Bilkent University

CS 319

Object-Oriented Software Engineering

Fall 2015

JAVA OO JUMP PROJECT

ANALYSIS and DESIGN REPORT

November 20th, 2015

GROUP #6 Members

Koral Yıldıran

Mahmut Yılmaz

Pınar Göktepe

Sinan Öndül

Table of Contents

Table of Figures

1. Introduction

Our project will be a game that is a clone of the legend game called Doodle Jump. Doodle Jump was built for mobile phones; we are going to build our game OO Jump for desktop.

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 character hits 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.

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.

Inspired by **http://doodlejump.org/**

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 Character

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

Figure 1 Character

2.1.4 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.



Figure 2 Brick Types

2.1.5 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.

Figure 3 Monster

2.1.6 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.

Figure 4 Bonus Types

2.2 Functional Requirements

2.2.1 Play Game

Player sees the game screen and controls the character Obori by using right and left arrow keys. If the character dies, player is able to enter the name if the score is greater than the lowest high score. Player is able to see top 10 high scores and able to return to home screen.

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.

2.2.4 Exit

Player is always able to completely exit from the game and also there is an exit button in the main menu.

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

Extendibility is the ease of adapting software products to changes of specification. Since change is pervasive in software development: change of requirements, of our understanding of the requirements, of algorithms, of data representation, of implementation techniques. Because support for change is important in software engineering, our system will support changes which are explained above. Any developer who has the source code, he/she can change it according to current specifications.

2.3.5 Portability

Portability is the ease of transferring software products to various hardware and software environments. Portability is supported when the hardware or the environment that the program will run, which are operating system, window system, are changed, the program can still run. Since our game will be .jar file which can be run on any hardware/software environment which supports JDK. In this way we will increase the portability of our program.

2.3.6 Robustness

Robustness is the ability of software systems to react appropriately to abnormal conditions. Robustness checks the correctness of the working process of the program. When abnormality happened during loading images and sounds, our system will give exceptions to user to inform about the error. Then it will terminate itself, if the problem occurs during the process of loading images. If the error occurs during the process of loading sound, it will give warning but continues running without sounds.

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 5.

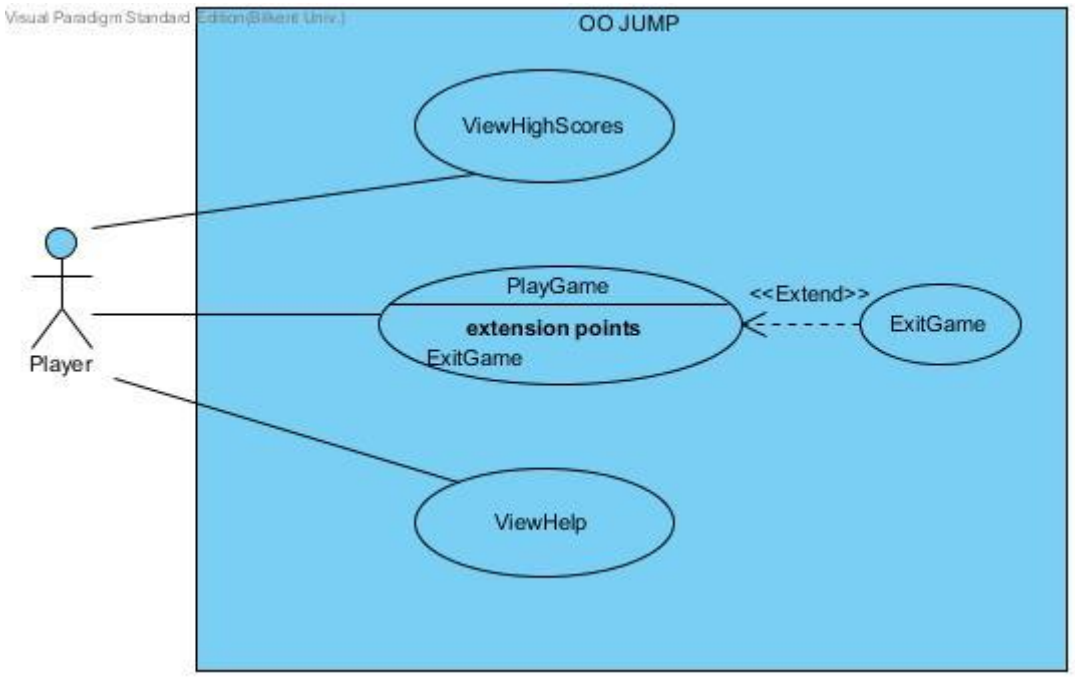

Figure 5 Main Use case diagram

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 6.

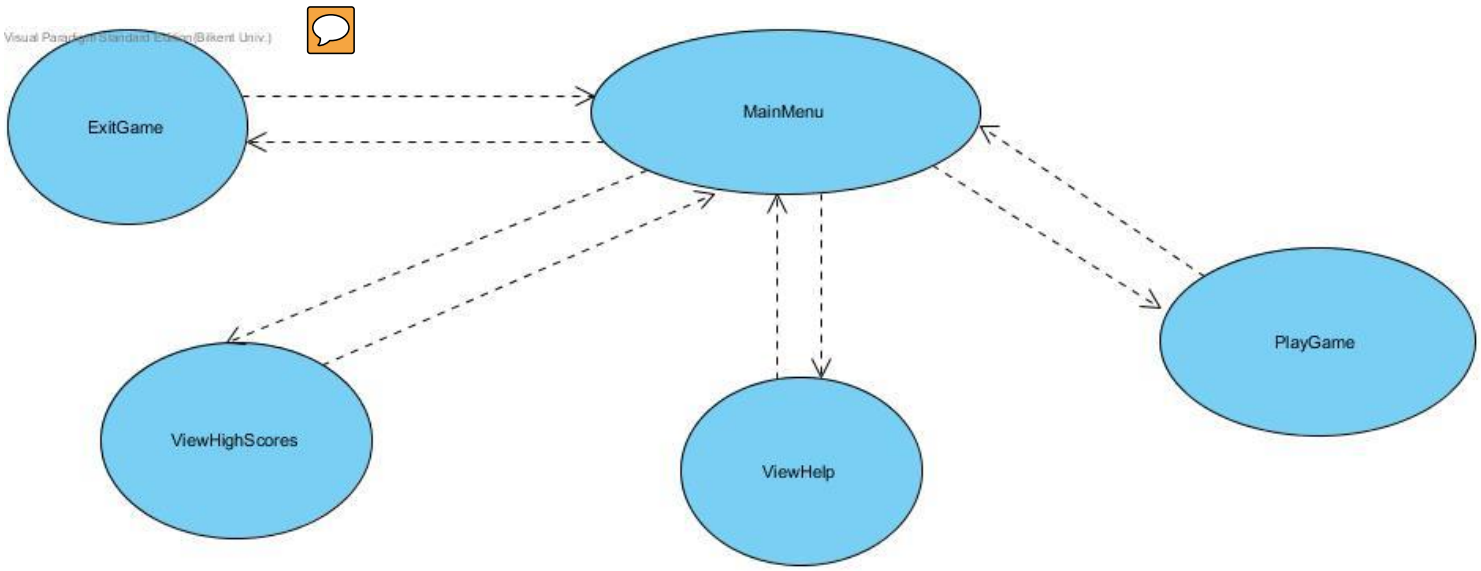

Figure 6 Navigational path of the system

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 7 Main menu image

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.

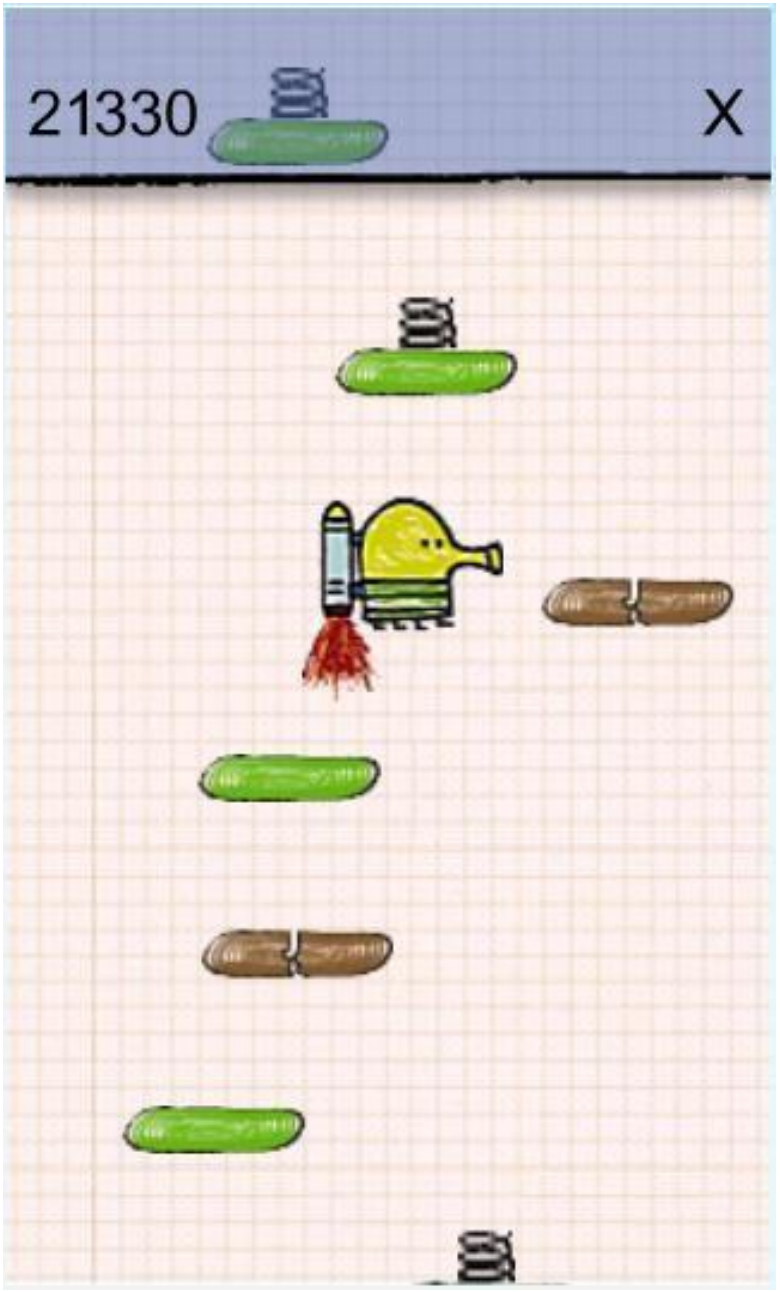

Figure 8 In game image

2.7.2.3 Character

Figure 9 Main character image

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

Figure 10 Standard Brick image

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

2.7.2.4.2 Breaking Brick

Figure 11 Broken Brick image

Breaking brick falls apart when the character is on it

2.7.2.4.3 Moving Brick

Figure 12 Moving Brick image

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

Figure 13 Coin image

Coin increases the player's total points.

2.7.2.5.2 Jet

Figure 14 Jet image

Jet enables the character fly upper levels for 5 seconds.

2.7.2.5.3 Trampoline

Figure 15 Trampoline image

Trampoline enables the character jumps upper levels for 2 seconds.

2.7.2.5.4 Shield

Figure 16 Shield image

Shield protects the character from monsters.

2.7.2.6 Monster

Figure 17 Monster image

Monster kills the character when they hit each other.

