Minesweeper Cricket

Sanoj S Vijendra

June 2023

Contents

1 Introduction

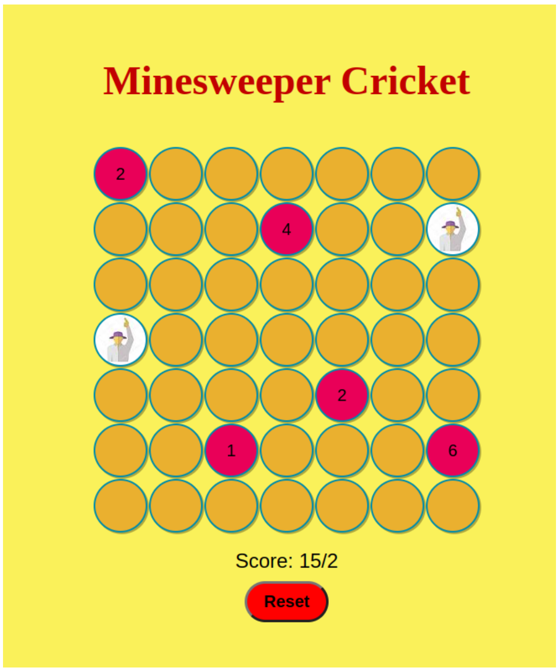
Introducing "Minesweeper Cricket," a captivating fusion of the classic Minesweeper game and the excitement of cricket. Uncover hidden runs on a grid while strategically avoiding fielders to score as many runs as possible before running out of wickets. With varying grid sizes, live score display, reveal options, and a leader-board, this game offers an addictive blend of strategic thinking and cricket thrills for players of all ages. Immerse yourself in the world of "Minesweeper Cricket" and aim for the highest score in this innovative twist on a beloved classic.

2 Customisation

2.1 Feel of Cricket

This game evokes the excitement and thrill of cricket by incorporating elements from the sport into its gameplay.

- As players uncover hidden runs on the grid, they experience the same anticipation and strategic decision-making that cricketers face on the field.
- There are hidden fielders present on the playing field randomly, if user clicks on them, he gets out just same as in actual cricket (when you attempt to smash a ball and find a fielder you get out).
- The game's immersive environment, coupled with live score updates and leaderboard rankings, creates a sense of competition and engagement akin to a real cricket match.



(a) Game Board

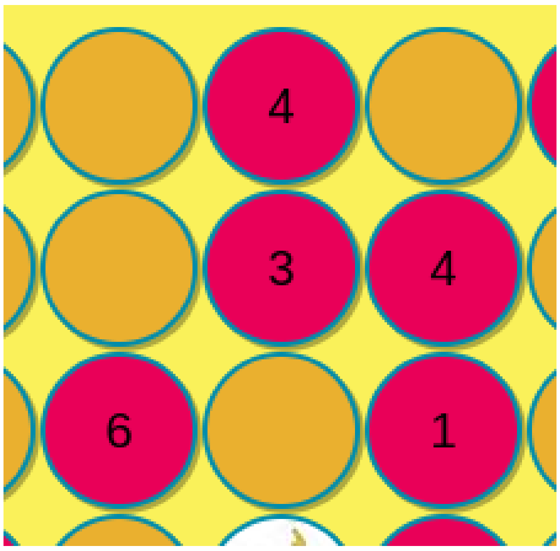
2.2 Grid-Size

This game allow user to select the appropriate grid size of his choice among 5x5, 6x6, 7x7, 8x8 and 9x9. [1]

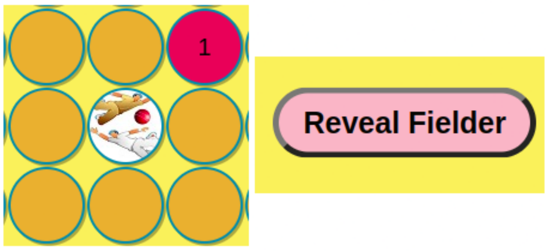
1. `[5 x 5]`: Number of fielders present on the field are 5 and user has 3 wickets.
2. `[6 x 6]`: Number of fielders present on the field are 6 and user has 4 wickets.
3. `[7 x 7]`: Number of fielders present on the field are 9 and user has 5 wickets.
4. `[8 x 8]`: Number of fielders present on the field are 11 and user has 5 wickets.
5. `[9 x 9]`: Number of fielders present on the field are 11 and user has 5 wickets.

