KOÇ UNIVERSITY

Escape From Koç

By Zombies

COMP 302 SOFTWARE ENGINEERING

Fall 2022

COMPUTER ENGINEERING DEPARTMENT

Zombies:

Bartu Uzun 72735

Ayten Dilara Yavuz 72281

Deniz Yılmaz 69743

Batuhan Altınyollar 69577

Berke Derin Berktay 72968

Table of Contents

Table of Contents 2

Introduction 3

Teamwork Organization 6

Use Case Diagram 6

Use Case Narratives 8

System Sequence Diagrams 14

- **SSD for Log in to the Game Use Case** 14
- **SSD for Design Building Use Case** 16
- **SSD for Escape the Building Use Case** 17

Operation Contracts 18

Sequence/Communication Diagrams 20

Class Diagram 25

Package Diagram 26

Design Pattern Discussions 27

Supplementary Specifications 29

Glossary 31

Design Choices 33

Introduction

Vision

Introduction

We consider a user controlled indie game that entertains the players by challenging them against time for finding a key through the buildings of KOÇ university and escaping. As the objective states, the name of our game is “Escape from Koç”. The game consists of multiple phases: building mode and running mode. In the building mode the players will decorate the buildings they will play the game in. In the running mode the player will try to find the key before the timer ends and lose all life. There will be obstacles such as three different types of aliens that each have their own unique method of making it harder for the player to find the key. Although there are challenges, the player will also be able to use 5 different types of power ups for their advantage. The players will be able to save and load their game progress by entering their unique username and password combination at the start of the game and clicking the save game button throughout playing.

Positioning

Business Opportunity

As the world is becoming more digital, the demand for games increases each day. Our game will be able to satisfy such people that are into indie games on desktop platforms. By being one of the finest quality games in its league, our game will have a chance to be successful.

Problem Statement

Even though there is an increasing interest and demand in graphically amusing games, the industry lacks top-down view indie games and ignores the requesting community of such game types in the market. With our game, we aim to appeal to these community members and give them a good quality game they deserve.

Product Position Statement

This game is made for people that understand quality gaming and have the time for playing it. For making the players feel more familiar with the environment they play in, we will give them the chance to build the buildings of Koç university as they desire before fighting with the aliens and exploring the buildings. We want the players to feel pure enjoyment with a top quality indie game with customizability. Escape from Koç is just the game they are looking for, and is appealing to everyone.

Stakeholder Descriptions

Market Demographics

As to speak of the audience, people from all ages can enjoy this unforgettable experience of their lives. Even though the game takes place in Koç University and the audience that have been in Koç University before will be much familiar with the buildings, it would not be a problem for anyone else to understand the game locations. Therefore it can be said that our market demographic consists of anyone that wants to spend some time playing an enjoyable game.

High-Level Goal	**Priority**	**Problems and Concerns**	**Current Solutions**
Esthetic and enjoyable game design	High	Introducing new gameplay mechanics and features to the game must be done with consideration, as failing to properly integrate them can negatively impact the player's understanding and enjoyment of the game. To ensure a seamless experience, graphical elements should be designed in a clear and organized manner and similar concepts should be visually differentiated to avoid confusion for the player.	The object designs are made in a simple 8-bit style to give style to the game design and simplify visualization for the player. By having an easy-to-learn control scheme, it can also enhance player engagement as the challenging parts of the game will depend on the player's abilities, rather than any issues with the game system.
Understandable objectives	High	Assessing the control system of a game is vital in determining its ease of use. Since our game introduces a lot of rules rapidly to the player, the control system should be simple for the player to handle.	Including a help screen in both phases building and running so that the player resolves any confusion from happening. Additional images for showing the corresponding object

		Keeping the control system consistent also impacts the game's level of difficulty. To align with our target audience, we must make sure the control system is easy to learn for the player.	and obstacles makes it easier to follow.
Manageable file handling	Medium	The compatibility of game state files across different operating systems is crucial, as file handling and formatting can vary greatly. It's important that the game is able to save and load these files seamlessly across different operating systems to ensure a smooth gaming experience.	Using file formats that are not specific to any particular system and using encodings that are widely supported, to make the game more compatible with different platforms and operating systems.
Smooth animations	Medium	The game should be optimized to run smoothly on even low-end machines, with minimal frame rates with the minimum being 30 fps.	The movement and animation of the player, blind alien and shooter aliens' bullets being managed so that they do not affect the frame rate of the game.
…	…	…	…

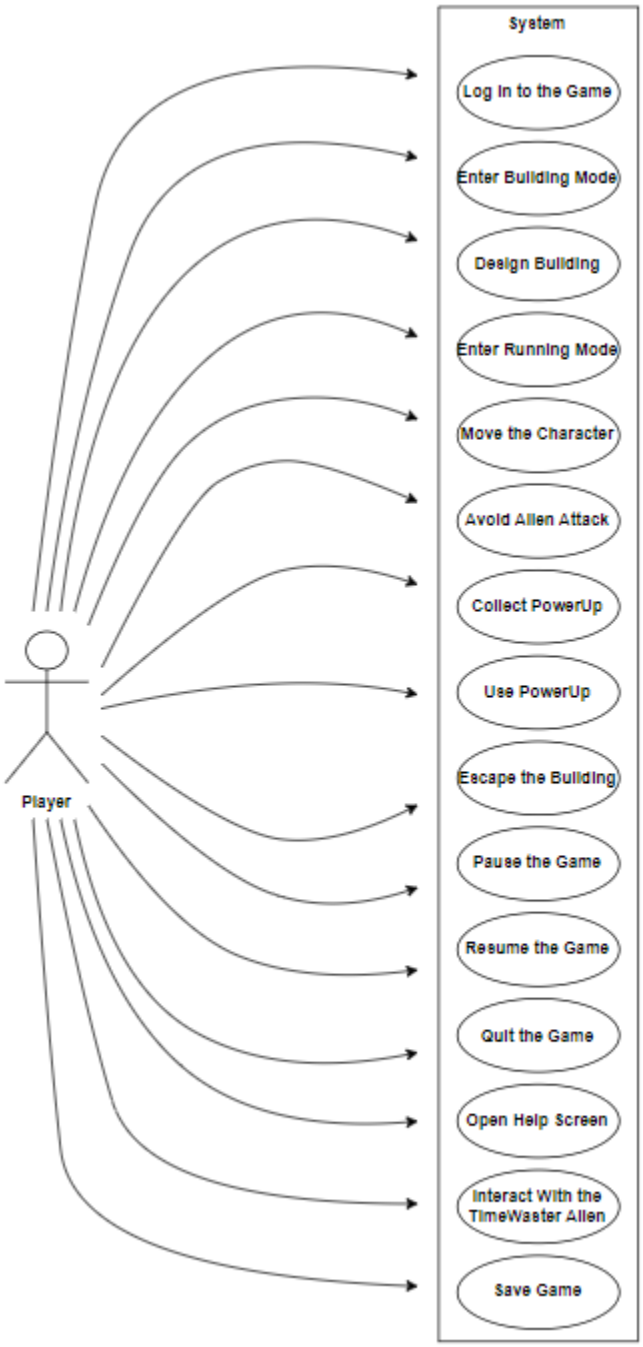
Teamwork Organization

Each week, we discussed the requirements of that week. Everyone voluntarily took responsibility for the task they could do. After that, everyone opened an issue on GitHub

and created a branch specific to the task. We brainstormed together to make sure our code and classes conform to patterns. Furthermore, we performed some difficult and complex tasks in mini groups instead of individually. We were in constant communication as a group to prevent possible conflicts in the code and to proceed in parallel. The workload distribution for weekly assignments are explained in detail in the meeting agendas. In general, our division of labor was as follows:

Bartu Uzun: Power-Ups, Aliens, Building Mode, Running Mode, Save/Load
Ayten Dilara Yavuz: Aliens, Running Mode
Deniz Yılmaz: Login Mode, Aliens, Running Mode
Batuhan Altınyollar: Save/Load
Berke Derin Berktay: Save/Load, Building Mode

Use Case Diagram



Use Case Narratives

Use Case UC-1: Log in to the Game

Primary Actor: The player

Stakeholders & Interests:

1. Player: Wants to login to the game
2. System: Wants to ensure that the player logins to the game successfully with a unique name and a password, and loads his or her game if in the database

Preconditions:

1. The game executable is run
2. Player has a unique login name and a password

Postcondition:

1. Player has logged in to the game successfully

Main Success Scenario:

1. System asks player for a username and a password
2. Player enters their unique username and password
3. System cannot find a previously saved game corresponding to the player's username
4. Player and their username-password combination is given to the database as a new entry
5. Player enters a new game.