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 18 Game over screen

**** 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 that 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 19.

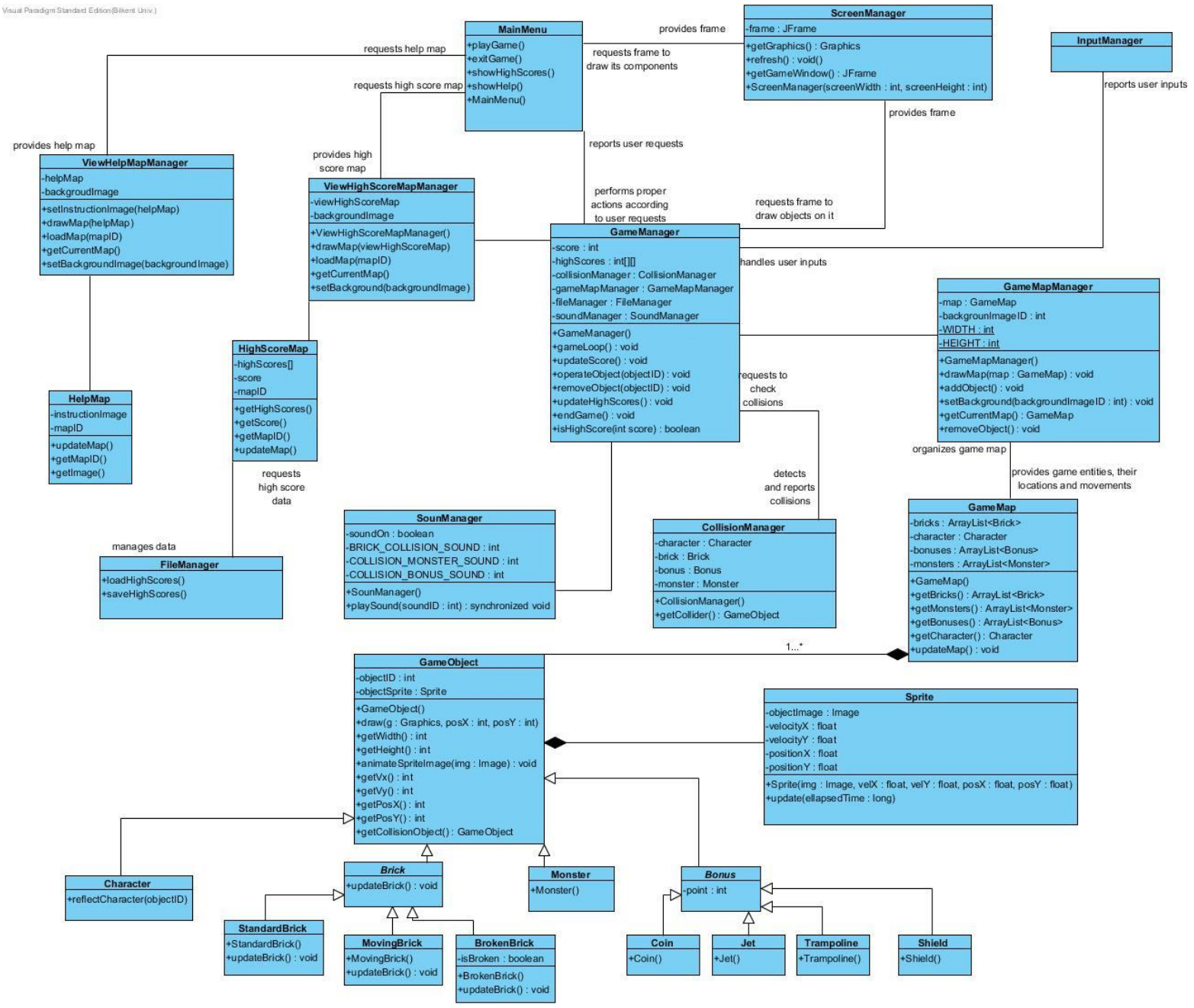

Figure 19 Class diagram of the game

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 character 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 is over. 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 20.

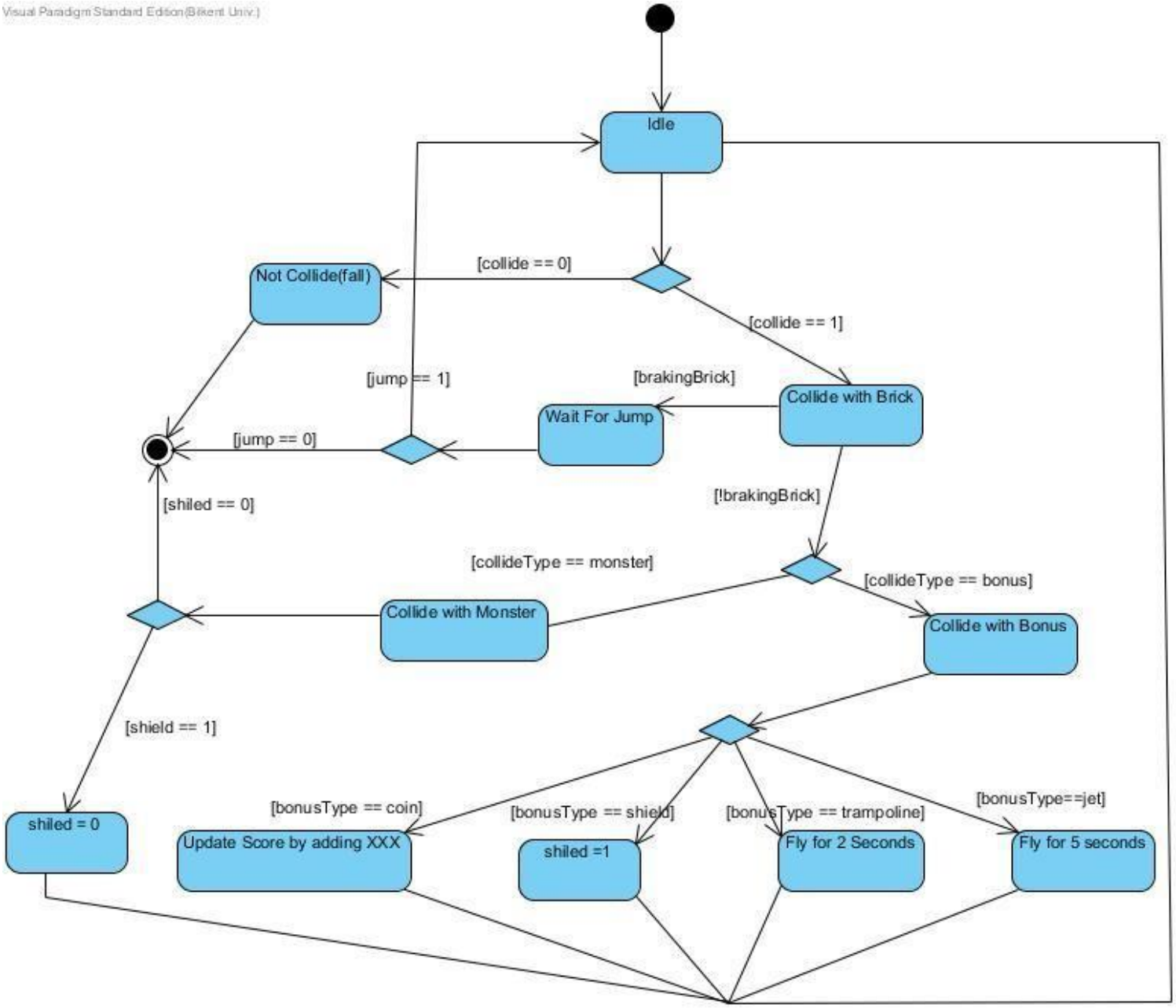

Figure 20 Statechart diagram of the character

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 21.

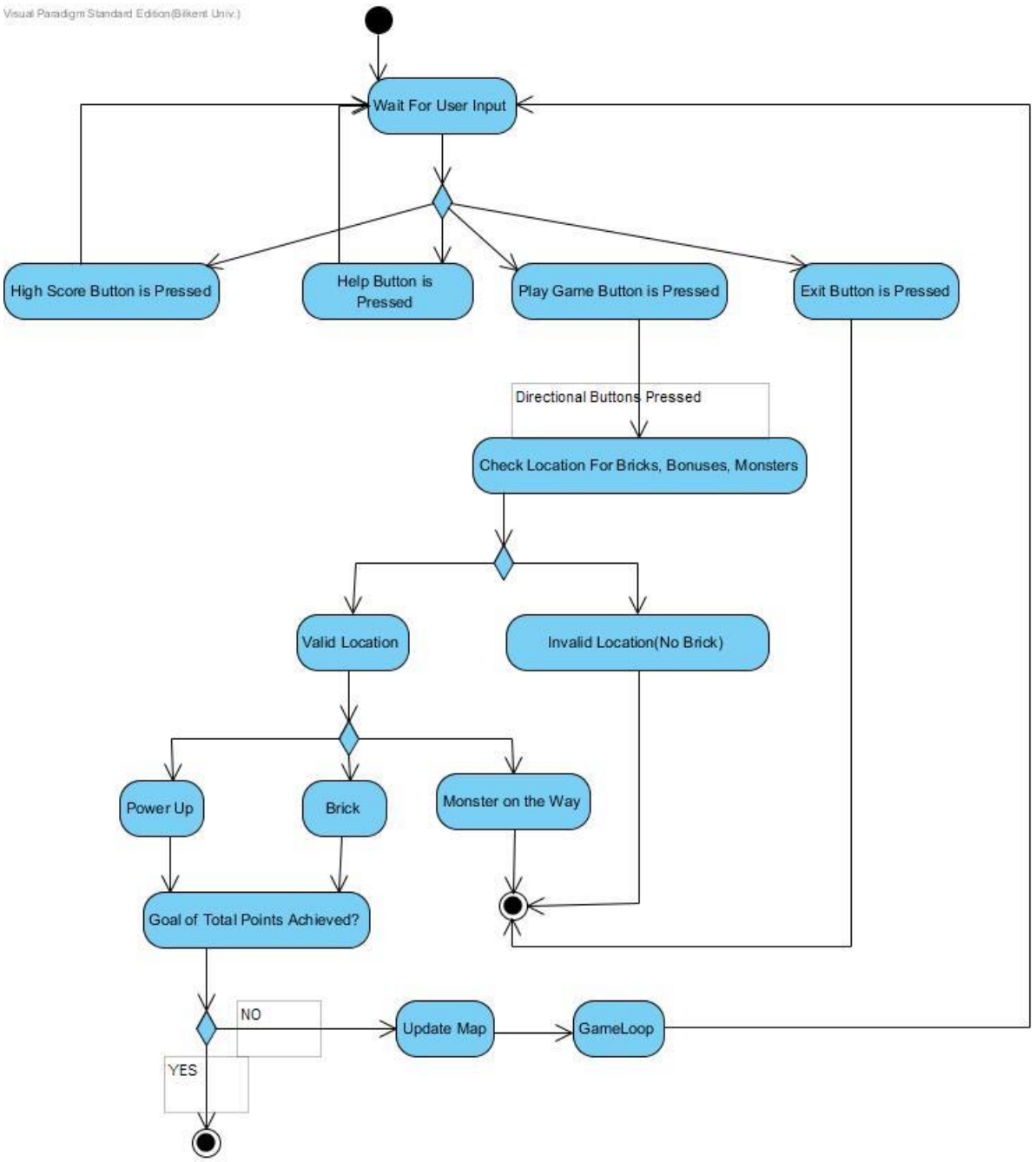

Figure 21 Activity diagram

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.

