KOÇ UNIVERSITY

# COMP 302: Software Engineering

# Final Report

# Fall 2022

# *OYUN*

## Group - 4:

Ömer Atasoy

Tuğra Demirel

Esmanur Eray

Ali Gebeşçe

Yakup Enes Güven

KOÇ UNIVERSITY

# Table of Contents

# A. INTRODUCTION AND VISION

**Revision History:**

| Version | Date | Description | Author |
|---|---|---|---|
| Draft | 25.10.2022 | First Draft. To be elaborated during revision | Esma Eray |
| Revision | 6.11.2022 | Revision. To be finalized | Ali Gebeşçe, Ömer Atasoy, Tuğra Demirel, Yakup Enes Güven |
| Finalization | 21.01.2023 | Finalization. Last version of the game | Esma Eray |

## 1. Introduction:

Escape From Koç is a fun game that takes place on a college campus. The player wanders around the faculty buildings level by level and tries to obtain hidden keys to eventually reach the last door in a limited time. The player has three lives which can be lost to randomly appearing aliens. There are multiple types of aliens which helps make the game more challenging. Escape From Koç also has entities like power ups and item bags just like some of the most popular games in the game industry. Randomly appearing power ups require the player to act fast in order to pick them which makes the game more interesting and eye-catching.

## 2. Positioning:

### 2.1 Business Opportunity:

Games that have the similar “escape from the room” and “searching for key” concepts are not new to the gaming industry. They are already widely played with various versions. However, a game specifically taking place in a college campus with an alien invasion theme is not common. Enriching the game with appealing graphics can attract attention of game players of all ages and gain the game a large popularity.

### 2.2 Problem Statement:

In order to make the game more outstanding than the games which have similar concepts, there needs to be features added to offer players a more exciting and challenging gaming experience with the element of competitiveness.

### 2.3 Product Position Statement:

Target group of Escape From Koç is people from all ages. Since the game is not very complicated and includes a help screen, it offers an opportunity to teach people with no gaming experience to be able to play the game. However, especially Koç students who like playing games and having fun are the main focus. Our game presents a desktop game with outstanding graphic features.

# B. TEAMWORK ORGANIZATION

The division of labor for both coding and documentation parts of the project differed from week to week. Since developing and documenting Escape from Koç was quite a long-term work, we gathered and had weekly meetings each week in order to both work together and assign individual assignments. Using the Git version control system helped us to keep in touch and progress synchronously. The duration of the project was 10 weeks.

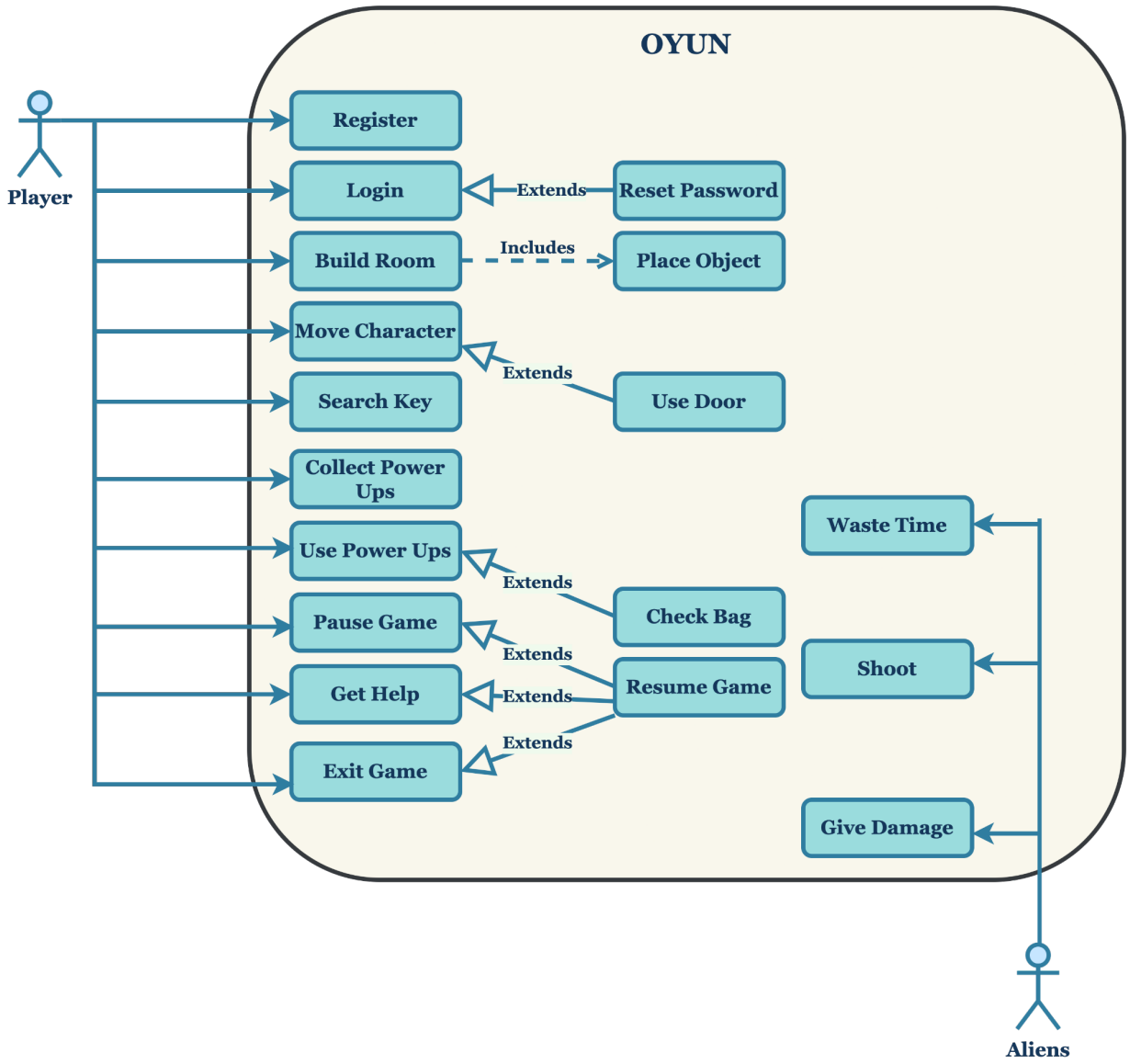
In the first 3 weeks, we met to analyze and understand the requirements of the project. The team worked together in order to complete Use Case Narratives and also prepare drafts for Use Case Diagrams, System Sequence Diagrams, Domain Model, UML Package Diagrams, UML Sequence Diagrams and UML Communication Diagrams.

In week 4, we completed UML Sequence and Communication Diagrams. Tuğra completed the Register and Login part. Ali completed Build Room and Move Character. Esma completed Pause Game. Ömer completed Collect Power Up. Yakup Completed Get Help and Exit Game.

The rest of the weeks, the focus shifted more on the actual coding. We helped each other when bugs were encountered and when we had trouble implementing our parts.

| Person | Contribution |
|---|---|
| Tuğra | Database Setup, Register/Login, Main Menu UI, Drawer with Adapter Pattern, ThrownBottle starter code, tryRegister test, Save/Load Game with Adapter Pattern |
| Ali | Room, BuildingObject, Register/Login UI, Pause/Top Panel UI, Bottle, BuildingObject Factory, ThrownBottle animation starter, isKeyFound and Bag class tests, BuildingMode |
| Esma | Pause/Resume, Model-View Separation enhancement, Bag UI implementation, Singleton fix, tryLogin test, general bug fixes |
| Ömer | Controllers, Timer, Model-View Separation enhancement, Protection Vest, PowerUp Singleton, ThrownBottle UsePowerUp test, ThrownBottle finalization, Bag class documentation |
| Yakup | Main Menu, Time/Health UI, Help Screen, Alien skeleton code, moveCharacter test |

# C.USE CASE DIAGRAM



# D.USE CASE NARRATIVES

## 1. Save Game

**Scope**: RunningModeScreen

**Level**: User Goal
**Primary Actor**: Player
**Stakeholders and interest:**
- Player: Wants to save their game state.

**Preconditions:**
- Application has successfully initialized.
- Player has logged in.
- Player has indicated his/her storage type when logging in.
- Game is in running mode.

**Success Guarantee:**
- Player may load their saved game later.

**Main Success Scenario:**
1. Player indicates that they want to save their game.
2. The BuildingObjects, Aliens, PowerUps, the Player's remaining health and time; the amount, position, and types of the relevant entities are saved to the player's choice of storage.

**Technology and Data Variations List:**
- 1a. Player indicates that they want to save their game via mouse or touchpad.

**Frequency of Occurrence:** Whenever a player wants to save their game.
**Open Issues:** None

## 2. Load Game

**Scope**: Main Menu
**Level**: User Goal
**Primary Actor**: Player
**Stakeholders and interest:**
- Player: Wants to retrieve their game state.

**Preconditions:**
- Application has successfully initialized.
- Player has logged in.
- Player has indicated their storage type when logging in.
- Game is in the main screen.

**Success Guarantee:**
- Player resumes their game from where they were left off.

**Main Success Scenario:**
1. Player indicates that they want to load their saved game.

2. The BuildingObjects, Aliens, PowerUps, the Player's remaining health and time; the amount, position, and types of the relevant entities (and possibly other data regarding the game state) are loaded from the player's choice of storage.
3. The game starts in the game screen where the player left off.

**Extensions:**

- 1.a. If the player does not have a saved game available, the game starts from scratch.

**Special Requirements:** Existence of a saved game structure in the database/file.

**Technology and Data Variations List:**

- 1a. Player indicates that they want to load their saved game via mouse or touchpad.

**Frequency of Occurrence:** Whenever a player wants to load a saved game.

**Open Issues:** None

# 3. Waste Time

**Scope**: Running Mode

**Level**: Subfunction

**Primary Actor**: Time-Wasting Alien (System)

**Stakeholders and interest:**

- Player: Will add a challenge to the game since the key will be harder to find.

**Preconditions:**

- Player has reached the Running Mode.
- A Time-Wasting Alien exists.

**Success Guarantee:**

- Changes the location of the key every 3 seconds, once, or never depending on the remaining time of the player.

**Main Success Scenario:**

1. The Time-Wasting Alien places the key under a random object in the Room.
2. The Time-Wasting Alien waits 3 seconds and goes back to 1.

**Extensions:**

- 1.a. If the player has between 70% and 30% of their time remaining, the Time-Wasting Alien does not change the location of the key and disappears after 2 seconds.
- 2.a. If the player has less than 30% of their time remaining, the Time-Wasting Alien only changes the location of the key once and it disappears.

**Special Requirements:** None

**Technology and Data Variations List:**

- None

**Frequency of Occurrence:** Whenever a Time-Wasting Alien exists in the Running Mode

**Open Issues:** None

# 4. Use Power Up

**Scope**: Running Mode
**Level**: User Goal
**Primary Actor**:Player
**Stakeholders and interest:**

- Player: Wants to use a storable power up.

**Preconditions:**

- Game is in running mode.
- Player has time.
- Player has health.
- The bag contains at least one power up if the power up is storable.

**Success Guarantee:**

- Initiate the effect of the power up. Number of items in the bag decreases by one if the power up is storable..

**Main Success Scenario:**

1. Player states that they want to use a power up.
2. The number of power ups in the bag decreases.

**Extensions:**

- 1a. The player presses H (selected power up is Hint):
    1. A region containing the key is highlighted for 10 seconds.
- 1b. The player presses P (selected power up is Protection Vest):
    1. Player is protected from the Shooter Alien for 20 seconds.
- 1c. The player presses B (selected power up is Plastic Bottle):
    1. Player presses a key to throw the Plastic Bottle:
        - 1a. Player presses A:
            1. Player throws the plastic bottle to the West.
        - 1b. Player presses D:
            1. Player throws the plastic bottle to the East.
        - 1c. Player presses W:
            1. Player throws the plastic bottle to the North.
        - 1d. Player presses X:
            1. Player throws the plastic bottle to the South.

