Middle East Technical University
Department of Computer Engineering

Detailed Design Report for Develop2Learn

Basak Ece CAN
Emel TOPALOGLU
Oytun ÖNAL
Hüseyin Cem ÖZTÜRK

Sponsored by

1/10/2012

Contents

Table Of Figures

1. Introduction

This document describes the detailed design strategies and structural properties of the fun-based chemistry learning game, which will be developed by Kiwi. It explains the data and interface designs of the project with the system architecture in order to help the developers for better design. This document will also guarantee that the design will correctly implement all the functionalities identified in the SRS document, it will be understandable, efficient, and open to upcoming changes.

1.1. Problem Definition

Currently in Turkey, students have difficulties in learning. This is caused by several reasons. One of the biggest problems is the lack of e-learning resources in schools. Currently MEB is preparing a countrywide project to enhance state of e-learning resources with a project called F@TIH[1]. Within the scope of it, MEB is aiming to distribute tablet computers to 13 million students in Turkey. So, they need plenty of e-learning materials.

Our sponsor W2I and we want to develop a learning material for the students that will be different than the ones already developed. It will be a chemistry learning game that will improve teenagers' knowledge of chemistry as well as improving their critical thinking and creativity skills while they are playing a highly enjoyable game.

1.2. Purpose

This detailed design report is aimed to serve as a guideline throughout the development of the project for the developers. It also details what the software requirements are and how they should be implemented.

Its audience consists of developers, which is Kiwi, to give a better understanding of projects, graphics designer to make it clear what should be designed for what purposes, project sponsors to ensure that we meet all the requirements and design project course instructors and teaching assistants to explain them in details what is being developed.

1.3. Scope

The scope of this document contains design patterns of our project. It will be a Unity based chemistry teaching game. Our main audience is high-school students. Our aim is to help them with learning chemistry. While having fun, users will improve their chemistry.

1.4. Overview

This document contains detailed design of our chemistry teaching game. In the introduction part, we will mostly give the problem definition and the scope of the project. The second part is the overall description of the system including our game scenario. Design constraints are mentioned in the third section. After that, data design and architecture design are given with illustrations. Some sketches and screenshots about the user interface are given in the sixth part. We have given information about Unity packages in section seven. The Gantt charts for the terms are given in the eighth section separately.

1.5. Definitions, Acronyms and Abbreviations

F@TIH: Fırsatları Artırma Teknolojiyi Iyilestirme Hareketi Projesi

MEB: Milli Egitim Bakanlıgı (Ministry of Education)

W2I: Words To Inspire

D2L: Develop To Learn

SDD: Software Design Document

DDD: Detailed Design Document

1.6. References

[1] http://fatihprojesi.meb.gov.tr/tr/index.php

[2] http://unity3d.com/support/documentation/Manual/android-API.html

[3] http://unity3d.com/support/documentation/Manual/GameObjects.html

[4] http://www.criticalthinking.org/pages/defining-critical-thinking/766

[5] http://www.uwosh.edu/faculty_staff/gutow/VSEPR_TUTORIAL/AX5_right.html

2. System Overview

Our game focuses on the problem solving abilities and encourages students to think creatively. In order to help students embrace the character in the game there will be a story part at the beginning of the game.

The Story at the Beginning

- Two friends are working in the lab. They make chemical experiments with the tubes of elements and compounds, using erlenmeyers, measuring the temperature of the compounds etc.
- Our character feels tired and goes out.
- When he came back he sees that the lab is messy, there are broken glass and shuffled papers everywhere. Moreover, his friend big-brain is not there.
- Then he founds a paper on the floor and realizes that this is a map that big-brain has forgotten.
- He decides to go out and look for him.
- When the goes out he saw that all the people in the neighborhood are turned into zombies.
- In order to survive and freeze zombies, he takes his liquid nitrogen gun. This starts the game stage 1.

Stage 1

- In stage 1 our character starts with a backpack and a gun that is used to freeze zombies. Moreover, many zombies approach to the character.
- If the character cannot shoot and allows zombies came near then they will hold the character and start to decrease the health.
- When health decreases to zero, the game starts from the beginning of the stage.
- If character is able to fire his/her gun and to freeze the zombies, they will drop different elements that can be collected to use in the game later.
- These elements will be able to be moved into the backpack by clicking on them.
- When user forgets to take the dropped element he/she cannot continue the game and the shininess around the element will increase to take attention.

- While the character goes forward, he/she needs to jump over the obstacles and freeze zombies.
- After beating some other zombies up, the character came across a tunnel entrance closed by metal fences. He cannot jump over, pass through or go around since it is the only entrance into the tunnel.
- Here we are going to give the player a hint that shows that these metal fences can be melted down by throwing acid on them.
- User opens backpack to mix elements in order to get acid.
- There are erlenmeyers in which player can drag the elements and create compounds. When user drags one element onto another a screen will pops up for user to enter the amount of contributions of each element. When they are entered a new screen will pops up saying the name of the compound.
- After creating the compound the erlenmeyer will be poured on the door to check if it can melt the door down or not. If it is acid door will be melted down, if not user will be asked to prepare another compound that shows acidic properties.
- He will be able to try until he finishes all the elements (actually the necessary elements, he will have more kind then necessary).
- When the elements finished the stage starts again and he will collect elements again by freezing the zombies.
- When he is able to create the true compound, tunnels entrance is opened and he is able to continue from the second stage.

Stage 2

- Stage 2 starts into a tunnel whose walls has some clues and tips about Lewis presentation of compounds.
- In this stage the player also needs to beat zombies up. However, this time when they freeze instead of dropping elements they drop stones.
- In order to continue the player should collect these stones by clicking on and sending them into the backpack.
- The health of the player in stage 2 is the same as stage 1. If the player's health decreases to zero, then player needs to start playing stage 2 from the beginning.

- At the end of the tunnel character faces a door on which some element symbols, and holes around them exists. These holes are for electrons they share when elements came together and make a compound.
- Here player needs to select stones in his/her backpack and drag them into the holes on the door.
- If the placement of stones does not fit to the Lewis representation, an earthquake happens and the wrongly placed stones drop onto the floor.
- If the placement of stones fits to the Lewis representation, the door opens and stage 2 ends.

We are planning to add other stages when we finish implementation of these stages. We have planned our data and system architecture in manner allowing adding new levels to the game.

3. Design Considerations

In this section, special design issues needing to be addressed or resolved before attempting to devise a complete design solution are discussed.

3.1. Design Assumptions, Dependencies and Constraints

In the development of D2L, some specific assumptions should be made considering the software and its use.

3.1.1. Hardware & Software Constraints:

D2L project will be developed to work on multiple platforms. So we will have different hardware and software constraints for these platforms.

For tablet computers (main purpose):

- Operating System: Android 2.0 or higher
- Processor: ARMv7
  (Possible FATIH devices will have Single core 1 GHz or Dual core 800 MHz)
- Memory: 512 MB

For personal computers:

- Operating System: Windows ® 2000 or higher/Mac OS X 10.4 or higher
- Processor: 2 GHz
- Memory: 512 MB

3.1.2. End-User Characteristics:

End-users in our case high school first year students should have a general knowledge of chemistry to be able to play game and pass levels.

3.1.3. Time Constraints:

The project started in the beginning of 2011-2012 academic year. By end of the first semester, the first level is planned to be completed to be able to test it with several students. Before 2012 June, we are planning to complete all coding part to be able to deliver it to all students before the beginning of 2012-2013 academic year.

3.1.4. Graphics Constraints:

Since the game's target audience is high school first year students, graphics should be cool and fancy enough to get their attention. Also, while making them that good, device specifications should also be considered.

Since the project's main platform will be the tablet computers that will be distributed by MEB and those devices may not be that fast, our purpose is to keep it as simple as simple as possible. So an implementation of a nice graphics 2D game will be the best solution.

3.2. Design Goals and Guidelines

While designing the D2L project our main purposes are to make it adaptable, sustainable and extensible and to develop critical thinking skills. Since some goals like desirability, ease of use, entertainment factor are obvious, we will not discuss them here.

3.2.1. Extensibility

High school fist year chemistry curriculum is so large that makes it impossible for us to implement game levels for each of these topics within such short time. Whenever a new topic is desired to be added, it should be implemented easily. Our aim is to make the game in such a way that it should be easy to extend the game by adding more levels with as few changes in the main parts as possible.

3.2.2. Adaptability

We are designing the game for high school freshmen. However, if desired, it should be possible to make scenario and puzzles to adapt other year's topics by using same design concept.

3.2.3. Sustainability

It is important to make our game sustainable. In the case of a problem we should provide required help and solutions.

3.2.4. Promote Critical Thinking

According to Michael Scriven & Richard Paul[4] Critical thinking is the intellectually disciplined process of actively and skillfully conceptualizing, applying, analyzing, synthesizing, and/or evaluating information gathered from, or generated by, observation, experience, reflection, reasoning or communication, as a guide to belief and action. Critical thinking is essential to effective learning and productive living. This is our main motivation when starting this project.

Currently there are lots of chemistry games but none of them promotes critical thinking skills in students. In our game, we are planning to develop such skills of students by reasoning, questioning and investigating.

4. Data Design

4.1. Data Description

In Unity development platform in order to ease our job we are planning to create some folders and put our data files into them. Following part is the list of these folders.

Scenes:

The scenes we are using in the game will be hold in this folder. The files have an extension special to Unity.

Libraries:

The packages we loaded from the Unity will be placed in this folder.

Scripts:

The scripts we write are going to be in this folder. Multiple extensions are allowed to be combined in one scene. JavaScript and C# scripts can be used.

Game Objects:

Unity allows us to create one of the following game objects that is coming with itself:

- Particle System
- Camera
- GUI Text
- GUI Texture
- Directional Light
- Point Light
- Spot Light
- Cube
- Sphere
- Capsule
- Cylinder

- Plane
- Cloth
- Empty Object (will be used to create our own object)

We will select some game object among these and add them under game objects folder.

For instance, for main character that is walking on the screen, a cube object with different textures will be used. Or for our main camera, we will use camera object and add our scripts to it in order to make it move as we like. And for other game objects of the game we will create empty objects and modify them in the way we like.

Materials:

The materials we are going to load into objects are going to be hold in this folder. This will include which shadier model is going to be used, the color of the object and the texture info of the material. The extension of files is .mat.

Textures:

The 2D graphics we are going to use will be placed in this folder. They can be any picture extension, but we are planning to use .png files.

Player Info:

This entity will be stored in *_info.kiwi file which is a special kind of data file. We will create this entity by ourselves and write some classes for keeping game data in a proper manner.

The first class we are planning to implement is GameConfigurations. In this one will keep game difficulty, current player, achievements etc. Another class that is required is LevelInformation which keeps user's game statistics within the level like health, score, zombies killed, time elapsed. We are planning to add more classes in case of requirement. Information of these classes will be saved to player's hard drive as game_info.kiwi or level_info.kiwi by saveGame method that we will implement. And

whenever a new game is started loadGame method will read all *_info.kiwi files to load game back to its previously recorded state.

4.2. Data Dictionary

Background:

Background will be the texture added cube object used to make user understand current atmosphere of the scene. It may be a street, a tunnel, a door or a laboratory depending on the user's progress.

Backpack:

Backpack will be the object where user keeps his collected items like elements, stones or salt. It will keep list of items in it. Also it will be activated from main menu when its item is clicked.

Big Brain:

Big brain is our character's lab companion who was turned into a zombie accidentally. During the game play character will follow its traces to reach it.
Since our aim will be to return big brain back to human form as well as people of the town. When character reaches it will prepare a solution or a formula to turn big brain into human form.

Camera:

A Camera is a device through which the player views the world. Unity's camera object already has some predefined functions. We will add our functions, to these already existing ones.

Character:

Character will be the most active component during game play. It will be composed of a texture added to a cube object. It will do some actions like jumping, moving, firing etc. In order to make it look like acting we will switch between character's different textures.

Clue:

Clues in the game play will help user to move forward in the game with small tips. It will have a string holding the clue to be displayed.

Door:

Door will be the object where Lewis puzzle is displayed. It will have two states either open or closed. If it is solved correctly it will allow user to pass another level of the game that will be designed later. Otherwise it will be shaken and open a new puzzle.

Element:

Element will be a collectible object that user can collect or combine. Once a zombie is frozen it will be visible to be dragged into backpack. When they are combined with each other they will be deactivated.

Erlenmeyer:

Erlenmeyer will be the game object where user can combine elements into it. When an element is dragged into it, amount of element will be entered and that much of that element will be active in erlenmeyer. After adding another element, result will be a compound in it.

Fence:

Fence is the object that is blocking entrance of the tunnel. It will be closed by default. It will be activated when acid is poured onto it.

Map:

Map will be the object to show the gamer his process in the game level. It will also help user to move through the game levels once activated.

Light:

We will use lights to illuminate the scenes and objects to create the perfect visual mood. We will use point lights, directional lights and spot lights correspondingly, we will not define any special function for them. The most important light for us is the one that we will attach on character.

Zombie:

Zombies will be our enemy in the game we will fight against. When user reaches some point in the game zombie's awake will be activated. When user shots it will be frozen.

4.2.1 Details of Data Objects

In section 4.2 we gave definitions of data objects. In this part we will explain them in a bit more detail. Finally in section 7.3 we will explain processing o them.

In this part instead of writing separate titles for each object, we put class diagrams in the top of objects to act as titles.

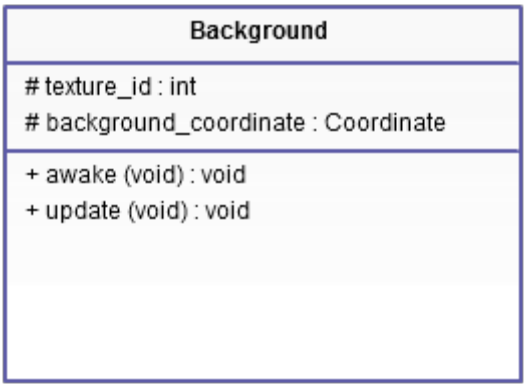


Figure 1: Class Diagram for Background

Background: Background will be the texture added cube object used to make user understand current atmosphere of the scene. It may be a street, a tunnel, a door or a laboratory depending on the user's progress.

texture_id: Id of the background texture.

background_coordinate: Coordinates of background.

awake: Starts background.

update: Updates background.

