Bilkent University

Department of Computer Engineering

CS 319 Term Project

Icy Tower

Analysis Report

Section 1

Group 1C

Project Group Members:

Cansu Yıldırım

Berke Soysal

Ozan Kerem Devamlı

Supervisor: Eray Tüzün

1.Introduction

We wanted to design and implement a new and extended version of the well-known action and platform game, *Icy Tower*. It was created by the Swedish game developer firm *Free Lunch Design* . The main goal of the game is climbing the Icy Tower by jumping through bars inside them [1]. What makes this game fun is the increasing level of difficulty as the game goes on, and the bonus points player get by jumping multiple rows at sequential jumps(Combo Jump as we call it). As all of us have been played this game when we were children, we decided to implement this game with passion. Also the structure of the game is seen compatible with the content of our course such that the game can be implemented with a manner of object oriented software engineering.

Our game will have features such that:

- Different levels of difficulty
- Different types of platforms
- Sound and music options
- Character options
- Control button options
- Continuously incrementing game speed
- List of high scores
- In-game pause option

Our game will have features that different from the original version game such as:

- Bonus and Traps that makes the game more fun

We will implement the game in Java environment with using the features of the JavaFX API. We choose JavaFX because it provides easy and smooth graphics and UI design properly while it does not make the integration of the objects difficult.

2. Overview

In this section, brief information about the game play, game objects and some features of the game will be introduced.

2.1 Gameplay

Icy-Tower is a single player platform game. The main objective of the player is to jump through the bars quickly to reach a higher place of the Icy Tower. When player makes the combo jump which means player jumps to the wall and then jumps to an above bar, this enables player to go up faster and to win more points. The game has no end, so there is no "top" of the tower. The bars that player jump will change (usually will make the game harder) as the game continues, there will be different advantages and disadvantages of each bar. There will be three different levels that player can choose at the beginning, easy, medium and hard. The game difficulty will be increased with speed of the game, frequency and length of the upcoming bars. Also, the game difficulty will be increased during the game time; as the game continues, the bars will drop down faster, the frequency of generating bars, and the size of the bars will be decreased.

2.2 Main Character

There will be different type of characters that player can choose from the character settings. The difference between the characters is their appearance. Character can jump, or move left/right. Character can make combo jump and get more points.

2.3 Map

Map will be generated randomly at each time the player goes above from the game screen. Map consists of bars, bonus and traps. At the beginning of the game the map will create the bars frequently, but as the game goes on, the map will move to upward faster and the frequency of generating bars will drop.

2.4 Bars

Bars are the objects that player can jump on. There will be different kind of bars (icy, wooden, sticky and hardly visible) and each of these has different characteristics. Icy bar inclines the characters to slip; wooden bar is shorter to make the player's getting acceleration harder; sticky bar makes the moves difficult and hardly visible bar cannot be seen easily so that in the further time of the game, it will be difficult to pursue the bars.

2.5 Additional Items

There will be different types of bonuses that some of them has advantages and some of them has disadvantages. These bonuses are collectible items. These are better explained in section 3.7.

2.6 Score

Game score will be increased as the player will go upward. If the player will jump to the bar with sliding on the wall (combo jump), s/he gets extra points. If a player gets the balloon bonus and moved up by the balloon, he will still get the points for going up. If a player gets a coin bonus, it will increase the score of the player.

2.7 Settings

Player can change the volume of the game sounds and music, change the game character, or can change the control buttons from the settings.

2.8 Difficulty

There will be three types of difficulty level in the game, these are: easy, medium and hard. The difficulty of the game is increased by decrementing the frequency of bars, speed up the game screen speed, and shrinking the size of the bars. Player can select game difficulty each time before he starts the game.

3. Functional Requirements

Functional requirements describe the interactions between system and its environment [2]. This section is for introducing functional requirements of the game Icy Tower.

3.1 Play Game

After entering the play game screen, players will be asked about the difficulty level of the game. After choosing the level, the game will start. In each level, player will try to jump on the bars in order to reach as above as possible while getting more points by making combo jumps as mentioned above. The speed of game screen increases by time passes. So the player must accelerate too. Player must be fast enough to not to fall behind the game screen. However, if the player is too fast and intends to exceed the game frame, game screen catches up with the player. In addition, the types of bars will also be changed. With the progression of the player, bars' properties will make the game harder for players. For example, the bars will be shortened. It will be harder to move on these shorter bars. From the level easy to hard, the speed of the game screen changes. Therefore, in the hard level, player experiences faster screen movement even at the beginning of the game. The player can control the character with the left-right and space button (or with A, D, W by changing the button settings). Moreover, some bonuses have place in the game. Balloon bonus helps the player to move up easily. When the player see the balloon, it is better to get it. Also, there are bombs as traps. If the character encounters with a bomb, it is better to escape to not to be killed.

3.2 Credits

Credits can be accessed by a button on the main menu. The player can see the names of the developers by this screen.

3.3 Settings

Settings can be accessed by a button on the main menu. Player can adjust the volume of the music; change the music and the control buttons to play the game from settings screen. Also player can see the different characters and select his/her favourite to play with it from this setting page. Sound Settings can also be accessible from the pause menu.

3.4 How to Play

How to Play screen informs the players about:

- The logic of the game
- Control buttons that will be used
- Settings options
- How to record your score
- Bonuses

By reading this, users can play the game without having any trouble.

3.5 High Scores

High Scores screen displays the top ten scores with names of the players in descending order. Therefore, the highest score will be at the top of the list. Every time a player makes a score which is suitable to be in the top ten, the new score will be recorded. System asks the player to type a name. With this name, the score takes its place on the list. So, the player can see the current list whenever he/she wants.

3.6 Exit

Exit screen works to quit the game. After entering exit screen, verification question shows up. If the player verifies the exit command, the system closes.

3.7 Additional requirements

We decided to make more fun by adding some more collectible items to the Game.

Balloon: Balloon collectible will enable players to fly for a while in the icy tower thereby to up go up without any trouble.

Time-bonus: Time-bonus collectible will make the speed of the game screen slower for a while. This will be very advantageous later in the game, as it will become too fast.

Bar-bonus: Bar-bonus collectible will make the bars longer and easily standable for the players for a while.

Coin: Players can also increase their scores by getting coin bonus.

Timekiller-bonus: Disadvantageous bonus is a single type of timekiller-bonus. This will make the speed of the game faster and give players a hard time.

4. Non-functional Requirements

Non-functional Requirements describe aspects of the system that are not directly related to the functional behaviour of the system [2]. This section will introduce the non-functional requirements of the game Icy Tower.

4.1. Usability

Icy Tower will be easily understandable and usable game for all people which is older than age 5. All menu pages in the game and the game itself will be simple and easy to understand. Even without take a look on “How to Play” the game logic will be understood by people.

4.2. Performance

The game performance is one of our main concerns about the game. The game is expected to run at 30 frame per second, without any flickering. The game will run without freezing/fps dropping with any computer which at least have Windows XP, Intel Pentium III as CPU, and any graphics card which have 32MB memory.

4.3. Extendibility

The game will be implemented in a way that it will be extendable. Extra features and extensions will be easy to implement. These implementations will be done without making dramatic changes on the core code of the game. We will obey to the Object Oriented Approach to maintain the extendable design.

4.4. Robustness

We are promising that no major bugs will be in the game unexpectedly closes, or user is being stuck on a place or a game freeze. We care about robustness for a better gaming experience to users.

4.5 Additional Requirements

Portability

The game is currently being developed for Desktop. However, since our game will be implemented in Java, Icy Tower will be a portable game and can be played in different devices, it can easily be converted to be executable on Android/IOS Platforms.