**Special Requirements:**

- Plastic Bottle Fools the blind alien. Once used, the Blind Alien moves to the Plastic Bottle's destination before going back to moving randomly.

**Technology and Data Variations List:**

- 1a. Player uses power ups via keyboard or mouse.

**Frequency of Occurrence:** Whenever a player wants to use power up.

**Open Issues:** None

# 5. Build Room

**Scope**: Building Mode
**Level**: User Goal
**Primary Actor:** Player
**Stakeholders and interest:**

- Player: Wants to build a map to play.

**Preconditions**:

- Player has successfully logged in.
- Game is not in the running mode.

**Success Guarantee:**

- A room is built for the running mode.

**Main Success Scenario:**

1. Player chooses a BuildingObject.
2. Player performs the Place Object.
3. Player clicks the "Next" button.

**Extensions:**

- 2a. There is a collision.
    1. Reject object placement.
    2. Player starts from step 2.
- 3a. The minimum object requirement is not satisfied.
    1. Prompt a message stating that "add more objects"
- 3b. Player is building SNA
    1. Prompt a message stating that "SNA is the last building of the map, do you want to start playing the game"
    - 1a. Player clicks "yes" button
        1. Game goes into running mode
    - 1b. Player click "no" button
        1. Wait in building mode

**Special Requirements:**

- If there is an intersection of objects immediately returns to its old location via moving animation.

**Technology and Data Variations List:**

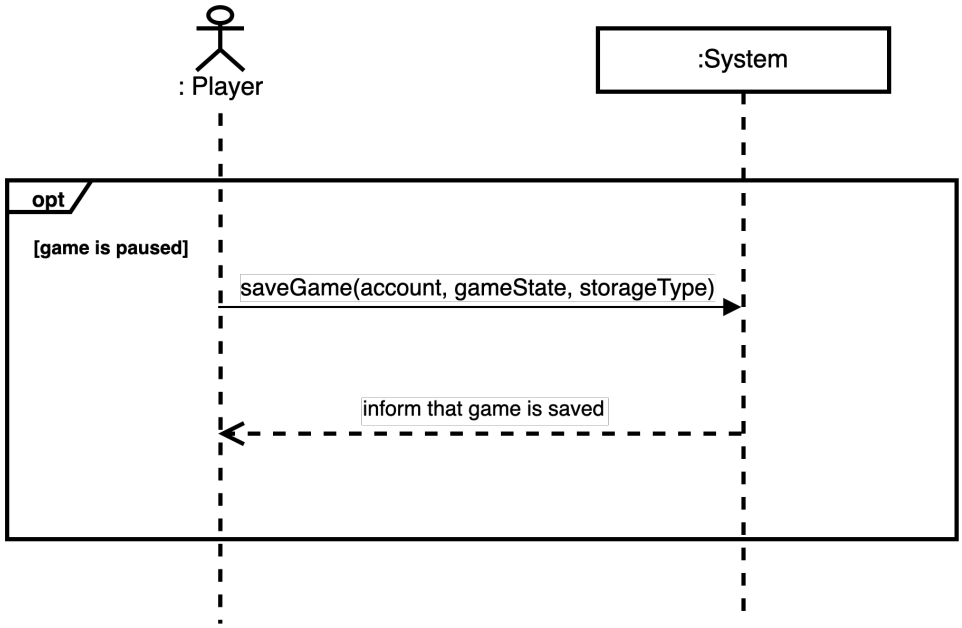
- 2a. Player clicks the BuildingObject and drag it to a valid location via mouse or touchpad.
- 3a. Player can activate the Next button via "the enter" key.

**Frequency of Occurrence:** Whenever a player wants to build a map.

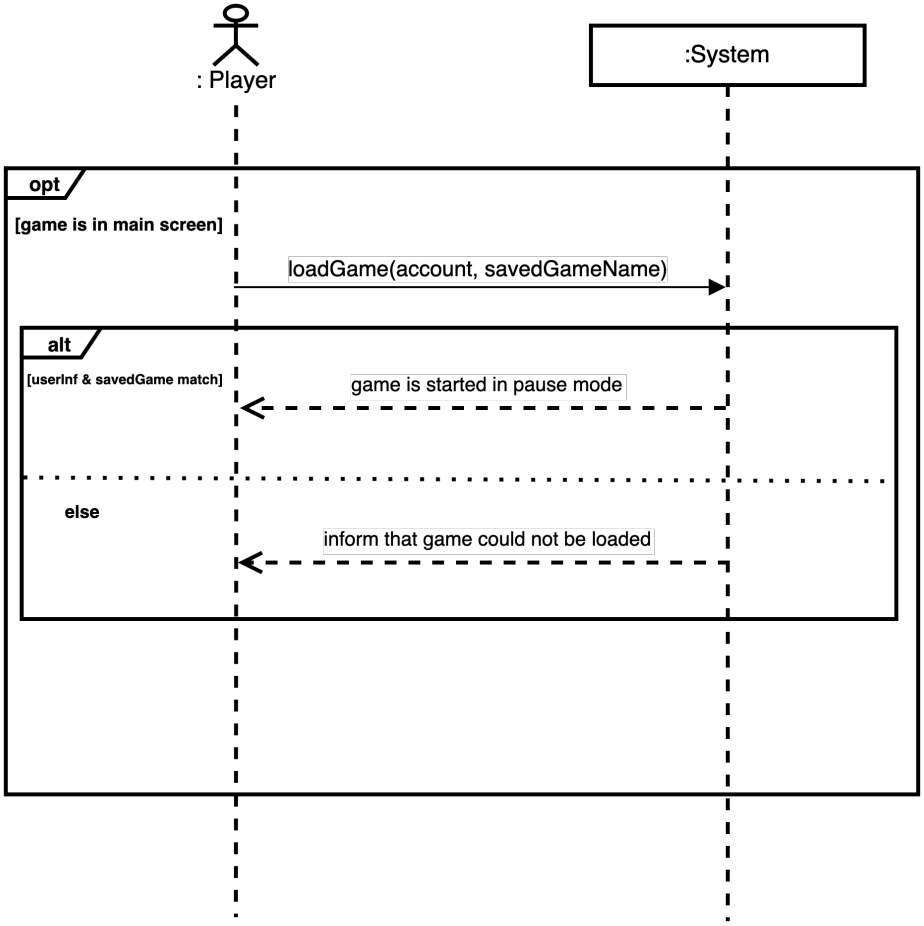
**Open Issues:** None

# E.SYSTEM SEQUENCE DIAGRAMS

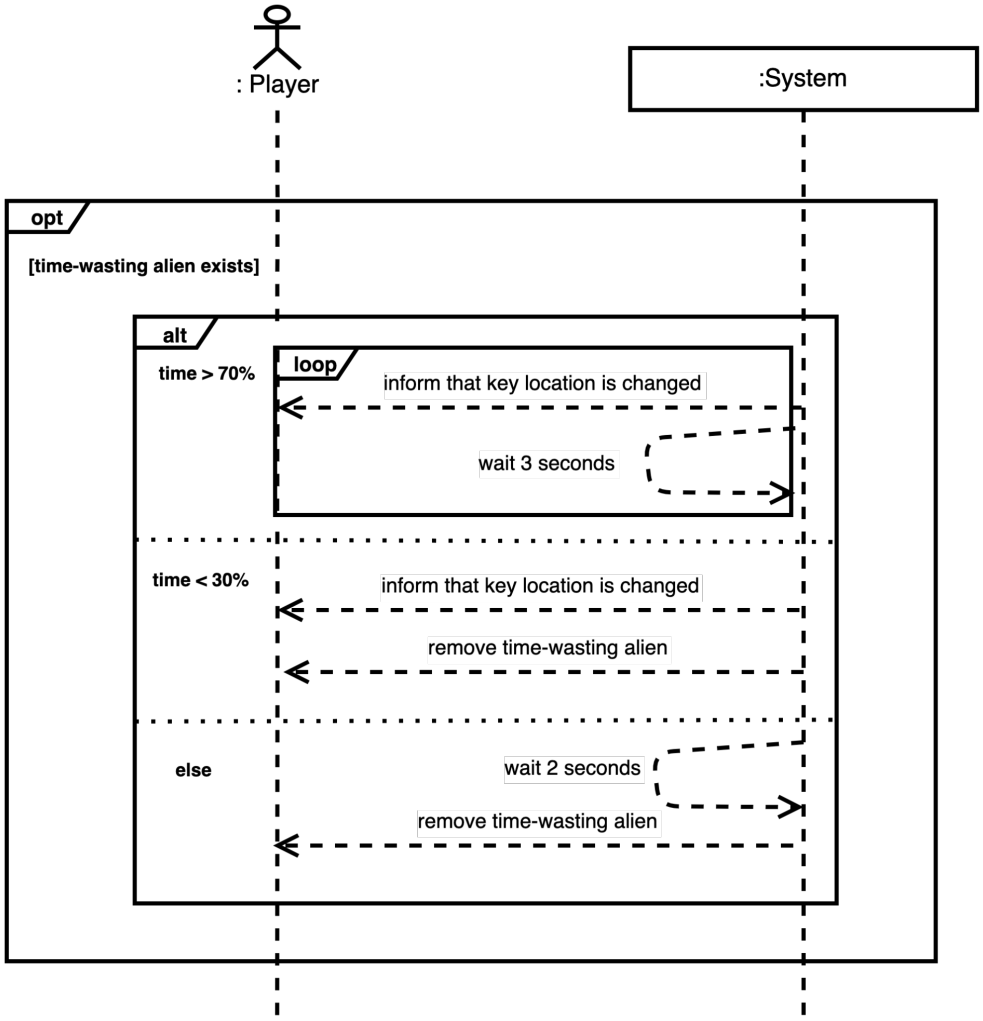
## 1. Save Game



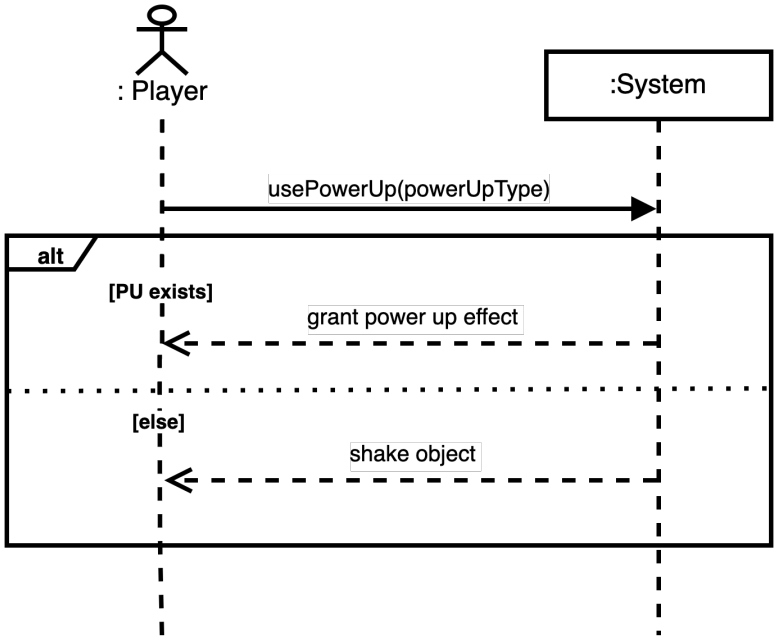
## 2. Load Game


: Player
:System
opt
[game is in main screen]
loadGame(account, savedGameName)
alt
[userInf & savedGame match]
game is started in pause mode
else
inform that game could not be loaded