2.3 Power Ups

- **Multiple Runs:** User can score more than singles by clicking on the grids. Runs present on the board are generated randomly. Runs which can be scored are 1,2,3,4 and 6. Probability of hitting sixes is around **1/3 !**.
- **Revealing Fielders:** If a players makes four successful moves without getting out then he is rewarded with a power up which will reveal the location of a fielder placed on the board randomly for 1 or 2 seconds. But there is also a catch with this. So, be wise to while using this feature.



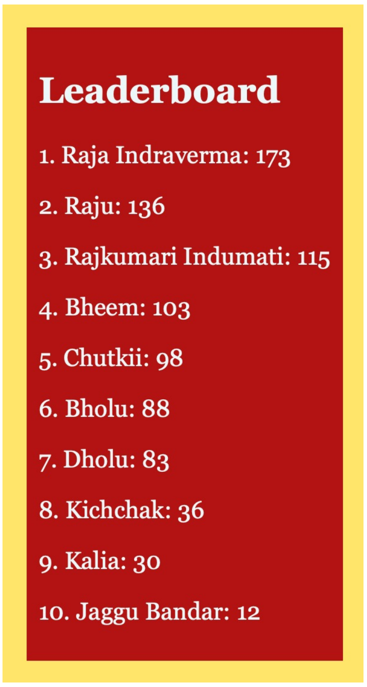
(a) Runs



(b) Fielders

2.4 Leaderboard

The leaderboard in adds an extra layer of competitiveness and motivation for players. It showcases the top performers, allowing them to compare their scores and achievements with other players. By achieving higher scores and completing levels with efficiency, players can climb up the leaderboard. With real-time updates and a ranking system, the leaderboard in the game adds a dynamic and interactive element to the game.



Leaderboard

1. Raja Indraverma: 173
2. Raju: 136
3. Rajkumari Indumati: 115
4. Bheem: 103
5. Chutkii: 98
6. Bholu: 88
7. Dholu: 83
8. Kichchak: 36
9. Kalia: 30
10. Jaggu Bandar: 12

3 CSS

In this section, I will explain the CSS I have used. [2]

3.1 Colors

The page layout is extensively designed for visual appeal using CSS. A combination of shades of red, orange and yellow is used to bring about contrast and make the color pallet look beautiful. Apart from this different hover colors are added on buttons.

3.2 Buttons

1. **Grid Size Button:** Initially button has rounded edge and have background color of pink. When user hover over the button, it will undergo a 360° rotation about X-axis and its color will change to blue in about 0.7 seconds to give a smooth transition. Also, inside text color will change to white. Additionally, a blurry shadow will be produced.

2. **Instruction/Hide:** Initially button has background color orange but when user hover over it, its color will change to green and its size will be scaled 1.1 times of original in about 1 second to give a smooth transition. Color of inside text will change from black to white. Also, "instructions" written on it changes to "hide" and vice-versa. Additionally, a blurry shadow will be produced.

3. **Start/Reset:** Initially button has background color of red but when user hover over it, its color will change to blue and its size will be scaled 1.1 times of original in about 1 second to give a smooth transition. Color of inside text will change from black to white. Additionally, a blurry shadow will be produced.

4. **Reveal Fielder:** This button when appears, start to blink with a time period of 1 second. Initially, it has background color of pink. On hovering over it, blinking stops and backgound color changes to blue. Color of text changes from black to white. This all takes about 1 second time which will give it a smooth transition. Additionally, a blurry shadow will be produced.