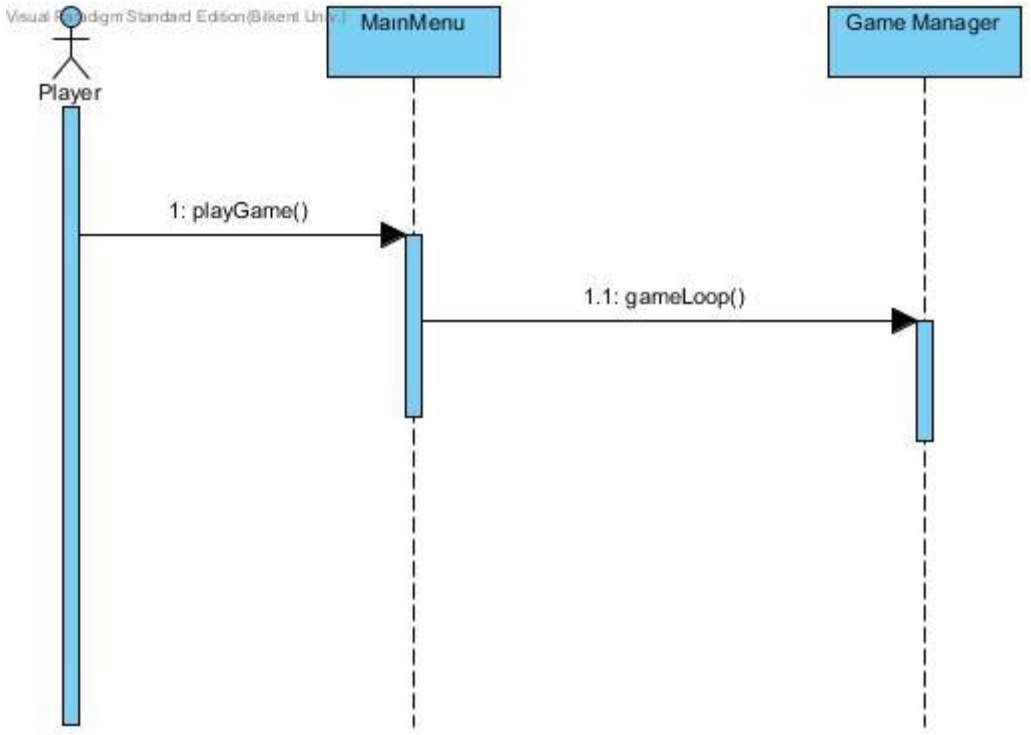

Figure 22 Start game sequence diagram of the game

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.

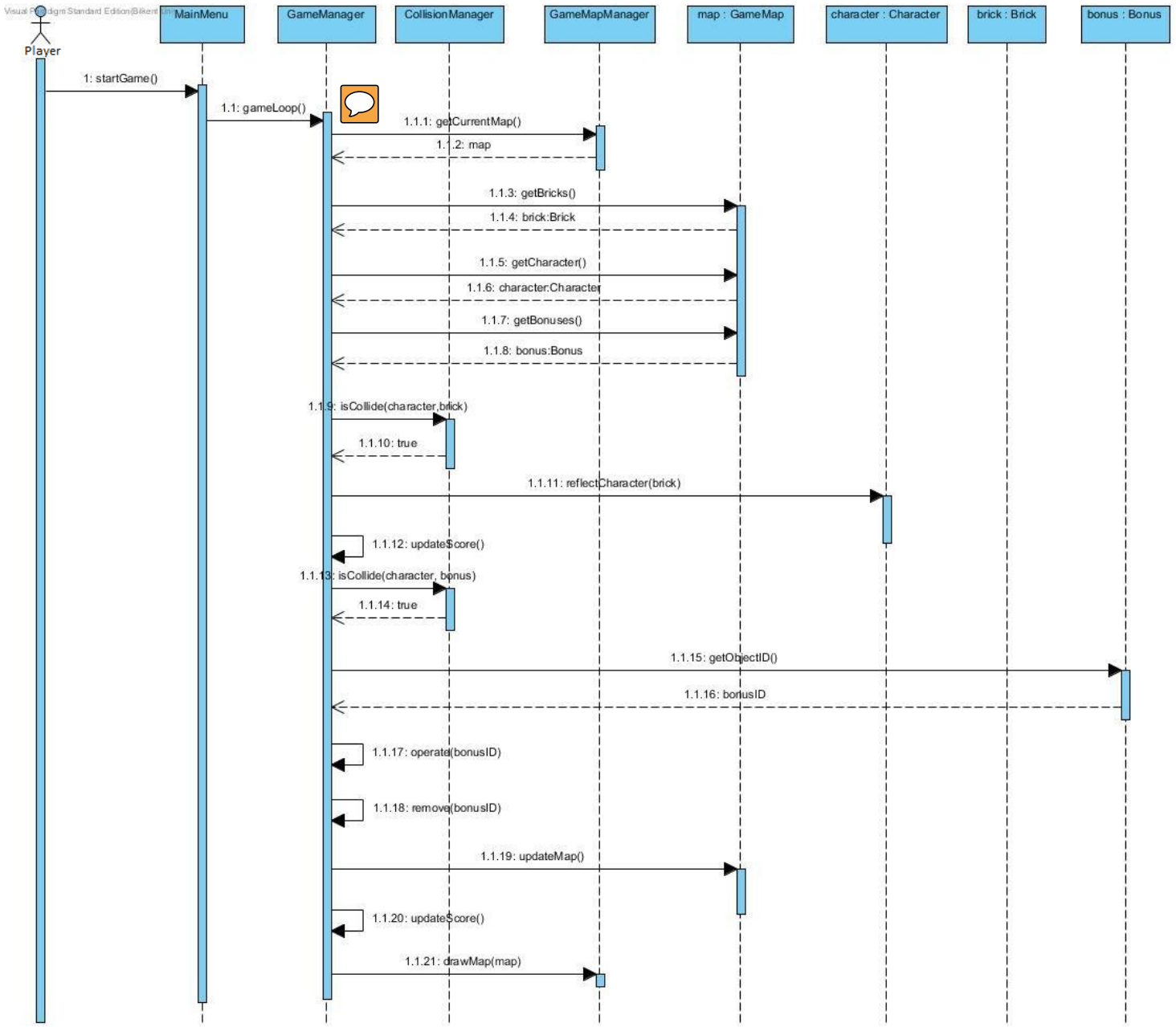

Figure 23 Bonus management sequence diagram of the game

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.

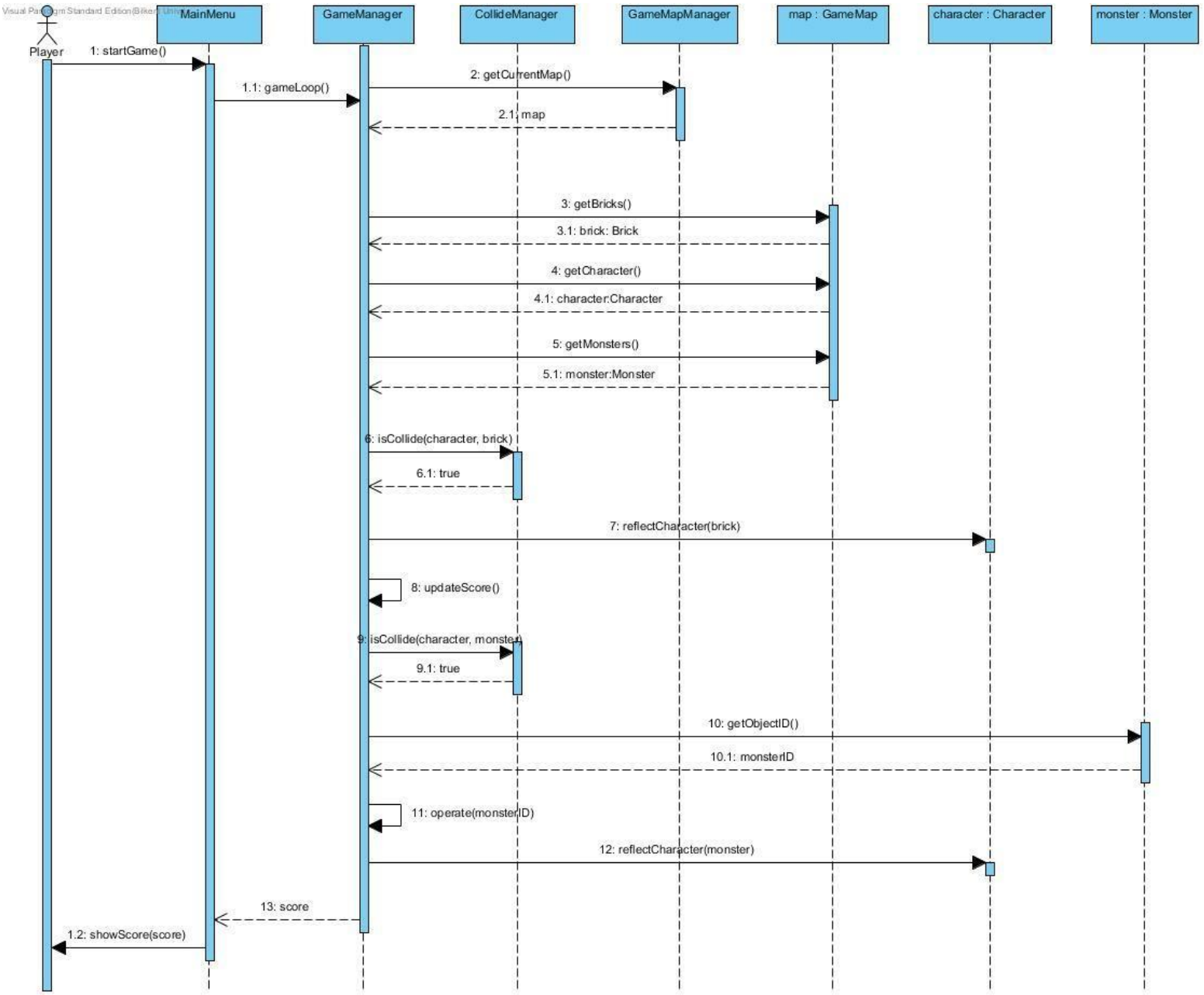

Figure 24 Monster management sequence diagram of the game

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).

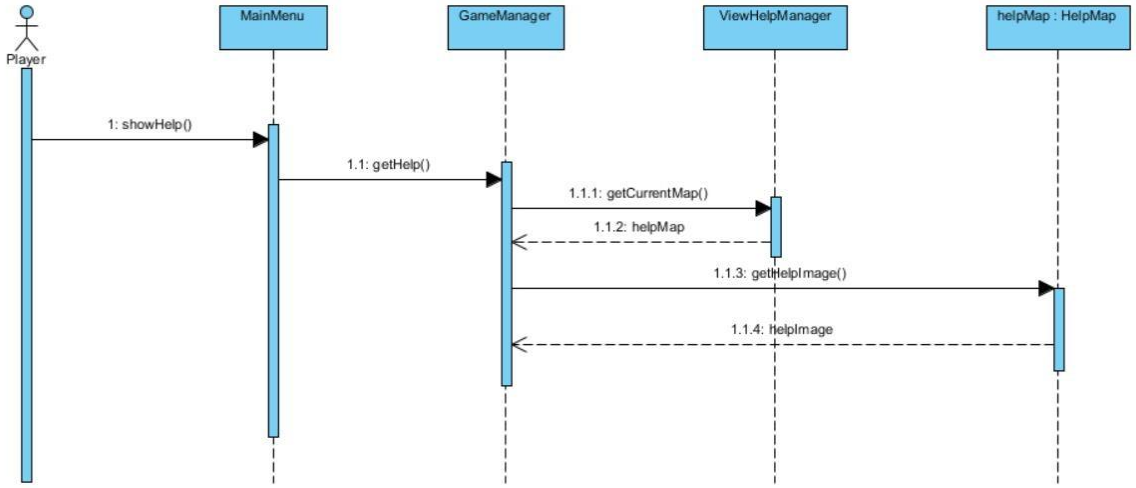

Figure 25 View help sequence of the game

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.