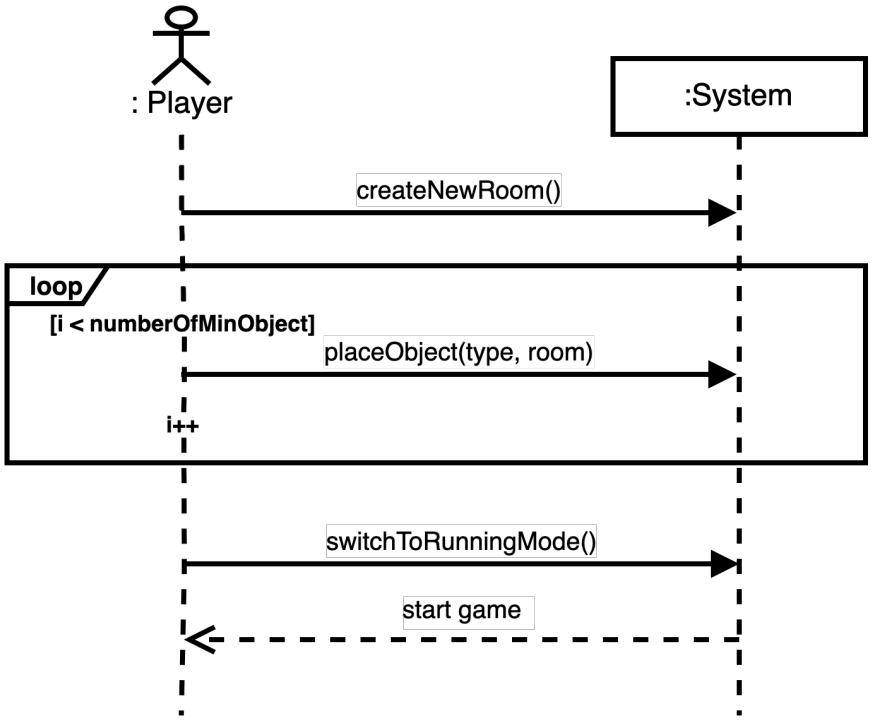
# 3. Waste Time


: Player
:System
opt
[time-wasting alien exists]
alt
time > 70%
loop
inform that key location is changed
wait 3 seconds
time < 30%
inform that key location is changed
remove time-wasting alien
else
wait 2 seconds
remove time-wasting alien

## 4. Use Power Up




## 5. Build Room



# F.OPERATION CONTRACTS

## 1. Contract CO1: saveGame

**Operation:** saveGame(account, gameState, storageType)
**Cross References:** UC1: Save Game
**Preconditions:**

- Application has successfully initialized.
- Player has logged in.
- Player has indicated his/her storage type when logging in.
- Game is paused.

**Postconditions:**

- Account.gameState was updated. (attribute modification)

## 2. Contract CO2: loadGame

**Operation:** loadGame(account, savedGameName)
**Cross References:** UC2: Load Game
**Preconditions:**

- Application has successfully initialized.
- Player has logged in.
- Player has indicated his/her storage type when logging in.
- Game is in the main screen.

**Postconditions:**

- Game.mode became “pause_mode”(attribute modification)
- Instances of the BuildingObjects, Aliens, PowerUps, the Player’s remaining health and time, Position, etc. was created (instance creation)
- sample_room was associated with e BuildingObjects, Aliens, PowerUps, (association formed)
- sample_player was associated with the Position and Bag (association formed)

## 3. Contract CO3: wasteTime

**Operation:** wasteTime()
**Cross References:** UC3: Waste Time
**Preconditions:**

- Game is in running mode.
- Player has time.
- Player has health.
- A Time-Wasting Alien exists.

- Key has not been found yet.

**Postconditions:**

- Key location will change.

# 4. Contract CO4: usePowerUps

**Operation:** usePowerUp()
**Cross References:** UC4: Use Power Ups
**Preconditions:**

- Game is in running mode.
- Player has time.
- Player has health.
- The bag contains at least one power up or non storable power up collected.

**Postconditions:**

- Power Ups on the bag will decrease if power up is storable.
- Power Ups used.

# 5. Contract CO5: createNewRoom

**Operation:** createNewRoom()
**Cross References:** UC5: Build Room
**Preconditions:**

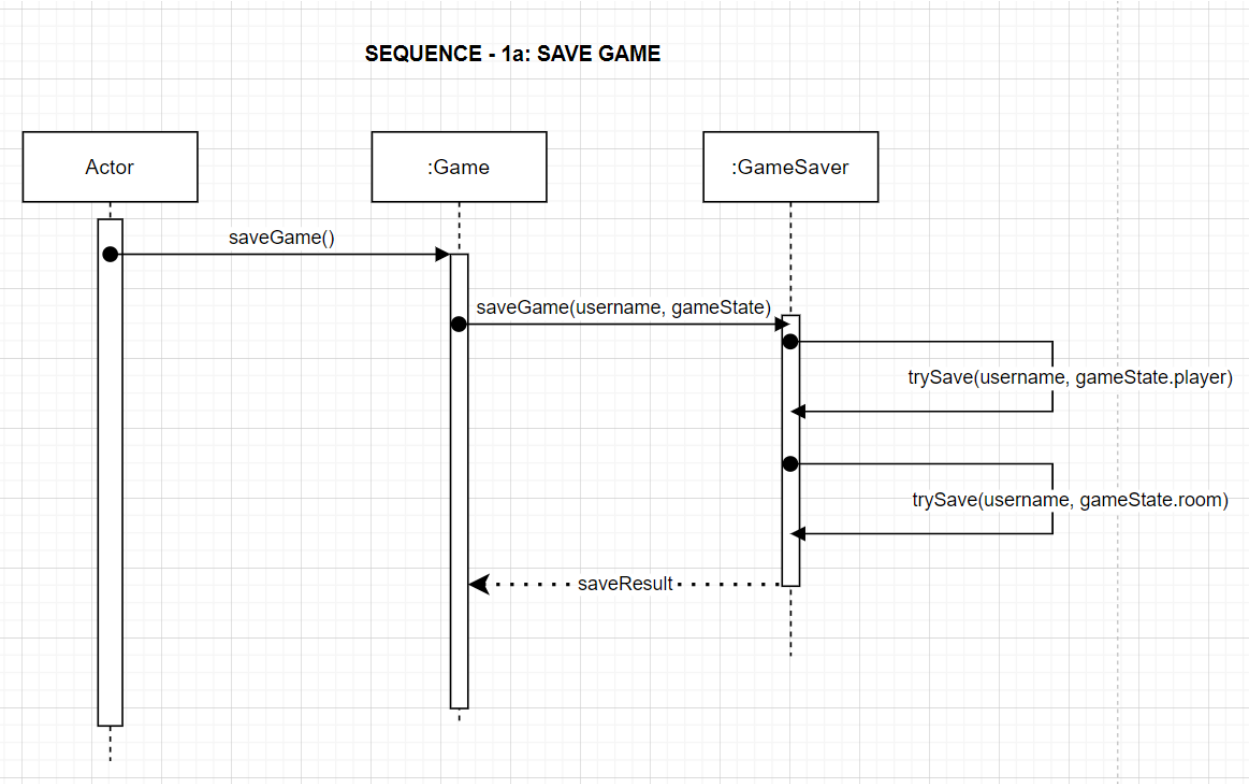
- Game is in building mode.

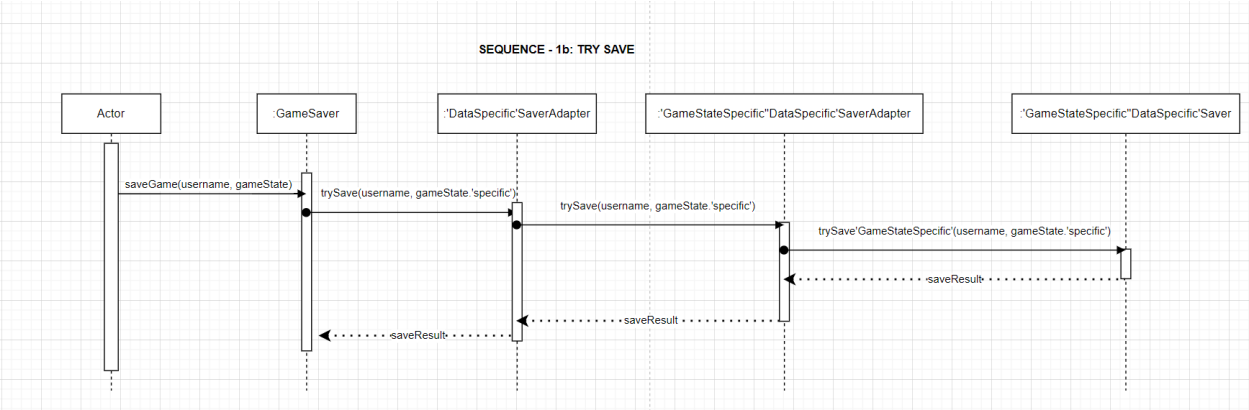
**Postconditions:**

- A Building instance “sample_building” was created (instance creation).
- sample_building was associated with the current Game (association formed).
- sample_game.mode became “building_mode” (attribute modification)

