

Project Milestone Minesweeper



3815ICT Software Engineering
s5085319 Sankit Man Shrestha

1. Requirements	3
1.1 Completed functional requirements	3
1.2 Ongoing functional requirements priority list	6
2. Use Cases	7
2.1) Difficulty Selection Use Case	7
2.2) Playing Game Use Case	8
2.3) Start Timer Use Case	9
2.4) Update High Score Use Case	10
3. Software Architecture	11
3.1 Tools used	11
3.2 Feasibility to change	12
3.3 Pattern Implementation/ Illustration	12
4. Design	15
4.1 Program Design Goals	15
4.2 Static Modelling	16
4.3 Dynamic Modelling	18
5. GUI design change and User Interface	23
6. Testing	24
6.1 Tool used	24
6.2 Tested Sections	24
6.3 Test plans	25
7. Persistent Data and Access control	25
8. Version control	26
References	27

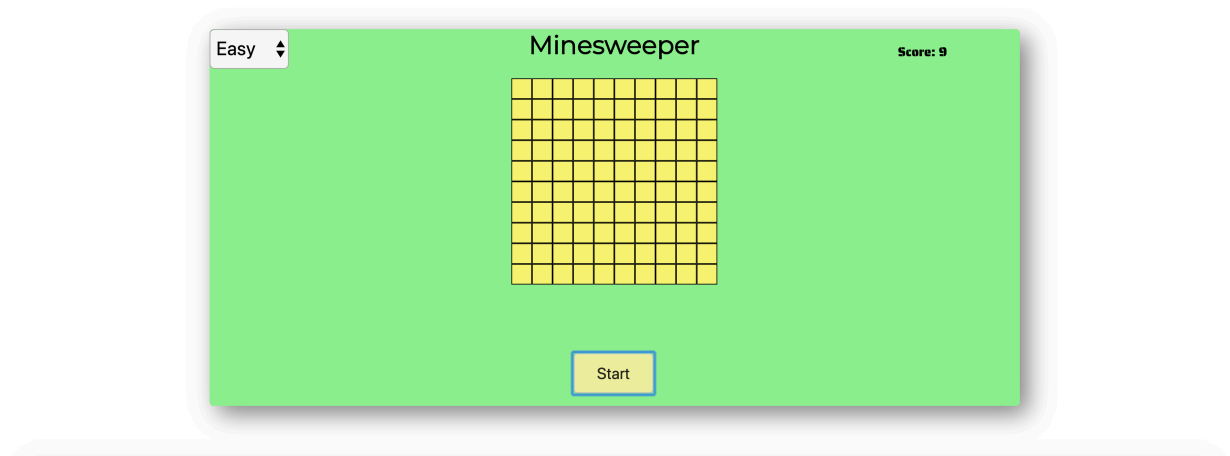
1. Requirements

1.1 Completed functional requirements

1) The home screen: The home screen is displayed as soon as the game is loaded. It holds the difficulty selector and start game button as shown in the figure below:

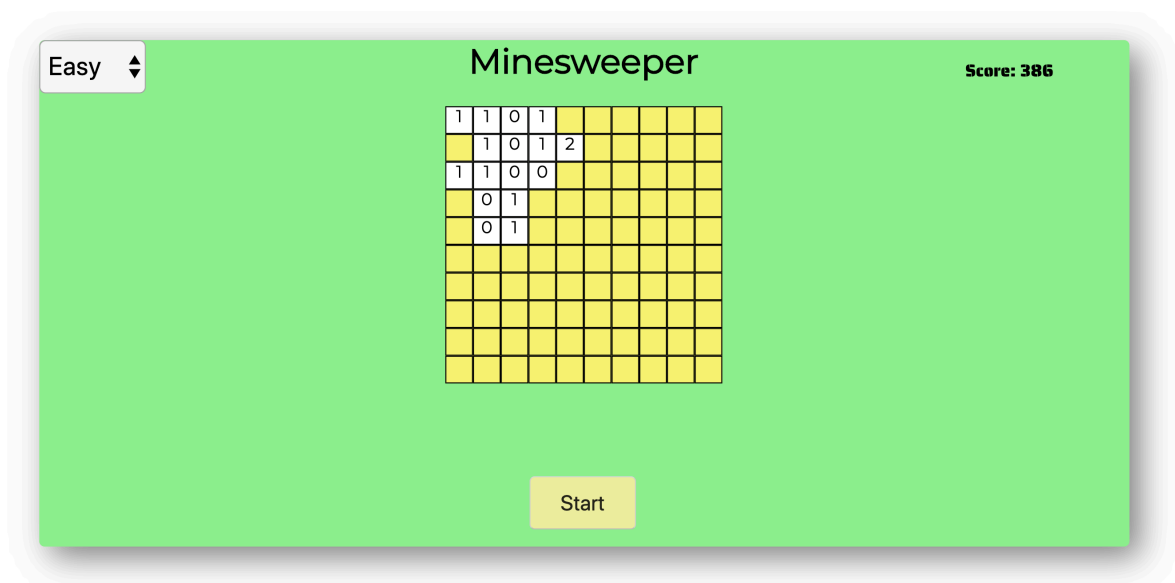


2) The start button: When the user presses the start button, the grid is displayed and the timer starts which is displayed in the top right of the game window as shown in the figure below:

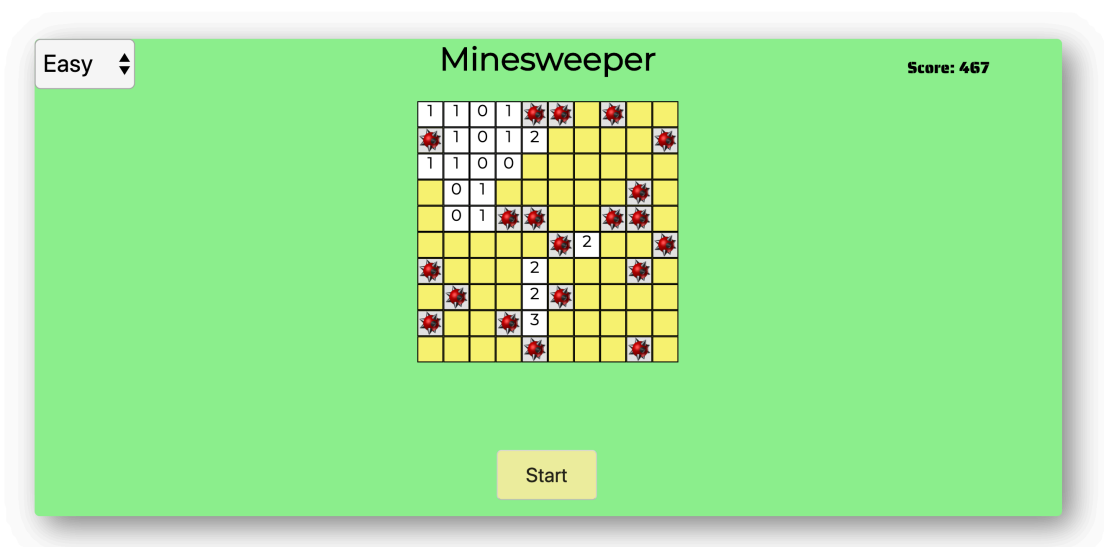
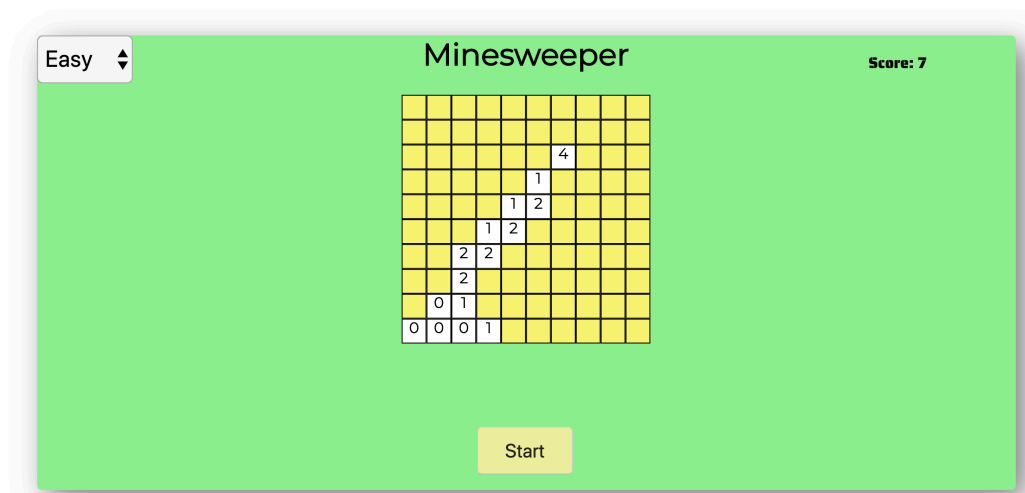
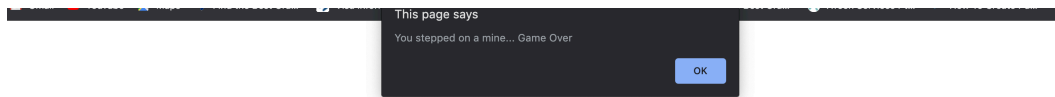


3) The grid: The requirements demand that the grid should be covered at the beginning of the game which is already accomplished as shown in the figure above.

4) The grid cells: The grid cells are required to be responsive to user interaction. When the user clicks on a cell using the left mouse button, the cell is revealed and a number is displayed within the cell which represents the number of mines in the neighbouring cells as shown in the figure below :



5) The mines: When the user clicks on a cell with a mine, the game alerts the user that a mine has been activated, displays all the mines in the grid, and stops the timer as shown in the figure :



1.2 Ongoing functional requirements priority list

This priority list is based on a scale of 1-5 where 5 is the maximum limit and 1 is the minimum.

ID	Requirement	Priority	Comment
1	User should be able to restart the game using the restart button	Level 5	It is a mandatory requirement for any game to have a restart button hence, it is a high-level priority.
2	User should be able to mark a cell which they suspect holds a mine.	Level 5	This is one of the key playing features defined under the concept of Minesweeper. Hence, it is a high-level priority.
3	User should be able to select a difficulty level and the grid changes according to the level selected	Level 4	The absence of this requirement does not prevent the flow of the game hence it is not Level 5 but its absence can make the game not interesting enough for users. Hence, it is a Level 4 priority and should be included in the final version of the game.
4	Audio effects	Level 2	The purpose of this requirement is to make the game more interesting and does not relate to any core functionality. Hence, it is a level 2 priority.