5. System Models

5.1 Use Case Model

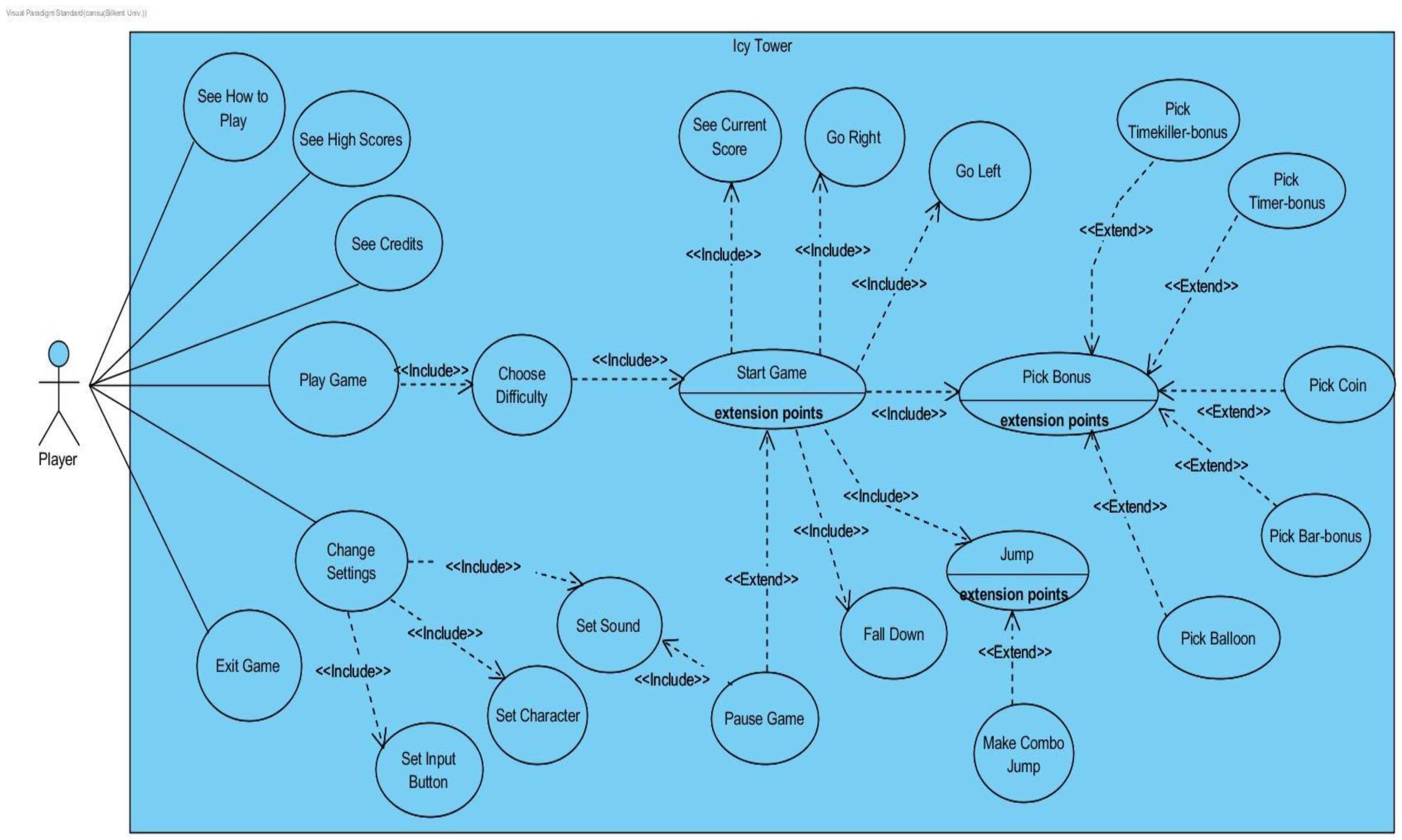


Figure 5.1.1 Use Case Model

Use Case: Play Game

Actor: Player

Stakeholders and Interests:

- Player wants to play the game
- System asks the difficulty level to the player and starts the game

Pre-condition: Player must be in the menu

Post-condition: System displays the difficulty levels

Entry-condition: Player selects the “Play Game” button from the menu

Exit-conditions:

- Player dies
- Player is hit by a bomb

Event Flow:

1. Player selects the “Play Game” button from the menu
2. Player selects the medium level of the game
3. Player goes left and jump to the bar
4. Player continues to jump to the bars
5. Player checks the current score on the screen
6. Player pick the balloon bonus
7. Player makes combo jump
8. Player falls down to the space
9. The game is over

Alternative Event Flow:

1- Player’s speed is less than the speed of the game screen

- Player falls behind the screen move

- Player becomes out of the game screen
- Player dies

2- Player wants to pause the game

- Player presses the button “Pause” on the game screen
- Game pauses

Use Case: Pick Bonus

Actor: Player

Stakeholders and Interests:

- Player pick one of the bonus items
- The game become easier or harder for a while
- Score increases if its coin bonus

Pre-condition: Player must be playing game

Post-condition: Score increases if its advantageous bonus

Entry-condition: Player picks the bonus item in the game

Exit-condition: Times up for bonus feature

Event Flow:

1. Player picks the timer-bonus
2. The speed of the game screen will be slower for a while
3. Times up for timer-bonus
4. Game will continue with the old screen speed

Alternative Event Flow:

1. Player picks the bar-bonus

- Player picks the bar-bonus

- The length of the bars becomes larger and easily standable
- Times up for bar-bonus
- Game will continue with the old bars

2. Player picks the coin

- Player picks the coin
- Score increases

3. Player picks the timekiller-bonus

- Player picks the timekiller-bonus
- The speed of the game screen will be faster for a while
- Times up for timekiller-bonus
- Game will continue with the old screen speed

Use Case: See How to Play

Actor: Player

Stakeholders and Interests:

- Player wants to learn how to play the icy tower game
- System shows the how to play screen

Pre-condition: Player must be in the menu

Post-condition: -

Entry-condition: Player selects the "How to Play" button from the menu

Exit-condition: Player selects the "Back to Menu" from the how to play screen

Event Flow:

1- Player wants to learn how to play

2- Player selects the "How to Play" button from the menu

3- System displays the how to play screen

Alternative Event Flow:

1- Player wants to go back to the menu

2- Player selects “Back to Menu” button from the how to play screen

3- System displays the menu

Use Case: See Credits

Actor: Player

Stakeholders and Interests:

- Player wants to see the credits of the icy tower game
- System shows the credits screen

Pre-condition: Player must be in the menu

Post-condition: -

Entry-condition: Player selects the “Credits” button from the menu

Exit-condition: Player selects the “Back to Menu” from the credits screen

Event Flow:

1- Player wants to see the credits of the game

2- Player selects the “Credits” button from the menu

3- System displays the credits screen

Alternative Event Flow:

1- Player wants to go back to the menu

2- Player selects “Back to Menu” button from the credits screen

3- System displays the menu

Use Case: See High Scores

Actor: Player

Stakeholders and Interests:

- Player wants to see the high scores of the game
- System shows the high scores screen

Pre-condition: Player must be in the menu

Post-condition: -

Entry-condition: Player selects the "High Scores" button from the menu

Exit-condition: Player selects the "Back to Menu" from the high scores screen

Event Flow:

1- Player wants to see the scores of the game

2- Player selects the "High Scores" button from the menu

3- System displays high scores screen

Alternative Event Flow:

1- Player wants to go back to the menu

2- Player selects "Back to Menu" button from the high scores screen

3- System displays the menu

Use Case: Change Settings

Actor: Player

Stakeholders and Interests:

- Player wants to adjust the game settings
- System asks which type of the setting the user wants to adjust

- System changes the chosen setting

Pre-conditions:

- Player must be in the menu
- Player must be in the pause menu for “Sound Setting”

Post-condition: Settings are updated

Entry-conditions:

- Player selects the “Settings” buttons from the main menu
- Player selects the “Sound Setting” button from the pause menu

Exit-condition:

- Player selects the “Back to Menu” button from the settings screen
- Player selects the “Resume Game” button from the pause menu

Event Flow:

1- Player selects the “Settings” button from the main menu

2- Player selects the “Sound Settings” button

- Player selects the music
- Player adjusts the volume
- Player repeats the steps 5 and 6

3- Player selects the “Character Settings” button

- Player selects the character type
- Player repeats the steps 5 and 6

4- Player selects the “Button Settings” button

- Player decides the buttons that will be used (right, left, space or A, D, W on the keyboard)
- Player repeats the steps 5 and 6

5- Player selects “Back to Menu” button

6- System displays menu

Alternative Event Flow:

1- Player selects the “Sound Setting” button from the pause menu

2- Player changes the music or volume

3- Player selects the “Resume Game” button

4- Game resumes

Use Case: Pause Game

Actor: Player

Stakeholders and Interests:

- Player wants to pause the game
- System displays the pause menu

Pre-conditions:

- Player must be playing the game

Post-condition: -

Entry-conditions:

- Player selects the “Pause” button from the game screen

Exit-condition: Player selects the “Resume Game” button from the pause menu

Event Flow:

1- Player selects the “Pause” button from the game screen

2- Game pauses

3- System displays the pause menu

Alternative Event Flow:

1- Player selects the “Resume Game” button from the pause menu

2- Game resumes

Use Case: Exit Game

Actor: Player

Stakeholders and Interests:

- Player wants to exit
- System closes the game

Pre-conditions: Player must be in the menu

Post-condition: System closes

Entry-condition:

- Player selects the “Exit” and “Yes” buttons respectively

Exit-condition: Player selects “No” button from the exit menu

Event Flows:

1- Player selects the “Exit” button from the menu

2- Player selects the “Yes” button from the menu

3- The game closes

Alternative Event Flow:

1- Player selects the “Exit” button from the menu

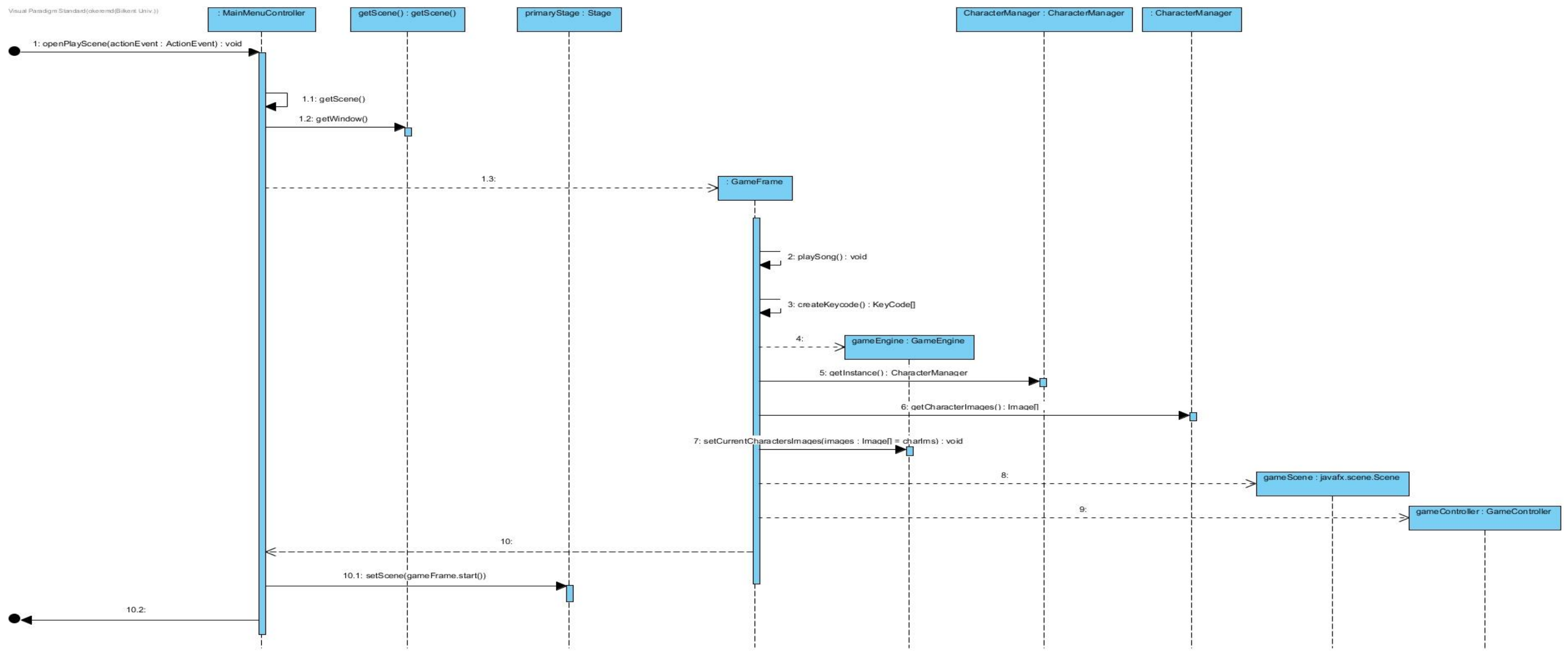
2- Player selects the “No” button from the exit menu

