

RESEARCH & PROJECT SUBMISSIONS



Program: CESS

Course Code: CSE334

***Course Name: Internet
Programming***

Examination Committee

Dr. Hisham Farag

**Ain Shams University
Faculty of Engineering
Spring Semester – 2020**



Student Personal Information for Group Work

Student Names:

Ahmed Mamdouh Mohammed

Nada Tarek

Student Codes:

16P6020

16P6053

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: Ahmed Mamdouh and Nada Tarek

Date: 21-5-2020

Submission Contents

01: Abstract	Page 3
02: Introduction	Page 5
03: Methods	Page 7
04: Requirements Analysis	Page 10
05: Design	Page 22



01

ABSTRACT



1.0 ABSTRACT

This report discusses in detail the development process of a matching cards web game. In the beginning, we introduced the game details, rules, the effect it has in improving the player's memory over time and the background knowledge needed to develop such a game. Afterwards, we discussed the approaches associated with developing such a game, then we wrote down the functional and non-functional requirements that are associated with this game. Afterwards, our architectural, data and component designs of this gaming system are discussed in detail with diagrams that show the exact design we had in mind when we developed this game and we showed screenshots of the user interface, then we attached the actual code of our game which we wrote based on the design diagrams. The code files are basically one HTML file, one CSS file and one JavaScript file. In the end, we showed screenshots of the game while it's being played to show the different states of the system, also we included all the sources and references we used. Each team member had specific roles that are listed in the table below.

Team Member's Name	Implementation Roles	Documentation Roles
Ahmed Mamdouh	HTML, restart button functionality, cards functionality and game logic.	Abstract, methods sections and we have both brainstormed and collaborated in the requirements and design sections.
Nada Tarek	CSS, start button functionality and attaching images to cards randomly.	Introduction, references, implementation sections and we have both brainstormed and collaborated in the requirements and design sections.

Table 1: Team Members' Roles



02

INTRODUCTION



2.0 INTRODUCTION

This memory matching cards game is a game in which there is a matrix of hidden images where each image has a copy of itself. Two players take turns to try and find as many matching images as possible before all cards run out. There are three levels for this game, they are:

- 1- Easy (matrix of 4x4 images)
- 2- Medium (matrix of 4x5 images)
- 3- Hard (matrix of 4x6 images)

The score of each player increases by two whenever he/she finds matching images. The player with a bigger score at the end of the game is the winner. The game is very important whether it is for improving your memorizing skills or for learning more about the development process of a web application using CSS, HTML and vanilla JavaScript. The idea we had to develop this game is to create a submission form which takes the players' names and avatars as input and the level (easy, medium or hard) of their choice. Afterwards, the user clicks on start which shows the game board containing the matrix of hidden images, the names and avatars of the players, the scores of the players, the current player's turn and a restart button which when clicked takes you back to the original submission form. When the game ends, either a celebration image appears to celebrate one of the two player's victory or a sad emoji image appears when the game ends in a draw.



03

METHODS



3.0 METHODS

This section discusses the various approaches and techniques that could be used to develop the matching cards memory game.

The first approach would be to use HTML and CSS to construct the user interface and vanilla JavaScript with the Document Object Model (referred to as DOM) to control and manipulate the HTML code based on the user's actions. The DOM allows us to manage the game logic whenever the user presses any of the hidden images or the start/restart button. However, this approach does not allow us to store the players' information as there is no backend server available, so players cannot create accounts and save their data, scores and results on the website. This approach is the one we decided to use in our project as creating user accounts is not needed in the requirements of this project.

The second approach would be to use JavaScript frameworks such as jQuery and React instead of vanilla JavaScript which would make the written JavaScript code a lot less than the first approach as these frameworks provide a lot of APIs to make it easier for us to code. Also, in an approach like this, Node.js framework could be used which is a JavaScript runtime environment which allows JavaScript to be used for backend development which could give us the option to allow the users to create accounts storing their data, scores and results. We decided not to use this approach as it was too advanced and not really needed in a web application as simple as this game is.



The third approach would be to use either the first approach or the second approach, but in addition, we would use a machine learning/deep learning JavaScript framework such as TensorFlow.js or Brain.js which would allow users to play against the AI in addition to playing against other players. The AI would have different difficulty levels. We decided not to use this approach as the required mode is to allow two players to play against each other.



04

Requirements Analysis



4.0 REQUIREMENTS ANALYSIS

4.1 Introduction to The Requirements

The function of this system is mainly to provide a smooth gaming experience following the previously mentioned rules of the matching cards memory game and it should be played by two players who take turns playing and competing against each other.

The scope is as the following, the two players should be able to choose their names and display picture. There should be three levels to choose from. There should be a start button that submits the choices that the players have chosen and starts the gaming session. Once the gaming session starts, the hidden images appear, and players take turns to get a higher score. Once a matching image pair is found, users cannot choose the same images again. The game ends when the cards run out and we either have a draw or one player would be the winner. There should be a restart button that allows the users to go back to the submission form at any time during the gaming session.

The success criteria are that the game should run smoothly without any bugs or errors with a nice looking and friendly user interface that isn't too complex for the user.

Some technical functionalities:

- 1- In the start and restart button, the DOM should be used to modify the HTML once any of the two buttons is clicked. If the start button is clicked, the HTML should be changed to show the matrix of hidden images. If the restart button is clicked, the HTML should be changed to make the image matrix disappear and show the submission form again, but it should also keep their previously chosen



names and display pictures instead of letting the users type their names and choose their display pictures again.

- 2- When a player clicks on a card, the image hidden in this card should be revealed and kept visible until the player clicks on another card and reveals another image. This is done by using a counter variable which is equal to 0 when it's the player's first click and it should be equal to 1 when it's his/her second click. The two images are then compared. If they match, then the player gets two points and the cards cannot be clicked on again however, if they don't match, the two images remain visible for 1 second for both players to be able to memorize them. After the second has passed, the images become hidden again and the player's turn is over.

4.2 Functional Requirements

The functional requirements of the matching cards game will be discussed in detail in this subsection. The functional requirements of our system are **a scoring system, multiple levels of difficulty, choosing a name and a display picture, a restart game functionality, game logic and announcing game results.**

4.2.1 Scoring System

This requirement is extremely essential for our game to work as expected. Whenever a player finds a matching pair of images, their score increases by two. If there is no match between the images, the player's score shouldn't change. By the end of the game session, the player with the higher score is the winner. If both players have the same score, it's a draw. During the game session, the score of each player should be visible.