2. Use Cases

2.1) Difficulty Selection Use Case

Name	Select Difficulty
Status	Ongoing
Description:	Player selecting the difficulty at the beginning of the game or after restarting
Actors:	Player
Preconditions:	1.The game should have just started or the player should wish to replay the game. 2. The game should not be in session.
Postconditions:	1.Either “easy”, ”medium” or “hard” should be selected as the difficulty level.
Flow:	1.The player runs the game. 2.The difficulty selection menu is displayed on the screen with options of easy, medium and hard. 3. The user selects the desired difficulty. 4. The difficulty is set on the basis of the user’s selection. 5. The game grid is displayed on the basis of the difficulty level selected.
Alternative Flows:	1. In step 1 of the normal flow, if a game session has just ended, the user presses the restart button and resumes at step 2 of the normal flow.

2.2) Playing Game Use Case

Name	Play Game
Use Case Status	Completed
Description:	Player plays the game trying to win
Actors:	Player
Preconditions:	1.The difficulty level should be selected. 2. The game should not be already in session.
Postconditions:	1.Player either completes the game or fails to complete it.
Flow:	1.The player selects the difficulty of the game and presses the start button. 2. The grid is displayed on the basis of difficulty selected and the player starts making moves. 3. The player successfully avoids clicking any cells with mine and completes the game.
Alternative Flows:	1. In step 3 of the normal flow, if the user clicks on a mine cell then, the game session ends displaying a message notifying the user that the game session has just ended and also all the mines in the grid.

2.3) Start Timer Use Case

Name	Start Timer
Use Case Status	Completed
Description:	The timer is started as soon as a game session starts
Actors:	System
Preconditions:	1. A new game session has just started.
Postconditions:	1. The timer starts to show the number of seconds passed since the current game session began.
Flow:	1. The player starts the game. 2. The timer is started.

2.4) Update High Score Use Case

Name	Update High Score
Use Case Status	Ongoing
Description:	System updates a high score if the user beats the set high score
Actors:	System
Preconditions:	1.The game session must have just ended. 2. The new score of the user should be greater than the current set high score.
Postconditions:	1.The high score is updated as the new score from the user.
Flow:	1.The player completes the game successfully and scores a new high score. 2. The new high score is saved within the system and displayed on top of the game window. i
Alternative Flows:	1. In step 1 of the normal flow, if the user does not complete the game successfully or does not score higher than the previous high score, the high score is not updated.

3. Software Architecture

The scale of the application in this project is quite small. Hence, it may seem like there are no particular advantages of adopting a fixed software architecture pattern. However, choosing an appropriate pattern enables us to easily manage the source code and also extend the functionality and scope of the project in a later date.

Model-View-Controller (MVC) is one of the most popular software architecture patterns that suite web-based applications. Since this game application has been built using Javascript, MVC is more than qualified to handle its architecture.

Reasons for using Javascript/HTML/CSS

1. Originally(In milestone 1), Java was used to create the mockup(prototype) of the game. However, due to familiarity with HTML/CSS and Javascript along with its libraries, I changed to the latter approach.
2. MVC pattern has been adopted for this project. This pattern can easily be maintained by using HTML and CSS for views and Javascript functions and objects for model and controllers.

The main reason for the change was due to personal familiarity with web-based applications.

3.1 Tools used

Jquery - jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers (js.foundation, 2019). This library has been used to detect user interactions with the view and invoke appropriate functions to carry out the desired activities.

3.2 Feasibility to change

The feasibility of the entire language to change is minimum ,however, the structure of the model class may be changed by implementing the use of ‘class’ keyword which was introduced in ECMAScript 2015 (Javascript updates). The reason behind the possibility is to implement private properties for security purposes.

3.3 Pattern Implementation/ Illustration

Model

Model represents the shape of the data and business logic. It maintains the data of the application. The model provides a prototype for the major data structures of the application. The model for this project is shown in the figure below:

```
// Minesweeper game Object

function Minesweeper() {
  this.level;
  this.highscore;
  this.timer;

  this.changebackground = function(element){ changebackground(element)};
  this.revealcells = function(focalpoint){ revealcells(focalpoint)};
  this.setupmines = function(){ setupmines()};
  this.starttimer = function(){ starttimer()};
  this.drawgrid = function(){ drawgrid() };

}
```

In javascript, function keyword can be used to define object prototypes which provide the shape of the main data structure. Here, Minesweeper is the main data structure that represents the game itself. It contains its members/properties like level, timer and high score. It also contains functions that can be called from the controller after user interaction.

Controller

Controller handles the user request. Typically, the users interact with View and then the view transfers the information of user interaction to controller and controller updates the model according to the user request. It also changes the view according to user interaction. Few functions defined in the controller for this project can be seen below :

```
6
7 $(document).ready(function()
8 {
9     //When browser window opens new game instance is created
10     var game = new Minesweeper();
11
12     $('#board').css("animation-name","showboard");
13     setTimeout(() => {
14         $('#board *').css("display","block")
15     },1500);
16
17 // When start button is pressed the following functions are invoked and a game session is started
18 $("#start button").click(function()
19 {
20     if(!$("#time").text())
21     {
22         game.starttimer();
23         game.drawgrid();
24         game.setupmines();
25     }
26     else
27     {
28
29     }
30 }
31 );
32 });
33
34 //function to start timer
35 function starttimer()
36 {
37     var seconds = 1;
38     this.timer = setInterval(function(){
39         $('#time').text(`Score: ${seconds}`);
40         seconds++;
41     },1000);
42 }
43
44 function drawgrid()
45 {
46     var gridsetup = "";
47     for(var rows = 1; rows <= 100; rows++)
48     {
49         gridsetup += `<li class = 'cell' value = 0 onclick = 'changebackground(this)'></li>`;
50     }
51     $("#grid ul").html(gridsetup);
52 }
```

As seen in the figure above, the `$(document).ready(function){}`; runs as soon as the browser document is ready and loaded. First, a new instance of the game Minesweeper is created and then the application waits for the user to interact with the game by the means of start button which triggers the function belonging to the game instance that was created before which causes the grid/playboard to appear and the user can interact with the board.