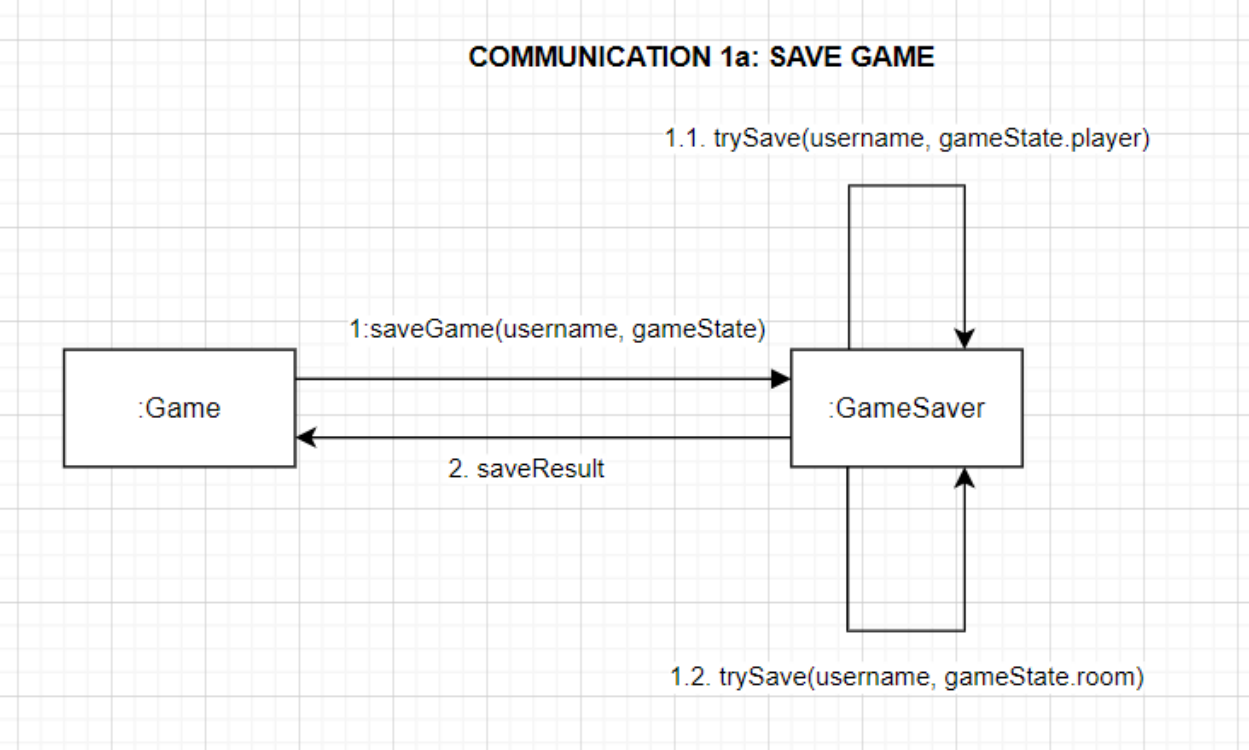
# G.INTERACTION DIAGRAMS

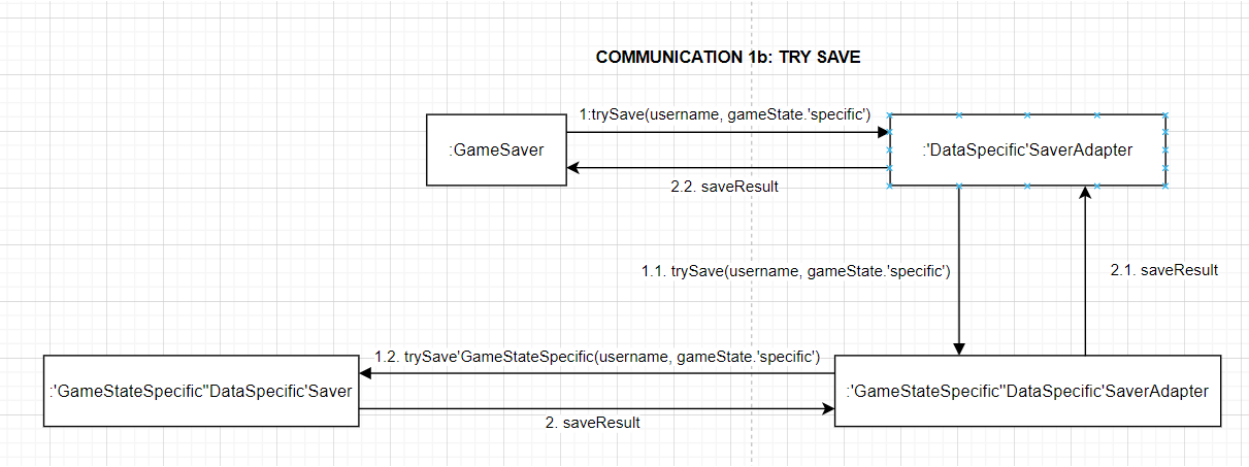
## 1. Sequence 1: saveGame/trySave



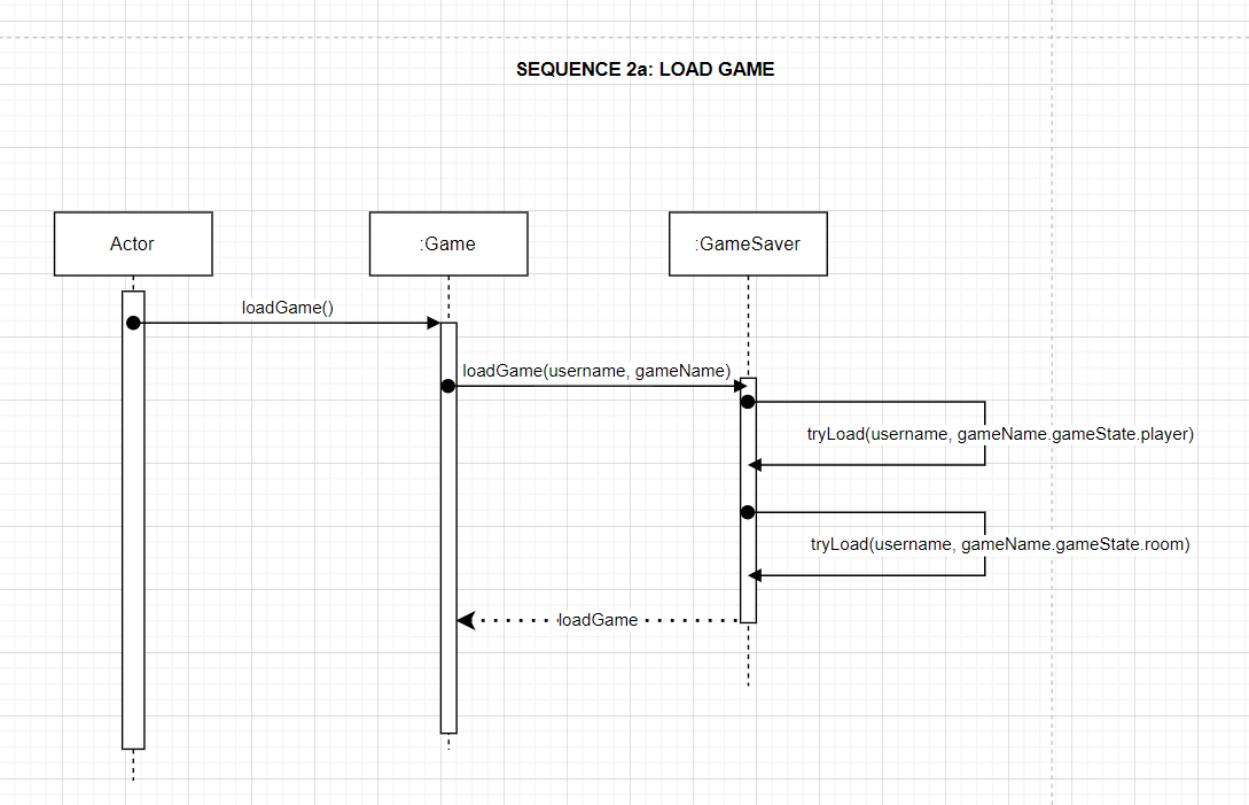



## 2. Communication 1: saveGame/trySave

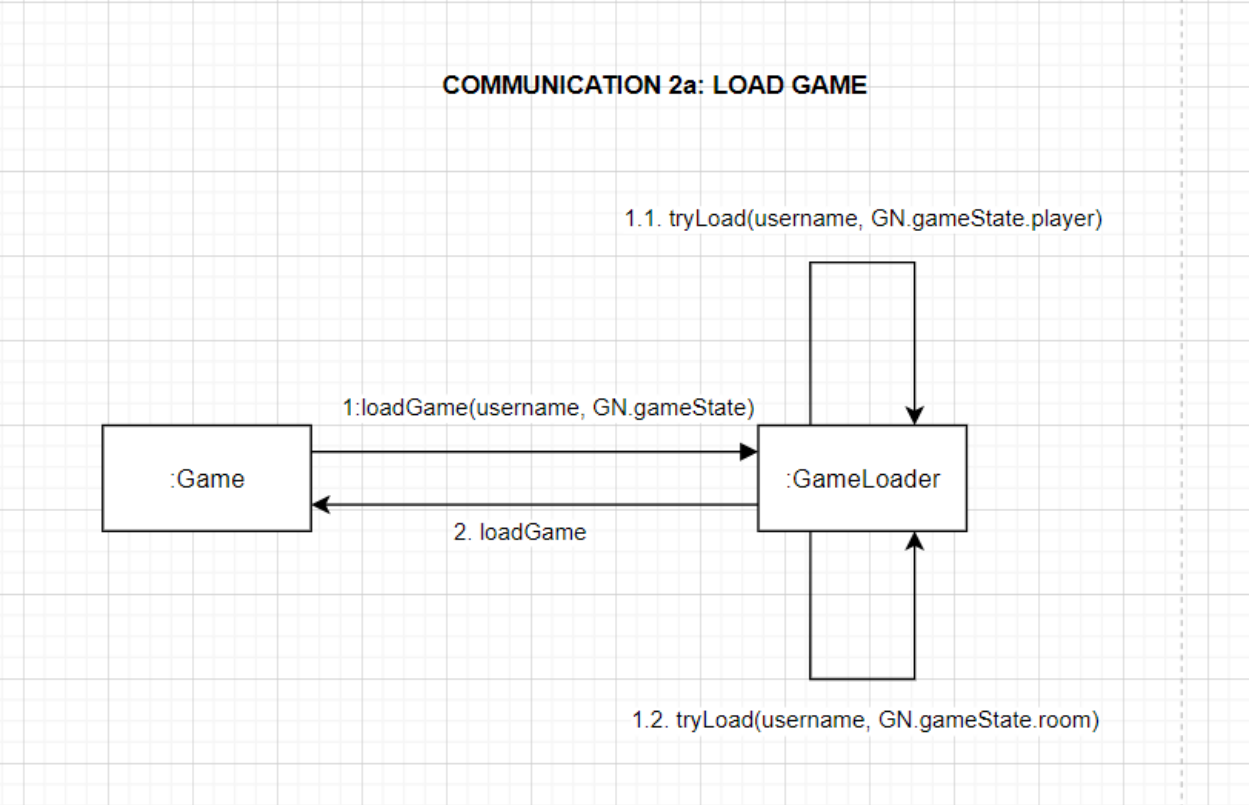


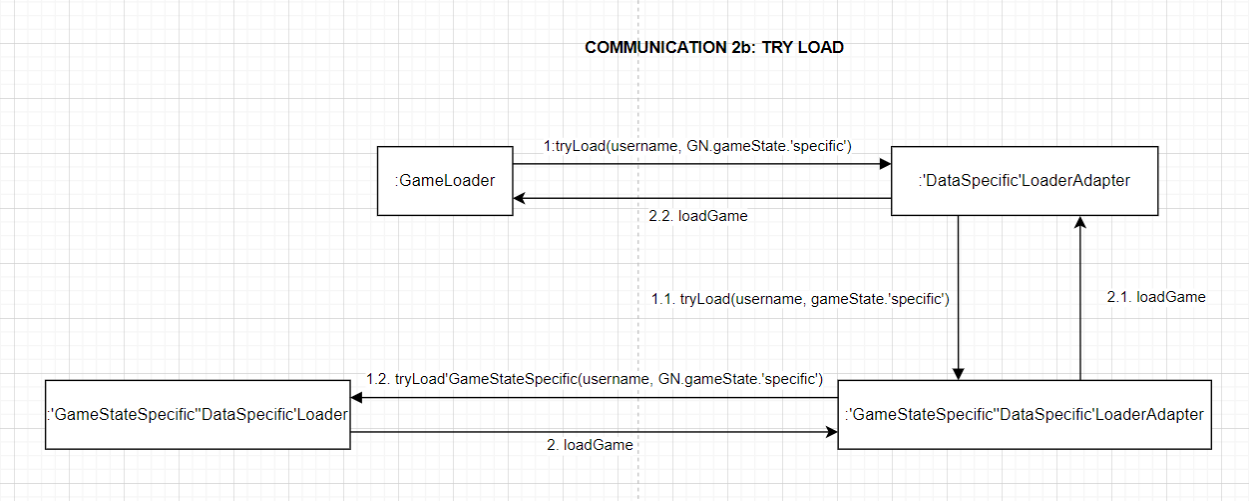



# 3. Sequence 2: loadGame/tryLoad

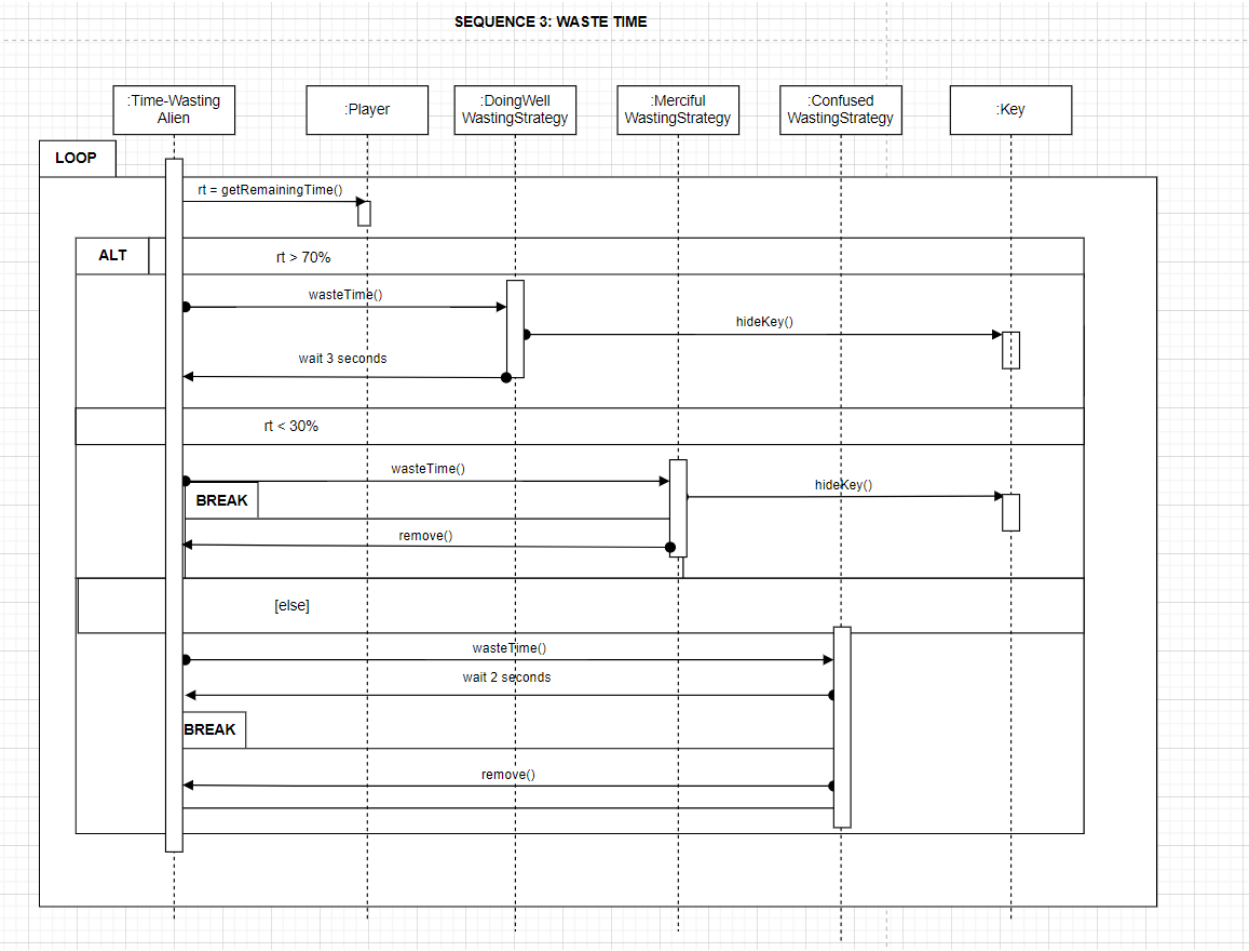


# 4. Communication 2: loadGame/tryLoad

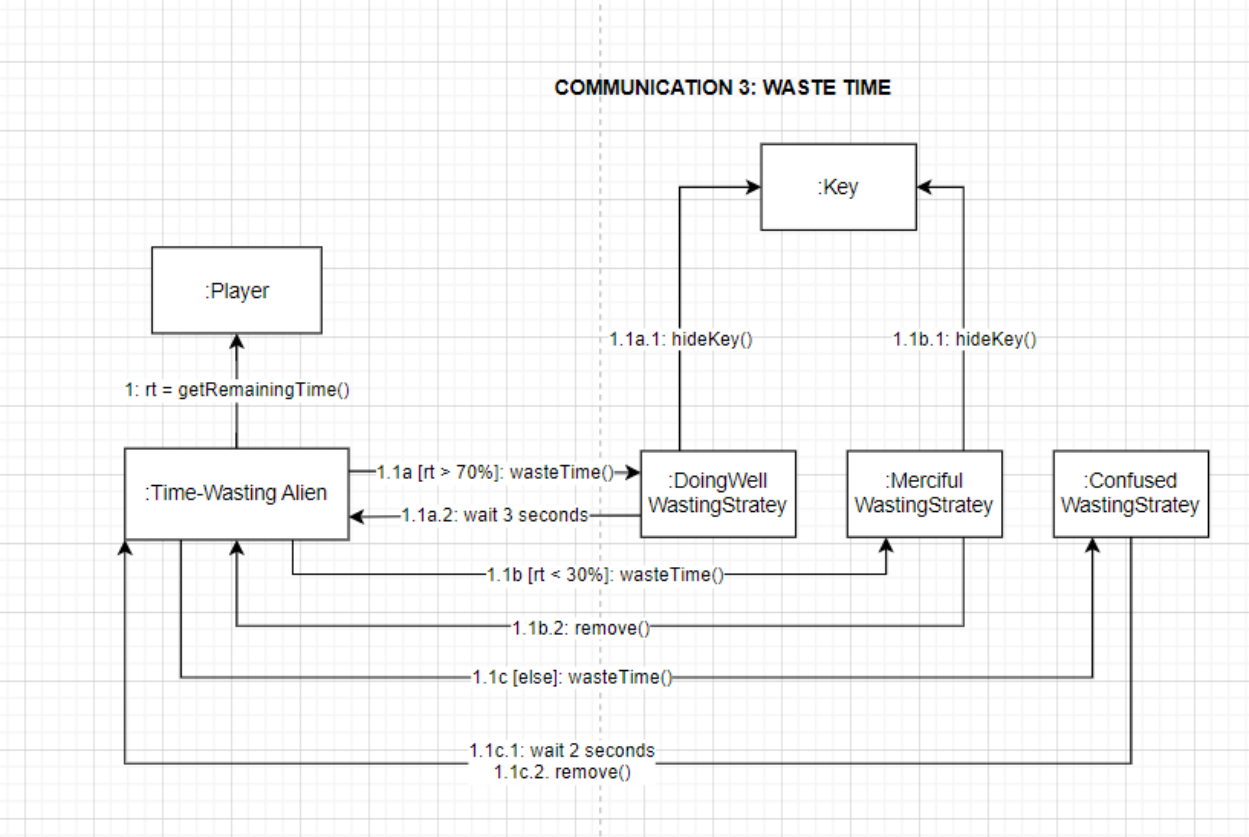


# 5. Sequence 3: wasteTime

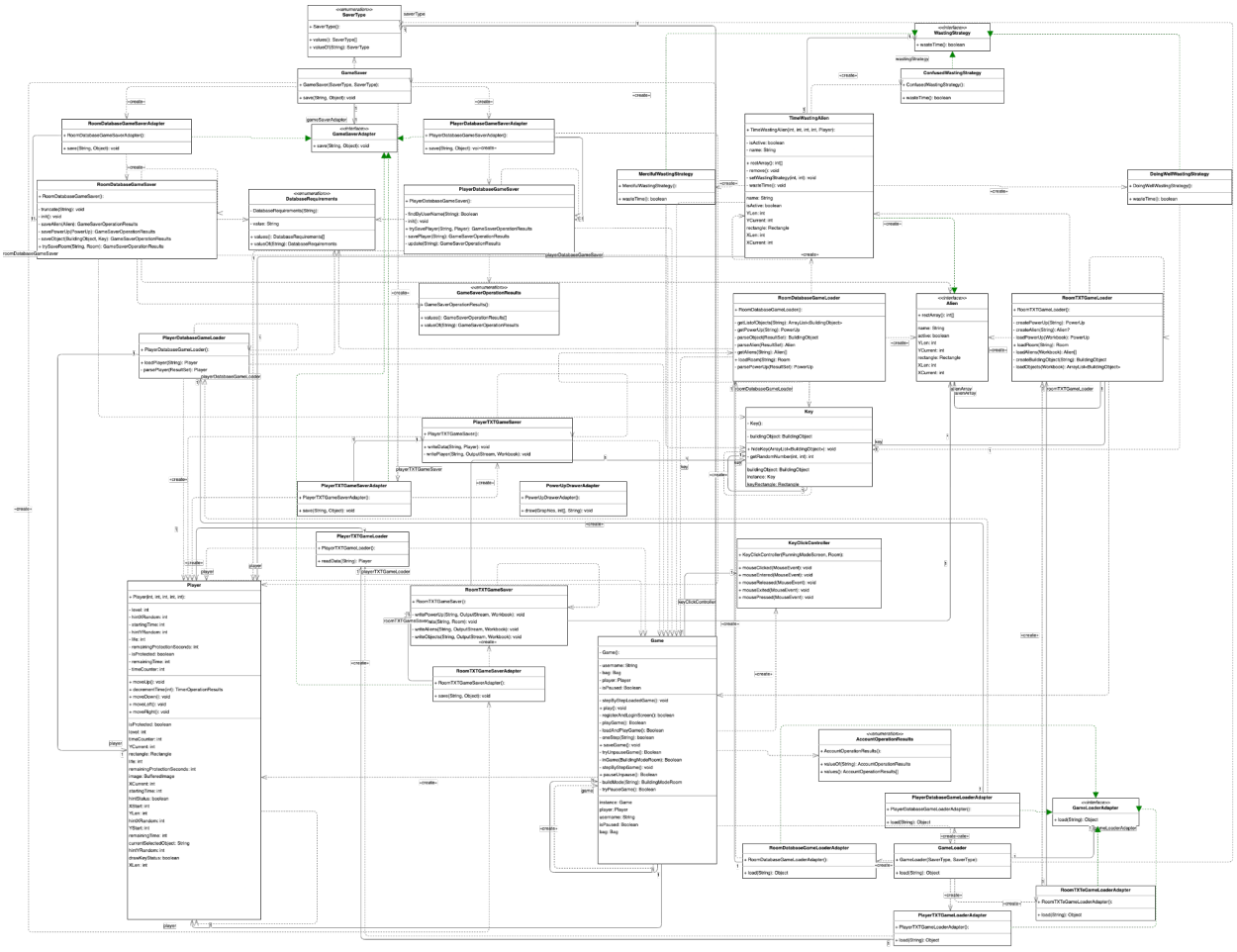


# 6. Communication 3: wasteTime

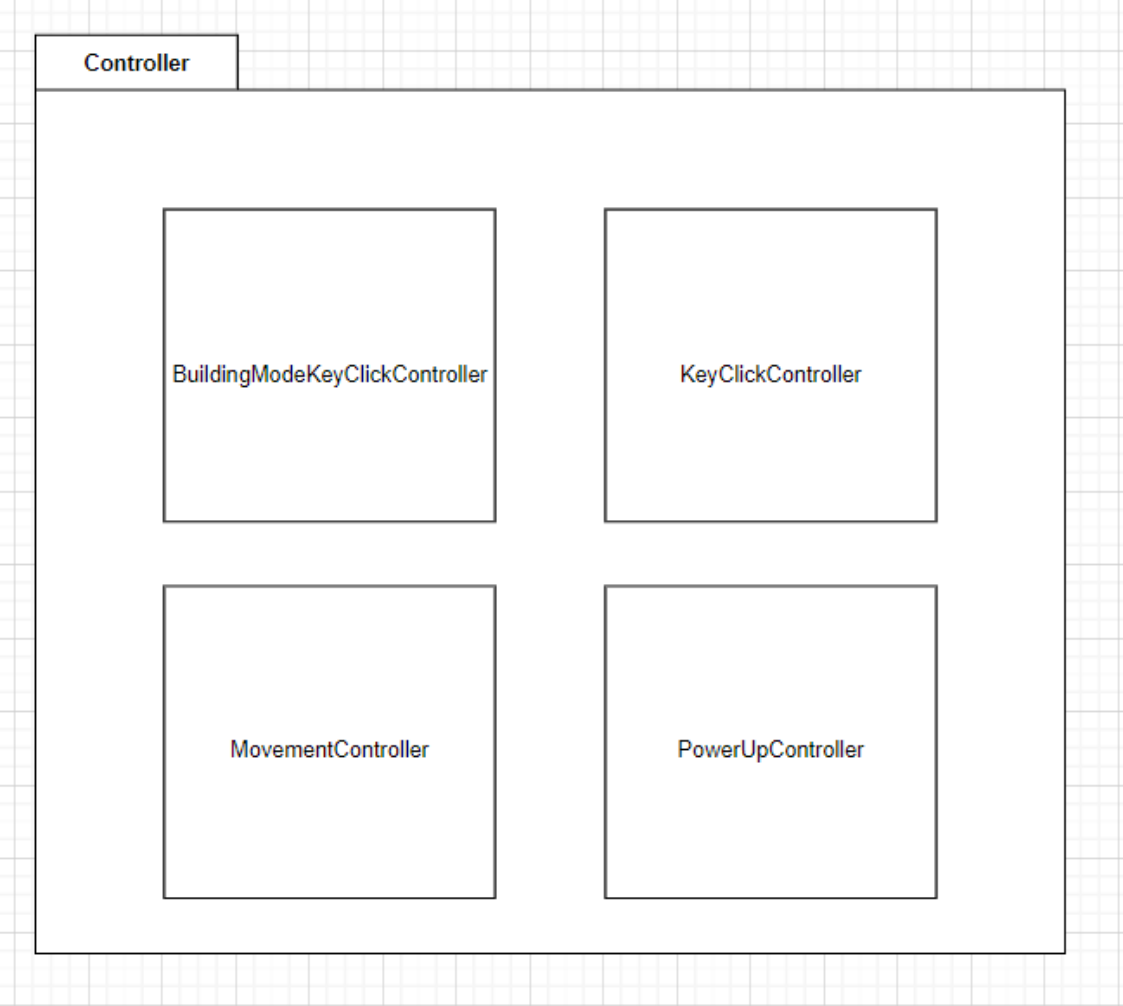


KOÇ UNIVERSITY

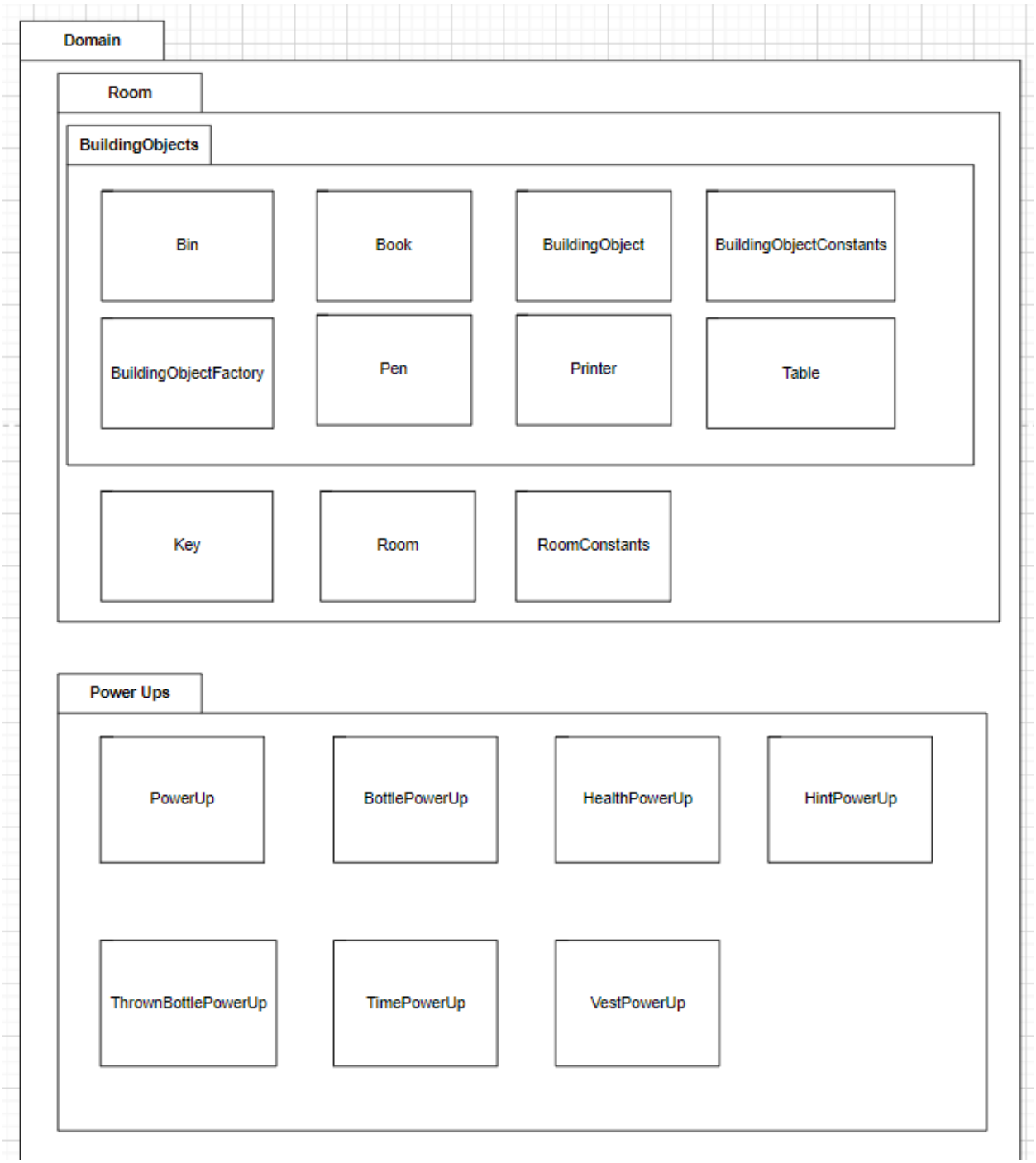
# H.CLASS DIAGRAMS

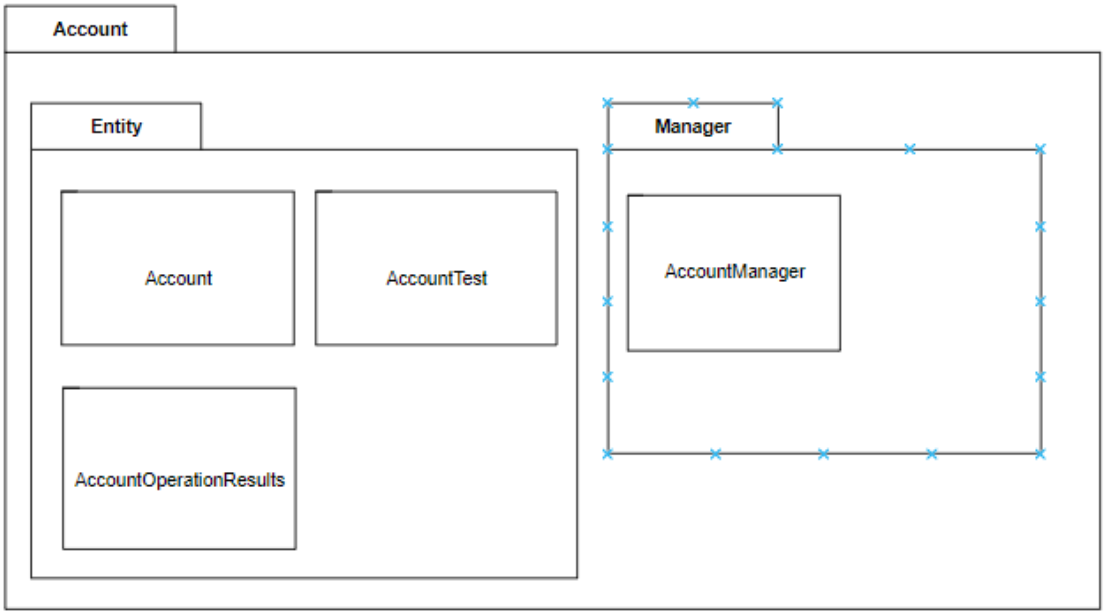


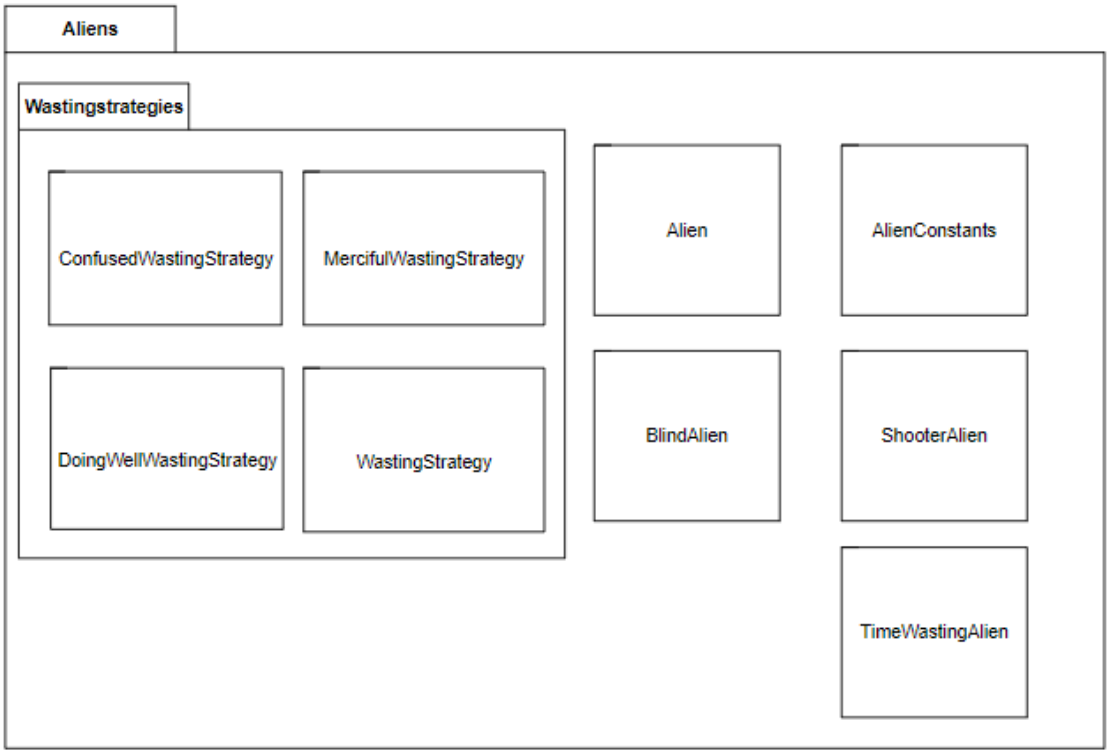
# I.PACKAGE DIAGRAMS

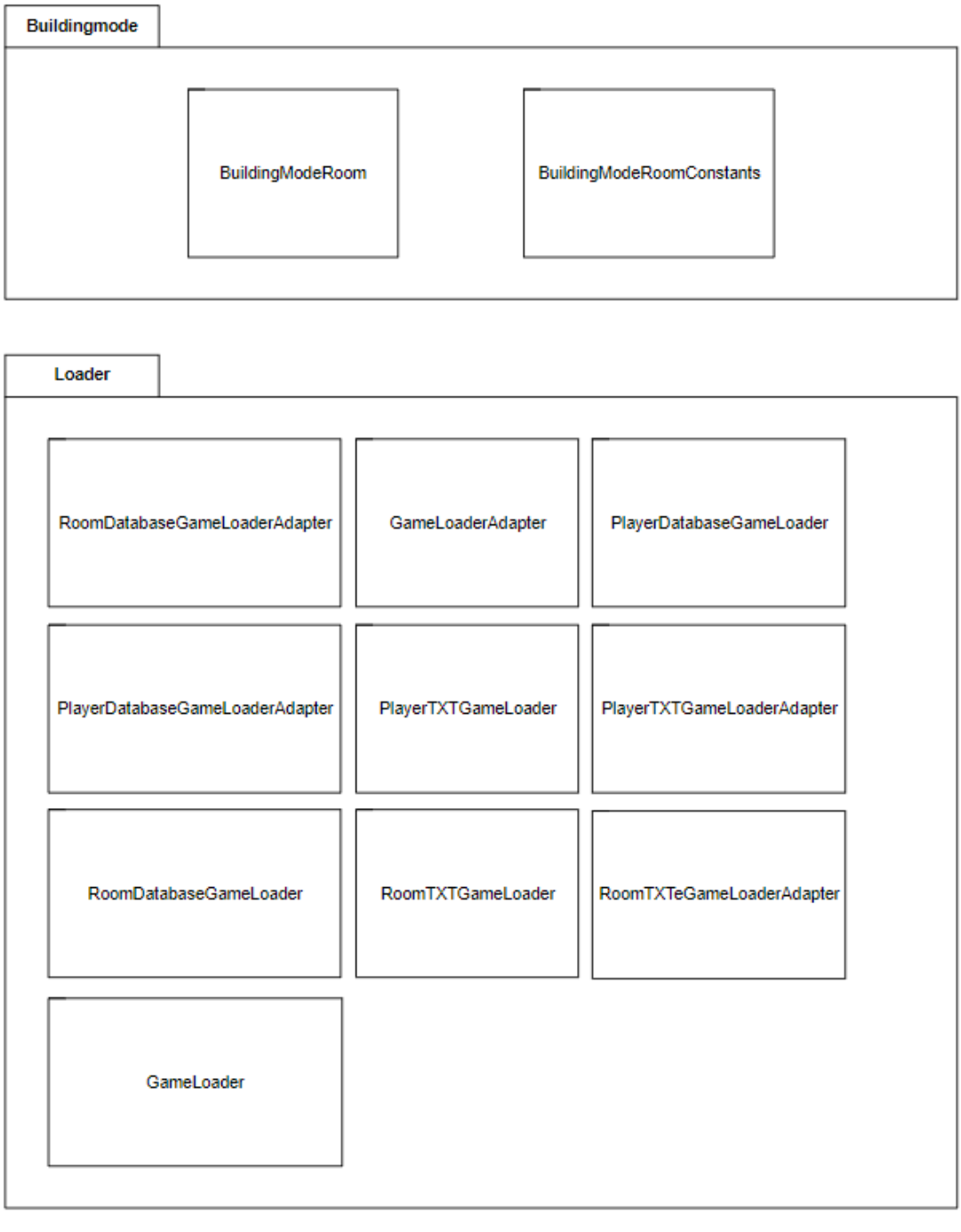



UI
Drawer
AlienDrawer
AlienDrawerAdapter
Drawer
DrawAdapter
ObjectDrawer
ObjectDrawerAdapter
PowerUpDrawer
PowerUpDrawerAdapter
GameScreen
BuildingModeScreen
HelpScreen
LoginScreen
ImageLoader
MainMenuScreen
PauseAndResumeScreen
RunningModeScreen
ScreenMaker


Domain
Room
BuildingObjects
Bin
Book
BuildingObject
BuildingObjectConstants
BuildingObjectFactory
Pen
Printer
Table
Key
Room
RoomConstants
Power Ups
PowerUp
BottlePowerUp
HealthPowerUp
HintPowerUp
ThrownBottlePowerUp
TimePowerUp
VestPowerUp


Account
Entity
Account
AccountTest
AccountOperationResults
Manager
AccountManager


Aliens
Wastingstrategies
ConfusedWastingStrategy
MercifulWastingStrategy
DoingWellWastingStrategy
WastingStrategy
Alien
AlienConstants
BlindAlien
ShooterAlien
TimeWastingAlien


Buildingmode
BuildingModeRoom
BuildingModeRoomConstants
Loader
RoomDatabaseGameLoaderAdapter
GameLoaderAdapter
PlayerDatabaseGameLoader
PlayerDatabaseGameLoaderAdapter
PlayerTXTGameLoader
PlayerTXTGameLoaderAdapter
RoomDatabaseGameLoader
RoomTXTGameLoader
RoomTXTeGameLoaderAdapter
GameLoader


Saver
GameSaver
GameSaverAdapter
GameSaverOperationResults
PlayerDatabaseGameSaverAdapter
PlayerDatabaseGameSaver
PlayerTXTGameSaver
PlayerTXTGameSaverAdapter
RoomDatabaseGameSaver
