Software Architecture Document

for

SOS

Version 1.0 approved

Prepared by

Kenneth Anglin	ID#:180907	Email:180907@gist.edu.cn
Ottor Mills	ID#:180917	Email:180917@gist.edu.cn
David Thomas	ID#:180912	Email:180912@gist.edu.cn
Nicoy Smith	ID#:180902	Email:180902@gist.edu.cn

Course Instructor: Thomas Canhao Xu

Course: SWEN3010

Date: October 25, 2018

Table of Contents

1 Introduction

1.1 Document overview

This document describes the architecture of the SOS system.
It describes:

- A general description of the system
- The logical architecture of software, the layers and top-level components
- The physical architecture of the hardware on which runs the software
- The justification of technical choices made
- The traceability between the architecture and the system requirements.

1.2 Abbreviations and Glossary

1.2.1 Abbreviations

- API: Application Programming Interface, a protocol used as an interface to allow communication between different components.
- CSS: Cascading-Style Sheets, document that describes the appearance of web pages
- JSON: JavaScript Object Notation, a text-based standard for human-readable data exchange.
- MVC: Model-View-Controller, a software architecture pattern that separates the physical way to store data, the business logic and the appearance to the users.
- SAD: Software Architecture Document.
- HTML: HyperText Markup Language is the standard markup language for creating web pages and web applications.
-

1.2.2 Glossary

- JavaScript: (originally) web-browser interpreted programming language for enhancing websites in a dynamic way.
- SQLite: is a relational database management system contained in a C programming library.
- Cordova: also known as apache Cordova is a mobile application development framework
- Software: is a term that can be used interchangeably with application.

1.3 Non-functional requirements

1. High performance: The system must be able to check all the grids after each play to know when the game is finish without having the users notice any delay in the game.
2. User friendly: the users should not have any problems with navigating through the game.
3. Failure tolerance: the system should be able to recover from a mishap and keep working with a simple restart.
4. Human errors: humans are the #1 source of involuntary (or voluntary) cause of problems in the systems. The system should always check the user input to avoid any error.

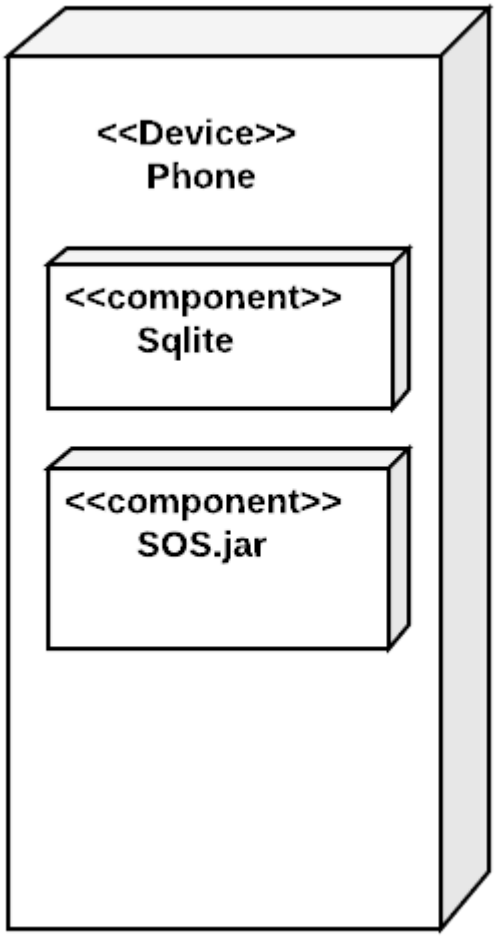
2 Architecture

2.1 Architecture overview

This software is a mobile native application that can run on both IOS and the Android platform through the use of the Cordova Bridge. It falls in the category of a gaming application that allows two to four persons to sign up before a new game is launched. Afterwards the game will display the users that are participating in the current round. Upon making a move the application will update the user interface or view to show which user should make a selection on the board. Whenever a user obtains an “SOS” in a straight line on the board three points are awarded to that user. The application will then highlight those tiles that were involved in the “SOS” so that other users are aware that those three tiles combined cannot be used to score any further points. The round can end in two ways; user selects the end button, or all the tiles on the board have been selected by the users. When the round comes to an end the application will display a scoreboard will the results of the round that was just completed. The scoreboard will also display a button that can be selected in-order to see the results of the previous game if stored in the database.

2.2 Physical architecture overview

The diagram below shows a deployment diagram for the SOS application.



