



# CS 319 - Object-Oriented Software Engineering

## Final Report

### Dragon Valley

Fall 2015 – Section 1 – Group 11

Ömer Akgül - 21301854

Güray Baydur - 21300715

Burak Bör - 21200595

Orkhan Namazov – 21002716

## Table of Contents

1. INTRODUCTION (PROBLEM STATEMENT).....	3
2. REQUIREMENT ANALYSIS.....	5
2.1 Overview.....	5
2.2 Functional Requirements.....	6
2.3 Non-Functional Requirements.....	8
2.4 Constraints.....	9
2.5 Scenarios.....	9
2.6 Use Case Model.....	15
2.7 User Interface.....	23
3. ANALYSIS.....	32
3.1 Object Model.....	32
3.1.1 Domain Lexicon.....	32
3.1.2 Class Diagram.....	36
3.2 Dynamic Models.....	39
3.2.1 State Charts.....	39
3.2.2 Sequence Diagrams.....	41
4. DESIGN.....	48
4.1 Design Goals.....	48
4.2 Sub-System Decomposition.....	49
4.3 Architectural Patterns.....	50
4.4 Hardware/Software Mapping.....	51
4.5 Addressing Key Concerns.....	51
4.5.1 Persistent Data Management.....	51
4.5.2 Access Control and Security.....	52
4.5.3 Global Software Control.....	52
4.5.4 Boundary Conditions.....	53
5. OBJECT DESIGN.....	54
5.1 Pattern Applications.....	54
5.2 Class Interfaces.....	57
5.3 Specifying Contracts.....	70
6. CONCLUSION AND LESSONS LEARNED.....	78

## **1. Introduction (Problem Statement)**

“Dragon Valley” is a basic game that originated from the old popular mobile phone game “Snake”. “Snake” is a game that player tries to control the snake -whose shape is like a straight line- with arrow keys and tries to eat as many as fruits without crossing over on its body (snake’s body grows when it eats a fruit). There are lots of games that used “Snake’s” template. Likewise, in our project, we used this template and tried to add new features to the original game to create a more fun and different game from its ancestor to break its monotonous gameplay which is the problem of the original game.

The purpose of “Dragon Valley” is the same as its ancestor: Try to eat as many as animals with our dragon without crossing over its body but there are different types of challenges and levels differ from the original game!

The game is a desktop application, it’s written with Java and can be controlled by arrow keys and space button on the keyboard.

This report is an analysis of the system we developed which can be seen as a requirements specification report. It includes description of the case, analysis of requirements like functional and non-functional requirements, constraints, scenarios of the system, use case diagrams, user interface visuals, class diagrams of object models, state charts and sequence diagrams of dynamic models.

Our project’s main goal is to provide a more intriguing and challenging game time distinctively than classical “Snake” game. The original game is promising a fun game time that will need player’s attention and reflex capabilities but it is very monotonous. Player always tries to survive and gain points in the same area and it can be boring after some play time. Our goal is to take this “surviving and gaining points” template and adding new features that would make the

game more fun. Briefly, we tried to create a more versatile game including new challenges and saving the temple from becoming an ordinary game.

There are different animals whose can give our dragon different points. There are some blocks and enemies and our dragon is capable of breathing fire from its head and with this fire, it can destroy these different types of foes to gain more points. There are different stages with different difficulty levels and our dragon must reach a point threshold to pass to next stage.

## **2. Requirement Analysis**

### **2.1 Overview**

“Dragon Valley” is a survival arcade game which is easy to play and provide fun game time while trying to reach next stages. Basically, the game have a board and all of the game continues on this board. Player controls our dragon with arrow keys on the keyboard whose shape is like a straight line which can turn left, right, top and bottom. Initially, it starts small and becoming longer with gaining more points. The game will be over if a player’s dragon crosses over with its own body or on an enemy or a block.

There are different types of animals in the game which our dragon have to go through and eat them. By doing this, player gains points according to type of the animal. They all have different point values. They randomly appear on the board and some of them appears rarely in compare to other animals.

Player have to reach a threshold to pass to the next level. In every stage, game becoming difficult and hard to survive on the board. There are different foes occurring in the next levels. There are blocks and enemies that player have to take them out using dragon’s fire by pressing space button on the keyboard.

Before starting the game, program asks user’s name and keeping user’s name and score after the game in the database. So, many players can play the game and compare their score with the other ones who played before. There is a leaderboard on the screen that player can see his/her place on the leaderboard. After game ends, there is a full leaderboard appears

on the screen and shows player's place in the leaderboard. Also there are other notifications appear on the screen when game is over or game is paused.

There will be a main menu when the program starts and there are some options on the menu like starting a new game, seeing instructions of the game (help), muting the game (there will be a music playing on the background), changing the difficulty of the game, checking the credits and the leaderboard.

## **2.2 Functional Requirements**

- The user should be able to select any feature of the menu.

1. The user should be able to enter a new game.
2. The user should be able to go into options.
3. The user should be able to learn the rules of the game from help.
4. The user should be able to see leaderboard.
5. The user should be able to quit the game.

- 1) The user should be able to enter a new game.

- a. The user should be able to restart game any time during a game stage.

- b. The user should be able to pass the current stage (choose the difficulty).

- c. The user should be able to end game.

d. The user should be able to see the player that was recently beaten and is closest to beat.

e. The user should be able to see her current score.

f. The user should be able to view the points from recently eaten animal.

g. The user should be able to fire with its dragon.

h. The user should be able to move his dragon to right, left, up and down.

i. The user should complete the stage until the given time ends.

2) The user should be able to go into options.

a. The user should be able to adjust sound.

b. The user should be able to select the difficulty level.

3) The user should be able to learn the rules of the game.

a. The user should be able to see the instructions.

4) The user should be able to see the leaderboard.

a. The user should be able to observe the top players with their highscores.

5) The user should be able to quit the game.

a. The user should be able to close the menu.

## **2.3    *Non-Functional Requirements***

### **A. Usability Requirements**

- 1) The player should be able to understand the flow of the game from help section.
- 2) When the player understands that she is going to lose, she can quickly restart the game from the menu which is located at the upper part of the screen.
- 3) The player should be able to quit the game whenever she wants.
- 4) Game screen must be designed properly for upper menu, blocks, enemies, dragon and etc.
- 5) The stages must be changed from the upper menu during gameplay.
- 6) Difficulty of the stages must be adjusted sensitively since the difficulties may be so hard or so easy to pass.

### **B. Implementation Requirements**

- 1) Adding blocks, animals, in other words, creating a stage must not be hard to apply on the system. The code must be clear enough to be understood by any developer.

### **C. Performance Requirements**

- 1) The run time of the game should be fast enough to play without delays.
- 2) Creation of stage should not take more than 2 seconds.



## 2.4 Constraints

The language of the implementation must be Java and the IntelliJ IDE and Eclipse must be used for developing the game. Most of the contemporary games are written in Java or C++ so Java is a good option to select. Also IntelliJ IDE and Eclipse is easy to use with Github, each member of the group can contribute via Github and see the results quickly in IntelliJ environment.

The game should be analyzed, designed and implemented according to instructions that is given in CS319 Object Oriented Software Engineering Course. The examples from the lecture slides are the primary source for the creation of each diagram. The criteria of the course is the fundamental obligatory for the project.

## 2.5 Scenarios

**Scenario Name:** Play the current stage successfully

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player clicks on the New Game Button. After that the system initializes a board object in which the stage will be illustrated. The board object creates grids in which the dragon will travel and also enemies and blocks are placed by calling drawBoard (BufferImage) method. Then according to default settings in the options menu, a stage object is created by calling createStage(StageNumber) method. In addition to the creation of submenu, animals, blocks ,dragon and board, the Game Controller object will start the game. The time to complete the current stage begins. For each moment Game Controller object updates each object inside the game by calling their update methods. It simply redraws blocks and enemies if they are not destroyed by the fire of the dragon and also redraws animals. If a animal is eaten by the dragon which is controlled by the player, the score object is updated. Player moves the dragon without collision with blocks and

enemies. The player earns enough points to open the door that allows player to continue on next stage. The player moves towards the door and the stage is completed.

**Scenario Name:** Play the last stage successfully

Player goes into the last stage of the game. After that the system initializes a board object in which the stage will be illustrated. The board object creates grids in which the dragon will travel and also enemies and blocks are placed by calling drawBoard (BufferImage) method. Then according to default settings in the options menu, a stage object is created by calling createStage(StageNumber) method. In addition to the creation of submenu, animals, blocks ,dragon and board, the Game Controller object will start the game. The time to complete the current stage begins. For each moment Game Controller object updates each object inside the game by calling their update methods. It simply redraws blocks and enemies if they are not destroyed by the fire of the dragon and also redraws animals. If a animal is eaten by the dragon which is controlled by the player, the score object is updated. Player moves the dragon without collision with blocks and enemies. The player earns enough points to open the door that allows player to continue on next stage. The player moves towards the door and the stage is completed. The Leaderboard and Database is updated if the player reaches the highest score. The system turns the player back to the main menu.

**Scenario Name:** Dragon dies by interaction with an enemy

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player clicks on the New Game Button. After that the system initializes a board object in which the stage will be illustrated. The board object creates grids in which the dragon will travel and also enemies and blocks are placed by calling drawBoard (BufferImage) method. Then according to default

settings in the options menu, a stage object is created by calling `createStage(StageNumber)` method. In addition to the creation of submenu, animals, blocks ,dragon and board, the Game Controller object will start the game. The time to complete the current stage begins. For each moment Game Controller object updates each object inside the game by calling their update methods. It simply redraws blocks and enemies if they are not destroyed by the fire of the dragon and also redraws animals. If a animal is eaten by the dragon which is controlled by the player, the score object is updated. Player moves the dragon without collision with blocks. However, with a little slow reaction, the player runs into an enemy. The Game Controller stops the Game and displays the Game over Screen.

**Scenario Name:** Dragon dies when time ends

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player clicks on the New Game Button. After that the system initializes a board object in which the stage will be illustrated. The board object creates grids in which the dragon will travel and also enemies and blocks are placed by calling `drawBoard (BufferImage)` method. Then according to default settings in the options menu, a stage object is created by calling `createStage(StageNumber)` method. In addition to the creation of submenu, animals, blocks ,dragon and board, the Game Controller object will start the game. The time to complete the current stage begins. For each moment Game Controller object updates each object inside the game by calling their update methods. It simply redraws blocks and enemies if they are not destroyed by the fire of the dragon and also redraws animals. If a animal is eaten by the dragon which is controlled by the player, the score object is updated. Player moves the dragon without collision with blocks and enemies. Nevertheless, player forgets to collect enough animals to obtain sufficient points for the current stage and the system notifies him by displaying a dynamic message that the game

will be over if he does not eat more animals. Moreover, player does not reach enough points to open the door so the game controller ends the game and the system displays game over screen.

**Scenario Name:** Dragon dies by interaction with a block

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player clicks on the New Game Button. After that the system initializes a board object in which the stage will be illustrated. The board object creates grids in which the dragon will travel and also enemies and blocks are placed by calling drawBoard (BufferImage) method. Then according to default settings in the options menu, a stage object is created by calling createStage(StageNumber) method. In addition to the creation of submenu, animals, blocks ,dragon and board, the Game Controller object will start the game. The time to complete the current stage begins. For each moment Game Controller object updates each object inside the game by calling their update methods. It simply redraws blocks and enemies if they are not destroyed by the fire of the dragon and also redraws animals. If a animal is eaten by the dragon which is controlled by the player, the score object is updated. Player moves the dragon without collision with enemies. Nevertheless, player does not react quick enough to prevent hitting a block. The Game Controller realizes a collision occurred at that moment and ends the game. The system displays the game over screen which is showing the reached score, “Return Main Menu” and “Restart Game” button.

**Scenario Name:** Changing Settings from main menu

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player wants to make some arrangements in the game settings so he clicks the Options Menu. The system displays 2

options which are adjusting sound (enable/disable) and choose difficulty level. Firstly, the player disables the sound and then it selects the difficulty level of the stages (hard, easy etc.). He also wants them to be his saved settings so he clicks on “Set Default” Button. The player turns back to Main Menu via Back Button.

**Scenario Name:** Skipping the current stage

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player selects the New Game Button and begins the game. However, player realizes that the stage is so easy to complete or he realizes that the stage is too hard to complete and his dragon will die soon. During gameplay, the player uses the submenu in the gameplay and clicks on Next Stage button to go on a harder level or he simply wants restart so he clicks on Restart Game Button placed in the submenu. According to the player's wish the system shows the appropriate menu.

**Scenario Name:** Looking to the Top players

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player selects the Leaderboard button. The system displays the top scorers of the game. The player observes the top scorers and clicks the Back Button to return Main Menu.

**Scenario Name:** Looking to the rules and instructions

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player selects the Help button. The system displays regulations of the game. The player observes instructions and clicks the Back Button to return Main Menu.

**Scenario Name:** Looking to Creators

Player opens the Main Menu of the Game. The System displays 6 Buttons with the names New Game, Leaderboard, Options, Help, Credits, Exit Menu. Player selects the Credits button. The system displays developers of the game. The player observes the name list and clicks the Back Button to return Main Menu.