RoomDatabaseGameSaverAdapter
RoomTXTGameSaverAdapter
RoomTXTGameSaver
SaverType
Bag
Game
Levels
DatabaseRequirements
Main
Player
PlayerConstants
PlayerOperationsResults

# J.DISCUSSION OF DESIGN, PATTERNS, AND PRINCIPLES

## 1. Controller

a. Is the first object beyond the UI layer that takes care of an input. In our case, MovementController, KeyClickController are examples of this pattern.
b. MovementController handles movement of the Player by constantly listening to arrow keys. If the user interacts with arrow keys, MovementController first checks whether direction is valid, and then relocates the user accordingly.
c. KeyClickController handles the mouse clicks on the Room for searchKey and collectPowerUp.
d. Putting user input handling in a separate class makes development easier by separating the domain layer, UI layer, and controller layer responsibilities. It is easier to pinpoint bugs and add features.
e. We have used the controller pattern to handle isKeyFound and checkPowerUp user inputs as shown below:

```java
package com.gurup.controller;

import java.awt.Rectangle;

public class KeyClickController implements MouseListener {

    private final Room room;

    public KeyClickController(RunningModeScreen runningModeScreen, Room room) {
        this.room = room;
        runningModeScreen.addMouseListener(this);
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        int xMouse = e.getX();
        int yMouse = e.getY();
        Rectangle mouseRect = new Rectangle(xMouse, yMouse, 1, 1);
        if (e.getButton() == MouseEvent.BUTTON1) { // left click
            room.isKeyFound(mouseRect);
        } else if (e.getButton() == MouseEvent.BUTTON3) {
            room.checkPowerUp(mouseRect);
        }
    }
```

## 2. Adapter

a. We used this pattern for Drawing Building Objects and Power Ups. We "adapted" the original java swing draw function to ObjectDrawer and PowerUpDrawer as shown below.

```
public class Drawer {   Deanntu, Today • Revert "Merge branch 'main' into preprod"
    3 usages
    DrawAdapter drawAdapter;
    2 usages   Deanntu