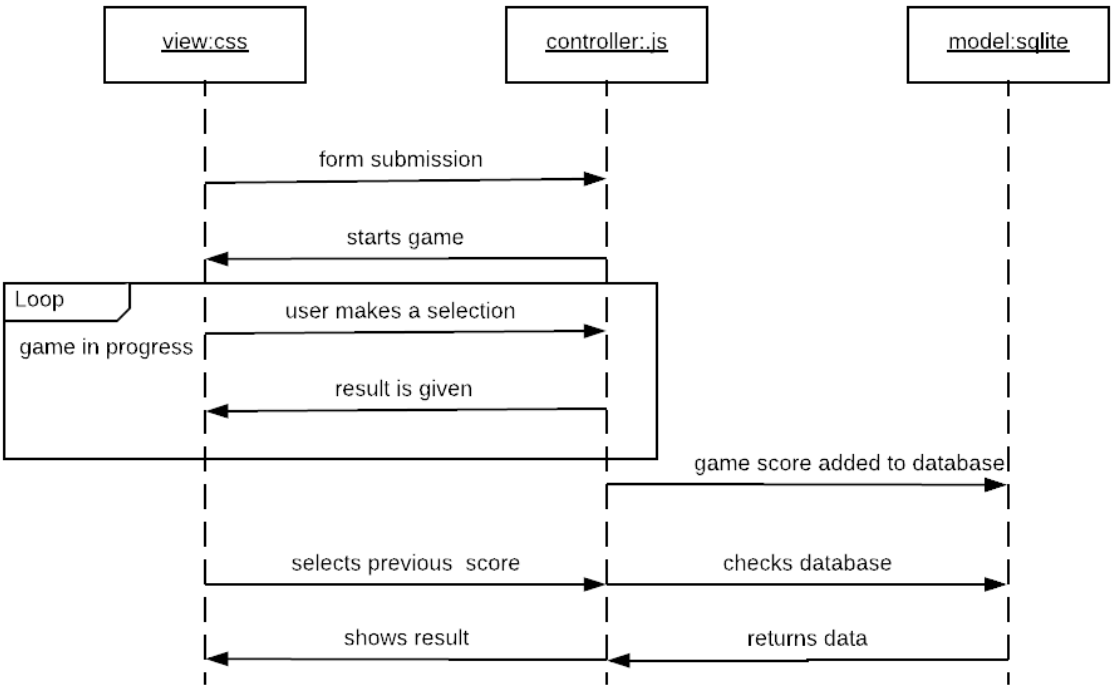
2.3 Breakdown of the architecture

This application follows the MVC architecture and an explanation for each layer is given. The main advantage in using this architecture is:

- The model and the controller layers are separated in order to improve the readability of the code which will make maintainability and upgrades easier. The model layer is used for data storage and retrieval.

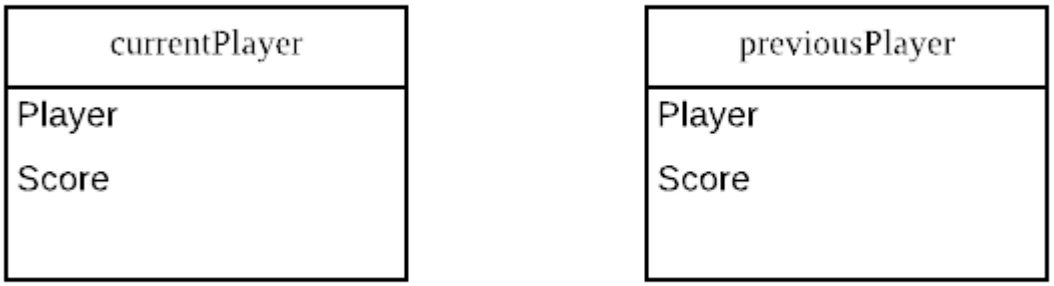
- The controller layer deals with the logical aspect of the application. This portion of the application can be reused across other component/application for data manipulation. Additionally, the traffic between the controller and model layers can be reduced to enhance the security of the data.

- The view layer is responsible for presenting the all the data and results from the logical application to the user through an interface.



