Bilkent University

CS 319

Object-Oriented Software Engineering

Fall 2015

JAVA OO JUMP PROJECT

ANALYSIS REPORT

October 28th, 2015

GROUP #6 Members

KoralYıldıran

MahmutYılmaz

Pınar Göktepe

SinanÖndül

Table of Contents

1. Introduction

Our project will be a game consisting of a character which is jumping continuously in a tower in order to survive. The tower will have different types of bricks which the character can stay on while camera angle is going up. The bricks will have different bonuses and also on these bricks there will be monsters to hinder the character from going up and there will be some bonuses to go higher in the tower. If the character cannot stay in a brick or hit a monster, the game will be over.

We will use java for implementing game and we will store scores in a text file. After the game is over high scores will be shown.

2.Requirement Analysis

2.1Overview

2.1.1 Gameplay

The character of the game always jumps even the player doesn't direct it. The player will need direction keys on the keyboard to play the game.The player needs to use the mouse in order to make a choice on the main menu.

2.1.2 Leveling

The game becomes harder when the player manages to go further; the number of monsters appearing increases and some of the bricks start moving one side to the other or some of them breaks down. So, game becomes harder and enjoyable by the time.

2.1.3 Bricks

Bricks are essential elements of the game. In gameplay, there will be 3 types of bricks.

1. Standard Bricks: These bricks will be fixed and won't be destroyed until the character moves from brick's maximum field of view. The character will use these bricks to go upward.
2. Moving Bricks: As standard bricks, these bricks will not be destroyed until disappearing from sight but they will be moving horizontally in order to make the character landing to a brick harder.
3. Broken Bricks: Those bricks will be fixed to a place but they will have limited time to break right after the player lands onto it.

2.1.4 Monsters

Monsters are designed in order to bely and astonish the player. While the character is rising, it confronts the monsters and monsters can stable or they can move left to right. Therefore, while player is rising, if it touches the monsters, it dies.

2.1.5 Bonuses

The bonuses are important parts of the game to make it more fun. The bonuses help the player to reach a higher brick or protect the character from monsters or directly give extra points. The player cannot know where and when the bonuses are showed up; they are randomly placed on the game board.

Jet: It gives the ability to reach higher bricks fast in a limited time.

Trampoline: It makes character pass over several bricks with one jump.

Shield: It protects character from monsters.

Coin: It gives extra points to the player.

2.2 Functional Requirements

2.2.1 Play Game

The main purpose of this game is to reach the highest brick. The bricks are placed randomly and if the character cannot stand any brick or the brick crashes or if the characterhits a monster, the character will fall down and the game will be over and user is asked to enter a name if the score is sufficient for top 10 and it is recorded to High Scores. The left and right sides of the playing field are connected to each other. That is, the character can pass to left side through the right side. To win the game player should collect 100000 points. While passing each brick, player gets 10 points. Player will see different types of bonuses. The bonuses are jet, trampoline, shield and coin. Some of these bonuses help the character to reach higher brick and coin directly increases the player's points. These extra features make the game more interesting and enjoyable.

2.2.2 View High Scores

Player is able to see highest 10 scores with the name of the player of all time is kept in the High Scores. Also, players' latest score is kept in the High Scores. Because of this list, players are encouraged in order to set a record.

2.2.3 View Help

Player can see the rules of the game and learn how to play the game. Also, the detailed information about bonuses, monsters and brick types can be seen from View Help.

2.3 Non-Functional Requirements

2.3.1Game Performance

We are planning to make OO Jump work with high performance and low memory costs. There will be dynamic displays such as monster motions, destroying and moving bricks and we don't want those features to lower the speed of the game. As the game is infinite and Java doesn't allow programmers to free heap memory directly, we will try to allocate minimum memory as possible for the objects. This game should work on every standard computer that supports Java.

2.3.2 Graphical Smoothness

This video game should have decent animations and graphics in order to be playable since it's an old fashioned, simple game. Our main goal is to make the game look good and work perfectly. We will try to make dynamic visuals and graphics as smooth as possible.

2.3.3 User-Friendly Interface

The game will have a simple and easy interface to keep the players comfortable. An interface shouldn't be difficult and ambiguous for players to understand. One of our main goals is to keep the interface as simple and user-friendly as possible.

2.3.4 Extendibility

As the developers of the game, we are aware of reusability and extendibilityis important for every project. We make OO Jump suitable for extensions and re-usability. The game will run on a computer that supports Java but other developers will have a chance to extend its running platforms such as IOS, Android and Windows Phones etc. Even after modifying the game for mobile devices, the devices with an “accelerometer” will have a chance to control the character by tilting their device.

2.4 Constraints

OO JUMP will be written in the JAVA and it will be available for desktop. It is played by using only direction keys on keyboard and the mouse.

2.5 Scenarios

2.5.1 Scenario 1: Play Game

2.5.1.1 Success Scenario

User has opened the game and clicked Play Game button to play the game.When the player reached the goal of total point which is 100.000, the game ends and asks user to enter a name and records it to the High Scores table if it is in top 10 and shows top 10 scores.

2.5.1.2 Alternative Scenario-1

User has opened the game and clicked Play Game button to play the game. If the character falls down, the game will be over, the game displays the score and if it is sufficient to enter top 10 asks the name and shows top 10 scores.

2.5.1.3 Alternative Scenario-2

User has opened the game and clicked Play Game button to play the game. If the character hits a monster and falls down, the game will be over, the game displays the score and if it is sufficient to enter top 10 asks the name and shows top 10 scores.

2.5.2 Scenario 2: View Help

User has opened the game and clicked View Help button to get help. Help window will show up. User can see the rules of the game; can be informed about the power-ups and monsters. Then, user is ready to play and clicks the Back button of the help window and returns to main menu.

2.5.3 Scenario 3: View High Scores

User has opened the game and clicked High Scores button to see the top 10 scores and the names of the players. The High Scores window will show up and user can see the scores and user can set scores to zero. Then, user clicks the Back button of the High Scores window and returns to main menu.

2.6 Use case Models

The main use case diagram of the system is shown below in figure 1.