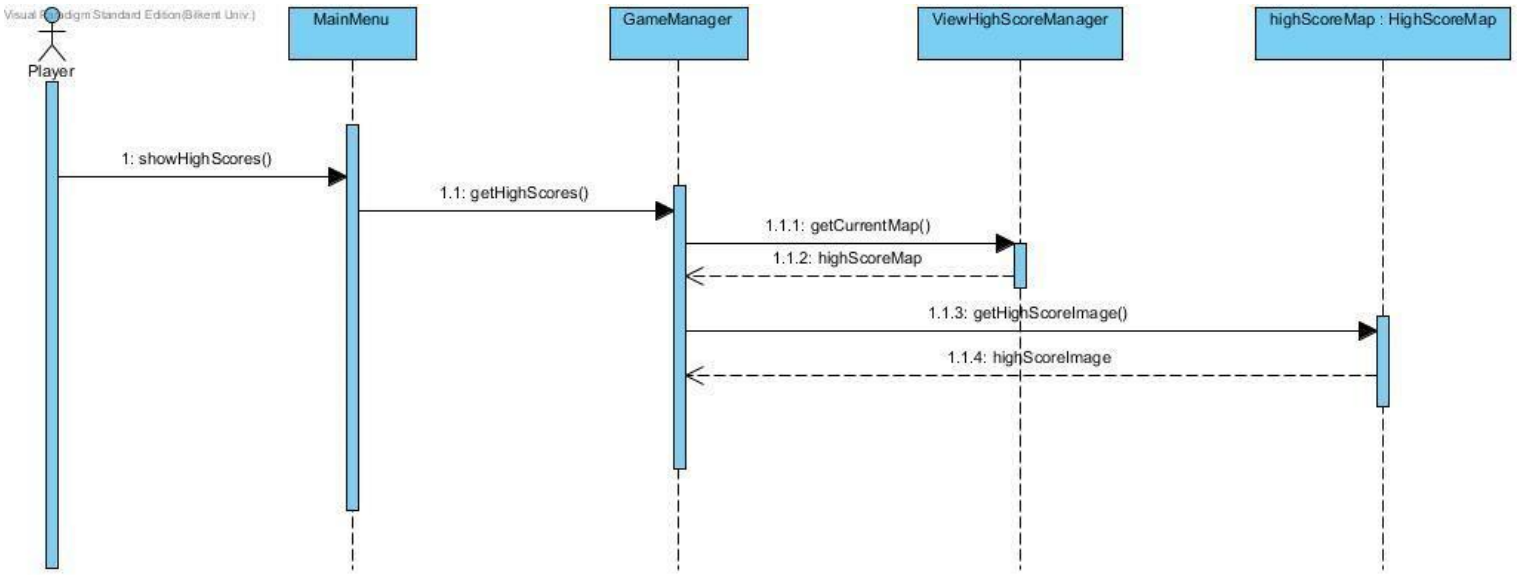

Figure 26 View High Scores sequence diagram of the game

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 Design

4.1 Design Goals

Identifying the class goals before writing the code for our software is crucial for identifying the aspects of our system should be focused on. Knowing that, we set our design goals mainly on non-functional requirements of our software that we defined in our Project Analysis Report. Essential design goals of the software are described below.

End User Criteria

Ease of Use: The game should be entertaining and must have a replay value in order to be played in the future. The players should not feel it difficult to use the system. The system will provide user-friendly and simple interfaces when navigating through menus and the player will easily find desired operations and perform them. Our system will perform actions from the mouse inputs from the user at main menu and will get keyboard operations when the game starts. These specifications make the system easy to use.

- **Ease of Learning**

As the developers of the game, we presume that the players don't have the knowledge on how to play the game, how to get better scores, lose conditions and power up properties. It's important for the player to get this information before setting a new game so we decided to add Help submenu to the main menu. The logic of the game is very simple that the user won't struggle after displaying how-to-play (help) manual.

Maintenance Criteria

- **Extendibility**

Our design will be suitable to add new functionalities and entities to the existing version because we know that new elements in the game brings excitement and motivation to the players as it is important for almost every software that exists.

- **Portability**

Portability is important in order to reach many users. Our project will be able to reach wide range of users. We determined that implementing the game in Java language would be the best for having a chance to deliver the game to a wide range of users, since Java Virtual Machine(JVM) is platform independent and will provide our game portability.

- **Modifiability**

Our system must be easy to modify in order to edit existing functionalities. We will minimize the coupling of the subsystems to avoid changes that affect the system components crucially, while modifying the functionalities.

Performance Criteria

- **Response Time**

As the developers, we are aware that response time is very important in a game since a game creates distractions and major "turn-offs" to a player when it lags. The game should not cause the player to lose interest. Our system will have a minimum response time to user actions and will display animations and effects smoothly.

Trade offs

- **End of Use and Ease of Learning vs. Functionality**

A game should not bother the user with too much functionality. The system should be understandable and easy to use. Our system will focus on the usability rather than functionality. It won't distract the users with hard to understand functionalities that will make the user step back from playing the game. . To be obvious, we put more priority to usability than functionality.

- **Performance vs. Memory**

We are determined to make the animations, sounds and graphic effects, and the transitions between those as smooth as possible. Performance is the main focus on our system. We kept memory usage as low as possible to gain performance. For example, the player won't be able to see below where he/she has already passed the map. So the bricks, monsters and bonuses won't be used again. Instead of keeping them in memory, we will use garbage collection to reuse the memory that those instances created in the on-going map, where we could continue to create new ones that would fill the memory. We will use dynamic memory as efficient as possible.

4.2 Subsystem Decomposition

In the subsystem decomposition part, the system is divided into relatively independent parts in order to explain how the system is organized. Decomposition of relatively independent parts have a huge importance, because while we identifying subsystems, our influence on features of the software system is really effective. The performance, extendibility and modifiability which are features of software can be affected. Especially, decomposition of parts has an important role with regards to creating high quality software and meeting non-functional requirements.

Figure 27 system shows three subsystems and they are focusing on different cases of software system.

- User Interface
- Game Management
- Game Entities

Even if these three subsystems are connected each other, but these connection have done by thinking any changes in future. In other words these connections might be changed according future actions or demands. Also, they are working on exactly different cases.

In figure 28 to it is seen that Game Management and Game Entities are bounded. In addition to these, only connection between User Interface subsystem and Game Management subsystem is also provided over Game Manager class. That's why when any error or changes happen in User Interface subsystem, only Game Manager class which has a role as interface of control unit affects.

If we chance to further think to figure 28, we realize that classes who are working on a common purpose are gathered and performing similar tasks.

Our main purpose was meeting our system goals when we identified subsystems. Because of this reason, we tried to design an effective software system with flabbily coupling and high cohesion. This condition not only provides us to have flexibility but also encourage us to enhance in future.