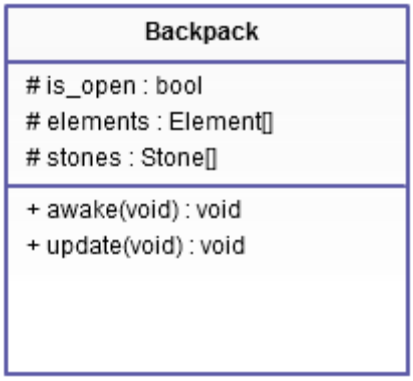


Figure 2: Class Diagram for Backpack

***Backpack*:** Backpack will be the object where user keeps his collected items like elements, stones or salt.

is_open: States whether the backpack is open or not.

elements: List of elements in backpack.

stones: List of stones in backpack.

awake: Starts backpack.

update: Updates backpack.

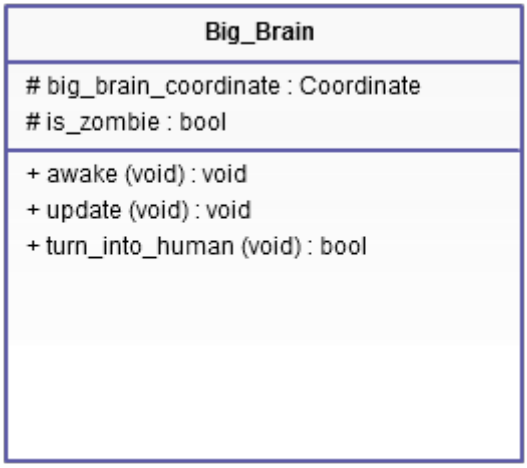


Figure 3: Class Diagram for Bigbrain

***Bigbrain*:** Big brain is our lab companion who was turned into a zombie accidentally. During the game play character will follow its traces to reach it. Since our aim will be to return big brain back to human form as well as people of the town. When character reaches it will prepare a solution or a formula to turn big brain into human form.

big_brain_coordinate: Coordinates of the bigbrain.

is_zombie: States whether bigbrain is zombie or not.

awake: Starts bigbrain.

update: Updates bigbrain.

turn_into_human: Turns bigbrain into human.

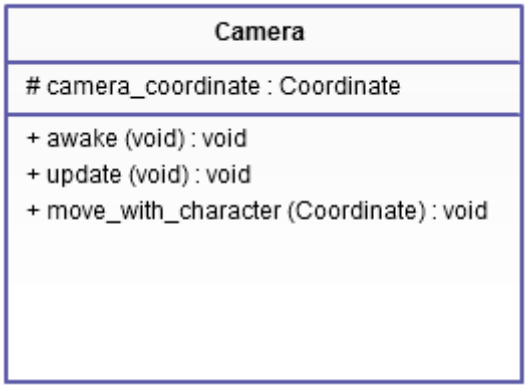


Figure 4: Class Diagram for Camera

Camera: A Camera is a device through which the player views the world. Unity's camera object already has some predefined functions. We will add the ones above, to these already existing ones.

camera_coordinate: Coordinates of the camera.

awake: Starts camera.

update: Updates what is being displayed on the screen.

move_with_character: Changes the coordinates of the camera according to the position of the character.

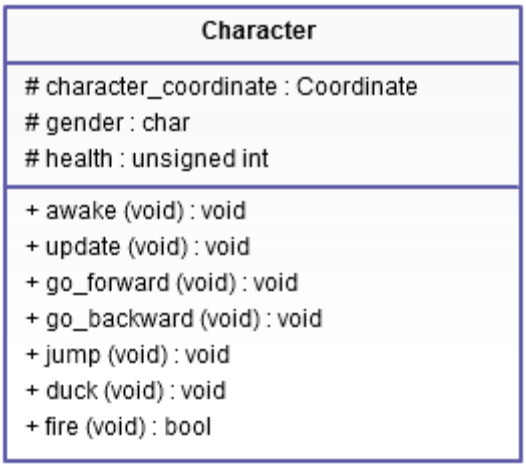


Figure 5: Class Diagram for Character

***Character*:** Character will be the most active component during game play. It will do some actions like jumping, moving, firing etc.

character_coordinate: Coordinates of character.

gender: Gender of the character.

health: Health of the character.

awake: Starts character.

update: Updates character.

go_forward: Moves character forward.

go_backward: Moves character backward.

jump: Character jumps.

duck: Character ducks.

fire: Character attacks.

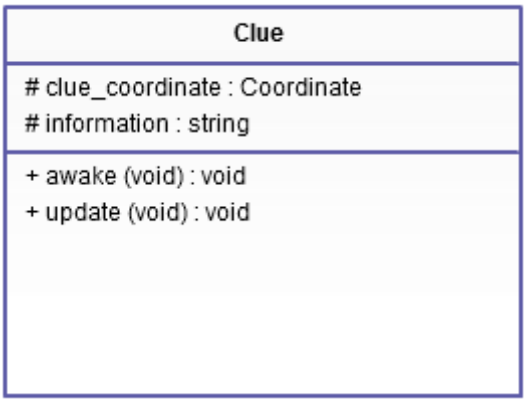


Figure 6: Class Diagram for Clue

Clue**:** Clues in the game play will help user to move forward in the game with small tips.

clue_coordinate: Coordinates of the clue.

information: Holds information.

awake: Starts clue.

update: Updates clue.

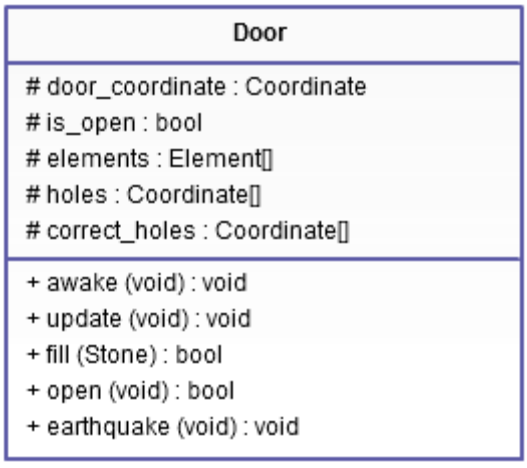


Figure 7: Class Diagram for Door

Door**:** Door will be the object where Lewis puzzle is displayed. If it is solved correctly it will allow user to pass another level of the game that will be designed later.

door_coordinate: Coordinates of the door.

is_open: States whether the door is open or not.

elements: List of elements on the door.

elements: List of holes on the door.

elements: List of correct holes on the door.

awake: Starts door.

update: Updates door.

fill: Fills the holes on the door.

open: Opens the door.

earthquake: Shakes the door.

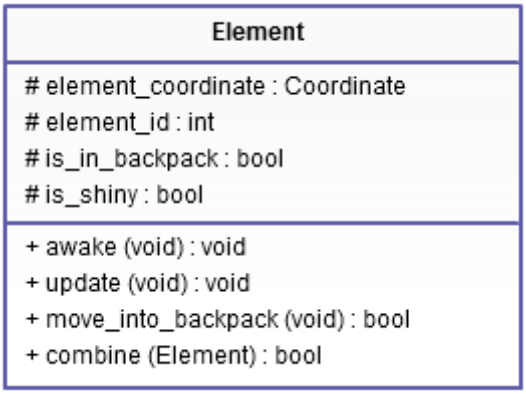


Figure 8: Class Diagram for Element

***Element*:** Element will be a collectible object that user can collect or combine.

element_coordinate: Coordinates of the element.

element_id: Id of the element.

is_in_backpack: States whether the element is in backpack or not.

is_shiny: States whether the element is shiny or not.

awake: Starts backpack.

update: Updates backpack.

move_into_backpack: Moves element to the backpack.

combine: Combines element with some other element.

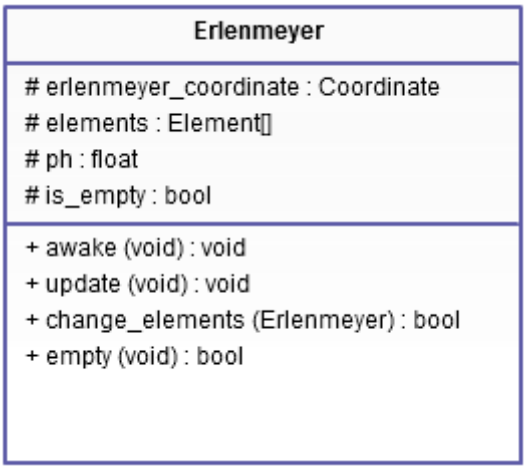


Figure 9: Class Diagram for Erlenmeyer

Erlenmeyer: Erlenmeyer will be the game object where user can combine elements into it.

erlenmeyer_coordinate: Coordinates of the erlenmeyer.

elements: List of elements.

ph: pH value.

is_empty: States whether the erlenmeyer is empty or not.

awake: Starts erlenmeyer.

update: Updates erlenmeyer.

change_elements: Changes elements.

empty: Empties the erlenmeyer.

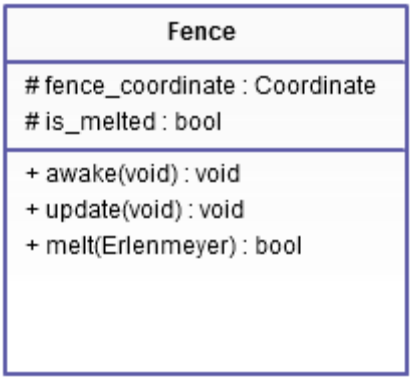


Figure 10: Class Diagram for Fences

***Fence*:** Fence will be the object that needs to be melted in order to get into the tunnel.

fence_coordinate: Coordinates of the fence.

is_melted: States whether the fence is melted or not.

awake: Starts fence.

update: Updates fence.

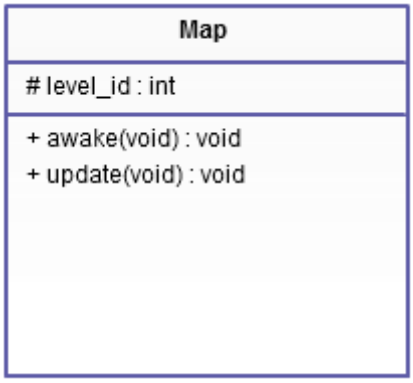


Figure 11: Class Diagram for Map

***Map*:** Map will be the object to show the gamer his process in the game level. It will also help user to move through the game levels once activated.

level_id: Shows which level is passed lastly.

awake: Starts map.

update: Updates map.

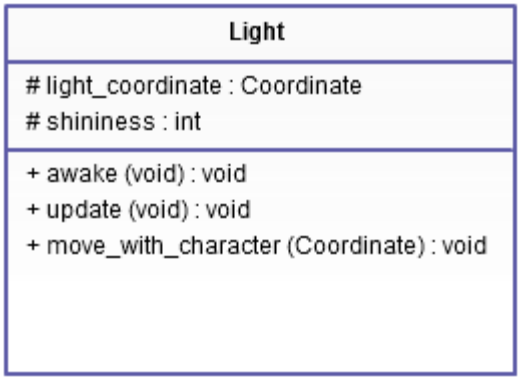


Figure 12: Class Diagram for Light

Light: We will use lights to illuminate the scenes and objects to create the perfect visual mood. We will use point lights, directional lights and spot lights correspondingly, we will not define any special function for them. The most important light for us is the one that we will attach on character.

light_coordinate: Coordinates of the light.

shininess: Shininess of the light.

awake: Starts light.

update: Updates light.

move_with_character: Changes the coordinates of the light according to the position of the character.

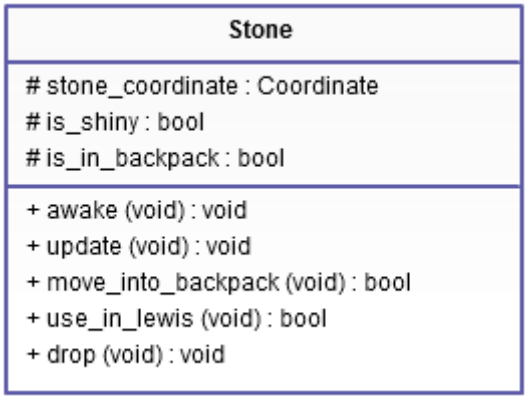


Figure 13: Class Diagram for Stone

Stone: Stone will be a collectible object that user can collect or combine.

stone_coordinate: Coordinates of the stone.

is_shiny: States whether the stone is shiny or not.

is_in_backpack: States whether the stone is in backpack or not.

awake: Starts stone.

update: Updates stone.

move_into_backpack: Moves stone into backpack.

use_in_lewis: Uses stone in Lewis door.

drop: Drops stone.

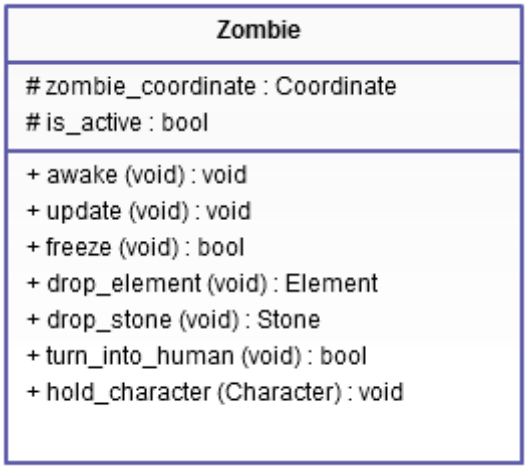


Figure 14: Class Diagram for Zombie

***Zombie*:** Zombie will be our enemy in the game that we will fight against to.

zombie_coordinate: Coordinates of the zombie.

is_active: States whether the zombie is active or not.

awake: Starts backpack.

update: Updates backpack.

freeze: Freezes the zombie.

drop_element: Zombie drops element.

drop_stone: Zombie drops stone.

turn_into_human: Zombie turns into human.

hold_character: Zombie holds the character.