3.3 Grid

The shape of the grid tiles is changed from the traditional square to circle by changing the border percentage. Also, each of the grid has a shadow following it. Whenever a tile is clicked, it undergoes a 360° rotation about the Y-axis. If clicked tile has no fielder on it, the tile will become pinkish (intially orange) and run which is scored will be displayed on it. Else if, there is a fielder, the tile will show a picture of umpire showing the out signal.

3.4 Reveal Fielder

When ”Reveal Fielder” button is clicked then randomly a tile is selected which have hidden fielder and it will change into a figure having fielder inside it for 1 second then return to original color.

3.5 Instruction Box

It is shown when instruction button is clicked and hides when hide button is clicked. When user hover over the instruction button it will scale to 1.1 times of its size in about 1 second to give a soomth transition effect. Additionally, a blurry shadow of the box will be produced.

3.6 Leaderboard Table

It is shown at the end of the game. It has background color brown and text color white. On hovering over it, it gets scaled to its 1.1 times of its original size in about 1 second to give a smooth transition effect. Additionally, a blurry shadow of the box will be produced.

3.7 Miscellaneous

The game tile has red-orange color and score section had black color. Font of all texts are Times New Roman.

4 JavaScript

in this section JS code of the game will be explained, especially different functions used. [3]

- **toggleInstructions():** This function is responsible for toggling the visibility of an *instructions* div and updating the text from ”Instructions” to ”Hide” of the corresponding button.
- **setGridSize(size):** This function is responsible for setting the grid size and adjusting the game elements accordingly. The function takes a parameter ’size’ which represents the desired grid size. This function also sets the number wickets corresponding to the grid size.
- **createGrid():** This function is responsible for dynamically creating a grid of blocks based on a selected grid size.
- **generateFielders():** This function is responsible for randomly generating the locations of fielders on the game-board according to the grid size.
- **startGame():** This function is called when start(”Are You Ready!”) button is clicked. This will initiate the game and set up the initial game state. Additionally, it calls *createGrid()*, *generateFielders()* and *loadLeaderboard()* functions within itself.
- **blockClick(event):** This function initially checks if /textitgameOver is true or not. If not, it will increment *successfulMoves* variable and then checks if it is divisible by 5 or not. If it is divisible, then it will activate ”Reveal Fielder” button until user does not click it or he loses a wicket. It then checks, if the clicked tile contains fielder or not. If it has fielder,

then it checks the number of wicket fell and calls *wicketfall()* or *displayGameOver()* and *updateLeaderboard()* functions accordingly. This function also assign random runs to the tiles based on assigned probability. It also updates the total score of the user. This function is also responsible for assigning rotating animation to the tile when it is clicked.

- **revealFielder():** This function is called when "Reveal Fielder" button is clicked. It reveals a randomly chosen fielder on the grid for a short duration.
- **wicketfall():** This function is called when a tile containing fielder is clicked, indicating that a wicket has fallen. It updates the *wickets, lives* and *successfulMoves* variable accordingly and hides the "Reveal Button". It then displays an alert box showing that wicket has fallen.
- **resetGame():** This function is called when the player clicks "Reset Button". This function resets the game and takes user to home screen.
- **displayGameOver():** This function is called when user loses all its wickets. It displays an alert box showing game over and showing final score of user. Then it calls the function *promptPlayerName()* and *displayLeaderboard().*
- **loadLeaderboard():** This function is responsible for loading the leaderboard data from the browser's local storage and updating the leaderboard display by calling *updateLeaderboard()* function.
- **addToLeaderboard():** This function is responsible for adding the current player's data to the leaderboard and storing it in the browser's local storage. It also checks the number of entries already in leaderboard. If it is greater than 10 then it compares new added score and sort the whole leaderboard in descending order. Then it truncates its number of entries to 10.
- **displayLeaderboard():** This function is responsible for rendering the leaderboard on the webpage.