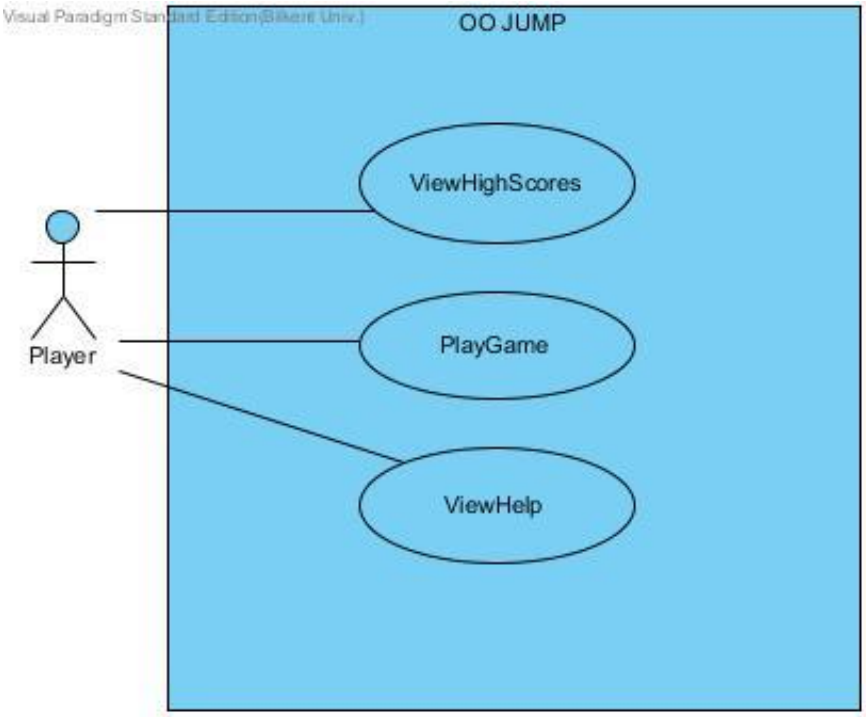


Figure 1: Main Use case diagram

2.6.1 Play Game (for Scenario 1)

Use case name: Play Game

Participating Actor: Player

Stakeholders and Interests:

-System: Wants to supply an enjoyable and proper OO JUMP game to user with no errors and keeps score and name of the player.

-Player: Aims to reach to goal of total points (100.000) and make high score.

Pre-Condition: The game will be opened with provided settings.

Post Condition: If player gets sufficient points to be in top 10 High Scores, High Score is updated by the system.

Entry Condition: Player has already opened the OO JUMP and clicked the Play Game button from Main Menu.

Exit Condition: Player clicks Back button and returns to main menu.

Success Scenario Event Flow:

1) Game is started by system.

2) Player starts playing from easy level.

3) System makes game harder in time while player's character The character doesn't die.

4) Player reaches the goal of total points and game ends.

5) System asks player's name if player's score is higher than the lowest score of the top 10 and records.

6) System shows the top 10 High Scores and names.

7) System returns to Main Menu.

Player repeats the steps 1-7 if he wants to play game again.

Alternative Scenario-1 Event Flow:

1) Game is started by system.

2) Player starts playing from easy level.

3) System makes game harder in time while player's character The character doesn't die.

4) Player can put the character on a brick and it falls down and game ends.

5) System asks player's name if player's score is higher than the lowest score of the top 10 and records.

6) System shows the top 10 High Scores and names.

7) System returns to Main Menu.

Alternative Scenario-2 Event Flow:

1) Game is started by system.

2) Player starts playing from easy level.

3) System makes game harder in time while player's character The character doesn't die.

4) Player makes The character hit a monster and it falls down and game ends.

5) System asks player's name if player's score is higher than the lowest score of the top 10 and records.

6) System shows the top 10 High Scores and names.

7) System returns to Main Menu.

2.6.2 View Help (for Scenario 2)

Use case name: ViewHelp

Participating actor: Player

Stakeholders and Interests:

-System: Shows the document of rules and playing guides of the game.

-Player: Wants to learn how to play the game.

Pre-Condition: The player should be on Main Menu.

Post Condition: -

Entry Condition: Player has already opened the game and in the main menu clicks the Help button.

Exit Condition: Player clicks Back button.

Flow of Events:

User reads the documentation

2.6.3 View High Scores (for Scenario 3)

Use Case Name: View High Score

Primary Actor: Player

Stakeholders and Interests:

-System: Shows the list of top 10 scores and names.

-Player: Wants to see the top 10 scores and names.

Pre-Condition: Player should be in main menu.

Post Condition: -

Entry Condition: Player should select the "View High Scores" from main menu.

Exit Condition: Player should select "Back" in order to return to play or main menu.

Flow of Events:

- Player clicks the "View High Scores" menu.
- System displays the latest score and highest score and names.

2.7 User Interface

2.7.1 Navigational Path

The navigational path of the system is shown below in figure 2.

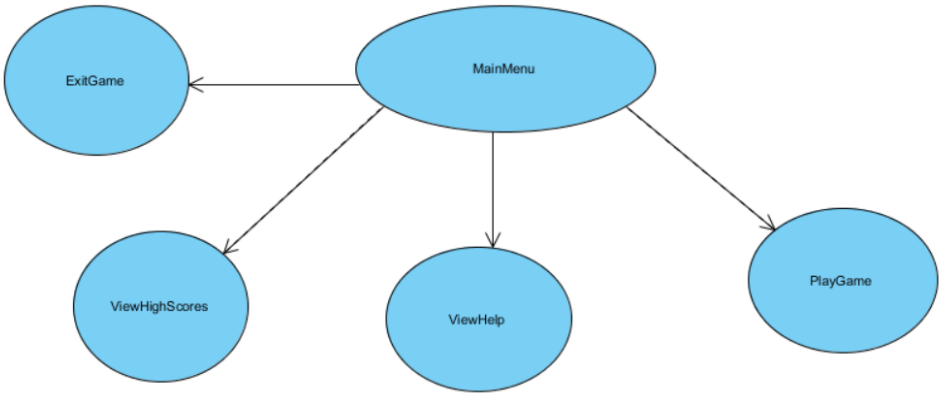


Figure 2: Navigational path of the system.

2.7.2 Screen Mock-ups

2.7.2.1Main Menu

When the game is opened, firstly main menu will come to screen. In this screen; Play Game, View Help, High Scores choices are available for the player.

Figure 3: main menu image of the game

2.7.2.2 Playing the game:

When the player selects the play button, the game starts. At the beginning game is provided in easy level but in time game becomes harder.