    public Drawer( @NotNull String type) {
        if(type.equals("PowerUp")) {
            drawAdapter = new PowerUpDrawerAdapter();
        }
        else if(type.equals("Object")) {
            drawAdapter = new ObjectDrawerAdapter();
        }
    }
```

b. Also, we used Adapter pattern for saving and loading games. The game has two different states to be saved and loaded, states are called room and player. Also our game was expected to support two different types of data storages, database and txt file by user choice. Our plan was creating different objects to save by our data type and data storage type. Then we were faced with incompatible interfaces.

c. We have decided to use Adapter Pattern to save and load games because this pattern lets us save and load our specific state from the specific data storage. For example, the PlayerDatabaseSaverAdapter class adapts the interface of the data saving class to work with the player game state and a database saving location. In this case the game state is player and storage location is the database. Other adapters were PlayerTXTSaverAdapter, RoomDatabaseSaverAdapter and RoomTXTSaverAdapter. Overall, we used the Adapter pattern because it can be a useful tool for adapting the interface of an existing class to meet the needs of a specific client, without modifying the underlying class.

GameSaver.java

```java
package com.gurup.domain.saver;

public class GameSaver {
    GameSaverAdapter gameSaverAdapter;

    public GameSaver(SaverType type, SaverType state) {
        if (type.equals(SaverType.DATABASE)) {
            if (state.equals(SaverType.PLAYER)) {
                gameSaverAdapter = new PlayerDatabaseGameSaverAdapter();
            } else if (state.equals(SaverType.ROOM)) {
                gameSaverAdapter = new RoomDatabaseGameSaverAdapter();
            }

        } else if (type.equals(SaverType.TXT)) {
            if (state.equals(SaverType.PLAYER)) {
                gameSaverAdapter = new PlayerTXTGameSaverAdapter();
            } else if (state.equals(SaverType.ROOM)) {