4.2.2 Multiple Levels of Difficulty

This requirement states that there must be various difficulty levels that the players can choose from. The easiest level should have a 4x4 image matrix which means that the game board should have 8 hidden image pairs (a total of 16 images). As the difficulty increases, the number of images in the game board should increase which would make the game harder for the players. We decided to have three levels: easy (16 total images), medium (20 total images) and hard



(24 total images). The user should be allowed to choose the level before starting a game session.

4.2.3 Choosing the Name and the Display Picture

This requirement states that the players should be allowed to choose their names and display pictures. Each player should type his/her name in a submission form. As for the display picture, the players should be allowed to choose from a set of images available by the website. During the game session, the name and display picture of each player should be visible.

4.2.4 Restart Game Functionality

This requirement states that there must be an option available to the user anytime during the game session to restart the current game session. It should remove the hidden images and take the users back to the submission form where they can choose their names, display pictures and/or the game level. When a restart is provoked, the system should be able to store the previously chosen names and display pictures and include them in the submission form to make it easier for the players to restart the game as fast as possible. They should be still allowed to change them if they wanted to.

4.2.5 Game Logic

This requirement states that the proper game rules and mechanics should be applied. The game's logic consists of two parts, the first part is revealing the hidden images once a card is clicked by a player. The second part is comparing the two revealed images. If the two images match, the player should earn two points. If the two images don't match, the player doesn't earn any points. Players take turns and compete to earn more points. When revealing the first image, the image should remain visible until the second image is also revealed. When comparing the two images, the two images remain visible for one second to allow the players to memorize their positions. If the two images match, they are discarded from the game board. If the two images don't match, they become hidden again in the same position.

4.2.6 Announcing Game Results

This requirement states that at the end of the game session, the results should be announced in a certain way. If any of the two player wins the game, a celebrating image should be visible with a sentence which states he/she was victorious. If the game ends in a draw, a sad emoji image should be visible with a sentence which states the game has ended in a draw.



4.3 Non-Functional Requirements

The non-functional requirements of the matching cards game will be discussed in detail in this subsection. The non-functional requirements of our system are **required resources, response time, reliability, usability, reusability and platform constraints**.

4.3.1 Required Resources

This requirement states that the game shouldn't take up more than **512 megabytes of RAM** in order to function smoothly and correctly.

4.3.2 Response Time

This requirement states that the maximum time difference between when a user clicks a button and when the function of that button is completed is **one second**.

4.3.3 Reliability

This requirement states that the game **mustn't tolerate any type of faults or failures**. As this game is fairly simple, this requirement is possible to complete.

4.3.4 Usability

This requirement states that the user **shouldn't click more than 3 clicks to use any game functionality** (Start game, restart game, select level, type name, choose display picture, reveal hidden image and compare revealed images).

4.3.5 Reusability

This requirement states that at least **50% of the code of this game system must be reusable**. The main screen of this game system, the start and restart buttons are completely reusable in any type of game. Any game could have a level, an option to type the user's name and an option to choose a certain display picture. Any game could also have a start and restart game functionality.

4.3.6 Platform Constraints

This requirement states that the web application must be able to run with appropriate page layout and full functionality on the **Google Chrome** browser.

4.4 System Models

This section includes the use cases, scenarios and dynamic model of our game system. It also includes the user interface structure and the navigations between different screens. As there are no objects in our simple game, we aren't going to construct an object model.

4.4.1 Use Cases

This game has various functionalities as we discussed in detail before. Users interact with the system to make use of these functionalities. The best way to visualize the interactions between the users and the system is to construct a **use case diagram**.

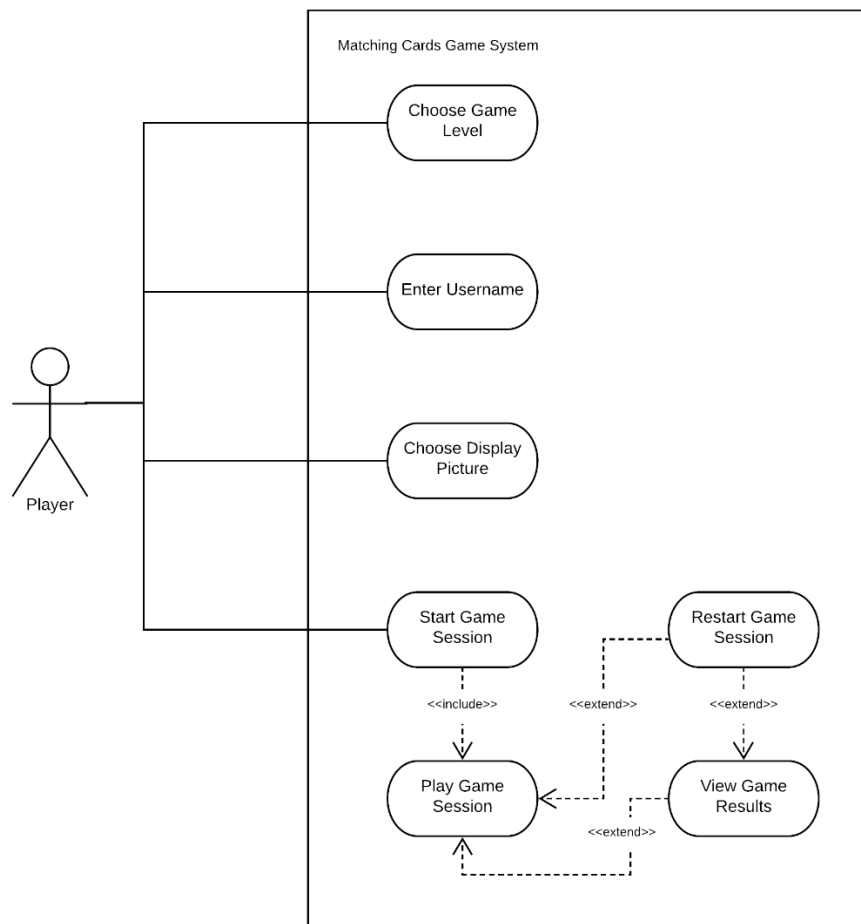


Figure 1: Use Case Diagram



As we can see, the player initiates most of the interactions with the system. When the Player presses start, the cards are shown on the game board, and players take turns to earn more points. During the game session, the player could either press the restart button to restart the game session or he/she could wait for the game to end and the results are announced. The restart button could still be pressed after the results are announced.