2.6 Use Case Model

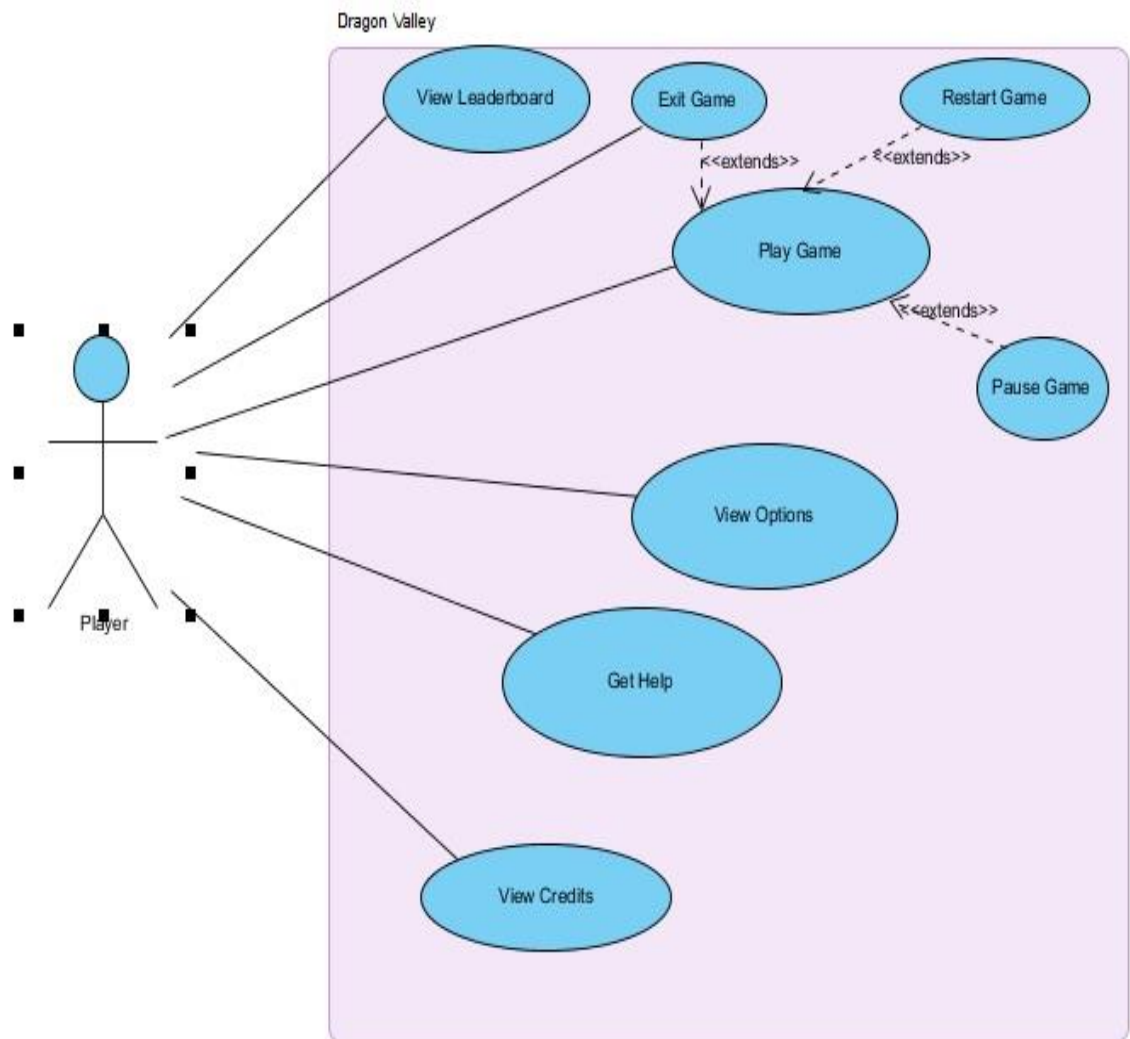
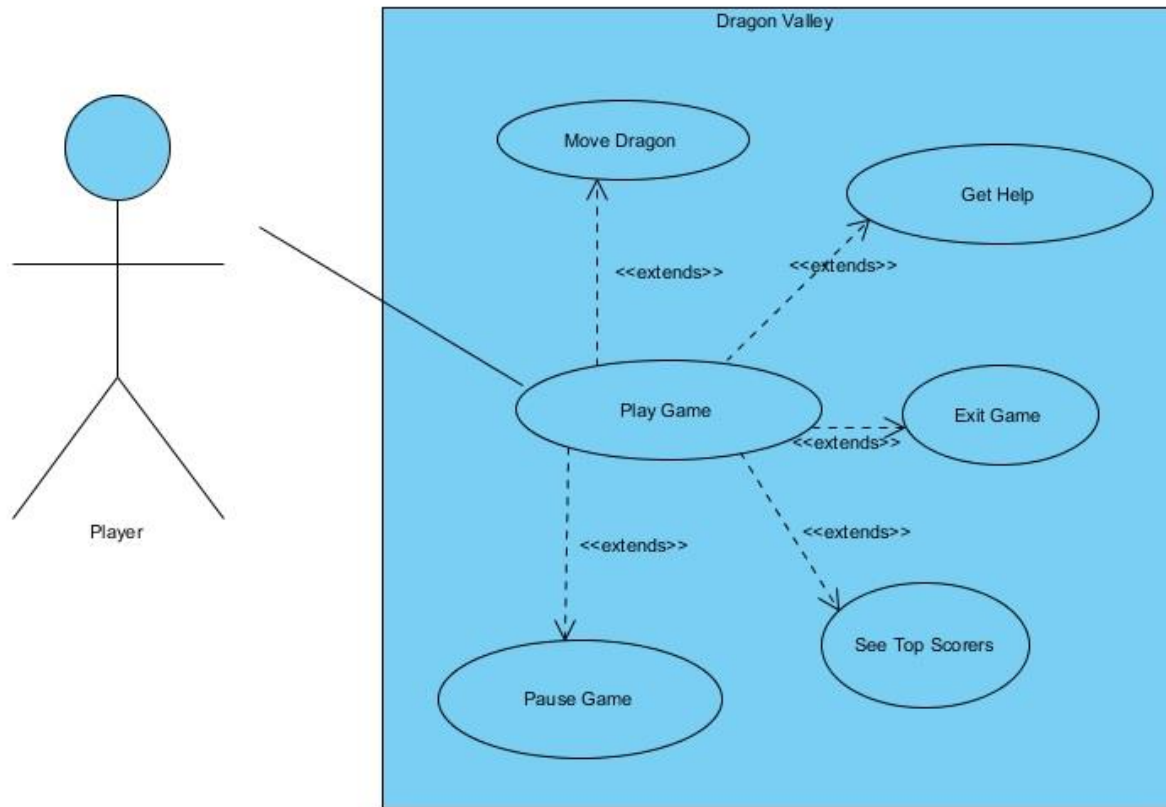


Figure 1.1: Use Case Model of Menu



**Figure 1.2: Use Case Model of in-game**

**Use Case Name: View Leaderboard**

**Primary Actor:** Player

**Stakeholders and Interests:**

- Player wants to see top 5 scores with player names.
- System shows top 5 players with their nicknames.

**Pre-conditions:** Top 5 scorers' points are kept record by the system

**Post-condition:** -

**Entry Condition:** Player selects "View Leaderboard" from Main Menu.

**Exit Condition:** Player selects "Back" to turn back Main Menu.



**Success Scenario Event Flow:**

1. Top 5 scorers are displayed by the system.

**Alternative Flows:**

A. If player desires to return main menu at any time:

A.1. Player selects “Return to Main Menu” button to return main menu.

A.2. System displays Main Menu.

**Use Case Name: View Options**

**Primary Actor:** Player

**Stakeholders and Interests:**

-Player desires to see the game options disabling or enabling sound.

-Player wants to choose the level difficulty.

-System updates the recently changed options.

**Pre-condition:** At first options are set as default. If Player arrange options different than default, changed settings will be saved and used by system.

**Post-condition:** Options are updated.

**Entry Condition:** Player selects “Options” button from main menu.

**Exit Condition:** Player selects “Back” to return main menu.

**Success Scenario Event Flow:**

1. Player presses “Options” button to make arrangements about game options.
2. Options of the game are displayed to Player in “View Options” screen by System.
3. Player adjusts settings according to her wish.
4. System updates game settings successfully.

**Alternative Flows:**

A. If Player desires to use default settings at any time:

A.1. Player chooses “Default” button from “View Options” screen.

A.2. System updates default game options.

B. If Player wants to return main menu at any time:

B.1. Player selects “Back” button from “Change Settings” screen.

B.2. System updates Game Options..

B.3. Player returns main menu.

**Use Case Name: View Credits**

**Primary Actor:** Player

**Stakeholders and Interests:**

-Player wants to see the creators of the game

**Pre-conditions:** -

**Post-condition:** -

**Entry Condition:** Player selects “View Credits” from Main Menu.

**Exit Condition:** Player selects “Back” to turn back Main Menu.

**Success Scenario Event Flow:**

1.System displays the developers.

**Alternative Flows:**

A. If player desires to return main menu at any time:

A.1. Player selects “Return to Main Menu” button to return main menu.

A.2. System displays Main Menu.

**Use Case Name: Get Help**

**Primary Actor:** Player

**Stakeholders and Interests:**

-Player wants to learn or revise the rules of the game.

**Pre-conditions:** -

**Post-condition:** -

**Entry Condition:** Player selects “Help”Button from Main Menu.

**Exit Condition:** Player selects “Back” to turn back Main Menu.

**Success Scenario Event Flow:**

1.System displays the instructions and rules.

**Alternative Flows:**

A. If player wants to return main menu at any time:

A.1. Player selects “Return to Main Menu” button to return main menu.

A.2. System displays Main Menu.

**Use Case Name: Play Game**

**Primary Actor:** Player

**Stakeholders and Interests:**

-Player desires to complete all stages and reach to the top of the leaderboard.

-The score of the Player recorded by the system.

**Pre-condition:** In the first run, the difficulty level is default as in the Options Menu.

**Post-condition:** According to the points of the player, the scoreboard is changed.

**Entry Condition:** Player chooses “Play Game” button from Main Menu.

**Exit Condition:** Player chooses “Return Main Menu” from Upper Menu.

**Success Scenario Event Flow:**

1.System starts the game.

2.The first stage begins with the default difficulty level. (Blocks are set, Time begins)

3.Player completes the stage by destroying blocks with fire, eating animals and earning sufficient points in the given time without hitting a block or an enemy.

4.System opens the door in the stage.

5.Player moves into to door and starts the next stage.

*Player does the steps 3 – 5 repeatedly until all stages are completed or Player hits a block or an enemy.*

6.System records the current points of the Player and if her score is greater than any other competitor on the Leaderboard, system updates the leaderboard.

7. System turns back to Main Menu.

**Alternative Flows:**

A. If player requests to pause the game at any time during the game:

A.1. Player presses the Pause Menu Button from the Upper Menu.

A.2. The game is paused by the System.

A.3. The pause menu is shown by the System.

A.3.1. If Player chooses “Continue” Button from pause menu, System continues the game.

A.3.2. If Player selects “Return to Main Menu” from pause menu, System directs Player Main Menu.

B. If dragon hits a block or an enemy.

B.1. The system stops the game.

B.2. The system shows the Game Over screen. (with Score, “Return Main Menu” and “Restart Stage” button)

B.2.1 If the player chooses “Return Main Menu” Button, the system displays the main menu.

B.2.2 If the player chooses “Restart Stage” Button, the system restarts the game.

C. If dragon eats a animal which is not dangerous.

C.1. The score screen is updated (score is increased) by the system with the point of the eaten animal type.

D. If the dragon eats a animal which is dangerous.

D.1 The score screen is updated (score is decreased) by the system with the point of the eaten animal type.

E. If the dragon collide with blocks the system ends the game and shows the game over screen.

F. If the given time approaches to finish time, the system notifies the player that the game is going to end.

G. If the player can not complete the game in the given time, the system ends the game and shows the game over screen.

H. If player hits an enemy, the system ends the game and shows the game over screen.

## *2.7 User Interface*

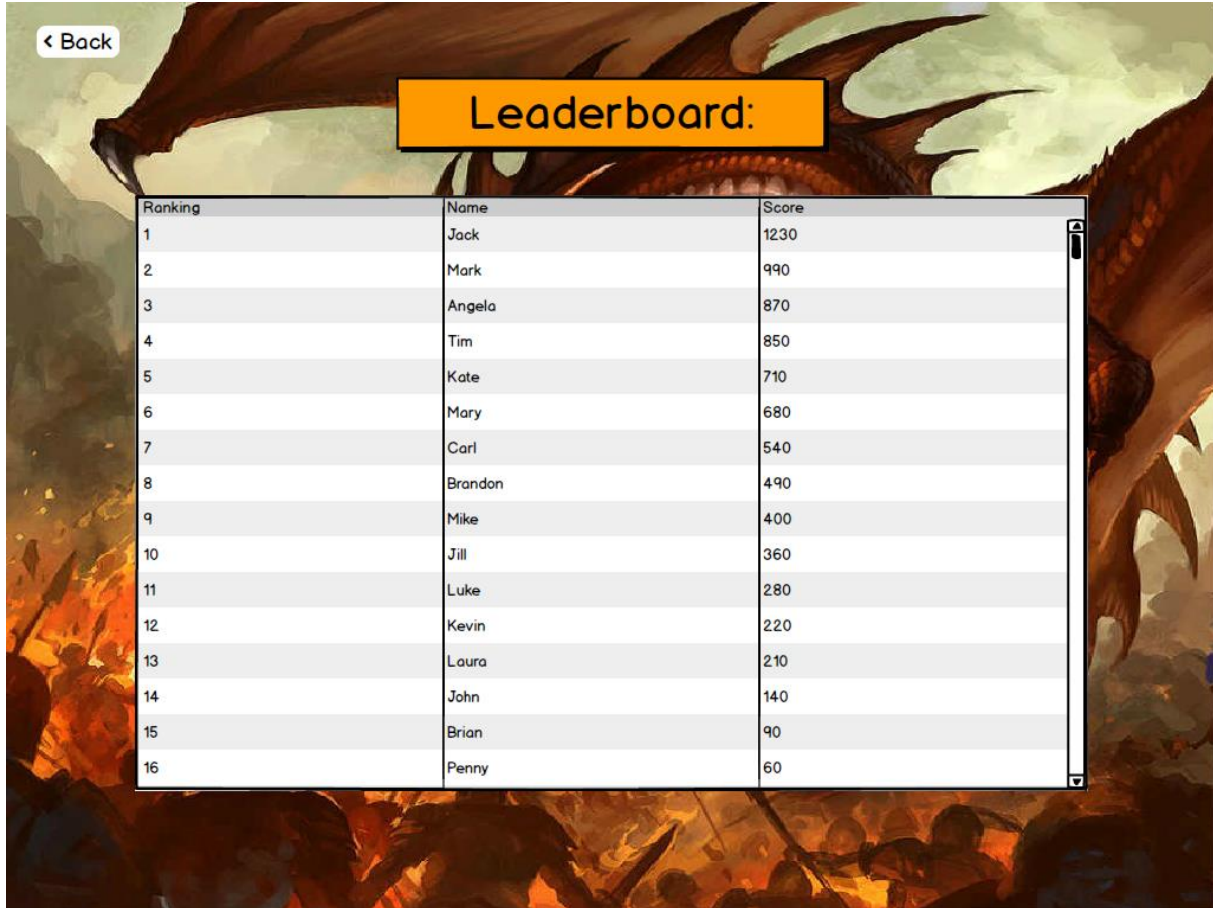
### **Main Menu:**



**Figure 2.1: User Interface of Main Menu**

When the application starts, user will encounter with this screen initially. On this interface, user can start a new game, check the leaderboard, check the help screen to learn game's instructions and rules, click options to go to the options screen to turn off/on the game's background music and to change the difficulty level, check credits to see the program's developers' information and exit the application.

## Leaderboard:



Ranking	Name	Score
1	Jack	1230
2	Mark	990
3	Angela	870
4	Tim	850
5	Kate	710
6	Mary	680
7	Carl	540
8	Brandon	490
9	Mike	400
10	Jill	360
11	Luke	280
12	Kevin	220
13	Laura	210
14	John	140
15	Brian	90
16	Penny	60

**Figure 2.2: User Interface of Leaderboard**

On this screen, user can check the leaderboard of the game to see his/her ranking. After playing the game user's name and score goes to the database and it keeps this scores. The leaderboard is the place that user can see this scores on the database sorted by points bigger to smaller. By clicking back button, user can return to main menu again.



