhysicsEngine
calculate(GameBoard gameBoard)
AbilityFactory
ExplosiveObstacleCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
GiftObstacleCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
ExplosiveFragmentCollisionBehavior
collide(PhysicalObject o1, PhysicalObject o2)
«interface»
MovementBehavior
move(PhysicalObject o1)
CollisionEngine
calculate(GameBoard gameBoard)

Package Diagrams



Discussion of Design Alternatives and Design Patterns and Principles

Controller Pattern

Our Game class is designed for getting all the information and interaction from UI and Ymir, then distributing it to all other domain objects. With this implementation, Game class becomes the one place in which UI and domain layers get in touch. This helps us to evaluate and decide how to distribute inputs of UI to domain. One advantage of this is that this reduces coupling between UI and domain layers as the interaction happens in one place. Another advantage is that this helps us to think how we will integrate the model view separation. A key point in using this pattern is to carefully assign the message received from the UI to responsible classes and avoid handling the tasks solely in Controller pattern as it can result in low cohesion or in some cases, result in high coupling with other classes. Alternative would be directly connecting UI classes to domain objects but It would increase the overall coupling.

Creator

We used this pattern and assigned responsibility of creating ball, obstacles, and paddle to GameBoard because GameBoard contains them and has initial knowledge to create them. Applying the creator pattern makes us think in terms of responsibilities and create instances of classes in the most relevant class. Even though in our project it was easier to identify which class should be based on the creator pattern, in some cases, if there are several candidate classes it may be difficult to choose the best one without violating high cohesion and low coupling principles.

Factory Pattern

For creating Obstacles, Magical Abilities, Observers, Borders we used factories. This made it possible to easily create those objects with a given type. An advantage of this pattern is that when we decide to improve the game or add new features to the classes which use factory design patterns, we were able to do so easily, since the desired features will only have to implement the interface class and its methods based on the features functionality. A minor disadvantage but worth mentioning in this approach may be the fact that while creating the objects if there are many different obstacles or magical abilities, the code will consist of an inevitable amount of if and else statements.

Singleton Pattern

As a team we planned our game project to have one Game object only, which will be used constantly through the instances of itself. Therefore, for the “Game” class as well as our factory classes ObstacleFactory and AbilityFactory we used the Singleton Pattern. This helps us achieve high cohesion and low coupling. Since those objects are accessed by many classes globally, singleton pattern made it easy to access them. However, one of the disadvantages of this approach is that later in our project we are going to use unit testing to test our code, and the singleton pattern makes it difficult to recognize dependency chains which leads to a more difficult unit testing process. Another disadvantage that of the singleton pattern is the underlying assumption that an object will be created only once. If we were to add

this game to an online platform where many people plays simultaneously, we would like to have more then one of this object to be able to response quickly. However, this pattern makes things complicated on this scenario.

Strategy Pattern

Collision handling is an important part of our project. However, it is not just a simple case in which we can apply the same logic in possible scenarios. Basically, our way of treating collisions may change in the presence of different magical abilities although the ways are closely related. Therefore, we decided to implement CollisionBehaviour following the strategy pattern. In this way we can have control over collision in the usage of magical ability differently.

We also use Strategy Pattern for the movement of the Physical Objects. Basically, in every game tick, our Physics Engine calls the move function the Movement Behavior of each Physical Object. This allows us to create one type of behavior once, and then use it multiple times: for example, we have StationaryMovementBehavior, which we used multiple times for stationary objects.

Adapter Pattern

For saving and loading the game, we realized that there might be different interfaces like from or to a file or database. Moreover, building a game or loading from a building mode can be treated as another interface. Although there are different options, they all have the same responsibilities. Therefore, we decided to use an adapter.

Low Coupling and High Cohesion

During all our discussions of design and how to assign responsibilities, low coupling and high cohesion are the two things we always have in consideration. We try to make our design to have low dependency, tolerance to change, and objects that are focused. We applied other patterns to reduce coupling and increase cohesion.

Supplementary Specifications

Revision History

Version	Date	Description	Author
Inception Draft	Oct. 29, 2021	First Draft. Will be defined primarily during elaboration.	SyntacticSugar
Phase 1	Oct. 31, 2021	Second draft of design and requirements and modeling.	SyntacticSugar
Phase 2	Dec. 19, 2021	Third and final draft of design and requirements and modeling	SyntacticSugar

Introduction:

This document is the repository for all Need For Spear requirements not captured in the use case narratives.

