



**BILKENT UNIVERSITY**

**CS 319: OBJECT-ORIENTED SOFTWARE ENGINEERING**

**DESIGN REPORT**

**DRAW IT!**

**Group 4 Section 1**

Görkem Çamlı - 21302603

Oğuz Bilgener - 21302523

Umut Cem Soyulmaz - 21201744

Hilal Öztürk - 21000633

Course Instructor: Hüseyin Özgür TAN

Teaching Assistant: Fatma BALCI

## **Table Of Contents**

<b>1. Introduction (Problem Statement).....</b>	<b>5</b>
<b>2. Requirement Analysis.....</b>	<b>5</b>
<b>2.1. Overview.....</b>	<b>5</b>
<b>2.2. Functional Requirements.....</b>	<b>7</b>
<b>2.3. Non-Functional Requirements.....</b>	<b>7</b>
<b>2.4. Constraints.....</b>	<b>7</b>
<b>2.5. Scenarios.....</b>	<b>8</b>
<b>2.6. Use Case Models.....</b>	<b>10</b>
<b>2.6.1. Use Case 1: StartGame.....</b>	<b>11</b>
<b>2.6.2. Use Case 2: ChooseWord.....</b>	<b>12</b>
<b>2.6.3. Use Case 3: Draw.....</b>	<b>13</b>
<b>2.6.4. Use Case 4: ViewAndGuess.....</b>	<b>14</b>
<b>2.6.5. Use Case 5: SwitchRoles.....</b>	<b>15</b>
<b>2.6.6. Use Case 6: ViewCredits.....</b>	<b>16</b>
<b>2.7. User Interface.....</b>	<b>17</b>
<b>3. Analysis.....</b>	<b>21</b>
<b>3.1. Object Model.....</b>	<b>21</b>

3.1.1. Domain Lexicon.....	21
3.1.2. Class Diagram(s).....	22
3.2. Dynamic Models.....	24
3.2.1. State Chart.....	24
3.2.2. Sequence Diagram.....	27
3.2.2.1. Scenario 1: “menu” Scenario.....	27
3.2.2.2. Scenario 2: “showCredits” Scenario.....	29
3.2.2.3. Scenario 3 “wordChoosing” Scenario.....	30
3.2.2.4. Scenario 4: “wordDrawing” Scenario.....	31
3.2.2.5. Scenario 5: “watchTheWord” Scenario.....	33
3.2.2.6. Scenario 6: “guessingWord” Scenario.....	34
3.2.2.7. Scenario 7: “roleExchanging” Scenario.....	36
4. Design.....	42
4.1 Design Goals.....	42
4.2 SubSystem Decomposition.....	44
4.2.1. User Interface Management Subsystem.....	45
4.2.2. Game Logic Subsystem.....	48
4.2.3. Network Management Subsystem.....	50

4.2.4. Storage Management Subsystem.....	52
4.3 Architectural Patterns.....	53
4.3.1. Model-View-Controller.....	53
4.3.2. Peer To Peer.....	54
4.3.3. Client-Server.....	54
4.4 Hardware/Software Mapping.....	55
4.5 Addressing Key Concerns.....	56
4.5.1 Persistent Data Management.....	56
4.5.2 Access Control and Security.....	57
4.5.3 Global Software Control.....	58
4.5.4 Boundary Conditions.....	58
4.5.5 Minimal Serval Design and Code Reusability.....	59
5. Conclusion.....	60

## Table Of Figures

<b>Image 1: Main Menu.....</b>	<b>20</b>
<b>Image 2: Login.....</b>	<b>21</b>
<b>Image 3: Host Game.....</b>	<b>21</b>
<b>Image 4: Join Game.....</b>	<b>22</b>
<b>Image 5: Choose Word Screen.....</b>	<b>22</b>
<b>Image 6: Drawing Screen.....</b>	<b>23</b>
<b>Image 7: Guessing Screen.....</b>	<b>23</b>
<b>Image 8: End Of Round.....</b>	<b>24</b>
<b>Image 9: Credits.....</b>	<b>24</b>
<b>Figure 1. Use Case Diagram.....</b>	<b>12</b>
<b>Figure 2. Class Diagram.....</b>	<b>28</b>
<b>Figure 3. State Chart Diagram 1: Passive Player State.....</b>	<b>30</b>
<b>Figure 4. State Chart Diagram 2: Active Player State.....</b>	<b>31</b>

<b>Figure 5. Sequence Diagram 1: “menu” Diagram.....</b>	<b>33</b>
<b>Figure 6. Sequence Diagram 2: “showCredits” Diagram.....</b>	<b>34</b>
<b>Figure 7. Sequence Diagram 3: “wordChoosing” Diagram.....</b>	<b>35</b>
<b>Figure 8. Sequence Diagram 4: “wordDrawing” Diagram.....</b>	<b>37</b>
<b>Figure 9. Sequence Diagram 5: “watchTheWord” Diagram.....</b>	<b>38</b>
<b>Figure 10. Sequence Diagram 6: “guessingWord” Diagram.....</b>	<b>40</b>
<b>Figure 11. Sequence Diagram 7: “roleExchanging” Diagram.....</b>	<b>41</b>
<b>Figure 12. Subsystem Decomposition.....</b>	<b>44</b>
<b>Figure 13. User Interface Management Subsystem.....</b>	<b>47</b>
<b>Figure 14. Game Logic Subsystem.....</b>	<b>49</b>
<b>Figure 15. Network Management Subsystem.....</b>	<b>51</b>
<b>Figure 16. Storage Management Subsystem.....</b>	<b>52</b>
<b>Figure 17. Component Diagram.....</b>	<b>55</b>
<b>Figure 18. Deployment Diagram.....</b>	<b>56</b>

## **1. Introduction - Problem Statement**

Few years ago, one of the most popular games was a game called draw the word. It was also part of TABU game and its concept is very basic. A person chooses a word from the card and tries to draw the word within a limited time. The other person tries to guess it within his/her time limit. A web-based and mobile based application of this game is played too much. However, we notice that this game never been a desktop-application game. Therefore, we as Group 4 decided to make one. “Draw It!” is an entertaining application which has lots of common points with famous “Draw Something” game. The “Draw It!” is a multiplayer game and its concept is mainly constructed on drawing the given words and guessing the drawn images in a particular time to gain points.

### **Problem Statement**

To be specific, “Draw It!” game is played with 2 players in multiple rounds. Each player turns changes within each round. But first they need to connect to the game. Host is opens the game and guest is the player who connects the game. Host is the first active player, who is responsible for choosing the word and drawing it. Passive player tries to find the word. They both have limited times for their tasks. Active player can hit finish button if s/he finishes drawing earlier or s/he can hit the x button and give up his turn. Same way, passive player can give up his turn by clicking ? button. When guessing part finishes, round finishes as well and players change the roles. This game circle continues right until one of the player clicks on the exit button.

During this report, general pre-analysis of the project are going to be investigated. In the first part, the main point is to clearly indicate the features of gameplay including rules. During

the next step, list of requirements and their analysis are explained properly. Final part of the report includes scenarios, use case models, UML diagrams and user interface figures.

## **2.Requirement Analysis**

Requirement Analysis part contains an overview, requirements for the application, scenarios, Use Case Models and the Use Case Interface of “Draw It!” application.

### **2.1. Overview**

“Draw It!” is a Java based computer game which requires a connection between computers. By using this connection, when a player is drawing an image, the other player is able to see the drawing at the same time. In this concept, the aim of the game is to reach the maximum points by drawing and guessing accurately the challenges. The game is played with rounds and at the end of every round, the duties of the players are replaced. During the drawing process, users are able to change the colour and size of the drawing stick to make it more understandable for the player who is guessing.

The “Draw It!” game consists of three main steps. In the first step, there is a menu with three buttons which are “Host”, “Join”, “Credits”. One of the users clicks on the “Host” button to create a new game and the only thing that the second player needs to click on the “Join” button to start the challenge. When the game is started, in the first round host player is the one who is going to draw a figure and the guest player is the one who guesses. Before starting to draw, active player, whose duty is to draw, chooses a word among three random words that the game offers. When the drawer player starts to create an image, the passive player, whose duty is to view and guess, is able to see the drawing in real time. The drawing process needs to be done in



45 seconds and after this 45 seconds, the guessing time period is started. The passive player has also 45 seconds to find the name of the object by filling the blank with letters that are located at the bottom of screen. If the passive player finds the word, the time period is over and passive player gets ten points but if they cannot make it, s/he gets zero points. After every round, the active player becomes passive player and the passive player becomes active player. The game continues until one of the players clicks on the finish button and when the game is finished, both of the players see their scores.

## **2.2. Functional Requirements**

- Players must be able to start or join a new game over a network.
- Players must be able to use and control their mouse during the game (while drawing and guessing).
- Players should change roles after each round.
- For each round, active player should be able to select his/her own word from the given words.
- Active user should be able to finish drawing whenever he wishes to.
- Passive user should be able to give up from his turn if s/he thinks s/he won't be able to find it.
- During the game, passive player should receive points according to his correct responses.
- Players must be able to exit from the game anytime they want to. In this case, game will be over for both players and the scores will be shown.
- Active player should be able to change the colour and the size of the brush.

- Players must be able to login or sign up to the game.

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

- Before starting the game, the connection between two players (host and guest) should not take more than 2 minutes long.
- Color panel at least should have blue, yellow, red and green colors.
- Brush must at least has 3 different size.
- 45 seconds should be given for both drawing and guessing parts of the round.
- Application should not allocate more than 1 GB of space.
- When passive player finds a word correctly, s/he should get 10 points.
- In the guessing box, 10 letters should be appear in order to help passive user to guess the word.
- Guessing box's color should turn red if the passive player fail to find the word, green otherwise.
- Players should be able to play game without further documentation.
- When a round crashes, system should be able to start a new round.
- Players should have unique usernames.

### **2.4. Constraints**

- The “Draw It!” game will be implemented in Java.
- The application should work in all operating systems.
- The application should work on the local network.

## 2.5. Scenarios

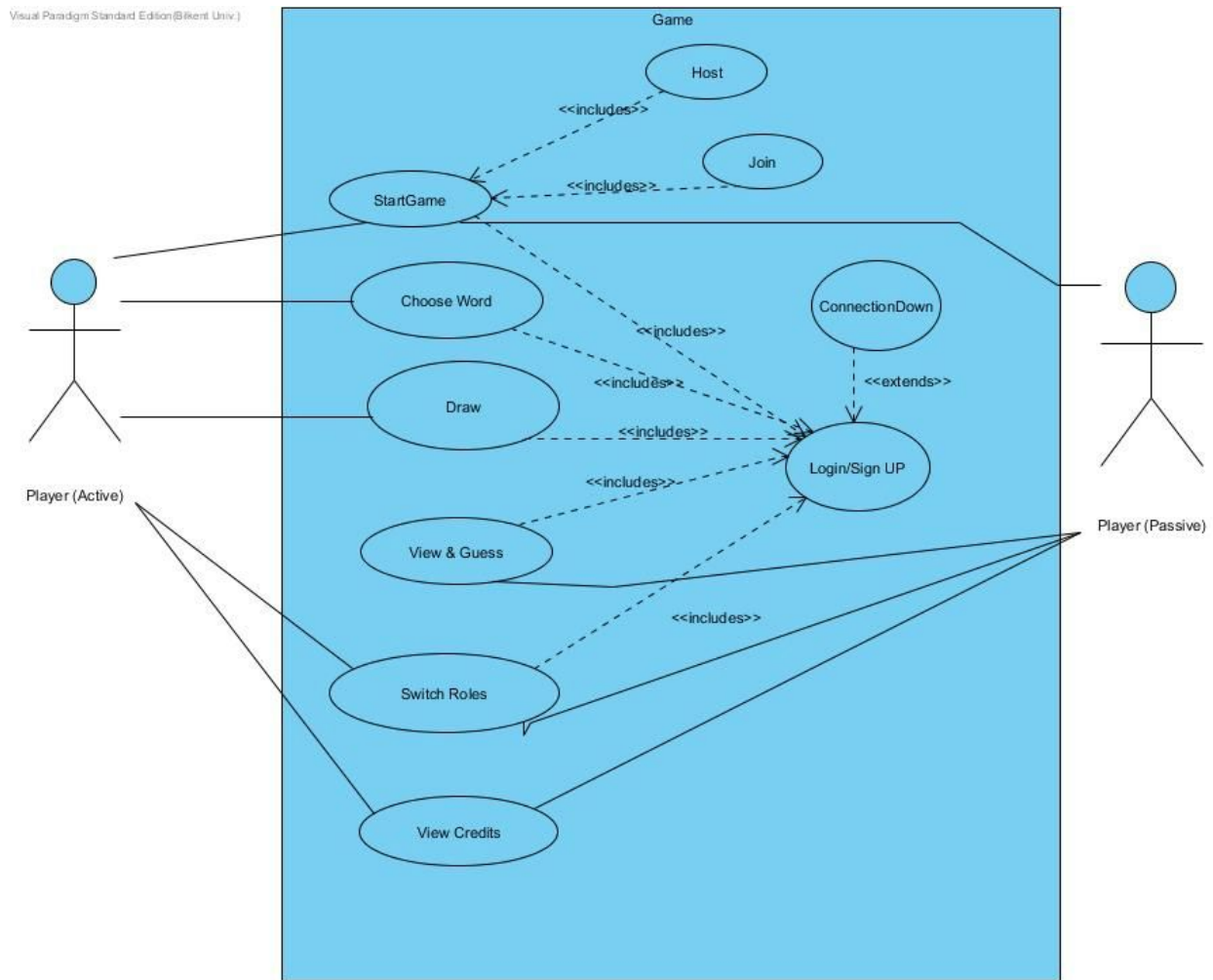
### **Participating Actors:** Player (John, Valerie)

John and Valerie are both very famous painters and they decide to meet in 18th September 2015 at the John's studio. After 6 pm, they get bored and want to try something new for them. Suddenly, John finds an application which is called as "Draw It!" in his personal computer and he asks Valerie to play this game together. Then, Valerie accepts the challenge and download the game for her own personal computer. John clicks on the icon of the game on his computer's desktop as well as Valerie. Both John and Valerie sign up for the game by declaring their username and password. They both login. John clicks on the "Host" button and waits and his username and password saved to the computer and his personal computer's IP address sent to the server network. Valerie clicks on the "Join" and she fills the blank space by entering the John's username to join his game. When the connection is provided, Valerie encounters with a waiting screen because at this moment John needs to choose a word among three randomly chosen words. After choosing a word from the list, John starts to draw the word and the game begins officially. When John is drawing the image, Valerie can see the mouse movements of John in real time by using the connection. Every time John releases the left mouse click after a drawing movement, the drawing piece for this period is seen by the Valerie at the same time. When John finishes his drawing, he can click on the "Check" option or he can wait until the time countdown hits to zero. If he cannot finish his drawing until end of the time limit (45 seconds), his turn passes. When he finishes his work before the time is up, entire image is seen on the screen of Valerie and the time countdown starts for her. During the guessing period, Valerie can fill the word space by clicking on the given letters in the guess box to find the word that John

chose. If Valerie thinks that she cannot find it, she can click on the “?” option to give up and her turn passes. In addition to this, if Valerie cannot find the word until the time is over, his turn passes as well. If she can find it before the time is up for her, she gains 10 points and the first round is over. At the end of the first round, John and Valerie changes their roles. For the second round, John becomes the passive player and Valerie becomes the one who draws. At the end of every round, game system changes the roles of the players. The game continues, until Valerie decides to go home to cook some meal for her children and she clicks on the “Stop Game” option. After finishing the game, they both sees their personal scores and Valerie wins the challenge. Before closing the application, John wonders the names of people who worked for the construction of this game and he goes back to the main menu to click on “Credits” to see the participants of the project.