4.4.2 Scenarios

This game has mainly two different scenarios that could occur if we look at the system from a high-level viewpoint. The possible scenarios are written in detail below.

Normal Scenario

1. User chooses the level of the game.
2. Each player enters his username.
3. Each player chooses his own avatar/display picture.
4. One of the players clicks Start.
5. Then the match begins as the player starts by flipping a card then flipping another and trying to find a card that has the same image as the first , If he can't find a pair, the flipped cards will be flipped back with the face down.
6. The players have the ability to restart the game after the game ends and the winner is announced.
7. In case the two players scored equal points then this would be considered draw then there won't be a winner.

Exception Scenario

1. User chooses the level of the game.
2. Each player enters his username.
3. Each player chooses his own avatar/display picture.
4. One of the players clicks Start.
5. Then the match begins as the player starts by flipping a card then flipping another and trying to find a card that has the same image as the first , If he can't find a pair, the flipped cards will be flipped back with the face down.
6. The players have the ability to click restart before beginning the game or in the middle of the game in order to return to the main menu and change the level, avatar or even the username.

4.4.3 Dynamic Model

The dynamic model shows all the different states of the system and the transitions from states to one another and what triggers these transitions. The dynamic model is best visualized by a state diagram.

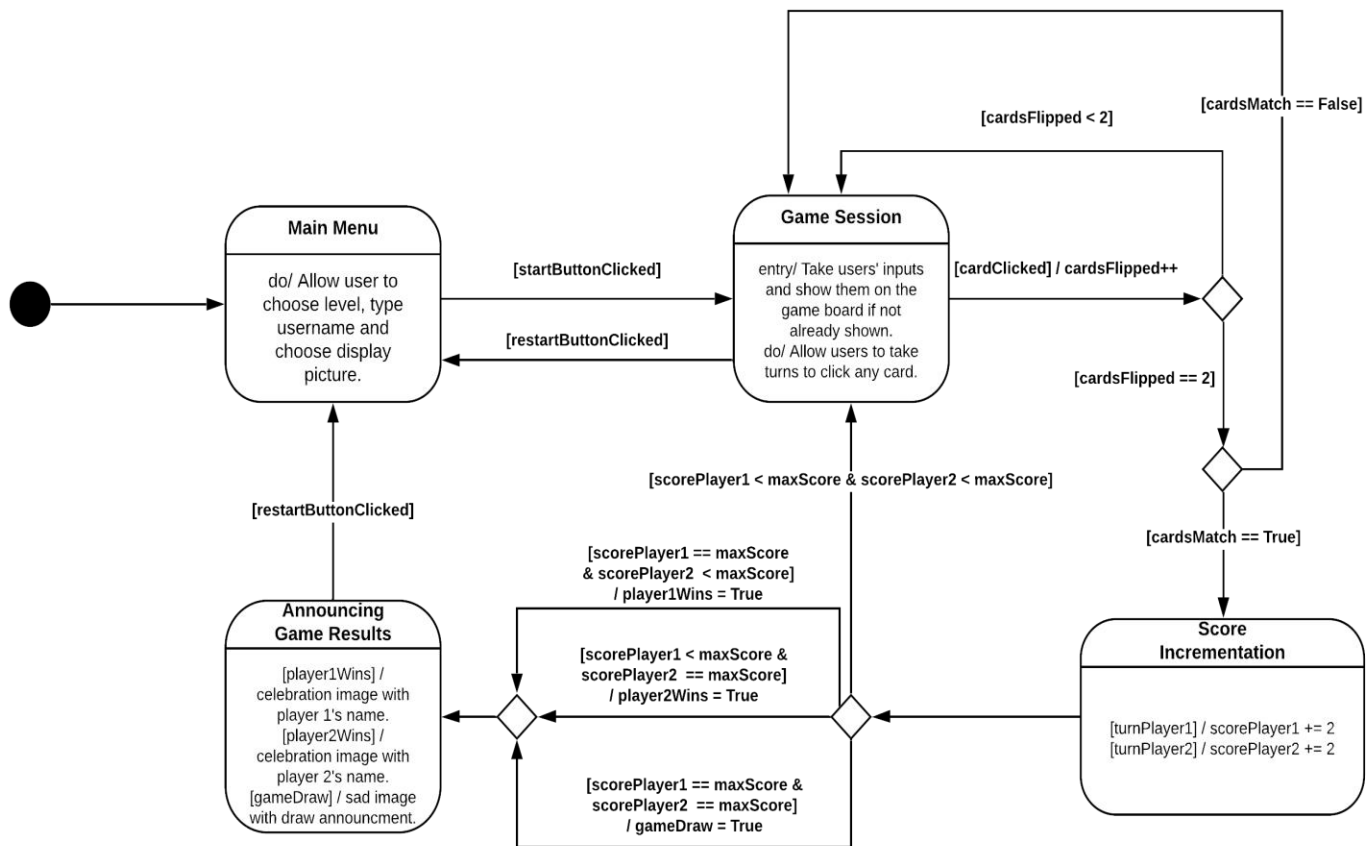


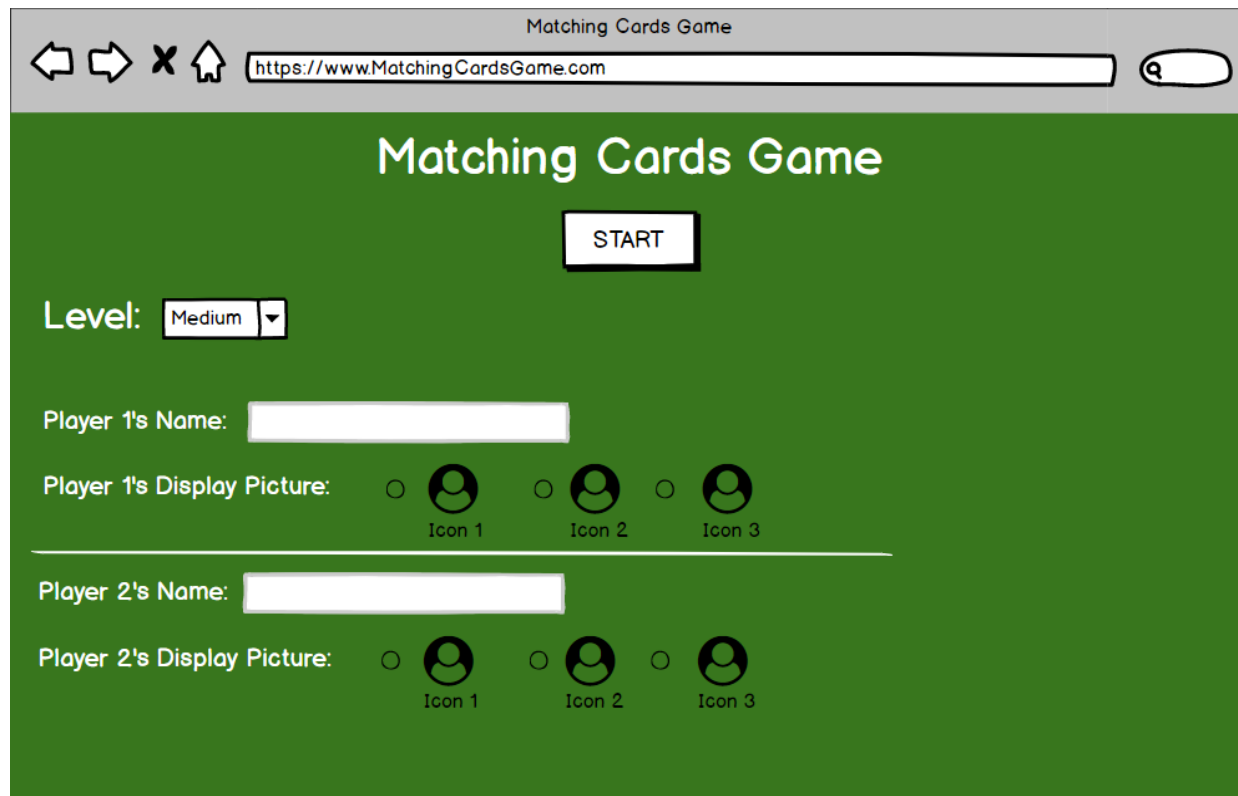
Figure 2: Dynamic Model

4.4.4 User Interface Mock-Up and Navigation

This game has three different screens, they are the main menu screen, the game session screen and the game results screen. We have constructed a mock-up for the user interface for each screen and the navigations between the screens will be detailed.