5. System Architecture

A general description of the D2L game system architecture is presented in the following parts of this section.

5.1. Architectural Design

Architectural design of the project D2L is given in the figure 15.

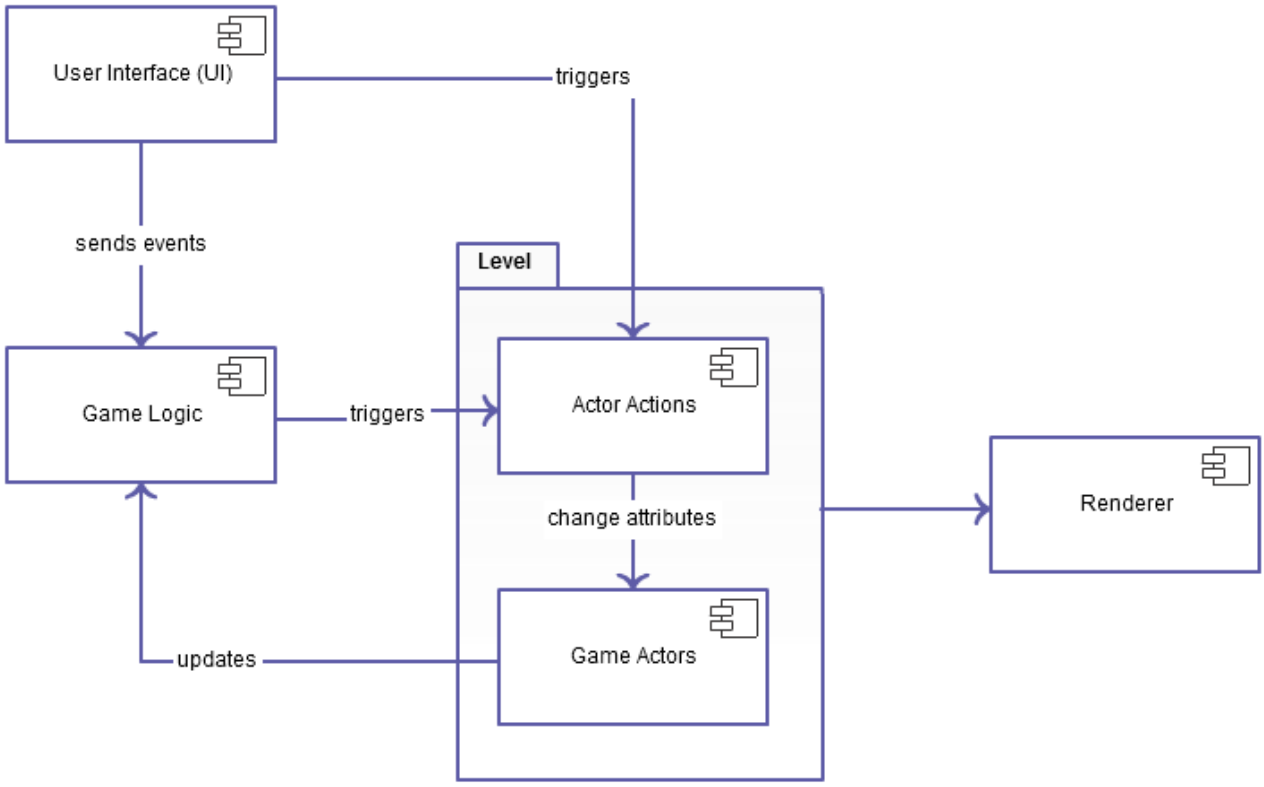


Figure 15: Architectural Design of the Project

The game consists five components. The user interface takes the inputs from the user and processes them. After handling events, it sends the results to the game logic. Moreover, these results are used to determine which actor actions need to be awakened. This is mostly handled by Unity.

Game logic keeps the current game state and according to that manages actions, objects and level changes. It is affected by the user interface and the actor attributes.

Game actors are the units in the game. Units can be game objects, camera, and light. Their attributes are sent to the renderer in order to be displayed on the screen. They have references to the Game Actions module that controls their behavior.

Actor actions keep the functionalities of the game actors. It also involves simple AI scripts. Game actors and actor actions can be thought as the levels or scenes in the game.

Renderer visualizes the levels looking from the camera perspective. Camera determines which objects are going to be displayed and sends them to the rendering pipeline. This module is handled by Unity.

5.1.1 About Unity

As we will use Unity in our development process, we would like to explain its features and characteristics that will effect our design decisions.

The most important specialty of Unity is there is no class, therefore the inheritance logic is absent in the Unity. Every object in a game is a GameObject. However, GameObjects don't do anything on their own. They need special properties before they can become a character, an environment, or a special effect. But every one of these objects does so many different things. In order to achieve these they act like containers. They are like empty boxes that can hold the different pieces that light a room or gravity to drop an object. So to really understand GameObjects, someone has to understand the pieces called Components.

Depending on what kind of object is wanted to be created, one needs to add different combinations of Components to the GameObject. Game object can be thought as cooking pot and components are like ingredients. It is also possible for one to create own component by writing scripts in UNITY. An example of game object without adding any special component is shown in figure 16.

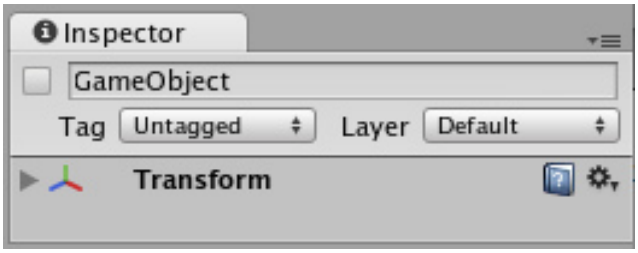


Figure 16: Empty Unity Game Object

An empty GameObject still contains a Name, a Tag, and a Layer. Every GameObject also contains a Transform Component defining the GameObject's position, rotation, and scale in the game world/Scene View. If a GameObject did not have a Transform Component, it would be nothing more than some information in the computer's memory. It effectively would not exist in the world.

In the figure 17, a GameObject called main camera is shown containing a different collection of Components. All of these Components provide additional functionality to the GameObject. Without them, there would be nothing rendering the graphics of the game for the person playing! Rigidbodies, Colliders, Particles, and Audio are all different Components (or combinations thereof) that can be added to any given GameObject.

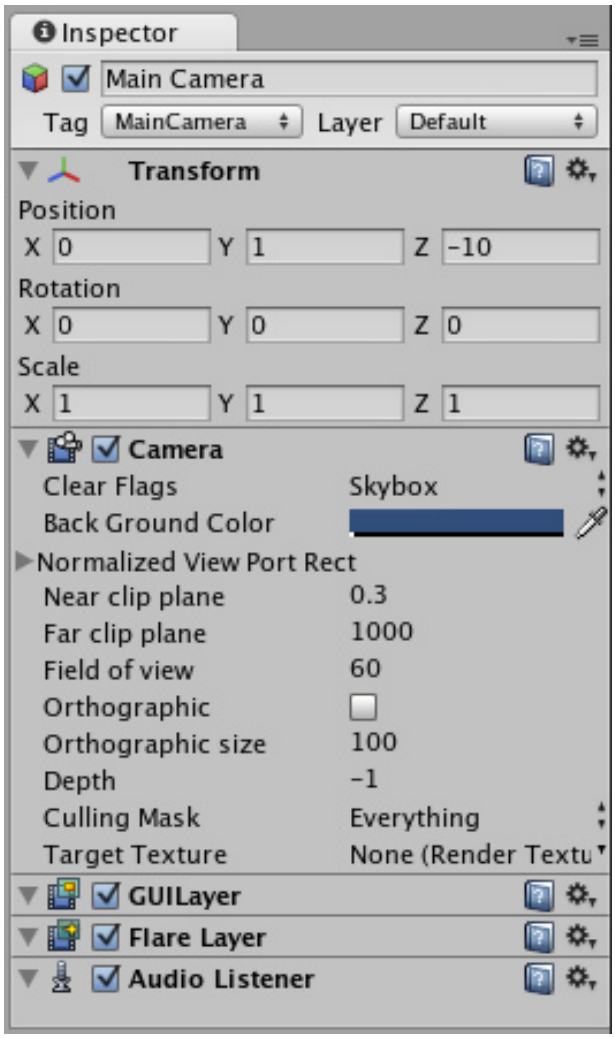


Figure 17: Sample Main Camera Game Object

When a script is created and attached to a GameObject, the script appears in the GameObject's Inspector just like a Component. This is because scripts become Components when they are saved - a script is just a specific type of Component. In technical terms, a script compiles as a type of Component, and is treated like any other Component by the Unity engine. So basically, a script is a Component that is created by user. User will define its members to be exposed in the Inspector, and it will execute whatever functionality is written. Scripting is what we will do mostly in the creation of the game since most of our game's requirements are not met by Unity's default components.

Please note that the components that we mention in this document don't correspond to Unity's Components. Data objects of this document are equivalent of GameObjects in Unity.

Unity also an input library to identify finger moves on screen. We will add this library to check user's taps on the screen for tablet gameplay. Also for PCs we just need to add event listeners to the game_objects in its update() function. In Unity the objects should have two major functions, awake() and update(). In awake() function the initial characteristics of objects are defined. This function is called once. In update() the necessary changes on objects are defined such as moving the object, changing colors, and interactions with the other objects. This function is called to create frames continuously.

5.2. Description of Components

The components we are going to use are explained in the following section in detail.

5.2.1. User Interface (UI)

UI allows us to create a variety of graphical user interfaces with functionality. In addition, it handles the events that have come from an input device.

5.2.1.1. Processing narrative for user interface

User interface module always keeps running behind of the system and waits if any input comes from the user it processes this event and makes this information accessible from other components.

5.2.1.2. User interface interface description

User interface become active when an input comes and after processing it returns it old passive waiting state.

5.2.1.3. Camera processing detail

User Interface module is handled by unity, so that we do not need to create any algorithm for this module.

5.2.1.4. Dynamic behavior of camera

Use interface interacts with game logic and triggers the actor actions.

5.2.2. Game Logic

Game Logic is used to determine the game scenario and to make the flow of the scenario logical.

5.2.2.1. Processing narrative for game logic

Game logic keeps track of the every game unit, the time, and the scenes of the game. It manages the actions of some actors. When an object becomes visible is decided with this module.

5.2.2.2. Game Logic interface description

Game logic can be active when a level is passed, user wants to save the game, objects act and change their state, and when it is necessary to help the connection of the actors each other. Since it knows the whole scenario every little detail of the scenario is handled with this module.

5.2.2.3. Game Logic processing detail

After starting the game, when player move along or some pre-defined time passes, it makes some game objects appear in the game and updates the scene. For instance, it determines when zombies come and start to attack the character in the first level. Furthermore every action made by the player that changes the scene also changes the game logic.

5.2.2.4. Dynamic behavior of Game Logic

A new user creation and starting the game is displayed if figure 18 showing interaction of the game logic with other components.

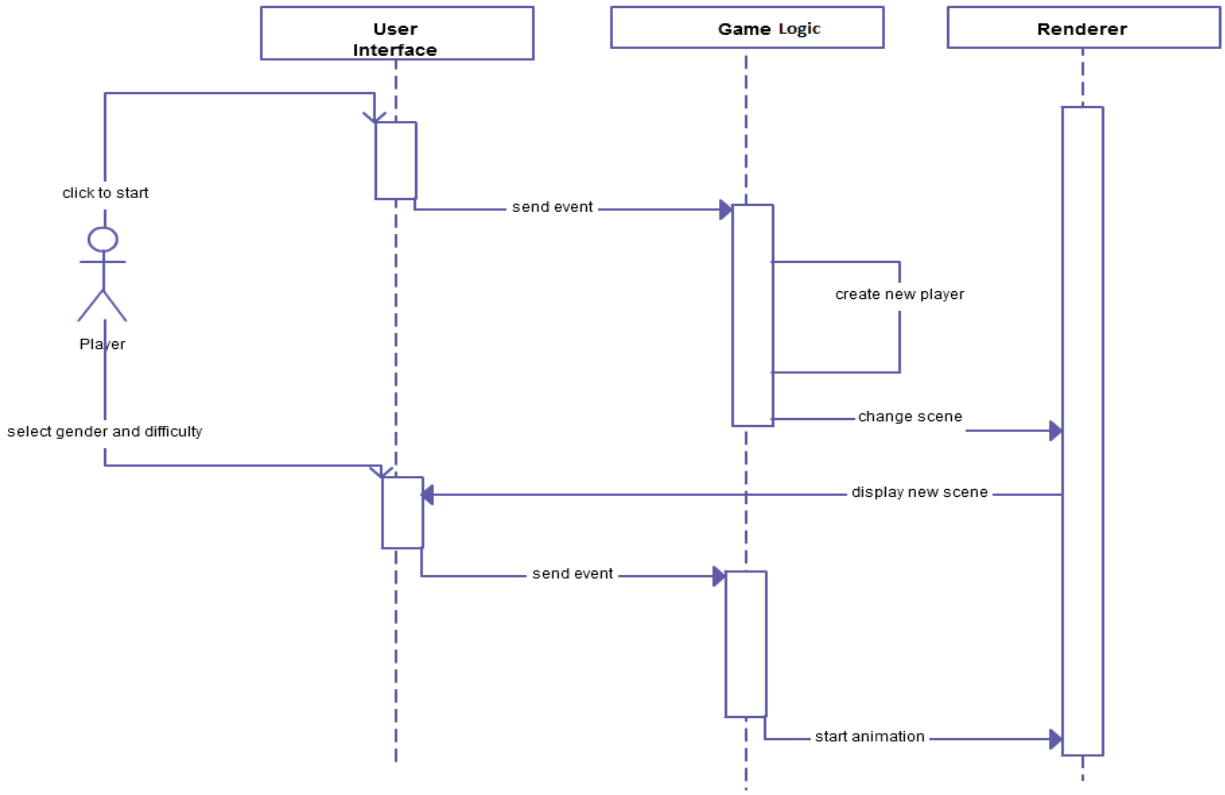


Figure 18: Dynamic Behavior of Game Logic for User Creation

Level selection is shown in the following sequence diagram namely figure 19.