## 2.6. Use Case Model

We intend to make our Use Case Diagram as simple as possible in order to be comprehensible by the client. Below is our Use Case Diagram and Use Cases.



**Figure 1. Use Case Diagram**

### ***2.6.1. Use Case 1: Login/Sign Up:***

***Use Case Name:*** Login/ Sign Up

***Participating Actors:*** Initiated by Player.

***Flow of Events:***

1. If player enters the game for the first time, he should sign up by entering his unique username and password.
2. System saves the users IP adress to the network and the username and password is saved onto the computer.
3. If it is not for the first time, when opening game.exe system automatically logs in.

***Entry Condition:***

- If for the first time, player should sign up the game, else player opens the game .exe.

***Exit Condition:***

- Player2 enters Player1 IP address.

***Quality Requirements:***

- Player's username should be unique.

Login/Sign up extends the connection Down use case. This is an exceptional case for the times when the connection fails. If fails, system tries to connect the network until the connection is successfull. Game continues with a new round.

### ***2.6.2. Use Case 2: StartGame***

Start game use case both includes Join and Host use cases. As they are very simple use cases, we explain it within the StartGame use case.

***Use Case Name:*** StartGame

***Participating Actors:*** Initiated by Player.  
Communicates with another Player.

#### ***Flow of Events:***

2. Player1 activates the “host” function of the main menu.
  2. Game shows the IP address of the Player1 in a new screen.
3. Player1 starts to wait until Player2 enters his IP address, Player2 enters Player1’s IP address.
  4. Game opens choose word step to Player1, when Player2 is waiting for the first round to start.

#### ***Entry Condition:***

- Player1 clicks on the host button and Player2 clicks on the join button.

#### ***Exit Condition:***

- Player2 enters Player1 username.

#### ***Quality Requirements:***

- Player2 must click on the join button and must enter Player1’s IP address at most 2 minutes.

### ***2.6.3. Use Case 3: ChooseWord***

***Use Case Name:*** ChooseWord

***Participating Actors:*** Initiated by Player.

***Flow of Events:***

1. Game shows the three words in a new window to Player.
2. Player chooses one of these word to draw.
3. Game starts the round by opening new drawing screen.

***Entry Condition:***

- Second player joins the game.

***Exit Condition:***

- Player clicks on the word he chooses.

***Quality Requirements:***

- -



**2.6.4. Use Case 4:                      Draw**

**Use Case Name:**                      Draw

**Participating Actors:**                      Initiated by Player.

**Flow of Events:**

1. Game starts the countdown of the timer.
2. Active Player (player who draws) starts to draw the word with any color and size he selects.
3. Active Player continues to draw until time is up or he clicks on the finish button.
4. Game finishes active player's part in the round.

**Entry Condition:**

- Active Player clicks on one the three words.

**Exit Condition:**

- When time is up.
- When Active Player clicks on the finish button.

**Quality Requirements:**

- Required time is 45 seconds.

### **2.6.5. Use Case 5: ViewAndGuess**

**Use Case Name:** ViewAndGuess

**Participating Actors:** Initiated by Player.  
Communicates with another Player.

**Flow of Events:**

1. Passive Player (player who guesses) sees each drawing movement of the the active player with real-time.
2. After drawing process ends, game initializes Passive Player's guess time and makes visible the guess box.
3. Within the given time, Passive Player tries to find the chosen word. If he clicks on the wrong letter he click on the Trash Bin icon. One clicked icon erases the all chosen letters and passive player should start to select them again.
4. If he thinks that he can't find the word, he clicks on the "?" button. If he finds the word or if time is up, his turn passes.
5. If passive player fails to find the word, uses box turns into red. Correct word shown. If word found correctly, guess box turns into green and player receives 10 points.

**Entry Condition:**

- Active player releases the mouse for the first time whilst he draws.

**Exit Condition:**

- Time is up.
- Passive player finds the word correctly.
- Passive player gives up and clicks on the "?" button.

**Quality Requirements:**

- Required time is 45 seconds.

#### **2.6.6. Use Case 6: Switch Roles**

**Use Case Name:** SwitchRole

**Participating Actors:** Initiated by Game.

Communicates with the other two Players.

**Flow of Events:**

1. When each round is finished, scores of the players is updated by Game Controller.
2. If player 1 is active player for the finishing round, guessing window is sent to him or vice versa. If player 2 is active player for the finishing round, guessing window is sent to her or vice versa.

**Entry Condition:**

- Round is finished.

**Exit Condition:**

- Both active and passive players takes new screen.

**Quality Requirements:**

- -

### ***2.6.7. Use Case 7: ViewCredits***

***Use Case Name:*** View Credits

***Participating Actors:*** Active and passive players

***Flow of Events:***

1. Players use Credits button on the main menu.
2. Game displays the contributors of the game.

***Entry Condition:***

- Press on Credits button.

***Exit Condition:***

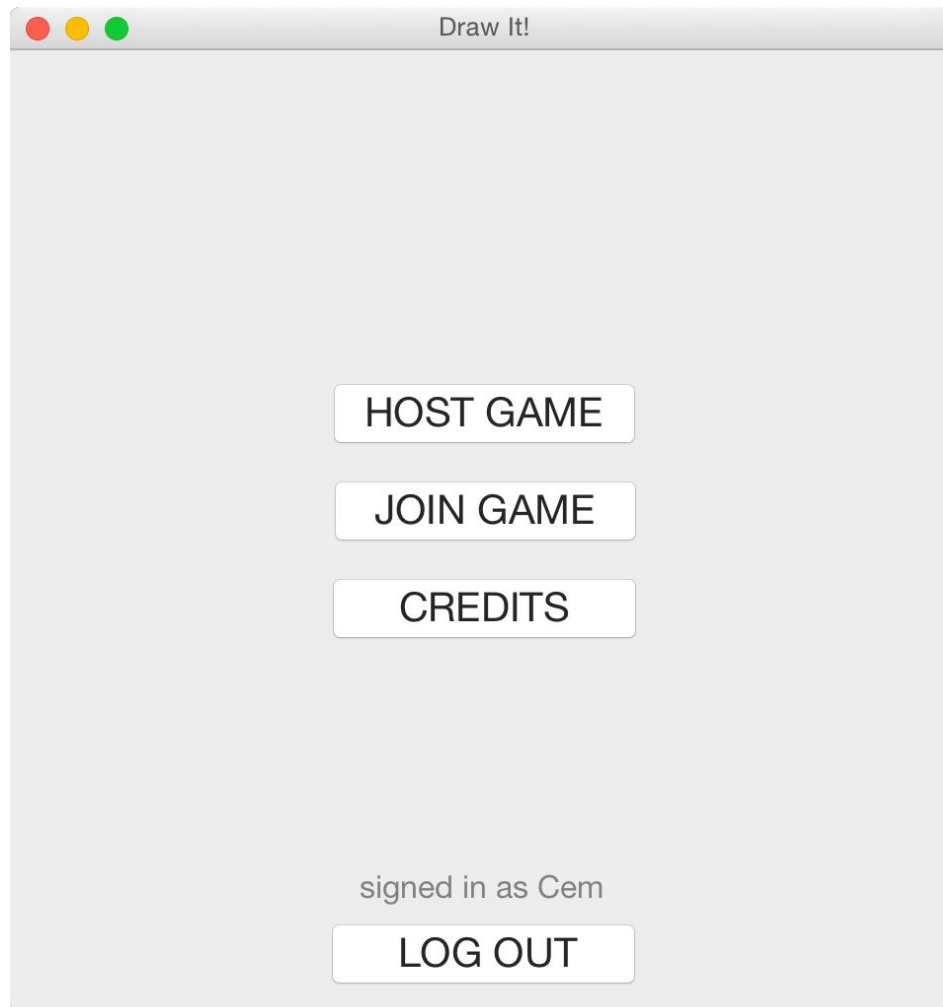
- Press “Back” button.

***Quality Requirements:***

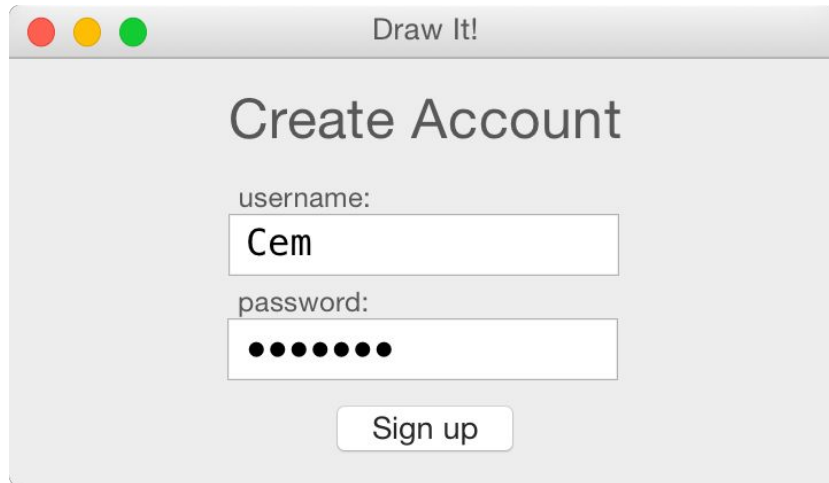
- -

## 2.7. User Interface

User Interface is one of the crucial thing in designing process. We care our application to be user-friendly. The images below are only rough sketches, for now.

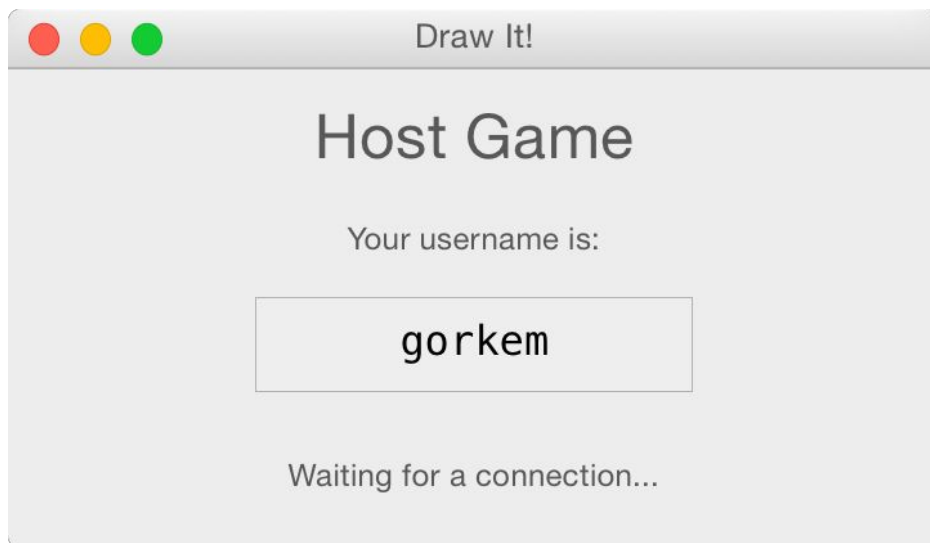


**Image 1: Main Menu**



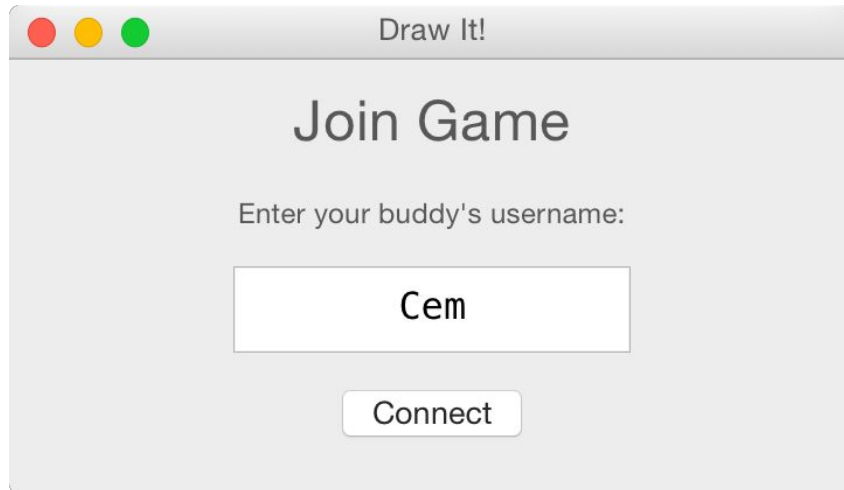
A screenshot of a macOS-style window titled "Draw It!". The window has a light gray background and a title bar with three colored window control buttons (red, yellow, green) on the left. The main content area features the heading "Create Account" in a large, dark gray font. Below the heading, there are two input fields. The first is labeled "username:" and contains the text "Cem". The second is labeled "password:" and contains seven black dots, indicating a masked password. Below these fields is a button labeled "Sign up".