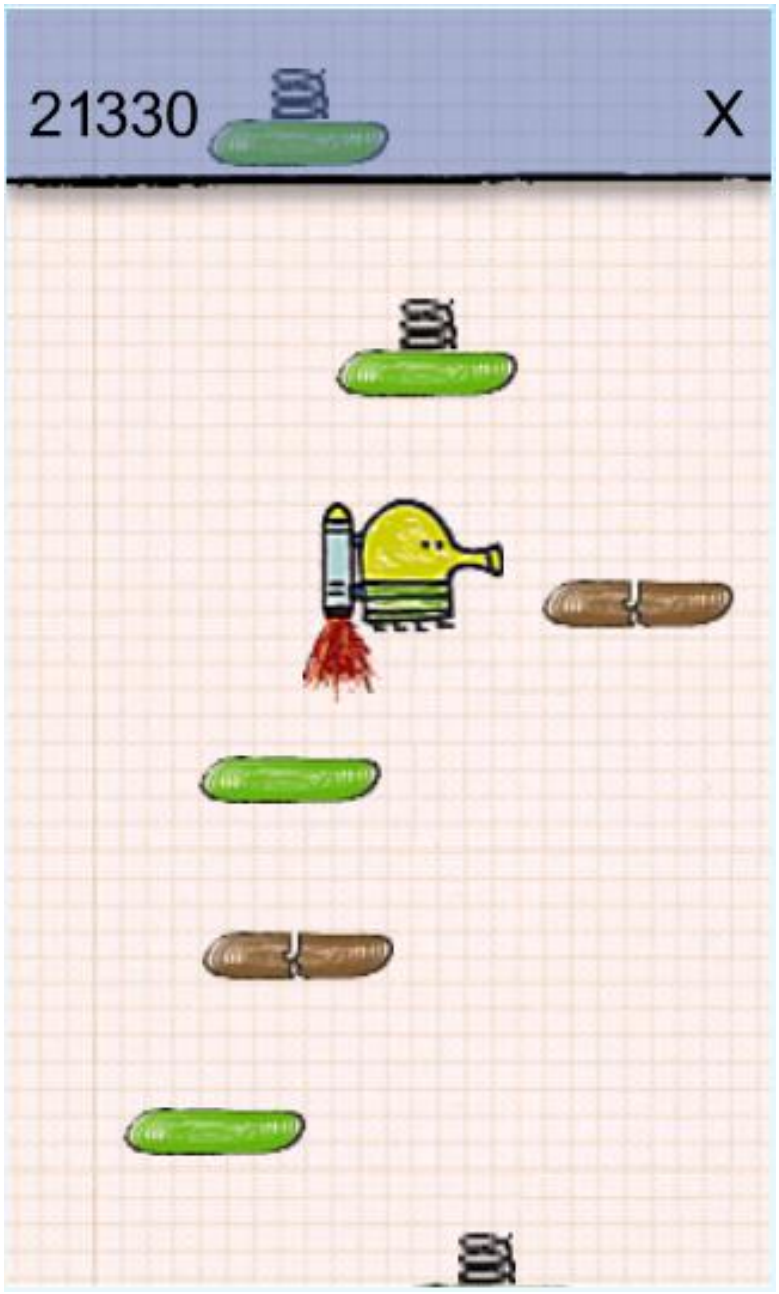


Figure 4: in game image of the game.

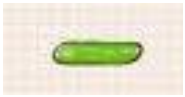
2.7.2.3 Character

The character of the game jumps in vertical direction continuously. Player controls the character by using direction keys on the keyboard.

2.7.2.4 Bricks

Different brick types are below with their explanations.

2.7.2.4.1 Standard Brick



Standard brick has no effect on character. It just increases the total points' of player.

2.7.2.4.2 Breaking Brick

Breaking brick falls apart when the character is on it

2.7.2.4.3 Moving Brick

Moving Brick moves in horizontal direction continuously.

2.7.2.5 Bonuses

Bonuses are listed below with their explanations.

2.7.2.5.1 Coin

Coin increases the player's total points.

2.7.2.5.2 Jet

Jet enables the character fly upper levels for 5 seconds.

2.7.2.5.3 Trampoline

Trampoline enables the character jumps upper levels for 2 seconds.

2.7.2.5.4 Shield

Shield protects the character from monsters.

2.7.2.6 Monster

Monster kills the character when they hit eachother.

2.7.2.7Game over Message Screen

When the player touches the monsters or cannot jump on the any bricks, he/she loses the game and falls down. The player can see the both his/her own score and highscore list. From this page the player can go to main menu by using menu button or he/she can restart the game by using play button.

Figure 5: Game over screen image of the game

**** Some of the screen shots are taken from the game called Doodle Jump.

3. Analysis

3.1 Object Model

3.1.1 Domain Lexicon

Manager: In our program, there are 8 manager classes which are GameManager, CollisionManager, FileManager, ScreenManager, InputManager, GameMapManager, ViewHelpMapManager and ViewHighScoresMapManager. Each of these classes is responsible different part of the game. Their common feature is to"manage" the behaviours of entity objects. We use the word manager in order to refer tocontrol ability of these classes.

Map: In our program, there are 3 map classes which are GameMap, HelpMap and HighScoreMap. Each map classes are responsible for maps of different windows of game. In our program the term "*map"* refers to layout of a screen with its components.

Game: This word basically refers to only the playing game part of the program. However GameManager class is the main class that controls whole application. Beside the GameManager class, GameObject class contains the actual game's objects that are started to use when player starts playing. Also GameMapManager is directly controls the game field from starting the game to the end of the game.

Character: Character is the main object for the game. It jumps up and down continuously. Player can control it with direction keys on key board.

Bonus: Bonus is refers to the power up on this game. There are different types of bonuses as it can be seen in class diagram. These bonuses have different effects on the character.

Monster: In our game, monsters are objects thcat kill the character if they collide.

Brick: In our game, bricks are the platforms for the character to stand. There are three types of bricks. First one is standard brick. This type of bricks may have bonuses or monsters on them. Second type is broken bricks. This type of bricks falls apart when the character is on them. If the player doesn't press the jump button in 1 second, the character falls down and game overs. Third type of bricks is moving bricks. This type of bricks is moving left and right continuously and they can carry bonuses or monsters on them.

3.1.2 Class Diagram

The class diagram of the game is shown below in figure 6.