## Help:

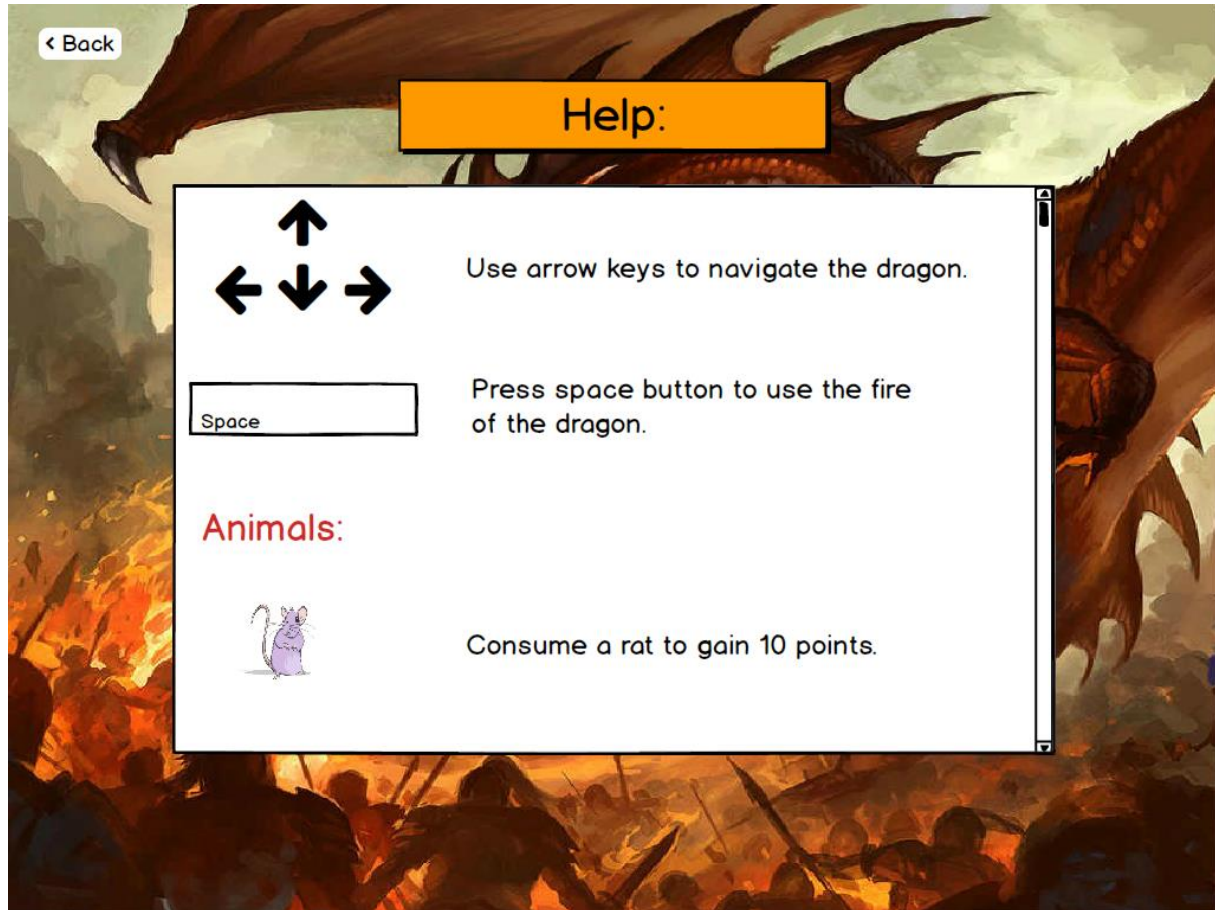
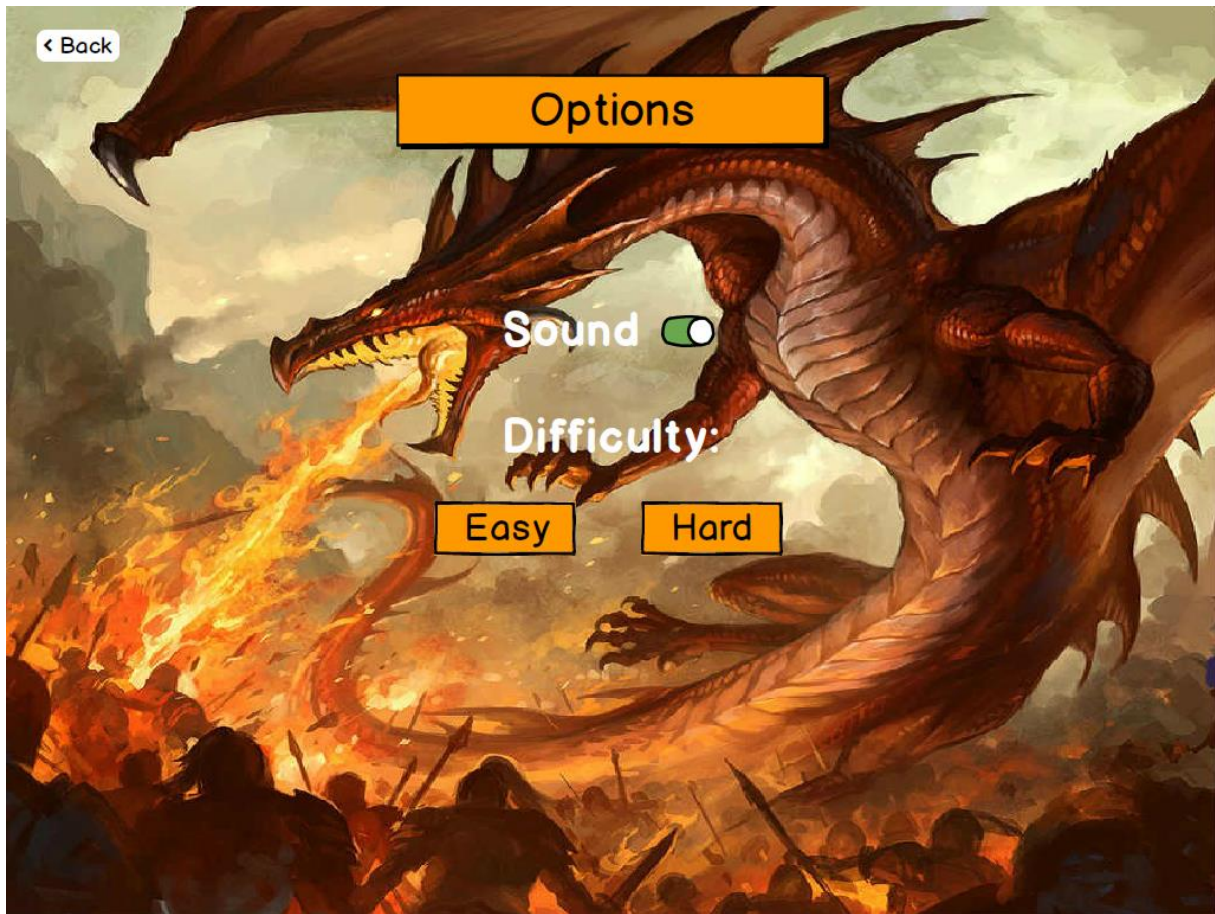


Figure 2.3: User Interface of the Help Section

On this screen, user can see the instructions and rules of the game like how can he/she play the game or which animal provides how many points etc. Like on the leaderboard screen, user can press back to go to the main menu.

## Options:



**Figure 2.4: User Interface of Options**

On the options screen, user can click the sound button to turn off/on the music of the game and choose a difficulty mode between easy and hard. If user selects hard mode, dragon become more fast and time for passing stages will decrease. By clicking back button, user can return to main menu.

## Credits:

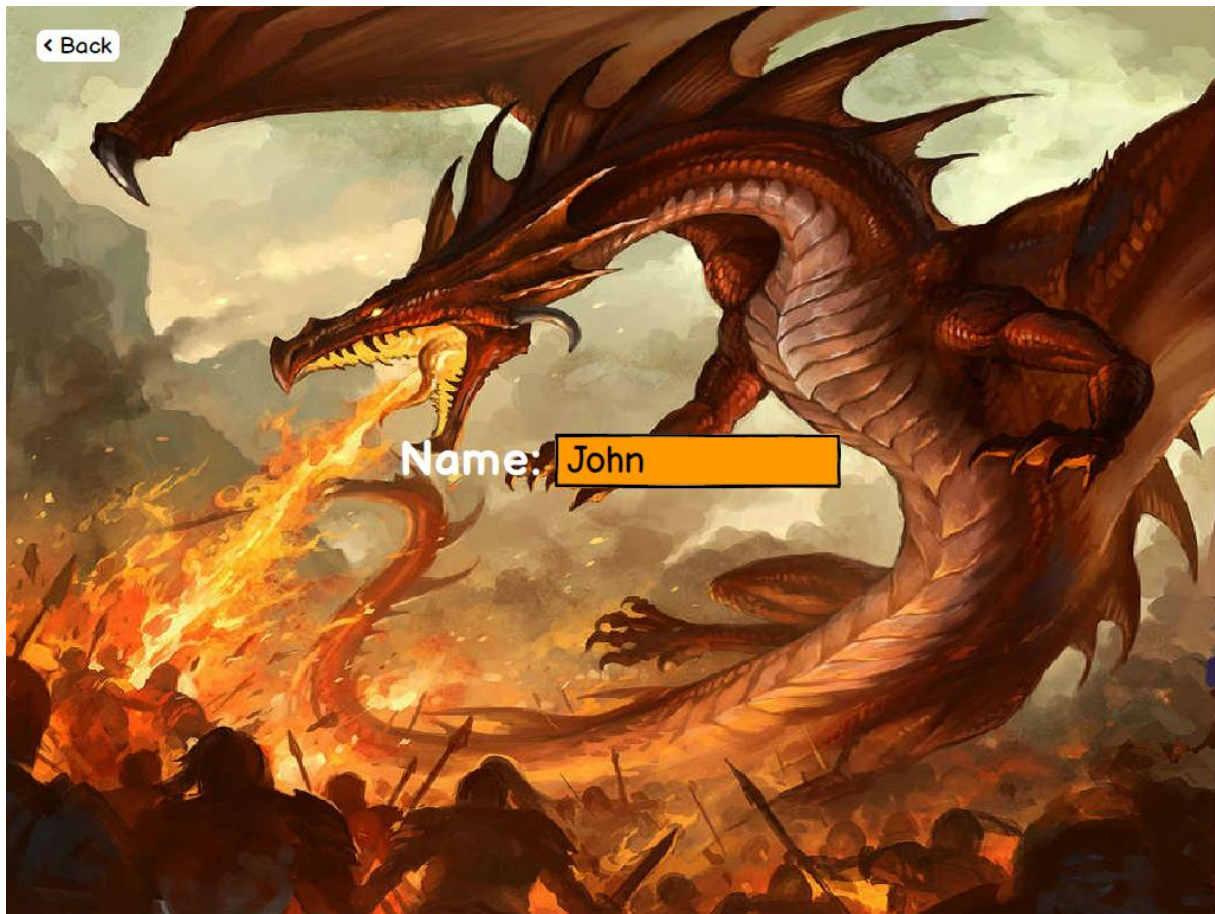


**Figure 2.5: User Interface of the Credits Section**

On this screen, user can see some information about application's developers. Back button does the same action like on the other screens.



## After New Game:



**Figure 2.6: User Interface of the After New Game Screen**

After clicking new game button on the main menu, application passes to this screen to learn user's name. User have to enter his/her name and after writing the name user have to press enter button on the keyboard to start the game.

## Game Starts:

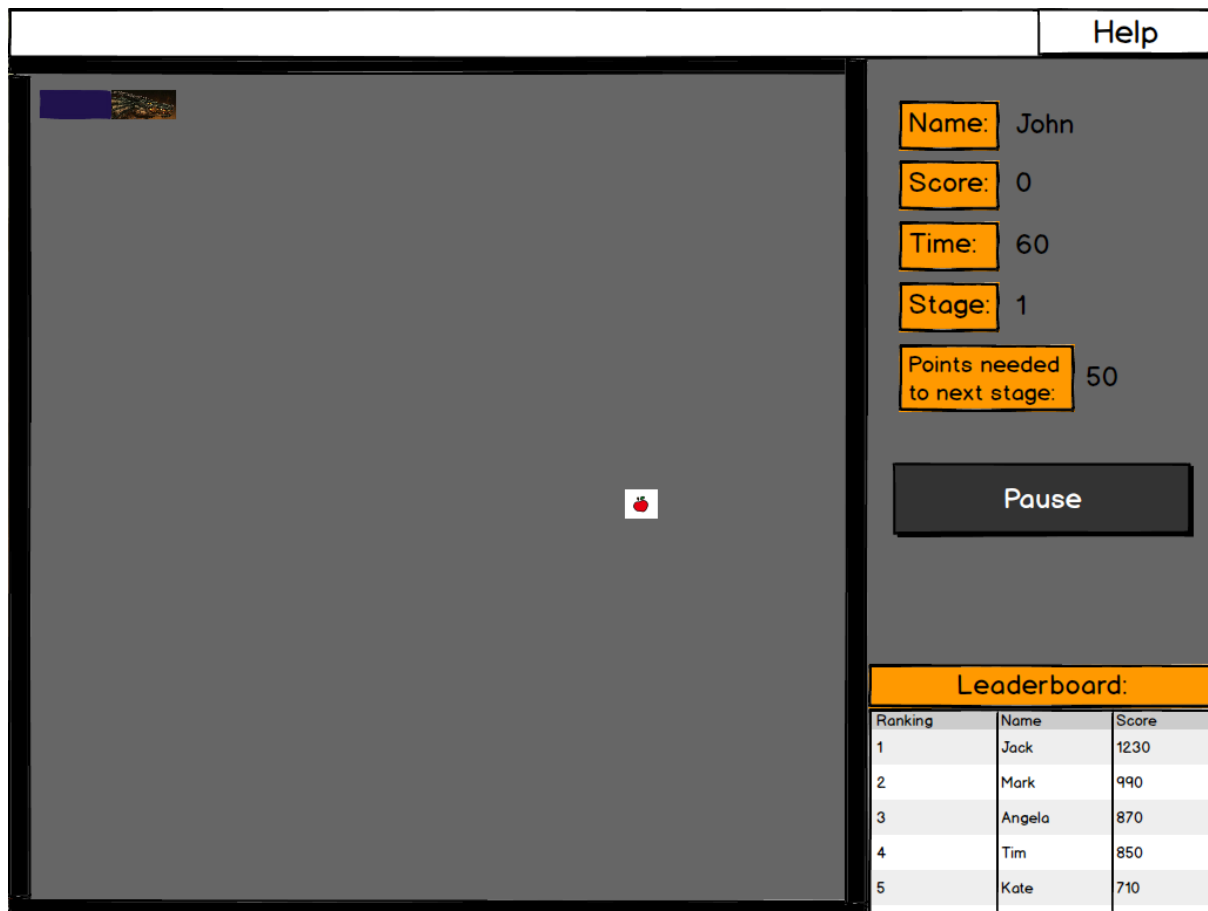


Figure 2.7: User Interface of just after game starts

After starting a new game, the game looks like this. On top left we can see our dragon and there is a animal to waiting to be consumed by our dragon. The left big square which our dragon and animal are inside, is the game board. Game continues on this place and if dragon touches the black borders of the board, game will be over. On the right section user can see the information about current game. User's name, total score of the user, remaining time to pass the stage, which stage user is on, points needed to pass to the next stage and a mini-leaderboard. There is a pause button, if user clicks this button game freezes until pressing this button again. Clicking the help button on the top right of the screen opens the help screen which can be reached from the main menu.