**Image 2: Login**

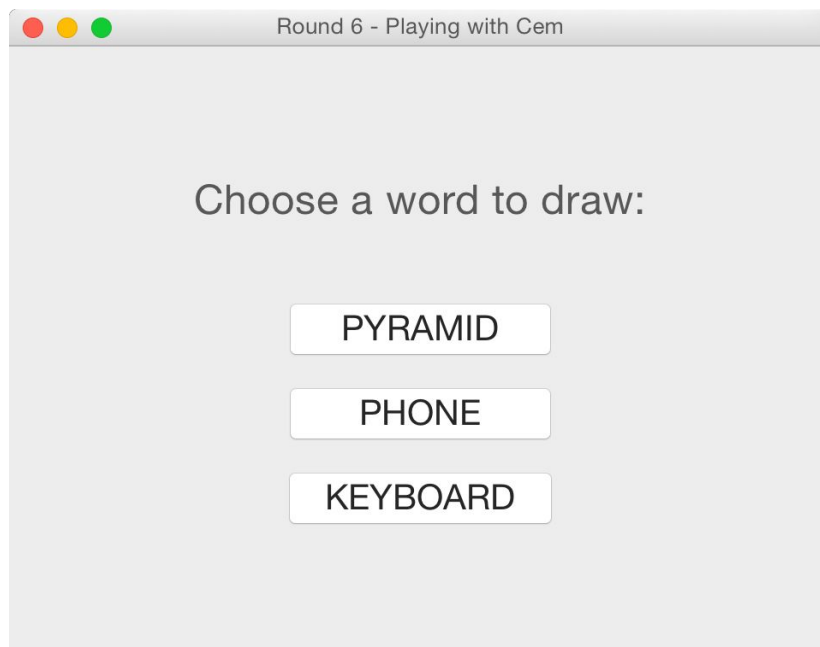


A screenshot of a macOS-style window titled "Draw It!". The window has a light gray background and a title bar with three colored window control buttons (red, yellow, green) on the left. The main content area features the heading "Host Game" in a large, dark gray font. Below the heading, the text "Your username is:" is displayed. Below this text is a rectangular box containing the text "gorkem". At the bottom of the window, the text "Waiting for a connection..." is displayed.

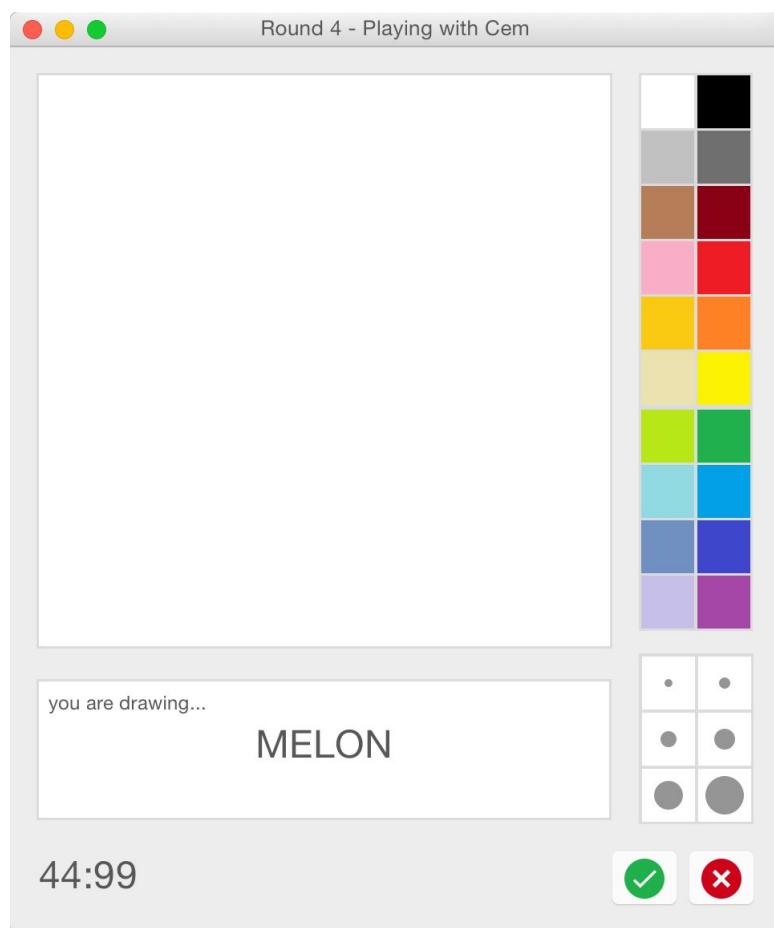
**Image 3: Host Game**



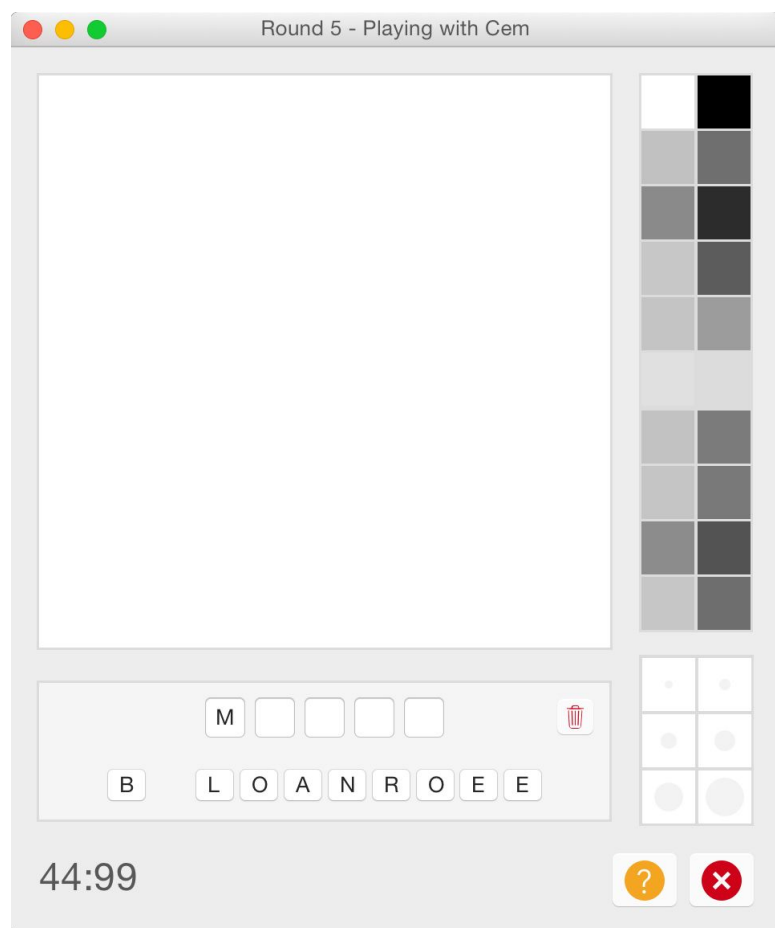
**Image 4: Join Game**



**Image 5: Choose Word Screen**

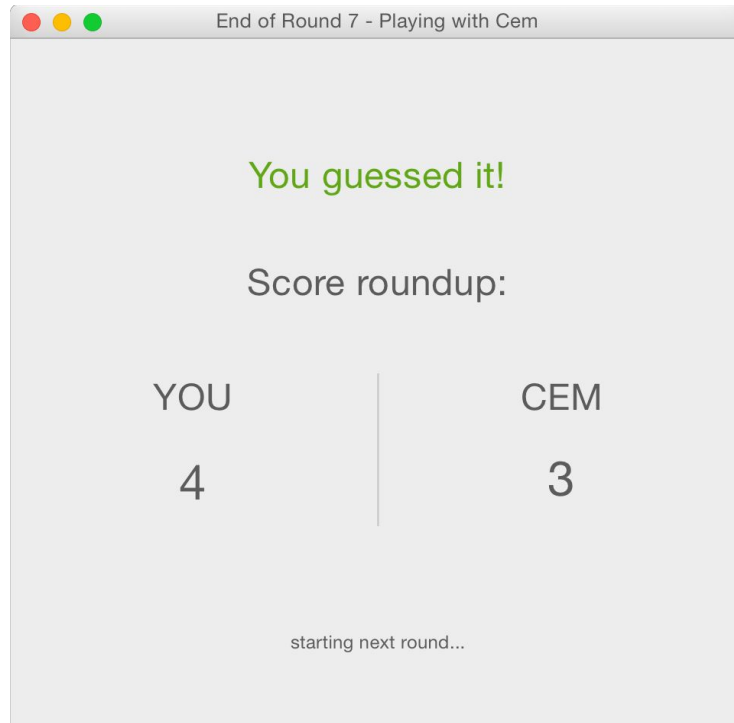


**Image 6: Drawing Screen**



**Image 7: Guessing Screen**





**Image 8: End Of Round**



**Image 9: Credits**

### 3. Analysis

Analysis part contains object model and dynamic models sections for the “Draw It!” application. More detailed explanation about “Draw It!” Game and diagrams are given in this section.

#### 3.1. Object Model

Object Model part includes Domain Lexicon and Class Diagram. Domain Lexicon consists of the “Draw It!” application keywords’ explanation and aims to clarify ambiguous words for the user.

##### 3.1.1. Domain Lexicon

**Active Player:** The player who draws in that particular round.

**Passive Player:** The player who guesses the word in that particular round.

**Drawing Movement/ Piece:** The mouse movement between mouse press and mouse release that Active Player made while drawing.

**Guess Box:** The box that Passive Player use while making his guess. It contains 10 random letters and n many empty letter boxes where n is the length of the chosen word.

**“?” Button:** This button is used by Passive Player when s/he thinks, s/he cannot find the word. When player clicks on that button s/he gave up from her turn.

**Close Button:** This button is used by both players when they want to quit from the game. When players press this button, they can see their total point in the game.

**Check Button:** This button is used by Active Player. S/he clicks on that when s/he thinks s/he finished the drawing.

**Trash Bin:** Passive user use this icon to clear the guessing area when he clicks on the wrong letter. Once clicked, all chosen letters are erased.

### 3.1.2. Class Diagram

In our class diagram, we have Player, Canvas, Piece, NetworkLayer, TurnTimer and differen control classes such as WordDrawingControl, WatchControl, GuessWordControl and GameController class.

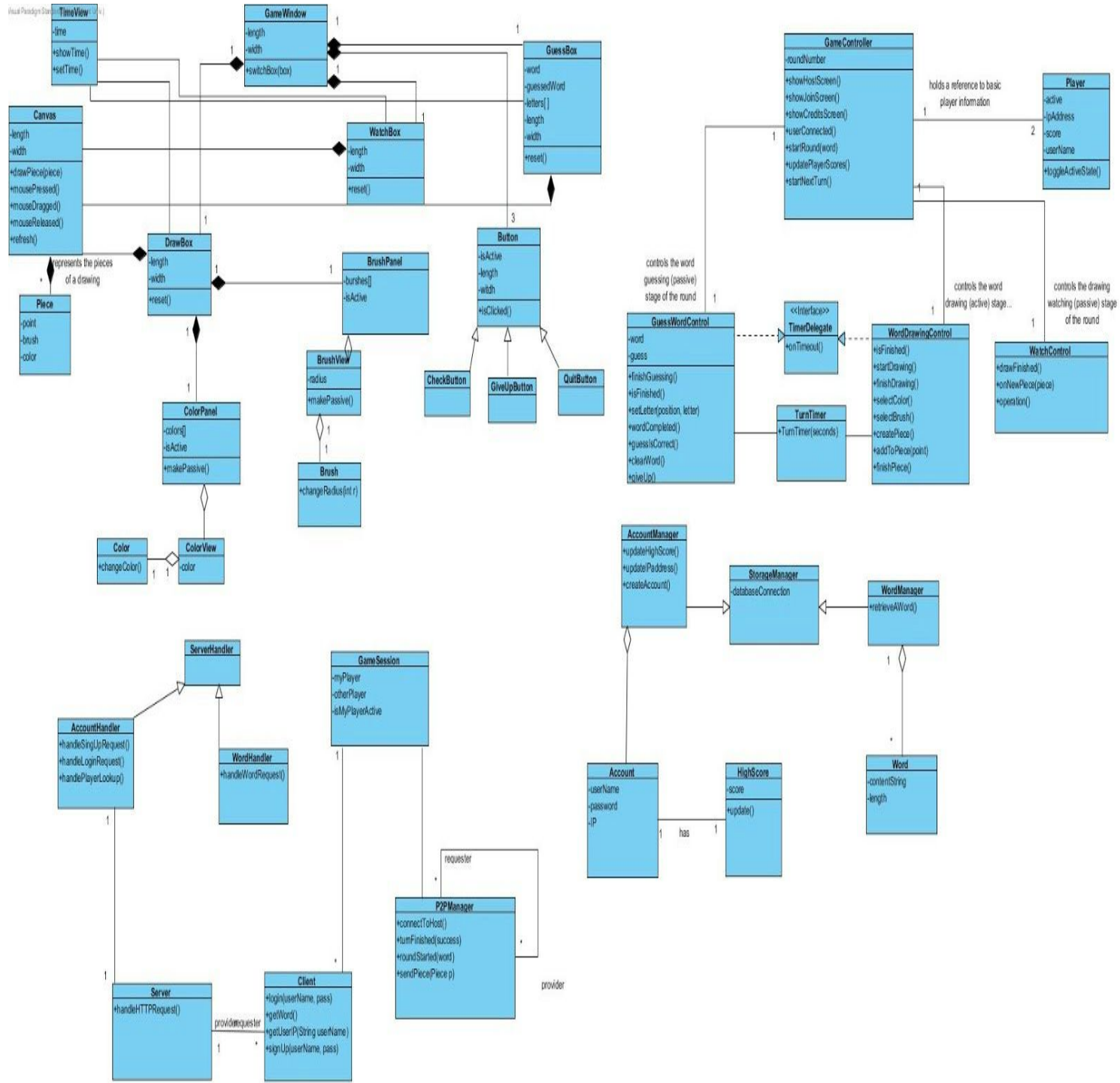
Game Controller class mainly responsible to showing screens and checking whether the users connected or not. It also responsible to starts the rounds and the turns, it updates the player scores as well. Player class holds a reference to basic player information such as IP address and score of the player. It has a variable called active as well, which is the information for whether the player is active or passive in that specific round. NetworkLayer class is the class which we make our connection and use for sending information. This class is for managing the network connection between the two game instances. ConnectToHost(), turnFinished(success), roundStarted(word) and sendPiece() method are parts of the NetworkLayer class. WatchControl controls the drawing watching stage of the round and it contains drawFinished(), onNewPiece(piece) and operation() methods.