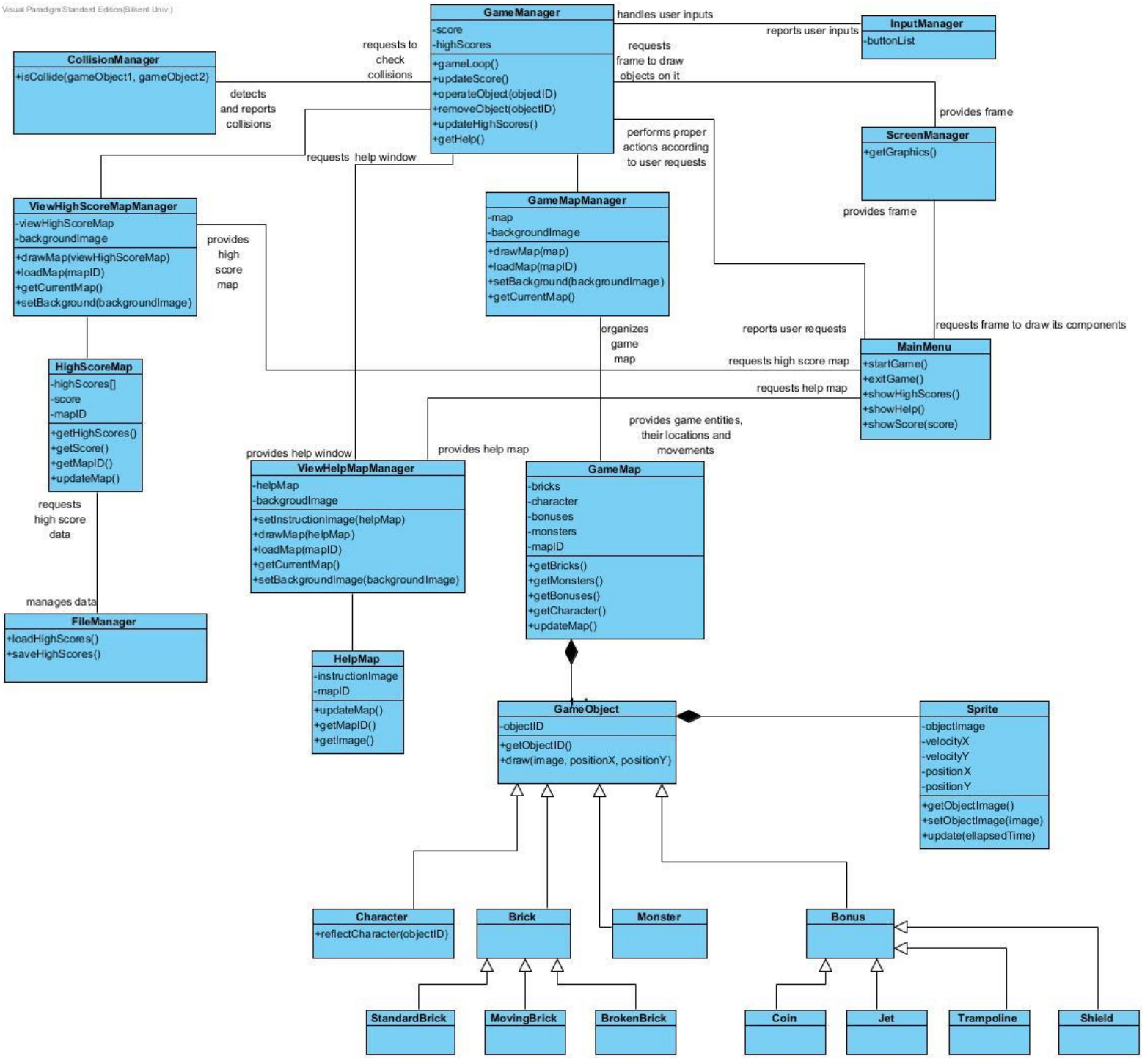


Figure 6: class diagram of the game

The object model of the OO JUMP game is illustrated above. Class Diagram of OO JUMP consists of 25 classes.

MainMenuclass is basically for creating and managing the main menu of the game. This class reports the requests' of the user to GameManager class.

Manager classes of our system carry out the task of organization of the whole game. These classes explained briefly below.

- **GameManager** class is where the whole game is organized and managed. This class complete the tasks by collaborating with other manager classes.

- **CollisionManager**class detects collisions and reports to GameManager class.

- **ScreenManager** class provides the actual window of the game. It also provides the graphic objects to the game frame.

- **InputManager** class reports the user inputs to GameManager class in order to handle the inputs.

- **FileManager** class manages the data related operations. Since the high scores will be kept in text file, the operations of this file will be managed throughout this class.

- **ViewHelpMapManager**class organizes the help window. It places the instruction list which comes from HelpMap class to the help window.

- **ViewHighScoreMamanager**class organizes the high score window. It places the player's score and high score list which is requested from FileManager class to the high score window.

- **GameMapManager** organizes the positions and the numbers of the game objects on the game map. This class enables to find the location of the game objects easily.

We have GameMap class, which consists of GameObjects. GameMap holds the bricks,character, bricks, bonusesand monsters. GameMapManagerplaces the bricks randomly in the game field. Monsters and bonuses are also randomly placed by this class. However the frequency of encountering with a monster or bonus will be changed as the game progresses.

In addition to these classes, we have a class named Sprite. This class is for moving image objects on the screen. So, this class has an image, its x and y positions and its x and y direction speeds as properties. Actually this class is needed for the movement of the character which is controlled by the player.

Beside these classes, there is a HighScoreMap class which contains the help window's components. These component are the score of player and the high scores list. This list will be provided by FileManagerclass.Moreover, there is a HelpMap class which holds the instruction list as image.

For the remaining part, we have our entity objects which inherit from a base class called GameObject. As a child class of GameObject class, Brick class also have 3 child classes which represent 3 different types of bricks. Also, Bonus class has 4 different types. So, it has 4 child classes. In order to construct our game in a well-organized way, we want to benefit the inheritance relation.

3.2 Dynamic Models

3.2.1 State Chart

The object waits for the player to start play. After player starts to play, system checks for is there any collision between a brick and character. If there isn't a collision character falls down and game overs. If there is a collision, system checks if the brick is breaking brick. If it is a breaking brick, waits for 1 second for another jump. If character doesn't jump, it falls down and the game is over. If the brick isn't a breaking brick, the system checks if there is any monster or bonus on this brick. If there is a monster on the brick, the character collides with the monster and if the character hasn't got a shield, it falls down and game is over. If the character has a shield and it collides with a monster, monster cannot kill the character and it continues to play. If there is a bonus on the brick, system checks the type of bonus. If the bonus is a coin, the values of the coin will be added to the total point of the player. If the bonus is a shield, the character has a protection from monsters for 10 seconds. If the bonus is a jet, the character flies for 5 seconds. If the bonus is a trampoline, the character makes a big jump and flies for 2 seconds. After these checks, if the character is still alive, it waits for player to play.

The state chart diagram of the character object is below in figure 7.