## Blocks:

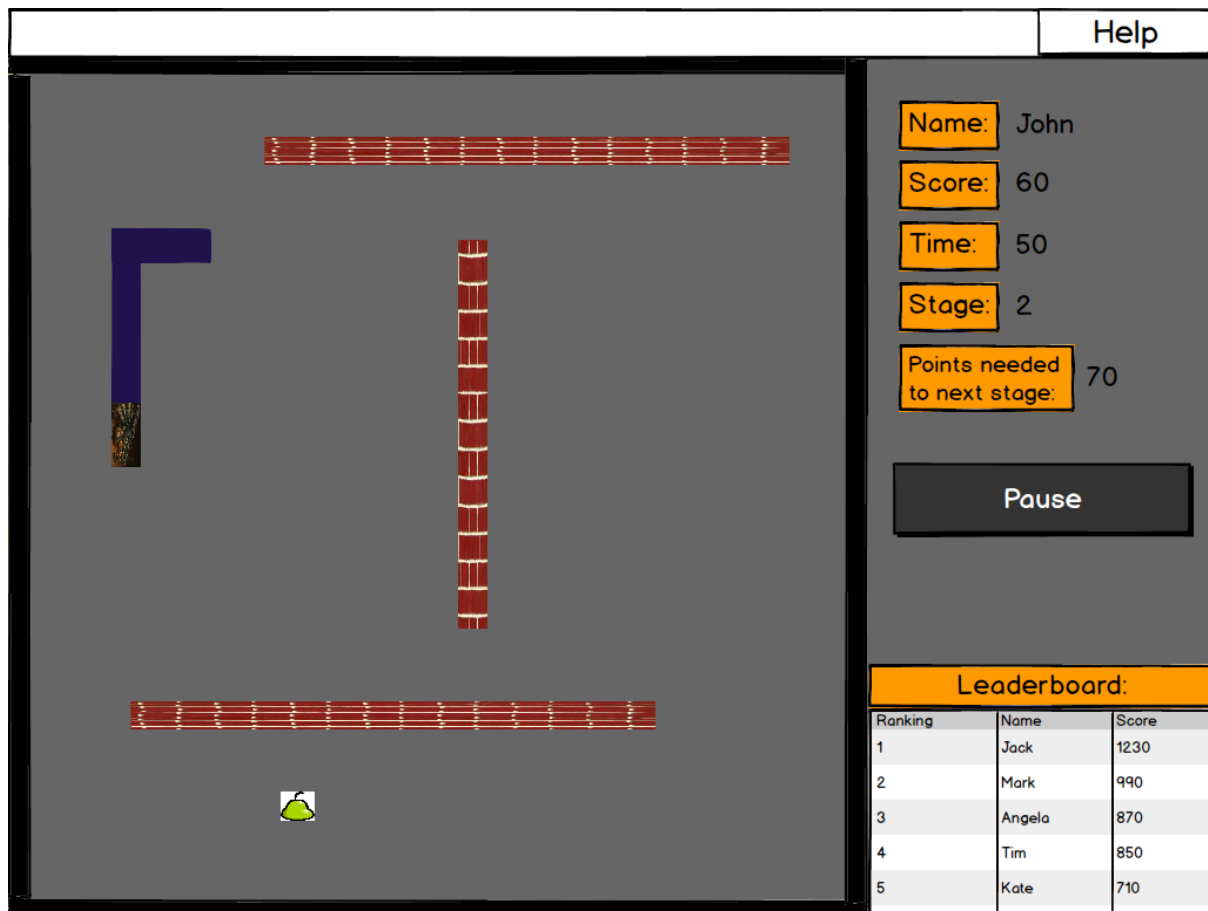


Figure 2.8: User Interface of blocks in the game

On this screen we can see that user passed to second stage and there are some blocks are appeared. Dragon must not touch these blocks if it wants to survive and pass to the next stages. Another type of animal can be seen in this visual.

## Further Levels, Fire and Enemies:

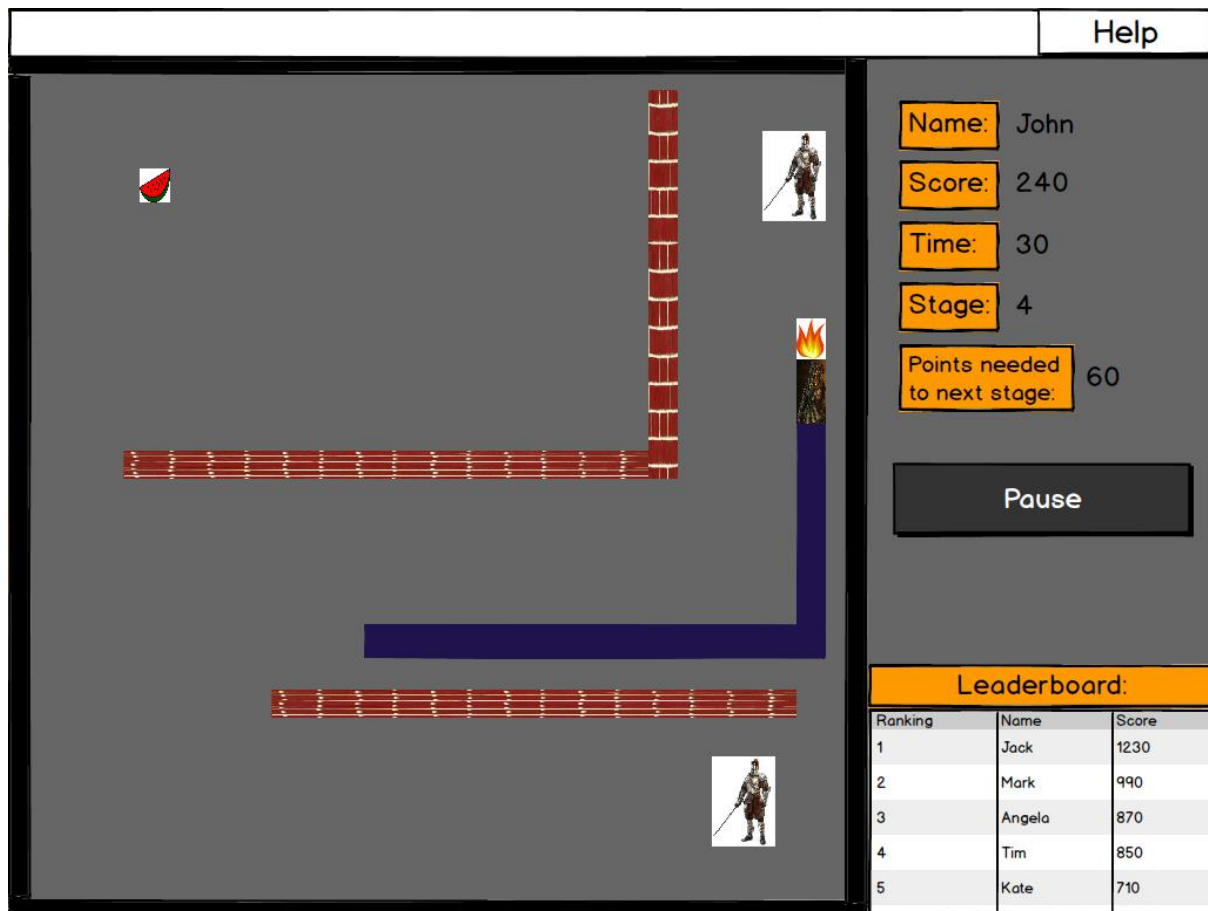


Figure 2.9: User Interface of enemies and fire

On this visual, it can be seen that player passed to a further level and game became more difficult than the first stages. Now dragon is longer, slides faster and there are more blocks appeared. As can be seen, there are enemies on the board. Our dragon can use its fire breath to destroy them and gain points for passing the next stages.

### **3. Analysis**

#### **3.1 Object Model**

##### **3.1.1 Domain Lexicon**

During the designing of a dragon valley game, more than one comprehensive domain had to be examined: gaming, game development, computer graphics game engines, user interface, content and logic. These examinations had been done both by doing research on existing systems and techniques, by researching on internet, or from our general knowledge. Most of result is known by gamers but all of them should be again written to this report.

#### **Game Engine:**

Game engine is core of our program. It never interacts with player but it will work behind of game infinitely. It checks all collusions, does calculations such as place of animals, blocks, enemies etc. Idea of this game engine is working autonomously at core of the game from game content, game logic, player.

**Board:** Board is a display of our game, it consists of grids. Everything thing is happening on this board visually. Game draws every object of game to board in order to show all events to user or player.

**Grid:** Grid is drawn on board in order to user can take all actions on it easily. We can see grid most of visual programs or games, in order to enable player to take action easily. Mostly, shape of grids are small squares.

**Draw:** Draw all objects that can be visualized to show user all objects on game. It provides user or player with graphical aspect of game.



**Real time:** It allows players to play in fair system. It means player can play for certain time interval and this interval is same for all players.

**X Width:** A graphical term, defines the X coordinate of a drawable object in the space.

**Y Length:** A graphical term, defines the Y coordinate of a drawable object in the space.

**Updates:** A game development term, update refers to applying all changes in the game logic to the game objects, usually dictated by the state of a game.

## **Game Control:**

Game control unit controls all game. Game control interacts with user or player and core of the game which is game engine directly. Hence, we can see effects of game control unit on both game engine and game logic.

**Main menu:** It is first interactive interface of game. Most of the games start with this state. It allows user to choose from menu what user wants to do such as new game, help, options etc.

**Options:** Options submenu can be chosen from main menu so user should adjust settings before game starts. This submenu enables user to adjust settings of game, choose difficulty level, or sound effects. Therefore, user can choose game-play difficulty as easy or hard depend on skills on this game.

**Pressed buttons:** pressed buttons allows to control game when user plays game. In this game arrow keys will control our game such that direction of dragon will

be chose by arrow keys. User will press one of the arrow keys where he/she wants to his dragon move on.

### **Game Logic:**

Game logic actually sum of game control, game engine, game content because without any logic our game cannot be played or cannot be run by the user. Game logic consists of how game should be played, which functions should be done on this game, and how to interact with user and core of program at the same time independently. After interacting with user and core of game, it decides what should game do whether continue to play or stop playing. Correct calculations also done in this stage and by result of these calculations it does some events. Of course, everything is happening by the obeying rules of game.

**Dragon:** Dragon is main actor on this game. Dragon refers to an object on this game which can fire to enemy or eat animals.

**Fire:** Fire is an object that refers to dragon that can kill dragon's enemies with fire. Dragon have limited fires to fire its enemies. Logic of fire like a bullet, it will be used for killing enemies of dragon.

**Enemy:** Enemies can be monster, people etc. that want to kill dragon. In this game enemies will try to kill dragon. However, dragon has to fight against to these enemies. Dragon should defense itself from these enemies by killing them.

**Game Stage:** Mainly most of games have many stages and in each of these stages user faces with lots of challenges. Every next stage always will be harder than current stage as well as dragon valley game. Dragon valley game also

provides users with challenging stages, so that latest stages of this game also will be hard to play. When result of playing on current stage will be success, player will move on to the other stage, otherwise player will return back to first stage.

**Miss:** Miss means unsuccessful attack to its enemy on this game. Unsuccessful attack brings you to out of fires or end of your fires. When fire does not meet with its target, you can create problem for yourself due to the fact that enemy can kill you.

**Hit points:** All of the games hold on players score to show that how good each player on certain stage or on certain game. Also on this game which is dragon valley, we trying to calculate points or scores of players. Dragon eats some animals and gains some points from these animals but when it eats bomb, dragon's points will be decreased.

Visual Paradigm Standard Edition(Bilkent Univ.)



**MainMenu:** The window that contains the options a player can choose. It is the first screen the player sees.

**Game:** The window that contains the GamePanel and the InfoPanel.

**InfoPanel:** This is panel in which the player can see the information about the game. It can be seen in the user interface section of the report. It basically displays the top three scores and gives the left time, stage number, collected points etc.

**GamePanel:** shows the game components such as the Dragon, Bricks, Door, Enemies etc. The class is the visualization of the game.

**Options:** The window which the user can use to change the difficulty and play with the sound volume.

**Help:** The window in which the user can learn how to play the game.

**Credits:** This window simply shows the developer information.

**LeaderBoard:** This window uses the DataBase class and pulls all the entries inside the DataBase in order to show the information to the player.

**DataBase:** This class is managed by the GameController and serves the purpose of saving player informations and scores to a DataBase. It allows simple operations such as read and writes but does not allow complex queries.

**Score:** a class the LeaderBoard and DataBase uses in order to keep the information of each individual player as an object.

**GameController:** This class is the brain of the game. It is the main controller. This class interprets all inputs from the user and adjusts the components accordingly. The class also checks for any collisions with the dragon, if the time is up, and if the sufficient points are collected in order to pass the stage.

**Notification:** This is a basic class that the GameController uses in order to display information such as the pause, stage number in the beginning of the stage, broken record etc.

**SoundManager:** Takes care of the sound effects in the game is controlled by the GameController.

**Stage:** is a class to manage all the game components. All of the model objects have their reference here. GameController uses this class in order not to deal with all of the game componenets.

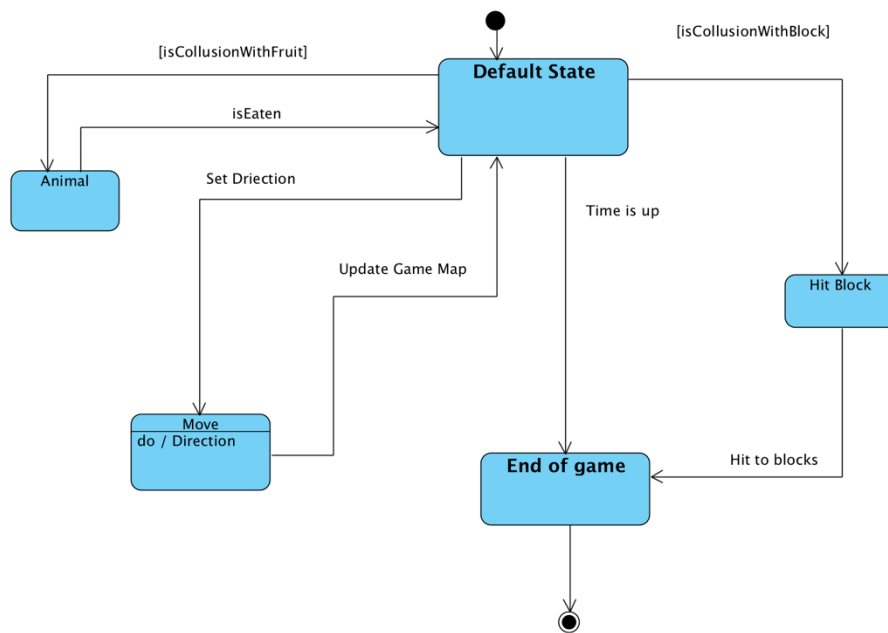
**Door, Enemies, Enemy, Board, Animals, Animal, Fire, Blocks, Block:** these are model classes they do not do any computation they exist to hold information of game components.

## **3.2 *Dynamic Model***

### **3.2.1 *State Chart***

#### **1. Game State Chart:**

Game starts from default state, so it means that playable environment for player is starting from default state. Game will load first stage to default state and default state will create game object. Initially, player will only interact with animals. Therefore, player will eat some of animals which were chosen before. Game controller always will check it can move or not when one of the arrow keys is pressed. If one of the arrow key is pressed such as right, left, up, down keys, game controller will change direction of dragon towards that arrow key pressed. For example, if dragon move to right side and player pressed “UP” key, dragon will change direction of itself to upward. However, if there is blocks and it will not move. It will go default state from move and if there occurs collusion between dragon and block, state will change to end of game. Secondly, it will always check for time. If time is up, state also will be changed to end of game. On the other hand, in some time period, if dragon cannot eat some amount of animal, or this time period will be expired, it means game is end.