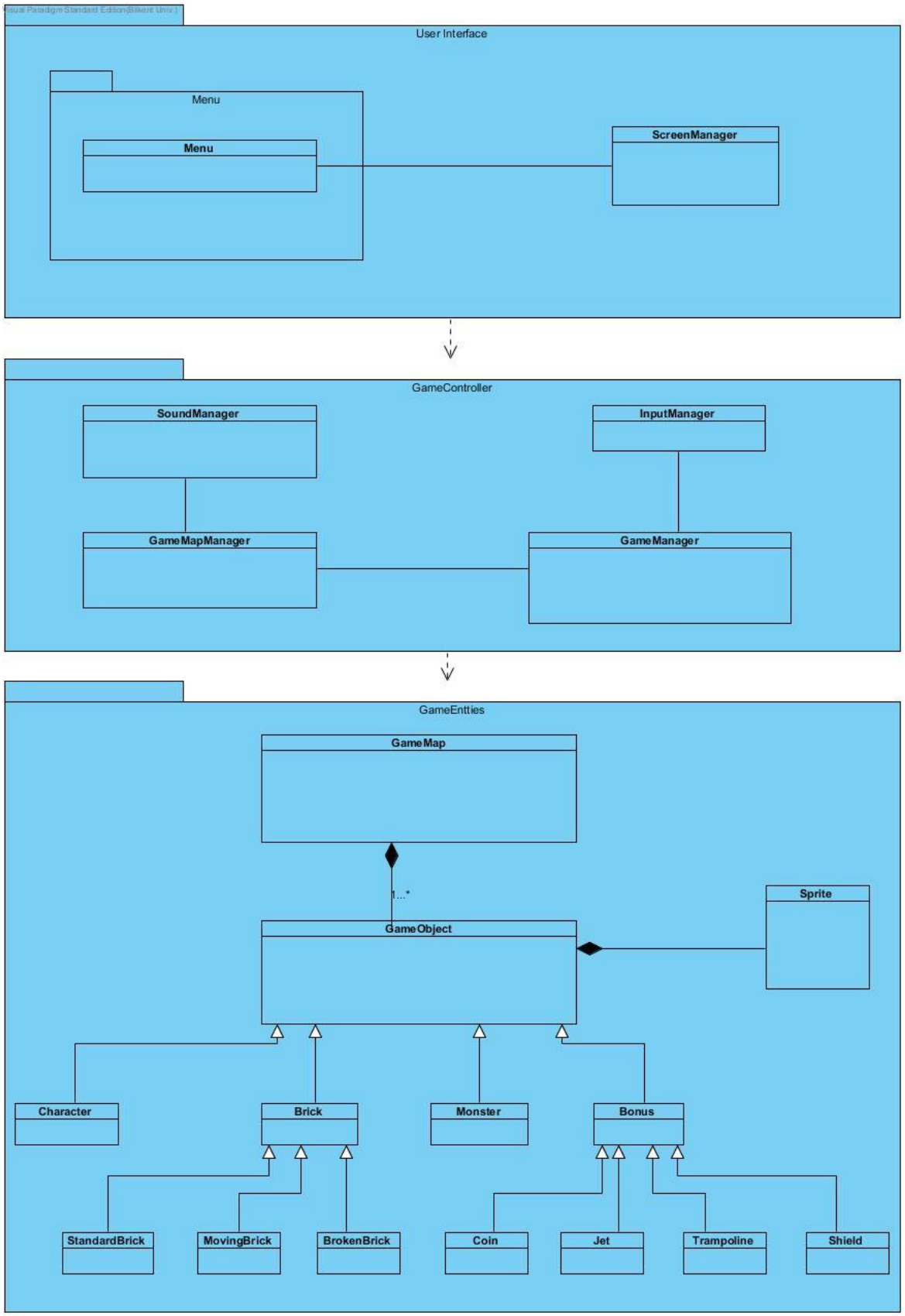

Figure 27: Subsystem Decomposition

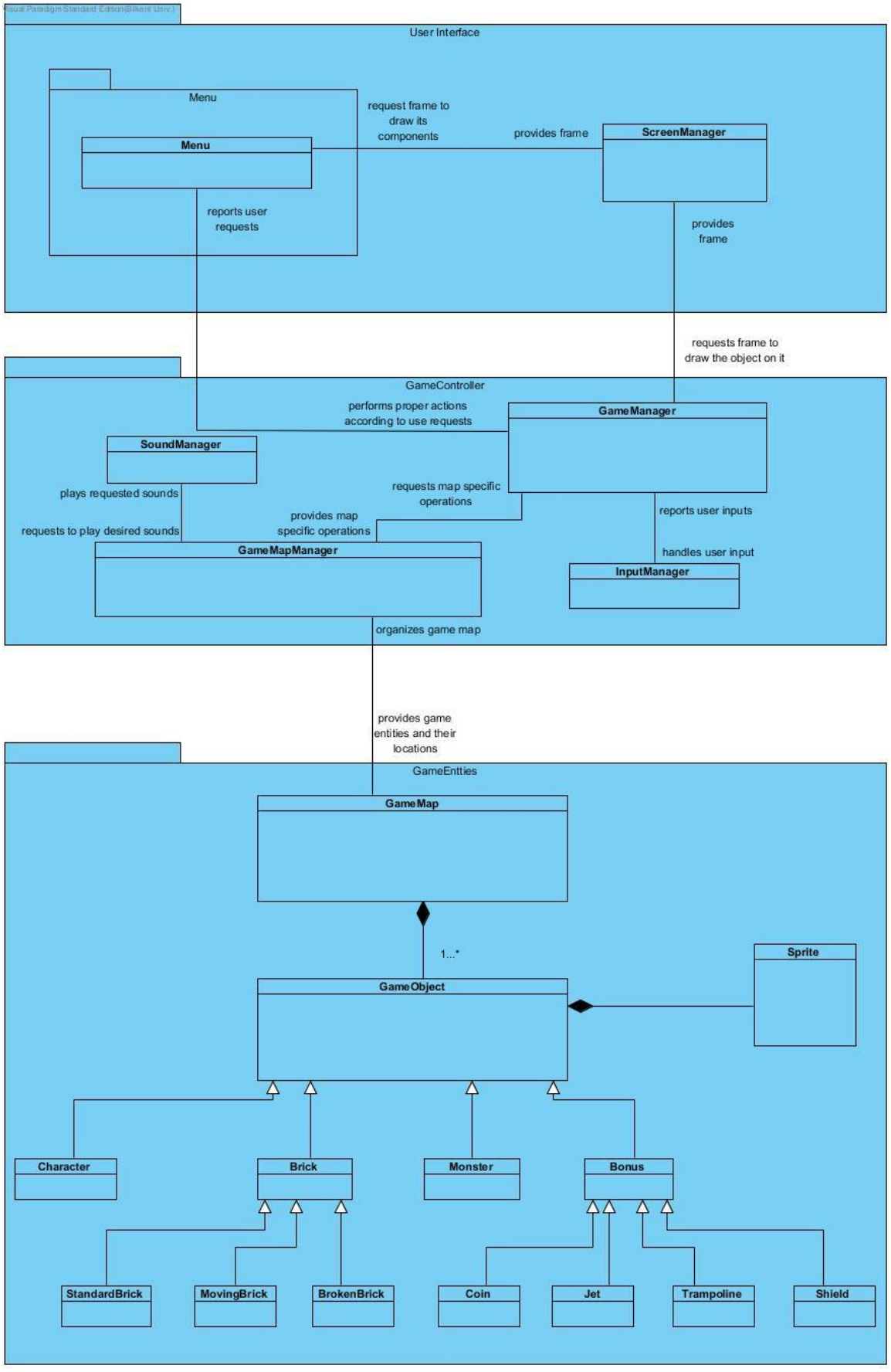

Figure 28 Detailed Subsystem Decomposition

4.2.1 User Interface Subsystem Interface

User interface subsystem provides graphical system components to our software. It manages transitions between panels which are constructed for variety of options from the main menu screen. The reference of UI subsystem to other subsystems is provided by menu class which is considered as an interface.

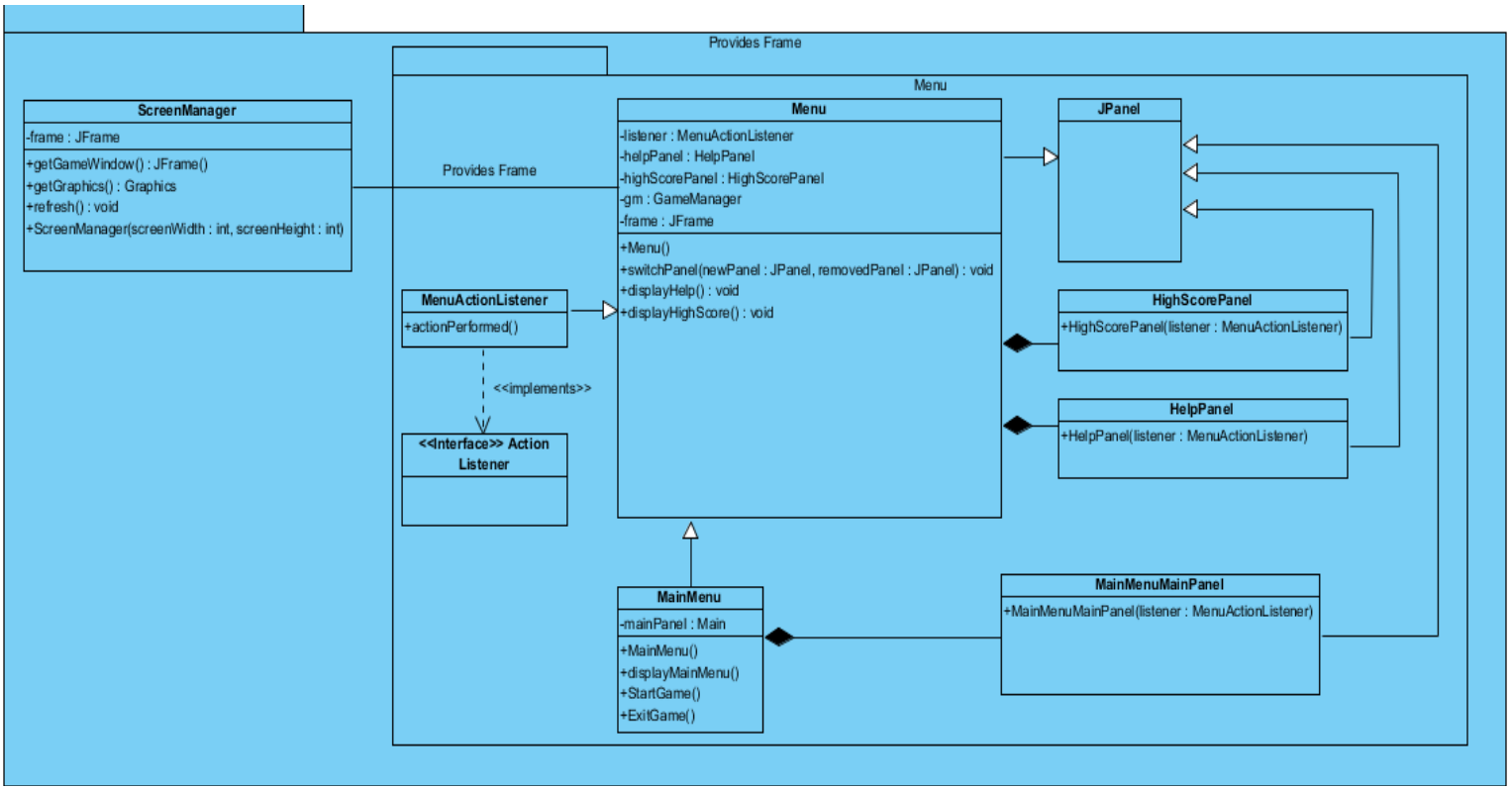

Figure 29 User Interface Subsystem

Menu Class

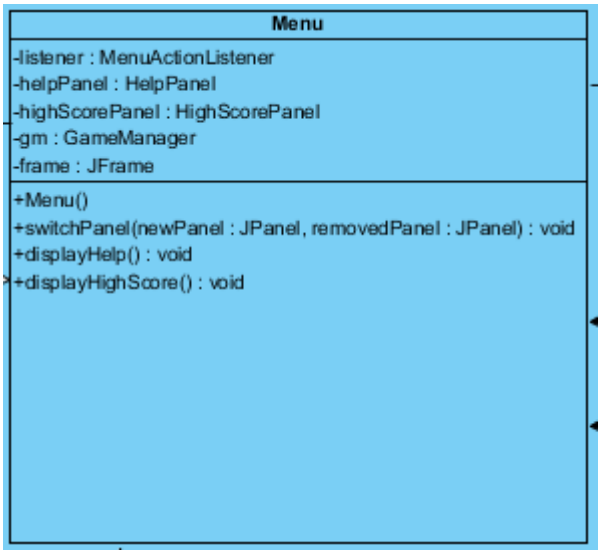

Figure 30 Menu Class

Attributes

privateJFrame frame: This frame will show all the visual context.

privateMenuActionListener listener: This attribute gets the user input from the graphical user interface.

privateHelpPanelhelpPanel: This is a JPanel type property of HelpPanel class that is used in graphical user interface and shows Help Menu on screen.

- This is constructed by HelpPanel class with its components like buttons, labels etc…

privateGameManagergm: GameManager is property of Menu class that provides reference to GameManagement subsystem, when Play Game button.

Constructors:

public Menu: Initializes *changeSettingsPanel, creditsPanel, helpPanel, gm and listener*properties.

Methods:

public void switchPanel(newPanel,removedPanel): It switches between panels on frame according to received selection from game menu.

public void displayHelp(): Includes switchPanel() method besides adding *helpPanel*to frame by replacing the existing panel on frame.

public void displaySettings(): Includes switchPanel() method and adds *changeSettingsPanel*to frame by replacing the existing panel on frame.

ScreenManager Class

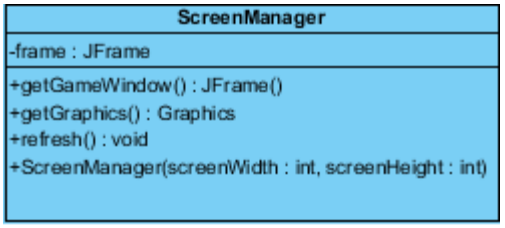

Figure 31 ScreenManager Class

Attributes:

privateJFrame frame: This is the main frame of the program that displays all the visual contexts of the software.

Constructor:

publicScreenManager(intscreenWidth, intscreenHeight): Get methods of height and width as parameter to construct a frame object.

Methods:

publicJFramegetGameWindow(): returns a JFrame object to display game screen.

public Graphics getGraphics(): returns a set of graphics for drawing to an off-screen image.

public void refresh(): It reconstructs the components in game frame.