View

View is a user interface. View displays data using the model to the user and also enables them to modify the data. ("MVC Architecture", 2019). Whenever the user interacts with the view, it informs the controller about the interaction and the controller handles the further steps. HTML and CSS were used to create the view for this application. The code for the view can be seen below:

```
<!DOCTYPE html>
<html>
<head>
  <title>Minesweeper</title>
  <link href="https://fonts.googleapis.com/css?family=Saira+Stencil+One&display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/css?family=Montserrat&display=swap" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="game.css">
</head>
<body>
  <div id = "board">
    <div id = "levelselector">
      <select>
        <option>Easy</option>
        <option>Medium</option>
        <option>Hard</option>
      </select>
    </div>
    <div id = "heading">
      <h1>Minesweeper</h1>
    </div>
    <div id = "timer">
      <p id = "time"></p>
    </div>
    <div id = "grid">
      <ul>
      </ul>
    </div>
    <div id = "start">
      <button>Start</button>
    </div>
  </div>

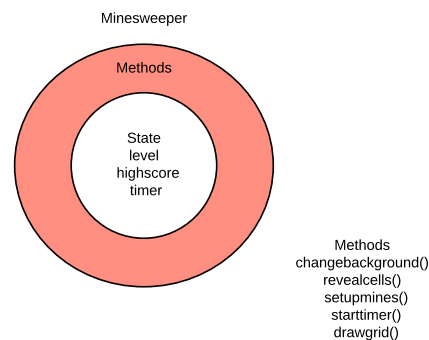
  <audio id="myAudio">
    <source src="Bomb.mp3" type="audio/mpeg">
    Your browser does not support the audio element.
  </audio>

  <script
src="https://code.jquery.com/jquery-3.4.1.min.js"
integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFLBw8HfCJo="
crossorigin="anonymous"></script>
```

4. Design

4.1 Program Design Goals

The program is designed based on the concept of object-oriented design. Since it is a small-scaled application, there is only one object that is the game. The abstraction for the game is called Minesweeper. When the browser runs, an instance of this abstraction is created which acts as the game object. The player interacts with the game through the view provided and the view informs the controller about the user interaction. The controller then interacts with the game through its properties and updates the view as well. The following diagram represents the game object.



Some methods are still to be implemented which are mentioned in the on-going functional requirements list.

4.2 Static Modelling

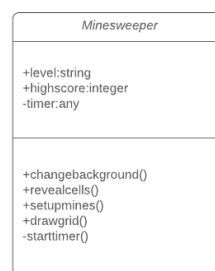
Class Diagram

Since there is only one class in this application. The entire game experience is formed by the interaction with this game object. There is no interaction between any two objects. The game class contains properties like level, high score and timer which define the state of the game.

The class diagram for the minesweeper class is given below:

Class Diagram with UML Notation

sankit shrestha | September 21, 2019



The implementation of this class in code is shown in the figure below:

```
// Minesweeper game Object

function Minesweeper() {
  this.level;
  this.highscore;
  this.timer;

  this.changebackground = function(element){ changebackground(element)};
  this.revealcells = function(focalpoint){ revealcells(focalpoint)};
  this.setupmines = function(){ setupmines()};
  this.starttimer = function(){ starttimer()};
  this.drawgrid = function(){ drawgrid() };
}
```

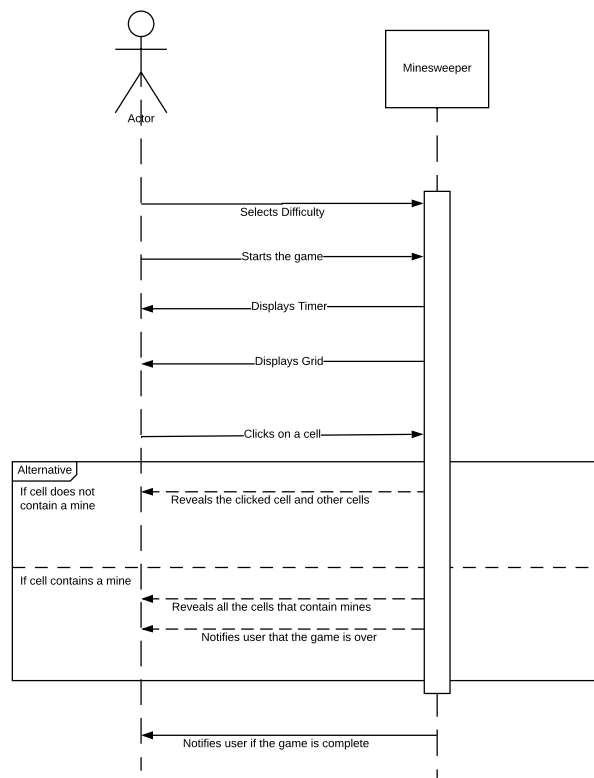

The methods defined within the game class change the state properties of the game. For example, the start timer function starts the timer property and reveal cells function later handles the stopping of the timer.




```
//function to start timer
function starttimer()
{
    var seconds = 1;
    this.timer = setInterval(function(){
        $('#time').text(`Score: ${seconds}`);
        seconds++;
    },1000);
}
```