3- System displays the menu

5.2 Dynamic Models

Dynamic models focus on the behaviour of the system. It describes the components of the system which have dynamic behaviour using Sequence diagrams and Activity diagram [2].

5.2.1 Sequence Diagrams



Visual Paradigm Standard(okeremd(Bilkent Univ.))
: MainMenuController
getScene() : getScene()
primaryStage : Stage
CharacterManager : CharacterManager
: CharacterManager
1: openPlayScene(actionEvent : ActionEvent) : void
1.1: getScene()
1.2: getWindow()
1.3:
: GameFrame
2: playSong() : void
3: createKeycode() : KeyCode[]
4:
gameEngine : GameEngine
5: getInstance() : CharacterManager
6: getCharacterImages() : Image[]
7: setCurrentCharactersImages(images : Image[] = charIms) : void
8:
gameScene : javafx.scene.Scene
9:
gameController : GameController
10:
10.1: setScene(gameFrame.start())
10.2:

Figure 5.2.1.1 Initializing game from menu

After user selects play game from main menu, openPlayScene method has invoked. Main menu initializes the first scene for GameFrame then invokes GameFrame. After GameFrame is constructed, the music starts to play and Keys of the game are taken. GameFrame initializes GameEngine, then takes chosen character's images from CharacterManager instance then passes the current characters images to GameEngine. GameFrame builds the scene with proper game related medias and initializes ButtonManager for user input and then starts to animate until game finishes.

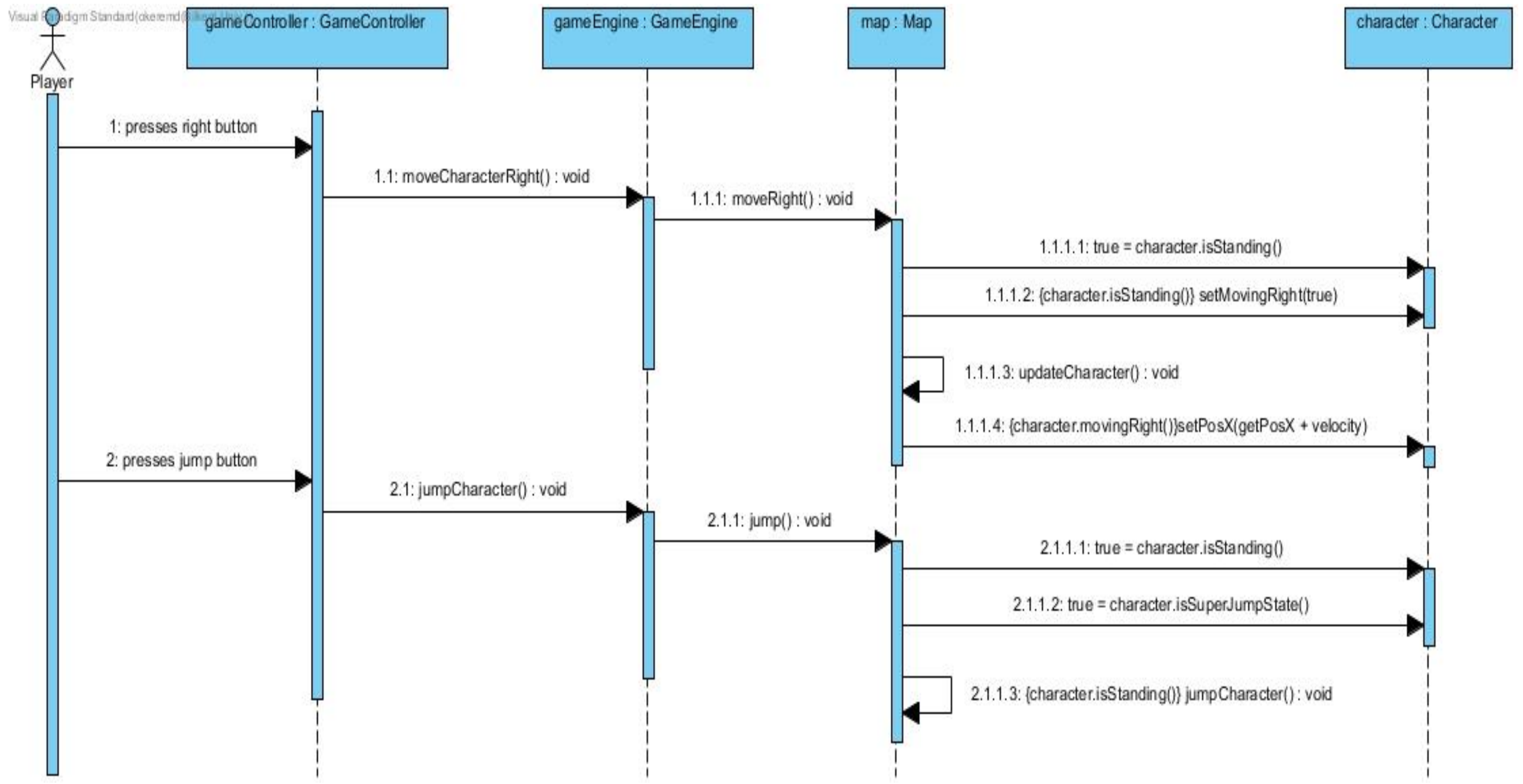

Visual Paradigm Standard(okeremd(Bilkent Univ))
Player
gameController : GameController
gameEngine : GameEngine
map : Map
character : Character
1: presses right button
1.1: moveCharacterRight() : void
1.1.1: moveRight() : void
1.1.1.1: true = character.isStanding()
1.1.1.2: {character.isStanding()} setMovingRight(true)
1.1.1.3: updateCharacter() : void
1.1.1.4: {character.movingRight()}setPosX(getPosX + velocity)
2: presses jump button
2.1: jumpCharacter() : void
2.1.1: jump() : void
2.1.1.1: true = character.isStanding()
2.1.1.2: true = character.isSuperJumpState()
2.1.1.3: {character.isStanding()} jumpCharacter() : void

Figure 5.2.1.2 Accomplishing a combo jump

An in game scenario that user controlles character to make a combo jump. First user needs to move the character in order to make a combo jump since standing without movement would not result in a combo jump. After user presses right movement button, GameController reads it and signals to GameEngine. GameEngine passes the signal to Map and Map checks if the character is in a state where he can move right, then set the character to moving right and update his position. When character moves right, user presses the jump button and GameController passes to GameEngine. Then GameEngine passes it to the Map and Map checks if the Character is on ground. After that, Map asks the character if he is in a combo jump state. Since character was moving, it returns true and Map calls the comboJump method of the character.