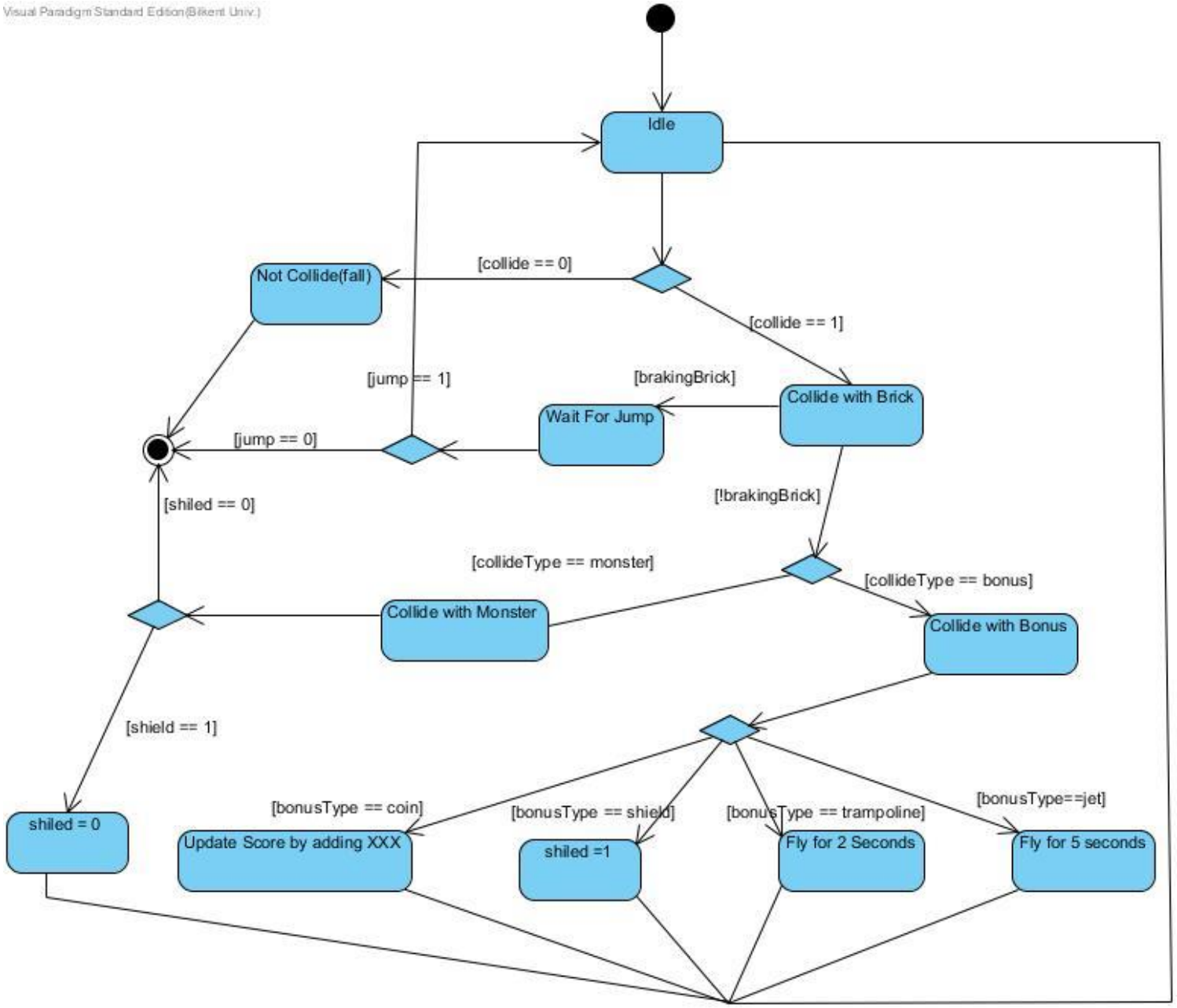


Figure 7: Statechart diagram of the character

3.2.1.1 Activity Diagram

After the game is started, system waits for the user input, simply directional keys. While there is an input, system updates the map and at the same time checks for the locations that the character moves. According to the things at those places, the system makes a decision that the place is valid or not. If the location is not valid, game is over. Otherwise,checks for what it is: if it is a power up or a brick the game continues and if the goal of total point is not achieved, system goes into game loop and waits for an input; if it is a monster, the game is over. The activity diagram of the character is shown below in figure 8.

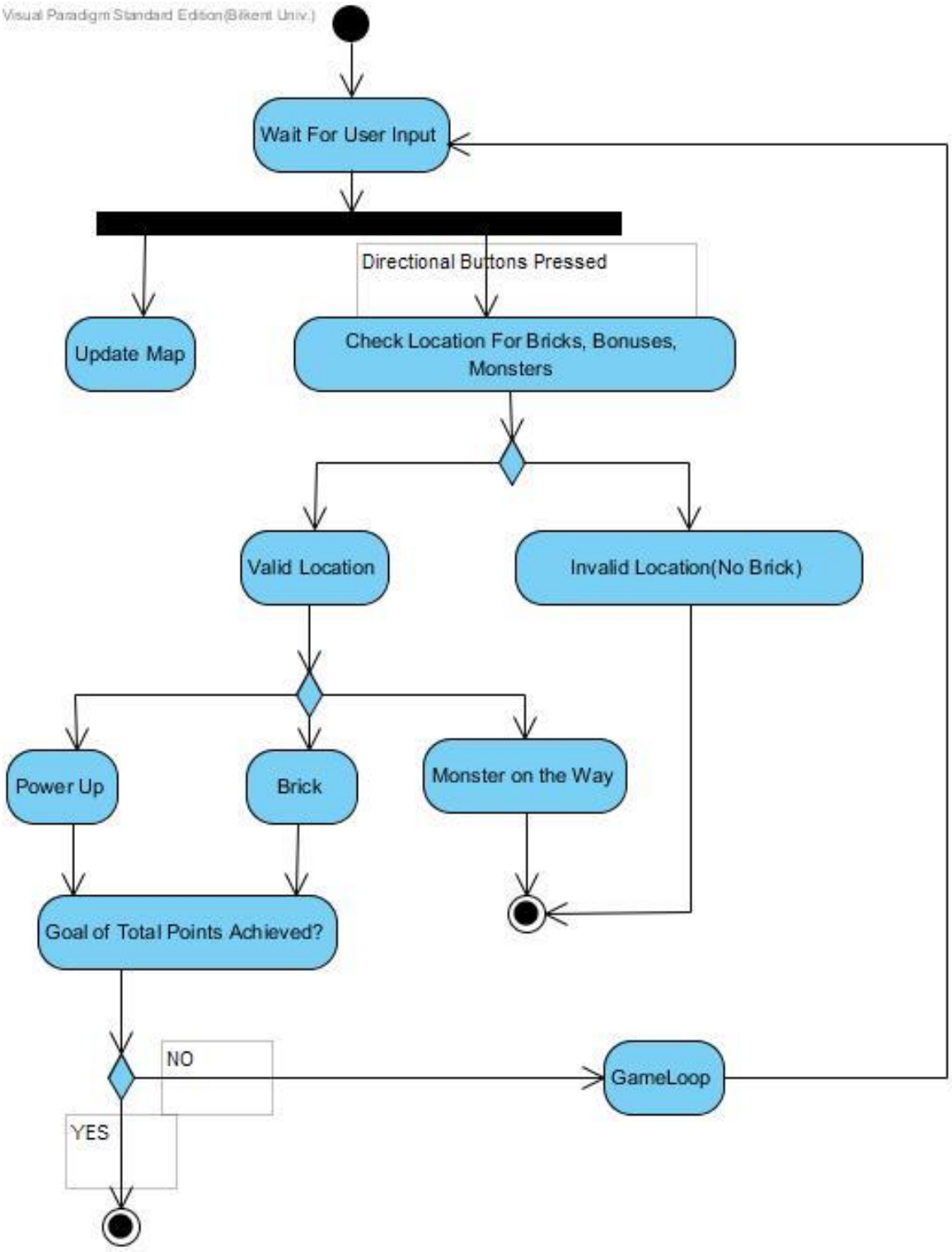


Figure 8: activity diagram of the character.

3.2.2 Sequence Diagram

3.2.2.1 Start Game

Player executes the program, he sees the opening window of the game, and then he chooses the Play Game section. He starts to play the game.