TurnTimer class is the class which we keep count of the drawing and guessing process. This class checks whether the time is up. Time starts when the class's constructor is called. WordDrawingControl and GuessWordControl are using the TimerDelegate interface.

TimerDelegate interface has the onTimeout() method which notify that time is up. WordDrawingControl is the part here we control drawing. It has simple feature like selecting color and brush. Other than these, it has also isFinished(), startDrawing() and finishDrawing() method, as well as createPiece, addToPiece(point) and finishPiece() methods. On the other hand, GuessWordControl is where the part we control guessing process. This part has guess and word variables in order to check whether our guess and word matches. finishGuessing(), isFinished(), setLetter(position, letter), wordCompleted(), guessCorrect(), clearWord() and giveUp() methods.

Also, we have Canvas and Piece classes. Our Piece class is representing a piece of drawing which we call drawing movement as well throughout our report. Piece class has point, brush and color variables. Canvas is basically for providing a surface to draw something or to see the drawn figure. Canvas class has drawPiece(piece), mousePressed(), mouseDragged(), mouseReleased and refresh() functions.

Figure 2. Class Diagram

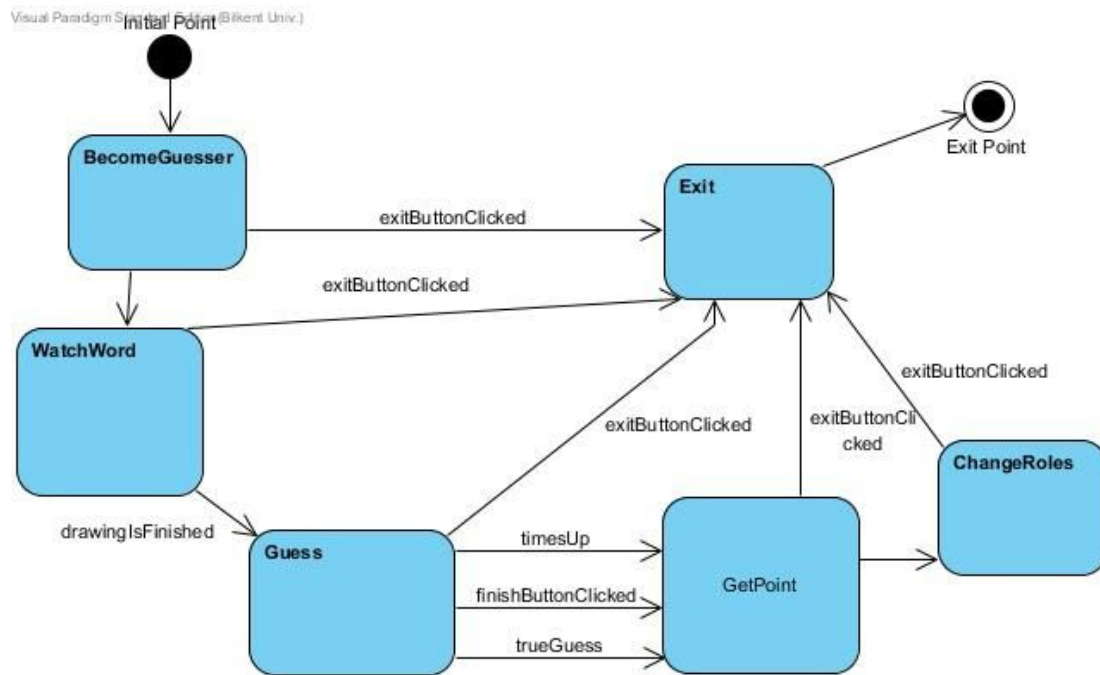


### **3.2. Dynamic Models**

Dynamic Models part includes the State Chart and Sequence Diagrams along with their explanation and scenarios with the aim of providing better understanding about “Draw It!” Game.

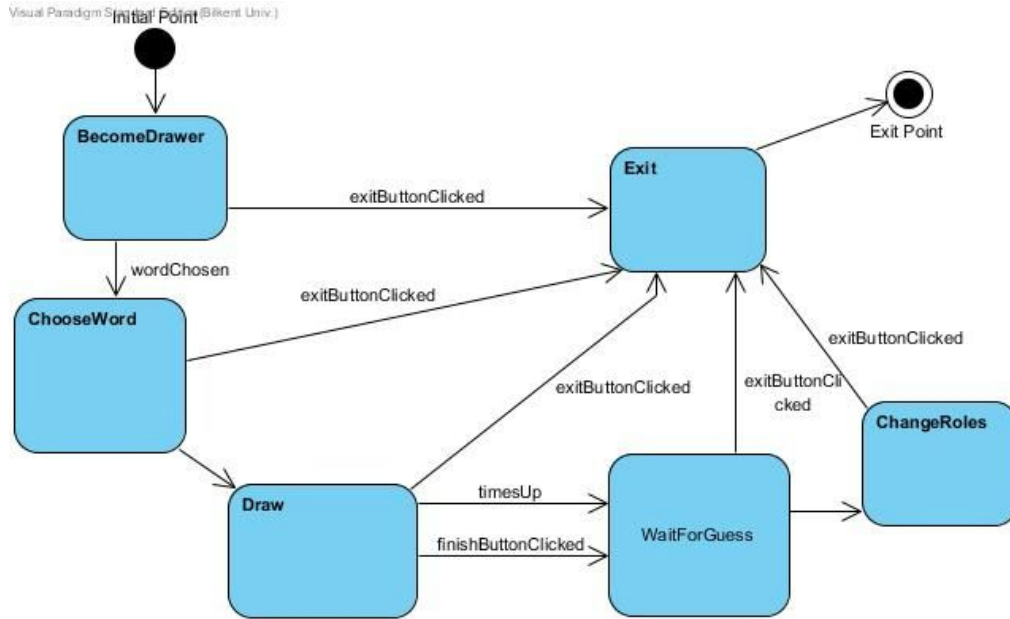
#### **3.2.1. State Chart**

We have 2 State Chart Diagrams. They are the states of Active Player’s and Passive Player’s separately. Overall, we introduce the behaviour of a player during the game process. There are two playing options which are hosting and joining. If the player chooses hosting, he needs to choose a word in the first step of the game. After the word choosing condition is finished, the next step is to draw the word. When he finishes the drawing process, he needs to wait for the end of guessing process of the other player. When the round is over, he becomes the player whose duty is to guess the word which is drawn by the other player. In short words, there is a role exchanging process. After this step, his role is turned into being a drawer. If at the beginning of the game, the player chooses the “Join” option, his movement starts from the BecomeGuesser state of the diagram and goes with the same cycle of hosting option and this cycle continues until one of the players wants to exit from the game. Players are able to exit from the game during the ChooseWord, Draw, WaitForGuess, BecomeGuesser, ViewDrawing and Guess states of the state chart steps. Also, following state chart diagrams are the simpler cases for active player and passive player states.



**Figure 3. State Chart Diagram 1: Passive Player State**

In this state chart diagram, we explain the behaviour of a passive player (the one who views and guesses the drawing). At the first step, he needs to view the drawing. When the drawing is done, in the guessing process, he needs to be successful to find the word or time needs to be up or the player needs to click on “?” option to go to the next step which is called as point check to see the current point situation. After points are shown, the lifetime of the passive player ends.



**Figure 4. State Chart Diagram 2: Active Player State**

In Active Player State chart diagram, we explain the behaviour of an active player (the one who is drawing). At first, the active player chooses a word to draw. After choosing a word he needs to draw it in 45 seconds or needs to click on the “Check” option to go the next step which is called as WaitForGuess. After the guess result comes, lifetime of the active player ends.



### 3.2.2. Sequence Diagram

This part contains scenarios and their sequential diagrams.

#### 3.2.2.1. Scenario 1: “menu” Scenario

**Scenario Name:** menu

**Participating actor instances:** john:Player, valerie:Player

**Flow of events:**

1. John and Valerie both click on the icon (exe) of the “Draw It!” application and they start the execution of the program.
2. John and Valerie both encounter with the main menu which has “Host”, “Join” and “Credits” options.
3. If they want to play the game, John clicks on the “Host” option and Valerie clicks on the “Join” option. This case could be in reverse roles.
4. If John or Valerie want to see the participants of the game project, only thing they need to do is clicking on “Credits” option.
5. If John or Valerie clicks on the “Host” option, s/he starts to wait for the other player to play the game and during this waiting process, he sees his device’s IP address. This IP address is for the other player because the one who wants to join

the game needs to use it. When the other player joins the game, John finishes his work with this step and goes to the part that he chooses the word.

6. If Valerie or John clicks on the “Join” button, Valerie needs to fill the IP address space by using the IP address that is shown in the host player’s screen. After filling this space, the connection is provided and the game begins.

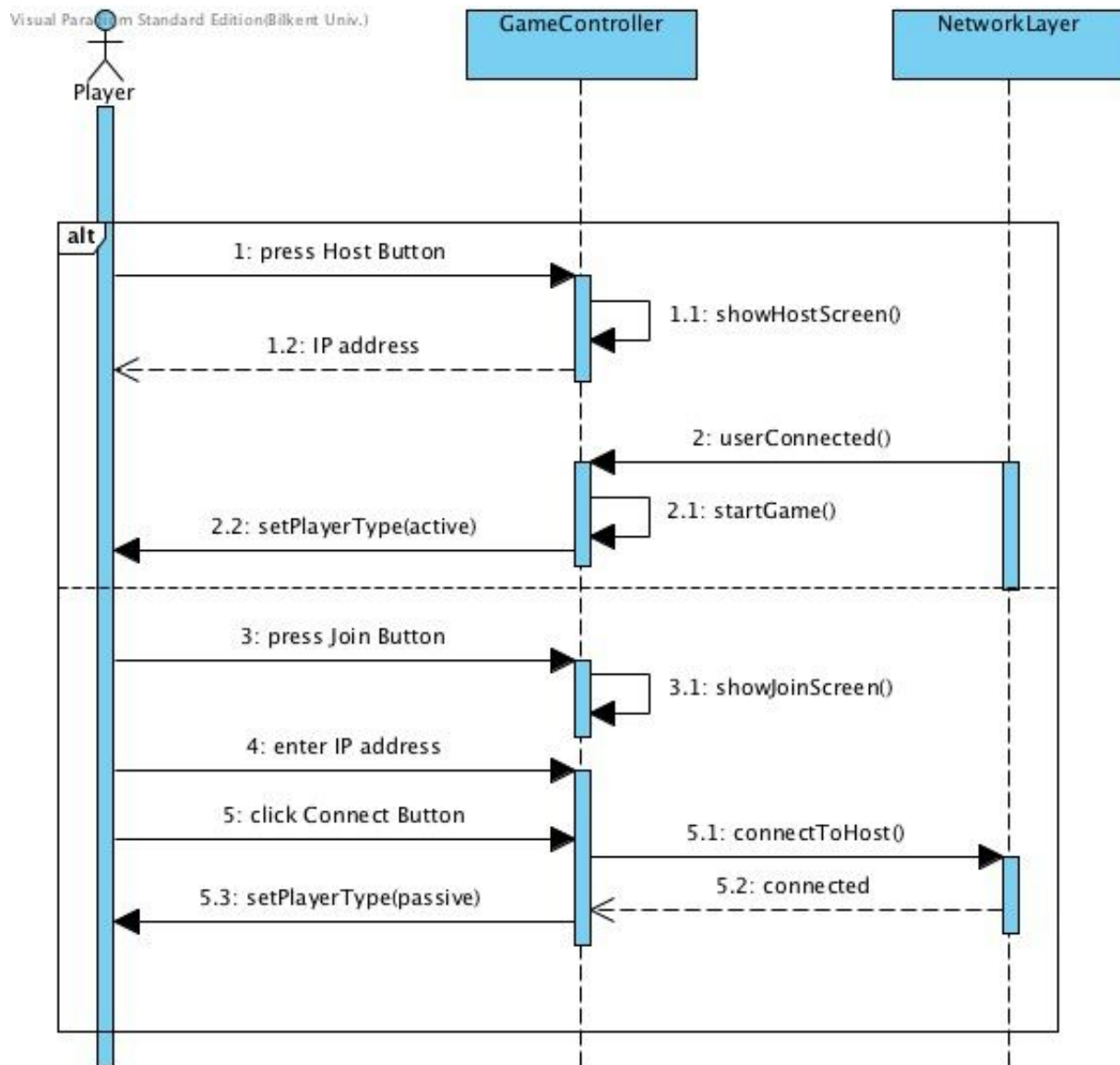


Figure 5. Sequence Diagram 1: “menu” Diagram

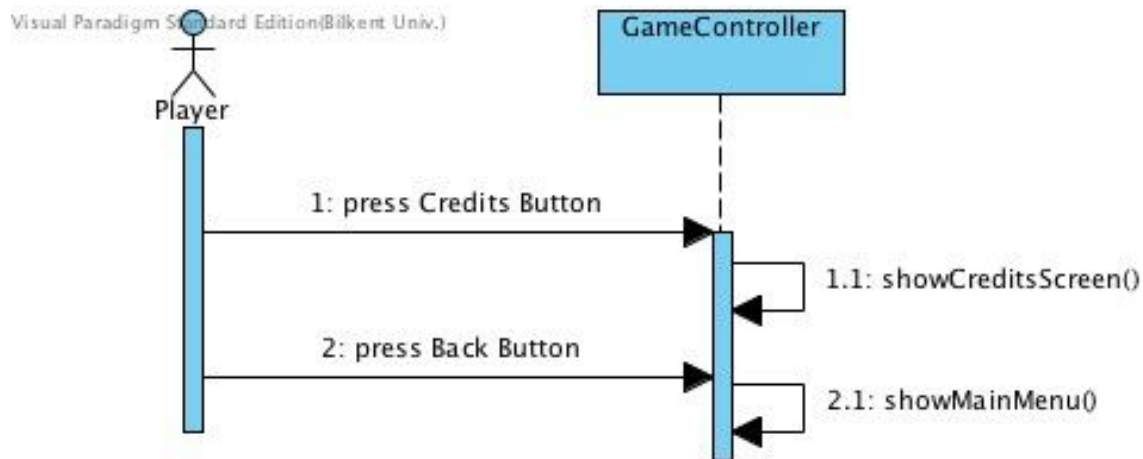
### 3.2.2.2. Scenario 2: “showCredits” Scenario

*Scenario name:* showCredits

*Participating actor instances:* john:Player

*Flow of events:*

1. One of the players clicks on the Credits button on the main menu.
2. The network sends new canvas which shows the names of contributors and contact information of this game to the player.
3. After choosing the “Credits” option, John finishes his work with the main menu.