5 Source Code

5.1 HTML

```html
<!DOCTYPE html>
<html>
<head>
    <title>Minesweeper Cricket</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <!-- a division which will link game with cricbuzz website and wikipedia of minesweeper. -->
    <div id="float">
        <a href="https://www.cricbuzz.com/">Live Cricket Feeds</a>
        <a href="https://en.wikipedia.org/wiki/Minesweeper_(video_game)">Official Minesweeper Info</a>
    </div>
    <h1>Minesweeper Cricket</h1>
    <div id="gridSizeOptions">
        <!-- buttons for selecting grid size. -->
        <button onclick="setGridSize(5)">5x5</button>
        <button onclick="setGridSize(6)">6x6</button>
        <button onclick="setGridSize(7)">7x7</button>
        <button onclick="setGridSize(8)">8x8</button>
        <button onclick="setGridSize(9)">9x9</button>
    </div>
    <div id="grid" style="display: none;"></div>
    <!-- this will show score of the player. -->
    <div id="score" style="display: none;">Score: <span id="scoreValue">0/0</span></div>
    <!-- start and Reset buttons. -->
    <button id="startButton" onclick="startGame()" style="display: none;">Are You Ready!</button>
    <button id="resetButton" onclick="resetGame()" style="display: none;">Reset</button>
    <!-- instruction button. -->
    <button id="instructionsButton" onclick="toggleInstructions()">Instructions</button>
    <div id="instructions" style="display: none;">
        <!-- set of Instructions. -->
        <h2 id="instructionTitle">Instructions</h2>
        <p>1.  Welcome! to this remix of minesweeper and cricket.<br>
        2.  Basic Rules are same as that of classic minesweeper.<br>
        3.  But, as cricket is mixed, runs are distrubuted randomly on board.<br>
        4.  User can score 1,2,3,4 or 6 runs in each turn.<br>
        5.  Beware! of fielders, they have occupied their places blindy on the field.<br>
        6.  Numbers of fielders on the board depends on the size of grid-board, so are the number of wickets.<br>
        7.  Score as many runs as you can before you run out of wickets and indulge yourself in happiness when you see your name on the leaderboard.<br>
        8.  Enjoy! the Game.</p>
    </div>
    <!-- reveal Button. -->
    <button id="revealButton" onclick="revealFielder()" style="display: none;">Reveal Fielder</button>
    <!-- this will generate leaderboard. -->
    <div>
      <ul class="leaderboard" id="leaderboard"></ul>
    </div>
    <!-- java script file "script.js" is linked. -->
    <script src="script.js"></script>
</body>
</html>
```