MainMenuClass

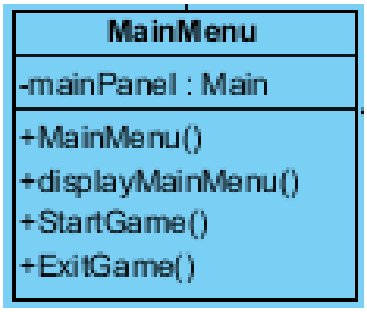

Figure 32 MainMenu Class

Attributes:

privateMainMenuMainPanelmainPanel: This MainMenuMainPanel type attribute of MainMenu class is used in graphical user interface to show Main Menu on screen.

Constructor:

publicMainMenu(): It initializes instances of MainMenu object.

Methods:

public void displayMainMenu(): This method includes switchPanel() method and adds *mainPanel*to frame by replacing the existing panel on frame.

public void startGame(): By the reference of *gm* attribute of MainMenu class (gm attribute is inherited from Menu class) this method invokes gameManagement subsystem to control gameplay routine.

public void exitGame(): This method closes the application.

- The methods that are inherited from parent Menu class.
  - switchPanel(newPanel : JPanel, removedPanel : JPanel),
  - displayHighScore()
  - displayHelp()

MenuActionListener

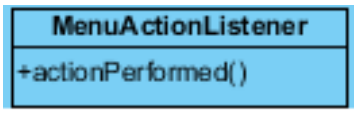

Figure 33 MenuActionListener

Methods:

public void actionPerformed(ActionEvent e): This method replaces actionPerformed method of ActionListener interface.

- HighScorePanel, HelpPanel, MainMenuMainPanel classes are not described in detail. These classes instantiate requested panels and these panels are placed on frame by MenuActionListener class.
-

4.2.2 Game Management Subsystem Interface

This subsystem is in the controller layer of our system. It has 5 classes. In this subsystem, our Façade class is GameManager class which performs the proper actions on game such as the interaction between objects and the effects on the current map. Since we have the Façade design pattern it is easy to use this subsystem by any other subsystem since the coupling between the subsystems is minimized. In figure 34 Game Management subsystem is visualized. Each class of this subsystem will be explained in detail in this section.

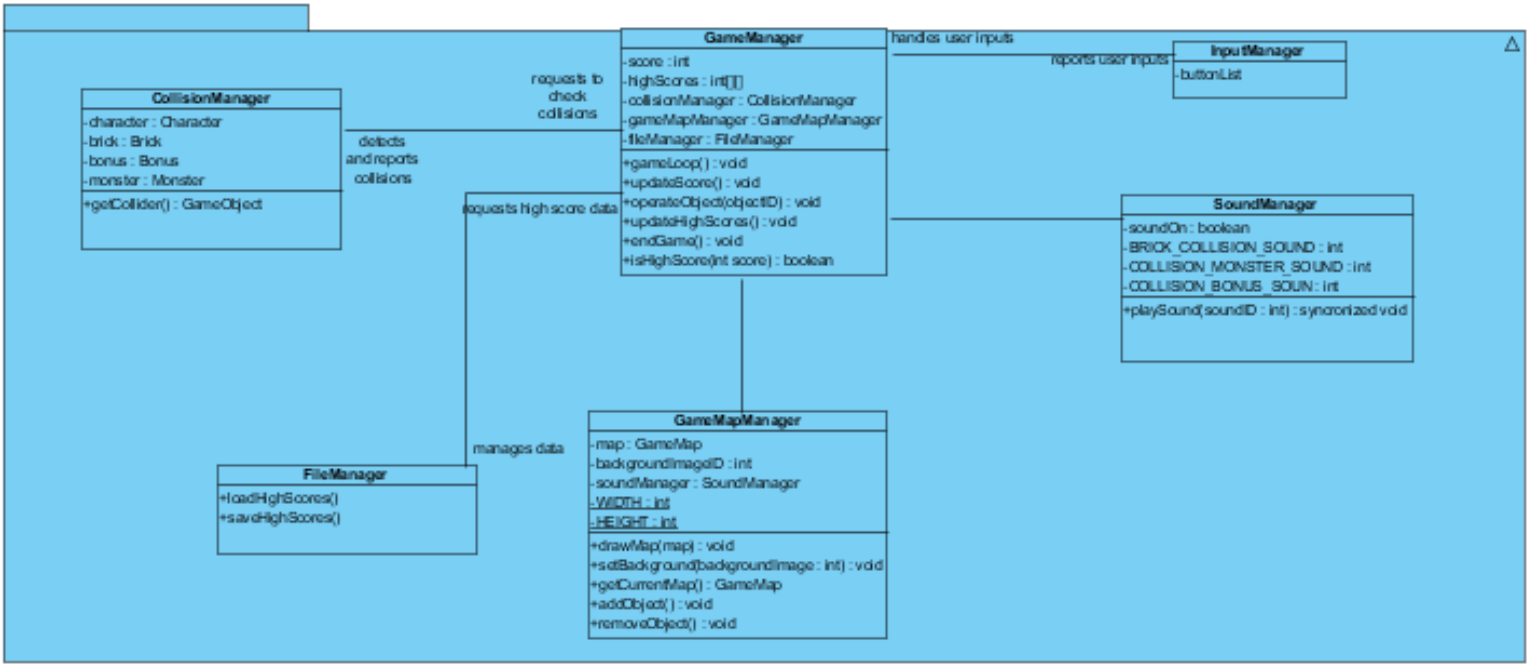

Figure 34 Game Management Subsystem

GameManagerClass

Figure 35 GameManager Class

Attributes:

privateint score: This attribute keeps the score of the player.

privateint [][]highScores: This attribute keeps the current high score list with names.

privateCollisionManagercollisionManager: This attribute is needed to reach to CollisionManager class.

privateGameMapmanagermapManager: This attribute is needed to reach to GameMapManager class.

privateFileManagerfileManager: This attribute is needed to reach to FileManager class.

Constructor:

publicGameManager(): It initializes the attributes of the GameManager for the first run of the system.

Methods:

public void gameLoop(): This method runs loop where the updates' of the game is done. This loop continues until the player loses or wants to exit from the game.

public void updateScore(): When player gains a bonus or passes a brick, this function increases the score of the player.

public void operateObject(intobjectID): When character encounters a bonus object or monster object, this function operates its effects on the character.

public void updateHighScores(): If the player's score is enough to be in high score list, this method gets the current list from file and updates the order of the list.

public void endGame(): This method terminates the game and returns the player to the Main Menu.

publicbooleanisHighScore(int score): This method checks whether the score of the player is enough to be in high score list.

InputManager Class

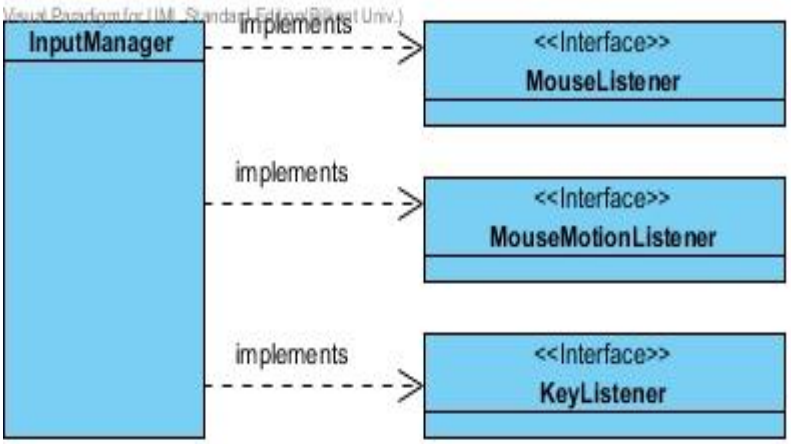

Figure 36 InputManager Class

This class is designed to detect the user actions performed by mouse(to make selection from the Main Menu.), and also performed by keyboard(to direct the character during the game). In this context, this class implements proper interfaces of Java.

SoundManagerClass

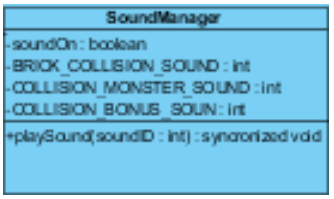

Figure 37 SoundManager Class

Attributes:

private Boolean soundOn: This attribute keeps the setting of sound.

private static int BRICK_COLLISION_SOUND: This is the id of brick collision sound.

private static int COLLISION_MONSTER_SOUND: This is the id of monster collision sound.

private static int COLLISION_BONUS: This is the id of bonus collision sound.

Constructor:

publicSoundManager(): Initializes the attributes of the SoundManager class.

Methods:

public synchronized voidplaySound(intsoundID): Plays the sound which is specified by soundID.

GameMapManagerClass

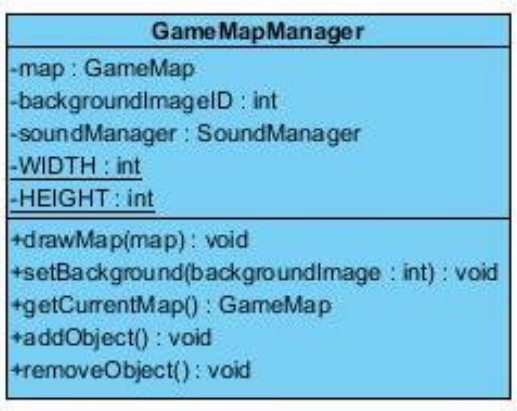

Figure 38 GameMapManager Class

Attributes:

privateGameMap map: This class performs the map specific operations by referencing to this object.

privateintbackgroundID: This is the id of the background image as it name refers.

privateSoundManagersoundManager:GameMapManager class changes the sound in background according to actions and events during the game by referring this object.

private final static int WIDTH: This is a final static attribute which is the width of the game map.

private final static int HEIGHT: This is a final static attribute which is the height of the game map.

Constructor:

publicGameMap(): initializes the attributes of GameMapManager with default values.

Methods:

public void drawMap(GameMap map): draws the current map with given attributes.

public void setBackgroundImage(intbackgroundImageID): sets the background image according to its id.

publicGameMapgetCurrentMap(): returns the current map which is processing by GameMapManager class.

public void addObject(intobjectID):While game continues, the game screen will slide and GameMapManager class will add new objects to the game map with this method.

public void removeObject(intobjectID): After the operation of an encountered object is completed, the encountered object is removed from the game by this function. In addition to this, this method is needed while game continues and the screen is sliding. The objects which are not in the scope of the current map anymore will be removed from the map.

CollisionManager class

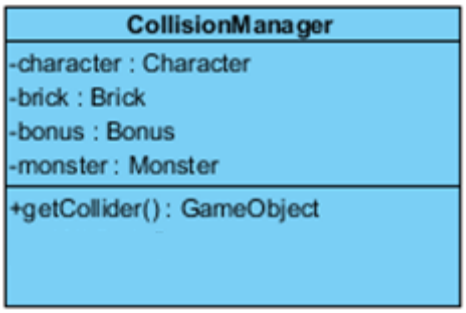

Figure 39 CollisionManager Class

Attributes:

private Character character: this class checks for the collision between character object and with other components in the game map.

private Brick brick: this class checks the collision between a brick and the character.

private Bonus bonus: this class checks the collision between a bonus and the character.

private Monster monster: this class checks the collision between a monster and the character.

Constructor:

publicCollisionManager(): initialized the attributes of the CollisionManager class with default values.

Methods:

publicGameObjectgetCollider(): This method returns null if there isn't any collision. If there is a collision, it returns the GameObject which is collided with the character.