*Figure 6. Sequence Diagram 2: “showCredits” Diagram*

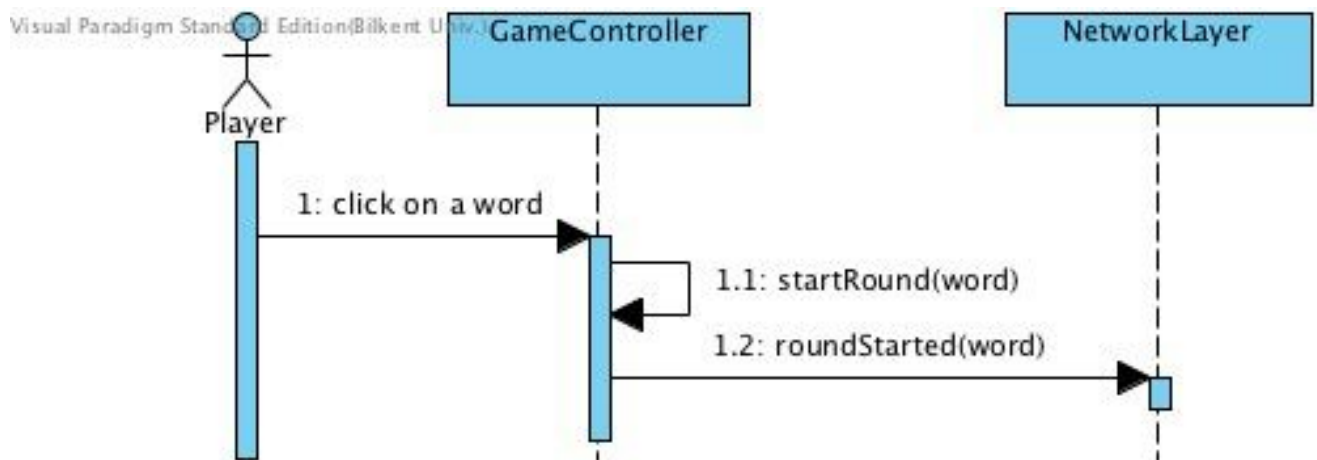
### 3.2.2.3. Scenario 3: “wordChoosing” Scenario

**Scenario Name:** wordChoosing

**Participating actor instances:** john:Player

**Flow of events:**

1. John encounters with three different words which are listed randomly.
2. John chooses one of them which he can draw.
3. After choosing one of them, John finishes his work with the word choosing step.



**Figure 7. Sequence Diagram 3: “wordChoosing” Diagram**

#### 3.2.2.4. Scenario 4: “wordDrawing” Scenario

*Scenario Name:*      wordDrawing

*Participating actor instances:*      john:Player , network:NetworkLayer

*Flow of Events:*

1. John may select a color from color panel.
2. He presses the mouse when mouse cursor points a specific point (point where he wants to start drawing) in the canvas.
3. He draws the figure partially or completely.
4. He releases his mouse.
5. System send the point coordinations of his drawing to the network.
6. If he didn't finish drawing, he repeats the first 5 steps.
7. If required time (45 seconds) is up or John clicks the finish (check) button, drawing word process ends.
8. The active player's (John's) part of the round finishes.

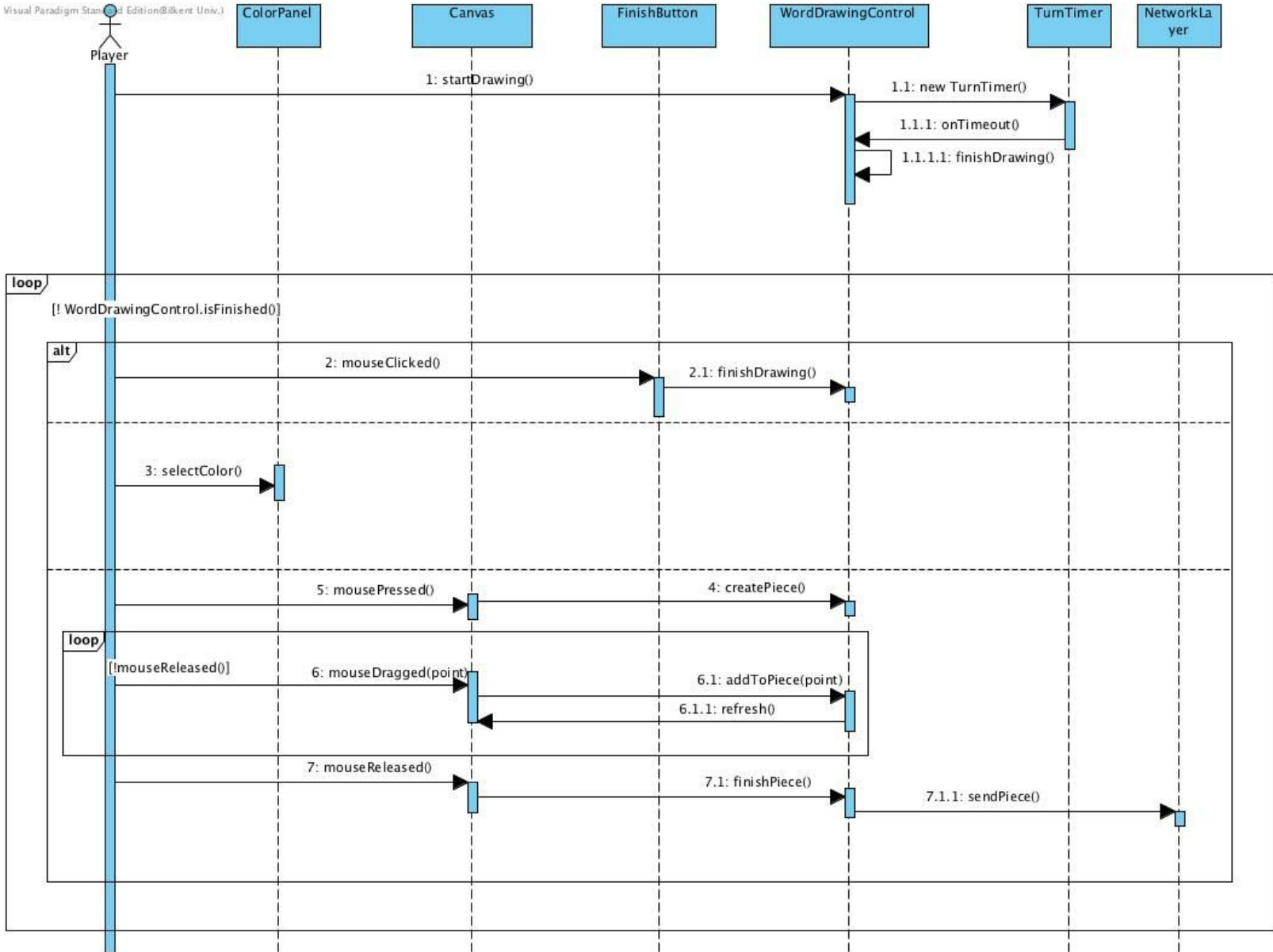


Figure 8. Sequence Diagram 4: "worldDrawing" Diagram

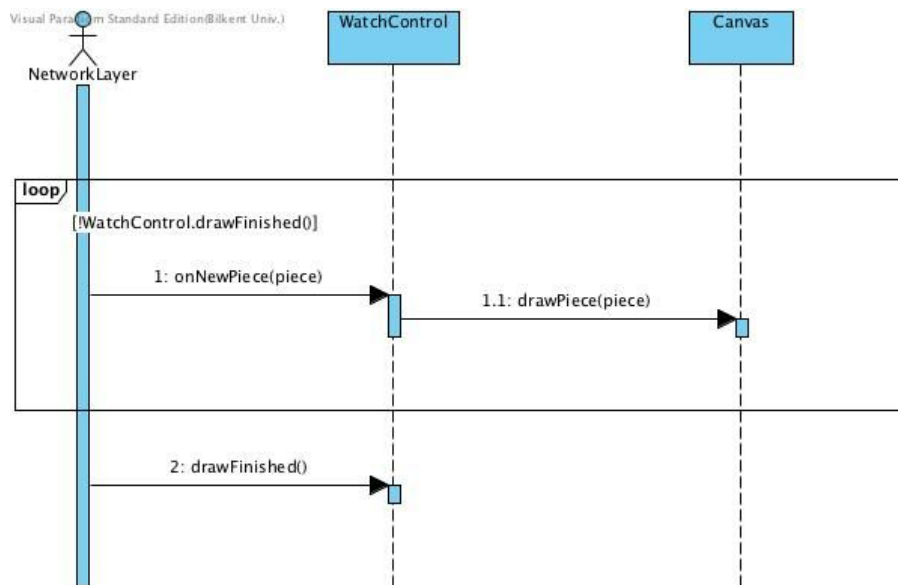
### 3.2.2.5. Scenario 5: “watchTheWord” Scenario

**Scenario Name:** watchTheWord

**Participating actor instances:** network:NetworkLayer

**Flow of Events:**

1. The coordinates of the drawn points come from the network to the passive player.
2. The points are drawn to the canvas as soon as they come according to their x,y coordinates and their colour.
3. If the required time, 45 seconds, didn't finish or if the active player (John, who draws) didn't click on the finish (check) button the first two step repeats.
4. If the required time, 45 seconds, finishes or if the active player (John, who draws) clicks on the finish (check) button, watchTheWord process ends and no data of points comes. Canvas stays in its last condition with the drawings.



**Figure 9. Sequence Diagram 5: “watchTheWord” Diagram**

### 3.2.2.6. Scenario 6: “guessingWord” Scenario

**Scenario Name:** guessingWord

**Participating actor**

**instances:** valerie:Player

#### **Flow of Events:**

1. System adds a box under Valerie’s (the passive player’s) canvas. Box consists of “\_empty letter boxes for each letter that word has.  
  
Ex: Word: Flower -> \_ \_ \_ \_ \_  
  
10 letters also given under the dashed lines. The letters have some extra letters as well as the word’s letters.
2. Valerie starts to guess the word by clicking the letters with the same order as the word she guess has.
3. If she clicks on the wrong word she press Trash Bin button and clear button clear all the letters and Valerie starts the guess the word all over again!
4. If she can’t find the word, Valerie clicks on the “?” button and that means she gave up from her turn. Her part ends.
5. When time is up to 45 seconds, the guessing time ends, therefore her part ends.
6. When her part ends, if the entered word correct guessing box colored to green. System gives 10 points to Valerie.
7. If it is false, first it is colored to red and 5 seconds later box colored into its original color and shows the correct word.
8. The round finishes.