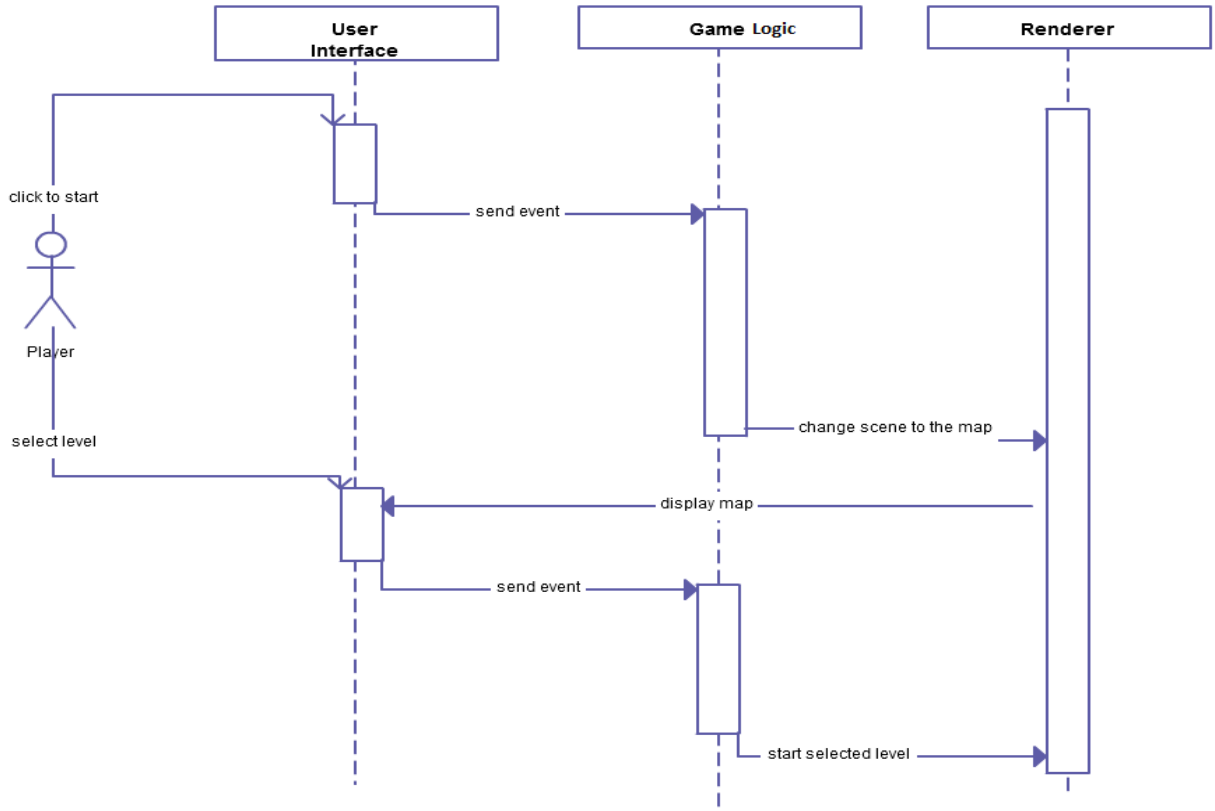


Figure 19: Sequence diagram for level selection

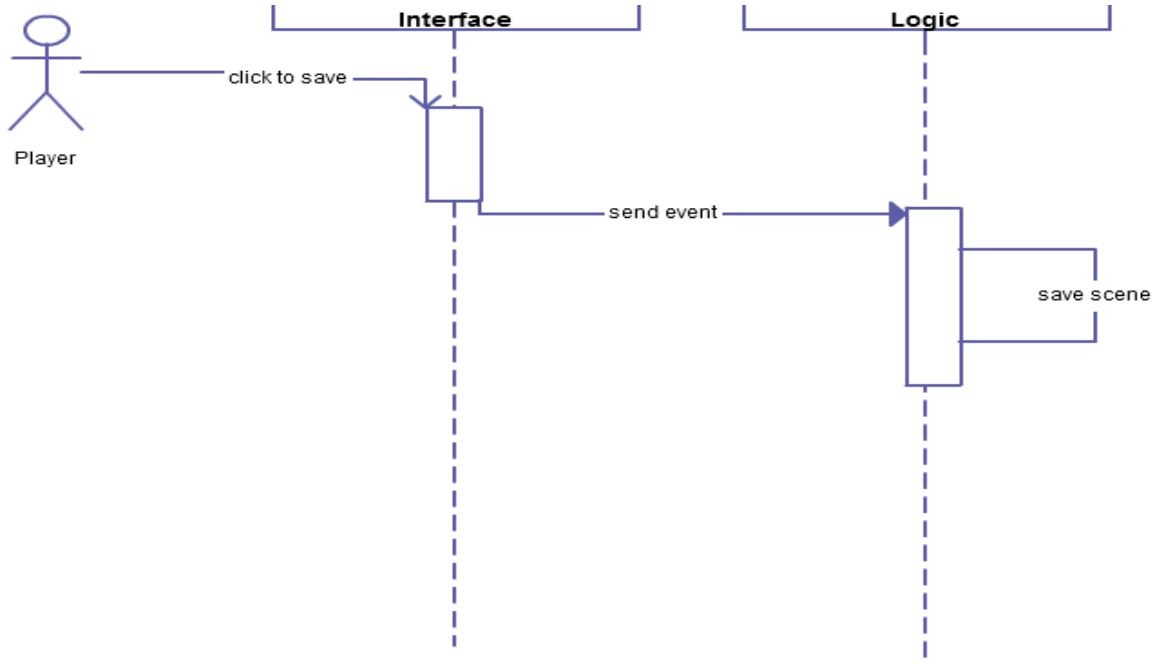


Figure 20: Sequence diagram for saving the game

Opening backpack is shown in the following figure 21.

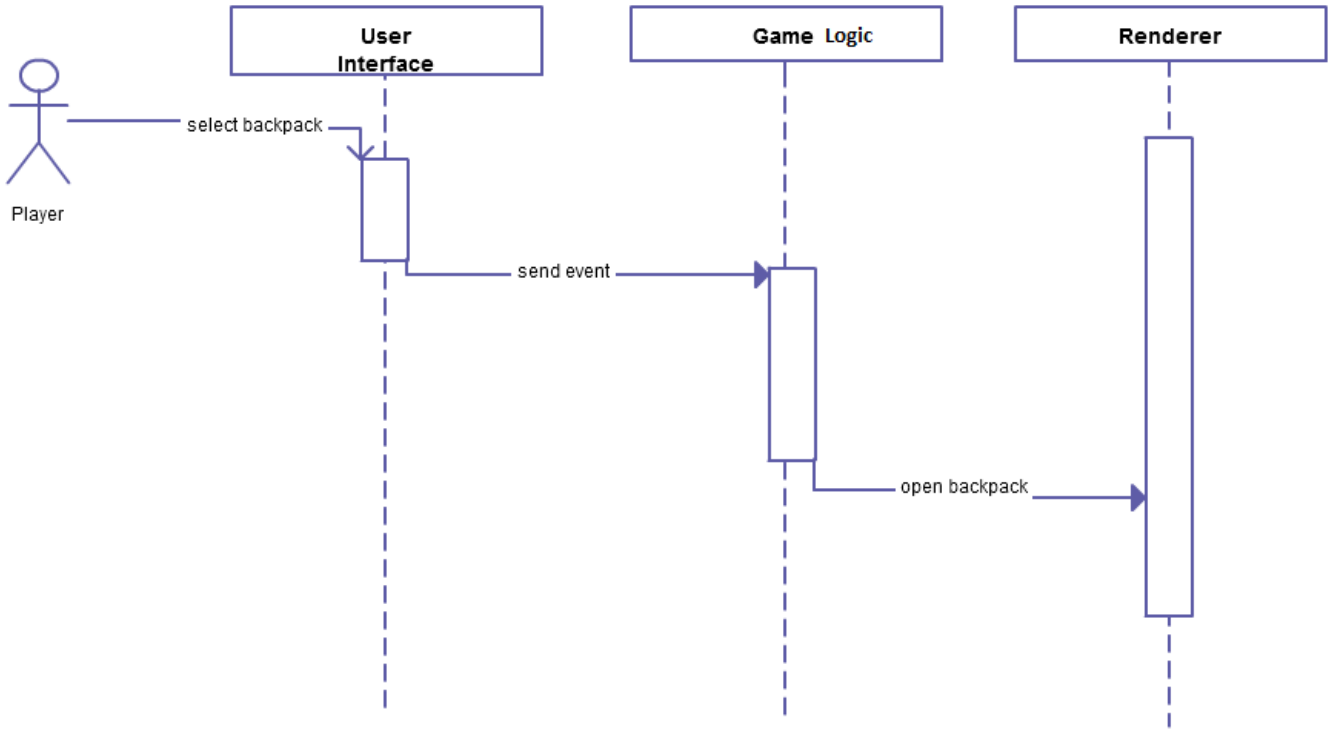


Figure 21: Sequence diagram for level selection

When the hints are going to be given can be seen in figure 22.

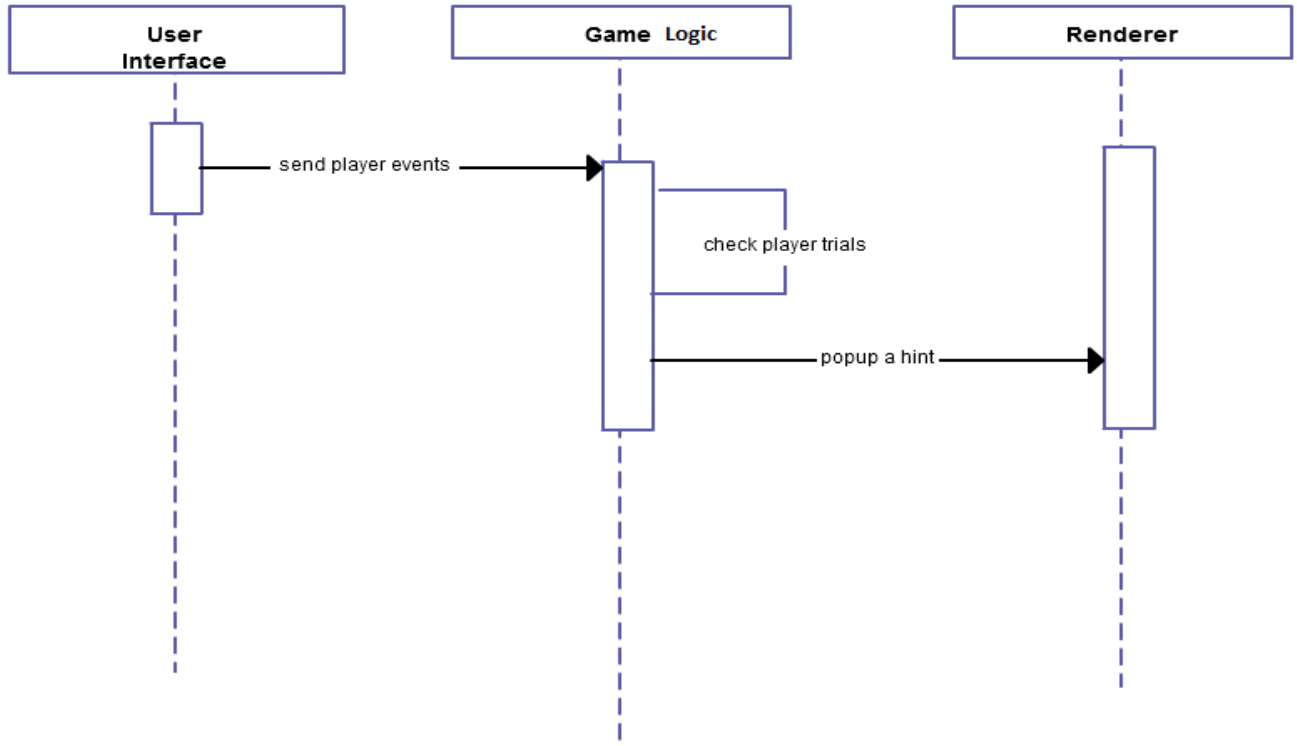


Figure 22: Sequence diagram for hints

5.2.3. Game Actors

Game actors are the units in the game that is usually a game object. Game actors should be added to every scene of the game. Here is the list of game actors in the scenes in figure 23.

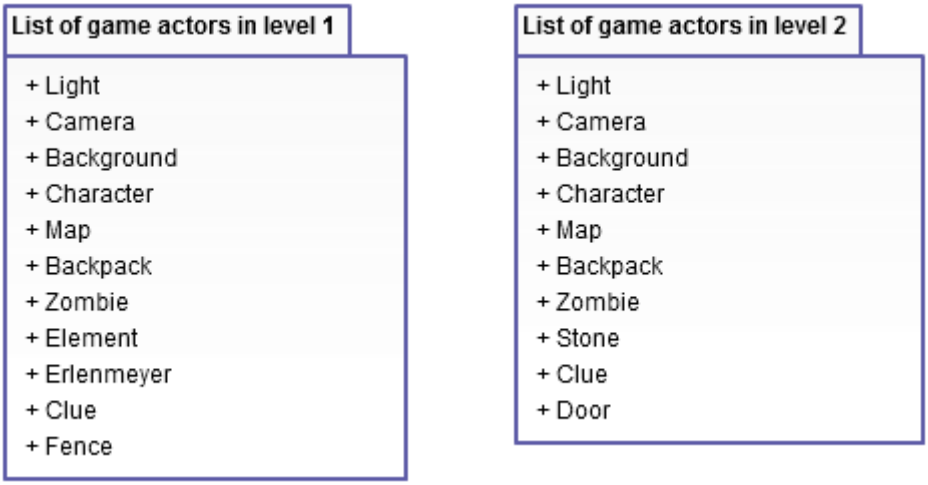


Figure 23: Lists of Game Actors for Level 1 & Level 2

Although, game menu, character selection and map selection screens are different scenes just like levels, their actors are not given here because they only consist one rectangle board with user interface buttons.