Main Menu Screen:

The first screen that appears when the game's website is opened is the main menu screen. In this screen, there should be a select game level option, an option for entering the username for both players, an option to choose the display picture for each player and there should be a start button which navigates to the game session screen. The figure below contains the mock-up for the main menu (This is not the actual user interface).



The mock-up shows a web browser window titled "Matching Cards Game" with the URL "https://www.MatchingCardsGame.com". The main content area has a green background and features the title "Matching Cards Game" in large white text. Below the title is a "START" button. Underneath is a "Level:" label followed by a dropdown menu currently set to "Medium". Below this are two sections for player information. The first section is for "Player 1's Name:" with a text input field, followed by "Player 1's Display Picture:" with three radio button options labeled "Icon 1", "Icon 2", and "Icon 3". A horizontal line separates this from the second section for "Player 2's Name:" with another text input field, followed by "Player 2's Display Picture:" with three radio button options labeled "Icon 1", "Icon 2", and "Icon 3".

Figure 3: Mock-up for Main Menu

Game Session Screen:

When the user clicks the start button in the main menu screen, this is the screen that appears. This screen contains the matrix of hidden images, the two players names and icons, each player's score, a label which shows whose turn it is and a restart button that when clicked navigates to the main menu screen. When the game ends, the system navigates to the game results screen which shows who won (or declares a draw if both players have equal points). The figure below contains the mock-up for the game session screen (**This is not the actual user interface**).