Alternative Flows:

a) Player has a previously loaded game in the database
    1. System does not start a new game, but instead the player's current game in the database is loaded in order for them to continue and finish it.

b) System recognizes the username player has entered, but the password they entered does not match with the database
    1. Player is asked to re-enter their username and password
    2. Player enters their username-password combination correctly
    3. Player enters their loaded game

Use Case UC-15: Save Game

Primary Actor: Player

Stakeholders & Interests:

1. Player: wants to save his records of the game for continuing from the same instance of the game.
2. System: wants to send the data of the players recordings such as items on the bag, alien count, alien location, alien type, object info, power up info, building index, player time left, etc. to a database and return them when the player logs in with the same username when restarting the game.

Pre-Conditions:

1. The player has entered the game with his personal unique username
2. The player has entered the running mode.
3. Game is in a paused state.
4. The player has clicked the Save button.

Post-Condition:

1. The game has been saved to the database successfully.

Main Success Scenario:

1. The unique username entered by the player exists in the database.
2. The game will save the data of the currently running game with the user's unique. name to the database such that the player can start from that saved point whenever they wish to.

Alternative Flows:

a) The username does not exist in the database
   1) There will be no rewriting, and a new document for the new user will be created in the database.

b) The game is over
   1) If the game is finished the game will be deleted from the database, thus it will not be possible to load the same game again.

c) The game disconnects from the database
   1) The game will not be able to send data to the database. Therefore save after the disconnection will not be official.

Use Case UC-3: Design Building

Primary Actor: Player

Stakeholders & Interests:

1. Player: Wants to design each building one by one
2. Game System: Wants to ensure player designs all of the buildings according to their constraints:
    - Player should put 5 objects inside the Student Center from the given object pool
    - Player should put 7 objects inside the CASE building from the given object pool
    - Player should put 10 objects inside the SOS building from the given object pool
    - Player should put 14 objects inside the SCI building from the given object pool
    - Player should put 19 objects inside the ENG building from the given object pool
    - Player should put 25 objects inside the SNA building from the given object pool

    - Objects player has put should not collide in any of the buildings

Pre-Condition:

1. Player should have logged in the game, and got their unique login name
2. Player should have opened the help screen and read the documentation of the game, so that they know how to play

Post-Condition:

1. Player has submitted the designs of all of the buildings successfully, and the game system accepted all of the submissions
2. Player is ready to start playing the game

Main. Success Scenario:

1. Player starts the game
2. Building mode is started
3. System shows the Student Center, the object pool that the player can choose objects from, and the instruction “Please put 5 objects into the Student Center”
4. Player puts the objects without collision
5. Player submits the Student Center’s orientation
6. System accepts the submission
7. Steps 3-6 are repeated for CASE, SOS, SCI, ENG, and SNA buildings, each with their own constraints

Alternative Flows

a. Player tries to submit any of the buildings with less than the required number of objects
    1. System does not accept the submission, and shows an error indicating that the player needs to put more objects to the building
    2. Player adds objects to the building until the total number of objects put is equal to the required amount
    3. Player submits the building

4. System accepts submission

b. Player tries to submit any of the buildings with more than the required number of objects
    1. System does not accept the submission, and shows an error indicating that the player needs to put a less objects to the building
    2. Player removes objects from the building until the total number of objects put is equal to the required amount
    3. Player submits the building
    4. System accepts submission

c. Player tries to submit the building while there is a collision (i.e. player has put the objects such that at least 2 of the objects are colliding)
    1. System does not accept the submission, and shows an error indicating that the player needs to make sure the objects they put do not collide with each other
    2. Player fixes all of the colliding objects
    3. Player submits the building
    4. System accepts submission

Use Case UC-9: Escape the Building

Primary Actor: The player

Stakeholders & Interests:

1. Player: Wants to escape from the building
2. System: Wants to ensure that the game works properly
3. Power-ups: Want to help player escape
4. Aliens: Want to stop player from escaping

Pre-Conditions:

1. The game has started
2. Player has escaped the previous buildings

Post-Condition:

1. Player escapes from the current building successfully.
2. Player has entered the next building.

Main Success Scenario:

1. Player moves around the building
2. Player checks objects inside the building to find the hidden key
3. Player repeats steps 1-2 until they find the object that has the key hidden inside

4. System tells the player that the key has been found, and the door is unlocked
5. Player moves through the door

Alternative Flows:

a) At any given time, player pauses the game
    1. After some time, player resumes it
b) Timer is up at any point before the player escapes
    1. Game is over
    2. System shows a “game over” screen which indicates that the game is finished, and that the player has lost the game
c) Player gets killed by an alien
    1. Game is over
    2. System shows a “game over” screen which indicates that the game is finished, and that the player has lost the game
d) Player escapes through the SNA building
    1. Game is over
    2. System shows a “game over” screen which indicates that the game is finished, and that the player has won the game

Use Case UC-14: Interact with the TimeWastingAlien

Primary Actor: Player

Stakeholders & Interests:

1. Player: wants to find the key and escape the building before the time runs out
2. TimeManager: wants to make sure the timer works correctly
3. TimeWastingAlien: wants to change the location of the key, its behavior will change according to the percentage of time left

Pre-Conditions:

1. Running Mode has started
2. A TimeWaster has been spawned
3. More than 70% of the total time remains

Post-Condition:

1. The player successfully finds the key and escapes from the building

Main Success Scenario:

1. Player searches through the building to find the key
2. In every 3 seconds, the TimeWaster will change the location of the key
3. Player finds the object that has the key before the timer runs up

Alternative Flows:

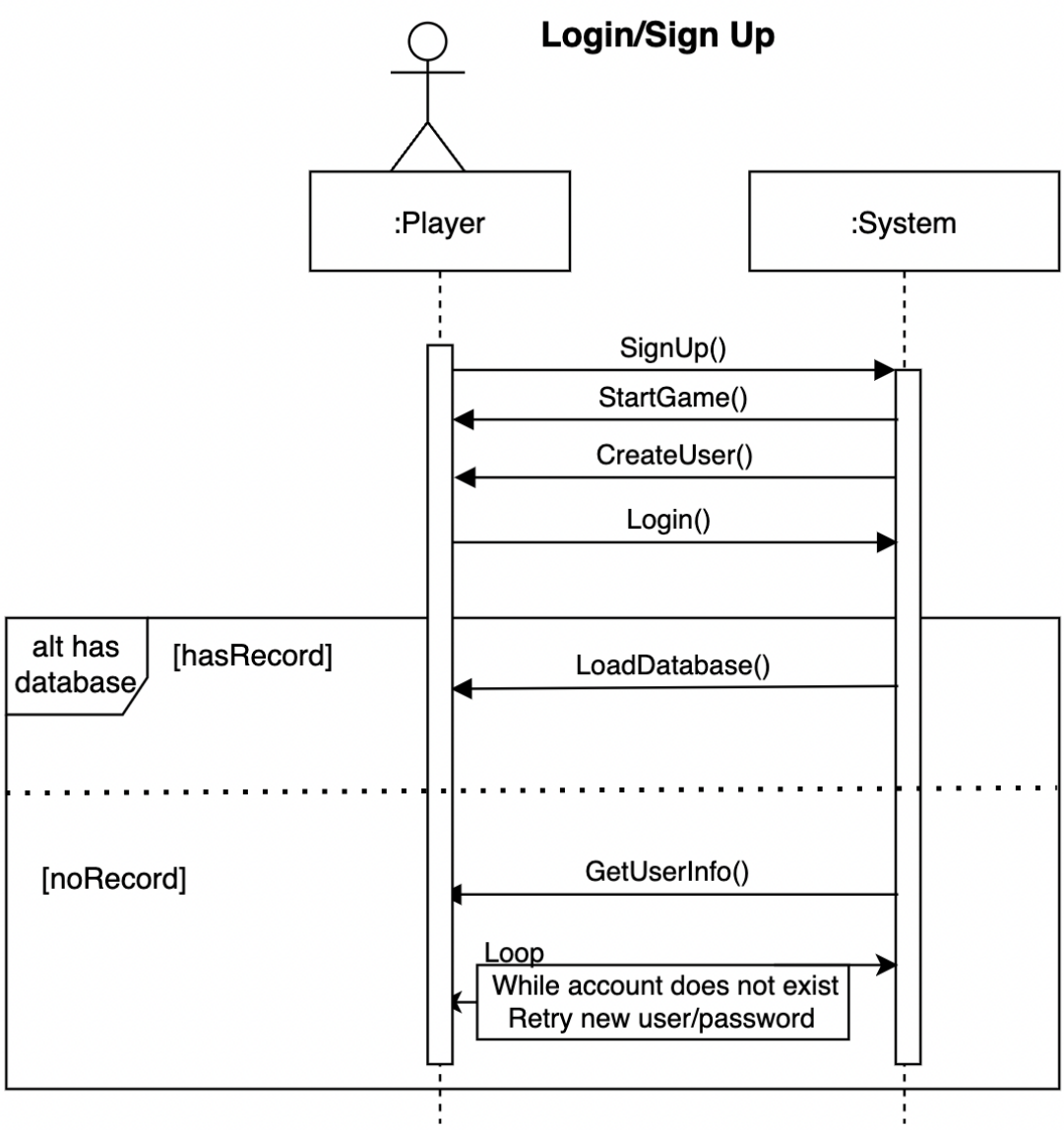
a) Time remaining is more than 30%, but less than 70% of the total time
   1) TimeWaster changes its behavior: now it waits for 2 seconds and then disappear without doing anything
   2) Player continues to search for the key before the timer runs up

b) Time remaining is less than 30% of the total time
   1) TimeWaster changes its behavior: it changes the location of the key once, and then disappears
   2) Player continues to search for the key before the timer runs up