4.3 Dynamic Modelling

Sequence Diagram

Since there is only one object/class, the sequence diagram shows the interaction between the actor/player and the game (Minesweeper). The sequence diagram is given below:

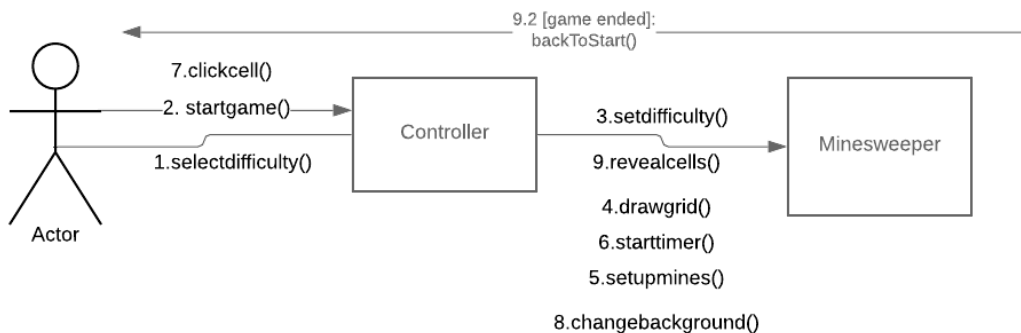


The messages denoted by  in the above diagram are self-generated whereas, the messages denoted by  are the reply messages. The messages sent by the user are read by the controller  which triggers the appropriate method belonging to the game. The detailed information about each message transaction in a numbered sequence along with code implementation and collaboration diagram is provided in the section below:

4.4 Collaboration diagram

UML Communication Diagram

sankit shrestha | September 23, 2019



1. Sets the difficulty - User selects the difficulty level which triggers the set difficulty method of the object and changes the level property of the game instance. The set difficulty method is yet to be implemented.
2. Start the game - The user clicks the start button which triggers all the methods necessary to set the game session and start it.

```
6
7 $(document).ready(function()
8 {
9     //When browser window opens new game instance is created
10     var game = new Minesweeper();
11
12     $('#board').css("animation-name","showboard");
13     setTimeout(() => {
14         $('#board *').css("display","block")
15     },1500);
16
17 // When start button is pressed the following functions are invoked and a game session is started
18 $("#start button").click(function()
19 {
20     if(!$("#time").text())
21     {
22         game.starttimer();
23         game.drawgrid();
24         game.setupmines();
25     }
26     else
27     {
28     }
29 }
30 }
31 );
32 }
33 };
```

MINESWEEPER

3. Start timer and display it: The `starttimer()` method starts the timer for the current game session and displays it in the top left corner of the game screen.

```
//function to start timer
function starttimer()
{
    var seconds = 1;
    this.timer = setInterval(function(){
        $('#time').text(`Score: ${seconds}`);
        seconds++;
    },1000);
}
```

4. Draw grid and display grid: The `drawgrid()` function changes the grid element of the view and sets up the required wiring between elements and the game object's methods.

After drawing the grid, the `setupmines()` method is called which sets up the mines inside the grid.

```
function drawgrid()
{
    var gridsetup = "";
    for(var rows = 1; rows <= 100; rows++)
    {
        gridsetup += `<li class = 'cell' value = 0 onclick = 'changebackground(this)'></li>`;
    }
    $("#grid ul").html(gridsetup);

    $("#grid").css({"position":"relative","width":"260px"});
}
```

```
//function for setting up the mines
function setupmines()
{
    var cells = document.getElementsByClassName('cell');
    var mineindex = [];
    var firstcolumn = [0,10,20,30,40,50,60,70,80,90];
    var lastcolumn = [9,19,29,39,49,59,69,79,89,99];
    while(true)
    {
        var mine = Math.floor(Math.random() * 100);
        if(!mineindex.includes(mine))
        {
            mineindex.push(mine);
        }
        if(mineindex.length >= 20)
        {
            break;
        }
    }

    mineindex.forEach((position)=>
    {
        cells[position].setAttribute("class","cell mine");
        try
        {
            if(!firstcolumn.includes(position) )
            {
                var minenumber = Number(cells[position - 11].getAttribute("value"));
                cells[position - 11].setAttribute("value",minenumber + 1);
            }
        }
        catch
        {
        }
        try
        {
            var minenumber = Number(cells[position - 10].getAttribute("value"));
            cells[position - 10].setAttribute("value",minenumber + 1);
        }
        catch
        {
        }
        try
        {
            if(!lastcolumn.includes(position) )
            {
                var minenumber = Number(cells[position - 9].getAttribute("value"));
                cells[position - 9].setAttribute("value",minenumber + 1);
            }
        }
        catch
        {
        }
        try
        {
            if(!firstcolumn.includes(position) )
            {
                var minenumber = Number(cells[position - 1].getAttribute("value"));
                cells[position - 1].setAttribute("value",minenumber + 1);
            }
        }
        catch
        {
        }
        try
        {
            if(!lastcolumn.includes(position) )
            {
                var minenumber = Number(cells[position + 1].getAttribute("value"));
            }
        }
    }
}
```