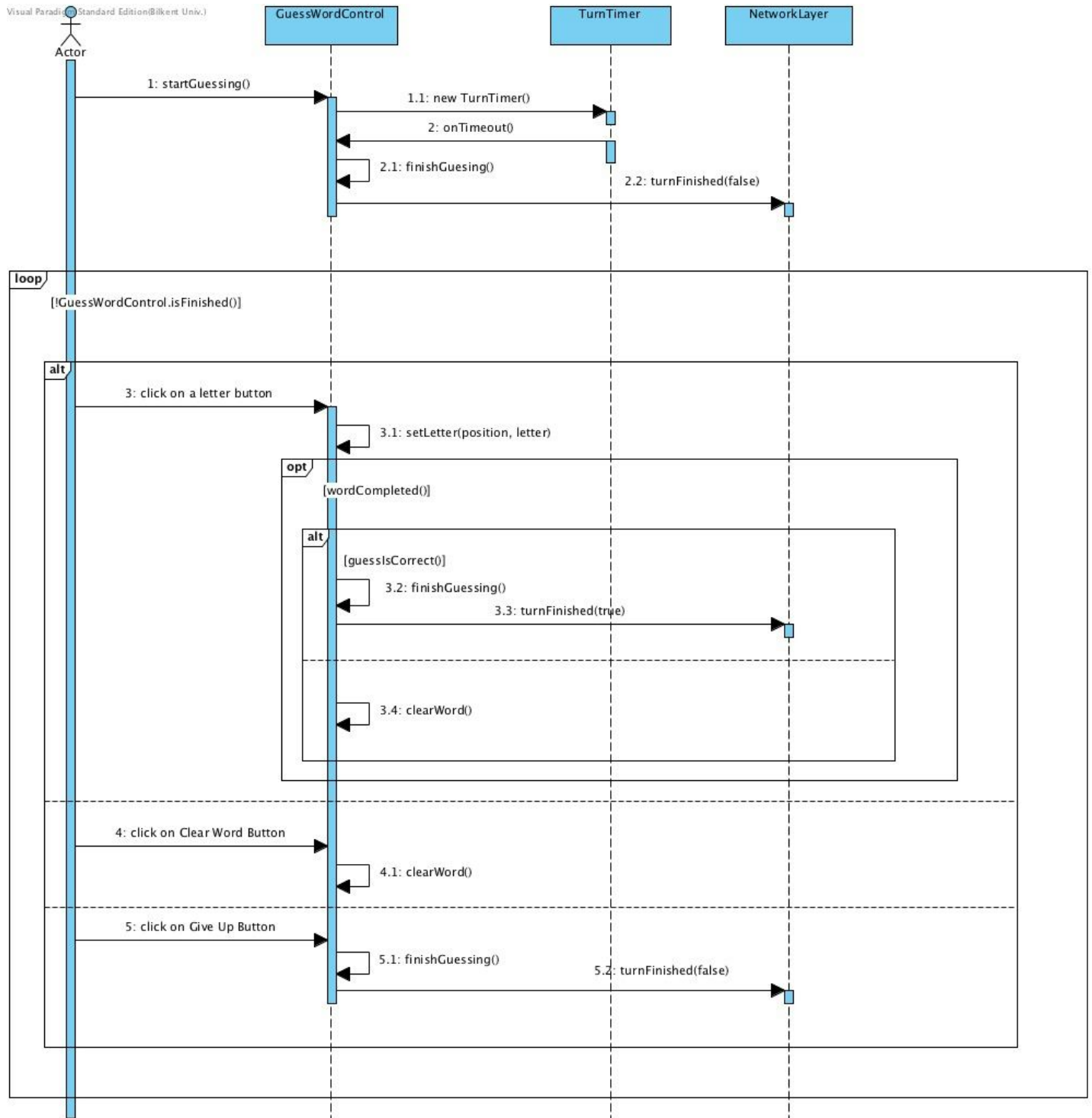


Figure 10. Sequence Diagram 6: “guessingWord” Diagram

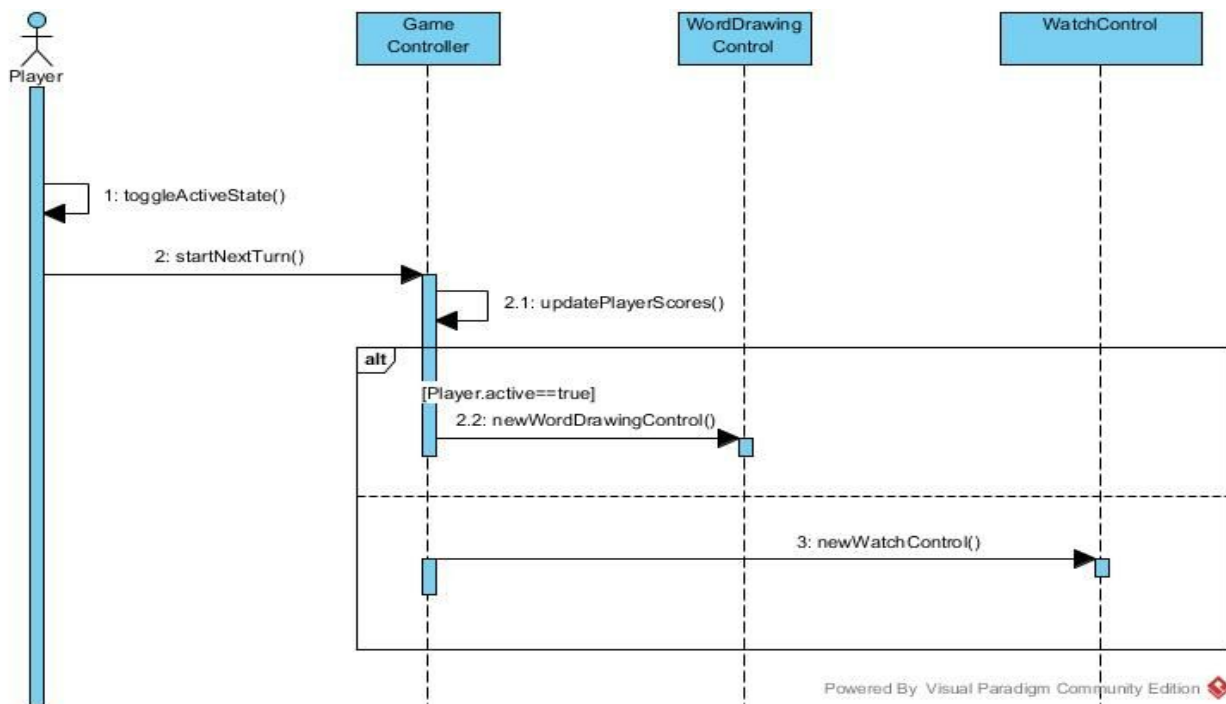
### 3.2.2.7. Scenario 7: “roleExchanging” Scenario

**Scenario Name:** roleExchanging

**Participating actor instances:** valerie:Player john:Player network:Network

**Flow of events:**

1. At the end of each round, role of the active and passive player is changed and if passive player guesses the drawn word correctly, his/her point is updated.
2. The new canvas is sent to Valerie over the network. If she is active player, the next round she will be passive player.
3. The new canvas is sent to John over network. If he is passive player, the next round he will be the active player.



**Figure 11. Sequence Diagram 7: “roleExchanging” Diagram**

## 4. Design

### 4.1. Design Goals

- **Extendibility:** Project should be extendible. In the future, we can add some new features or modes. Our game should be improvable for new functions with little modifications.
- **User Friendliness:** Menus and the logic of the game should be easy to understand. Users should not use too much time with adapting the game environment.
- **Security & Safety:** Project should be safe for users and their data. To make the project safe, all users need to fill the spaces which are ID and Password. By doing this, their data are going to be saved.
- **Development cost:** Project development should not be expensive for developers. The only thing that costs some money is to rent a server by paying approximately \$10 per month and it can be considered as cheap.
- **Readability:** Since the project is based on GUI methods, it is quite readable for programmers who have some GUI background.
- **Availability:** The applications should be available for everybody whose devices are adaptable for Java.
- **Ease of use:** The usability level of the application should be high and this high level is provided by user friendly structure of the design. In addition to this, rules of the game is not hard to understand. The only thing that users need to do is just having fun.
- **Fault Tolerance:** Project should tolerate the system faults such as connection down. When the connection is over during the game, the game stops the rounds and then both of the users see their scores without closing the application.

- **Response time:** The quality of response time is essential for this project since the drawing process can be observed in real time by the guesser player. To improve the response time level, the projects requires a socket connection.
- **Minimum number of errors:** Project should include minimum number of errors. Since the structure of the projects is based on GUI implementation, the possibility of encountering with programming errors is quite low.

## **Trade-Offs**

### **Functionality vs. Usability:**

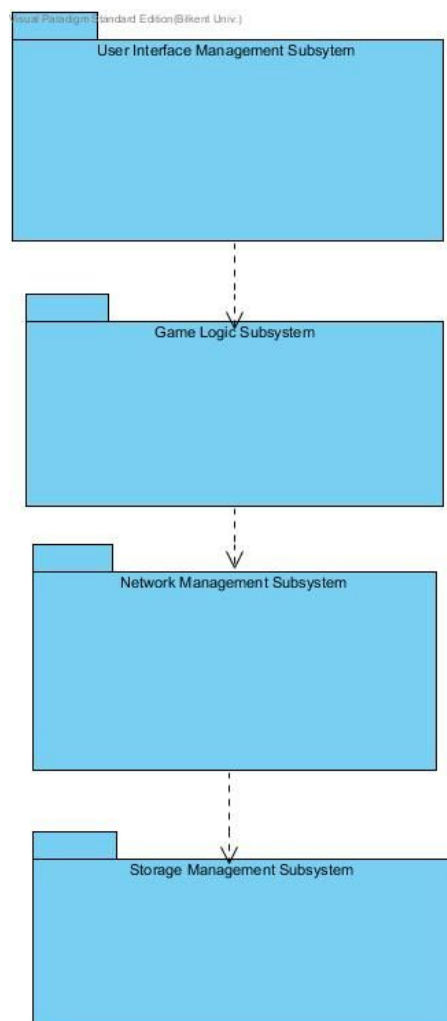
Since user-friendly design is essential to increase the application's usability, we need to focus on the ways how to make the usability level of application as high as possible. In that sense our game's functionality is not too complex to prevent its usability. For example, one can play the game with a help of mouse and a virtual keyboard to guess the word. No complicated special keys are needed to learn. No complex functions for users to get confused. Hence, decrease the usability of the game.

### **Performance vs. Memory:**

Sometimes performance of a game and memory storage can be at odds. In order to have high performance for the game, its memory allocation should be low. Having 40-50 MB of space requirement would be sufficient for a high performance.

## 4.2. Sub-system Decomposition

Our system is divided into 4 subsystems, in 4 layers. The first three layers are contained within Draw It Java game application run on the players' computers whereas the last one is contained within the Java server application run on a web server machine. When we were dealing with this process, we tried to implement the requirements of cohesion and coupling notations. Cohesion measures the dependence among classes and coupling measures the dependencies between subsystems. To make our project more efficient, we have to reach the situation which has maximum cohesion and minimum coupling as much as possible. When we were working on this, we focused on good demonstrations of classes with their proper relationships between each other and we tried to save every subsystems' features without dealing with problems which are related to the other subsystem.



*Figure 12. Subsystem Decomposition*

#### **4.2.1. User Interface Management Subsystem**

This subsystem consists of classes to draw the dynamic User Interface and take input from the player. Since the project requires high qualified user-friendly design for users, User Interface Management Subsystem has a very important role in our design process. By focusing on the main game window screen, we describe our User Interface Management Subsystem with additional design classes. In order to increase the functionality level of the application, we use several important features such as colour panel, finish buttons, brush radius changing etc. To explain this subsystem, we can simply divide it into two main parts as drawing process and guessing process.

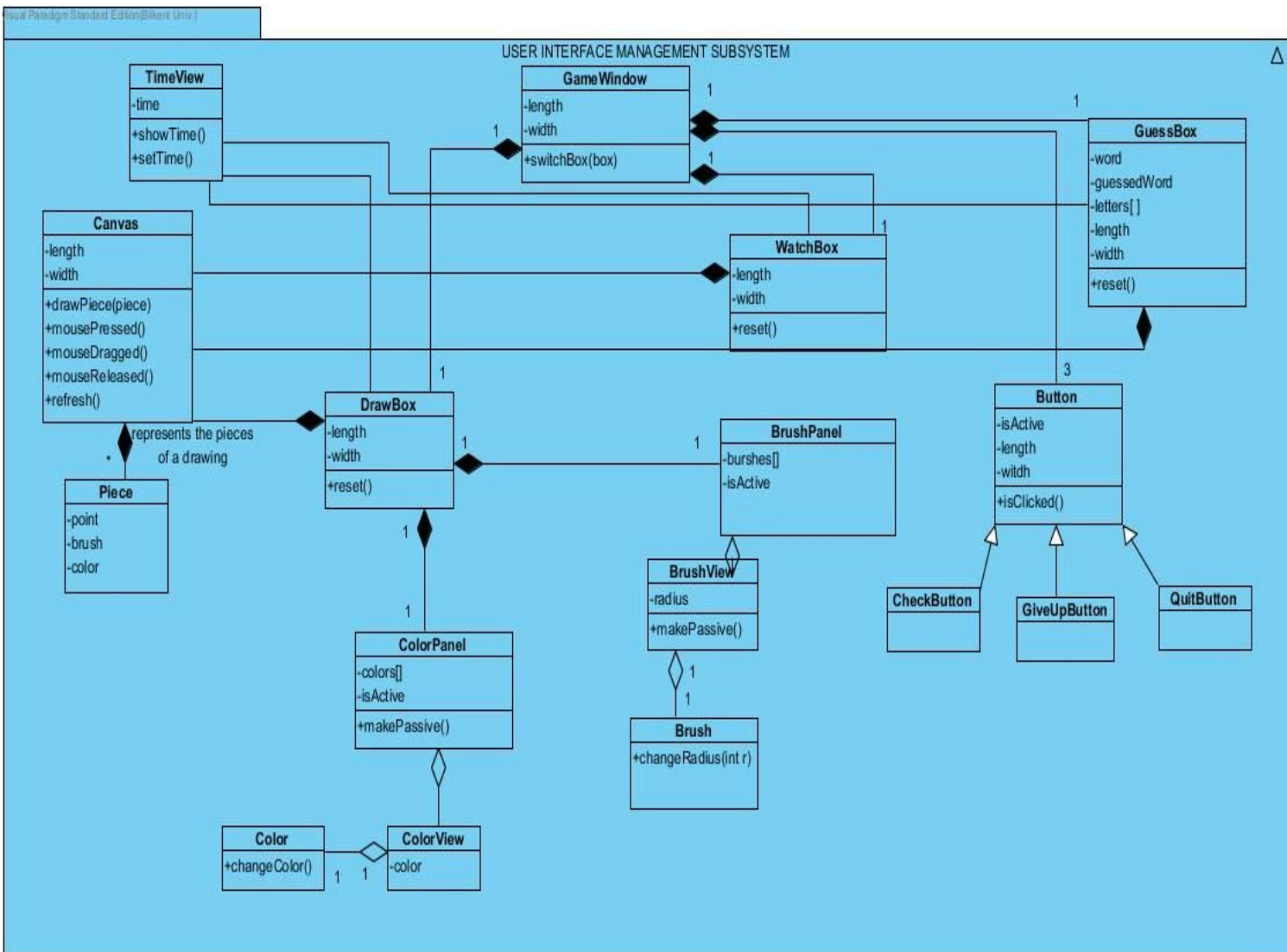
In drawing process, user is playing the game by just drawing the word and because of that his canvas object is activated by the application due to DrawBox. Every mouse movement on the canvas with left click is named as piece and these pieces are transferred to the guesser player. In addition to this, there is a colour panel which provides user to choose colours to make his drawings more understandable. Also there is a brush panel which helps user to change the radius to create more successful drawings. Both colour panel and brush panel are located on the 'GameWindow'.