Functionality:

Logging and error handling: Log all errors to persistent storage.
Environment: The game should be platform independent.
Capacity: Single player.
Pluggable Rules:

Usability:

Human Factors: Players play the game on computer screen

1. All game components can be seen from 1 meter
2. Colors should be appropriate both for differentiating objects
and following the ball easily.
3. Avoid colors related with color blindness
4. Statistics and equipped features should be easily seen and followed

Reliability:

Recoverability:
If there is a failure due to the hardware or connection, the player may try to save the game and reload the game / restart the computer.

Performance:

The goal of our project is to present a game that can be completed, saved or loaded without any errors. Our aim is to build a game which can easily be played and that provides quick responses for all user inputs. Therefore, we will aim to achieve a simple but efficient UI for this game project.

Supportability:

Adaptability: It should work on the recent versions of supported OSs.

Purchased Components:

None.

Implementation Constraints:

Software will be built over JAVA using JAVA libraries.

Interfaces:

Hardware Interfaces: None.
Software Interfaces:

Glossary

Term	Definition and Information	Format	Validation Rules	Aliases
Warrior	The user			
Enchanted Sphere	The ball that player directs using paddle to hit obstacles	Graphical Object		Ball
L	Noble Phanstasm's length	A rational number that is equal to 10%*Screen width		
Noble Phantasm	The paddle that user moves to catch the enchanted sphere and to direct the enchanted sphere to hit obstacles. Can be moved horizontally and be rotated	Graphical Object Length : L Thickness: 20px Speed: *If the left or right arrow is pressed and released: The noble phantasm should move by an offset equal to L/2 with a speed of L/second.* *If the button is down, it should move with the speed of 2*L/second.*		Paddle
Wall Maria	One of 4 types of obstacles. Can be destroyed and disappear after one hit	Graphical object Speed:		
Steins Gate	One of 4 types of obstacles. Has specific number on it Can be broken and disappear after number of specified hits.	Graphical object		
Pandora's Box	One of 4 types of obstacles.	Graphical		

	Explodes after a hit and starts to fall towards Noble Phantasm. If it hits the paddle, user loses a chance	Object		
Gift of Uranus	One of 4 types of obstacles. Can be broken and disappear after one hit. After it is destroyed, drops a box towards the paddle. If paddle catch the box, box opens and a magical ability is earned	Graphical Object		
Score	The numerical value that will indicate the total of the points which the player earns throughout the game.	NewScore = OldScore + 300/(Current Time-GameStartingTime)		
Health	The numerical value of the remaining lives which the player has during gameplay.	Number		Lives, chance
Chance Giving Ability (Magical Ability)	A powerup which appears on the gameplay screen. It increases the player's chances by 1.	Graphical Object		
Noble Phantasm Expansion (Magical Ability)	A powerup which appears on the gameplay screen and that can be collected by the player to be used directly or kept for later use. It doubles the length of the Noble Phantasm.	Graphical Object	Once activated, it only lasts for 30 seconds.	T
Magical Hex (Magical Ability)	A powerup which appears on the gameplay screen and that can be collected by the player to be used directly or kept for later use. It adds two magical canons on both ends of the Noble Phantasm.	Graphical Object	Once activated, it only lasts for 30 seconds.	H

Unstoppable Enchanted Sphere (Magical Ability)	A powerup which appears on the gameplay screen. It makes the Enchanted Sphere so powerful that it breaks all kinds of obstacles and passews through them.	Graphical Object	Once activated, it only lasts for 30 seconds.	
Ymir	A sorcerer that will coin toss in every 30 seconds and if it is a success activate one of the 3 magical abilities (Infinite void, double accel, and hollow purple) to make game more difficult for player			
Infinite Void (Magical Ability)	A powerup which is activated by Ymir that choses 8 obstacles (if there are less obstacles in the game all of them) and freezes them.		Once activated, it only lasts for 15 seconds.	
Double Accel (Magical Ability)	A powerup which is activated by Ymir that speeds down the enchanted sphere by half.		Once activated, it only lasts for 15 seconds.	
Hollow Purple (Magical Ability)	A powerup which is activated by Ymir that puts 8 purple colored empty obstacles.		Once activated, it only lasts for 15 seconds.	

Appendix

Title Screen

Game Screen