5.2.3.1. Processing narrative for game actors

Game actors have the position in the game space, material of the object in order to determine the illumination of the light and the color, and texture attached to them. In the game scenario game actors are created, their attributes, which are just mentioned, are changed and they are destroyed.

5.2.3.2. Game actors interface description

Some game actors are already in the game at the beginning, such as character, background, and zombie. Their states are updated by the specific actor actions.

5.2.3.3. Game actors processing detail

After the game started with the trigger of the user interface and the game logic modules, game actor action functions are called. These actions alter the game actors.

5.2.3.4. Dynamic behavior of game actors

Interactions of the game actor module with other modules are displayed in following figures from figure 24 to figure 30. Their explanations are written under the figures.

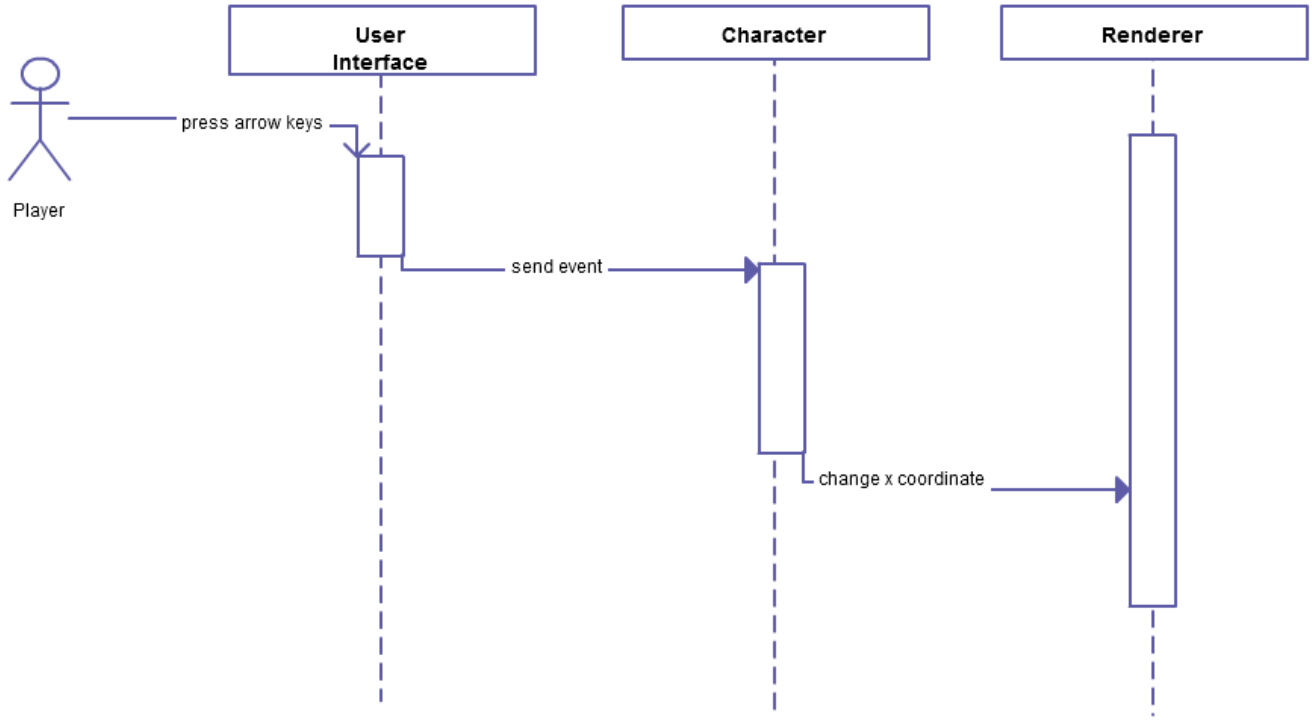


Figure 24: Sequence Diagram for Move Character

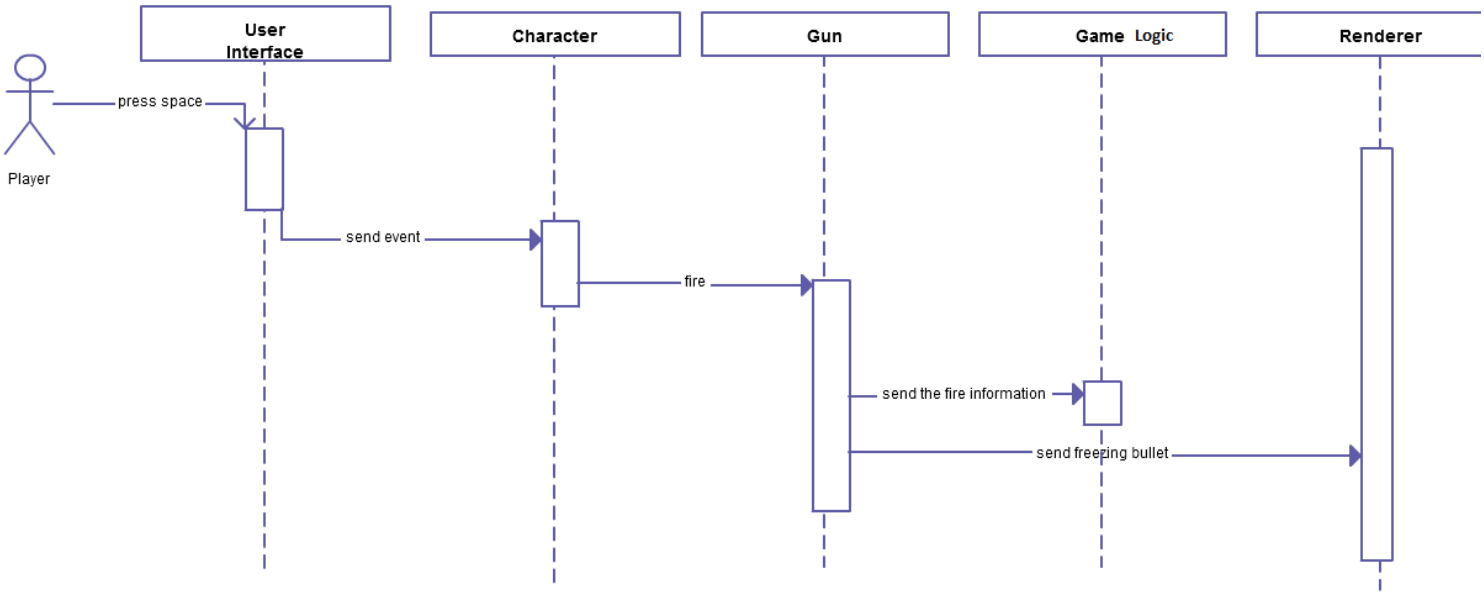


Figure 25: Sequence Diagram for Firing

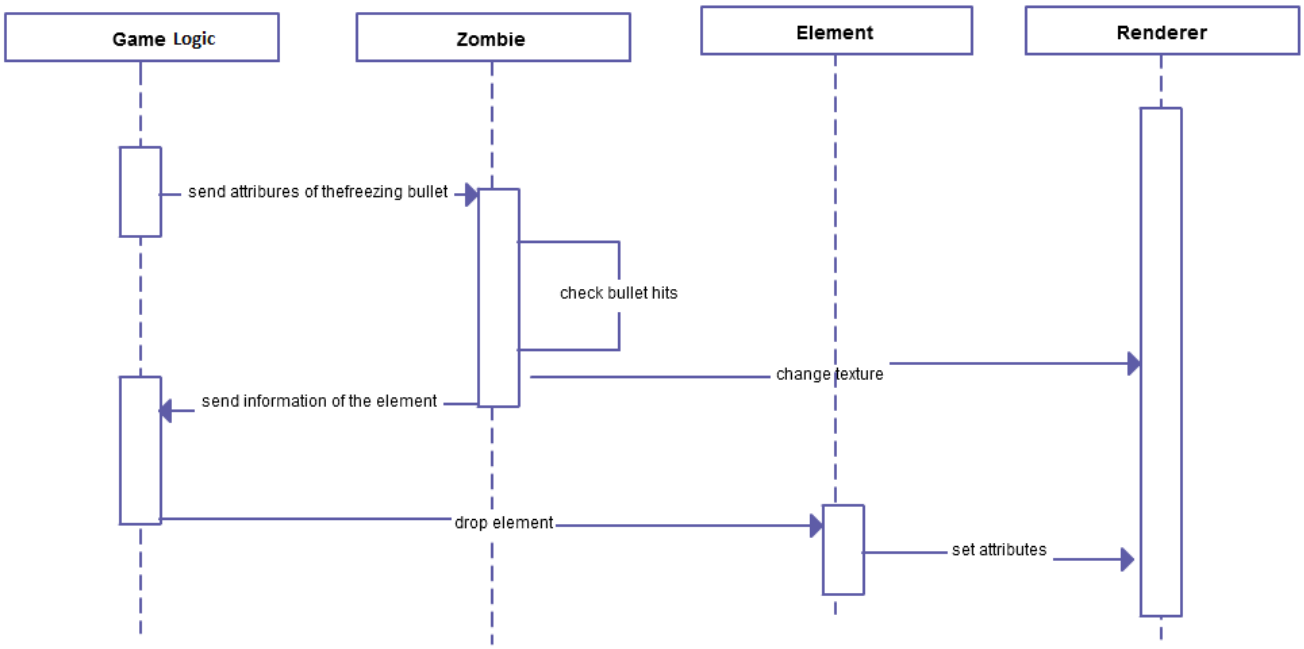


Figure 26: Sequence Diagram for Freeze Zombie

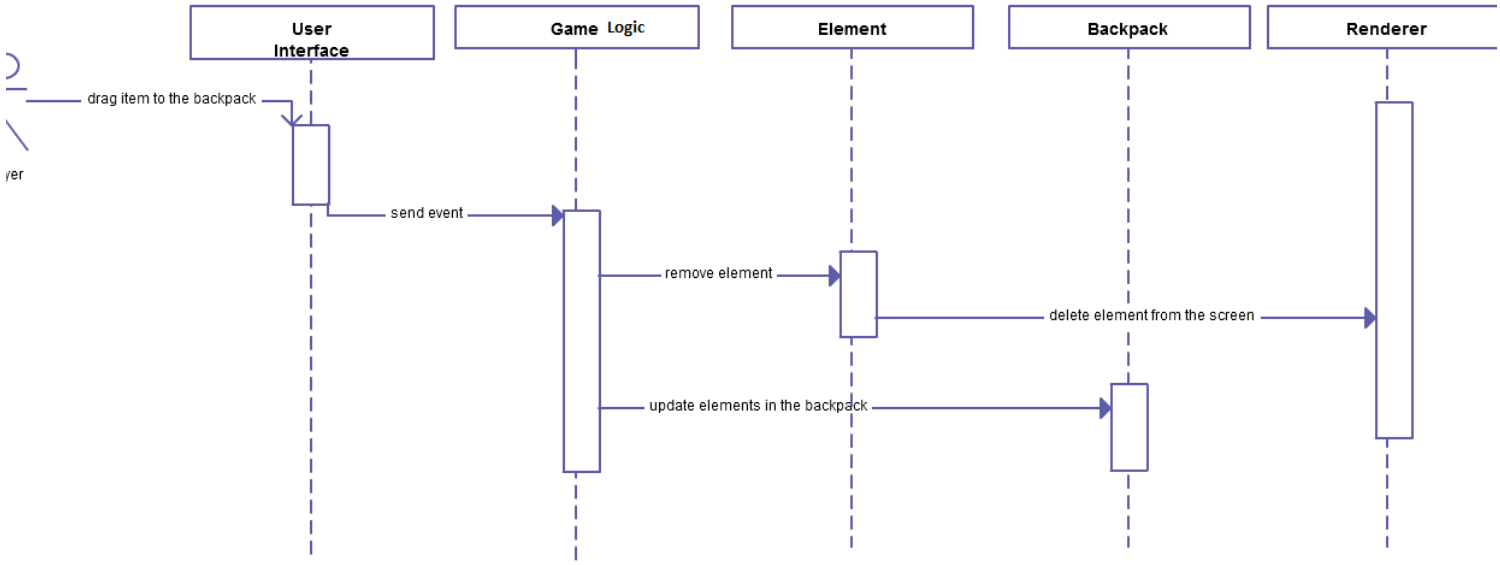


Figure 27: Sequence Diagram for Element Collection

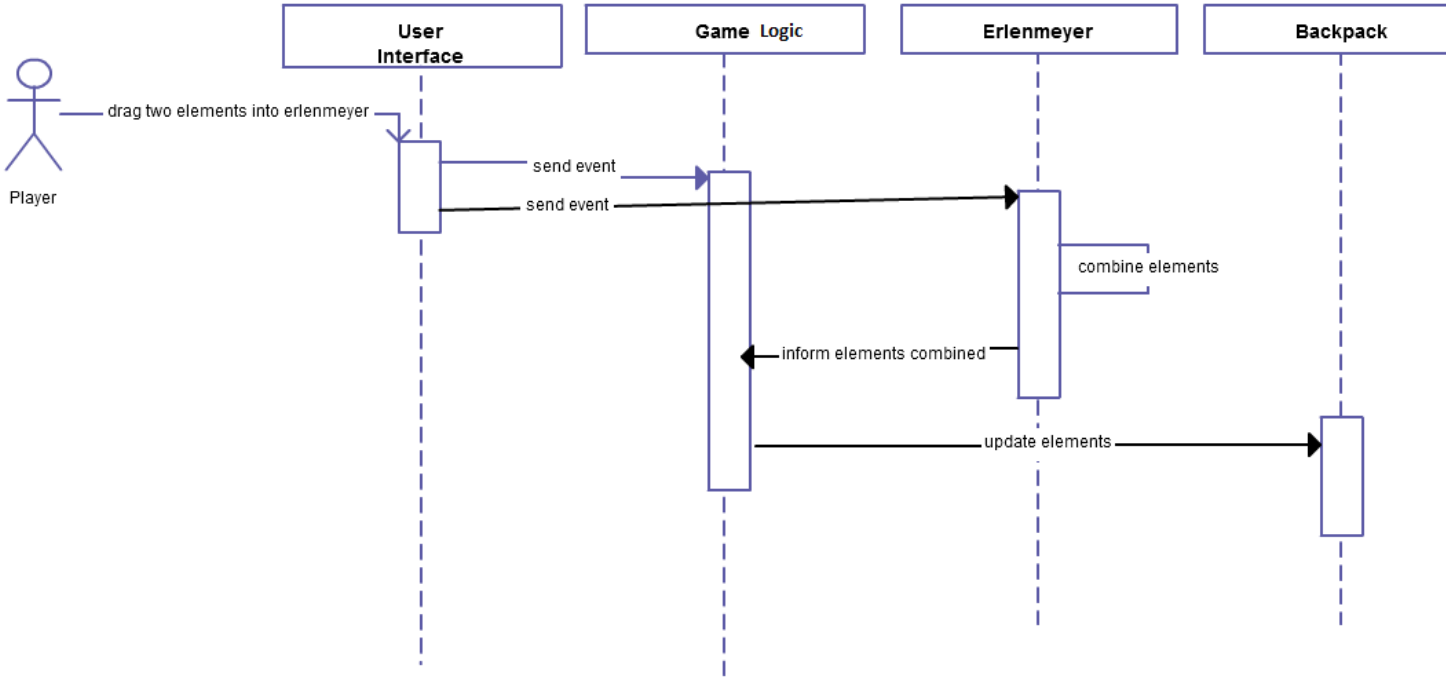


Figure 28: Sequence Diagram for Element Combination

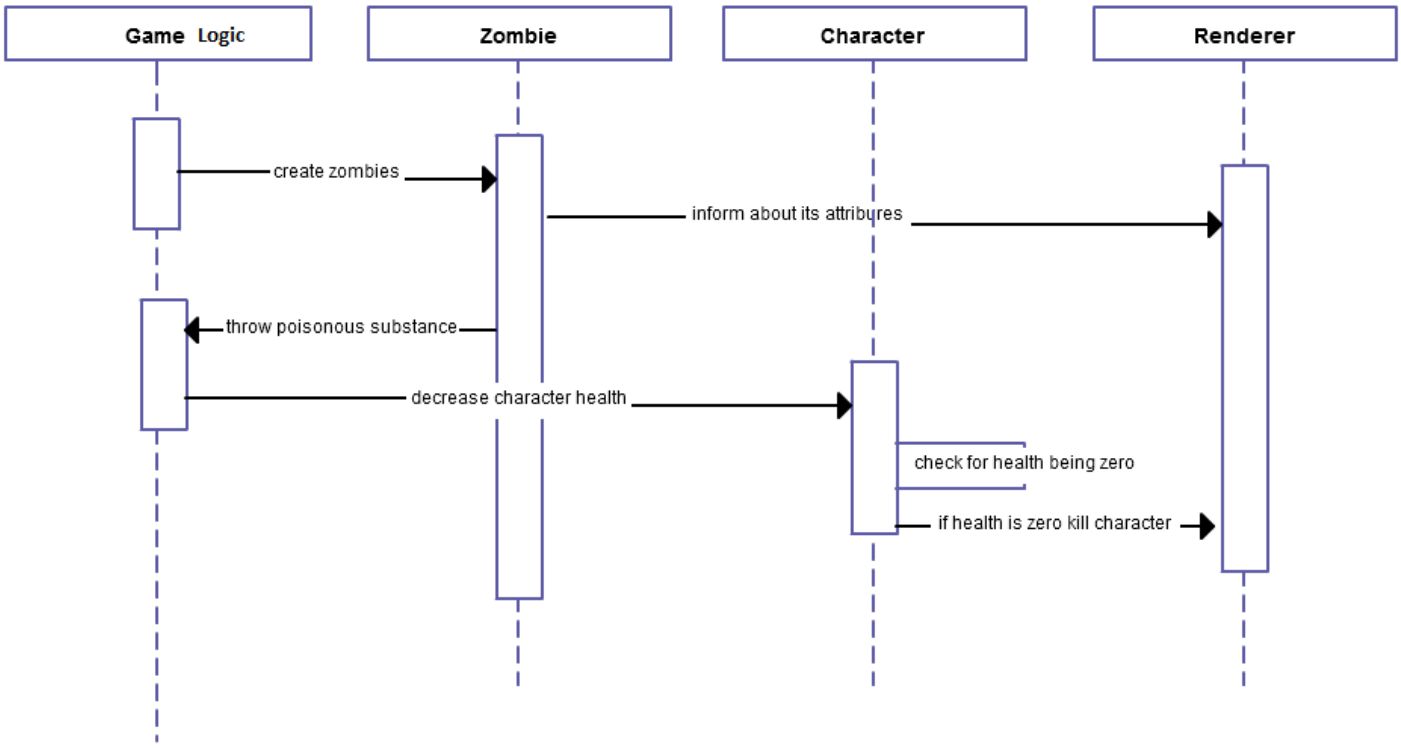


Figure 29: Sequence Diagram for Killing Character

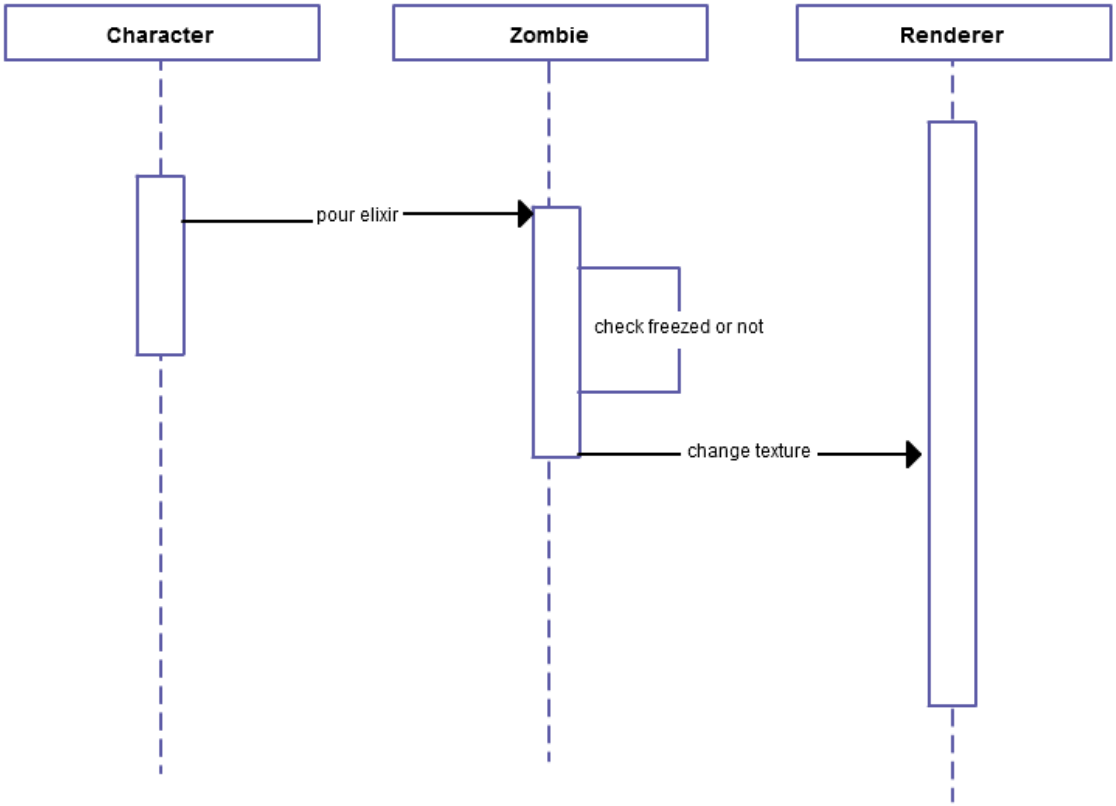


Figure 30: Sequence Diagram for Turning Zombies into Human

5.2.4. Actor Actions

Actor actions are the scripts that are added to the actor in order actors to gain functionality.

5.2.4.1. Processing narrative for actor actions

Actor actions are always active in the scenario. They are called when user interface module takes an input or in the game flow game logic activates a game object.

5.2.4.2. Actor actions interface description

The keyboard arrows and space buttons, mouse clicks, can determine which actor action is called and if the game is played in tablet PCs on screen controls.

5.2.4.3. Actor actions processing detail

Actors have two main functions: awake() and update(). In awake function, their initial attributes are set. Update function is called when each frame is displayed. In update function we call appropriate actor actions.

5.2.4.4. Dynamic behavior of actor actions

Actor actions are embedded into game objects. As it is previously mentioned, they are triggered by user interface and game logic, and they affect game actors.