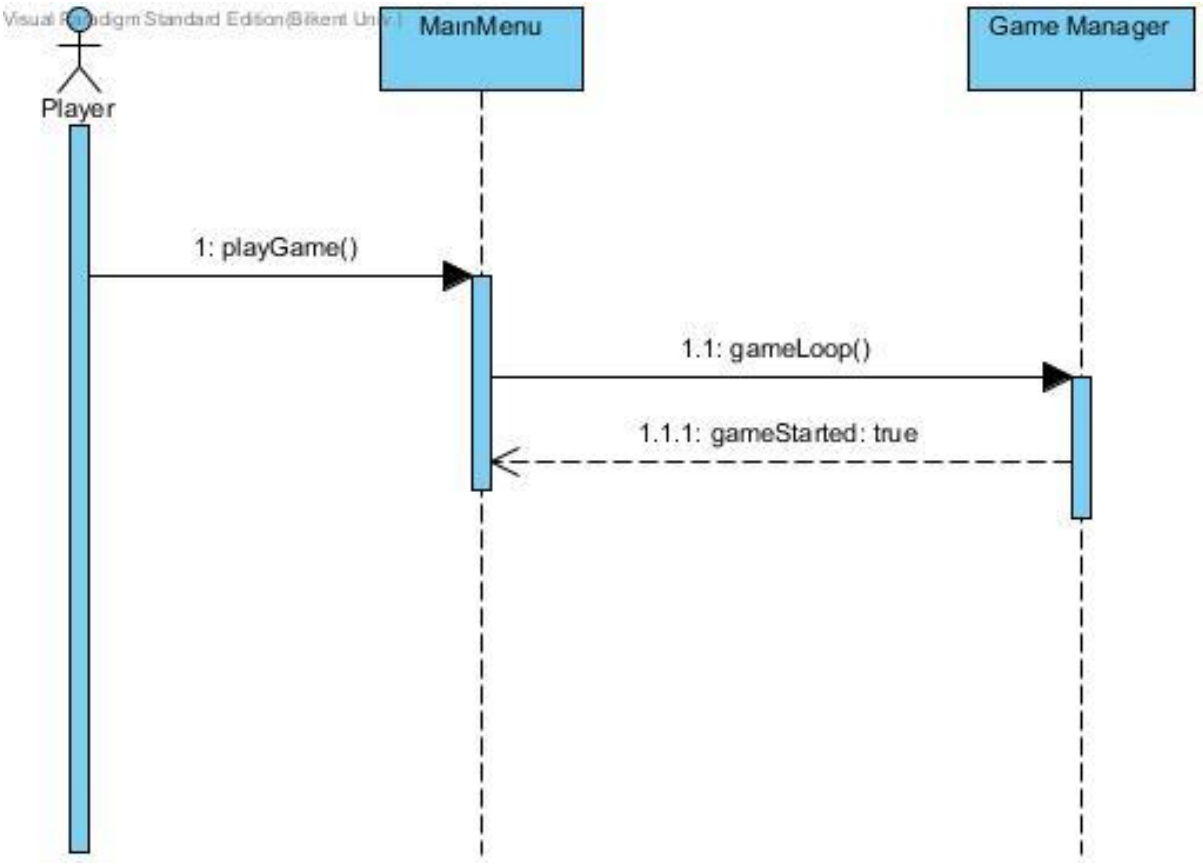


Figure 9: Start game sequence diagram of the game

In this sequence diagram, player executes the OO JUMP game. Then he requests to play the game. GameManager contains the main window of the game. It also contains the other important windows and their components visually.

GameManager is in interaction with the Play Game Window and its contents. The player sends a request for playing the game. GameManager responds to the player and after starting the game, player can play the game. In this example, player wants to play the game.

3.2.2.2 Bonus Management

Player requests to start game by pressing Play Game button from Main Menu. As the previous sequence diagram shows the process of starting game, player starts to play the game. System enters game loop which manages the whole game dynamics. Assuming that user presses the direction keys on the keyboard in order to put the character on a brick.