4.2.3 Game Entities Subsystem Interface

This subsystem holds the domain specific objects of our system. It has 13 classes. Our game objects inherit from an abstract base class called GameObject. In this subsystem, our Façade class is GameMap class which performs the proper actions on game entities such as creating and modifying them. Since we have the Façade design pattern it is easy to use this subsystem by any other subsystem since the coupling between the subsystems is minimized. In figure 40 Game Entities subsystem is visualized. Each class of this subsystem will be explained in detail in this section.

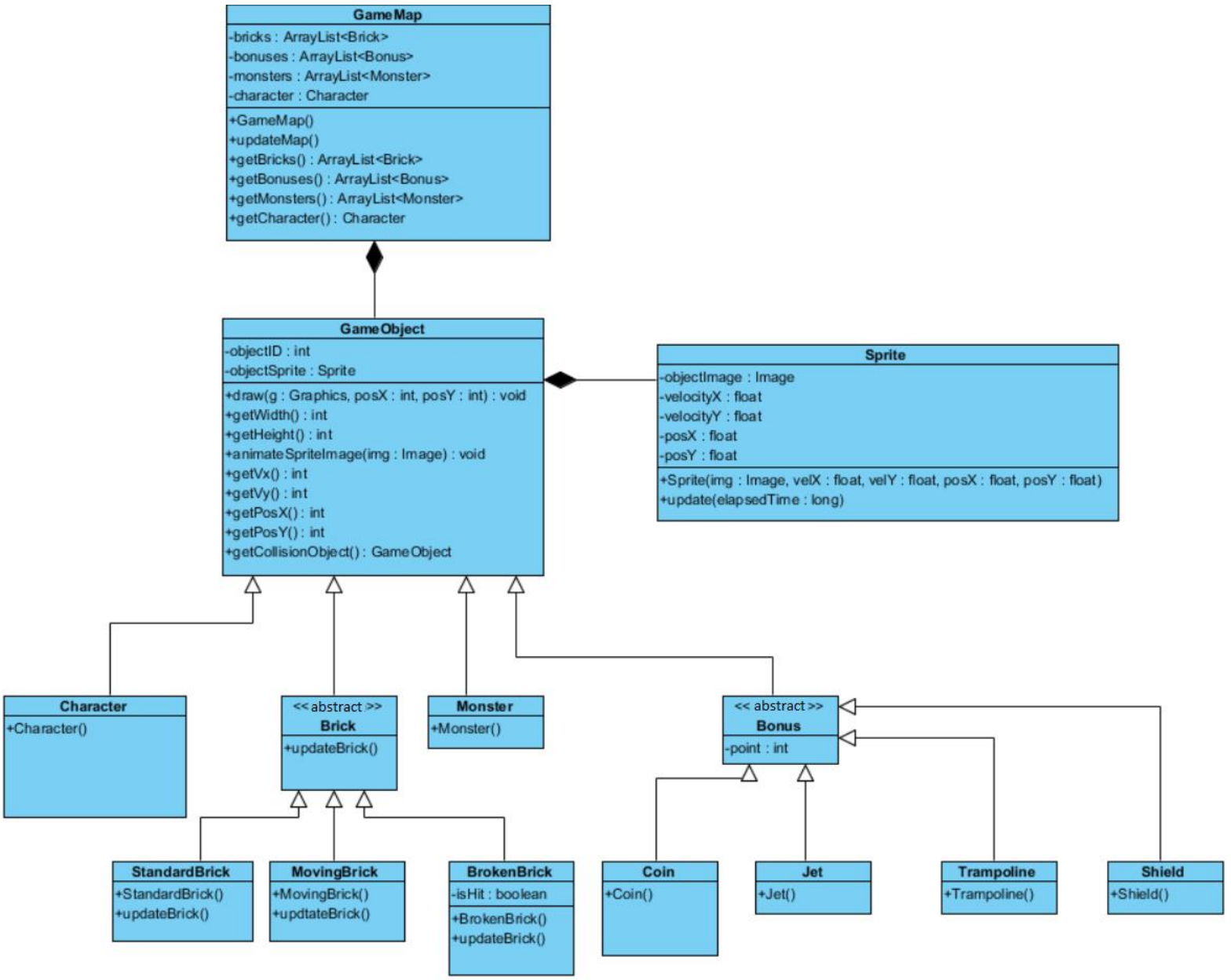

Figure 40 Game Entities Subsystem

GameMap Class

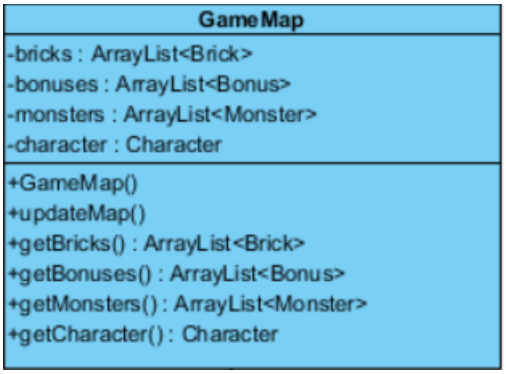

Figure 41 GameMap Class

- This class is designed to be the Façade class of the GameEntitiessubsystem, therefore it includes methods related to creating, modifying and deleting the game entities. Communication between the other components of the GameEntities subsystem is performed by this class

Attributes:

privateArrayList<Brick> bricks: this attribute represents the bricks that will be shown in the map.

privateArayList<Bonus>() bonuses: holds the reference to the all Bonus objects exists on the map.

privateArrayList<Monster>() monsters: holds references to all Monster objects on the map

private Character character: holds the reference to the Character in the map.

Constructors:

publicGameMap(): constructs a game map using character, bricks and bonuses with default attribute values.

Methods:

publicArrayList<Brick>getBricks(): returns the bircksarraylist.

publicArrayList<Bonus>getBonuses(): returns the bonuses arraylist.

publicArrayList<Monster>getMonsters(): returns the monsters arraylist.

public Character getCharacter(): returns the character object.

public void updateMap(): This method calls all add and remove methods that are specified above and according to the results of them, i.e. using the bricks, bonuses, monsters arraylists and character object, repaints the map. It will be called frame per second.

Sprite Class

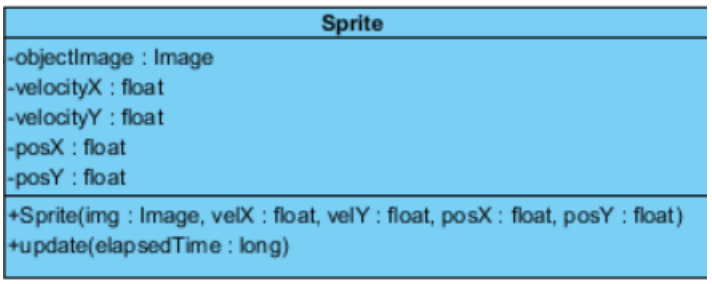

Figure 42 Sprite Class

- This class is just for updating the position of an Image object by providing velocity values for x and y directions, and updating its location by multiplying the velocity values by elapsed time in update method of this class.

Attributes:

private Image objectImage: image object whose position will be updated according to other attributes of the class

private float velocityX: X direction velocity value.

private float velocityY: Y direction velocity value.

private float posX:X position of theimage.

private float posY: Y position of the image.

Constructors:

public Sprite(Image img, intvelX, intvelY, float posX, float posY):creates a Sprite object with given attributes.

Methods:

private void update(long elapsedTime): if charachter is below the midpoint of the map , it updates the posX and posY attributes of the img attribute by multiplying the velX and velY values by elapsed time. Otherwise, makes character stay at midpoint of the map and gives

reverse velocity to all other objects. These are valid for Y-dimension. X-dimension works fine anyways.

GameObject Class

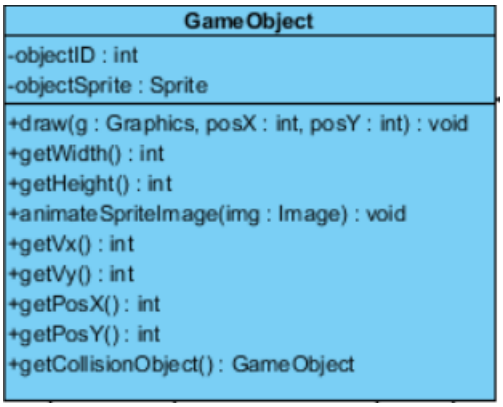

Figure 43 GameObject Class

- Our aim is to change the states of the objects as follows: with every state change we change the image of the game objects therefore we map these states as ids to load the proper image.

Attributes:

privateObjectSpriteobjectSprite: This attribute of Game Object class is an instance of Sprite class.

privateintobjectID: objectID attribute of Game Object class is used by some of child classes of Game Object class to specify the type of other objects of child classes.

Methods:

public void draw (Graphics g, intposX , intposY): This draw methods basically provides each GameObject to draw itself. For instance, a brick draws itself by using an instance of Graphics Class and two integers which indicate coordinates on the screen.

publicintgetWidth ( ) : This getWidth method returns the width of the image of the game objects, by referencing objectSprite attribute..

publicintgetHeight(): This getHeight method returns the height of the image of the game objects, by referencing objectSprite attribute.

publicintgetVx (): This getVx method returns the velocityX attribute by referencing the objectSprite attribute.

publicintgetVy(): This getVy method returns the velocityY attribute by referencing the objectSprite attribute.

publicintPosX (): This posX method returns the posX attribute by referencing the objectSprite attribute.

publicintPosY(): This posY method returns the posY attribute by referencing the objectSprite attribute.

public void animateSpriteImage (Image img): This method animates the img by using other images that are tied to it.

publicGameObjectgetCollisionObject( ): This method returns the object that the character collides with.

Character Class

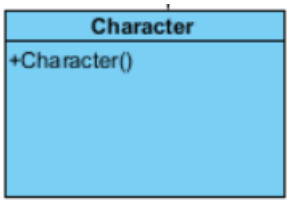

Figure 44 Character Class

Constructors:

public Character(): This constructor of Character class creates an instance of Character object. It takes no parameters since the Character instance is created once and it is at the beginning of the game and this point is already fixed, there is no need to take position variables.

Brick Class

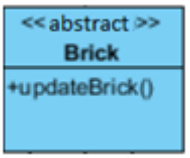

Figure 45 Brick Class

This Brick class is a parent class for all Brick types.

Methods:

public void updateBrick(): this method is called when there is a collision between character.

StandardBrick Class

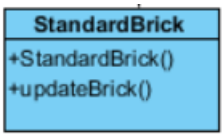

Figure 46 StandarBrick Class

Constructors:

publicStandardBrick(): This constructor of StandardBrick class creates an instance of a StandardBrick.

Methods:

public void updateBrick(): This method does nothing for this type of Brick.

MovingBrick Class

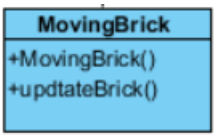

Figure 47 MovingBrick Class

Constructors:

publicMovingBrick(): This constructor of MovingBrick class creates an instance of a MovingBrick.

Methods:

public void updateBrick(): This method updates the X- position of an instance.

BrokenBrick Class

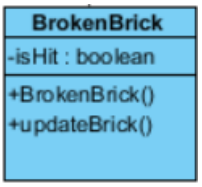

Figure 48 BrokenBrick Class

Attributes:

booleanisHit: this attribute is false at the start and when there is a hit, it will turn to true and it will be used to initiate the method updateBick().

Constructors:

publicBrokenBrick(): This constructor of BrokenBrick class creates an instance of a BrokenBrick.