2.3.1 Model Layer

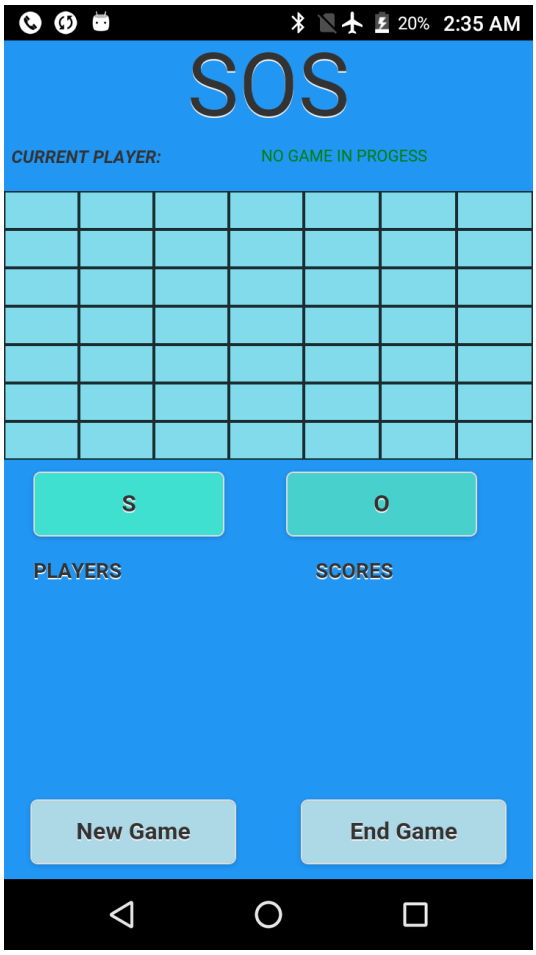
The model layer is responsible for the storage and retrieval of data in the database of the application. This layer is independent of both the control layer and view layer; therefore, the database can be used in other application views. The database server that is used in this application is the SQLite database. The model responds to the request made by the controller and displays the updated data in the view layer.