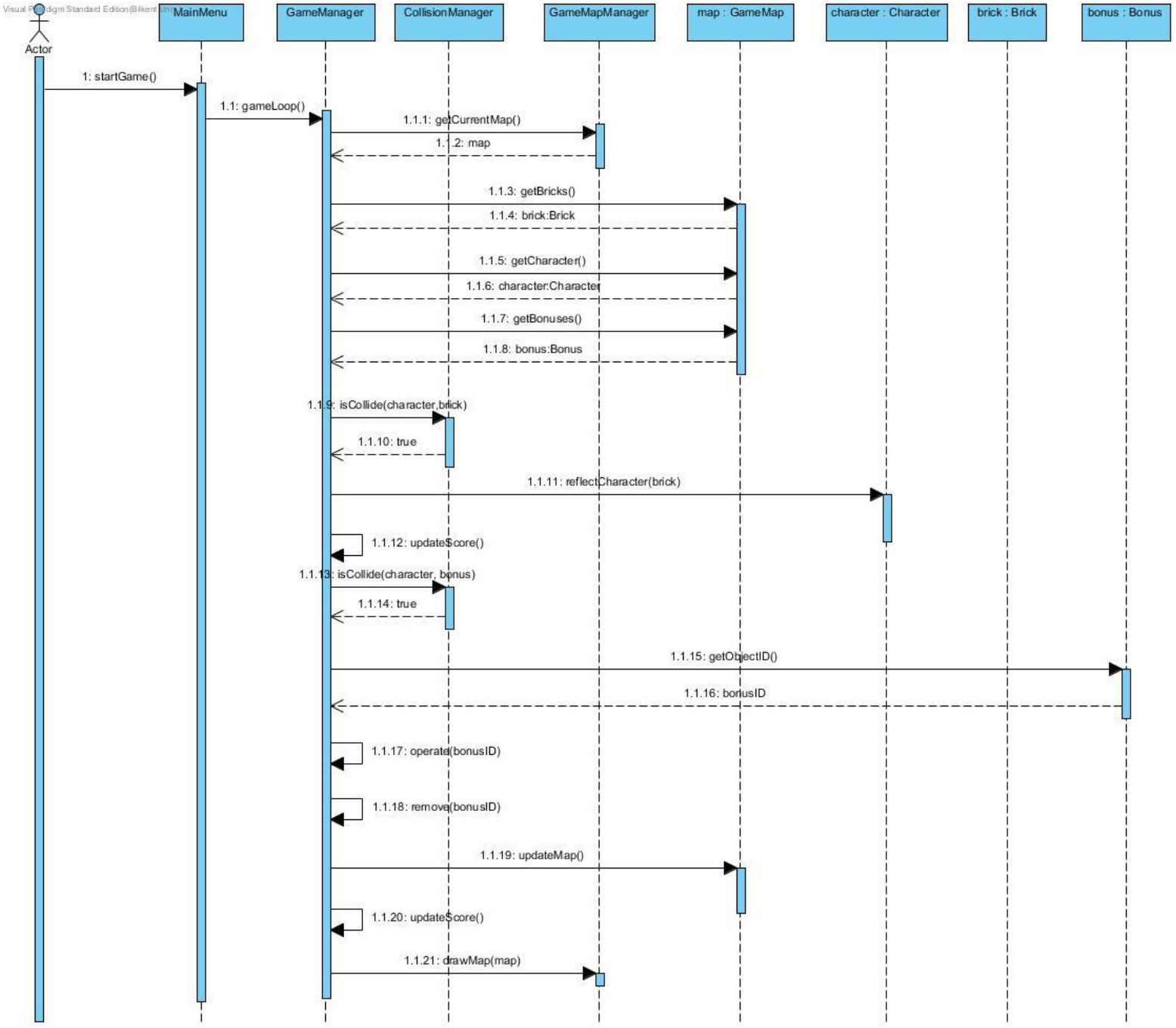


Figure 10: Bonus management sequence diagram of the game

System checks if the character touches to brick. If it is happened system puts the character on the brick. Then system checks if there are bonuses on the brick. If there is a bonus, system reflects due to the type of the bonus. If the bonus is a coin, system adds extra points to the player. If the bonus is a jet, the character flies to higher in tower for 5 seconds. If the bonus is a trampoline, the character makes a big jump and flies for 2 seconds. In this scenario, bonuses are placed randomly on the randomly selected bricks. Then assuming that player uses direction keys on the keyboard to direct the character to a brick with bonus. System checks the encounter between bonus and character. If there is an encounter, system applies the bonus which represented by id, after system updates the score of the player.

3.2.2.3 Monster Management

Player requests to start game by pressing Play Game button from Main Menu. As the previous sequence diagram shows the process of starting game, player starts to play the game. System enters game loop which manages the whole game dynamics. Assuming that user presses the direction keys on the keyboard in order to put the character on a brick.

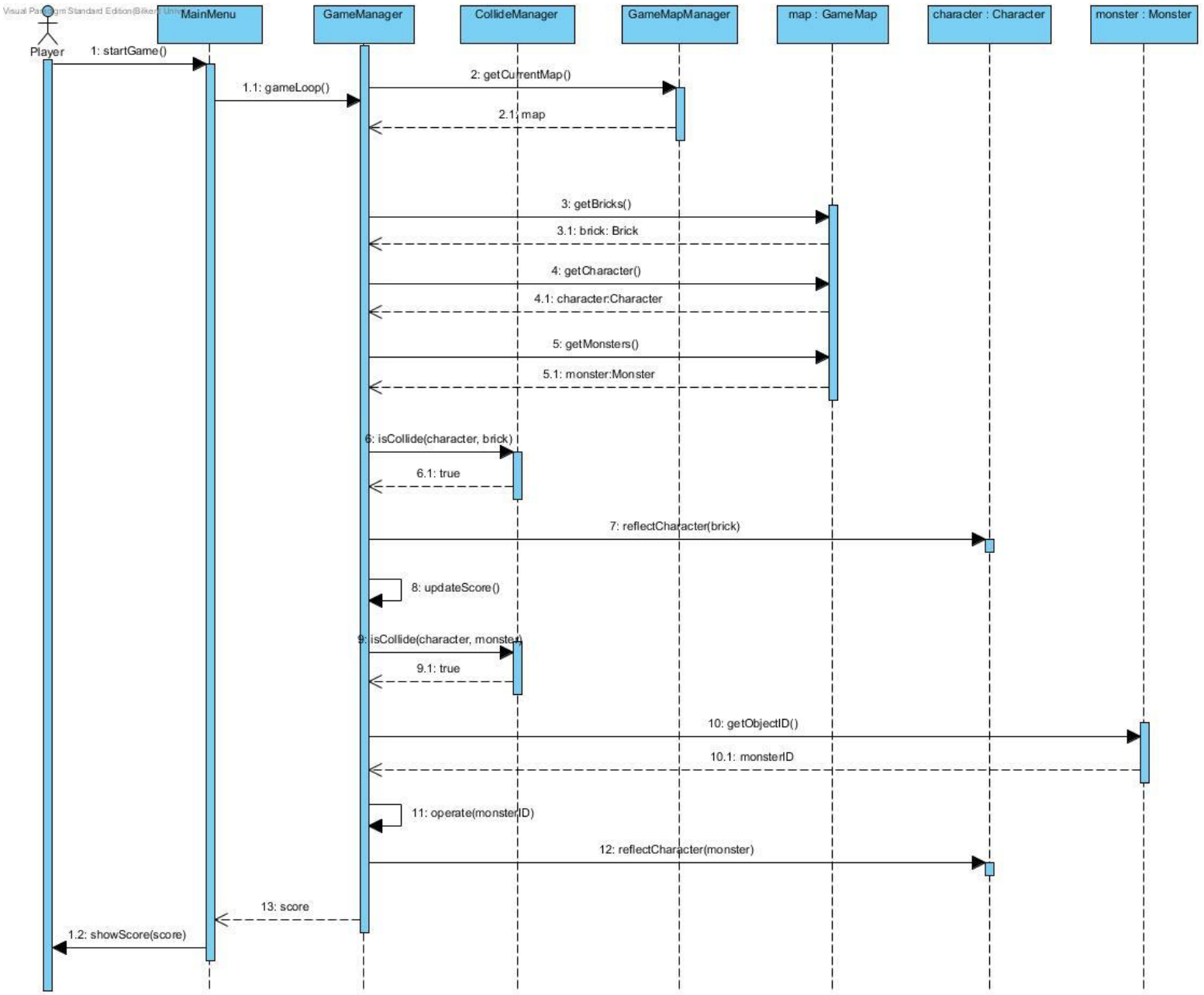


Figure 11: Monster management sequence diagram of the game.

System checks if the character touches to brick. If it is happened system puts the character on the brick. Then system checks if there are monster on the brick. If there is a monster, system reflects to character and kills the character and game is over if it hasn't got shield. If it has got a shield, the character continues the game.

3.2.2.4 View Help

Player executes the program, he sees the opening window of the game, and then he chooses the help document to read. He gets information about how to play the game (i.e. the instructions).