**Figure 4.1: Game State Chart**

## 2. Character State Chart

Our character state is dragon that we will play this game as dragon. In this game, dragon will move until he will hit to blocks, or when he obtain appropriate score to pass stage. When he gets appropriate score, door will open and dragon can move from this stage to another stage. More general statement is “move” in this game. Dragon will eat animals, fire to enemies, try to don’t touch walls or blocks, so dragon will get points from eating animals and killing enemies. In this way, dragon will become larger and it can open door in order to go next stage. When dragon eat animal it will return our main state which is “move” state. When dragon fires to enemy and kills enemy it will return to again move state. “Move” state always checks door is opened or closed. It will continue check whether door opened or closed until door opens. In order to check door is opened, it will go to “door” state. When it will get that door is opened, stage or game ends.



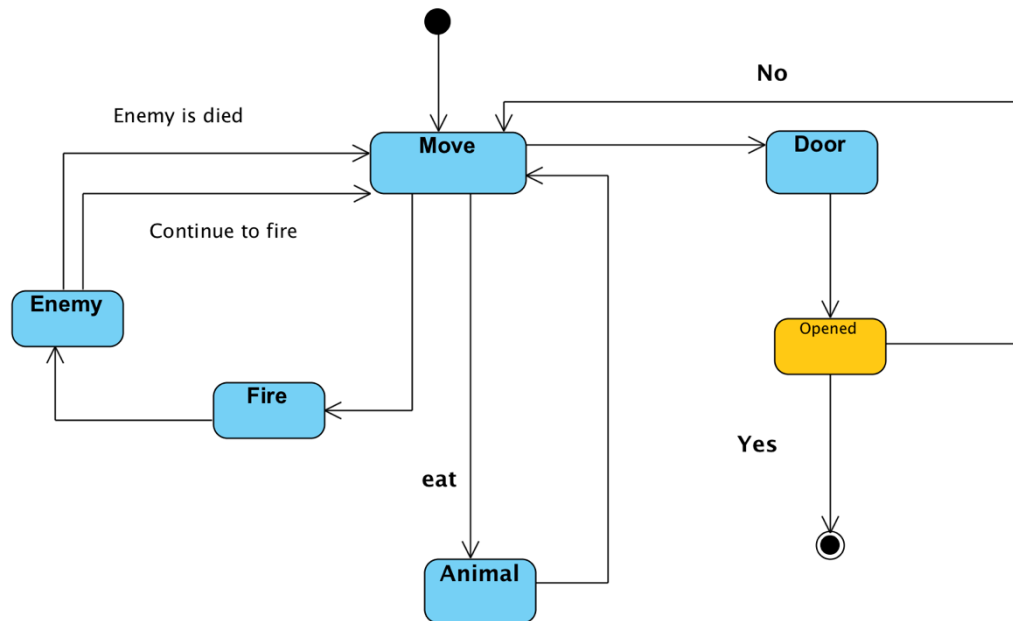


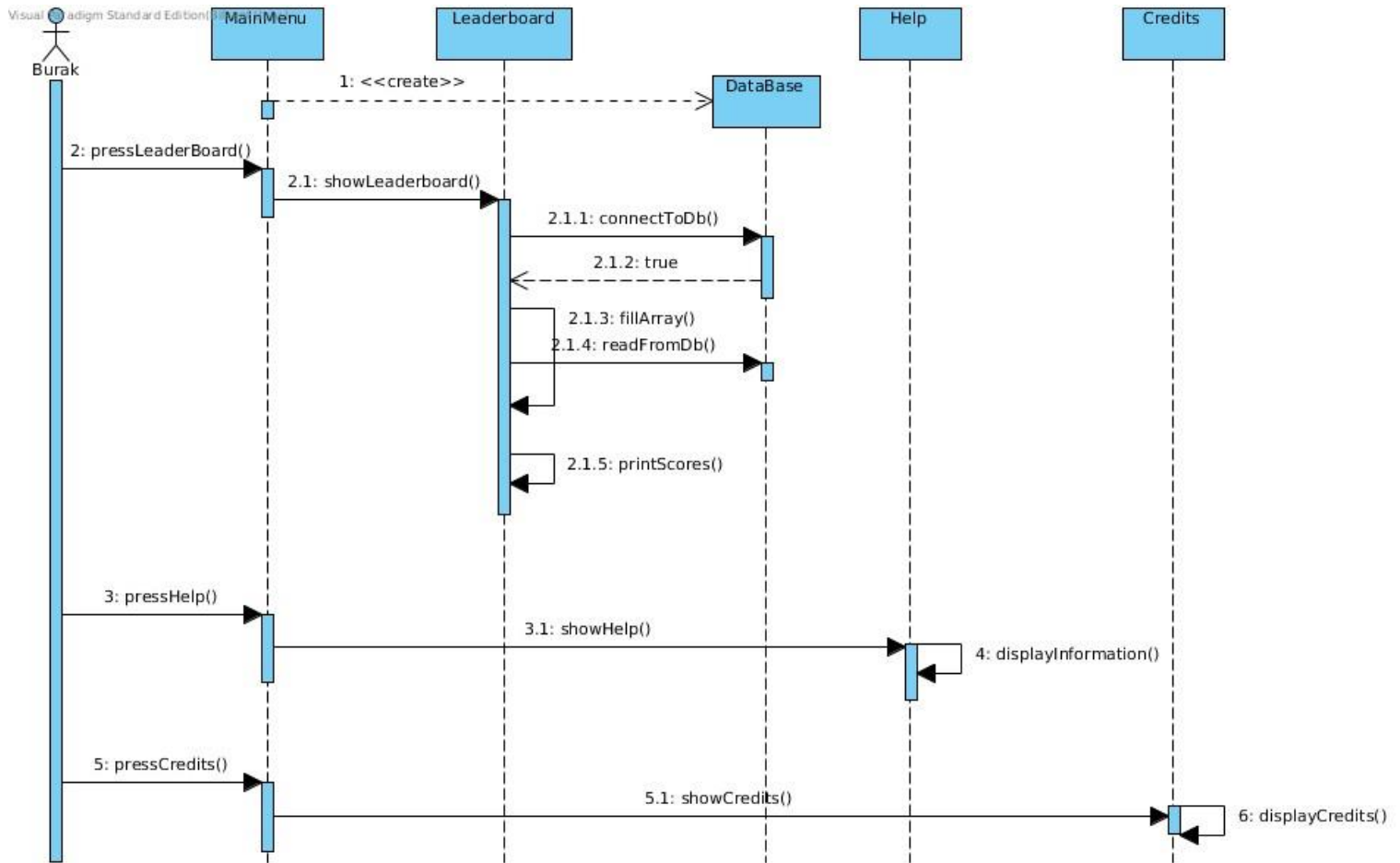
Figure 4.2: Character State Chart

### 3.2.2 Sequence Diagrams

Scenario name: Main menu operations.

Scenario: Burak starts the game and sees the main menu. He first takes a look at the leaderboard option then he continues by viewing help and credits.

The purpose is to show how the main menu works. Note that we show the new game option in the next sequence diagram.



**Figure 5.1: Main Menu Operation Sequence Diagram**

Description:

After the game is started by the user the main menu pops up. The DataBase class is instantly instantiated without any user request. This is done to make the DataBase ready for the GameController class or the LeaderBoard class. When the user (Burak) clicks on the leaderboard option a new window pops up showing the previous high scored from other players as we can see in the user interface section. The selection makes the LeaderBoard connect to the DataBase and read from it until it fills an array with all the points. Next the values are displayed.

The user proceeds with selecting help which simply calls the help window and displays the information needed to play the game. Finally the user presses Credits which is very similar to the help screen but instead the displayed text is the names of the developers.

Scenario name: Basic Gameplay

Scenario: Omer starts the game and selects new game. He is greeted with the game and starts playing the game until he wins it or he fails it the game is over. This scenario shows how the game will execute when someone plays it.

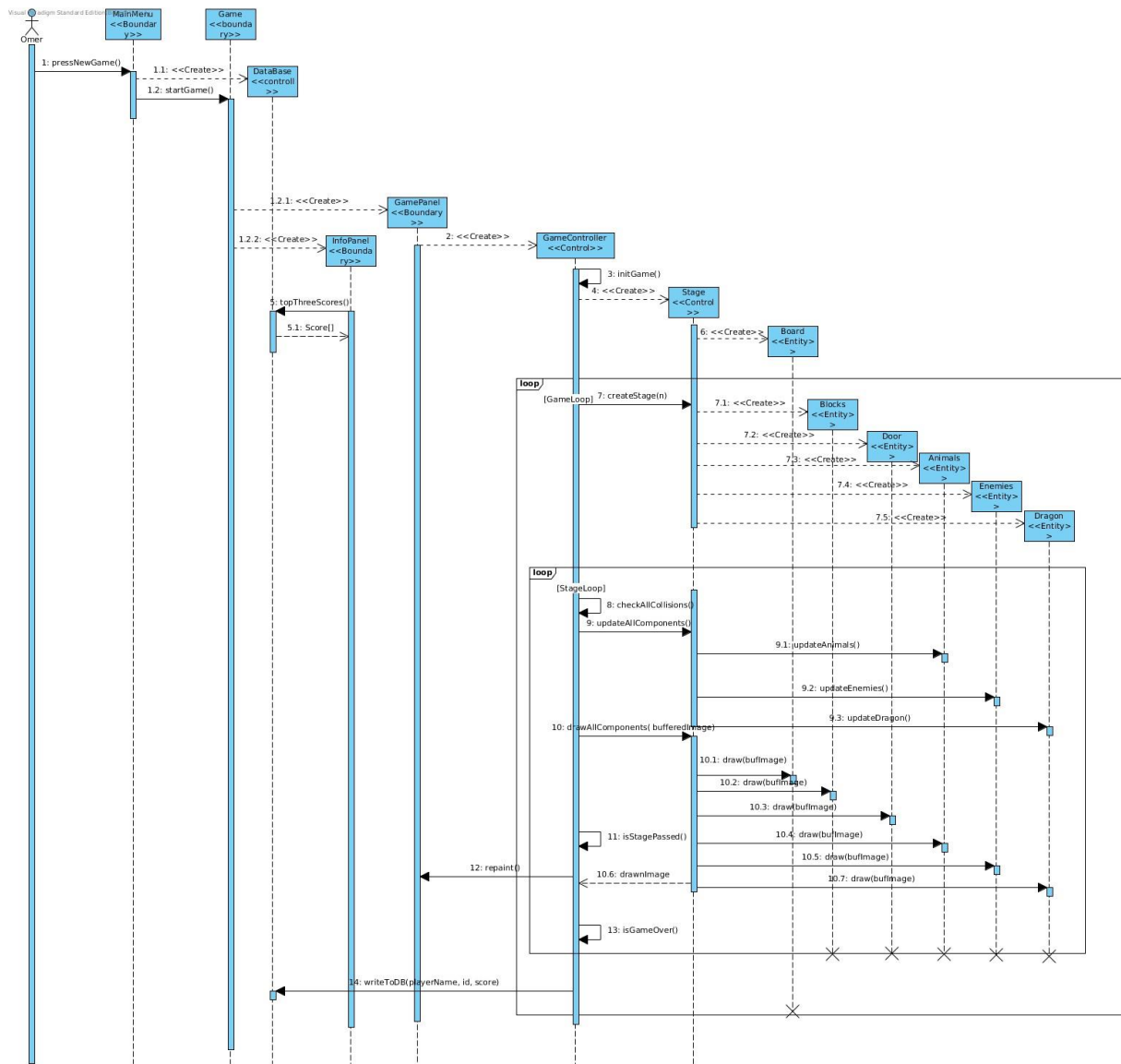


Figure 5.2: Basic Gameplay Sequence Diagram

### Description:

Once the user (Omer) selects new game option the program shows the game window. This window instantiates two panels: InfoPanel and GamePanel which are used to show information about the game and the game screen respectively. The InfoPanel uses the DataBase object the MainMenu created and gets the top three scores to display them. The GamePanel starts the game by instantiating a GameController object. The GameController class is the main controller in the game and all of the logic lies here.

The GameController creates a Stage. The Stage class then continues to create all the elements the game needs in order to be playable. The stage does this by the invocation of createStage(n). The Stage creates the stage requested by creating an instance of each Dragon, Door, Blocks, Enemies, Animals and Board. These are created inside the game loop because when we progress and proceed to the next stage these objects will have to be reinitialized. Board is outside the loop since it does not need to change between stages.

The GameController continues by checking for any collisions between elements. Since the stage has all the game elements it takes the references and compares for collisions. The stage loop ends if the stage is passed( either a win or a loose.) the game loop ends if the player loses or passes all stages successfully.

If we take a closer look at the diagram the life line of MainMenu is only active for a little time. This is because once the play game button is clicked the MainMenu screen will disappear and we won't need it anymore. We see that the boundary class Game, InfoPanel, GamePanel are all active until the game ends, this is because they are view elements which will help the user to interact with the game.

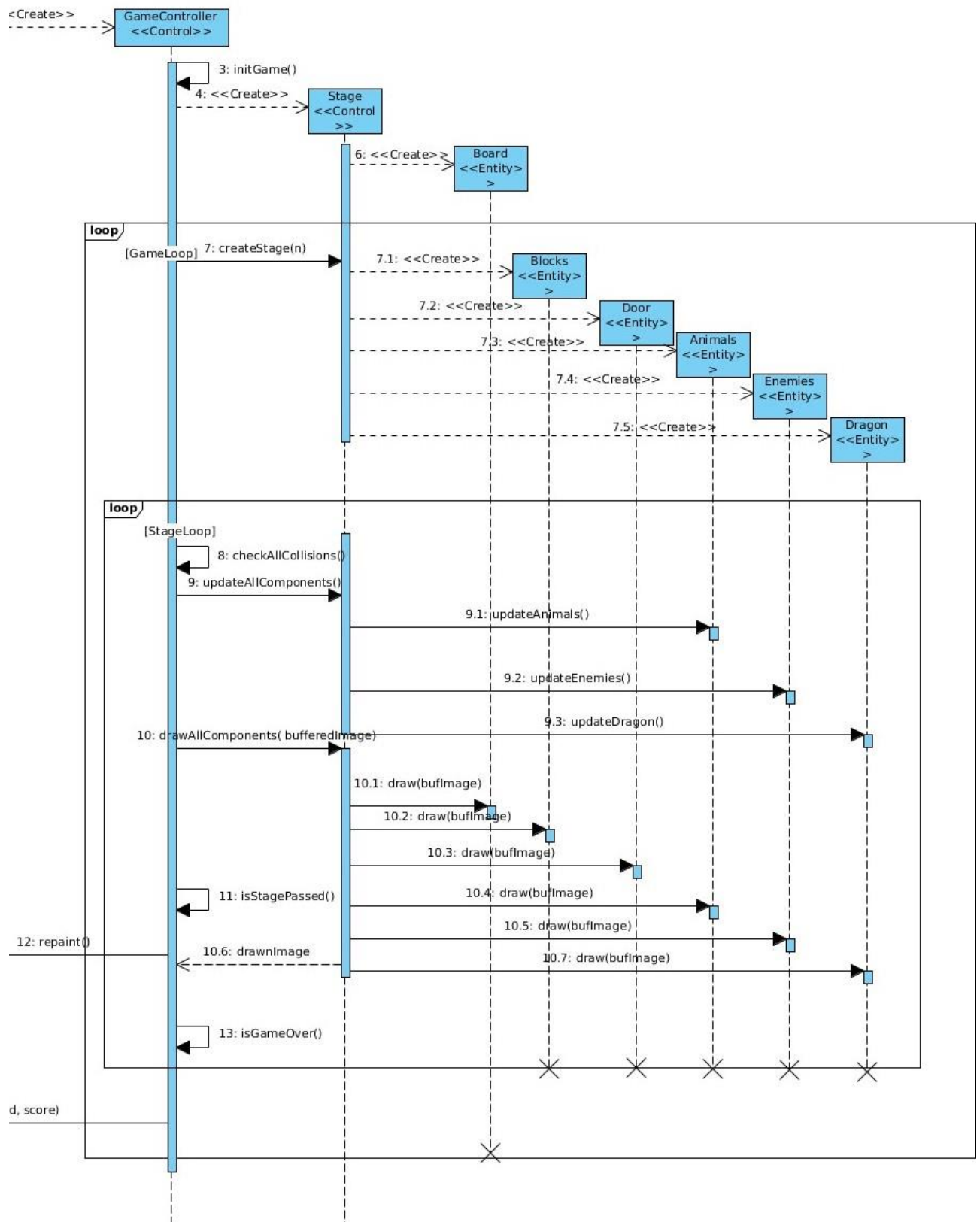


Figure 5.3: Main Point of the Sequence Diagram

Now let's take a closer look at the main point of the sequence diagram. This is the main game loops. This is where all the logic of the game is processed and sent to the screen. We see that there are two controllers one is Stage and the other one is GameController. The GameController class uses the stage in order to control the entity elements. GameController is the main controller object where the Stage is just a step to manage the model objects. The model objects are just used to get the information of the game and they are also used to get the visual information of the models.