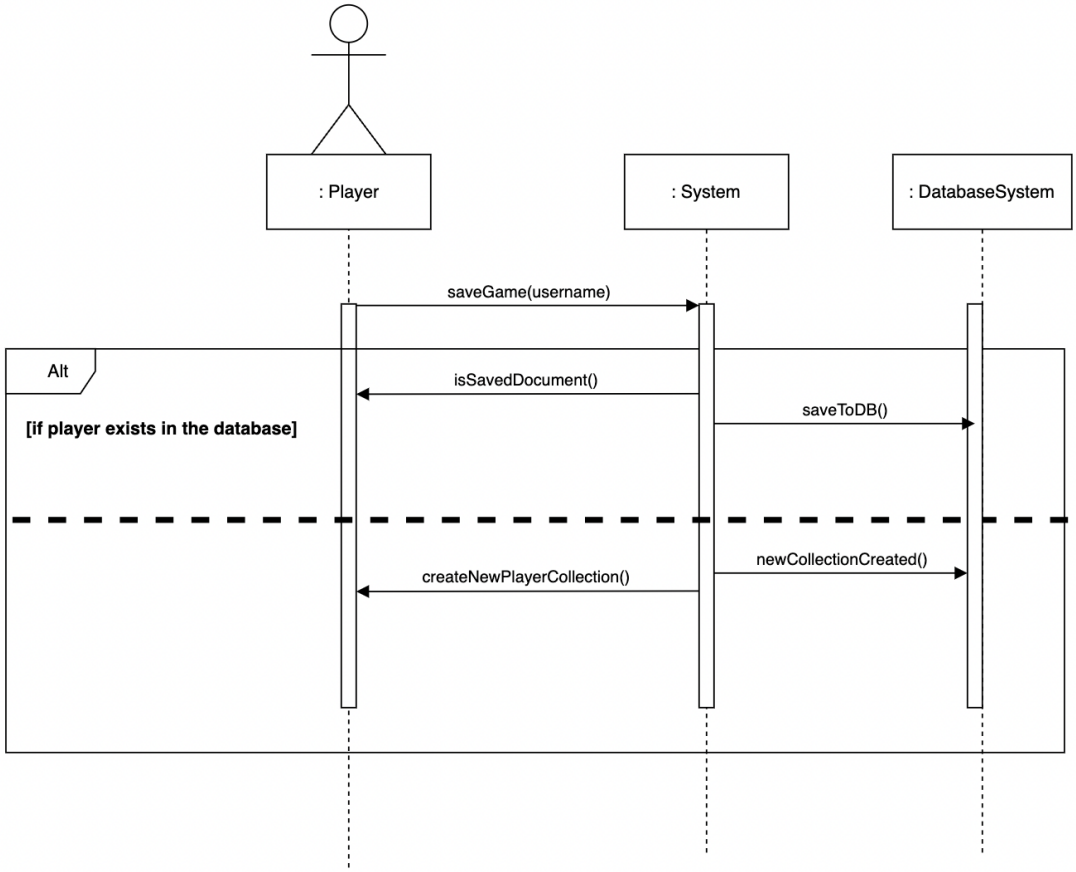
c) Player gets killed by one of the aliens before the time runs up
   1) The game ends
   2) System shows a “game over” screen which indicates that the game is over and that the player has lost the game