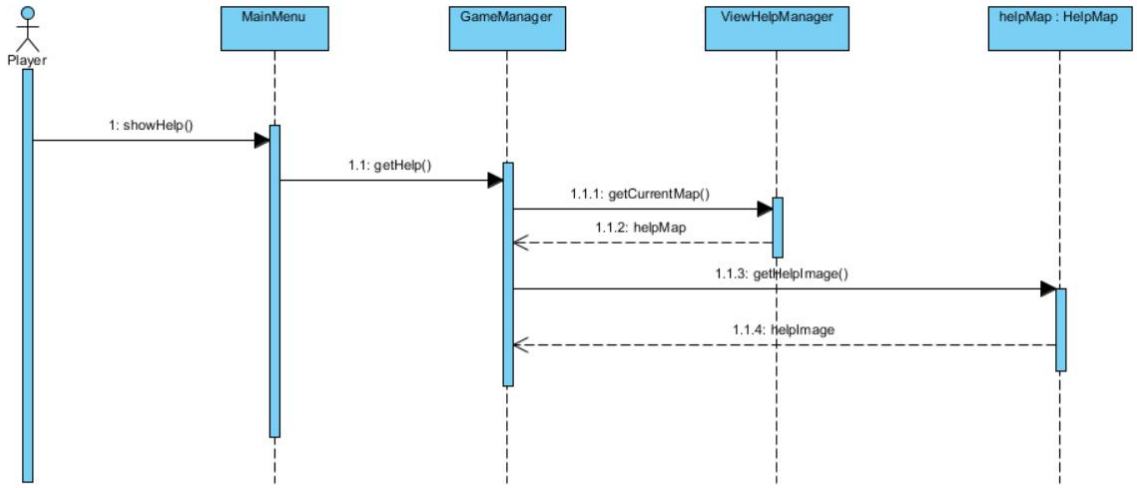


Figure 12: View help sequence diagram of the game

In this sequence diagram, player executes the OO JUMP game. Then he requests to see the help document related to the instructions of the game. GameManager contains the main window of the game. It also contains the other important windows and their components visually.

GameManager is in interaction with the Help Window and its contents. The player sends a request for opening the help window. ViewHelpManager responds to the player and after opening the help documents window. In this example, player wants to see the instructions help page which contains the related information about the features of the game.

The lifeline Help Documents sends the requested documents to GameManager, then the player is able to read the help documents about the game instructions of OO JUMP.

3.2.2.5 View High Scores

Player executes the program, he sees the opening window of the game, and then he chooses the High Scores section to look at. He gets information about the scores and players in top 10.

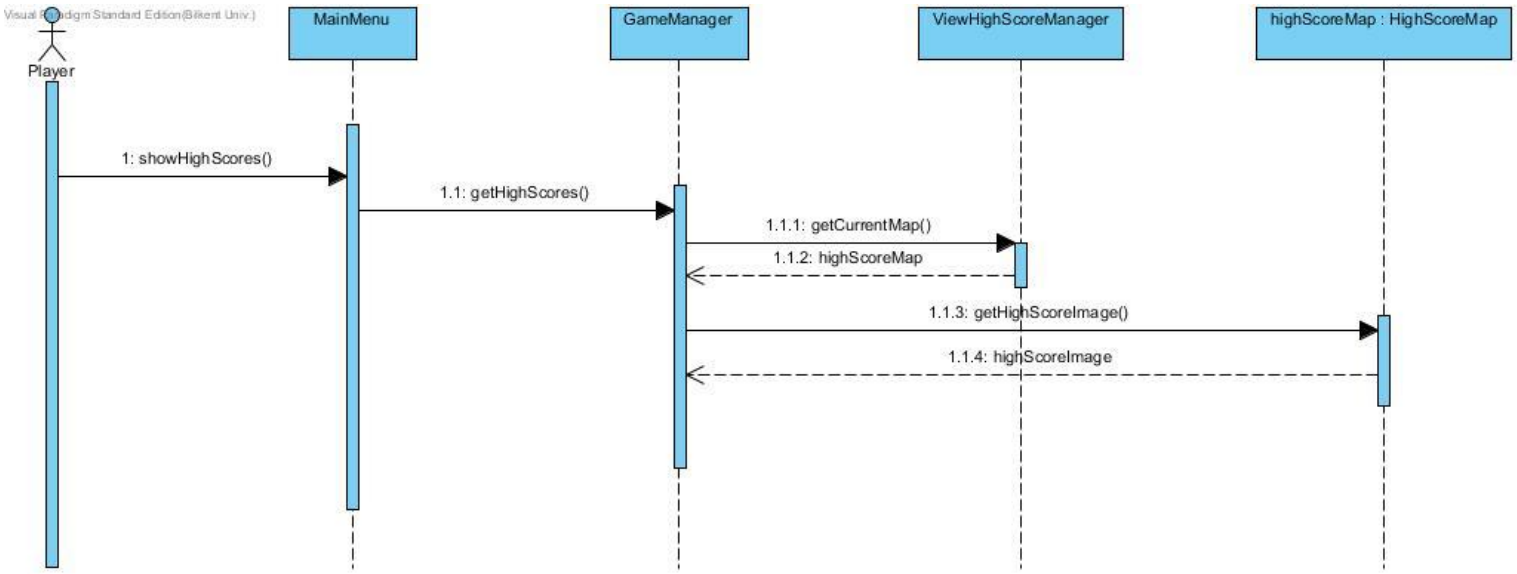


Figure 13: View high scores sequence diagram of the game

In this sequence diagram, player executes the OO JUMP game. Then he requests to see the High Scores of the game. GameManager contains the main window of the game. It also contains the other important windows and their components visually.

GameManager is in interaction with the High Scores Window and its contents. The player sends a request for opening the High Scores window. GameManager responds to the player and after opening the High Scores window, player looks at the scores and the players in top 10. In this example, player wants to see the top10 high scores.

4. Conclusion

In our report, we created an analysis report to design and implement an adventure game called “OO Jump”. Aim of the game is guiding “The character” through bricks and dangerous monsters without falling down or getting bit by a monster in a never ending map.

In requirement specification part, we tried to examine all the occurrences while playing the game. In our project design, we want to fulfil all the functional and non-functional requirements and specified all the requirements.Our analysis report includes two parts. First part is requirement specification and second part is our system model.

Our System Model consists of 4 parts.

1. Use case model
2. User-Interface
3. Class model
4. Dynamic models

We planned and decided what use cases we should have and identified which ones to include in our project. We specified the requirements and use cases.

Second part of the analysis report is user interface, screenshots from the genuine program and navigational path diagram. We will try to keep user interface simpler and good looking as much as we can since we know that having a user friendly interface is one of our main goals. We created a navigational path from the use cases.

. In our class diagram we wanted to produce the best class model since the implementation and design process are the most important parts. We paid attention to the possible classes and connections between them.

Dynamic model includes sequence diagrams, a state chart diagram and an activity diagram. In our analysis, we have shown the possibilities that will constitute the important and crucial parts of the program.

In statechart diagram, we examined all the possible states that the character may encounter. The diagram is based on the collisions of the character with bricks, destroying bricks, monsters and bonuses. The diagram shows all the possibilities to continue or end the game.

Our activity diagram simply explains the game. It includes activities of the character, bricks, monsters and bonuses.

To conclude, we tried to create a good analysis report to guide us through in our design and implementation process. We want to deal with fewer problems as possible in the future when we are implementing our program.