5.2 CSS

```css
body {
  display: flex; /* generating flex container */
  flex-direction: column; /* specifies main direction of flex axis as column */
  align-items: center; /* aligns items in flex container to the centre */
  justify-content: center;
  height: 100vh; /*set body element to 100% of viewport height */
  margin: 0; /* removes default margin by the browser*/
  font-family: Arial, sans-serif; /* specifies font-family of text within "body' */
  background-color: #FFE569; /* specifies background colour of webpage */
  position: relative; /* add relative positioning to the body */
}

h1 {
  margin-top: 20px; /*specifies top margin used by h1 tag */
  text-align: center;
  color: #B70404;
  font-size: 48px;
  font-family: 'Times New Roman', Times, serif;
  font-weight: bolder; /* makes text inside h1 tag bolder */
}

#gridSizeOptions {
  display: flex;
  flex-wrap: wrap; /* if child element exceeds the length of container, it will
    wrap into next line */
  justify-content: center;
  margin-top: 20px;
}

#gridSizeOptions button {
  margin: 5px; /* specifies margin of given for button inside the division
    gridSizeOptions */
}

#grid {
  display: grid; /*specifies that the element with the id "grid" should be a grid
    container. This means that its child elements will be laid out using the grid
    layout model */
  grid-template-columns: repeat(var(--grid-size), 1fr); /* The
    grid-template-columns property is used to define the number and width of
    columns in the grid. In this case, it uses the repeat() function with the value
     var(--grid-size) as the number of columns. The --grid-size is a CSS custom
    property that should be defined elsewhere and determines the number of columns
    in the grid. Each column is set to occupy an equal amount of available space
    using the 1fr unit. */
  grid-gap: 2px; /* The grid-template-columns property is used to define the number
     and width of columns in the grid. In this case, it uses the repeat() function
    with the value var(--grid-size) as the number of columns. The --grid-size is a
    CSS custom property that should be defined elsewhere and determines the number
    of columns in the grid. Each column is set to occupy an equal amount of
    available space using the 1fr unit. */
  margin-top: 20px; /*The margin-top property specifies the amount of space to be
    added above the element. In this case, a margin of 20 pixels is added above the
     "grid" element, creating a gap between it and the elements above it. */
}

.block {
  /* These properties set the width and height of the .block element to 60 pixels,
    making it a square shape. */
  width: 60px;
  height: 60px;
  /*This property sets the background color of the .block element to a specific hex
     color code (#E8AA42), giving it a yellowish-orange color.*/
  background-color: #E8AA42;
  border: 2px solid #068DA9; /*he border property sets the border style of the .block element. In this case, it creates a solid border with a width of 2 pixels and a color of #068DA9, which is a shade of blue.*/
  border-radius: 50%; /*This property sets the border radius of the .block element to 50%, resulting in a circular shape. The value of 50% means that the border will be curved equally on all four corners, creating a circle.*/
  display: flex;/*The display property with the value flex is used to establish a flex container. This allows the .block element to use flexbox layout properties.*/
  /*These properties are used to align the content inside the flex container both vertically and horizontally. align-items: center; centers the items vertically, and justify-content: center; centers the items horizontally.*/
  align-items: center;
  justify-content: center;
  font-size: 20px; /*This property sets the font size of the text inside the .block element to 20 pixels.*/
  cursor: pointer; /*This property sets the mouse cursor to change to a pointer when hovering over the .block element. It indicates that the element is clickable or interactive.*/
  transition: transform 0.5s ease; /*The transition property is used to specify the transition effect for the .block element. In this case, it applies a smooth transition to the transform property over a duration of 0.5 seconds, using an ease timing function. This allows for a smooth animation or transformation when the element changes.*/
  box-shadow: 1.8px 1.8px 0px rgba(0, 0, 0, 0.3); /*This property adds a box shadow effect to the .block element. It creates a subtle shadow with a horizontal offset of 1.8 pixels, a vertical offset of 1.8 pixels, a blur radius of 0 pixels, and a color defined by the RGBA values (0, 0, 0, 0.3).*/
}

#score {
  margin-top: 20px; /*This property sets the top margin of the #score element to 20 pixels. It creates a vertical space between the element and the element above it.*/
  font-size: 24px; /*This property sets the font size of the text inside the #score element to 24 pixels. It determines the size of the displayed text.*/
}

button {
  margin-top: 20px; /*This property sets the top margin of the button element to 20 pixels. It creates a vertical space between the button and the element above it.*/
  padding: 10px 20px; /*This property sets the padding of the button element. It adds 10 pixels of padding on the top and bottom, and 20 pixels of padding on the left and right. Padding creates space between the content inside the button and its border.*/
  font-size: 40px; /*This property sets the font size of the text inside the button to 40 pixels. It determines the size of the displayed text.*/