System Sequence Diagrams

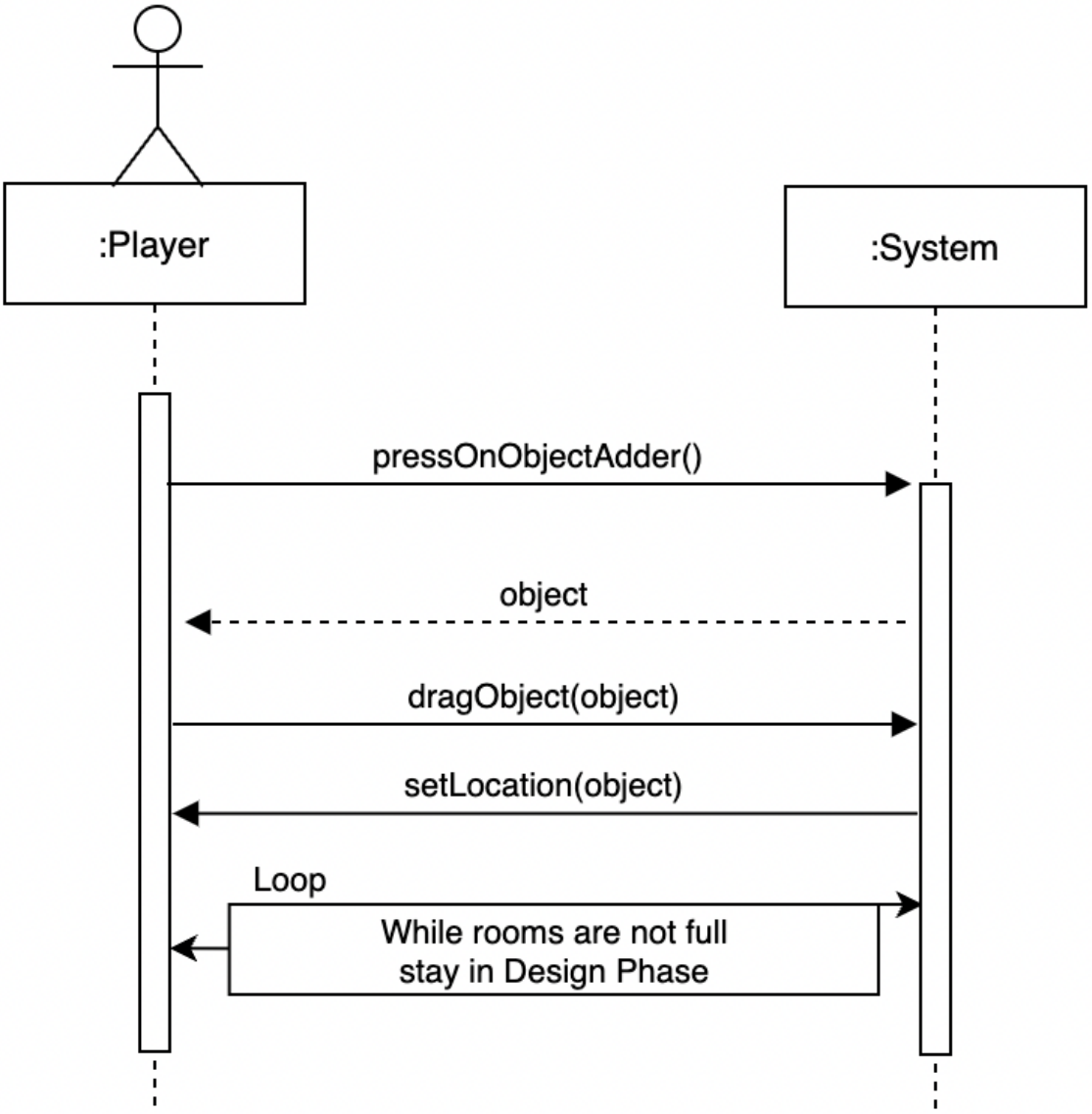
SSD for Log in to the Game Use Case



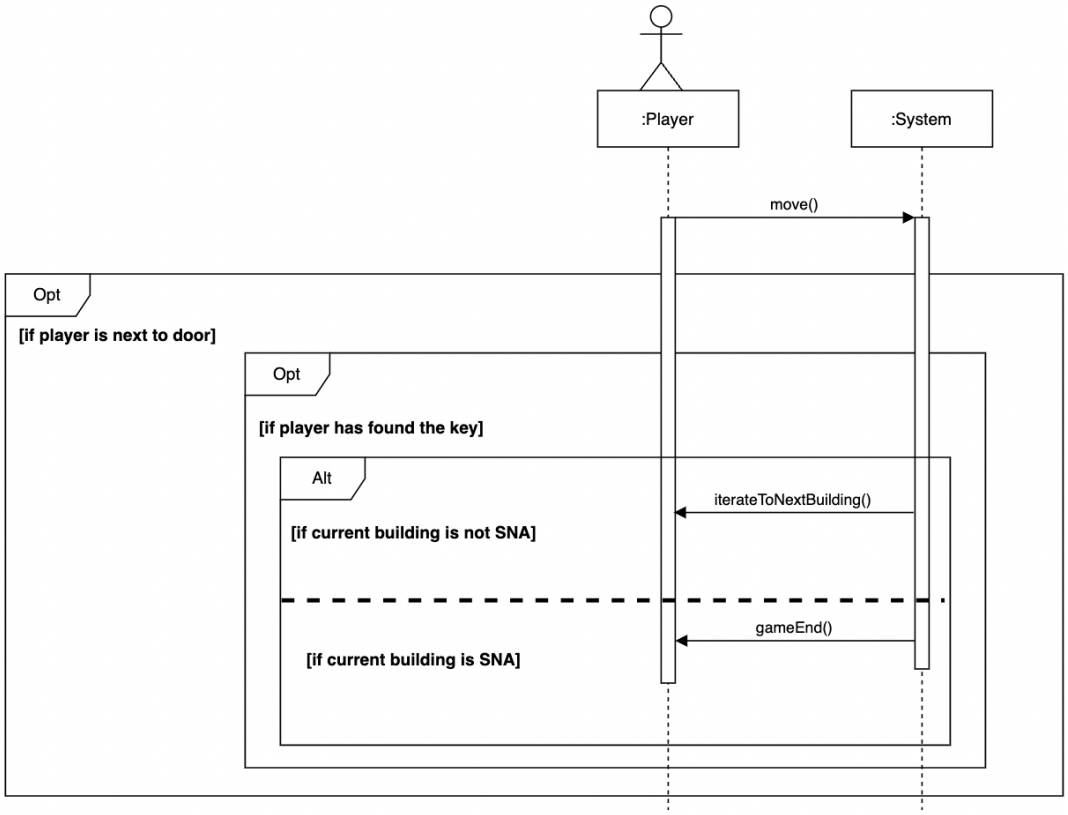
SSD for Save Game Use Case



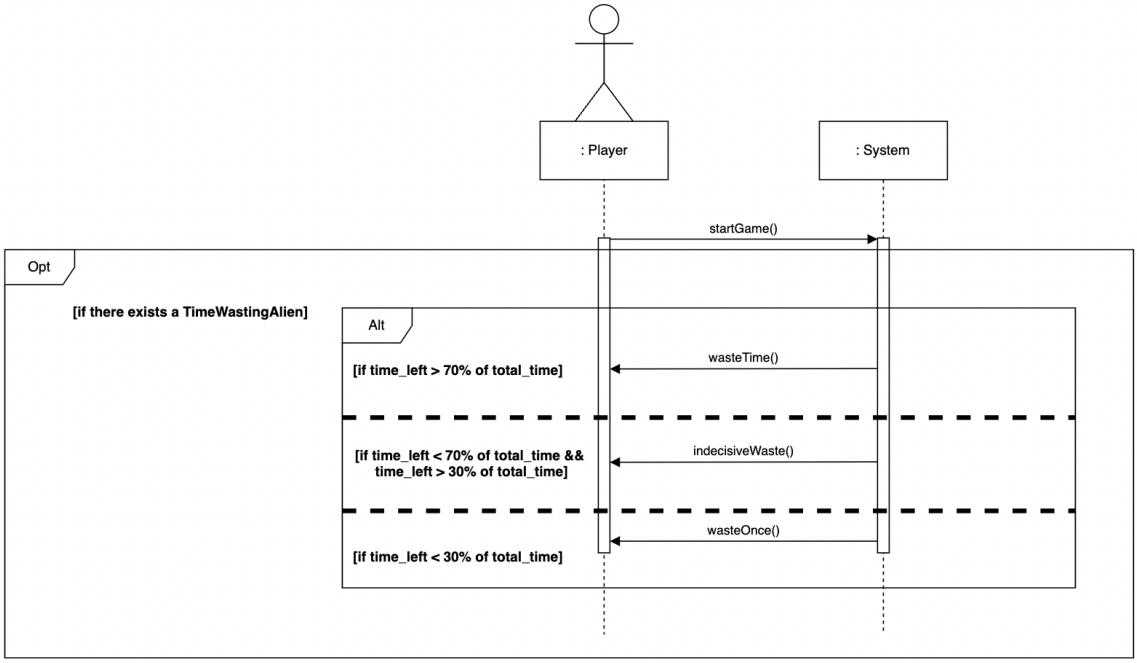
SSD for Design Building Use Case



SSD for Escape the Building Use Case



SSD for Interact with the TimeWastingAlien Use Case



Operation Contracts

Contract CO1: Log in to the Game
Operation: LogIn(loginName)
Cross References: UC1 Log in to the Game
Preconditions:

- The game executable is run.
- Login button is pressed.

Postconditions:

- A unique name was created by the player.
- The game has been ready to be played.

Contract CO14: Save the Game
Operation: Save(username)
Cross References: UC1 Log in to the Game
Preconditions:

- The game is in running mode.

- The player clicked the save button.

Postconditions:

- The last state of the game was saved in a file for the current player.

Contract CO3: Design Buildings
Operation: DesignBuildings()
Cross References: UC3 Design Buildings
Preconditions:

- The game is in the Building Mode.
- The player has logged into the game.

Postconditions:

- All buildings were designed by the player in accordance with the desired rules.

Contract CO9: Escape the Building
Operation: EscapingBuilding()
Cross References: UC9 Escape the Building
Preconditions:

- The game is in the running mode.
- The player has logged into the game.
- The player has escaped the previous buildings.

Postconditions:

- Player has entered the next building.

Contract CO13: Interact with the TimeWaster
Operation: AttackBehavior.attack()
Cross References: UC13 Interact with the TimeWaster
Preconditions:

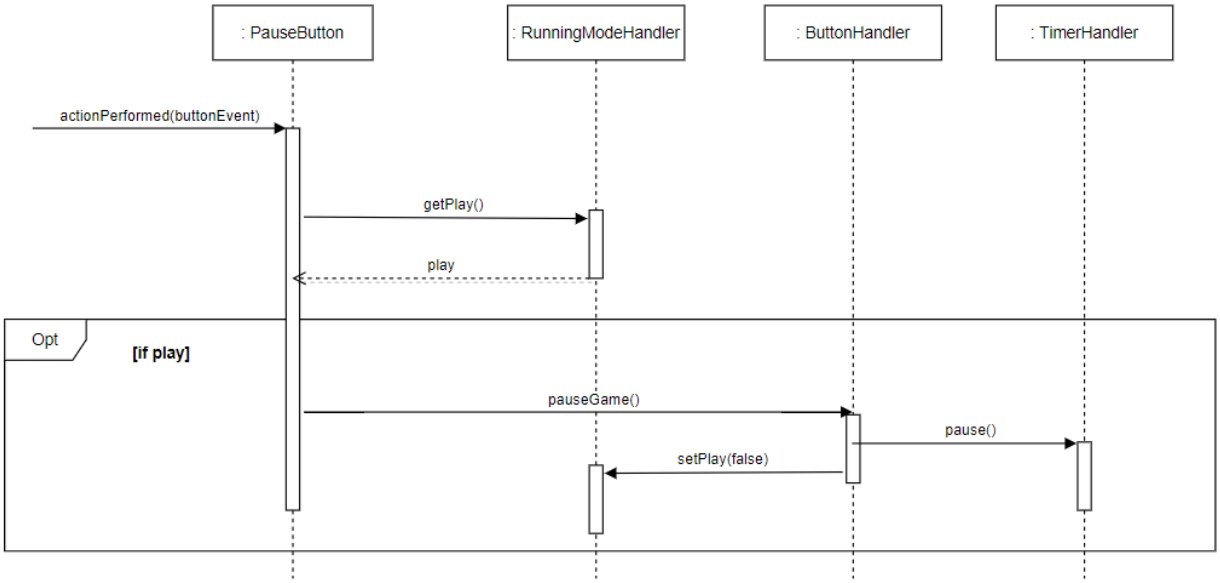
- The game is in running mode.
- The player has logged into the game.