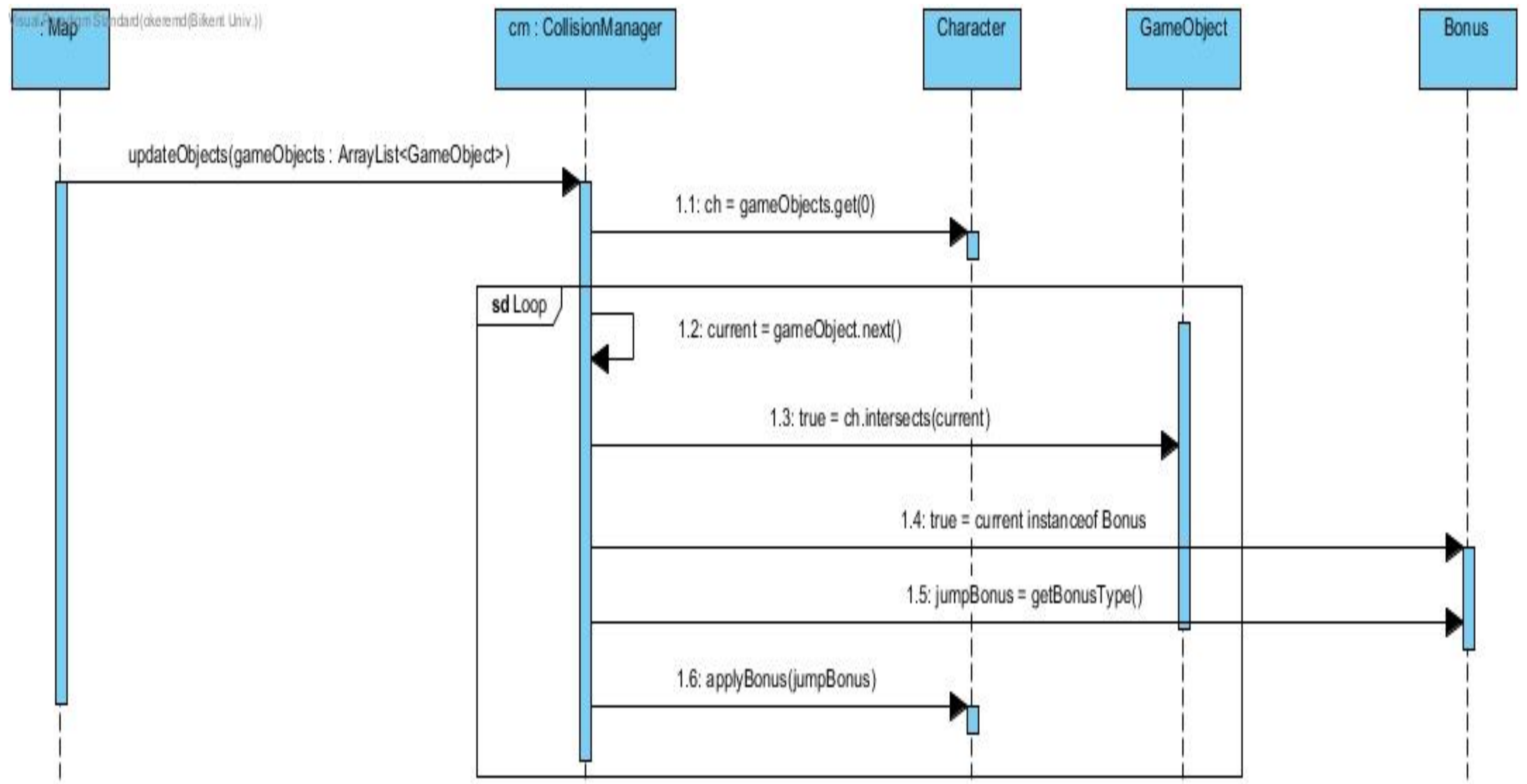

Visual Paradigm Standard(okeremd(Bilkent Univ.))
: Map
cm : CollisionManager
Character
GameObject
Bonus
updateObjects(gameObjects : ArrayList<GameObject>)
1.1: ch = gameObjects.get(0)
sd Loop
1.2: current = gameObject.next()
1.3: true = ch.intersects(current)
1.4: true = current instanceof Bonus
1.5: jumpBonus = getBonusType()
1.6: applyBonus(jumpBonus)

Figure 5.2.1.3 Taking a bonus

Map checks whether the character takes a bonus or not in every game loop. Since map is already checks collision between character and the bars by Collision Manager, we added the condition for checking bonuses in Collision Manager. If the intersection between character and a game object is detected, Collision Manager looks for the type of the gameObject.If it is a bonus, then Collision Manager takes the type of the bonus and calls the applyBonus() method with that type of bonus.

5.2.2 Activity Diagram

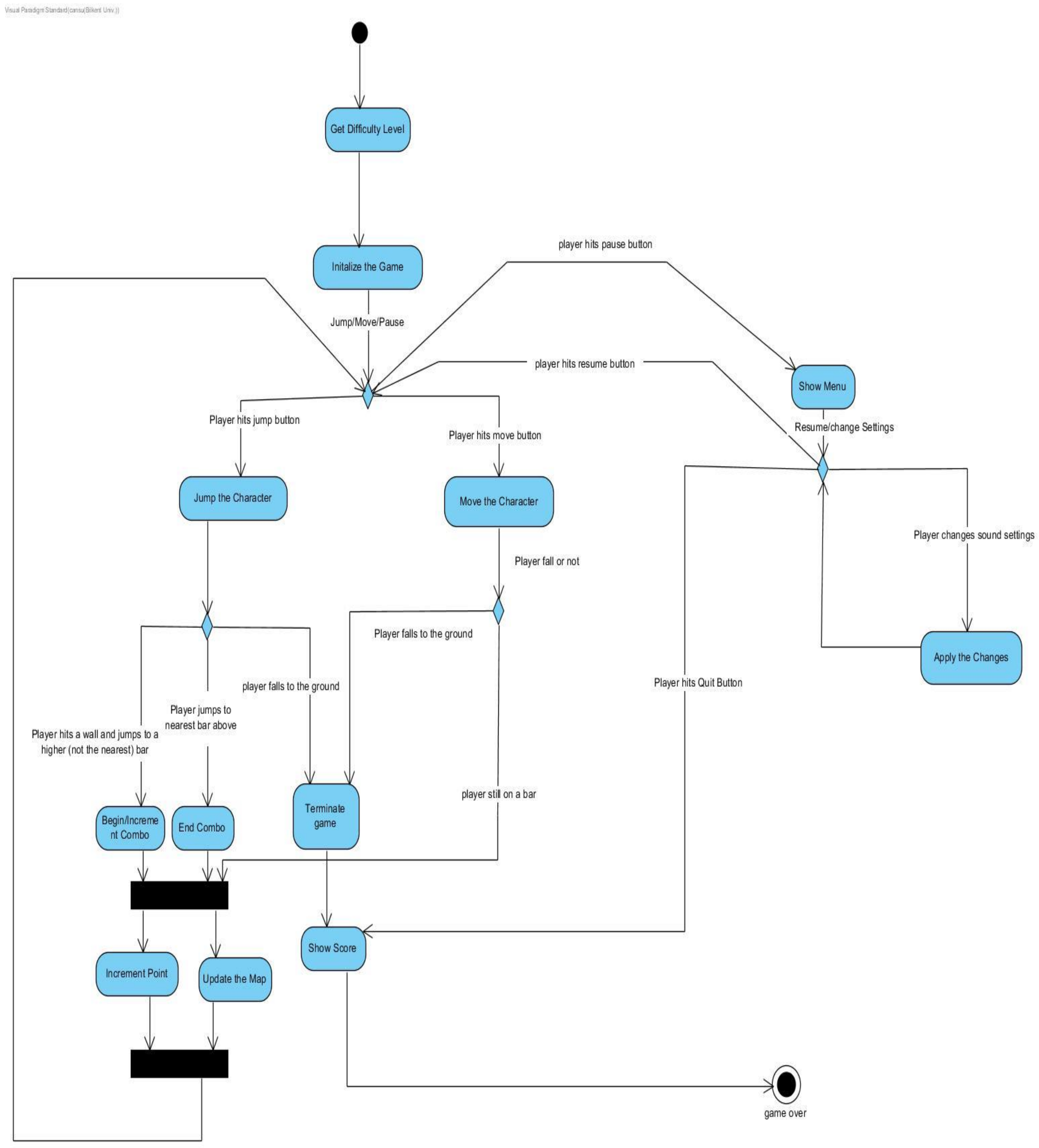
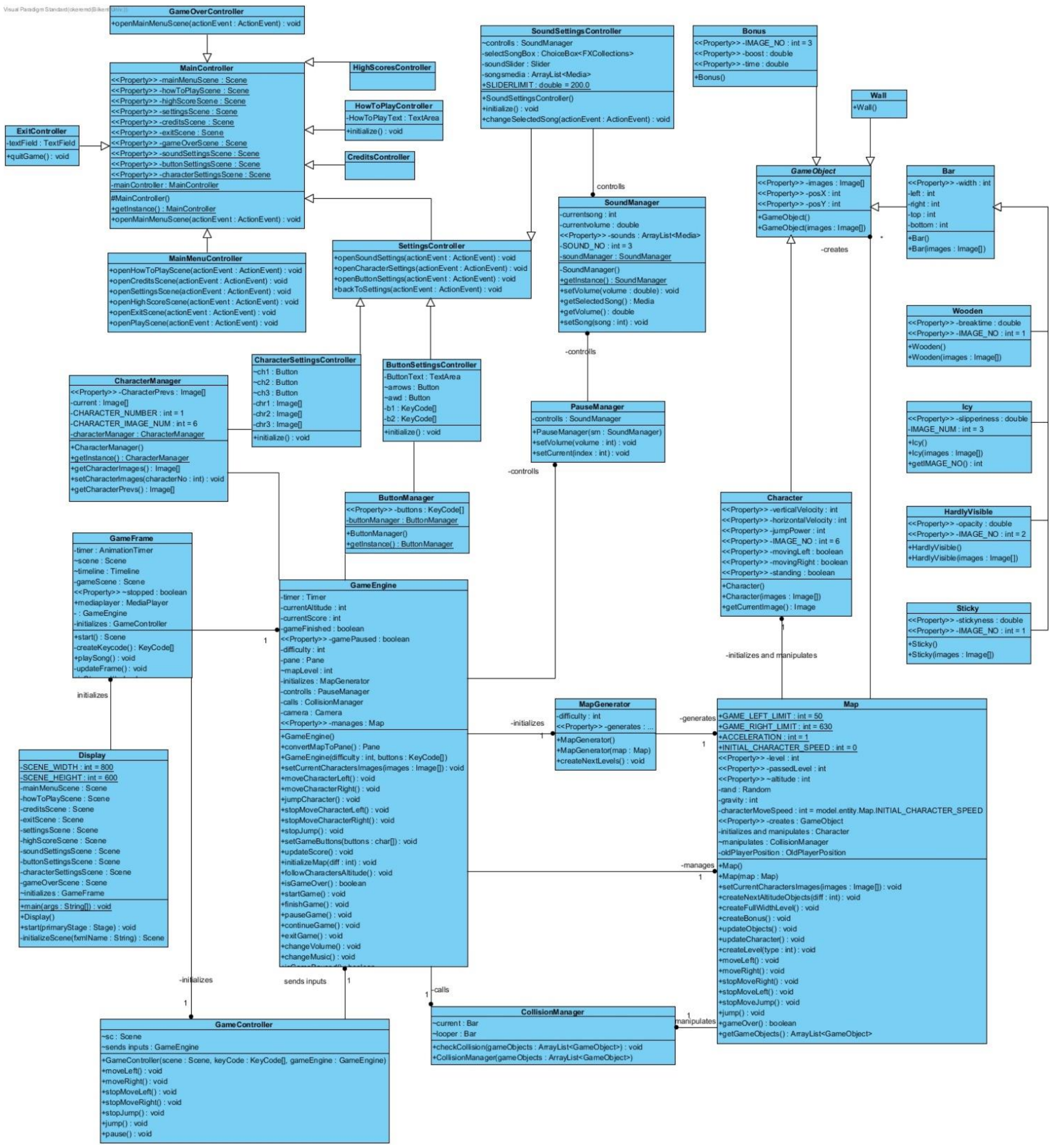