The outer loop in this diagram shows us the GameLoop. This loop will only work when the inner loop has finished. When the inner loop finishes we understand that a stage has been either passed or failed. The GameLoop reacts by destroying each instance of game component to reset the game. Then the GameLoop re-instantiates all of the game model in order to set up a new stage. If the inner loop is terminated because of a game over then the GameLoop will exit as well. The inner loop is called the StageLoop since it represents a stage of the game. This loop will check for collisions using the user input but the input is not shown in the diagram to avoid confusion. After the collision checking is finished the GameController will update all of the necessary elements according to the user input. The user input is not shown again as before. After all the game elements are updated the controller gets the information about the visuals of the game elements by calling the `drawImage()` function. This function does not draw anything but it gathers the graphics of the game elements. The painting is later done by calling `repaint` of the `GamePanel` in order to satisfy the MVC architecture. We see that our controllers do exactly what they are supposed to do, they handle the user input which is received from the boundary objects, they update the models and finally they reflect the results on the screen

## 4. Design

### 4.1 Design Goals

**Well-defined interfaces:** Our program's interfaces are well defined and easy to understand. In our system we generally tried to keep it simple to make it a user-friendly program. So a user can easily understand and use the interface of the program if he/she never used a system like ours before.

**Ease of learning:** Our game is easy to understand and play. User can reach the instructions menu from the main menu and easily understand how to play the game. Also, there is a pause button while playing the game and user can pause the game and click the help button to go to this instructions menu while playing the game.

**Good documentation:** Good documentation is very important on the design stage and also on the other stages. Report must be detailed and easy to understand to create better conditions for the developer and the client of the program.

**Adaptability:** We used Java language in our project to make it run on most platforms. Because Java is supported on most platforms, we choose this language to make our program run in most of platforms.

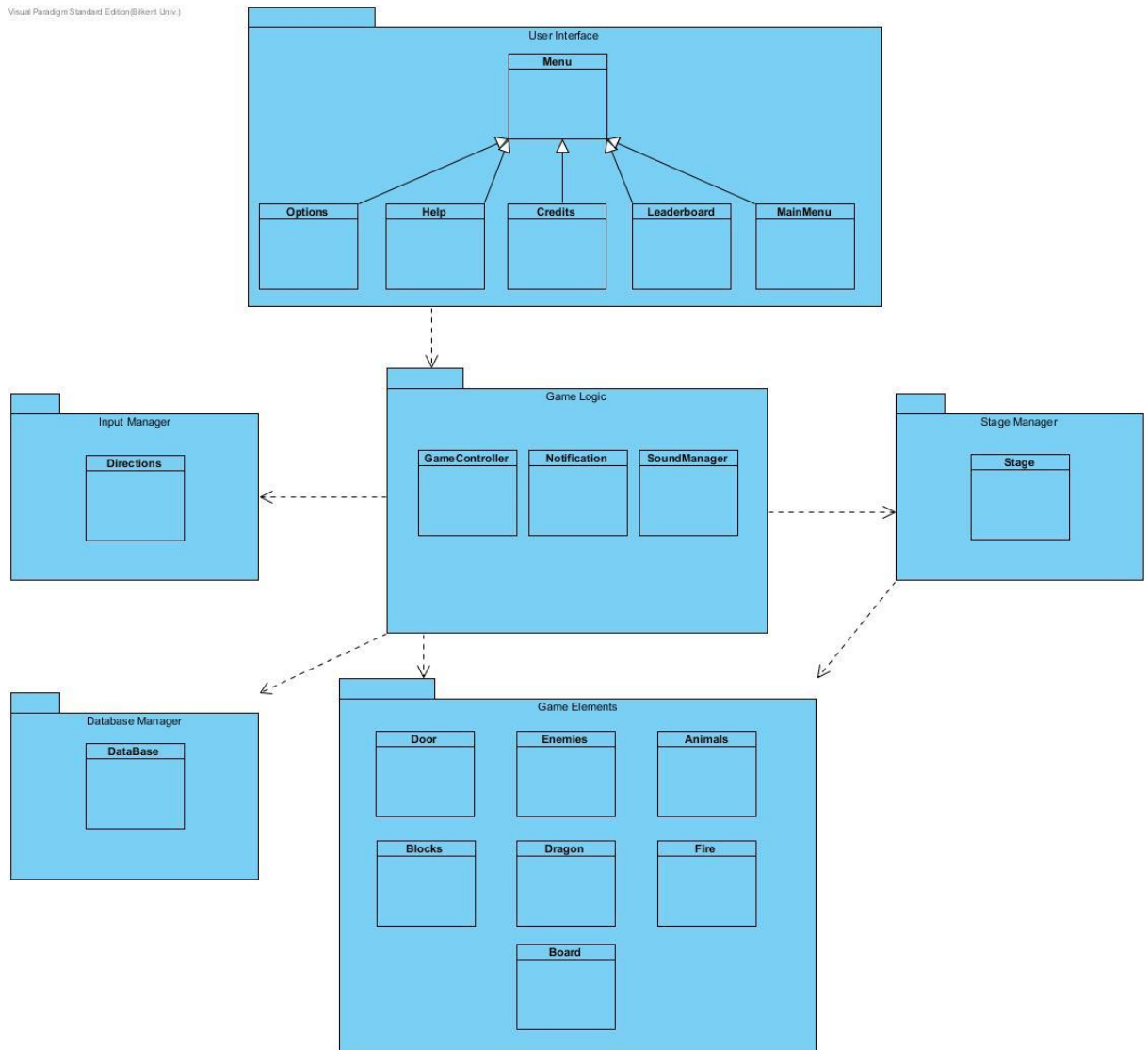
#### Trade Offs:

**Adaptability vs. Performance:** We used Java in our project to make the program run on most platforms. Our goal is to make the program reach lots of people but using Java has some downsides like worse performance when compared to other languages like C++.

**Ease of Learning vs. Functionality:** We made the program user-friendly. Our main goal is to make people understand and play the game easily. For this we kept program's functionality on a basic level.



## 4.2 Sub-System Decomposition



**Figure 6.1: Subsystem Decomposition**

Like it must be done in all projects, we divided our system into subsystems to make the project organized. Our system composed of 6 subsystems which are User Interface, Input Manager, Game Logic, Stage Manager, Database Manager and Game Elements. On the top layer User Interface can be seen. This subsystem includes our menus in the program and it calls Game Logic as we see it on the second layer. Game Logic includes our GameController class which actually does most of the things in our program as can be seen in our class diagram (Figure 3.1). Game Logic calls the other 4 subsystems. Input manager includes Directions class which includes attributes of our dragon's movement. Stage manager includes Stage class which organizes the stages of the game also, it calls

Game Elements subsystem. Database Manager is called by Game Logic and it includes DataBase class which holds the users' information like name and score. Finally, Game Elements subsystem includes 7 classes which are the elements can seen on the game board –including game board itself- while playing. It is also called by Game Logic subsystem. As can seen, Game Logic calls 4 subsystems and it is the heart of our system.

### ***4.3 Architectural Patterns***

Our system is decomposed into three main layers. These are the user interface layer, the controller layer, and the game model layer. The user interface layer has the responsibility to interact with the user and get inputs from him/her. Then the UI layer gives these information to the controller layer. The controller layer does the necessary computations with these inputs using the services the game model layer provides. The controller layer gets information about some objects according to the input and makes changes these informations according to the game rules. The game model layer is mainly used to store the information of the game. The information this layer holds will vary widely from game to game because it depends heavily on the user. No computation is done on this layer, nonetheless it is absolutely essential to the system for it to work.

We can say that there is a “calls” relation between UI layer and the controller layer. The UI will call some of the controller components in order to notify it and this way the controller will be activated with the initiation of the UI. The game model and the controller have a “provides services for” relation. For instance the controller will need some information in order to do its computations. In this case controller will use the services of game model and obtain these information.

Since in our architecture there is almost no direct connection between the UI and the game model layer we can say that our program is an instance of closed architecture (opaque layering).

Our architectural style as you may have noticed is MVC ( model, view, controller.). The User interface layer is the view, the controller layer is the controller and the game model layer is the model. As mentioned before the UI will interact with the user and get inputs, the controller will handle all the core computations of the game and the model will hold the information of that instance of the game.

#### ***4.4 Hardware/Software Mapping***

The game will be developed in java thus it will need a JRE to run on. To interact with the game the player will need a keyboard and a monitor. A mouse is optional but will definitely increase the user experience. The CPU, and memory of the hardware does not matter anything that can run a Java virtual machine will be just alright. The game needs non volatile memory in order to save the high scores locally on the computer a couple of megabytes should be enough for the whole game.

#### ***4.5 Adressing Key Concerns***

##### ***4.5.1 Persistent Data Managment***

Some files and scores of players will be saved on our computer in order to keep data. The game keeps names and scores of players in database in order to show them at leaderboard and we can compare scores of players. It is easy to read from database scores of players and put it to high scores. Therefore, we can maintain and manage users or players on this database. There are some images and music files, which will be saved on our computer to

have interesting and better experience on our game. These things will make this game funny and it will give nice appearance so player will not get bored. At each level of the game, there are different images, music so for each level it should be saved on a persistent storage in order to use always.

#### ***4.5.2 Access Control and Security***

The only actor that is interacting with the system is the gamer and therefore, an access control strategy is not considered as a necessity in the dragon valley system. One may think that an administrator for the dragon valley game could be created for the alterations of the game speed, rules and etc. but doing so, separating of use cases will be needed but user should also have access to these options in our decision and it is concluded that there is no need for an administrator. To look from the security perspective, it is thought to have a login page in the system (username and password) but then observations showed that there is no need to have safety measurements like this since there is no special data for each player. Finally, the Dragon Valley system is open to the use of all gamers thus no further access control explanations are needed. The user may not have access to all methods directly but the gamer has sufficient access to several methods that invoke other methods accordingly.

#### ***4.5.3 Global Software Control***

Since the Dragon Valley system's architectural style is Model-View-Controller (MVC), the global control flow of the system is event-driven. To be more specific, the system waits for an input from the user in order to update its each view. Input Manager Subsystem listens the keyboard event listeners and change the directions of the dragon accordingly by sending the input to Game Elements Subsystem which is the model part of the Dragon Valley System architecture. In Game Elements Subsystem decides whether the views of each model

should be updated or not. The main controller of the system is located in the Game Logic Subsystem and the Game Controller class controls the flow of the whole data through the game. Since there is no too many complex models in the game it is decided to have only one controller that is centralized under the Game Logic subsystem.

#### ***4.5.4 Boundary Conditions***

**Initialization:** At the beginning, the allocated memory is used to load the program and the execution of the code starts. GPU handles the user interface components so GPU also have some data. After these, game main menu is presented to the user which consists of several useful buttons such as Help,Exit,Credits. If the player clicks on Play Game Button, the components of the game are initialized such as the dragon, animals, blocks, sub menu components(exit game, restart game buttons) and so on.

**Termination:** The whole termination occurs when user click Exit button either on the main menu screen or on the submenu of the gameplay screen. The current score of the user before termination is written to the database.

**Failure:** If a failure occurs during initialization, the system stops and closes itself immediately. During the game, if a problem occurs the game stops and since the current score of the user will be saved into the database there will be no critical issue to overcome. While termination, if a failure happens system will shut down itself and if the user opens the game again the system work from the beginning(main menu).

## **5. Object Design**

### **5.1 Pattern Applications**

#### **Singleton Pattern:**

Singleton pattern is usually used to manage the number of instances a class has. If multiple instances will create confusion then we do not allow an instance to be generated and we provide one instance that everyone can access. In our Database class we can observe a singleton pattern. We do not need to have multiple instances of the Database class so we have just one. This object is public and static so that any other class may access it if it is necessary.

#### **Façade Pattern:**

Façade pattern is used in order to reduce the complexity of a subsystem and provide a usable interface for it. In our project we used the Facade pattern to manage the model objects and make the job of the controller easier. GameController class is our main controller and the Stage class acts as an interface to access the model classes. For instance if the GameController wants to change the location of the Dragon it will need to access the Dragon model. To do this it will tell a Stage instance to do this for it. Using the Stage class we have a greatly simplified controller class.