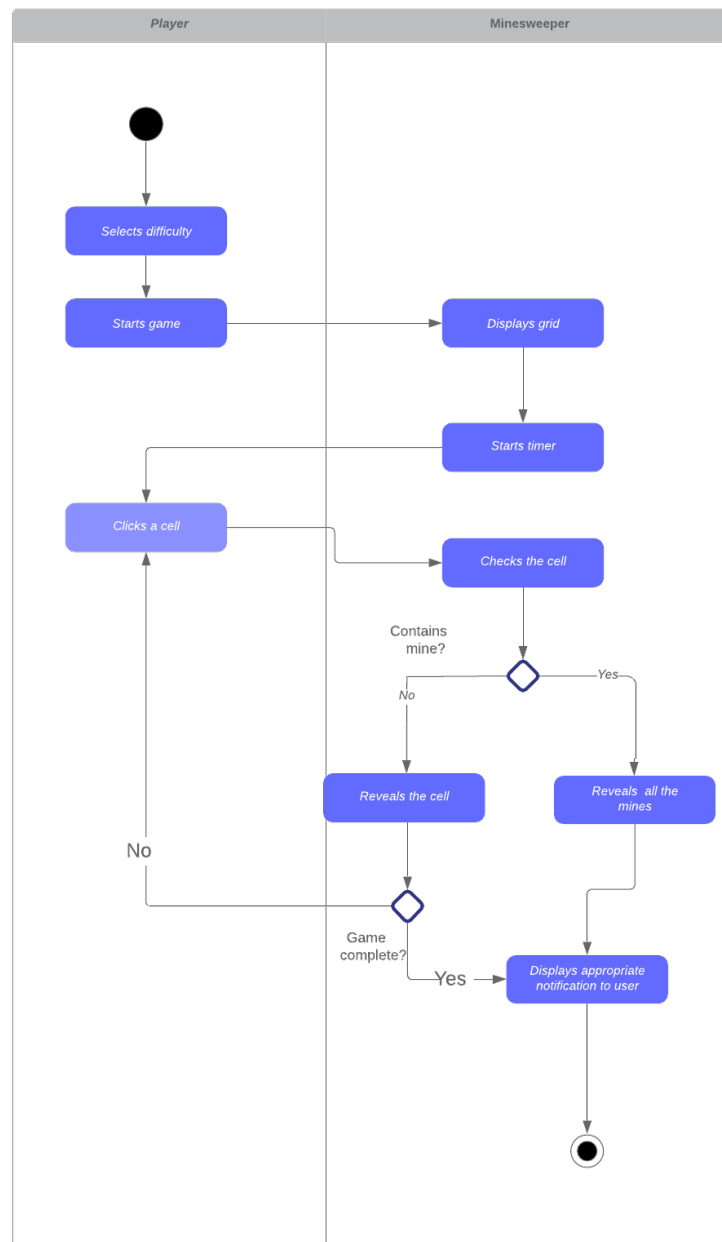
5. Reveal cells - The `changebackground()` method is called whenever a user clicks a cell. This method checks if the cell contains a mine. If the cell contains the mine, it stops the timer, displays all the mines and notifies user that the game is over but if the cell does not contain a mine, it calls the `revealcells()` function which reveals the underlying content of the clicked cell and possibly neighbouring cells with the help of an algorithm.

```
// Function to reveal the clicked cell and neighbouring cells
function revealcells(focalpoint)
{
    var originalFocalPoint = focalpoint;
    var revealnumber = Math.floor(Math.random() * 10) + 1;
    var cells = Array.from(document.getElementsByClassName("cell"));
    var directions = ["northwest","north","northeast","east","southeast","south","southwest","west"];
    var direction = directions[Math.floor(Math.random() * 8)];
    var firstcolumn = [0,10,20,30,40,50,60,70,80,90];
    var lastcolumn = [9,19,29,39,49,59,69,79,89,99];

    for(var x = 1; x < revealnumber; x++)
    {
        try
        {
            if($cells[focalpoint].attr("class") !== "cell mine")
            {
                if(direction === "northwest" && !firstcolumn.includes(focalpoint))
                {
                    var minenumber = $(cells[focalpoint]).val();
                    $(cells[focalpoint]).css("background-color","white");
                    $(cells[focalpoint]).html('<p>${minenumber}</p>');
                    focalpoint -= 11;
                }
                else if (direction === "north")
                {
                    var minenumber = $(cells[focalpoint]).val();
                    $(cells[focalpoint]).css("background-color","white");
                    $(cells[focalpoint]).html('<p>${minenumber}</p>');
                    focalpoint -= 10;
                }
                else if (direction === "northeast" && !lastcolumn.includes(focalpoint))
                {
                    var minenumber = $(cells[focalpoint]).val();
                    $(cells[focalpoint]).css("background-color","white");
                    $(cells[focalpoint]).html('<p>${minenumber}</p>');
                    focalpoint -= 9;
                }
                else if (direction === "east" && !lastcolumn.includes(focalpoint))
                {
                    var minenumber = $(cells[focalpoint]).val();
                    $(cells[focalpoint]).css("background-color","white");
                    $(cells[focalpoint]).html('<p>${minenumber}</p>');
                    focalpoint += 1;
                }
                else if (direction === "southeast" && !lastcolumn.includes(focalpoint))
                {
                    var minenumber = $(cells[focalpoint]).val();
                    $(cells[focalpoint]).css("background-color","white");
                }
            }
        }
    }
}
```

Activity Diagram

The following diagram represents the activity diagram for this application. The diagram does not include the functionalities/actions that have not been implemented in the current version of the game.