Figure 5.2.2.1 Activity Diagram

This is the activity diagram for the general flow of the game. Firstly system gets the difficulty level from the user. Then the game is initialized. User jumps, or moves during the game. If the jump button is pressed, system checks whether the player makes a combo jump (hit a wall and jump to higher and not nearest bar). If he makes a combo jump, combo jump counter keeps going, and the player keeps getting points and the map is being updated, player may end the combo, if he steps to nearest bar above. Or he can move left or right. Then the map is updated. Player may fall to space, then score is shown and game is over. The game may be paused and sound settings can be done. If the resume button is pressed in the menu, game continues.

5.3 Object and Class Model


Visual Paradigm Standard(okeremd(Bilkent Univ.))
GameOverController
+openMainMenuScene(actionEvent : ActionEvent) : void
MainController
<<Property>> -mainMenuScene : Scene
<<Property>> -howToPlayScene : Scene
<<Property>> -highScoreScene : Scene
<<Property>> -settingsScene : Scene
<<Property>> -creditsScene : Scene
<<Property>> -exitScene : Scene
<<Property>> -gameOverScene : Scene
<<Property>> -soundSettingsScene : Scene
<<Property>> -buttonSettingsScene : Scene
<<Property>> -characterSettingsScene : Scene
-mainController : MainController
#MainController()
+getInstance() : MainController
+openMainMenuScene(actionEvent : ActionEvent) : void
HighScoresController
HowToPlayController
-HowToPlayText : TextArea
+initialize() : void
CreditsController
ExitController
-textField : TextField
+quitGame() : void
MainMenuController
+openHowToPlayScene(actionEvent : ActionEvent) : void
+openCreditsScene(actionEvent : ActionEvent) : void
+openSettingsScene(actionEvent : ActionEvent) : void
+openHighScoreScene(actionEvent : ActionEvent) : void
+openExitScene(actionEvent : ActionEvent) : void
+openPlayScene(actionEvent : ActionEvent) : void
SoundSettingsController
~controlls : SoundManager
-selectSongBox : ChoiceBox<FXCollections>
-soundSlider : Slider
-songsmedia : ArrayList<Media>
+SLIDERLIMIT : double = 200.0
+SoundSettingsController()
+initialize() : void
+changeSelectedSong(actionEvent : ActionEvent) : void
controlls
SoundManager
-currentsong : int
-currentvolume : double
<<Property>> -sounds : ArrayList<Media>
-SOUND_NO : int = 3
-soundManager : SoundManager
-SoundManager()
+getInstance() : SoundManager
+setVolume(volume : double) : void
+getSelectedSong() : Media
+getVolume() : double
+setSong(song : int) : void
SettingsController
+openSoundSettings(actionEvent : ActionEvent) : void
+openCharacterSettings(actionEvent : ActionEvent) : void
+openButtonSettings(actionEvent : ActionEvent) : void
+backToSettings(actionEvent : ActionEvent) : void
CharacterSettingsController
~ch1 : Button
~ch2 : Button
~ch3 : Button
~chr1 : Image[]
~chr2 : Image[]
~chr3 : Image[]
+initialize() : void
ButtonSettingsController
-ButtonText : TextArea
~arrows : Button
~awd : Button
-b1 : KeyCode[]
-b2 : KeyCode[]
+initialize() : void
-controlls
PauseManager
-controlls : SoundManager
+PauseManager(sm : SoundManager)
+setVolume(volume : int) : void
+setCurrent(index : int) : void
-controlls
CharacterManager
<<Property>> -CharacterPrevs : Image[]
-current : Image[]
-CHARACTER_NUMBER : int = 1
-CHARACTER_IMAGE_NUM : int = 6
-characterManager : CharacterManager
+CharacterManager()
+getInstance() : CharacterManager
+getCharacterImages() : Image[]
+setCharacterImages(characterNo : int) : void
+getCharacterPrevs() : Image[]
ButtonManager
<<Property>> -buttons : KeyCode[]
-buttonManager : ButtonManager
+ButtonManager()
+getInstance() : ButtonManager
GameFrame
-timer : AnimationTimer
~scene : Scene
~timeline : Timeline
-gameScene : Scene
<<Property>> ~stopped : boolean
+mediaplayer : MediaPlayer
- : GameEngine
-initializes : GameController
+start() : Scene
-createKeycode() : KeyCode[]
+playSong() : void
-updateFrame() : void
initializes
1
GameEngine
-timer : Timer
-currentAltitude : int
-currentScore : int
-gameFinished : boolean
<<Property>> -gamePaused : boolean
-difficulty : int
-pane : Pane
~mapLevel : int
-initializes : MapGenerator
-controlls : PauseManager
-calls : CollisionManager
-camera : Camera
<<Property>> -manages : Map
+GameEngine()
+convertMapToPane() : Pane
+GameEngine(difficulty : int, buttons : KeyCode[])
+setCurrentCharactersImages(images : Image[]) : void
+moveCharacterLeft() : void
+moveCharacterRight() : void
+jumpCharacter() : void
+stopMoveCharacterLeft() : void
+stopMoveCharacterRight() : void
+stopJump() : void
+setGameButtons(buttons : char[]) : void
+updateScore() : void
+initializeMap(diff : int) : void
+followCharactersAltitude() : void
+isGameOver() : boolean
+startGame() : void
+finishGame() : void
+pauseGame() : void
+continueGame() : void
+exitGame() : void
+changeVolume() : void
+changeMusic() : void
Display
-SCENE_WIDTH : int = 800
-SCENE_HEIGHT : int = 600
-mainMenuScene : Scene
-howToPlayScene : Scene
-creditsScene : Scene
-exitScene : Scene
-settingsScene : Scene
-highScoreScene : Scene
-soundSettingsScene : Scene
-buttonSettingsScene : Scene
-characterSettingsScene : Scene
-gameOverScene : Scene
~initializes : GameFrame
+main(args : String[]) : void
+Display()
+start(primaryStage : Stage) : void
-initializeScene(fxmlName : String) : Scene
-initializes
1
sends inputs
1
-calls
1
GameController
~sc : Scene
~sends inputs : GameEngine
+GameController(scene : Scene, keyCode : KeyCode[], gameEngine : GameEngine)
+moveLeft() : void
+moveRight() : void
+stopMoveLeft() : void
+stopMoveRight() : void
+stopJump() : void
+jump() : void
+pause() : void
CollisionManager
~current : Bar
~looper : Bar
+checkCollision(gameObjects : ArrayList<GameObject>) : void
+CollisionManager(gameObjects : ArrayList<GameObject>)
1
manipulates
-initializes
1
MapGenerator
-difficulty : int
<<Property>> -generates : ...
+MapGenerator()
+MapGenerator(map : Map)
+createNextLevels() : void
-generates
1
-manages
1
Bonus
<<Property>> -IMAGE_NO : int = 3
<<Property>> -boost : double
<<Property>> -time : double
+Bonus()
Wall
+Wall()
GameObject
<<Property>> -images : Image[]
<<Property>> -posX : int
<<Property>> -posY : int
+GameObject()
+GameObject(images : Image[])
-creates
*
Bar
<<Property>> -width : int
-left : int
-right : int
-top : int
-bottom : int
+Bar()
+Bar(images : Image[])
Wooden
<<Property>> -breaktime : double
<<Property>> -IMAGE_NO : int = 1
+Wooden()
+Wooden(images : Image[])
Icy
<<Property>> -slipperiness : double
-IMAGE_NUM : int = 3
+Icy()
+Icy(images : Image[])
+getIMAGE_NO() : int
HardlyVisible
<<Property>> -opacity : double
<<Property>> -IMAGE_NO : int = 2
+HardlyVisible()
+HardlyVisible(images : Image[])
Sticky
<<Property>> -stickyness : double
<<Property>> -IMAGE_NO : int = 1
+Sticky()
+Sticky(images : Image[])
Character
<<Property>> -verticalVelocity : int
<<Property>> -horizontalVelocity : int
<<Property>> -jumpPower : int
<<Property>> -IMAGE_NO : int = 6
<<Property>> -movingLeft : boolean
<<Property>> -movingRight : boolean
<<Property>> -standing : boolean
+Character()
+Character(images : Image[])
+getCurrentImage() : Image
1
-initializes and manipulates
Map
+GAME_LEFT_LIMIT : int = 50
+GAME_RIGHT_LIMIT : int = 630
+ACCELERATION : int = 1
+INITIAL_CHARACTER_SPEED : int = 0
<<Property>> -level : int
<<Property>> -passedLevel : int
<<Property>> ~altitude : int
-rand : Random
-gravity : int
-characterMoveSpeed : int = model.entity.Map.INITIAL_CHARACTER_SPEED
<<Property>> -creates : GameObject
-initializes and manipulates : Character
~manipulates : CollisionManager
-oldPlayerPosition : OldPlayerPosition
+Map()
+Map(map : Map)
+setCurrentCharactersImages(images : Image[]) : void
+createNextAltitudeObjects(diff : int) : void
+createFullWidthLevel() : void
+createBonus() : void
+updateObjects() : void
+updateCharacter() : void
+createLevel(type : int) : void
+moveLeft() : void
+moveRight() : void
+stopMoveRight() : void
+stopMoveLeft() : void
+stopMoveJump() : void
+jump() : void
+gameOver() : boolean
+getGameObjects() : ArrayList<GameObject>