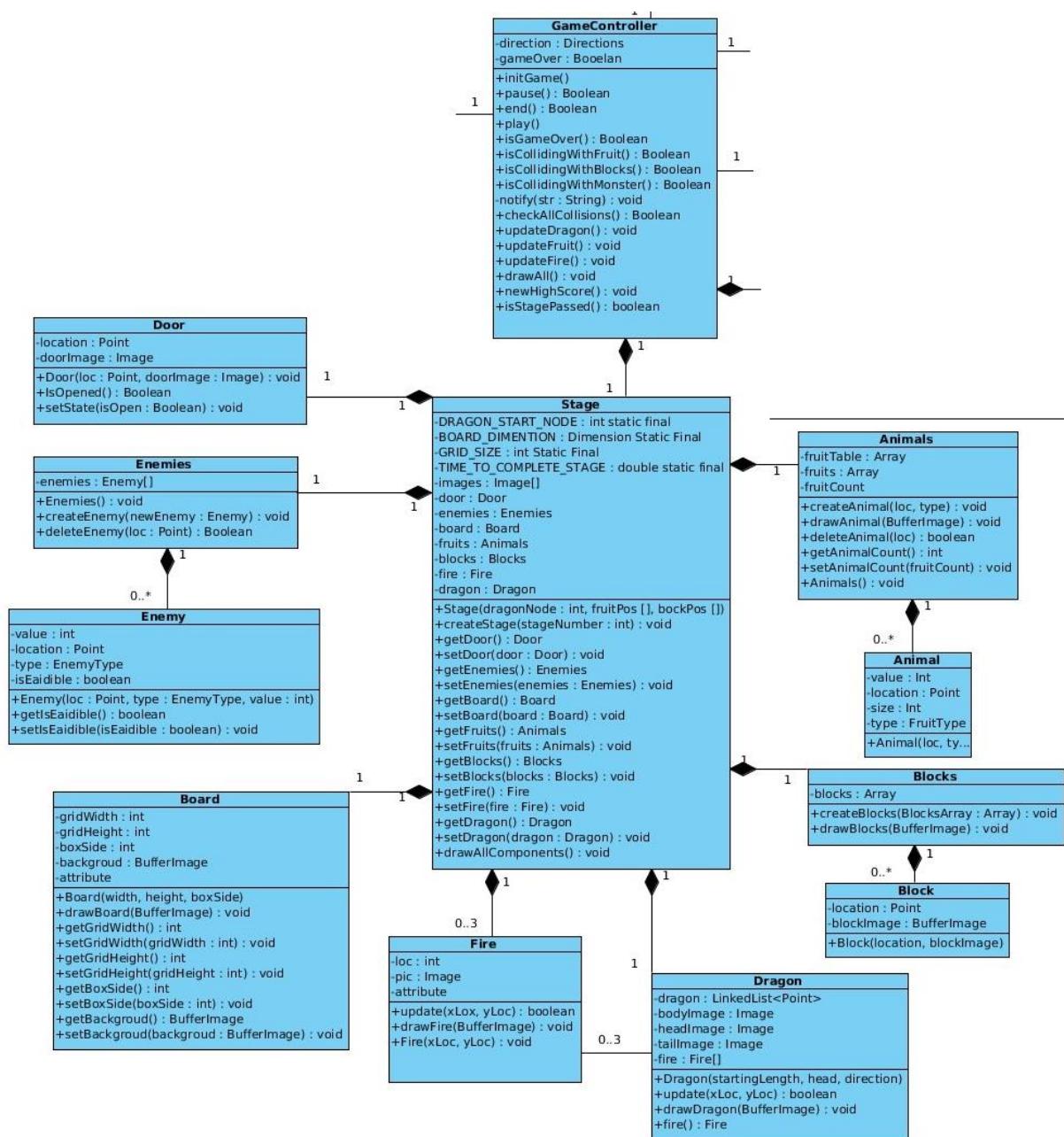


Figure 7.1: Façade Pattern

As we can see in our UML class diagram the GameController class does not have any connection with the model objects. The connections that it has are only other services. The Stage class has all of the references to the model objects and has the necessary methods in order to provide an interface to control these objects.

**Observer Pattern:**

The Architectural Style of the Project is chosen as Model-View-Controller(MVC), therefore, Observer pattern is one of the crucial patterns of the Dragon Valley. The observer pattern works in a way that when the score of the player is changed the database's update() method updates Leaderboard, InfoPanel which are observers. Database does not know how InfoPanel and Leaderboard updates itself and that's the main goal of observer pattern.

**Adapter Pattern:**

In software engineering, the adapter pattern is a software design pattern that allows the interface of an existing class to be used from another interface. It is often used to make existing classes work with others without modifying their source code. An adapter helps two incompatible interfaces to work together. In our program, adapter pattern enables two incompatible classes to work together. There classes are Database and Mongodb. MongoDB is a cross-platform document-oriented database. In order connect to database and to use it, we need adapter class, so it is Database class in our case. Database class lets Mongodb and LeaderBoard class to work together. So, LeaderBoard can connect to database using Database class's methods. It can manage database, add, remove modify in database easily. Therefore, Adapter pattern provides us with easy access to other objects that monogdb without modifying our class.



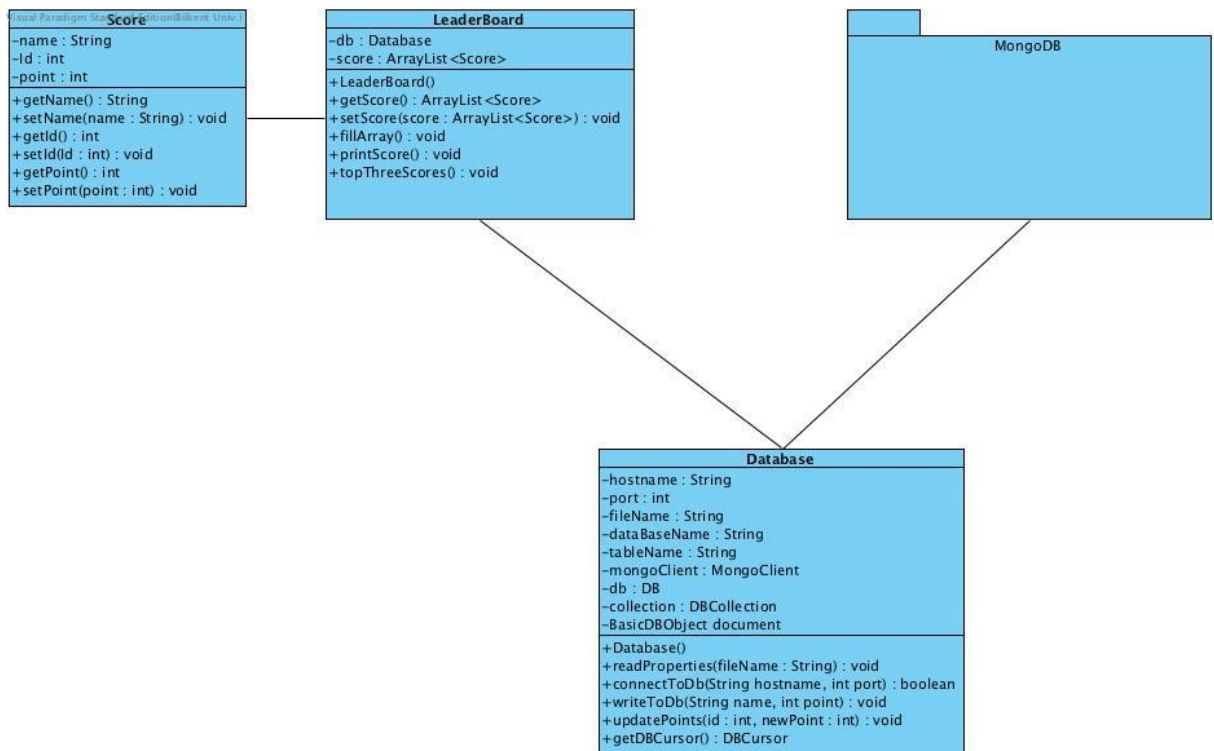


Figure 7.2: Adapter Pattern

## 5.2 Class Interfaces

### Game:

Game inherits from the java swing libraries JFrame. This class is only used as a boundary since it is a view class. The class acts as container for two panels: InfoPanel, GamePanel. These two panels show all the information the player needs to see in order to play the game flawlessly. The Game class will only be used if the user decides to select the play game from the menu. This class does not contain much as it only has two JPanel s in it and has code in order to set the sizes of these JPanels. The class also provides a public constructor.

### GamePanel:

The GamePanel inherits from the JPanel class. This class used to show the user the game screen(the thing you look at while playing the game). This class is mainly used by the GameController since it is where the contents of the game are displayed. It has one major method which is paint(

image: BufferedImage): void. This method is used by GameController in order to print the game to the screen.

### GameController:



**Figure 8.1: GameController Class**

GameController class is the most important class in the program. It is the main controller of the whole game. GameController implements the runnable interface in order to make create a thread to run the game in. This class has 3 major private properties: db:DataBase, notify:Notification, stage:Stage. These properties are used in order to control the game. The most important methods are run() and move() the run method is called continuously as it the the method called in the Thread and the run method calls the move method in order to make the calculations for the Dragon. The move method also check for all collisions between the game elements.

## Stage:

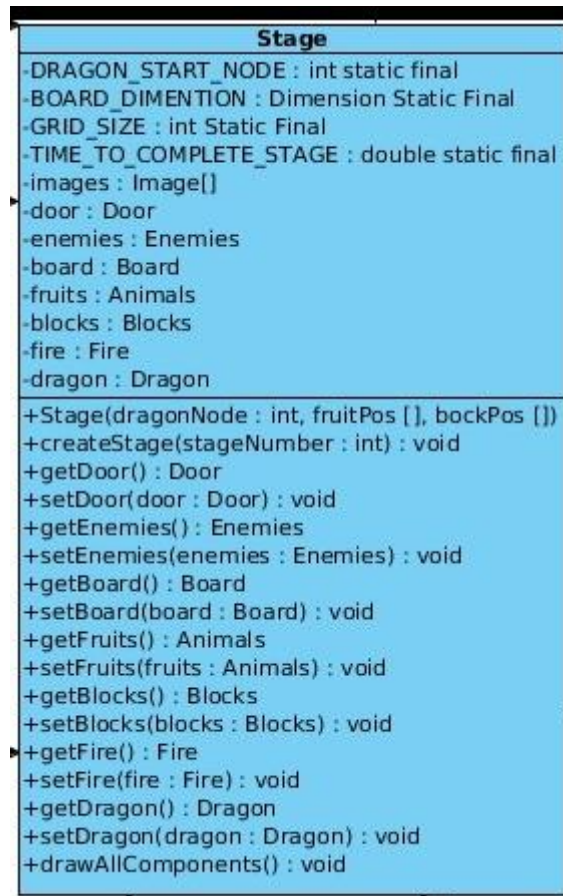
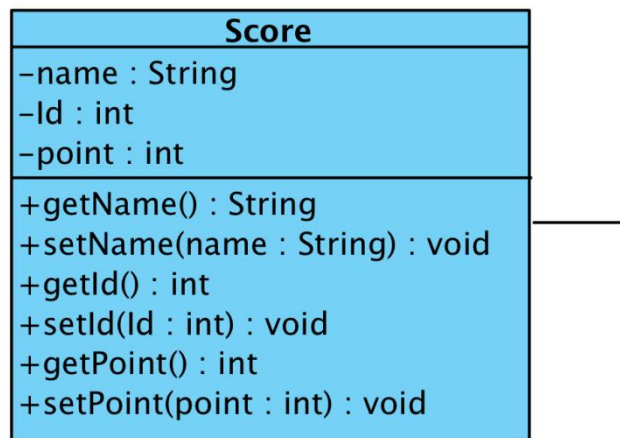


Figure 8.2: Stage Class

Stage is the class used in order to implement the facade pattern. It has many private properties in it. The important properties are all references to model objects such as Dragon, Blocks, Enemies etc. The methods of the stage are mostly getters and setters in order to provide the GameController references to the model objects if needed. The most important methods Stage has are createStage() and drawAllComponents(). The createStage() method is used in order to reset the stage and start a brand new one this method is essentially the method to manage the model objects. The drawAllComponents() method is used by the controller in order to get the visual information from the Stage and print these on the screen.

**Score:**



**Figure 8.3: Score Class**

Score Class is used to take name of user, and point of his/her after that it will write these data to database. So, Score class has 3 private property such that name(String), id(int), point(int). It has just one default constructor. However there are 6 public methods. These 6 methods are setter and getters of private variables. These methods used for easy get value or set it to score object. Due to the fact that score variables are private, we can't access to them directly so we have to use these methods. Name of these methods are `getName()`, `setName(String name)`, `getId()`, `setId(int id)`, `getPoint()`, `setPoint(int point)`.

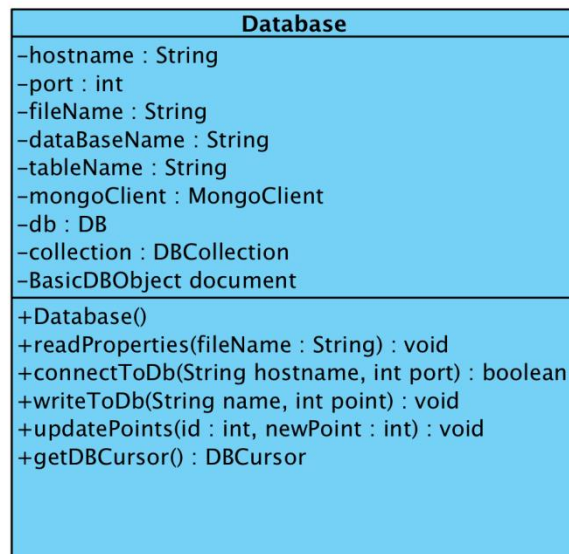
## LeaderBoard:

LeaderBoard
-db : Database -score : ArrayList<Score>
+LeaderBoard() +getScore() : ArrayList<Score> +setScore(score : ArrayList<Score>) : void +fillArray() : void +printScore() : void +topThreeScores() : void

**Figure 8.4: LeaderBoard Class**

Leaderboard class has two private variables or properties. They are db and type is Database, second one is array list so, score and type is ArrayList<Score>. The array list of score will store all score objects in this list. LeaderBoard class has one default constructor that will just call fillArray() method. Lets talk about our methods. There are 6 methods that will help us to get data and put it leaderboard. User can see all his/her opponents' score and name. It will get all players name, score and will add all of them to array list as score object. However, it will just show 10 best players score at leaderboard panel. First method is getScore(), it will return score Arraylist which stores all of the database object or score objects in order to other classes can also use these data. Second important method is fillArray() which takes all database objects and maps it to array after that it sets name, id and point by using score class's set methods. When it ready it returns Score as object and finally it writes this object to score array list. printScores(), and topThreeScores methods print result of score to panel. Finally, readFromDb() method is able to read all data from database.

## Database:



**Figure 8.5: Database Class**

Database class is like an adapter. It enables our program to connect mongodb which is a cross-platform document-oriented database and create database if it is not exists, and it also create table or collections of course if they are not exist. Main job of database class to enable other classes to connect mongodb that is our database. With the help of this class other class can reach all data in the database. Database class has 9 private property, 3 of them is used for to connect Mongodb (database) and else is used for create database or reach our data in database. Names of these private variables are hostname (String), port(Integer), filename(String), dbName(String), mongoClient(MongoClient), db(DB), collection (DBCollection), document (BasicDBObject). Database class has 1 constructor. Actually, most of the processes are going in this constructor. Therefore, firstly, it reads all properties from properties file. To read these data there is readProperties(String fileName) method which will read all properties and initialize them to local variables in order to use them in our program. When all variables are initialized, it calls connectToDB(String hostname, int port) method to connect mongodb(database). If this method return true, it starts to create database and tables, if it not exists, otherwise it will not create and will continue to run other lines of code. If it can't connect to mentioned

database, it will try for several times to connect. At the end if it can't connect to there it will show error message. When our database class is connected to mongodb, so we are ready to write or read to/from there. In order to write to database, it has writeToDb(String name, int point) method that will write user name and his/her score to application's database. We have one more method that updatePoint (int id , int newPoint). It can update point or score of current player.

### **InfoPanel:**

InfoPanel class uses methods of Leaderbord. It enables us to put top three players name and score to gameplay GUI, so, at the right side we can see top 3 players name and score while playing this game. In addition, it will print users own score and his/ her name.

### **Help:**

Help class includes all instructions about the game. Player can read all instructions such that play pause/button, which case to move or where should put his name etc. Help class will be used just for getting information about game.

### **Animal:**

Animal Class represent the animal feature of the game which is the dragon's main food. The animal class has 3 private properties that are value(int), location(Point), size(int). The value is the score point of the animal. Since there can be different animals in the Dragon Valley, each type of animal has different values that increments the score by the specific value. The Animal must also have a Point property to get a place in the board in which the game is played. An animal can contain more than one box in the game screen therefore a private size property is declared. The Animal Class has no

default constructor but it has a constructor with its all parameters given as arguments. Additionally, getter and setter methods are also provided by the Animal Class.