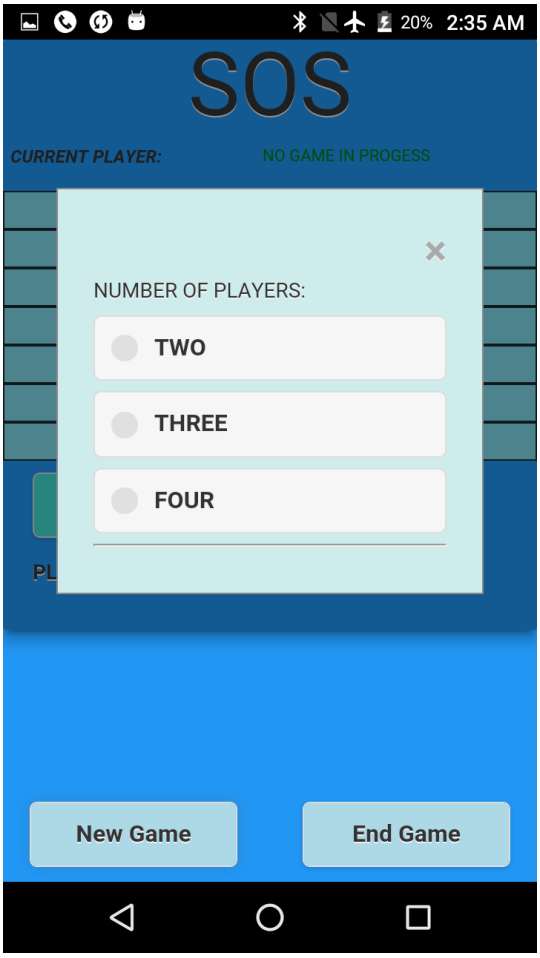


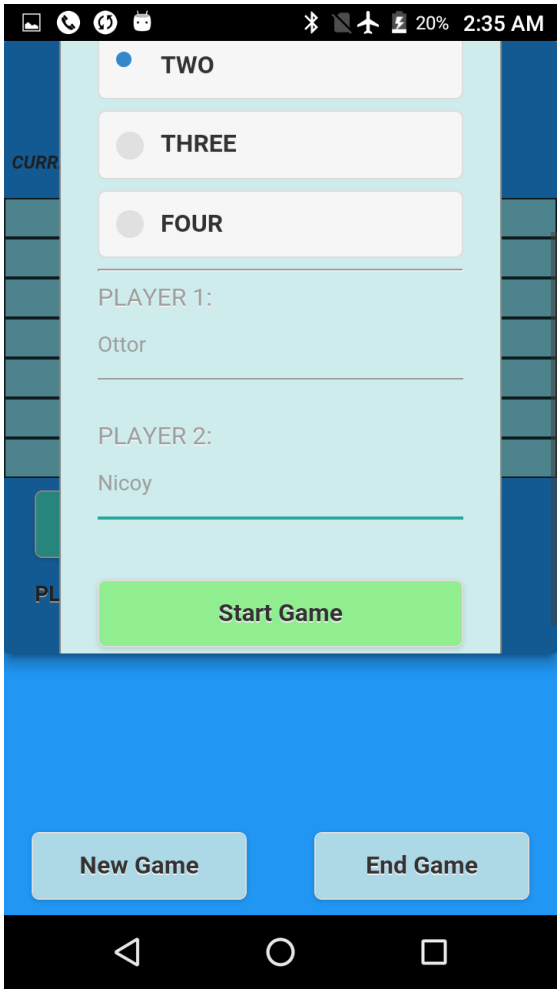
2.3.2 View Layer

This layer is used to present the application to the users in a form where the request and corresponding process results can be understood. The view layer is used in this application to show the user the player submission form, the playing board, the scoreboard etc.

Below are screenshots of the different views that the users will encounter.

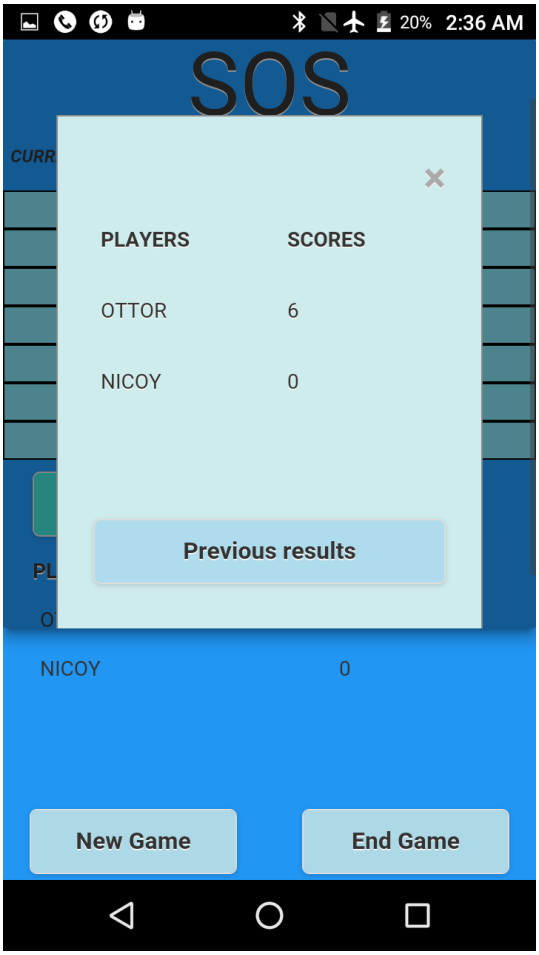





20% 2:35 AM
TWO
THREE
FOUR
PLAYER 1:
Ottor
PLAYER 2:
Nicoy
Start Game
New Game
End Game