Postconditions:

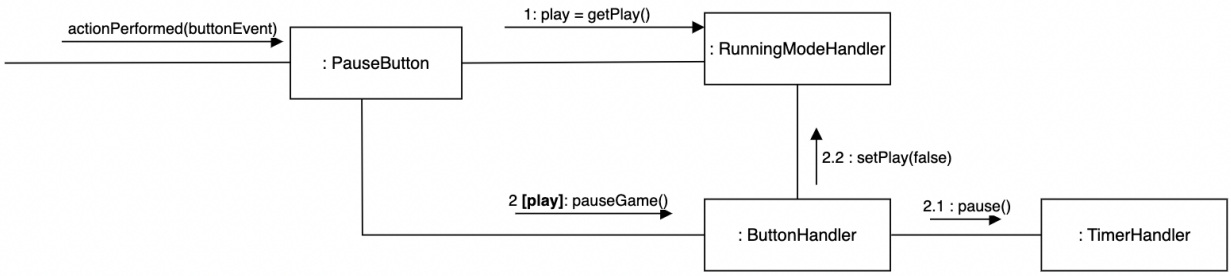
- TimeWaster's attack behavior was set according to the time left.
- According to TimeWasters' attack behavior, the key location was changed.

Sequence/Communication Diagrams

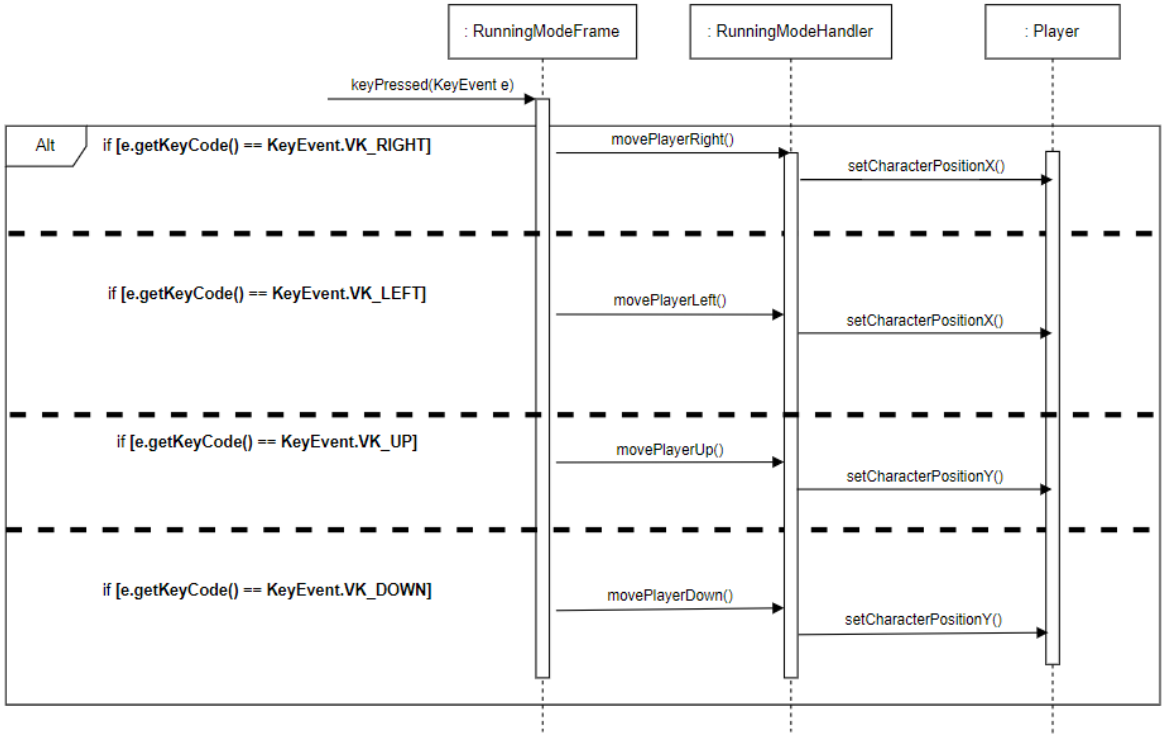
Sequence Diagram: Pause the Game



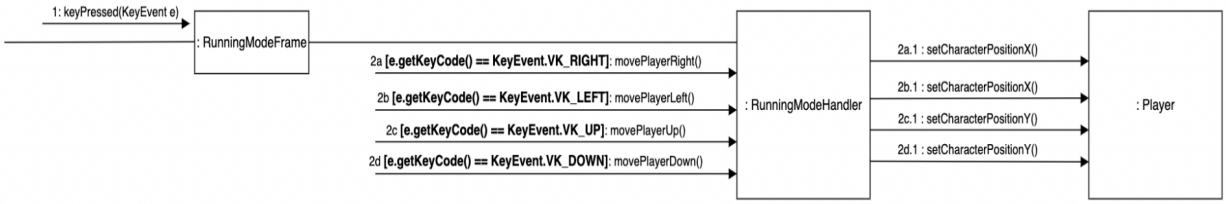
Communication Diagram: Pause the Game



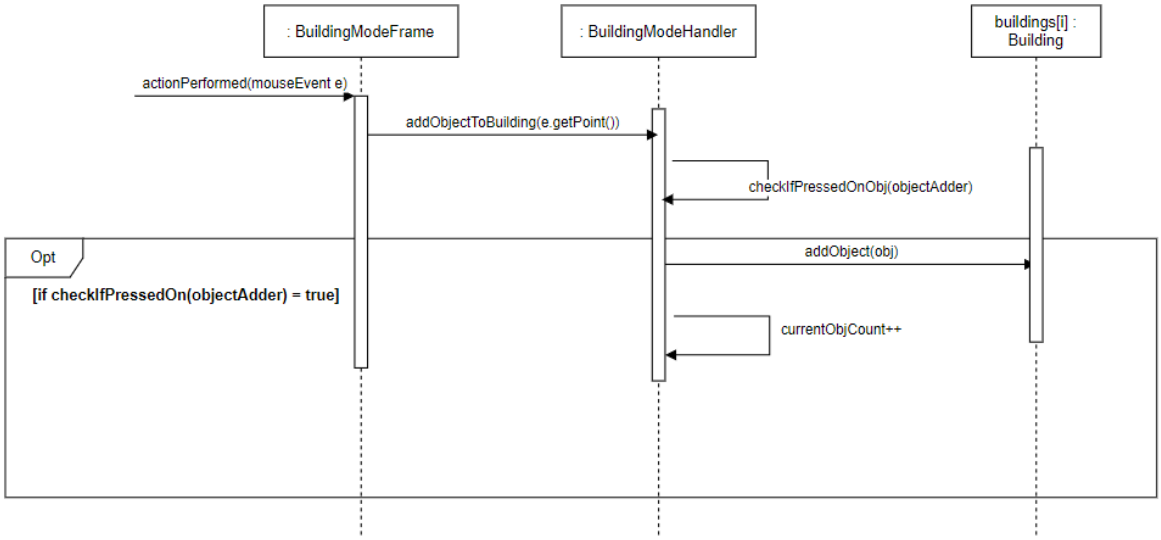
Sequence Diagram: Move the Character



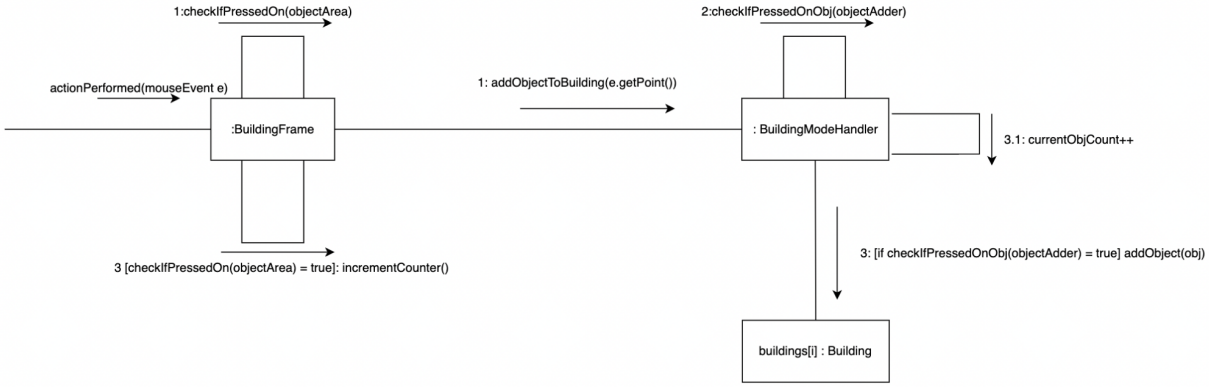
Communication Diagram: Move the Character