### **Animals:**

Animals class is written for management purposes of the Animal Class. The class declares different types of animals in order to supply different foods with different scores for the dragon. Shark, Wolf, Bear, Kingkong (all of them private final static int) are the animals of the game with different points. A public Animals Table property(int[]) is also initialized in order to access from outer classes. The main purpose of the animal class is to create an array to draw each animal on the game screen. Therefore an animals property(ArrayList<Animal>) is declared. A default constructor is supplied to initialize animals arraylist. The creation of an animal defined in void createAnimal() method in order to add an animal to animals arraylist. Moreover, if a deletion is seen necessary (meaning that an animal is eaten by the dragon or game is over), a boolean deleteAnimal(Point location) method is also defined. It simply calls each animal from the animals arraylist and removes the animal from its location. The Animals class also have void drawAnimals(BufferedImage image) method to draw each animal to the board in the arraylist. Additionally, getter and setter methods are also provided by the Animals Class.

### **Enemy:**

Enemy Class represent the enemy feature of the game which is the dragon's main competitor, when the dragon collides with an enemy it loses point and even dies. The Enemy class has 3 private properties that are value(int), location(Point). The value is the score point of the enemy. Since there can be different enemies in the Dragon Valley, each type of enemy has different values that decrements the score by the specific value. The Enemy must also have a Point property to get a place in the board in which the game is played. The Enemy Class has no default constructor but it has a



constructor with its all parameters given as arguments. Additionally, getter and setter methods are also provided by the Enemy Class.

### **Enemies:**

Enemies class is written for management purposes of the Enemy Class. The class declares different types of enemies in order to attack the dragon. Khaleese, Robb Stark, Lord Stark, Tywin Lannister (all of them private final static int) are the enemies of the game with different points. A public Enemy Table property(int[]) is also initialized in order to access from outer classes. The main purpose of the enemy class is to create an array to draw each enemy on the game screen. Therefore an enemies property(ArrayList<Enemy>) is declared. A default constructor is supplied to enemies arraylist. The creation of an enemy defined in void createEnemy1() method in order to add an enemy to enemies arraylist. Moreover, if a deletion is seen necessary (meaning that a collision with an enemy occurred or game is over), a boolean deleteEnemy(Point location) method is also defined. It simply calls each enemy from the enemies arraylist and removes the enemy from its location. The Enemies class also have void drawEnemies(BufferedImage image) method to draw each enemy to the board in the arraylist. Additionally, getter and setter methods are also provided by the Enemies Class.

### **Block:**

Block Class represent the Block feature of the game which is the dragon's main obstacle. The block class has 2 private properties that are location(Point), blockImage(BufferedImage). There is no different type of blocks therefore a value property is not seen necessary. A block has an image to see the visual perception of itself. The Block must also have a Point property to get a place in the board in which the game is played. The Block Class has no default constructor but it has a constructor with its all parameters given as arguments. Additionally, getter and setter methods are also provided by the Block Class.

**Blocks:**

Blocks class is written for management purposes of the Block Class. The class declares same types of blocks in order to create blocks as obstacles for Dragon. The main purpose of the blocks class is to create an array to draw each block on the game screen. Therefore a block property(`ArrayList<Block>`) is declared. A default constructor is supplied to initialize blocks arraylist. The creation of an block defined in `void createBlockl()` method in order to add a block to blocks arraylist. Moreover, if a deletion is seen necessary(meaning that the dragon is hit to a block), a boolean `deleteBlock(Point location)` method is also defined. It simply calls each block from the blocks arraylist and removes the block from its location. The Blocks class also have `void drawBlocks(BufferedImage image)` method to draw each block to the board in the arraylist. Additionally, getter and setter methods are also provided by the Blocks Class.

**Door:**

Door Class represent the door feature of the game which is the dragon's access to other stages. The door class has 3 private properties that are `state(boolean)`, `location(Point)`, `doorImage(BufferedImage)`. The state property is needed because before the sufficient points are reached, the door's state should be closed. After the points that is enough for player to go to next stage the door will be opened and player is expected to direct dragon to the door. The Door must also have a Point property to get a place in the board in which the game is played. The Door Class has no default constructor but it has a constructor with its all parameters given as arguments. Additionally, getter and setter methods are also provided by the Door Class.

### Options:

Options class is a simple menu in which the user chooses options to check the sound and change the difficulty level. The options class inherits JPanel and implements Mouse and Key Listeners.

When mouse is clicked on soundButton(JButton) the button checks the soundOpen(boolean) property and disables or enables the sound if the sound is open or not.

### Credits:

Credits Class is a simple menu in which the user observes the creators of the game. With a static array of strings, the drawCredits(Graphics g) simply draws each names on the credits screen.

### MainMenu:



**Figure 8.6: MainMenu Class**

This is our MainMenu class of our program. It doesn't include any attributes or methods but it is connected to other submenus which are Help, Options, LeaderBoard, Game and Credits classes. MainMenu is the screen appears when user opens the game. This class provides that screen.

## Board:



Figure 8.7: Board Class

This is the Board class. The Board is one of the crucial classes because game is been played on this board. Its attributes are some integer types that determines the width and height of the board and side of the box. Also this class includes an image attribute which will be the background picture of the game. There is a constructor Board(width, height, boxSide) method. Also there is a drawBoard method which draws the board literally and there are some setter and getter methods for the attributes. Also it is a child class of the Stage class.

## Fire:

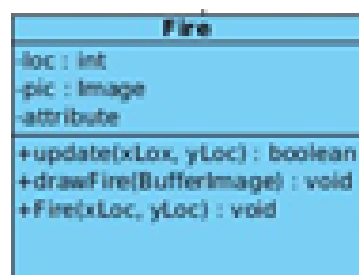


Figure 8.8: Fire Class

This is the Fire class of our system. Fire is a weapon in our game which our dragon uses to destroy enemies or blocks to get more points. It has 2 attributes which are loc and pic. loc is the location of the fire which is an integer and, pic is an image which is the fire itself. There is constructor Fire and update methods which are about fire's spot in the board and drawFire method is using for drawing the fire image to the board. Fire class is a child of the Stage class and it connects with Dragon class.

### Dragon:



**Figure 8.9: Dragon Class**

This is the Dragon class. It is used for defining the dragon in the game that user controls. It calls 3 images for its tail, body and head. Also it has a Fire attribute. Our dragon is made from Linked List so it has a linked list attribute. It has a constructor with parameters of its starting length, head location and direction. It has an update method to update dragon's spot in the board and also, it has a fire method for using fire breath and finally it has a drawDragon method to drawing dragon to the board. It is a child of the Stage class like all game elements.

## 5.3 *Specifying Contracts*

### GameController:

**1- Context** GameController::isCollidingWithAnimal(): boolean

**pre:** Animal.getAnimalCount() != 0

**post:** updateAnimal()

In order to get true as a result first the animal count on the board must be greater from 0 otherwise there will be no animal to collide with. When the method results in true then the GameController will update the animal by deleting it and then create a new one on the game board.

**2- Context** GameController::isCollidingWithEnemy(): boolean

**pre:** Enemy.isEdible == true

**post:** updateEnemy()

If the collision happens and the pre condition is satisfied then the GameController will treat the Enemy as an animal and will do the same thing as if an animal was eaten.

**3- Context** GameController::isCollidingWithEnemy(): boolean

**pre:** Enemy.isEdible == false

**post:** isGameOver = false, notify()

As in the previous one collision is detected and the enemy was not edible. In this case the game has ended and the notify() method is called in order to let the user be aware of the situation.

#### **4- Context GameController::initGame(): void**

**pre:** isGameOver == true

**post:** Stage.createStage()

In order to call the initGame the game must be over or not started yet in both cases isGameOver is false. After the initGame ends the createStage() must have also been called and executed.

#### **5- Context GameController::drawAll()**

**pre:** Stage.drawAllComponents()

**post:** GamePanel.draw()

The drawAll() method of GameController is only called after the Stage class has gathered all the visual information needed with the drawAllComponents() method. GameController::drawAll() is called the GamePanel.draw() must also be called in order to actually update the image seen on the monitor.

#### **Stage:**

#### **6- Context Stage::createStage()**

**pre:** GameController.isGameOver == true

**post:** all model references are renewed.

In order to call the createStage() the game should be over whether it is because the player lost or the stage was passed. After the call is made the model object references that the Stage class has, should all be instantiated.

#### **7- Context Stage::setX( arg)**

**post:** X = arg

There are several setters in the Stage class which all do the same thing. They do not need any pre conditions after they are executed the variable corresponding to X will be updated.

#### **Database:**

#### **8- Context Database:: readProperties(fileName:String)**

**pre:** filesExists(filename:String)

In order to read properties that contain hostname and port to connect db, firstly it will check existence of file.

#### **9- Context Database:: connectToDb(hostname:Sting, port:int)**

**post:** isConnected () = true;

If Database can connect to monogdb(database) isConnected() will return true otherwise it will be false.

#### **10- Context Database:: writeToDb(name:String, point:int)**

**pre:** isConnected()

In order to write all data to database, we need to get true from isConnected method. If it is not true, it will try again to connect but if it is false it will print out error message.



### **11- Context** LeaderBoard::fillArray()

pre: db.getDBCursor() != null

db is database object which was created first. In order to use fillArray() method, we need to get DBCursor object. It will fill array using these objects

### **12- Context** LeaderBoard::fillArray()

post: setArray(Score score)

Leaderboard class's fillArray() method called to fill array with Score objects, when array is fully filled, it will be set to score arraylist . After that we can get score arraylist whenever we need to.

### **13- Context** Score:: setPoint(int point)

pre: getName() != ""

In order to set point to score object, it checks whether name is null or not, if it is not null it lets to set name.

## **Enemy:**

### **14- Context** Enemy:: getValue (): int

**post:** value = this.value

After the getValue() value is returned.

### **15- Context** Enemy:: setEnemies(enemies: ArrayList<Enemy>)

**post:** this.animals = animals

After the setEnemies (enemies: ArrayList<Enemy>) method, the parameter must equal to enemies.

### **Animal:**

**16- Context** Animal:: getLocation (): Point

**post:** location = this.location

After the getValue() value is returned.

**17- Context** Animal:: setLocation (location: Point)

**post:** this.location = location

After the setLocation (location: Point) method, the parameter must equal to location.

### **Block:**

**18- Context** Block:: getBlockImage( BufferedImage)

**post:** blockImage = this.bockImage

After the getBlockImage blockImage is returned.

**19- Context** Block:: setBlockImage (blockImage: BufferedImage)

**post:** this.blockImage = blockImage

After the setBlockImage (blockImage: BufferedImage) method, the parameter must equal to blockImage.

### **Door:**

**20- Context** Door:: setState(isOpened:boolean)

**post:** state = isOpened

After the setState(isOpened:boolean) method, the parameter must equal to state.

**21- Context** Door:: getDoorImage( BufferedImage)

**post:** doorImage = this.doorImage

After the getDoorImage doorImage is returned.

### **Animals:**

**22- Context** Animals:: getAnimals (): ArrayList<Animals>

**post:** result = animals

After the getAnimals () animals is returned.

**23- Context** Animals:: setAnimals (animals: ArrayList<Animal >)

**post:** this.animals = animals

After the setAnimals (animals: ArrayList<Animal >) method, the parameter must equal to animals.

### **Board:**

**24- Context** public void setGridWidth (int gridWidth)

**Post:** this.gridWidth = gridWidth

After the setGridWidth(int gridWidth) method called, the grid width have to be equal to parameter value.

**25- Context** public void setGridHeight (int gridHeight)

**Post:** this.gridHeight= gridHeight

After the setGridHeight(int gridHeight) method called, the grid height have to be equal to parameter value.

**26- Context** public void setBoxSide(int boxSide)

**Post:** this.boxSide= boxSide

After the setBoxSide(int boxSide) method called, the box side value have to be equal to parameter value.

**27- Context** public void setBackground(BufferImage background)

**Post:** this.background = background

After the setBackground(BufferImage background) method called, the background Picture have to be updated to the parameter image.

**28- Context** public int getGridWidth()

**Pre:** setGridWidth(int gridWidth)

In order to call the gridWidth attribute, it must be setted before.

**29- Context** public int getGridHeight()

**Pre:** setGridHeight(int gridHeight)

In order to call the gridHeight attribute, it must be setted before.

**Fire:**

**30- Context** public boolean update(int xLoc, int yLoc)

**Post:** this.fire.xLoc = xLoc

`this.fire.yLoc = yLoc`

After calling `update(xLoc, yLoc)` method, Fire object's `xLoc` and `yLoc` must be equal to parameters.

**Dragon:**

**31- Context** `public boolean update(int xLoc, int yLoc)`

**Post:** `this.dragon.xLoc = xLoc`

`this.dragon.yLoc = yLoc`

After calling `update(xLoc, yLoc)` method, Dragon object's `xLoc` and `yLoc` must be equal to parameters.

## ***6. Conclusions and Lessons Learned***

In this Analysis and Design report we outlined all the features in order to develop our Dragon Valley game. The two main parts of the report will be used to further develop and implement the game. The first part which comes after a brief introduction is the Requirements Analysis. The Requirements Analysis basically tells us how the game should be in terms of a user point of view. This part puts restrictions on the implementation and specifies the general outlines of the game.

In 2.2, 2.3, 2.4 we see how the game should respond to the user's actions. In essence this section shows the main functionalities of the game together with the restrictions it must follow.

In 2.5 we describe situations in which the user can end up in. This part tells how the program might execute. Some of these scenarios are described in much more detail in the UML sequence diagrams located in 3.2.2 .

In 2.6 the report presents use case diagrams. These diagrams show the the sates the game can be in and which states the user can end up in.

In 2.7 we can see the gui of our game. We have a couple of suggested gameplay footage as well as the functionality of how these scenes will respond to the user.

The second major part of the report is the Analysis part. This part will be used in order to code the game in java. The attributes and the methods of each class are shown and how these classes behave can be found in this part.

In 3.1.1 the report defines the domain lexicon. This section shows what the game consists of. It demonstrates how the main parts of the game make the game work.

In 3.1.2 the report shows the main structure of the implementation. This uml class diagram can be directly used to generate the outline of the code (dummy classes that have all the attributes and methods but do not have implementation.). It is the blueprint of the code and perhaps the most important diagram for a non-developer to understand the structure of the program.

In 3.2.1 the report indicates a state chart. This state chart shows what states the can be in. This diagram also shows how transitions between states can occur.

In 3.2.2 we report the sequence diagrams. These sequence diagrams show how the execution of the game happens in terms of the classes of the program. The sequences start with the invocation of the user and continues by creating new objects and interacting with them.

The second part which is section 4 is design section. The design part basically tells us about how we decomposed our system and our design goals.

The third part which is section 5 is object design section.

In 5.1 we clarified the design patterns we used in our project which are singleton, adapter, façade and observer.

In 5.2 we explained our classes in the system and what they do.

Finally, in section 5.3 we explained some contracts from some of our classes.

In essence this report is our guideline to develop the Dragon Valley game.

### **Lessons Learned:**

From this project we learned some lessons that it is crucial to clarify the aspects of the project before implementing it. Writing down the requirement analysis, the object and dynamic models, design (subsystem decomposition and other design aspects) and object design aspects before implementing is a good way to create a better project. Without calculating this aspects before implementation, the creation of the project would be hard and could cause some downsides while implementing it. So an analysis, a design and an object design report is crucially important on a way to create a project.