5. GUI design change and User Interface

The GUI of the application/game has changed drastically since milestone 1 due to the fact that the entire codebase was transferred from Java to Javascript. The design aspect of the game has been modified completely. The changes in the design of the GUI are discussed below:

1. Changes in colour palette- The GUI consisted of mainly white colour in the beginning. However, it has been changed to the following colours:

- i. First primary colour  - light green

- ii. Second primary colour  - yellow

The light green colour has been used for the entire game window which houses the user interface.

The yellow colour has been used for the grid that the player interacts with.

2. Changes in icons used - The icons used in the game have gone through the following changes:

- i. Mine icon- Previous design stated that if the user clicks on a mine, the following icon would be displayed : 

However, this has been changed to the following icon in the updated

design: 

- ii. Cell icon - Previous design stated that the following icon would represent a cell :



However, this has been changed to the following icon in the updated

design: 

6. Testing

6.1 Tool used

A library of Javascript called “Jasmine” was used to conduct testing activities for the game. Jasmine is a behaviour-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests ("introduction.js", 2019).

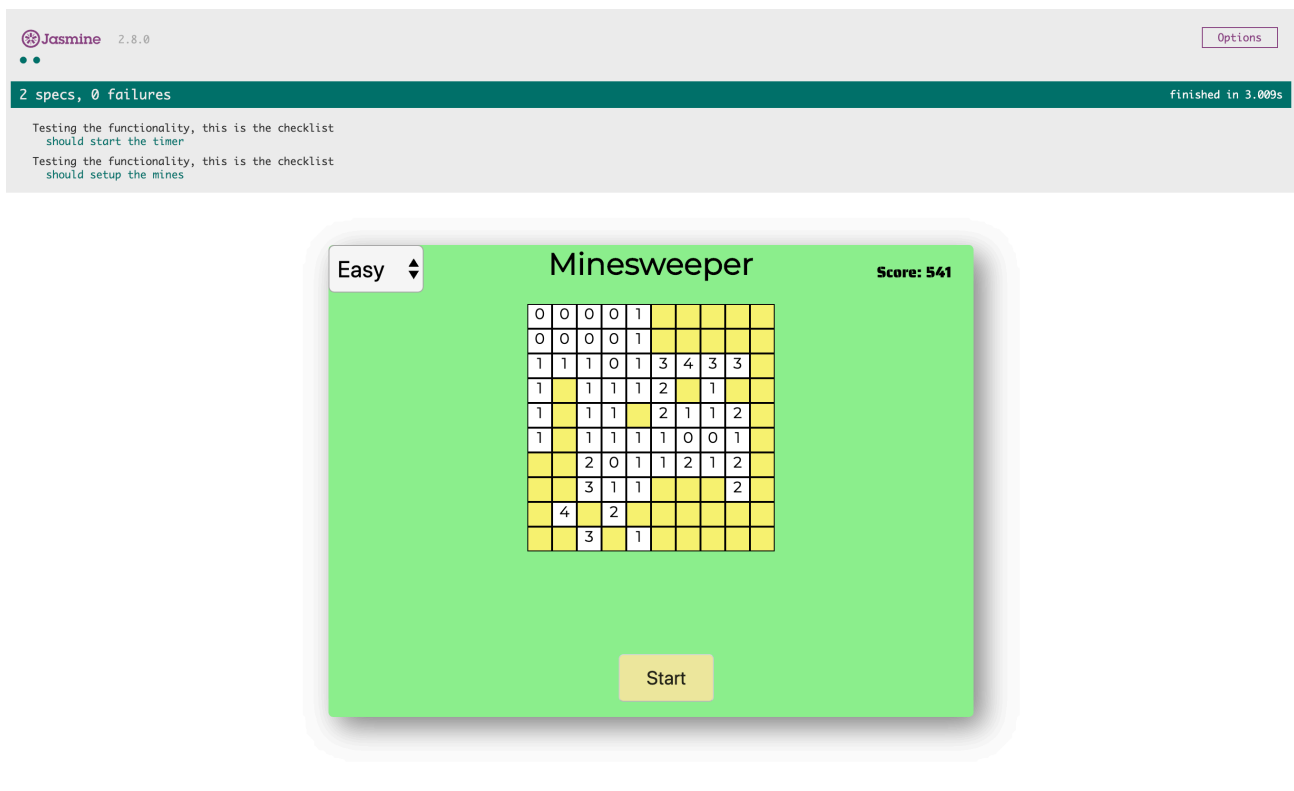
6.2 Tested Sections

All the functions in the working version of the game have been tested. Some of the unit tests are given below :

```
describe('Testing the functionality, this is the checklist', ()=>{
  it('should start the timer', async ()=>{
    let game = new Minesweeper();
    game.starttimer();
    let promise = new Promise((resolve,reject)=>
    {
      setTimeout(()=> resolve($('#time').text()),3000);
    });
    let result = await promise;
    expect(result).toEqual('script.js:303 ');
    clearInterval(game.timer);
  })
});

describe('Testing the functionality, this is the checklist', ()=>{
  it('should setup the mines', ()=>{
    let game = new Minesweeper();
    game.drawgrid();
    game.setupmines();

    let minecells = document.getElementsByClassName('cell mine');
    expect(minecells.length).toBe(20);
  })
});
```

As seen in the above figure, the two functions namely `starttimer()` and `setupmines()` have been tested through the jasmine framework and have passed the test.