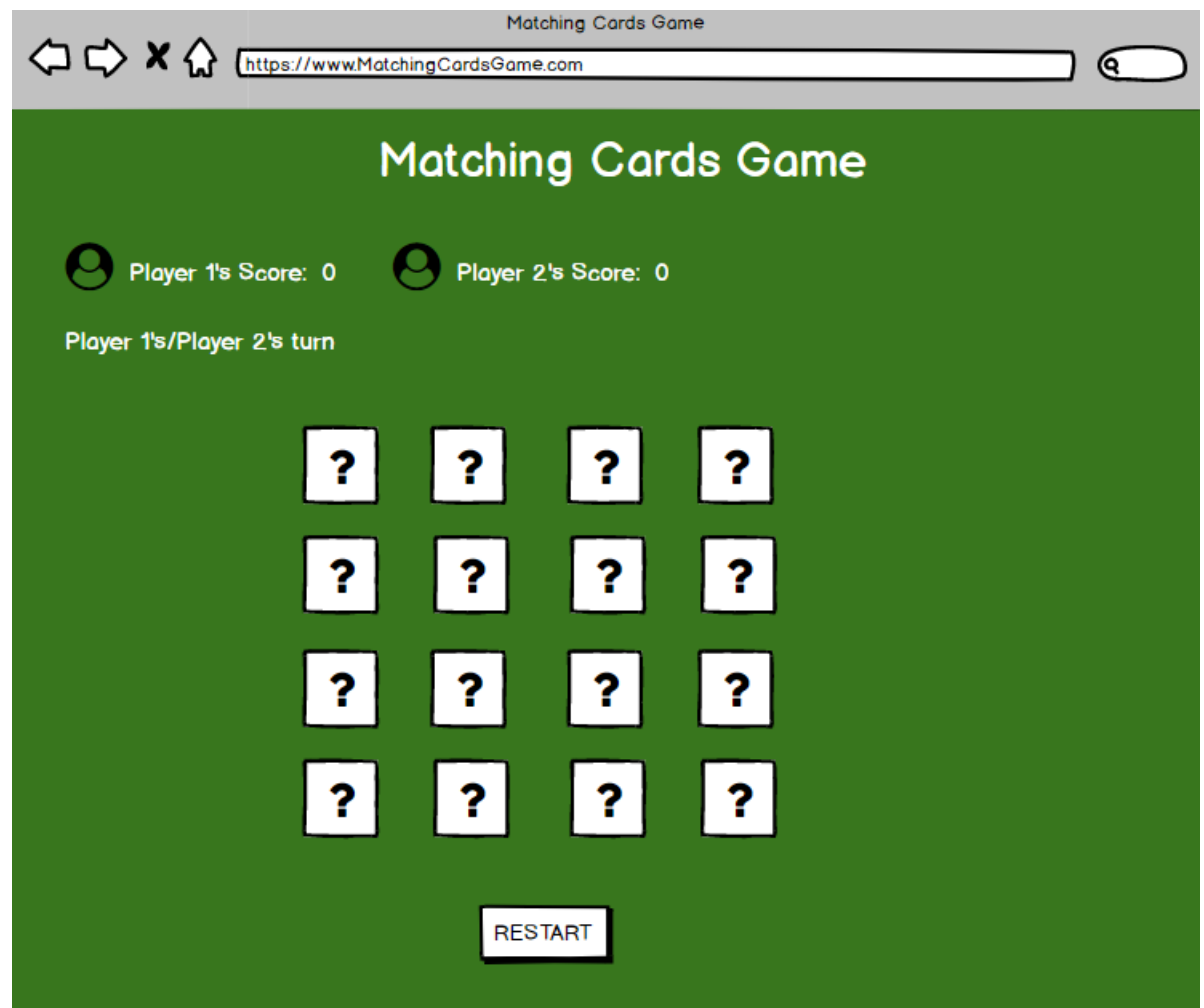


Figure 4: Mock-up for Game Session Screen

Game Results Screen:

When the game session ends, the website navigates to the game results screen which contains the result of the game. If a player won, there should be a sentence declaring the player who won and a celebrating image. If it was a draw, there should be a sentence declaring it was a draw and a sad emoji image. There should also be a restart button that is used to navigate to the main menu screen if needed. The figure below contains the mock-up for the game results screen (**This is not the actual user interface**).

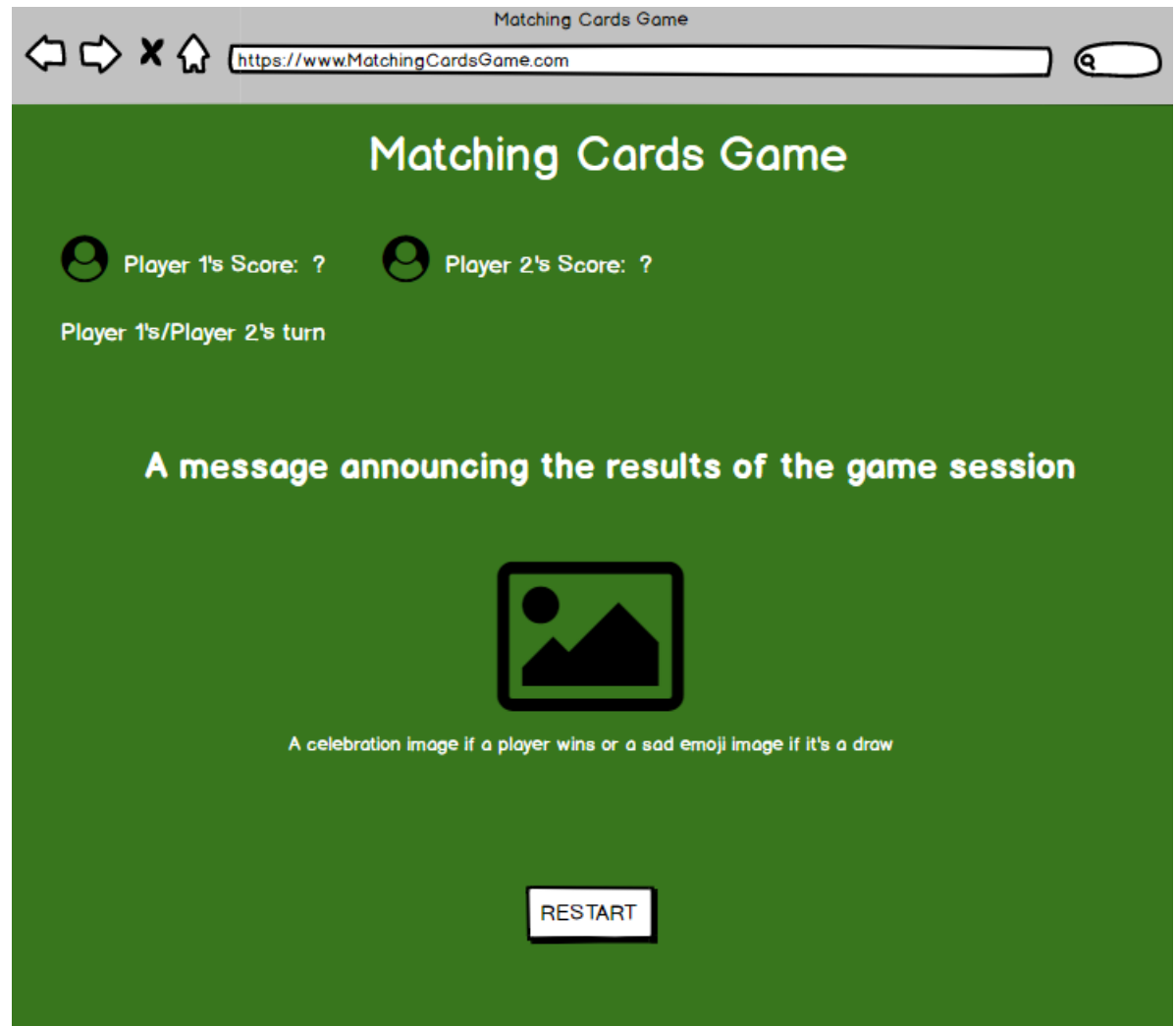


Figure 5: Mock-up for game results



Navigations:

All the navigations between different screens in our system are detailed in the table below.

<i>Navigate from</i>	<i>Navigate to</i>	<i>Method</i>
Main menu screen	Game session screen	Start button
Game session screen	Main menu screen	Restart button
Game session screen	Game results screen	Game session ending
Game results screen	Main menu screen	Restart button

Table 2: Screen Navigations Table



05

DESIGN

5.0 DESIGN

5.1 Architectural Design

The architectural design shows the various subsystems that the game system is composed of. Also, it shows how these various subsystems interact with each other to achieve the desired functionality.

The diagram below introduces the major subsystems that our system is decomposed into.