Controller Classes(MainMenuController, GameController etc.):

These classes are responsible for handling user actions on each frame, for instance MainMenuController is responsible for user actions on MainMenu. Each of these controller takes input from a fxml file(Simply files that creates a GUI using JavaFX) and sends the user actions to related entity classes. For instance CharacterSettingsController handles the changes made to Character Appearance by sending user inputs to CharacterManager.

ButtonManager

ButtonManager class will give the opportunity to the user to define her own buttons to play the game. There will be 3 buttons (left, right and jump) and ButtonManager will hold each of them individually. By default, left-right arrows and space button are arranged.

SoundManager

SoundManager class will have methods to deal with the music and game sounds. User should be able to determine the volume of the music and game sounds, and can change the music as well. SoundManager will provide the features. SoundManager will be accessible from the Pause Menu as well.

Display:

Display class is responsible for initializing the scenes for the GUI.

GameFrame:

GameFrame class is responsible for rendering game objects during the gametime.

PauseManager:

PauseManager will give user to change sound settings while stops the game.

MapGenerator:

Map generator will create a map according to game difficulty.

CollisionManager:

CollisionManager will detect the interactions between map objects and the character.

Map:

Map will generate the GameObjects depends on the level and difficulty of the game. Higher levels and harder difficulties will cause harder map generations (with shorter or different types of bars).

GameObject:

GameObject will be the abstract class that contains information every game object should contain such that x and y positions, images etc.

Bar:

Bar will be an abstract class that holds the information that every bar should contain such that 2 images (ends and middle), length and height of bar.

Sticky:

Sticky will be a type of bar that slows down the character.

Icy:

Icy will be a type of bar that causes character to slip and accelerate.

Wooden:

Wooden will be a type that does not change the speed of character.

Character:

Character will be the object that user controls. It will have several images to create vision of motion. It will have speed and jump constants that determine the distance.

Wall:

Wall will be the left and right ends of the map.

Bonus:

Bonus will be the collectible that causes to player to boost its speed for a certain period of time.

5.4 User interface - Navigational paths and Screen mock-ups

5.4.1 Navigational Path

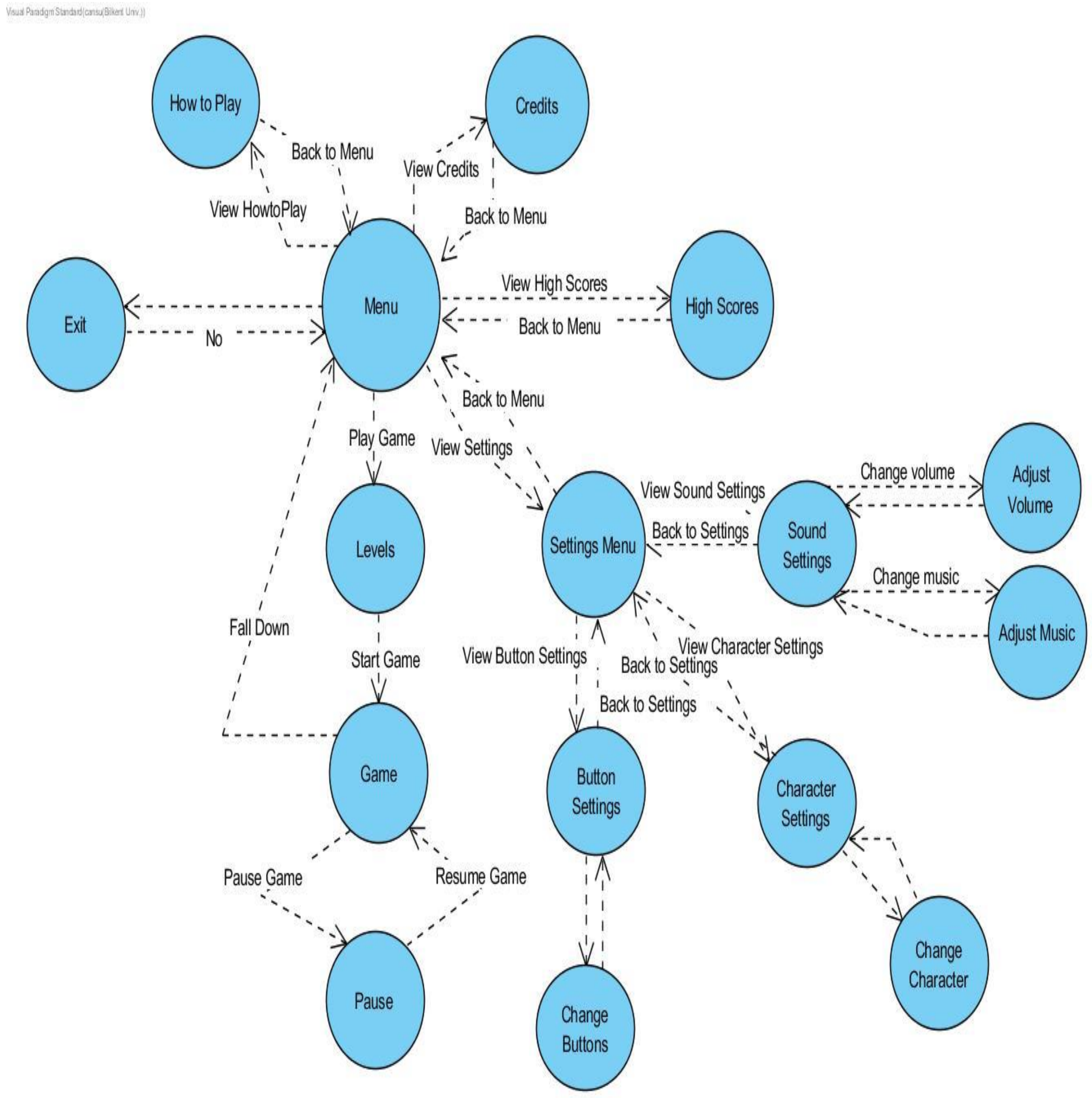


Figure 5.4.1.1 Navigational Path

5.4.2 Screen Mockups

- Main Menu

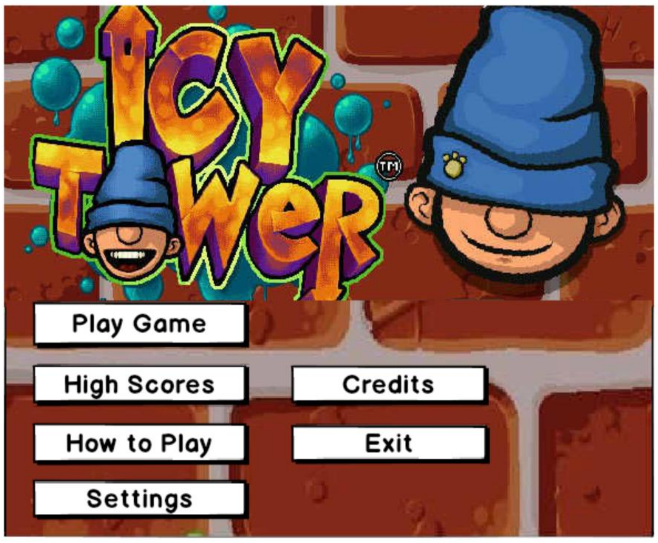
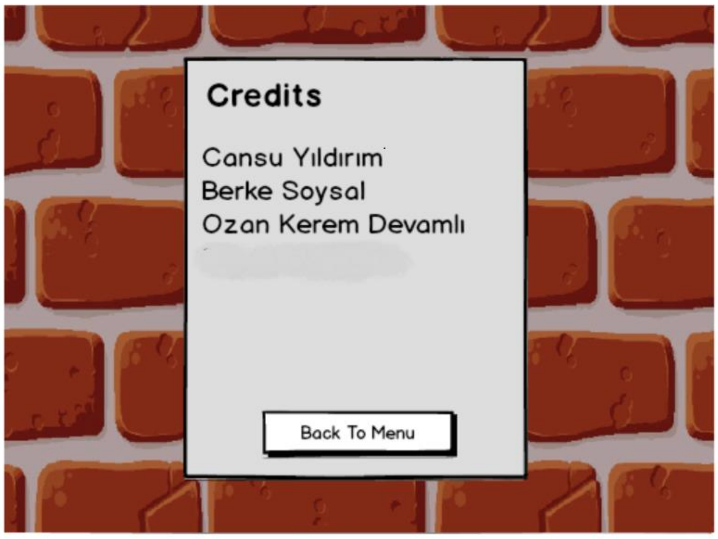


Figure 5.4.2.1 Main Menu Mockup

In the main menu there are 6 buttons. Play Game, High Scores, How to Play, Settings, Credits and Exit buttons. By clicking each of the buttons, relative pages open. When the play game button is clicked, icy tower will start.