                gameSaverAdapter = new RoomTXTGameSaverAdapter();
            }
        }
    }

    public void save(String username, Object o) throws Exception {
        gameSaverAdapter.save(username, o);
    }
}
```

GameLoader.java

```java
package com.gurup.domain.loader;

import com.gurup.domain.saver.SaverType;

public class GameLoader {
    GameLoaderAdapter gameLoaderAdapter;

    public GameLoader(SaverType type, SaverType state) {
        if (type.equals(SaverType.DATABASE)) {
            if (state.equals(SaverType.PLAYER)) {
                gameLoaderAdapter = new PlayerDatabaseGameLoaderAdapter();
            } else if (state.equals(SaverType.ROOM)) {
                gameLoaderAdapter = new RoomDatabaseGameLoaderAdapter();
            }

        } else if (type.equals(SaverType.TXT)) {
            if (state.equals(SaverType.PLAYER)) {
                gameLoaderAdapter = new PlayerTXTGameLoaderAdapter();
            } else if (state.equals(SaverType.ROOM)) {
                gameLoaderAdapter = new RoomTXTeGameLoaderAdapter();
            }
        }
    }

    public Object load(String username) throws Exception {
        return gameLoaderAdapter.load(username);
    }
}
```

## 3. Strategy

a. We have utilized the Strategy Pattern in designing the updated Time-Wasting Alien's functionality. We needed to implement similar, but distinct Time-Wasting functions depending on the remaining time of the Player.

b. We have designed a WastingStrategy interface which the DoingWellWastingStrategy, MercifulWastingStrategy, and the

ConfusedWastingStrategy classes implement. The wasteTime() method in these classes are called when the player has more than 70%, less than 30%, and between 30% and 70% of their time remaining, respectively.

c. The use of the Strategy Pattern makes it easier to switch between these different functionalities when necessary. We can simply set the Strategy of the Time-Wasting Alien to whatever is necessary at that point and call the wasteTime() function on the Strategy without caring for what the particular Strategy is at that moment.