Methods:

public void updateBrick(): This method updates the remaining time of the BrokenBrick instance after there is a hit.

Bonus Class

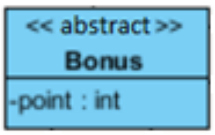

Figure 49 Bonus Class

This class is an Abstract Class for all Bonus types.

Attributes:

privateint point: this attribute is different for all Bonus types and it resembles the point that the character gains when having bonus.

Coin Class

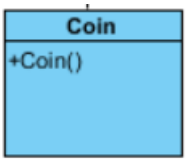

Figure 50 Coin Class

Constructors:

public Coin(): This constructor of Coin class creates an instance of a Coin.

Jet Class

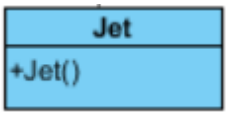

Figure 51 Jet Class

Constructors:

public Jet(): This constructor of Jet class creates an instance of a Jet.

Trampoline Class

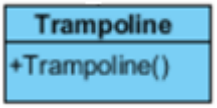

Figure 52 Trampoline Class

Constructors:

public Trampoline(): This constructor of Trampoline class creates an instance of a Trampoline.

Shield Class

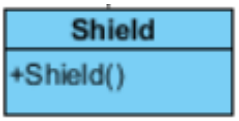

Figure 53 Shield Class

Constructors:

public Shield(): This constructor of Shield class creates an instance of a Shield.

4.3 Architectural Patterns

4.3.1 Layers

In our system decomposition we decomposed the system into three layers, User Interface, Game Controller and Game Entities. These are decomposed as hierarchical each of them having the related subsystems. Our top layer User Interface is the most complicated one and it includes much hierarchy since it is not used by any other layer above and it is responsible for the interaction with the user. Following layer is Game Controller layer; in this layer all the game logic is controlled. Our bottom layer is Game Entities layer, in which all the necessary entity objects are brought together. Our layer decomposition also proposes the closed architectural style, in which a layer can only access to the layer below it. Our layer decomposition is depicted in figure 54 below.

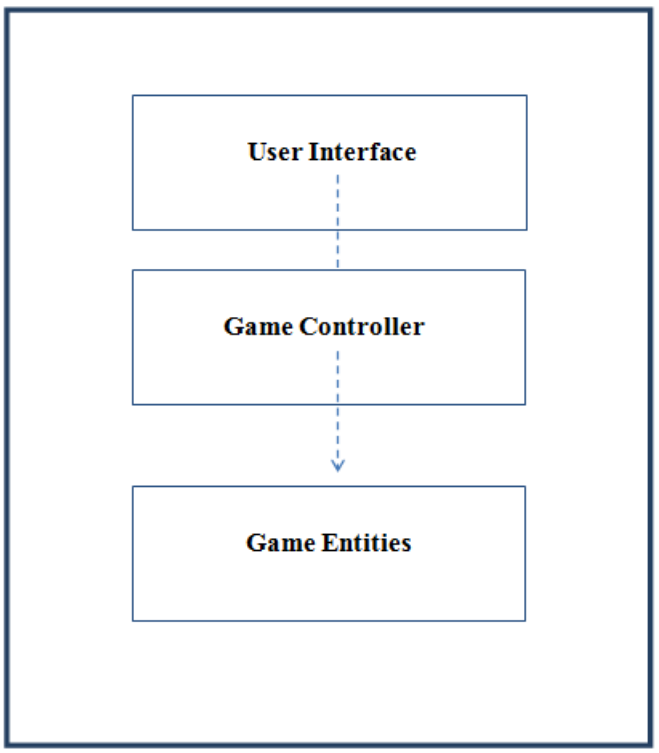

Figure 54 Layers of System

4.3.2 Model View Controller

In this architectural style, the purpose is classifying the subsystems into three parts, called model, view and controller. By dividing the subsystems into three parts, we isolate the game objects from the user by adding a controller part between them. In our system, we grouped our game objects into game entities layer which is the model of our system. The game objects of our system is only accessed and controlled by manager classes which are grouped under Game Controller layer that is the controller part. We grouped the classes which are responsible for providing the interaction between user and system into User Interface layer, this layer is the View part since it just communicates with the model part via controller part. By this architecture it is achieved that changes on the interfaces do not change the model of the system, therefore it is a good choice to use MVC for games.

4.4 Hardware Software Mapping

The programming language that will be used for implementing the core design of our game project is JAVA. We will use JDK and J2SE Platforms to develop the program and corresponding JRE package will be required to run our Java application on Windows. With the help of platform dependency feature of java, the game will be run on any computer with basic software are installed. OOJUMP game runs on one PC at a time. There will be no multiplayer option and no network connection feature.

For data storage issue, we will use text file (.txt). We don't keep large scaled data; we'll just keep the high scores. Also we just need to write and read the data rather than work on these data, we prefer to use text file to storage.

In terms of hardware, our system needs keyboard, mouse and speaker. With the direction keys on the keyboard, user can pass his/her inputs to the system during the game. With mouse user can transmit his/her inputs from the Main Menu (PlayGame/ Help/ HighScores). During the game, according to the activities different sounds will be played background. The deployment diagram in figure 55 shows the relation between the hardware components. The node PC with OOJump component is the hardware device that the program is run on. Its component OOJump is composed of 3 different components which are shown in figure 56 component diagram. Keyboard and mouse are the decives where the input providers to PC. During the game, different sounds will be provided throughout the speaker device. Their relations are shown in figure 55.

Component diagram in figure 56 shows the components of OOJump game. The subsystems of the game are components of OOJump. The contents of these subsystems are explained in detail in section 4.2 Subsystem Decomposition.

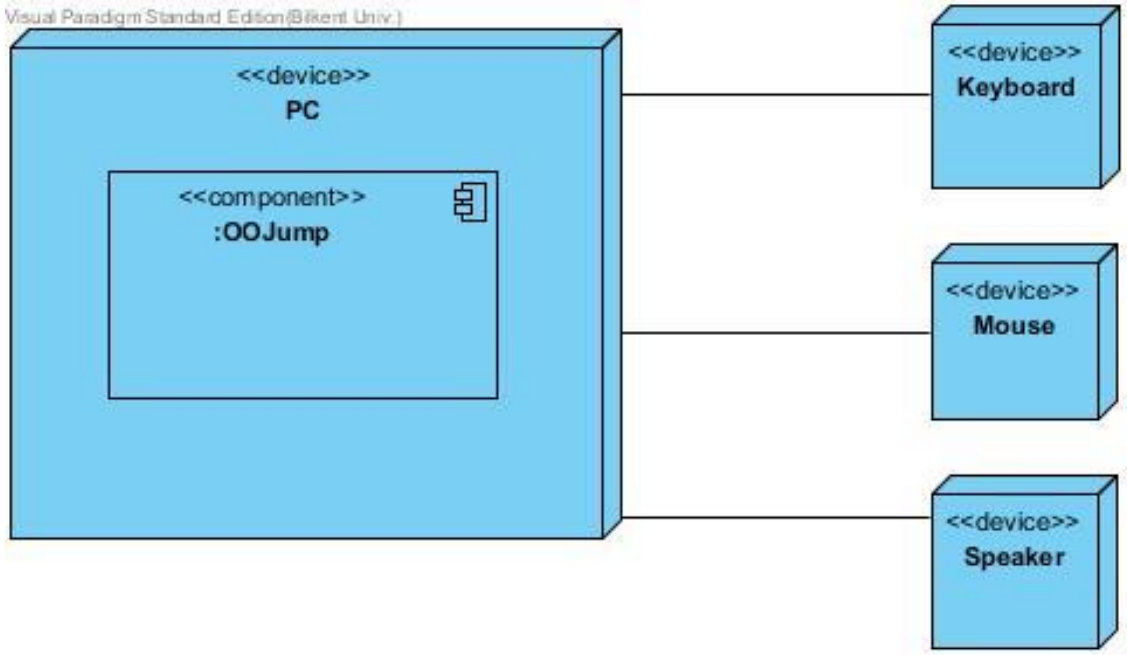

Figure 55 Deployment Diagram

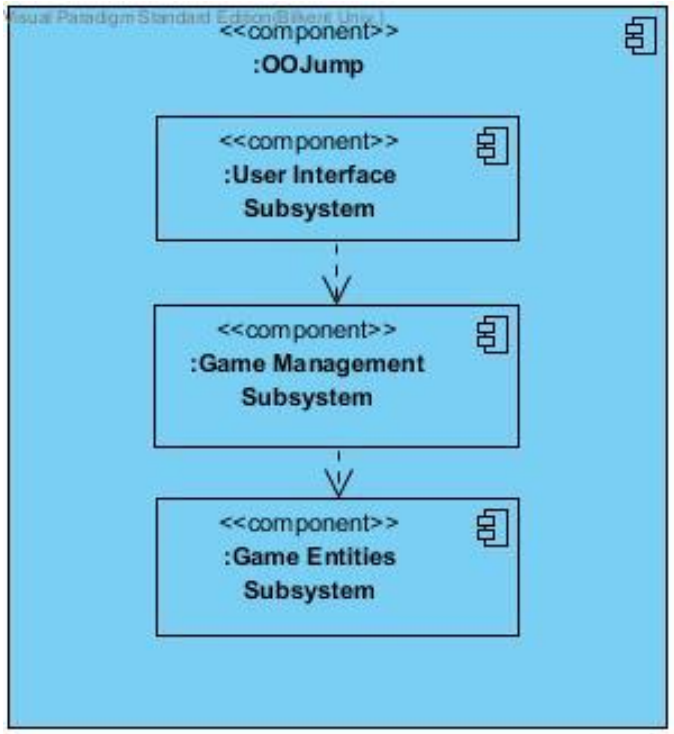

Figure 56 Component Diagram

4.5 Addressing Key Concerns

4.5.1 Persistent Data Management

Our game doesn't need a database system as the map is randomly created when the player achieves to go further. Therefore, there is no possibility that the map will be corrupted. High Score table will be saved as a txt file to the hard disk drive. Since the game is not online and every player stores the high scores that are done only at that computer, txt file will be stable, i.e. readable and writable by one system. We also plan to store sound effects and game object images in hard disk drive with proper and simple sound, image formats such as .gif, .waw.

4.5.2 Access Control and Security

Our game OOJUMP will not require any kind of network connection or user authentication at all. There won't be a need to create a profile, and even it's not possible to create a profile. The ones, who have oojump.jar file in their computer that runs JAVA, can play the game. Therefore, there will be no kind of security issues in OOJUMP.

4.5.3 Global Software Control

Our system has event driven software control. A main loop waiting for an event and whenever there is an event, it's dispatched to the appropriate objects and takes action specified to the event.

4.5.4 Boundary Conditions

4.5.4.1 Initialization

Since OOJump does not require an installation because it will be run on executable .jar file rather than .exe or such extension.

4.5.4.2 Termination

OOJump can be terminated / closed by clicking “Quit Game” button in Main Menu. Since the system doesn’t provide a pause screen during the game, if player wants to quit during game play, the player should press “x” button on top of the game screen. When player presses the “x” button, system redirects the player to Main Menu.

4.5.4.3 Error

If an error occursin which game resources could not be loaded such as images, the system gives an error message and tries to reload. However if an error occurs during the load of sound, the game won’t give an error message to the user and the game will continue without sound.

If program did not respond because of a performance issue, even we try to keep these kinds of problems minimum, Player will lose all of current data.

5. 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 fulfill 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 state chart 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.