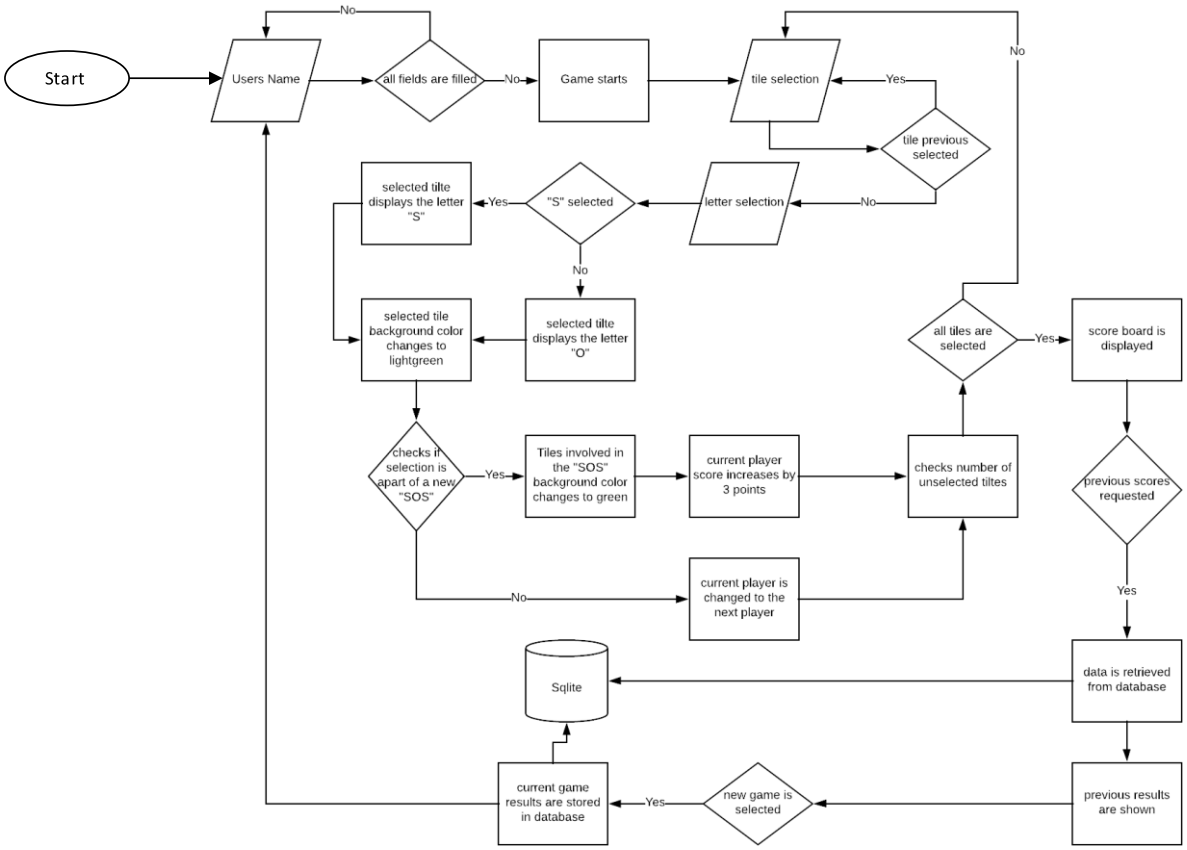

20% 2:36 AM
SOS
CURRENT PLAYER:
OTTOR
S
O
PLAYERS
SCORES
OTTOR
0
NICOY
0
New Game
End Game


20% 2:36 AM
SOS
CURRENT PLAYER:
NICOY
S
S
O
S
S
S
O
PLAYERS
SCORES
OTTOR
6
NICOY
0
New Game
End Game


20% 2:36 AM
SOS
PLAYERS
SCORES
OTTOR
6
NICOY
0
Previous results
NICOY
0
New Game
End Game

2.3.3 Controller Layer

This layer also called the business logic layer is used for the manipulation of data and requests received from the view layer.



3 Code Snippets

3.1 Model Layer

The code below shows how the 4th username variable is checked to determine if it is not empty. If it is not empty the 4th username variable along with the other 3 username variables are added to the database along with their respective scores.

```
function addDB(){

    if( document.getElementById('rname4').innerHTML != ""){
        localStorage.setItem(document.getElementById('rname1').innerHTML, document.getElementById('rscore1').innerHTML);
        localStorage.setItem(document.getElementById('rname2').innerHTML, document.getElementById('rscore2').innerHTML)
        localStorage.setItem(document.getElementById('rname3').innerHTML, document.getElementById('rscore3').innerHTML);
        localStorage.setItem(document.getElementById('rname4').innerHTML, document.getElementById('rscore4').innerHTML);
```

The code below shows how the database is searched to see if there is any data stored from the last match, if the search results is not empty the data that is retrieved.

```
function previousResults(){
    document.getElementById('lastResults').style.display = "block" ;

    if( localStorage.length == 4){

        document.getElementById('rnam1').innerHTML =  localStorage.key(0);
        document.getElementById('rnam2').innerHTML =  localStorage.key(1);
        document.getElementById('rnam3').innerHTML =  localStorage.key(2);
        document.getElementById('rnam4').innerHTML =  localStorage.key(3);
        document.getElementById('rscor1').innerHTML = localStorage.getItem(localStorage.key(0));
        document.getElementById('rscor2').innerHTML = localStorage.getItem(localStorage.key(1));
        document.getElementById('rscor3').innerHTML = localStorage.getItem(localStorage.key(2));
        document.getElementById('rscor4').innerHTML = localStorage.getItem(localStorage.key(3));
```

The code snippets above show how the model layer of the MVC model is utilized.