- Credits



Credits

Cansu Yıldırım
Berke Soysal
Ozan Kerem Devamlı

Figure 5.4.2.2 Credits Screen Mockup

Credits screen shows the developers of the game.

- Play Game

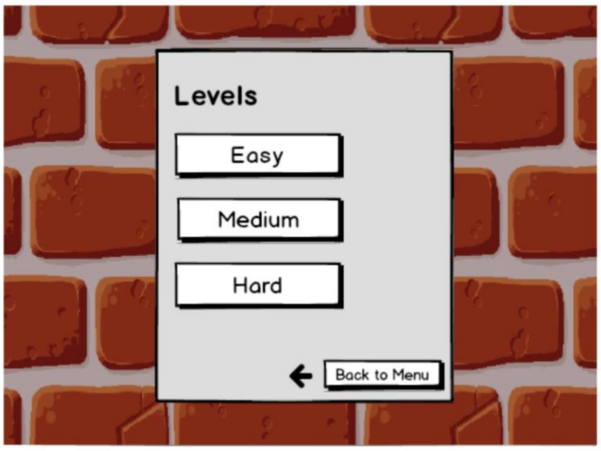


Figure 5.4.2.3 Start Menu Mockup

When the player selects “Play Game” button on the main menu, the difficulty level will be asked. After choosing the level, the game will start.

- Settings

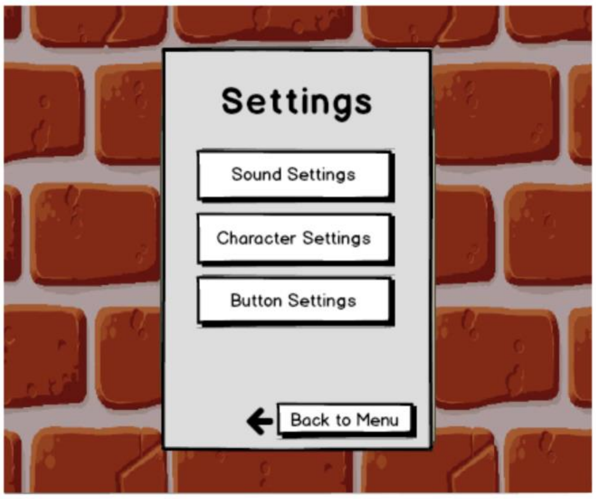


Figure 5.4.2.4 Settings Screen Mockup

In the settings screen, there are 3 buttons for sound setting, character setting and button setting. Player can click these buttons to change the relevant settings. Or player can go back to the main menu by clicking the Back to Menu button.

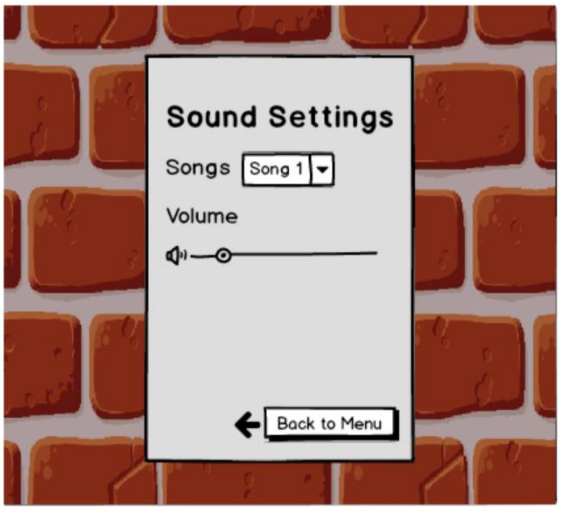


Figure 5.4.2.5 Sound Settings Screen Mockup

In the Sound Settings screen, player can change the music that is played during the game. Also, player can arrange the volume of the sound.

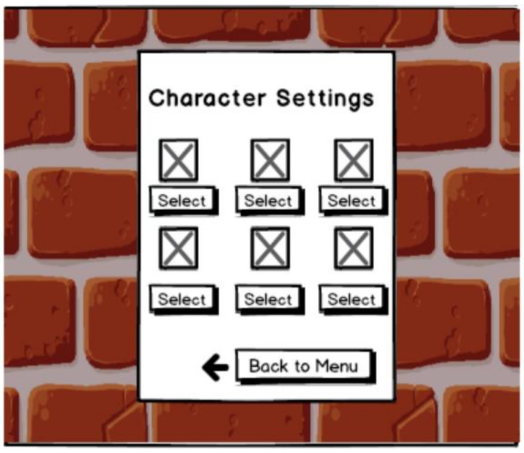


Figure 5.4.2.6 Character Settings Screen Mockup

In the Character Settings screen, player can select the character. The characters appearances will be different. Player must click the button under the chosen character.

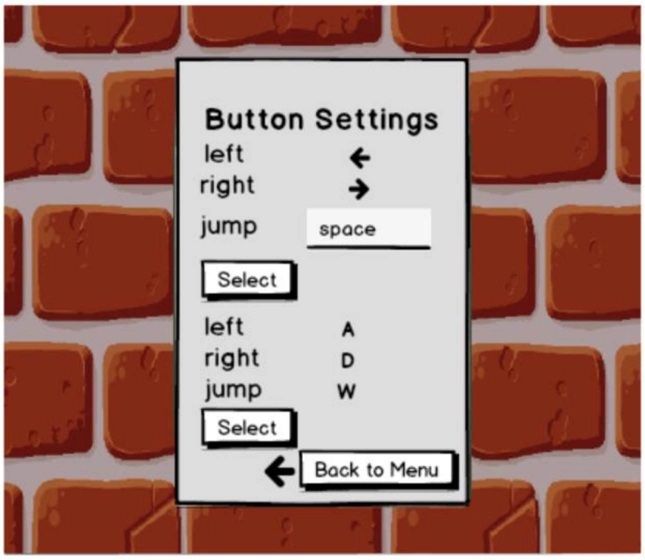


Figure 5.4.2.7 Button Settings Screen Mockup

In the Button Settings screen, player can choose the control buttons that will be used during the game. If player wants to control the character with right-left arrow and space, s/he must click the upper button. If s/he wants to control with A,D,W, s/he must click the below button. If player does not adjust the button setting, game will automatically be controlled by left,right arrow and space.

To go back the main menu, player must select the “Back to Menu” button on the bottom of the screen.

- Exit

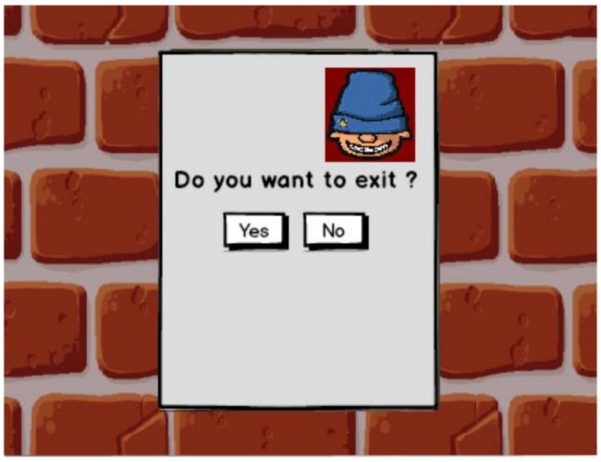


Figure 5.4.2.8 Exit Screen Mockup

Exit screen displays the question in order to be sure about the player's decision. If the player is sure about quitting the game, s/he must click the "Yes" button, otherwise "No" button to go back the main menu.

- High Scores

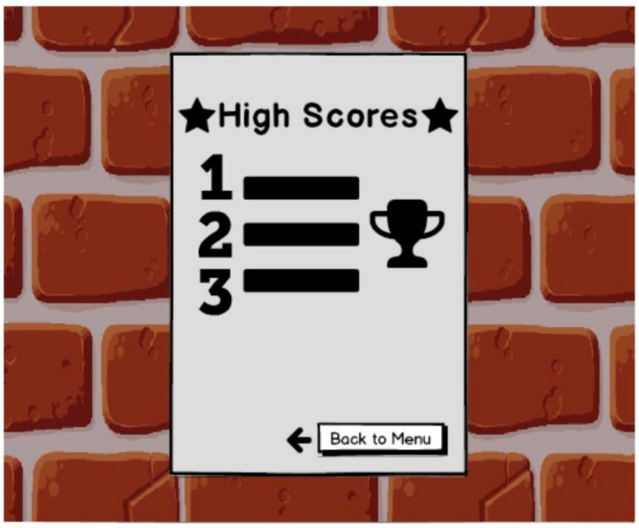


Figure 5.4.2.9 High Scores Screen Mockup

High Scores screen shows the scores of the players. When a player gets a score that is proper for the top ten, a name will be asked. With this name, the score will take its place in the list. To turn the main menu, player must select the Back to Menu button.