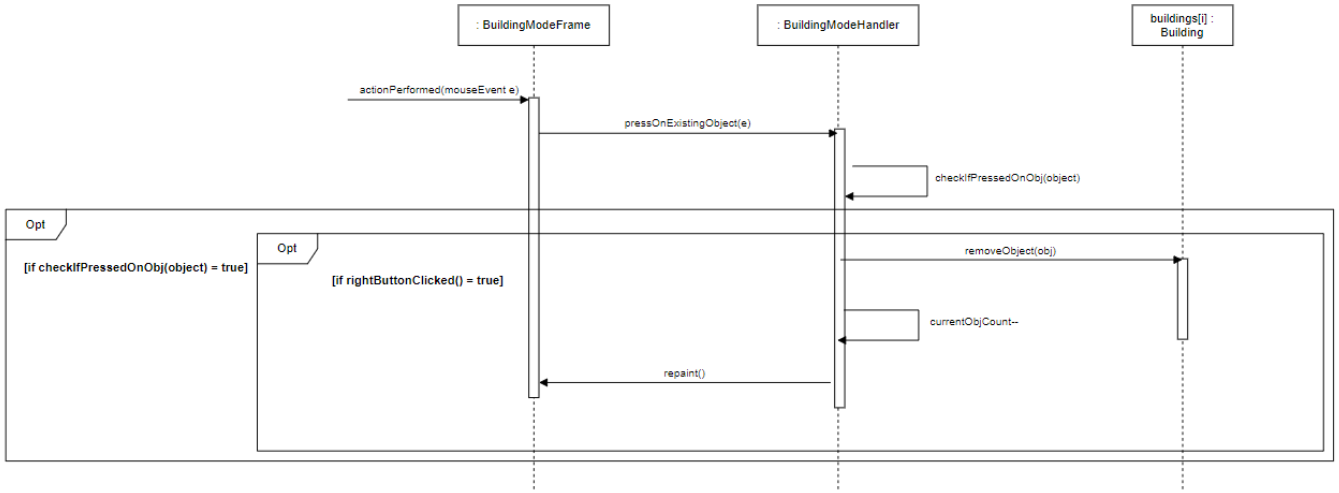
Sequence Diagram: Add Object to the Building



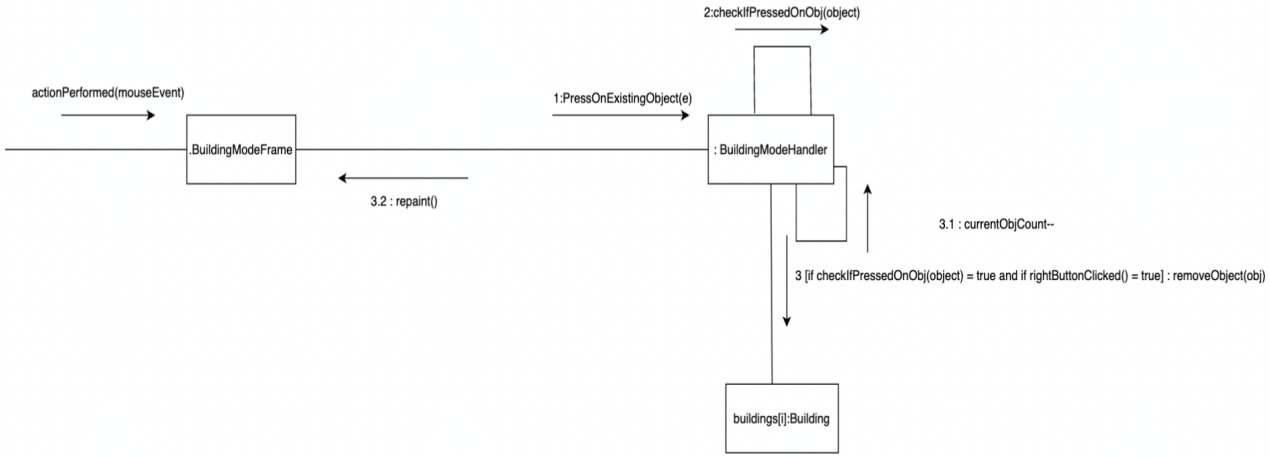
Communication Diagram: Add Object to the Building



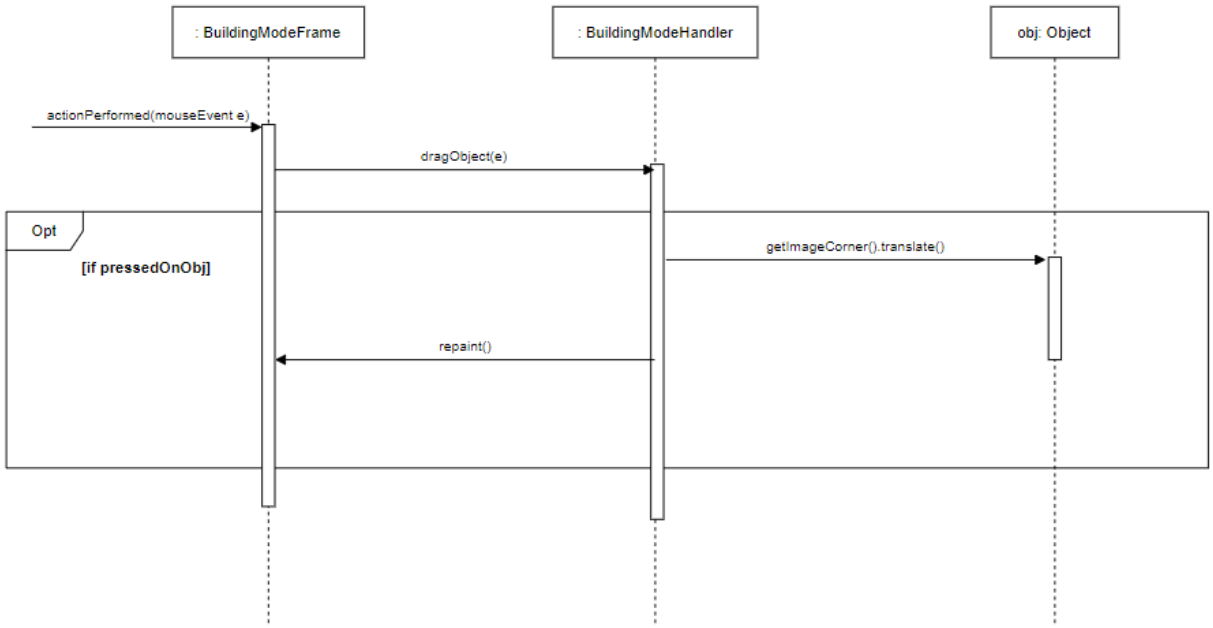
Sequence Diagram: Remove Object from the Building



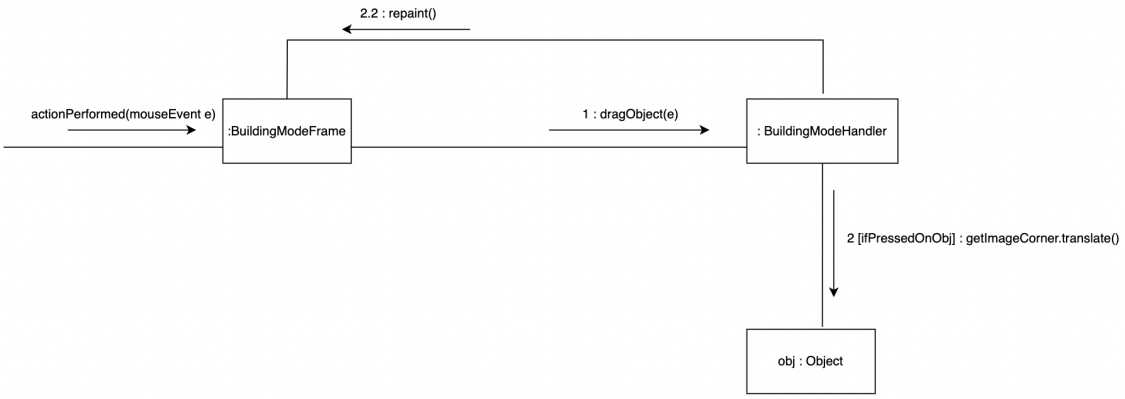
Communication Diagram: Remove Object from the Building