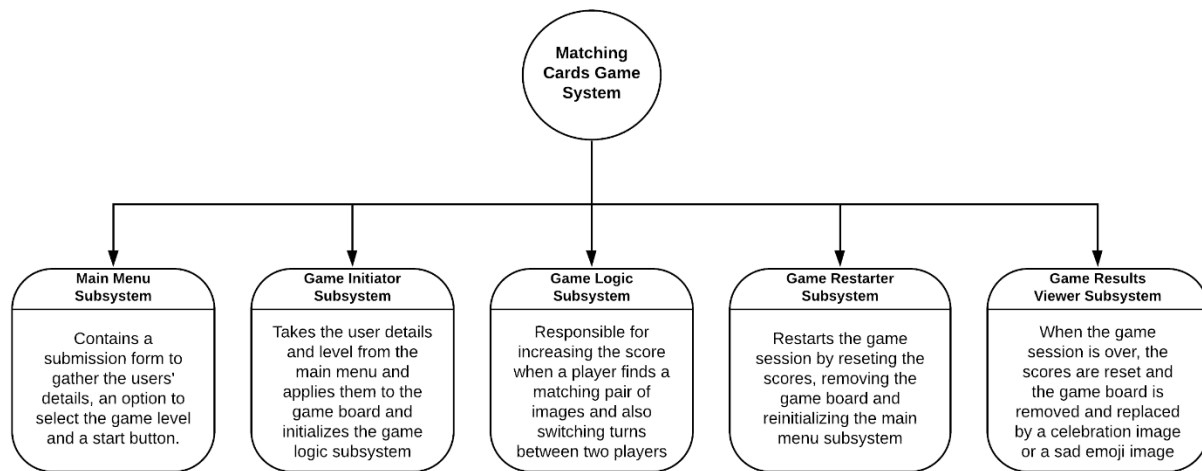


Figure 6: Modular System Structure

To add more details to the above diagram, we made another diagram which shows the interactions between these subsystems and each other and it also shows the interactions between these subsystems and the data repository.

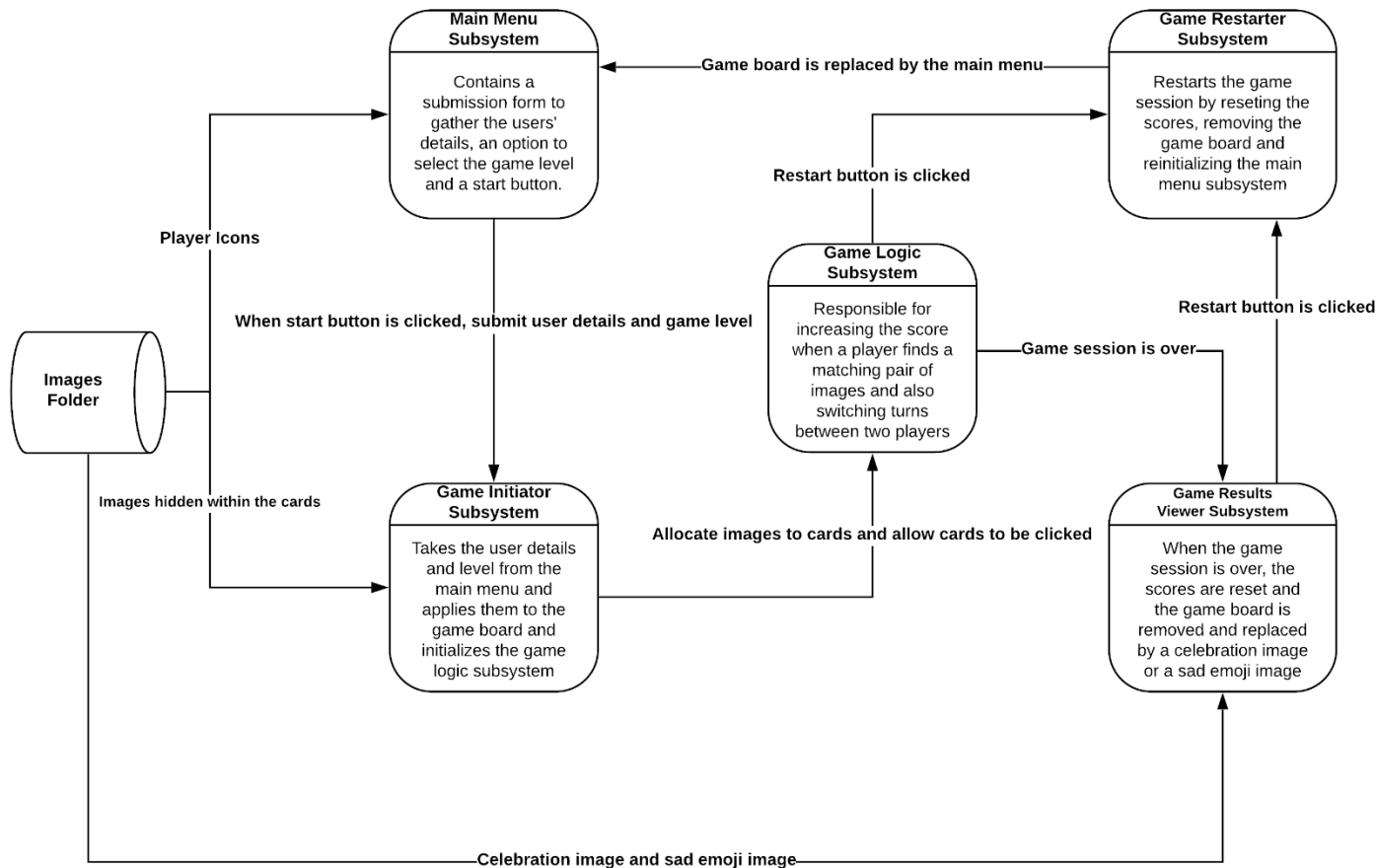


Figure 7: Detailed Modular System Structure

5.2 Data Design

The data involved in our game system are basically the images. The images could be player icons, hidden within the cards during the game session or viewed to celebrate a victory or sorrow over a draw.

5.2.1 Player Icons

The player icons available are 3 different images that each player gets to choose from. The images are loaded directly onto the main menu where there are radio buttons to allow each player to pick one of the images as their player icon during the game session. The image selected will be displayed beside the player's name during the entire game session. There was no need to use a data structure to store the player icons as the number of all possible player icons is limited.