6.3 Test plans

Every functionality that will be added to the current version will be unit tested to ensure that they work independent of other methods and later an integration test will be conducted to assess the collaborative work of all the components.

7. Persistent Data and Access control

The functional requirements of this game do not include support for multiple users. No sensitive information of the players is recorded and stored which reduces the need for strict access control and security. Hence, the level of access control is minimum.

The local storage of the window i.e the browser will be used to store the high score of the game since we only have to deal with a tiny amount of data and no sensitive information. Implementation of database storage or any other technology is not necessary.

The high score can be set and retrieved by using the following syntax:

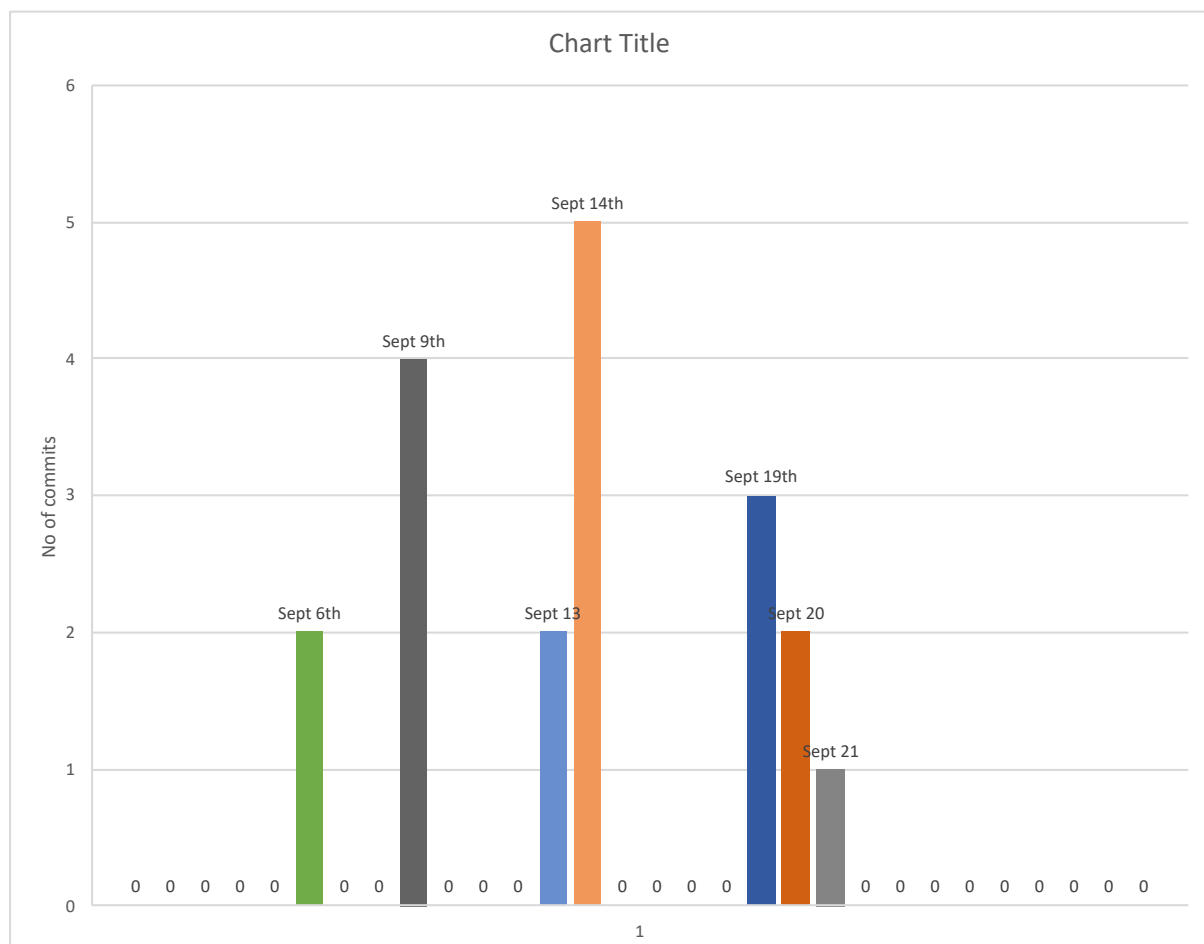
```
// Store
localStorage.setItem("highscore", 299);
// Retrieve
document.getElementById("highscore").innerHTML
= localStorage.getItem("highscore");
```

8. Version control

GitHub was used for the version control purposes of this application.

Since the scope of the application is quite small and the entire project is to be completed individually, no additional branches were created. Instead, all the changes during the development were made to the local repository and after testing the functionality, the changes were uploaded to the master branch in GitHub.

A graph showing the number of commits made to the git repository in September is given below:



References

introduction.js. (2019). Retrieved 22 September 2019, from <https://jasmine.github.io/2.0/introduction.html>

MVC Architecture. (2019). Retrieved 20 September 2019, from <https://www.tutorialsteacher.com/mvc/mvc-architecture>

js.foundation, J. (2019). jQuery. Retrieved 20 September 2019, from <https://jquery.com/>