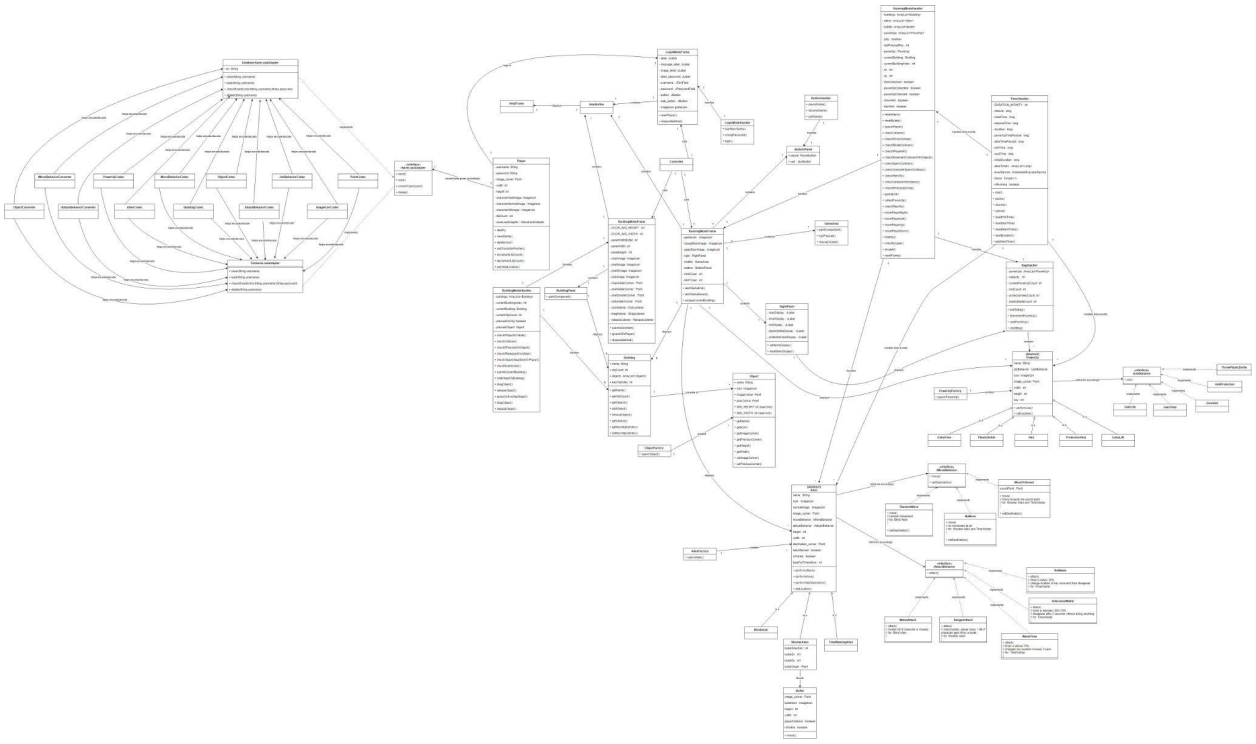
Sequence Diagram: Drag Objects Inside the Building



Communication Diagram: Drag Objects Inside the Building



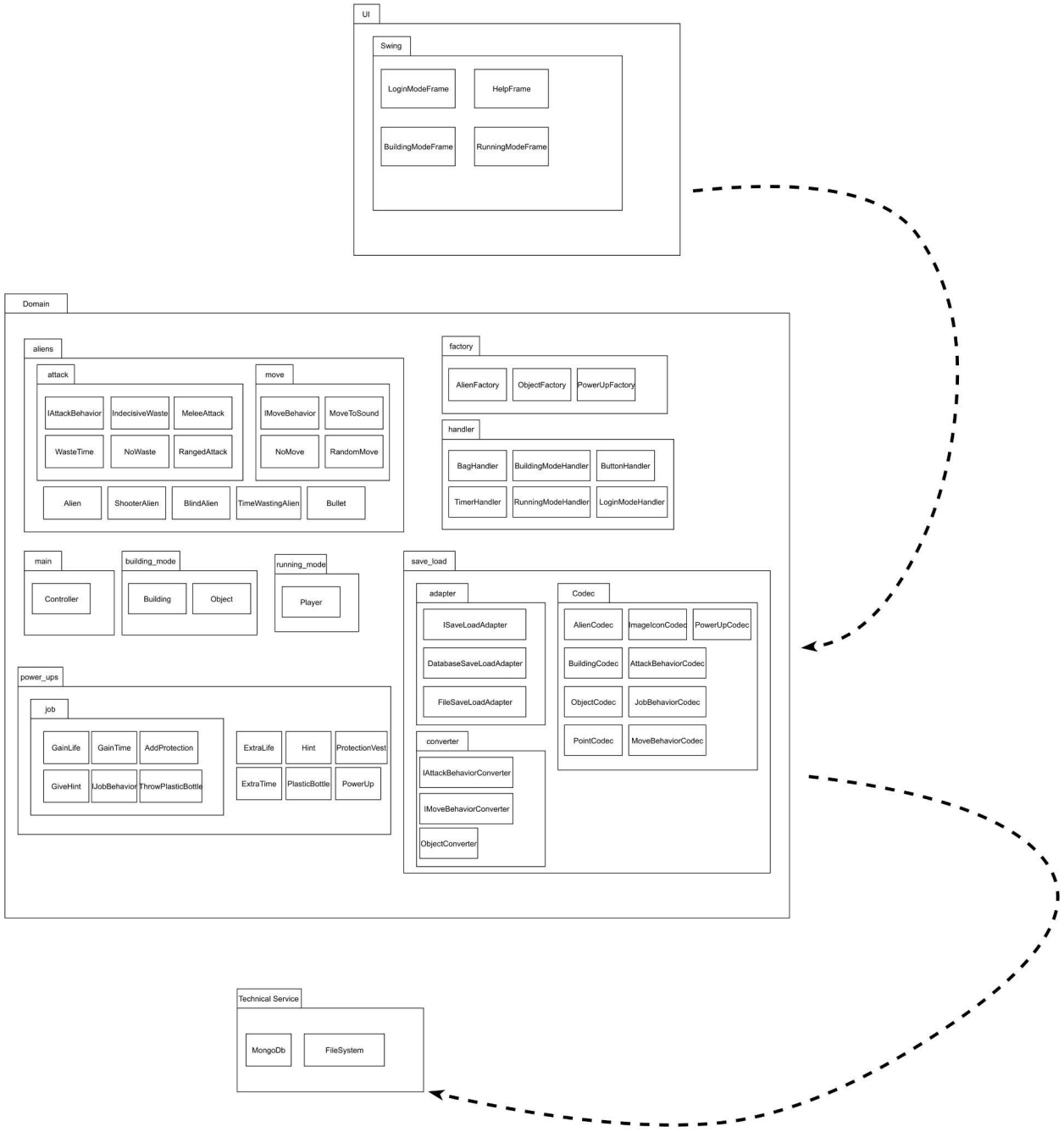
Class Diagram



Link to the class diagram:

https://drive.google.com/file/d/1idwzpFzfaCRHmU6V3PgngfSzK54bp-7q/view?usp=sharing

Package Diagram



Design Pattern Discussions

Information Expert: Information Expert pattern suggests that when assigning a responsibility, we should assign it to the class that has all the necessary information to perform that responsibility. In our software, there are many examples of this pattern, especially in the Handler classes.

Information Expert Pattern Example in TimerHandler Class:

TimerHandler class is the information expert of all of the time-related events in the game. It has the information of the total time, total time passed, time passed since a powerUp has been spawned, time passed since an alien has been spawned, it has info about each alien in the game and the time passed since the last time they moved and attacked. Thus, we gave it the responsibility to manages all these time related events such as deciding that a powerUp should be spawned/despawned, deciding that an alien should perform attack/move, or deciding that the game is over.

Information Expert Pattern Example in BagHandler Class:

BagHandler is the information expert of all the powerUps that are inside the player's bag. Thus, we gave it the responsibility to decide if player can collect more powerUps, or if a player can cast the powerUp of which they pressed the corresponding button.

Low Coupling: Low coupling pattern suggests removing unnecessary coupling between classes. Throughout our code, we applied low coupling pattern such that our classes are mostly only connected to their related mode handler (e.g. PowerUp class is related to RunningModeHandler) and some of the classes that they contain/use or the classes that contain/use them. Thus, unnecessary coupling remains low throughout the code. As a result of that, making a change in a class is easy, because it does not have huge impact in the rest of the code.

High Cohesion: High cohesion pattern suggests that operations inside a class should be well related to each other. Our code supports high cohesion such that all of our classes are focused on one part or one element of the game. Thus, all the methods inside our classes are somehow related to each other. As a result, our code is easier to understand, follow along, and debug, because we know which classes have which kinds of methods already. Also, applying high cohesion helps us to lower the coupling in the software (low coupling).

Polymorphism: We apply polymorphism in our design twice: in aliens, and in powerUps. TimeWastingAlien, ShooterAlien, and BlindAlien are all subclasses of the Alien class, which contains more general instances and methods that all three kinds of aliens have in common. Similarly, ExtraTime, ExtraLife, Hint, PlasticBottle, and ProtectionVest are all subclasses of the PowerUp class. By using

polymorphism, we can iterate through lists of Alien/PowerUp and do the same jobs in the same ways on all kinds of Aliens/PowerUps without knowing their exact type.