In watching and guessing process, guesser player watches the drawing in real time with his own canvas which is provided by WatchBox. After watching the drawing, the guessing process is started and the guesser player tries to guess the word which is located on his own canvas by using the guess box. In this case the canvas and the guess box are both provided by GuessBox. There are some additional features that are used for both two parts of the subsystem such as buttons and timer. Timer is used to limit players for their drawing and guessing process

and it is provided by GameWindow. Buttons' duties can be differentiated according to players' roles. There are three main buttons which are CheckButton, GiveUpButton and QuitButton. QuitButton is used for both drawing and guessing process to quit from game and to see the scoreboard. CheckButton is used in the drawing process to say that "I am done." and it finishes the countdown directly. GiveUpButton means that the guesser player cannot find the word and he does not want to wait until the end of countdown. If we look at the big picture, there are three main parts which are GuessBox, DrawBox and WatchBox. All of these three parts are connected to the GameWindow. This is the demonstration of User Interface Management Subsystem of the "Draw It!".

These classes consist of the View part of Model-View-Controller architecture, which is explained in detail in the next section.

Figure 13. User Interface Management Subsystem





#### 4.2.2. Game Logic Subsystem

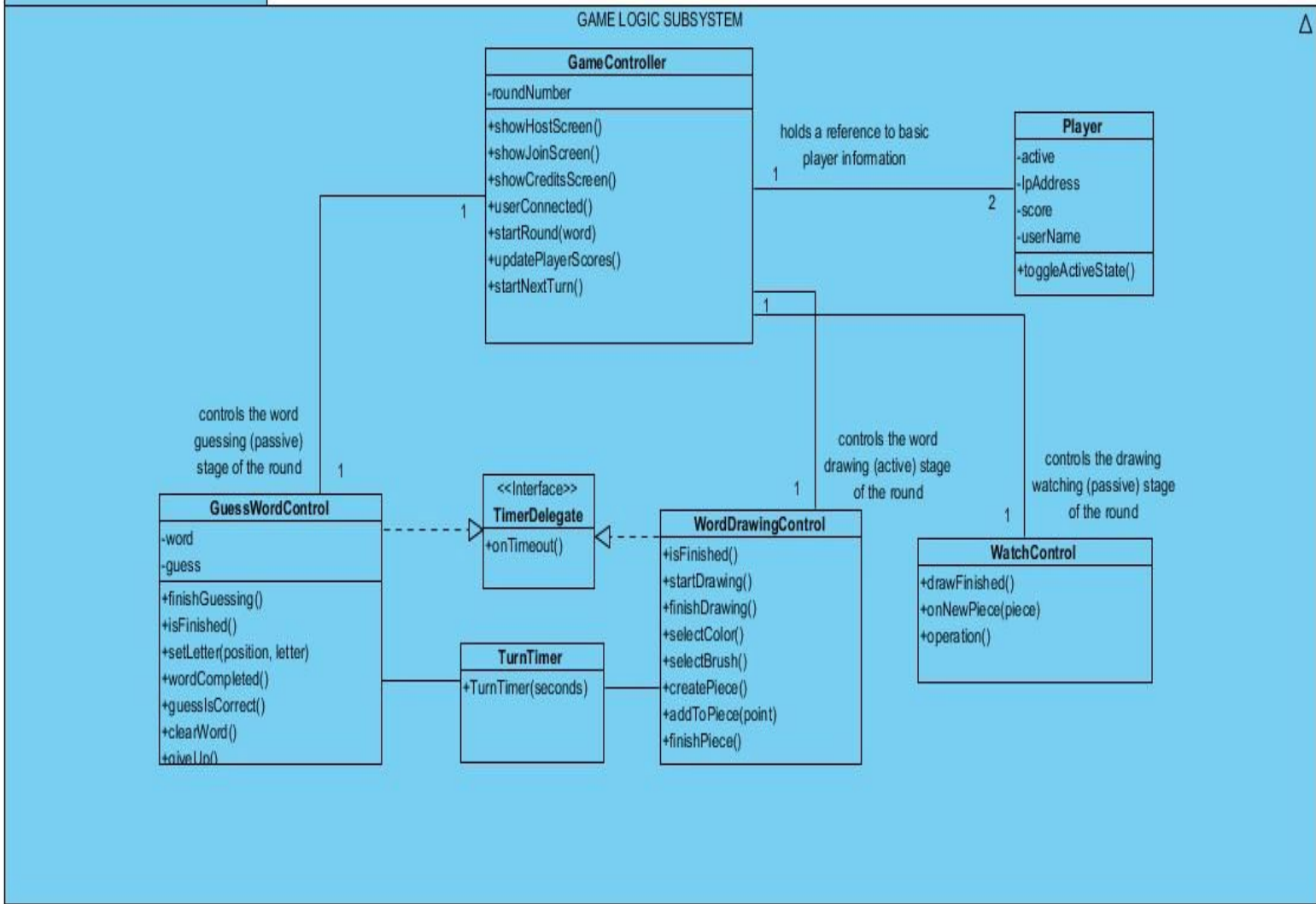
This subsystem consist of controller classes that govern the gameplay. All the game logic, is contained within this subsystem. It is designed in such a way that for a different platform, User Interface Subsystem can be replaced with another implementation and the Game Logic Subsystem would still be able to provide the gameplay through its interface.

This subsystem contains the main class responsible from the program, 'GameControl'. It also contains controllers with more particular responsibilities, such as 'WatchController', 'GuessWordController' and 'WordDrawingController'. These classes control certain stages of the gameplay.

In WordDrawingController, the drawing process is managed properly. In WatchController, the watching process of the guesser player is managed according to rules of the game. In the GuessWordController, the guessing process is managed with rules as well. The common point of these three classes is using time related classes which are TimeDelegate (interface) and TurnTimer. In addition to these, there is a Player class which is connected to the GameController class with players' Ip addresses, scores and user names.

In order to communicate with the other player and keep the game going, this subsystem is coupled with 'Network Management Subsystem'. As described in the several sequence diagrams in the previous sections, in each gameplay scenario the player takes an action to advance in the gameplay, so 'Network Management Subsystem' is notified by their actions.

Figure 14. Game Logic Subsystem



#### **4.2.3. Network Management Subsystem**

Network Management Subsystem is responsible for the all networking tasks in the game. These tasks include managing communication between computers of players, retrieving and sending latest user information, high scores and retrieving a word.

Each game of Draw It is played by two players using different computers. Since players are supposed to play this game on their individual computers, there are two game instances needed and these instances need to be connected. This subsystem provides this feature in its P2PManager class, which handles a Peer-to-Peer connection between identical game instances. This is explained further in the next section.

This subsystem also manages the relationship between the game instances and the server, whose main purpose is storing and retrieving player account information and a word list. Game client can send a few different requests to the server, such as login, sign up, lookup a player, get a new word. Server handles these individual requests using different classes such as `AccountHandler` and `WordHandler`, according to the kind of the request.

When the IP address of the other player is known, `otherPlayer` in `GameSession` is updated. As a result, the game is ready to begin, P2PManager can use this information.

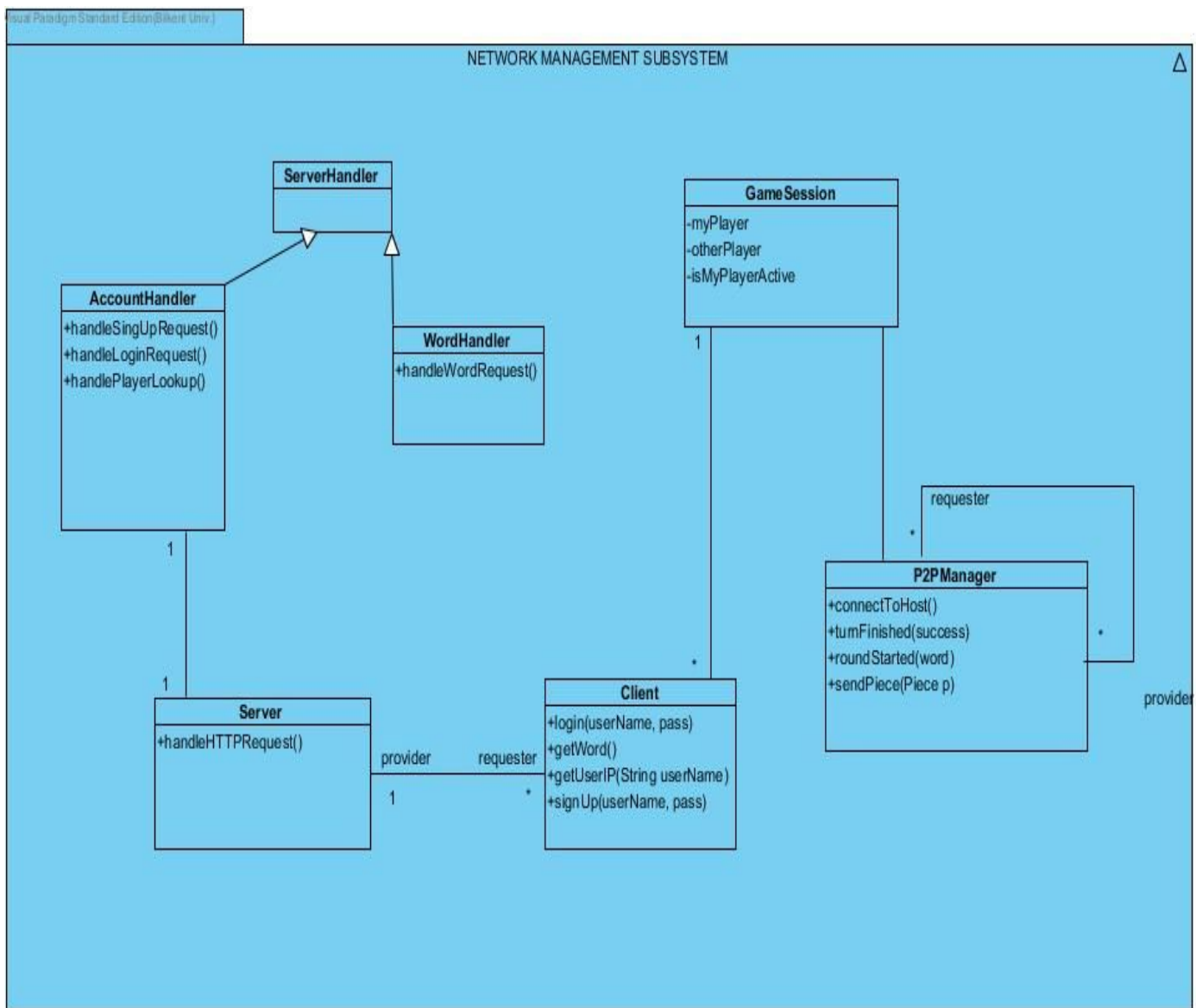


Figure 15. Network Management Subsystem

#### 4.2.4. Storage Management Subsystem

This is a fairly simple subsystem that lies on the server. Its main duty is to provide persistent data storage for the Network Management Subsystem. As Network Management Subsystem receives a request on the server side, it needs to retrieve the relevant data and send it as the response. This subsystem is the middleware between the server part of Network Management Subsystem the database server software. It simply retrieves data from the database using JDBC and passes it to its caller, such as the Handlers in Network Management Subsystem.

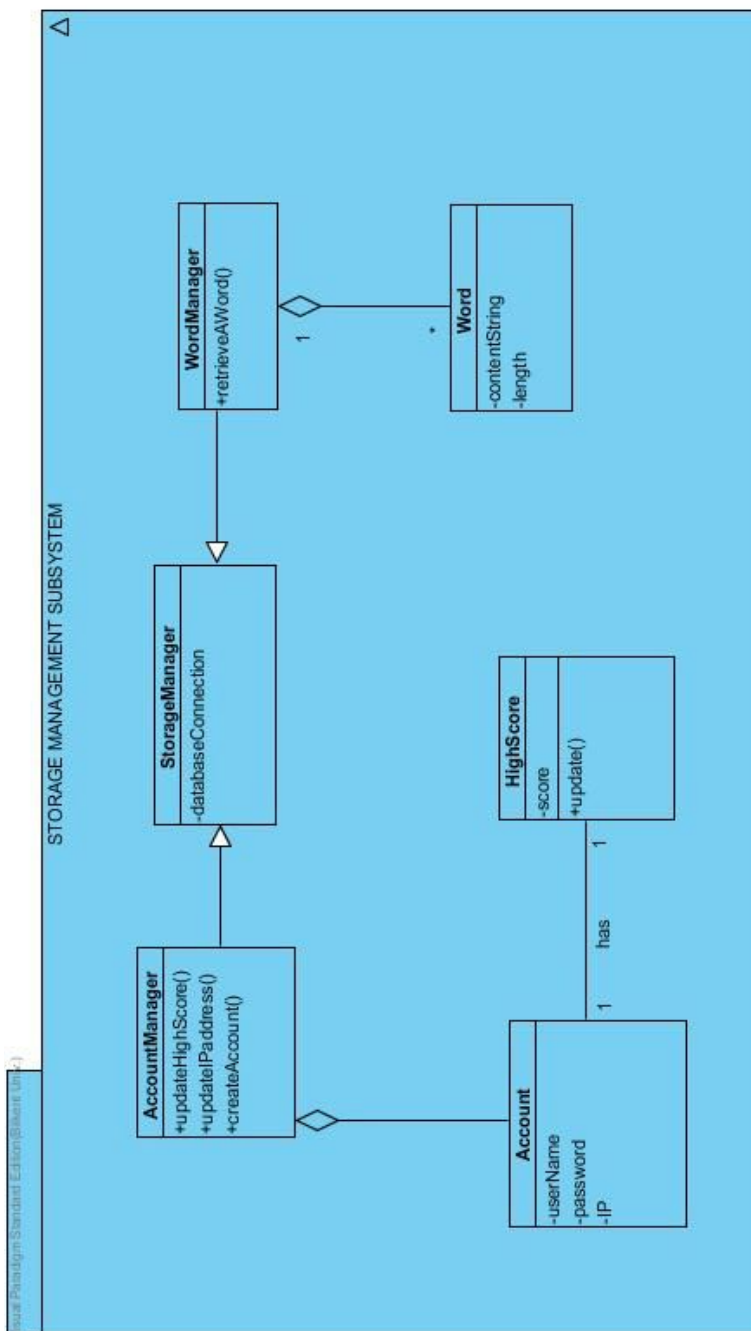


Figure 16. Storage Management Subsystem

### 4.3. Architectural Patterns

Our system is essentially a two-player graphical game on a network with a complementary server for account management. Hence our system uses a variety of architectural patterns, each of them to accomplish a different goal. Namely, the system uses Model-View-Controller architecture for managing the user interface in each of the game instances. The system also uses Peer-to-Peer architecture to set up and sustain the communication between two game instances. Finally the system uses a simple client-server architecture to simplify the game setup between two computers on the internet, keep high scores and retrieve a word from the most recent list of words.