5.2.5. Renderer

Renderer is the component that is responsible to display the scene that is seen from the camera.

5.2.5.1. Processing narrative for renderer

Renderer is a part of Unity3D development tool. Unity handles this process. Our game will be seen 2D because of the camera position, but the entire scene is actually a 3D environment.

5.2.5.2. Renderer interface description

Without waiting any game object changing its attributes renderer always working to display frames. Unity can render over 9000 frames per second (fps). However, it is highly dependent to the complexity of the game objects and the graphics card.

5.2.5.3. Renderer processing detail

Scenes main camera determines the objects that are going to be rendered. Then sends them to the rendering pipeline. Unity allows users to access the surface and vertex shaders so that they can change the default settings to get more realistic images.

5.2.5.4. Dynamic behavior of renderer

Renderer directly related to the game actor and game logic components.

5.3. Design Rationale

We are forced to use Unity while developing our game, so we chose this design to be parallel to the Unity Development Tool.

5.4. Traceability of requirements

When we look at Unity deeply we have seen that most of the requirements we have determined do not match with the Unity. Nearly all the requirements are in the Unity packages. We do not need to make a design for them. But still we made a traceability matrix that shows which component satisfies which requirement by using section numbers as can be seen from the figure 31.

DDD Reference Section	***Functional Requirements***	***SRS Reference Section***
7.1	Create new character	3.2.1
7.1	Chose character	3.2.2
7.1	Choose avatar	3.2.3
7.1	See achievements	3.2.4
7.2	Game control	3.2.5
7.2	Feedback	3.2.9
7.4	Kill enemy	3.2.6
7.4	Collect item	3.2.7
7.4	Kill character	3.2.8

Figure 31: Traceability of Requirements Matrix

6. User Interface Design

6.1. Overview of User Interface

For our chemistry teaching game, graphical user interface, which will provide users an initiative experience, will be implemented. Our main purpose, while designing GUI, is to increase playability as much as possible. Moreover GUI should be user-friendly. User plays the game via control button implemented on GUI. There will be direction controller which enables user to make their character move. Also attack button and jump button will be provided. In addition hints and tips will be available through GUI.