5.2.2 Images Hidden Within the Cards

These images are what the players try to find matching pairs of. When the user presses the start button, these images are loaded into an array and then each element in that array is duplicated. The elements of the array are then shuffled using a shuffling algorithm. Finally, each image in this array is given a certain position on the game board, but they are not shown to the user. Each image position is basically a card with a question mark on it. As the level increases, the number of images that will be loaded into the array will increase.

5.2.3 Game Results Images

There are two possible images that could be shown during the announcement of the game results. They are a celebration image or a sad emoji image. The celebration image is shown when one player is victorious over the other while the sad emoji image is shown when there is a draw. The images are loaded directly from the folder during the end of the game session. There was no need to use a data structure.

5.3 Component Design

This section goes into the details of how each functionality we included in the requirements section is fulfilled. All functionalities are fulfilled using five components. The components of our system are **a main menu component, a game initiator component, a game logic component, a restart game functionality component and an announcing game results component.**

5.3.1 Main Menu Component

This component is responsible for providing a user interface which allows the user to choose the game level, enter the players' names and choose the players' icons. The three game levels available are easy, medium and hard. They will be placed in a selection list as options. The players' names will be entered by the user through a text field. The players' icons will be shown as options where the user can choose a certain player icon by clicking the respective radio button. The start button initiates the game initiator component once clicked. The previous functionalities we mentioned are done through basic HTML without writing any JavaScript code.

5.3.2 Game Initiator Component

This component is initiated when the start button is clicked. It's responsible for taking the input that the user entered in the main menu component and viewing it in a certain way in the game board. It's also responsible for showing a number of cards on the game board and allocating an image randomly to each card. The number of cards visible depends on the game level chosen by the user. There's also a restart button available that initiates the restart game functionality component once clicked. The pseudocode below shows how the functionalities are completed using this component. There are comments which help in understanding the pseudocode and add more details.

```
GameInitiatorComponent():

    // Reading players' names using DOM
    player1Name = document.getElementById('player1name').value
    player2Name = document.getElementById('player2name').value

    // Reading the selected user avatar using DOM
    let avatar1Elements = document.getElementsByClassName('avatar1')
    let avatar2Elements = document.getElementsByClassName('avatar2')
    for (i = 0; i < avatar1Elements.length; i++) {
        if (avatar1Elements[i].checked)
            player1Avatar = avatar1Elements[i].value
        if (avatar2Elements[i].checked)
            player2Avatar = avatar2Elements[i].value
    }

    // Removing the main menu interface using DOM
    clearMainMenu()

    // Show the player names, scores and whose turn it is on the game board using
    DOM
    showPlayerInfo(player1Name, player2Name, player1Avatar, Player2Avator)

    // Loading images. The imagesArray contain the paths of every image available
    in the game
    imagesArray = loadAllImages()

    // HTML for each game level
    gameHTMLEasy = getGameBoardHTML('easy') // 16 cards
    gameHTMLMedium = getGameBoardHTML('medium') // 20 cards
    gameHTMLHard = getGameBoardHTML('hard') // 24 cards

    // Get chosen game level by the user
```



```
level = getGameLevel()

// Modify game board based on the chosen level
if (level == 'easy') {
    element.innerHTML = gameHTMLEasy
    imagesPaths.splice(8, 11) // keep only 8 images
} else if (level == 'medium') {
    element.innerHTML = gameHTMLMedium;
    imagesPaths.splice(10, 11) // keep only 10 images
} else if (level == 'hard') {
    element.innerHTML = gameHTMLHard; // keep all images
}

// Allocate a position for each of the images in the images matrix
imageMatrix = new Array(imagesArray.length * 2)
for (i = 0; i < imageMatrix.length; i++) {
    if (i < imageMatrix.length / 2)
        imageMatrix[i] = imagesArray[i]
    else {
        imageMatrix[i] = imagesArray[i - (imageMatrix.length / 2)]
    }
}

// Shuffling the images matrix so that the positions are random at each
gaming session
shuffleArray(imageMatrix)

// Adding functionality to each card
cardClassElements = document.getElementsByClassName("card")
for (i = 0; i < cardClassElements.length; i++) {
    cardClassElements[i].onclick = GameLogicComponent(i, imageMatrix)
}

// Show the restart game button
element = document.getElementById("restartGame");
element.innerHTML = showRestartButton()

end GameInitiatorComponent
```