#### 4.3.1. Model-View-Controller

We use the Model-View-Controller pattern to organize the code in the game application. However, we use a different and simpler definition of Model-View-Controller pattern: “Ideally, a model object has no explicit connection to the user interface. Controller objects tie the Model to the View”.

(<https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html>)

Our models are simpler classes which just represent some data, mainly because our models are either quite static (e.g. Player) or too dynamic ('Piece's getting created all the time, as one of the players keeps drawing) and the data for creating new models often come from the network or the canvas, which naturally go through a Controller first. In this particular case, applying the Observer pattern would make the code more complicated and error-prone than it needs to be.

In our system, Game Logic Subsystem handles the Controller part of the MVC pattern. User Interface Subsystem handles the View part of the MVC pattern. Classes that represent data, such as Player or Piece handle the Model part of the MVC pattern.

#### **4.3.2. Peer-to-Peer**

In order to achieve our response time goal, the connection between the two game instances, which is briefly explained in Network Management Subsystem section, will be using Peer-to-Peer architecture. This way, the two game instances will talk directly to each other during the game. The messages to be sent through the connection will include notifying the other side about what the player has just drawn and what word a player guess for that drawing. This communication will be done over a TCP socket connection and data to be sent over will be Pieces and Words.

#### **4.3.3. Client-Server**

This system maintains the communication between the game application and the account management server by sending HTTP(S) requests. Every time a player launches the game, their current IP address detected from the network is sent to the server by sending a HTTP(S) request. Hence, every player is mapped with their most recent IP address. This way, when a player wants to join a game hosted by another player over the internet, all they need to do is type in the player's name (username). Then, by sending a simple HTTP(S) request, the game application can match the username with their most recent IP address. Having this server makes it easier for

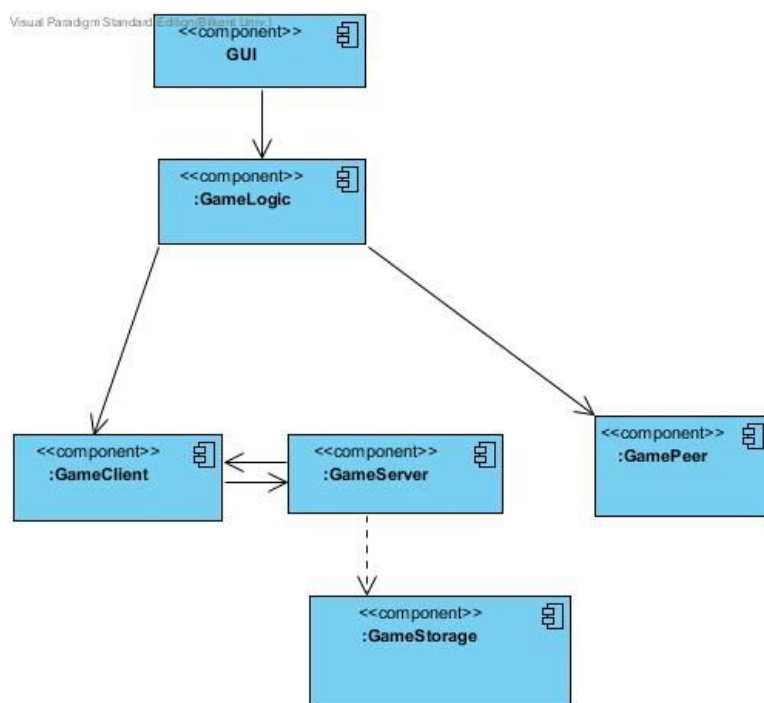
players to set up a new game. The players can simply type in a player name to host or join a game, instead of entering long IPv4 or even IPv6 addresses.

Another facility provided by the client-server architecture is retrieving a word from the same server. This is a fairly simple task, in which the client sends another kind of HTTP(S) request to our server, and the server retrieves a word from its word database and sends it back as the response of the request.

#### 4.4. Hardware-Software Mapping

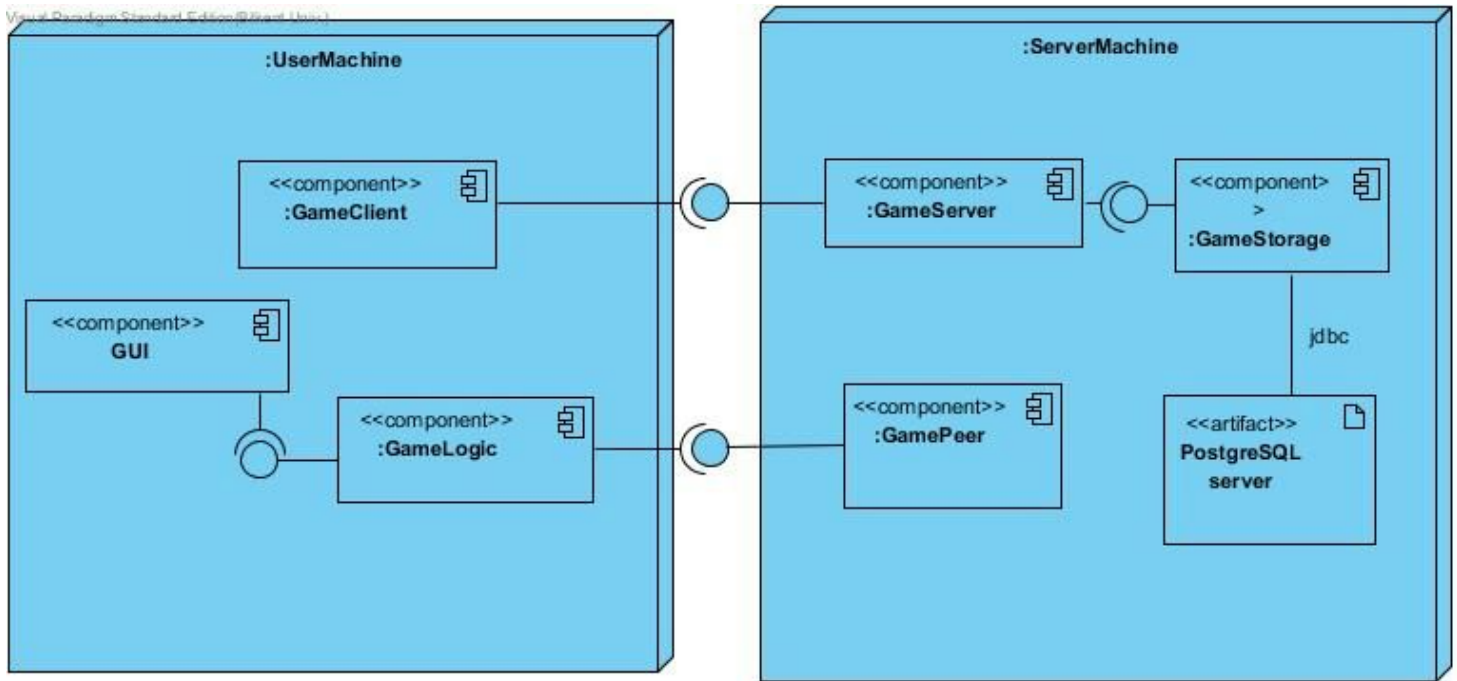
Our server machine is going to run a GNU/Linux operating system such as Ubuntu, and it will be running our server application listening on port 80/443. PostgreSQL Database Server will be running on the same machine to help the application server

Our game application will be Java program that runs on any major desktop operating system. The application, which is the client in this case, will communicate with the server over the HTTP(S) protocol. To play the game with another player, the game application will talk to another identical game application over the network.



*Figure 17. Component Diagram*





*Figure 18. Deployment Diagram*

## 4.5. Addressing Key Concerns

In this part, we describe the key points which are needed to ensure that subsystem decomposition can account for any constraints and addresses all the nonfunctional requirements during the implementation period. There are five main key concerns which are persistent data management, access control and security, global software control, boundary conditions and minimal server design.

### 4.5.1 Persistent Data Management

There are three persistent data objects in this system which are account (player), high score and words. Although our persistent data is not so large and complicated, we are going to use a relational database management system, such as PostgreSQL. In the year of 2015, using a powerful RDBMS along with JDBC offers a better overall solution, even on tiny datasets, than

writing to and reading from a plain text file. A RDBMS ensures data integrity, and writing code that uses JDBC to talk to the database system is actually easier than writing code for an error-prone plain text file reader/writer. Each player decide their own usernames when they start to play the first game. They use their passwords to log onto their accounts. The second data object is high score. When users play the game, the scores of each player is stored and the game displays the high score to the players in descending order. Words are also stored in the database and retrieved by the game clients. This way, we can update the word list in the database and clients can use the newly-added words.

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

Since each player can login the game only with their username and password, their passwords should be hashed, salted, stored and protected by the server. Each player has their own username and password. In addition to this, it is possible to play the game with your friend's username and password. Only player account and high score are saved within the game. So the game does not save drawings or something else. For every turn, system clears the drawing canvas and the answers for the drawings.

Game playing is executed with player over an individual socket communication channel and this makes it hard to abuse the system, as long as Man-in-the-middle attacks are prevented by using secure communication channels. The game is available for everyone who downloads the game to the computer. Except from the game package, nothing else is protected on the computer.

### **4.5.3 Global Software Control**

In this system, event driven control flow is decided to use because this system is promoted by object oriented software. And the flow of the system is under control of the events. With the Model-View-Control architecture of the system, the sequence of the program depends on events. These events are done by mouse clicks in our system. Each mouse click updates the views.

Also user interface subsystem of this game takes inputs from the players and transfers them to the controller. Controller determines what kind of event it is and according to this information, it makes the necessary changes in the views. Since there are different objects that make decisions on the events by evaluating the mouse clicks, it is possible to state that the control is a part of design that is distribution of dynamic behaviors.

### **4.5.4 Boundary Conditions**

The boundary conditions is related with the starting and ending conditions of the system.

- **Initialization**

Initialization of the system is beginning with the opening of the jar file. There is no need to set up anything. Additionally, this game can be run on the environment which java runtime is installed on the computer. The user should click double to the jar file of the game. This events starts the game and thus this enables players to join the game.

- **Termination**

In order to leave the game, the player should click on the 'X' button on the game screen. This button ends the termination of the game. Also, the highscore section in the database is updated before the game termination.

- **Failure**

When the connection between the players is collapsed, system tries to reconnect until it connects successfully and recovers the game. However, they will continue to play with a fresh round. When the connection between client and server fails, system also tries to reconnect. Another exception throws is that when a user enters wrong username and password. Game does not allow user to play the game.

#### **4.5.5. Minimal Server Design and Code Reusability**

Given the limited amount of time to implement the project, we are trying our best to minimize the time we will spend on developing the complementary server application. Hence, we designed our the server application to be as small as possible. We are going to share the model classes such as Account and Word between the game client application and server application. We are going to use a modern web framework for Java which allows us to get rid of unnecessary code and focus on our core tasks, which are basically retrieving from and saving to the database, after an authentication phase. Such a web framework is Play! Framework. For more information please visit <https://playframework.com/>.

## 5. Conclusion

In the Analysis Report, we made the first step towards the Object-Oriented Design Concept which is designing the project. We prepared a detailed design for “Draw It!” application. Our report has two main parts one is Requirement Analysis and the other is Analysis section.

In the Requirement Analysis section, we first give an overview about our application. We explained Functional, Non-Functional Requirements and the application’s constraints. Our criteria was to consider all possible situations for the user can perform. Therefore, we tried to find all possible situations and create scenarios based on these information. While designing our models, we always keep in mind to fulfill these requirements and scenarios we came up with. At the end of this part, we provide Use Case Model and User Interface, because the interaction between application and user is very important, we explain them as easy and as understandable as we can.

In the second part, Analysis Model, we gave more detailed explanation of our application by using Object and Dynamic Models. This part is where we gave more specific design decisions about the application. In the Object Model, we first gave the Domain Lexicon which is very essential for the understandability of our application. We tried to clarify ambiguous words and we noted our key words. By specifying these crucial words, we became ready to identify our design in a more detailed manner. We provide Class Diagram. In the Dynamic Model part, we gave Start Chart and Sequence Diagrams which are also very essential for the design of our

implementation. We tried to be as specific as we can in order to have a solid design that we can implement.

To sum up, our purpose to writing this report was to design and implement our application easily in the future. We tried to make it as precise as possible in order to better understand our problem and solution domains. Only by thinking deeply about the process, we can come up with a better design and solution. This report will be our future reference and guide for our following process in the “Draw It!” project. That is why we understand the importance of this report and we took it seriously and tried our best.