6.2. Screen Images

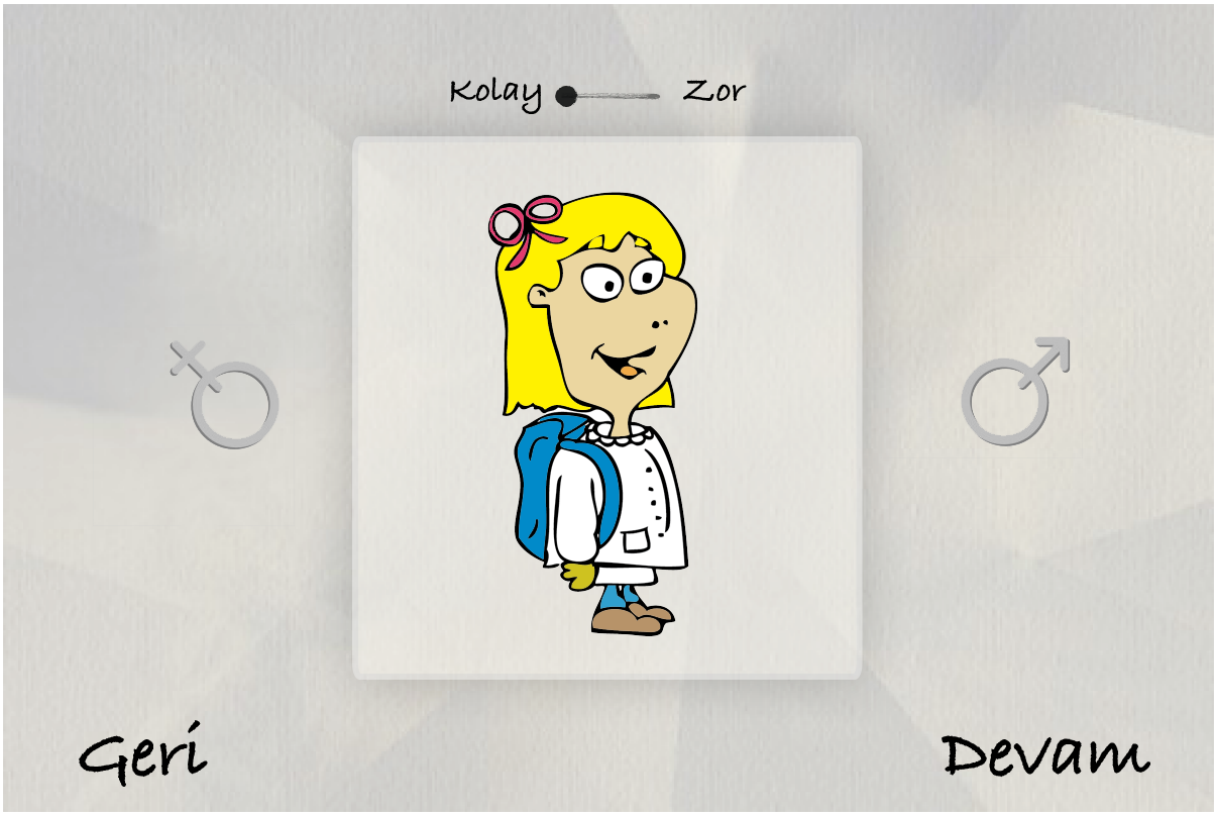


Figure 32: Character Selection Screen

Figure 33: Main Menu Screenshot

Figure 34: Character And BigBrain Working in the Laboratory

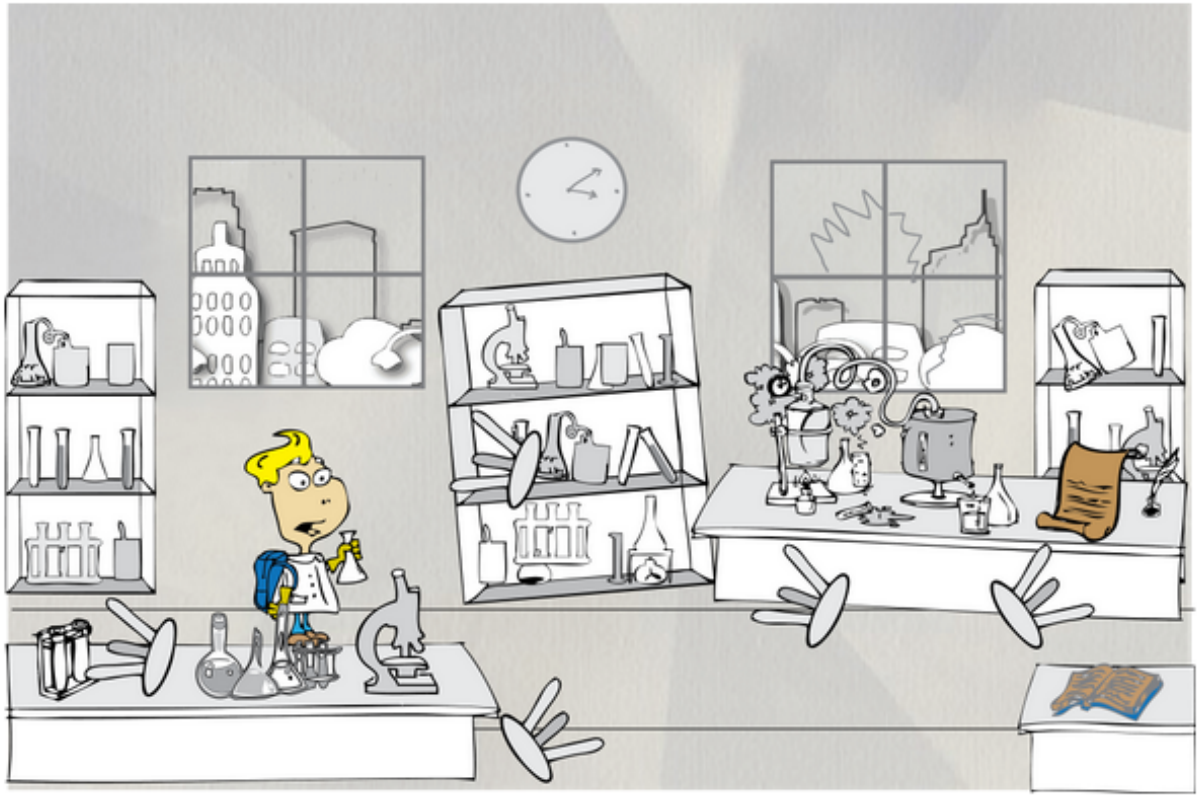


Figure 35: Screen Shot Showing When Character Returns, BigBrain Has Escaped

Figure 36: Screen Shot Showing Character Taking the Map

Figure 37: Screen Shot Showing Character Facing with a Zombie

6.3. Screen Objects and Actions

In first screen there is going to be:

- Play Game
- Options
- Achievements
- Quit

In the map screen there is going to be:

- Acid and bases
- Lewis Structure
- 3rd concept (TBD in the second term)
- 4th concept (TBD in the second term)

In the game screen there is going to be:

- Direction
- Jump
- Attack
- Map
- Tips
- Backpack
- Appropriate game texture

7. Detailed Design

7.1. User Interface

7.1.1. Classification

User interface is a component in our game whose functions are mostly handled by Unity.

7.1.2. Definition

As it can be understand from its name this component handles with everything on the screen. It takes user's inputs from the screen and processes them to make them possible to use in other components.

7.1.3. Responsibilities

This component is mainly responsible from displaying game graphics to user.
It is also responsible from taking user inputs from the screen by using Unity's input library. After that it needs to process them in order to send results to game logic, such as "play game button is clicked". These inputs are also used in the determination of which game actor is being called.

7.1.4. Constraints

Since this component is responsible for taking user's input and process it, it should respond as fast as possible to user's interactions with the screen. For instance if user clicks shoot button repeatedly with time interval less than 0.5 seconds, UI should respond to these clicks like less 0.1 seconds so as to give enough processing time for other components.

It is also important for tablet interfaces to make them perceive only one finger touch on the screen since we don't use multi touch controls. We should ignore second finger's touch if there is already a finger moving on the screen.

Another constraint of UI is to make it stable. Stability should be concerned every part of

our project since it is a game and user's don't want it to freeze in the middle of the game play.

7.1.5. Compositions

As it can be seen from the figure 15 referencing game architecture this component is not composed of any other classes.

7.1.6. Uses/Interactions

User interface sends events to game logic and triggers actor actions as it can be seen from the figure 15. As it processes user inputs it should call game logic to check if there is a scene change, level change, opened backpack view etc. Also it should call required game actor such as frozen zombie, tunnel entrance etc. then display them on the screen by placing them their game coordinates as saved in their Unity GameObjects.

7.1.7. Resources

To be able to use UI component we will mostly use Unity's default scripts, functions and libraries. We will use character controller and touch libraries of the unity to process user inputs and write our scripts to process them.

Unity itself controls what is being displayed on the screen and calls update function when required.

7.1.8. Processing

The user interface handler makes the suitable implementations after getting the keyboard actions or tablet controls. After gathering the information from the input handler, it makes processes them in order to call required game component. Then, it displays results of these components (game actors and game logic) on the screen. An example load level code is given below.

```
/* Example level loader */

function OnGUI () {
	// Make a background box
	GUI.Box (Rect (10,10,100,90), "Loader Menu");

	// Make the first button. If it is pressed, Application.Loadlevel (1) will be executed
	if (GUI.Button (Rect (20,40,80,20), "Level 1")) {
		Application.LoadLevel (1);
	}

	// Make the second button.
	if (GUI.Button (Rect (20,70,80,20), "Level 2")) {
		Application.LoadLevel (2);
	}
}
```

7.2. Game Logic

7.2.1. Classification

Game logic is a component in the game that will include scripts to control scenario flow.

7.2.2. Definition

Game Logic is used to determine which part of the game scenario is currently running and makes logical switches between parts of the scenario.

7.2.3. Responsibilities

Game logic is mainly responsible for providing smooth run of the scenario. Game logic keeps track of the every game unit, the time, and the scenes of the game. According to commands of UI it changes game actors.

It is also responsible for deciding which unit should be displayed on the screen now. For instance if user reaches at a certain point on the floor, game logic checks coordinates of this point and activates a zombie actor to attack user.

7.2.4. Constraints

Since game logic is responsible for the smooth run of the scenario it is important for us to make it such a way that every actor is being called in the correct place. It is also important for game logic being stable too.
Game logic should also respond as quickly as possible to UI's calls.

7.2.5. Compositions

Game logic is not composed of any other classes. It is a logical unit by itself.

7.2.6. Uses/Interactions

As seen in the figure 15, game logic interacts with UI, Actors and Actor actions.
Firstly receives results of input from UI, according to state of the game it triggers actors, and also according to result of the actor actions game state may be changed.

7.2.7. Resources

Game logic does not use any external resources. It is just like UI consists of scripts that we will write in order to check current state of the game.

7.2.8. Processing

Game logic knows the place of the every object and when and how they should react and interact. For instance when start game command is received from the UI, a script called GameInstantiator will be called. The GameInstantiator holds references to the textures, items on the screen, the Player GameObject that manages player-configuration and -settings, the InputControl GameObject that is used to process user input, and a reference to the PlayArea GameObject that defines the playable area of the texture, spawn points of the zombies etc. Its screenshot is shown in figure 38.

Figure 38: Game Instantiator Script

It places all of these objects, places them inside itself and calls UI to update what is displayed on the screen. Or when shoot button is pressed command is received from UI it calls Zombie actor to activate its freeze function.

7.3. Actors

7.2.1. Classification

This component can be classified as game objects.

7.2.2. Definition

Actors component consists of the units in the game that are usually a game object. Game actors should be added to every scene of the game. Here is the list of game actors in the scenes.

- Camera
- Light

- Background
- Character
- Map
- Big_brain
- Backpack
- Zombie
- Element
- Erlenmeyer
- Clue
- Stone
- Door
- Fence

7.3.3. Responsibilities

7.3.3.1. Camera

Camera is a device through which the player views the world.

7.3.3.2. Light

We will use lights to illuminate the scenes and objects to create the perfect visual mood.

7.3.3.3. Background

Background will be the texture added cube object used to make user understand current atmosphere of the scene.

7.3.3.4. Character

Character will be the most active component during game play. It will do some actions like jumping, moving, firing etc.

7.3.3.5. Map

Map will be the object to show the gamer his process in the game level. It will also help user to move through the game levels once activated.

7.3.3.6. Big_brain

Big brain is our lab companion who was turned into a zombie accidentally. During the game play character will follow its traces to reach it.

7.3.3.7. Backpack

Backpack will be the object where user keeps his collected items like elements, stones or salt.

7.3.3.8. Zombie

Zombie will be our enemy in the game that we will fight against to.

7.3.3.9. Element

Element will be a collectible object that user can collect or combine.

7.3.3.10. Erlenmeyer

Erlenmeyer will be the game object where user can combine elements into it.

7.3.3.11. Clue

Clues in the game play will help user to move forward in the game with small tips.

7.3.3.12. Stone

Stone will be a collectible object that user can collect or combine.

7.3.3.13. Door

Door will be the object where Lewis puzzle is displayed. If it is solved correctly it will allow user to pass another level of the game

7.3.3.14. Fence

Fence will be the object that needs to be melted in order to get into the tunnel.

7.3.4. Constraints

This component has many constraints since it has many game objects. These objects should be updated and their interactions should be made properly.

7.3.5. Compositions

As we explained before, this component includes units below;

- Camera
- Light
- Background
- Character
- Map
- Big_brain
- Backpack
- Zombie
- Element
- Erlenmeyer
- Clue
- Stone
- Door
- Fence

7.3.6. Uses/Interactions

Interactions of this component showed in figure 15.

7.3.7. Resources

There is no need for the use of external resources. It will consist of unity's game objects and our scripting.

7.3.8. Processing

7.3.8.1. Camera

There will be one main camera in the game that will follow character throughout the game play. When character jumps or crouches it will not move upwards or downwards. It will focus on the character and have it in the center during game with move_with_character function.

User starts level
Awake function is called
While level continues

Update function is called

If character moves

move_with_character function is called with the corresponding character coordinates

If level ends

Destroy camera object

7.3.8.2. Light

In a level there will be various number of lights to illuminate all objects in the game. Most of them will be constant lights. However the light of the character will be dynamic and move with character.

7.3.8.2.1. User light object

User starts level

Awake function is called

While level continues

Update function is called

If character moves

Light's move_with_character function is called with appropriate coordinates

If level ends

Destroy light object

7.3.8.2.2. Other light objects

User starts level

Awake function is called

If level ends

Destroy light object

7.3.8.3. Background

Game will start with the laboratory background. In here no action will take place, only story of the game will be told. When main game play starts street background will become active. After solving liquid puzzle tunnel background will

be displayed with tips on the walls. At the end of the tunnel a door background will become active to solve a puzzle.

Story begins
    Laboratory background awakened
Game play begins
    Street background is awakened
    While game play is active
        Street background is updated
    User faces with a puzzle
        Street background is destroyed
    If user solves liquid puzzle
        Tunnel background is awakened
    While game play is active
        Tunnel background is updated
    User faces with a puzzle
        Tunnel background is destroyed
        Door background is awakened
    While game play is active
        Door background is updated
Game play ends
    Backgrounds are destroyed

7.3.8.4. Character

When a level starts character will start to walk in the street or jump, when it faces zombies it will fire to shoot them. What is more, before the game play, character's gender will be chosen.

User chooses character from menu
    Character is awakened
    Character's gender is set
Game play starts

while game play is active

if user clicks move forward button

go_forward function is called

if user clicks move backward button

go_backward function is called

if user clicks jump button

jump function is called

if user clicks fire button

fire function is called

if user clicks crouch button

crouch function is called

if user is shot

health is decreased

character's coordinate is updated

character is updated

Game play ends

Character class is deactivated

7.3.8.5. Map

In the storyline when character returns to laboratory she will find a map shining on the floor. When clicks or touches on it will be activated. From that on the map will provide user a walk-through of the game. It will have tags like "pass the acid liquid, open the Lewis door".

In the story line

When user reaches laboratory map becomes shiny

When user clicks on, map it is activated.

In game play

Map icon is always displayed in the screen

If user clicks on map icon

Map screen is opened

If user clicks any of the points in the map

User is forwarded to that level

Map is updated

If user moves to another level

level_id is updated

End of game play

Map is deactivated

7.3.8.6. Big_brain

In the story line when character returns to the lab, he will understand that the big brain has escaped from the lab and turned into a zombie. Character's aim is to find big brain and turn him into human form. When character reaches him, character will prepare a compound to turn in to human.