3.2 View Layer

The code below shows how the slots are created in the game to allow players to input “S” or “O”. A table is used to accommodate the user’s selection. Each cell is also customized so that they are uniformed in all scenarios. Each cell is given an id to be identified by the controller layer and customized if a requested is triggered from the view layer.

```
<table id="gameGrid" style=" float:left; height: 182px; background-color: green; width:100%;">
        <tr style="width:100%">
        <td style="padding:0; margin:0;"><div class="grid-item" id="1" ><p id="t1"></p></div></td>
        <td style="padding:0; margin:0;"><div class="grid-item" id="2" ><p id="t2"></p></div></td>
        <td style="padding:0; margin:0;"><div class="grid-item" id="3" ><p id="t3"></p></div></td>
        <td style="padding:0; margin:0;"><div class="grid-item" id="4" ><p id="t4"></p></div></td>
        <td style="padding:0; margin:0;"><div class="grid-item" id="5" ><p id="t5"></p></div></td>
        <td style="padding:0; margin:0;"><div class="grid-item" id="6" ><p id="t6"></p></div></td>
        <td style="padding:0; margin:0;"><div class="grid-item" id="7" ><p id="t7"></p></div></td>
        </tr>
```

The code below shows how the two input buttons are implemented. When either of these buttons are clicked by the user it will trigger a request from the view layer and be sent to the controller layer for an appropriate response.

```
<div style="background-color:#2196F3;">
    <table>
        <tr style="background-ground-color:#2196F3">
            <td style="padding:0"><button id="sos1" style="background-color:#40E0D0; width:130px; margin-left:20px; margin-bottom:0;">S</button></td>
            <td style="padding:0"><button id="sos2" style="background-color:#48D1CC; width:130px; margin-bottom:0;">O</button></td>
        </tr>
    </table>
</div>
```

The code below shows how the user interface of the application is designed so as to give it a uniform look to the user.

```
body {
    -webkit-touch-callout: none;                /* prevent callout to copy image, etc when tap to hold */
    -webkit-text-size-adjust: none;             /* prevent webkit from resizing text to fit */
    -webkit-user-select: none;                  /* prevent copy paste, to allow, change 'none' to 'text' */
    background-color:#E4E4E4;
    background-image:linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
    background-image:-webkit-linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
    background-image:-ms-linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
    background-image:-webkit-gradient(
        linear,
        left top,
        left bottom,
        color-stop(0, #A7A7A7),
        color-stop(0.51, #E4E4E4)
    );
    background-attachment:fixed;
    font-family:'HelveticaNeue-Light', 'HelveticaNeue', Helvetica, Arial, sans-serif;
    font-size:12px;
    height:100%;
    margin:0px;
    padding:0px;
    text-transform:uppercase;
    width:100%;
}
```

The code above shows how the view layer of the MVC architecture is utilized.

3.2 Controller Layer

The code below shows how the logical operations that take place when a new game is requested from the view layer which was triggered by the user. The function first checks if the request is legitimate, afterwards it assigns the values that were inputted by the users to the username variables. The scores are then set to null value to prevent it from having a previous score from the last match if there was one. The current player is then set to the person's name that was entered first in the input slot that was shown by the view layer. The Boolean legal game and picker are changed to true so that the game can start and the user can select a slot to put the letter "S" or "O".

```
function startGame(){

    if (checkValidity() == true )
    {
        person1 = document.getElementById('p1').value
        person2 = document.getElementById('p2').value
        person3 = document.getElementById('p3').value
        person4 = document.getElementById('p4').value

        document.getElementById('curplayer1').innerHTML = person1;
        document.getElementById('curplayer2').innerHTML = person2;
        document.getElementById('curplayer3').innerHTML = person3;
        document.getElementById('curplayer4').innerHTML = person4;

        //all the scores are changed changed to empty
        document.getElementById('score1').innerHTML = "";
        document.getElementById('score2').innerHTML = "";
        document.getElementById('score3').innerHTML = "";
        document.getElementById('score4').innerHTML = "";

        document.getElementById('currentPlayer').innerHTML = person1;
        legalGame = true;
        picker = true;
```

The above code demonstrates an example of the controller layer aspect of the game.

Percentage Work Contribution

Below is a table showing the percentage effort contributed by each member of the group:

Name	ID	Effort (%)
David Thomas	180912	30
Ottor Mills	180917	23.33
Nicoy Smith	180902	23.33
Kenneth Anglin	180907	23.33

Course Instructor Thomas Canhao Xu

Course SWEN3006

Date October 25, 2018