5.3.3 Game Logic Component

This component is responsible for switching turns between the two players, increasing a player's score when he/she finds a matching pair of images, removing the found matching pair of images from the game and ending the game when the images run out. This component is initiated when a card is clicked during the game session. The pseudocode below shows how the functionalities are completed using this component. There are comments which help in understanding the pseudocode and add more details.

```
GameLogicComponent(cardIndex, ImagesMatrix):
    // This function contains the game logic and the events that occur when
    // players click the cards
    // It takes two parameters, the first is the index of the card that is
    // clicked.
    // The second one is the images matrix where each card index correspond to an
    // image

    currentPlayer = 1 // variable which determines whose turn it is
    counter = 0 // variable which determines if it's the first clicked card or
    the second clicked card
    if (currentPlayer == 1) { // Player one's turn

        if (counter == 0) { // first card clicked

            // Getting the cards elements from the document
            let cards = document.getElementsByClassName('card')

            // Modifying the clicked card's image
            cards[clickedCardIndex].innerHTML =
getHTMLofImage(imagesMatrix[clickedCardIndex])
            firstClickedCard = imagesMatrix[clickedCardIndex]
            previousImagePosition = clickedCardIndex
            counter = 1
        }
        else if (counter == 1) { // second card clicked

            if (clickedCardIndex == previousImagePosition) { // If player clicked
            same card, do nothing
                return
            }

            // Getting the cards elements from the document
            cards = document.getElementsByClassName('card')

            // Modifying the clicked card's image
```



```
        cards[clickedCardIndex].innerHTML =
getHTMLOfImage(imagesMatrix[clickedCardIndex])
        secondClickedCard = imagesMatrix[clickedCardIndex]

        // Checking if the two visible images match
        if (firstClickedCard == secondClickedCard) { // Matching images
            waitOneSecond()
            cards[previousImagePosition].innerHTML = ''
            cards[clickedCardIndex].innerHTML = ''
            document.getElementById("turn").innerText = turnPlayer2
            counter = 0
            currentPlayer = 2 // switch turns
            scorePlayer1 += 2 // Increase score of the player
            scoreValue1 = document.getElementById("scoreValue1")
            scoreValue1.innerText = "" + scorePlayer1
        }
        else { // Images don't match
            waitOneSecond()
            cards[previousImagePosition].innerHTML = hideImage()
            cards[clickedCardIndex].innerHTML = hideImage()
            document.getElementById("turn").innerText = turnPlayer2
            counter = 0
            currentPlayer = 2 // switch turns
        }
        GameResultsComponent(scorePlayer1, scorePlayer2)

    else if(currentPlayer == 2){ // Player 2's turn
        if (counter == 0) { // first card clicked

            // Getting the cards elements from the document
            let cards = document.getElementsByClassName('card')

            // Modifying the clicked card's image
            cards[clickedCardIndex].innerHTML =
getHTMLOfImage(imagesMatrix[clickedCardIndex])
            firstClickedCard = imagesMatrix[clickedCardIndex]
            previousImagePosition = clickedCardIndex
            counter = 1
        }
        else if (counter == 1) { // second card clicked

            if (clickedCardIndex == previousImagePosition) { // If player clicked
same card, do nothing
                return
            }
        }
    }
```

```
// Getting the cards elements from the document
cards = document.getElementsByClassName('card')

// Modifying the clicked card's image
cards[clickedCardIndex].innerHTML =
getHTMLOfImage(imagesMatrix[clickedCardIndex])
secondClickedCard = imagesMatrix[clickedCardIndex]

// Checking if the two visible images match
if (firstClickedCard == secondClickedCard) { // Matching images
    waitOneSecond()
    cards[previousImagePosition].innerHTML = ''
    cards[clickedCardIndex].innerHTML = ''
    document.getElementById("turn").innerText = turnPlayer1
    counter = 0
    currentPlayer = 1 // switch turns
    scorePlayer2 += 2 // Increase score of the player
    scoreValue2 = document.getElementById("scoreValue2")
    scoreValue2.innerText = "" + scorePlayer2
}
else { // Images don't match
    waitOneSecond()
    cards[previousImagePosition].innerHTML = hideImage()
    cards[clickedCardIndex].innerHTML = hideImage()
    document.getElementById("turn").innerText = turnPlayer1
    counter = 0
    currentPlayer = 1 // switch turns
}
GameResultsComponent(scorePlayer1, scorePlayer2)
}
end GameLogicComponent
```

5.3.4 Restart Game Functionality Component

This component is responsible for restarting the state of the game system and returning to the main menu. The scores of both players gets reset and the game board is removed. This component gets initiated when the restart button gets clicked. The pseudocode below shows how the functionalities are completed using this component. There are comments which help in understanding the pseudocode and add more details.



```
RestartGameComponent(player1Name, Player2Name, player1Avatar, player2Avatar):

    // Reset the scores and make current the current turn belong to player 1
    scorePlayer1 = 0
    scorePlayer2 = 0
    currentPlayer = 1

    // Make the restart button disappear when it is clicked
    let element = document.getElementById("restartGame")
    element.innerHTML = ""

    // Make the level options appear when restart is clicked
    element = document.getElementById("level")
    element.innerHTML = getLevellistHTML()

    // Make the score of each player and whose turn it is disappear
    element = document.getElementById("score")
    element.innerHTML = ''
    element = document.getElementById("turn")
    element.innerHTML = ''

    // Show the name and avatar submission form
    element = document.getElementById("game");
    element.innerHTML = getSubmissionFormHTML()

    // Cache the names and avatar choices of the previous players in the form
    let avatar1Elements = document.getElementsByClassName('avatar1')
    let avatar2Elements = document.getElementsByClassName('avatar2')
    for (i = 0; i < avatar1Elements.length; i++) {
        if (avatar1Elements[i].value == player1Avatar)
            avatar1Elements[i].checked = true;
        if (avatar2Elements[i].value == player2Avatar)
            avatar2Elements[i].checked = true;
    }

    document.getElementById('player1name').value = player1Name
    document.getElementById('player2name').value = player2Name

    // Show the start game button
    element = document.getElementById("startGame")
    element.innerHTML = getStartButtonHTML()

end RestartGameComponent
```

5.3.5 Announcing Game Results Component

This component is responsible for comparing the scores of both players at the end of the game. If one player has a higher score than the other player, then this player is the winner of the game. If both players have the same scores, then it's a draw. A celebration image is shown if there's a winner, while a sad emoji image is shown when it's a draw. This component is initiated when the game session is over. The pseudocode below shows how the functionalities are completed using this component. There are comments which help in understanding the pseudocode and add more details.

```
GameResultsComponent(scorePlayer1, scorePlayer2, imageMatrix):
    if ((scorePlayer1 + scorePlayer2) == imageMatrix.length) { // Condition that
is true when the game board runs out of images
        if (scorePlayer1 == scorePlayer2) { // Condition that is true when both
players have the same score at the end
            waitOneSecond()
            let gameHTML = document.getElementById("game")
            gameHTML.innerHTML = getSadEmojiHTML() // Remove the game board and
show a sad emoji image declaring that it's a draw
            [scorePlayer1, scorePlayer2] = [0, 0] // reset players' scores

        }

        else if (scorePlayer1 > scorePlayer2) { // Condition that is true when
player 1 has a higher score than player 2
            waitOneSecond()
            let gameHTML = document.getElementById("game")
            gameHTML.innerHTML = getSadCelebrationHTML(player1Name) // Remove the
game board and show a fireworks image declaring that player 1 has won
            [scorePlayer1, scorePlayer2] = [0, 0]

        }

        else if (scorePlayer1 < scorePlayer2) { // Condition that is true when
player 2 has a higher score than player 1
            waitOneSecond()
            let gameHTML = document.getElementById("game")
            gameHTML.innerHTML = getCelebrationHTML(Player2Name) // Remove the
game board and show a fireworks image declaring that player 2 has won
            [scorePlayer1, scorePlayer2] = [0, 0] // reset players' scores

        }
    }
end GameResultsComponent
```




5.4 Human Interface Design

This section goes into the details of how the user is going to interact with our game system. The user interface is simple as it doesn't require a lot of clicks from the user to achieve a desired functionality. We're going to discuss how the user would use each screen in our interface.