In the story line

Big brain turns into zombie and escapes form the lab

In game play

When character reaches big brain it becomes active

Big brain's coordinates are updated

If user reaches big brain

If correct compound is prepared

Big brain's turn_into_human is called

End of game play

Big brain is deactivated

7.3.8.7. Backpack

Backpack is activated during game play, when user collects elements, stones etc. they will become reachable from the backpack by clicking on the backpack icon.

In game play backpack becomes actives

When user clicks backpack item

Elements, stones becomes visible if any

Update backpack

End of game play

Backpack is deactivated

7.3.8.8. Zombie

Zombies are normal people that were turned into zombie by big brain. When player completes the game user will be able to turn them into human.

However, during game play user will face them as zombies and will freeze them with nitrogen gun. When they are frozen they will drop elements or stones correspondingly.

In game play backpack becomes actives

When user reaches corresponding part of the level

Zombie is awakened

If user freezes zombie

Zombie drops item (stone or element)

Else

Zombie hurts user

Zombie's hold_user function is called

Zombie is updated

End of game play

Zombie's turn_into_human function is called

Zombie is deactivated

7.3.8.9. Element

Element will be dropped by zombies, and user will collect them by dragging to the backpack. When required they can be combined with same kind.

In game play

When user kills a zombie

Element is awakened& becomes shiny

If element is shiny

User drags element to backpack by move_into_backpack

is_in_backpack becomes true

If user opens backpack

If user combines element by dragging it to another element

combine function is called

End of game play

Element is deactivated

7.3.8.10. Erlenmeyer

Erlenmeyer will be used to combine elements in the required parts of the game play such as acidic liquid puzzle. It will hold the pH of objects whose are combined within it. If a wrong combination is prepared user will be able to empty it and prepare a new compound in it.

In game play

Erlenmeyer is activated

Erlenmeyer is set to be empty

If user opens backpack

If user drags elements to Erlenmeyer

Erlenmeyer's elements are changed

pH is changed

If compound is wrong

Erlenmeyer is emptied

Erlenmeyer's elements are set to null

Erlenmeyer is updated

End of game play

Erlenmeyer is deactivated

7.3.8.11. Clue

When user reaches previously determined parts of the game play, clue object will appear there and will be collectible. As they are collected, their tips will appear in the screen.

In game play

User reaches corresponding level part

Clue is activated

If user clicks on it

Clue's information string is displayed on the screen

End of game play

All clues are deactivated

7.3.8.12. Stone

Zombies will drop stone, and user will collect them by dragging to the backpack. In addition, they will be placed into the holes on the door. When user makes a mistake the earthquake make them drop into the floor. Therefore, player should drag them to the backpack and use them again.

In game play

When user kills a zombie or earthquake happens

Stone is awakened& becomes shiny

If stone is shiny

User drags stone to backpack by move_into_backpack

is_in_backpack becomes true

If user opens backpack

If user drags stone to the hole of the door

use_in_lewis function is called

End of game play

Stone is deactivated

7.3.8.13. Door

Door will be active when user reaches it. It will have holes on itself to be filled with stones in order to display correct Lewis formula. If stones are put correctly it will be opened, otherwise there will be an earthquake and all stones drop, holes will be empty again.

In game play

When user reaches corresponding part

Door's elements are set

Door's is open is set to be false

Door is activated

If user drags a stone to any hole

Corresponding hole is filled

If all holes are filled

If correct_holes are true

Door's is open set to be true

Open is called to make user to move onto another level

Else

earthquake is called

elements are changed

holes are emptied

Door is updated

End of game play

Door is deactivated

7.3.8.14. Fence

Fence will be active when user reaches it. It will block the entrance and needs to be melted down.

In game play

When user reaches corresponding part

Fence is activated

If user throws potion (not acid)

Nothing happens

If user throws acid

Fence melts down

Fence is updated

End of game play

Fence is deactivated

7.4. Actor Actions

7.4.1. Classification

Some of these actions are provided by Unity, while most of them will be developed.

7.4.2. Definition

This component provides various actions for the actors such as movement and changing the texture.

7.4.3. Responsibilities

Actor actions component is responsible for the actions like movements and graphical changes. Triggered by UI and game logic components, it activates the corresponding functions. Within these functions, this component can change the actors' attributes.

For example, after user presses the jump button, UI triggers the actor actions, which causes it to jump the character. To make the character jump, it changes the coordinates of the character.

7.4.4. Constraints

There are not any constraints for this component.

7.4.5. Compositions

The actor actions component consists of many scripts having different actions for different actors each.

7.4.6. Uses/Interactions

As seen in the figure 15, the actor actions are triggered by UI and the game logic. After processing the input, they change the responding actors' attributes.

7.4.7. Resources

For this component, some of the default Unity scripts will be used, but some scripts developed for this project will also be used.

7.4.8. Processing

Scripts, which are the parts of this component, are bound to the game actors. They listen to the triggers for possible changes. If any listened trigger sends a signal, the script starts to run. After making the necessary process corresponding to the specific trigger, it changes the actor's attributes.

7.5. Renderer

7.5.1. Classification

This component is an inner part of the Unity's game engine.

7.5.2. Definition

The renderer renders the given game scene.

7.5.3. Responsibilities

This component is responsible for drawing the game world, with all of its actors, to the screen.

7.5.4. Constraints

Since this is a part of the Unity, there is no important constraint for the developers.

7.5.5. Compositions

No internal architectural details are known for this Unity component.

7.5.6. Uses/Interactions

As seen in the figure 15 under the chapter 5.1, this component reads the position and the texture information from each actor in the level.

7.5.7. Resources

There are no resources for the renderer

7.5.8. Processing

For each frame, this component reads the position and the texture information for the actors. Then, it draws them to the screen.

8. Libraries and Tools

8.1. Libraries

Since we will use the Unity environment, we won't need any external libraries. But we will still use Unity's packages that we will explain below.

8.1.1. Character Controller

Character controller package includes the generic controller scripts that can be used for third-person or first-person games.

8.1.2. Light Flares

This package contains necessary scripts to create light flares.

8.1.3. Particles

This package contains necessary scripts to create particle graphics.

8.1.4. Physics Materials

This package has the scripts to adjust friction and bouncing effects of colliding objects.

8.1.5. Projectors

Projectors are used to project materials onto objects. This package helps to use the projectors.

8.1.6. Scripts

The Scripts package contains basic scripts such as camera scripts, general scripts and utility scripts.

8.1.7. Standard Assets (Mobile)

This library includes components specific to mobile environments.

8.1.8. Toon Shading

This library includes the shader scripts to create realistic shading effects.

8.1.9. Water

This package includes the water effect scripts.

8.1.10. Touch

This package helps us to create tablet controls easily by identifying finger's position or movements on the screen.

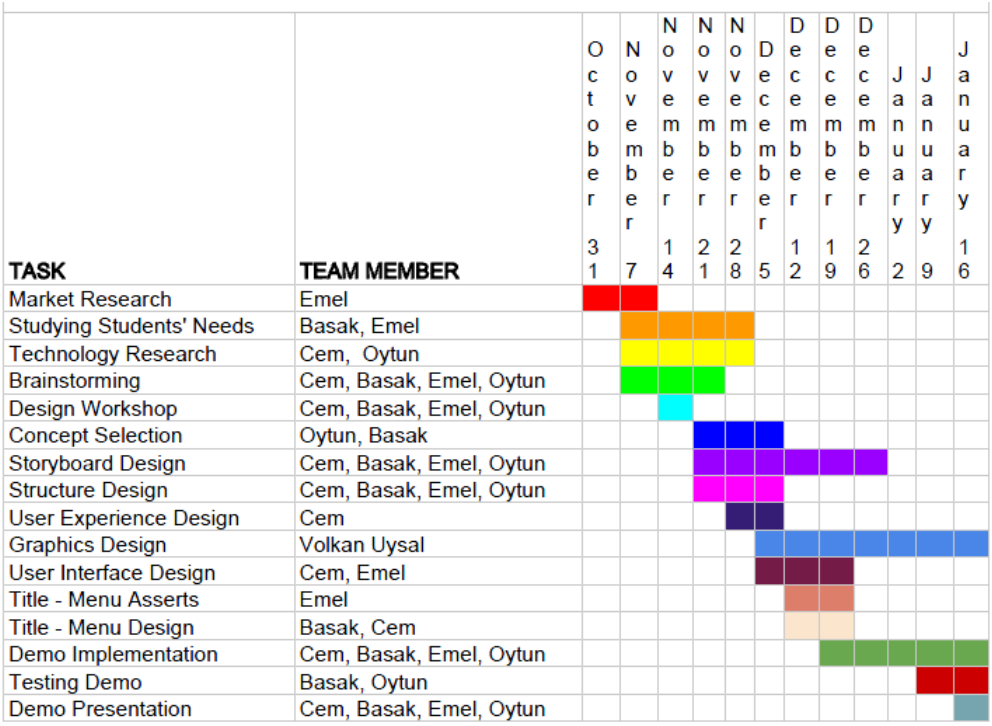
8.2. Tools

8.2.1 Unity

Unity is a cross-platform integrated game development tool. Unity differentiates itself with its integrated engine and various deployment platforms. This integrated engine makes it possible to easily test the game where the extensive variety of the deployment platforms enables us to deploy the game to the different platforms without thinking about platform differences.

9. Time Planning

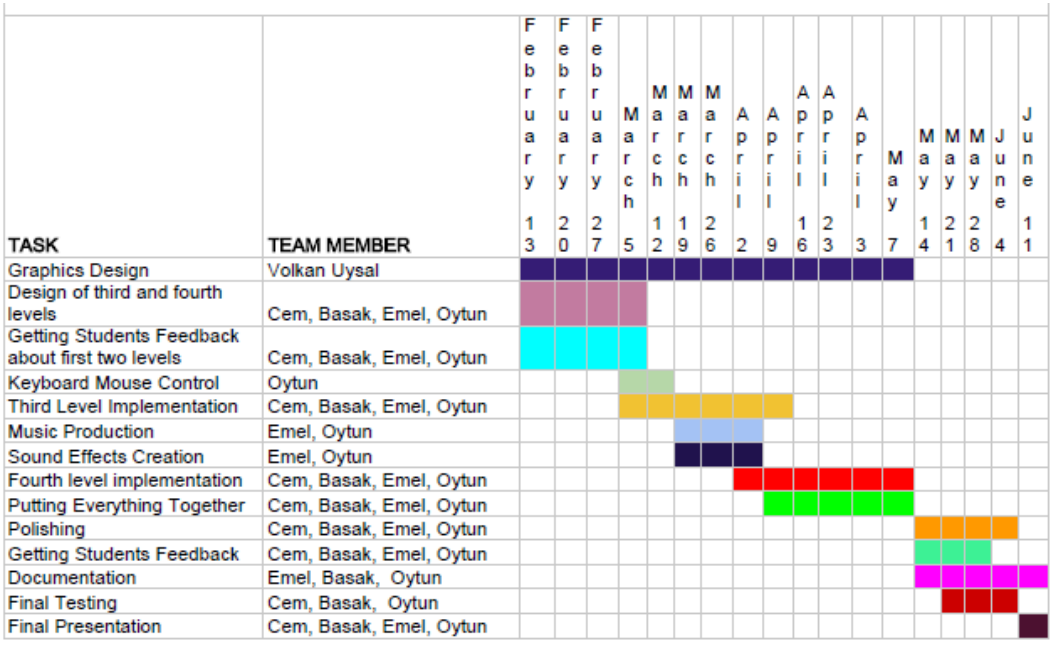
9.1. First Term Gantt Chart



TASK	TEAM MEMBER	October 31	November 7	November 14	November 21	November 28	December 5	December 12	December 19	December 26	January 2	January 9	January 16
Market Research	Emel												
Studying Students' Needs	Basak, Emel												
Technology Research	Cem, Oytun												
Brainstorming	Cem, Basak, Emel, Oytun												
Design Workshop	Cem, Basak, Emel, Oytun												
Concept Selection	Oytun, Basak												
Storyboard Design	Cem, Basak, Emel, Oytun												
Structure Design	Cem, Basak, Emel, Oytun												
User Experience Design	Cem												
Graphics Design	Volkan Uysal												
User Interface Design	Cem, Emel												
Title - Menu Asserts	Emel												
Title - Menu Design	Basak, Cem												
Demo Implementation	Cem, Basak, Emel, Oytun												
Testing Demo	Basak, Oytun												
Demo Presentation	Cem, Basak, Emel, Oytun												

Figure 39: First Term Gantt Chart

9.2. Second Term Gantt Chart



TASK	TEAM MEMBER	February 13	February 20	February 27	March 5	March 12	March 19	March 26	April 2	April 9	April 16	April 23	April 3	May 7	May 14	May 21	May 28	June 4	June 11
Graphics Design	Volkan Uysal																		
Design of third and fourth levels	Cem, Basak, Emel, Oytun																		
Getting Students Feedback about first two levels	Cem, Basak, Emel, Oytun																		
Keyboard Mouse Control	Oytun																		
Third Level Implementation	Cem, Basak, Emel, Oytun																		
Music Production	Emel, Oytun																		
Sound Effects Creation	Emel, Oytun																		
Fourth level implementation	Cem, Basak, Emel, Oytun																		
Putting Everything Together	Cem, Basak, Emel, Oytun																		
Polishing	Cem, Basak, Emel, Oytun																		
Getting Students Feedback	Cem, Basak, Emel, Oytun																		
Documentation	Emel, Basak, Oytun																		
Final Testing	Cem, Basak, Oytun																		
Final Presentation	Cem, Basak, Emel, Oytun																		

Figure 40: Second Term Gantt Chart

10. Conclusion

In this detailed design report, we have explained our game design in detail. First of all, this report consists of representation of the system, assumptions and dependencies. Then, we defined and explained data structures and architectural components. we have given information about user interface and the libraries. Finally, Gantt chart notation is given at the end of the document. This report is going to be very helpful in the future for understanding and implementing design patterns. To sum up, this detailed design report will be the guideline of our project this year.