- Pause Menu and Sound Setting

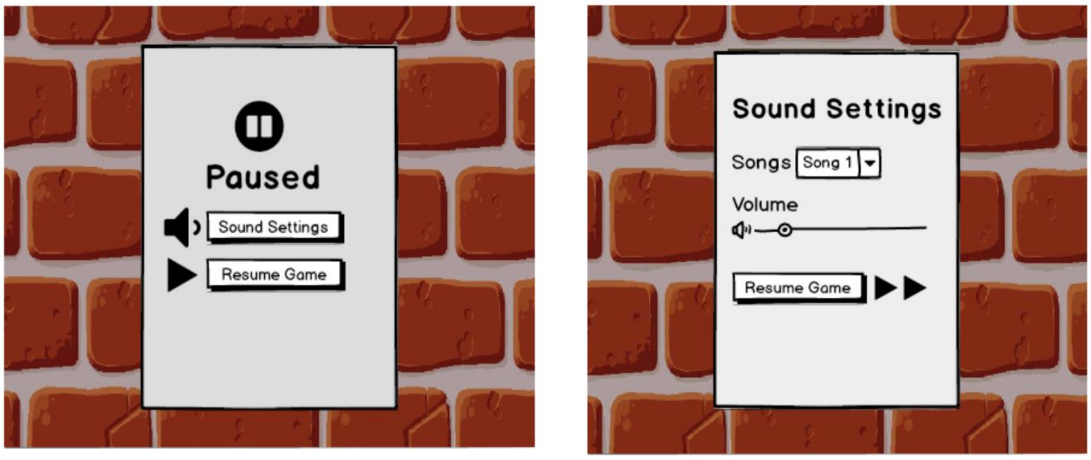
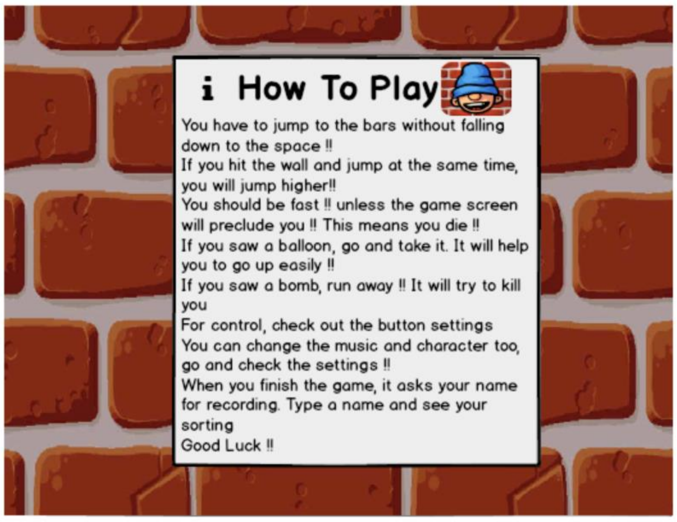


Figure 5.4.2.10 Pause Menu Screen and Sound Settings Screen Mockups

To open the pause menu, player must be playing the game. If the player selects the pause button on the game screen, pause menu will be opened. From this menu, player can go to the sound settings and adjust the sound. On the sound setting menu, player must click the resume game button to resume the game. Or player can click the resume game button from the pause menu, if s/he did not go to the sound settings.

- How to Play



i How To Play

You have to jump to the bars without falling down to the space !!
If you hit the wall and jump at the same time, you will jump higher!!
You should be fast !! unless the game screen will preclude you !! This means you die !!
If you saw a balloon, go and take it. It will help you to go up easily !!
If you saw a bomb, run away !! It will try to kill you
For control, check out the button settings
You can change the music and character too, go and check the settings !!
When you finish the game, it asks your name for recording. Type a name and see your sorting
Good Luck !!

Figure 5.4.2.11 How To Play Screen Mockup

How to play screen displays the logic of the game and general information about the game. By clicking the button "Back to Menu", player can go back to the main menu.

- Game Screen

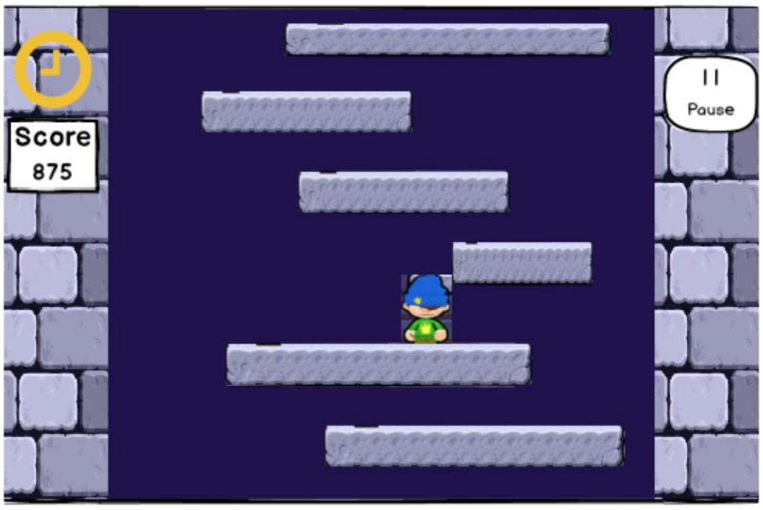


Figure 5.4.2.12 Game Screen Mockup

The game screen of our game will be like this. There will be bars that character can move above them. There will be pause button to pause the game and open the pause menu. The current score of the game can be seen while the game continues. Also, there will be a clock to calculate when the game screen increases its moving speed.

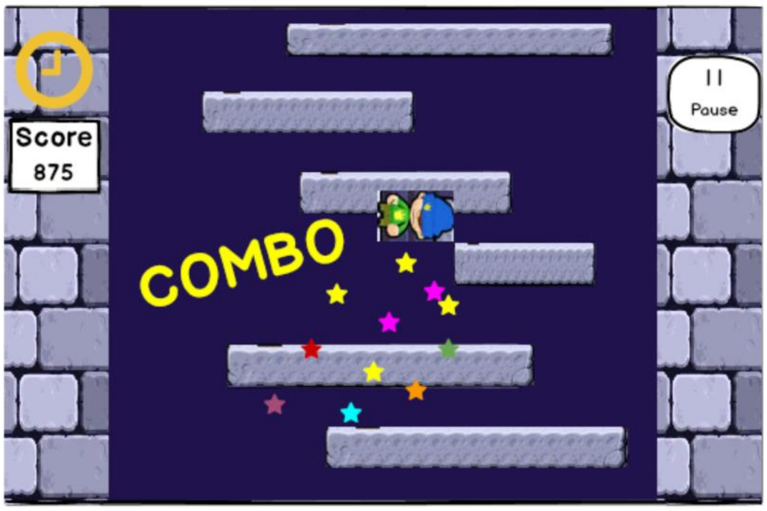
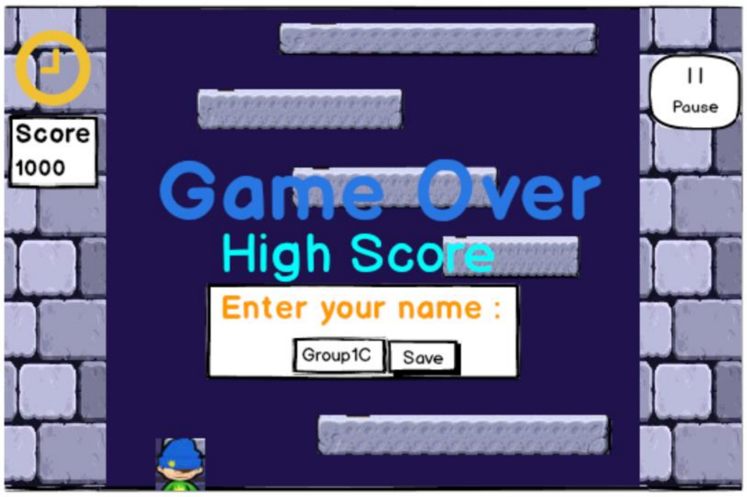


Figure 5.4.2.13 Game Screen Mockup

When player makes the combo jump, character will go upward spirally. “Combo” writing and colourful stars will appear on the screen in order to urge the player about making more combos.



“Game Over” writing will appear in the pursuit of the character’s fall. If the character makes new high score as explained more detailed above, player will be asked to type a name. After typing a name, player must click the “Save” button to have a place in the high scores list. After the name is saved, game returns to the main menu.

6. Improvement Summary

Improvements for Implementation:

We made remarkable progress with implementation. We implemented the basic game engine and the GUI. We made slight improvements for the object design of the project.

As for the game engine, we implemented movements of the game character, implement the map generation class, and created the bar game objects. The player can jump to bars, move left or right, and the camera goes higher as the player moves upward. What we are still working on is: collectible objects, combo jump for player, more smoothness for animations and movement, a better camera movement algorithm, and the score counter.

We find images for characters and game objects, and make animated images for a better user experience.

For the GUI part, it's almost complete. All menu frames except HighScoreFrame and PauseMenu are functionable. After implementing the score counter, the HighScore menu will be functionable.

Improvements for the Analysis & Design:

Use Case diagram and Dynamic Models (Sequence Diagrams and Activity Diagram) are updated. The functionalities that player can use are better explained in Use Case diagram by expanding the use cases for play game.

Class Diagram is updated. While we are implementing, we realized that some of the classes are not needed and some new classes are needed. Thus, this brings a new and more complicated class diagram. We mainly expanded our class model with controller for every menu frame. Then we decided some of the manager classes are just overheads, we decide to remove them.

New functional and non-functional requirements are added. For functional requirements, different types of collectible bonuses are added to the game. We decided that it will be a good idea to make our game work on mobile devices, so we add portability as a non-functional requirement.

7. Glossary & References

[1] "Icy Tower." Encyclopedia Gamia, gaming.wikia.com/wiki/Icy_Tower.

[2] Bruegge. *Object-Oriented Software Engineering*.