Factory: The factory pattern is useful when object creation has a lot of details. A factory class encapsulates the object creation for all types of the object on hand. In our code, the factory pattern is used for creating aliens and power ups since they have multiple subclasses. In the AlienFactory class, aliens of different Alien subclasses (BlindAlien, ShooterAlien, and TimeWastingAlien) are created efficiently. Similarly, in the PowerUpFactory class, power ups of different PowerUp subclasses (ExtraLife, ExtraTime, Hint, PlasticBottle, and ProtectionVest) are created here. With this pattern, low coupling is achieved.

Controller: Our software applies controller pattern to achieve model-view separation. In our software, UI layer elements take the input from the user. Then, they send these inputs to their corresponding controller classes which we call handlers (e.g. RunningModeFrame feeds user inputs to RunningModeHandler). These handlers take the inputs, do the necessary calculations, and give their outputs to the UI. UI then shows these outputs to the user. We also have a Controller class that controls the main flow of the game.

Singleton: In our code, our handler and frame classes are singleton, meaning that instead of creating an instance for them, we use a static instance, and other classes call getter of this instance to reach the content of these classes. We only need one instance of these frames and handlers each, and we cannot make these classes' methods static because we need them to be instance methods so as to provide object-oriented communication between classes. With this pattern, we avoid creating multiple frames/handlers of the same type while one is enough.

Strategy: The aliens of our game have different move and attack behaviors depending on their types. Because of this variation, the strategy pattern is implemented to represent each behavior of aliens. In our code, the Alien subclasses use a move and an attack behavior represented by interfaces IMoveBehavior and IAttackBehavior. This way, implementations of behaviors and Alien subclasses are separated and not locked into each other so other types of Aliens can reuse move and attack behaviors. If wanted, new behaviors can be added without changing Alien subclasses and existing behavior classes. There are 3 different move behaviors and each of them are specified in the following classes that implement IMoveBehavior interface: MoveToSound, NoMove, and RandomMove. Similarly, 5 different attack behaviors are specified in the following classes that implement IAttackBehavior interface: IndecisiveWaste, NoWaste, WasteTime, MeleeAttack, and RangedAttack.

Adapter: In our code, the adapter pattern is used to implement the Save/Load functionality.
Our game provides two options to the player: saving to a database or to local files using a file system. To choose one of these options, the player needs to press the save/load button in the login mode frame. In this switching between save/load systems implementation, we used the adapter pattern such that both save/load mechanisms (file and database) implement the same interface ISaveLoadAdapter to save, load, or delete the game data.

The flow of our game is as follows: User chooses the save/load system, logs in to the game, enters the building mode to design buildings, then enters the running mode to play the game. If we did not have implemented this pattern, when user decided to switch between the save/load systems it would require changing the code. To avoid that, the adapter pattern is implemented to adapt the user's existing game progress to the game without changing the whole code via adding interfaces like a SaveLoadAdapter.

Supplementary Specifications

Functionality

- **Building Mode:**

Left click: Player left clicks on one of the objectAdders to create new object, or an already existing object, then drags the pressed object to the location player wants it to be located at.
Right click: Player right clicks on an already existing object to delete that object.

- **Running Mode:**

Right key: Character moves to the right as the player presses the right arrow key.
Left key: Character moves to the right as the player presses the left arrow key.
Up key: Character moves to the right as the player presses the up arrow key.
Down key: Character moves to the right as the player presses the down arrow key.

Left click: Player left clicks on objects to find the key. To click objects, the player must be near the objects.
Right click: Power-ups are collected when the player right clicks on the power-ups.

H key: To use the "Hint" power-up, the player presses the H key. Works only if the player has the "Hint" power-up.
P key: To wear the "Protection vest" power-up, the player presses the P key. Works only if the player has the "Protection vest" power-up.
B key: To activate the "Plastic bottle" power-up, the player presses the B key. Works only if the player has the "Plastic bottle" power-up.
After activating the "Plastic bottle" power-up, another key has to be pressed to move the bottle. These keys are as follows:
W key: The bottle moves in the north direction when the player presses the W key. Works only if the "Plastic bottle" power-up is activated.
A key: The bottle moves in the west direction when the player presses the A key. Works only if the "Plastic bottle" power-up is activated.

X key: The bottle moves in the south direction when the player presses the X key. Works only if the “Plastic bottle” power-up is activated.
D key: The bottle moves in the east direction when the player presses the D key. Works only if the “Plastic bottle” power-up is activated.

Usability:

The font used in the game should be clear and readable by the user since it will provide crucial information such as time, remaining lives and inventory of the character. The power-up icons should be clear and distinguishable from each other. All the objects in the buildings should be visible to the player. Animations should be smooth and understandable.
Game should support a load/save system such that player can quit the game after saving it, and when they return to play it again, they should be able to resume their game by entering their unique username and password while logging in.

Reliability:

The game should not crash often. If errors occur (i.e. when power-ups are collected etc.) the game should have a way of troubleshooting.
If the game crashes, the player should be able to restart the game from the last saved point.

Performance:

The character should move smoothly. Blind alien’s moves should be smooth as well. Game events (such as finding a key or gaining extra lives etc.) and user interactions (such as pausing/resuming and quitting the game) should occur as quickly as possible and be visible on screen.

Supportability:

The system should adapt to different screen settings. Also the system should support multiple computer OSs.

Glossary

Term	Description
Player	User who is playing the game.
Character	Player controls the character throughout the game to find keys and escape all the buildings.
Shooter Alien	The alien who shoots bullets at the player.
Blind Alien	The alien who can’t see but can hear sounds. Its choke attack is fatal.
Time Waster Alien	The alien who wastes player’s time by switching the keys’ places in the buildings. Time Waster will behave according to the time left.
PowerUps	Special abilities that help the player to escape the game
Aliens	Aliens are units of the game that spawn randomly around the current building. They have different abilities, and their goal is to stop the player from escaping.
Plastic Bottle PowerUp	A power-up that is used to deceive the Blind Alien. Player can throw the plastic bottle to one of the four directions (North, West,

	South, or East). When player throws it, all the blind aliens in the building will move towards that direction.
Protection Vest PowerUp	A power-up that helps the player not take damage from the bullets. If the player is hit by a bullet while they have an active protection vest power-up, they will not lose a life. Instead, their protection vest power-up will disappear. This power-up lasts for 20 seconds.
Extra Life PowerUp	A power-up that adds one extra life to the player when collected.
Extra Time PowerUp	A power-up that adds 5 extra seconds to the remaining time when collected.
Life Count	Player has 3 lives at the beginning of the game. If player loses all lives they have, they lose the game.
Hint PowerUp	A power-up that gives the player information about keys' locations in the buildings.
Bag	Shows the player what collected but not activated power-ups they currently have.
Key	Required to unlock the doors of the buildings the player is trying to escape.
Door	Gates that allow players to escape the building.
Building Mode	Game's mode where the user places objects into each building. Occurs at the start of the game.
Running Mode	Game's mode where the user plays the game as the player.
Objects	Objects are all the goods, furniture, etc. inside the buildings. Objects might have a hidden key inside of them.
Buildings	Game consists of 6 buildings. Player needs to design these buildings in the Building Mode. Then, they need to escape from

	these buildings one by one to win the game in the Running Mode.
Help Button	Player presses help button to open the help screen. Help screen has information about the building mode, running mode, and their components.
Pause/Resume Button	Player can pause the game state during the Running Mode. Then, they can resume it whenever they want to.
Play Button	When player enters the running mode, they need to press the play button to start the game.

Design Choices

Design Choices: Database

If player enters a username that does not exist in the database, a new game is created. If the username entered exists in one of the previously saved games, and if the corresponding password matches with the one stored in the database, that saved game will be loaded automatically. If entered username exists in the database, but the corresponding password is wrong, player will be warned that the password-username combination they entered is wrong.

If player wins or loses the game, game data will be deleted from the database completely. If player decides to play the game with the same username, a new game will be created.

Design Choices: Building Mode

Player will be asked to build each building one by one. To add an object to the building, they need to press one of the objects on the right side of the game using the left click and drag that object to the building. If they want to relocate one of the objects they have put, they need to press on the existing object with left click and drag it to the place that they want to relocate. If they want to delete one of the objects, they need to press on the object using right click. Once they finished designing the building, they will press the submit button. If their submission is correct, system will accept it and display the next building. If there are no buildings left to design, system will start the running mode.

Design Choices: Interaction With Objects/Aliens/PowerUps

Player cannot pass through the objects, however, they can pass through aliens and powerUps. We decided to disallow moving through objects because it is easier to determine that player is touching the object (they can only collect a hint if they are touching the object) when they cannot pass through it.

Because in the later buildings there are lots of objects, we decided to allow passing through aliens and powerUps, because otherwise player could find themselves stuck, making the gameplay less fun. BlindAlien can pass through the objects, because we wanted to demonstrate it as a scary monster that can pass through anything and instantly kills the player if it touches them.