# 4. Factory

a. Factory pattern was useful since the creation of BuildingObjects was complex and putting this responsibility in any other class would lower the cohesion of our project.

b. We used Factory pattern to create BuildingObjects with BuildingObjectsFactory as shown below

```
BuildingObject newBuildingObject = buildingObjectFactory.createBuildingObject(buildingObjectName, xCurrent, yCurrent, xLen, yLen);
buildingObjects.add(newBuildingObject);
```

```
public class BuildingObjectFactory {
    public BuildingObject createBuildingObject( @NotNull String name, int xCurrent, int yCurrent, int xLen, int yLen) {
        BuildingObject returnValue;
        switch (name) {
            case "Bin":
                returnValue = new Bin(xCurrent, yCurrent, xLen, yLen);
                break;
            case "Book":
                returnValue = new Book(xCurrent, yCurrent, xLen, yLen);
                break;
            case "Pen":
                returnValue = new Pen(xCurrent, yCurrent, xLen, yLen);
                break;
            case "Printer":
                returnValue = new Printer(xCurrent, yCurrent, xLen, yLen);
                break;
            case "Table":
                returnValue = new Table(xCurrent, yCurrent, xLen, yLen);
                break;
            default:
                throw new IllegalArgumentException("Unknown Building Object " + name);
        }
        return returnValue;
    }
}
```

## 5. Expert

a. Has all the necessary information needed to perform a responsibility.
b. In our case, Room knows which object contains a key and so it can unlock the door if the key exists.
c. Also in our case, Bag has all the information needed to use a power up. It knows if the power up exists and so it can activate the power up or shake the slot in the bag accordingly
d. By assigning the responsibility of creation to a class that has the necessary information, we reduce unnecessary coupling. The alternative would require us to get the information needed for the creation of said instance from the expert anyway, leading to higher coupling.
e. Below is the isKeyFound() method, an example of the Expert pattern in our code.

```
public Boolean isKeyFound(Rectangle rectMouseClick) {
    // MODIFIES: nothing
    // REQUIRES: rectangle that represents the mouse click
    // EFFECTS: Returns true if the key is found, false otherwise:
    // If the mouseRectangle does not intersect with the key, return false
    // If the game is paused, return false
    // If the mouseRectangle intersects with the key and player is near to object that contains the key, return true
    // If the mouseRectangle intersects with the key and player is not near to object that contains the key, return false
    if (Room.isPlayerFoundKeyBefore) {
        //System.out.println("You have already found the key for this room");
        return false;
    }
    if (!rectMouseClick.intersects(new Rectangle(xStart, yStart, xLimit, yLimit))) {
        // System.out.println("Did not click inside the room");
        return false;
    }

    if (Game.getIsPaused()) {
        // System.out.println("Cannot look for key if the game is paused. ");
        return false;
    }
    BuildingObject containerObject = key.getBuildingObject();
    Rectangle playerRect = player.getRectangle();
    for (BuildingObject bo : objects) {
        if (bo.getRectangle().intersects(rectMouseClick)) {
            if (!playerRect.intersects(bo.getRectangle())) {
                System.out.println("Player is not next to the object");
                return false;
            }
            if (bo.equals(containerObject)) {
                System.out.println("Key Found");
                Room.isPlayerFoundKeyForRoom = true;
                Room.isPlayerFoundKeyBefore = true;
                drawKeyForAMoment();
                player.setRemainingTime(player.getRemainingTime() + 50);
                return true;
            }
            System.out.println("Key not found");
            return false; // will be changed to bo.shake() to shake object
        }
    }
    System.out.println("Not an object!");
    return false;
```

## 6. Low Coupling and High Cohesion

a. While designing the class and interaction diagrams, we tried to preserve the Low Coupling and High Cohesion principles.

b. While refactoring our code to implement the new patterns (Adapter, Factory, and Singleton), we strived to keep coupling low and cohesion high even as the code got more and more complex.
c. Our Search Key Use Case did not follow low coupling and high cohesion in the first draft. We then updated the design to support low dependency, and to increase maintainability and reusability.
d. We have seen that as code got more and more complex, trying to maintain low coupling and high cohesion in our code proved to be a worthwhile investment of our time since it made enhancements and bug fixes so much easier.

# 7. Singleton

a. Singleton pattern makes it so that at most one instance of a Singleton class can exist at any given time.
b. We used Singleton for objects that will have more than one instance at the same moment.
c. This lowered coupling in our code since we did not have to move a class instance around and we could simply access the instance using the getInstance() method.
d. We used Singleton Pattern to initialize our game and powerUps as shown below:

```
public class Main {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Game.getInstance();
        Game.play();
    }
}
```

```
private Game() {

}

public static synchronized Game getInstance() {
    if (game == null) {
        game = new Game();
    }
    return game;
}
```

```
private ThrownBottlePowerUp(Player player) {
    this.player = player;
}

public static synchronized ThrownBottlePowerUp getInstance(Player player) {
    if (thrownBottlePowerUp == null) {
        thrownBottlePowerUp = new ThrownBottlePowerUp(player);
    }
    return thrownBottlePowerUp;
}
```

# K.SUPPLEMENTARY SPECIFICATIONS

| Version | Date | Description | Author |
|---|---|---|---|
| Draft | 31.10.2022 | First Draft. To be elaborated during revision. | Yakup Enes Güven |
| Revision | 02.11.2022 | Revision is done. | Ali Gebeşçe, Ömer Atasoy, Tuğra Demirel, Yakup Enes Güven |
| Finalization | 21.01.2023 | Finalization is done. | Esma Eray |

## Introduction

This is the part where all requirements of *OYUN: Escape From Koç* that are not captured in the use cases are specified.

## Functionality

***Logging and Error Handling***

System logs all errors to the console to analyze and handle errors with future updates.

***Security***

All usages require a valid username and password pair, which satisfies the following:

- Length of a password cannot be shorter than 8 characters.

## Usability

***Human Factors***

1. ***Visual Ease of Use***
    - Players should see all the visual elements in the game from 50 cm, including relatively small objects such as keys and power ups.
    - The coloring of images should be in harmony with each other.
    - There will be several buttons for this game:
        - Login
        - Register
        - Next (While Building the map)
        - Pause
        - Resume
        - Help
        - Exit
            - Each button should be positioned at “easy-to-access” locations
2. ***Auditory Ease of Use***
    - We may add an extra feature which provides feedback with sound whenever Player achieves or fails in the game since it would enhance the experience.

## Reliability

***Recoverability:*** The player can save their game and later load it to recover.

## Supportability

***Adaptability:*** The game should be platform-independent.

## Implementation Constraints

The Game should be implemented with Java and Java Swing.

## Performance

***Goals***

- Players should login the game in less than 2 seconds after entering a valid username, password pair.
- Animations should be smooth (High FPS rate) and compatible with other java swing based games.

# L.GLOSSARY

**Revision History:**

| Version | Date | Description | Author |
|---|---|---|---|
| Draft | 31.10.2022 | First Draft. To be elaborated during revision | Ömer Atasoy |
| Revision | 2.11.2022 | Revision. To be finalized | Ali Gebeşçe, Esma Eray, Tuğra Demirel, Yakup Enes Güven |
| Finalization | 21.01.2023 | Finalization. Last version of the game | Ömer Atasoy |

| Term | Definition and Information | Format | Aliases |
|---|---|---|---|
| Player | The user that interacts with the game. | | |
| Time | An integer value that is decremented every second. Starts from 5x(number of objects in the building). Can be increased by 5 if the player collects the Extra Time Power Up. The Player loses if they lose all their lives. | Integer | |
| Bag | The player's inventory that stores their Power Ups. | | |
| Key | Allows the player to move to the next room. | | |

| | | | |
|---|---|---|---|
| Alien | Creature that tries to hinder the player's progress. All kinds of aliens appear randomly in the buildings every 10 seconds. They cannot move between buildings. | | |
| *Blind Alien* | Can kill the player if they are close. Moves randomly unless a bottle is thrown, in which case it moves towards the bottle's destination. Its attacks cannot be blocked by the Protection Vest. | | |
| *Shooter Alien* | Can kill the player if they are within its range. Cannot move. Shoots once every second. Its attacks can be blocked by the Protection Vest. | | |
| *Time-Wasting Alien* | Changes the location of the key randomly. Can determine that the player is doing well, can determine that the player is doing poorly and become merciful, or can be confused regarding the player's progress. Its behavior depends on its current state. Cannot move. Cannot attack the player. | | TW Alien |
| Life | Required by the player to keep playing. Decreases when the player is killed by the Shooter Alien or the Blind Alien. Increases if the player gains the Extra Life Power Up. | Integer | |
| Power Up | Helps the player escape from the aliens and find keys. Appears randomly every 12 seconds in random locations. Disappears if the player does not collect them in 6 seconds. Collected by the player via right-clicking. Player does not need to be close to the Power Up to collect it. | | PU |

| | | | |
|---|---|---|---|
| *Extra Time* | Grants the player 5 more seconds. Consumed immediately when collected. | | |
| *Hint* | Gives a hint to the player about the key's location. Once used, a region containing the key is highlighted for 10 seconds. The player can press "H" to use it. Can be stored in the bag for later use. | | |
| *Protection Vest* | Protects the player from the Shooter Alien. Once used, the protection lasts for 20 seconds. The player can press "P" to use it. Can be stored in the bag for later use. | | |
| *Plastic Bottle* | Fools the Blind Alien. Once used, the Blind Alien moves to the Plastic Bottle's destination before going back to moving randomly. The player can press "B" to use it and then has to press one of "A", "D", "W", or "X" to decide on its direction. The secondary buttons mean "West", "East", "North", and "South", respectively. Can be stored in the bag for later use. | | |
| *Extra Life* | Adds one extra life to the player's lives. Can cause the player to exceed their maximum lives. Consumed immediately when collected. | | |
| Room | The "worlds" where the game takes place in. Is home to the Player, Aliens, Objects, PowerUps, and the Door. There are 6 of them, each with their respective minimum quota of objects. | | Building |
| *Student Center* | Is the first building. Must contain at least 5 objects. | | |

| | | | |
|---|---|---|---|
| *CASE Building* | Is the second building. Must contain at least 7 objects. | | |
| *SOS Building* | Is the third building. Must contain at least 10 objects. | | |
| *SCI Building* | Is the fourth building. Must contain at least 14 objects. | | |
| *ENG Building* | Is the fifth building. Must contain at least 19 objects. | | |
| *SNA Building* | Is the sixth/final building. Must contain at least 25 objects. | | |
| Building Mode | The first phase of the game. During this phase, the player designs the buildings by placing objects in locations they desire. Two objects cannot be placed in the same location. There must be a certain minimum number of objects placed in each room during this phase (explained above). | | |
| Running Mode | The second phase of the game. During this phase, the player is placed in a random location. The player must look behind objects to find the hidden key and move to the next room until they move out of the final room to finish the game. The Timer and the Aliens are activated and the player must care for them. The Power Ups are activated too. | | |
| Pause Button | Used to pause the game. | | |
| Save Button | Used to save the game on a storage of the Player's choice | | |

| | | | |
|---|---|---|---|
| Login Screen | The starting screen where the player can login or register to their account. The player can also choose where they want their game saved, or decide to play without logging in. | | |
| Main Menu | The screen that comes after the Login Screen where the player can start a new game, load a previously saved or see the help screen. | | |
| Help Screen | Presents the player with instructions on how to play the game. Available on the Main Menu. | | |
| Object | A piece of furniture that occupies one Square. The Key is hidden behind one of the Objects. When next to an Object, the player can left-click it to check if the Key is hidden behind it. | | BuildingObject |
| Door | The gate to the next level or the end of the game. The player can only walk through the Door if they have collected the Key in the Building. | | |

# M.REFERENCES


1. Loop breaking notation in interaction diagrams inspired from: https://circle.visual-paradigm.com/iteration-loop-break/
2. Larman, C. (2005). *Applying Uml and patterns: An introduction to object-oriented analysis and design and the unified process* Prentice-Hall.
3. *Design Patterns in Java Tutorial*. Tutorials Point. from https://www.tutorialspoint.com/design_pattern/index.htm
4. https://stackoverflow.com/questions/13029261/design-patterns-factory-vs-factory-method-vs-abstract-factory
5. https://refactoring.guru/design-patterns/java
6. https://www.tutorialspoint.com/design_pattern