  font-weight: bolder; /*This property sets the font weight of the text inside the button to "bolder", which makes the text appear thicker and bolder than the default font weight.*/
  width: 400px; /*this property sets the width of the button element to 400 pixels. It determines the horizontal size of the button.*/
  height:150px; /*This property sets the height of the button element to 150 pixels. It determines the vertical size of the button.*/
  border-radius: 100px; /*This property sets the border radius of the button element to 100 pixels. It rounds the corners of the button, creating a circular shape due to the equal width and height values.*/
  border-width: 5px; /*This property sets the width of the border around the button element to 5 pixels. It determines the thickness of the button's border.*/
  background-color: #F3BCC8; /*This property sets the background color of the button to a specific color represented by the hexadecimal value #F3BCC8. It determines the color that fills the button's area.*/
  transition: background-color 1s ease, transform 0.7s ease, box-shadow 0.85s ease; /*This property specifies the transitions that should be applied to the button when certain properties change. In this case, it defines transitions for the
```

```
    background color, transform, and box shadow with different durations and easing
     functions.*/
}

button:hover {
  background-color: #30A2FF; /*This property changes the background color of the
    button when it is being hovered over by the cursor. It sets the background
    color to a specific color represented by the hexadecimal value #30A2FF.*/
  color: white; /*This property changes the text color of the button when it is
    being hovered over by the cursor. It sets the text color to white, making the
    text visible against the new background color.*/
  transform: rotateX(360deg); /*This property applies a 3D transformation to the
    button when it is being hovered over by the cursor. It rotates the button
    around the X-axis by 360 degrees, creating a spinning effect.*/
  box-shadow: 7px 7px 3px rgba(0, 0, 0, 0.3); /*This property adds a box shadow to
    the button when it is being hovered over by the cursor. It creates a shadow
    effect with an offset of 7 pixels horizontally and vertically, a blur radius of
     3 pixels, and a color defined by the rgba value (red, green, blue, alpha). In
    this case, the shadow has a black color with an opacity of 0.3.*/
}

#instructionsButton {
  position: absolute; /* This property positions the button element absolutely
    within its containing element, allowing you to precisely control its placement.
     It is positioned relative to its nearest positioned ancestor or to the initial
     containing block if there is no positioned ancestor. */
  top: 10px; /* This property sets the distance between the top edge of the button
    and the top edge of its containing element to 10 pixels. It determines the
    vertical positioning of the button. */
  right: 30px; /*This property sets the distance between the right edge of the
    button and the right edge of its containing element to 30 pixels. It determines
     the horizontal positioning of the button.*/
  font-size: 12px; /*This property sets the font size of the text inside the button
     to 12 pixels. It determines the size of the displayed text.*/
  /*These properties sets the width and height of the button element to
    automatically adjust based on its content. The button will take up the
    necessary width and height to fit its content.*/
  width: auto;
  height: auto;
  border-radius: 5px; /*This property sets the border radius of the button element
    to 5 pixels. It rounds the corners of the button, creating a slightly curved
    appearance.*/
  border-width: 1px; /*This property sets the width of the border around the button
     element to 1 pixel. It determines the thickness of the button's border.*/
  text-align: center; /*This property aligns the text inside the button element to
    the center. It horizontally centers the text within the button.*/
  background-color: #F79327; /*This property sets the background color of the
    button to a specific color represented by the hexadecimal value #F79327. It
    determines the color that fills the button's area.*/
}

#instructionsButton:hover {
  transform: scale(1.2); /*This property applies a scaling transformation to the #
    instructionsButton element when it is being hovered over. It scales the button
    by a factor of 1.2, making it 20% larger than its original size.*/
  background-color: #00DFA2; /*This property changes the background color of the #
    instructionsButton element when it is being hovered over. It sets the
    background color to a specific color represented by the hexadecimal value #00
    DFA2.*/
  color: white; /*This property changes the text color of the #instructionsButton
    element when it is being hovered over. It sets the text color to white,
    ensuring it is visible against the new background color.*/
  box-shadow: 3px 3px 2px rgba(0, 0, 0, 0.3); /*This property adds a box shadow to
    the #instructionsButton element when it is being hovered over. It creates a
    shadow effect with an offset of 3 pixels horizontally and vertically, a blur
    radius of 2 pixels, and a color defined by the rgba value (red, green, blue,
    alpha). In this case, the shadow has a black color with an opacity of 0.3.*/
}

#instructions {
  position: absolute; /*This property positions the #instructions element
    absolutely within its containing element. It allows you to specify the exact
    position of the element using the top and right properties.*/
  top: 50px; /*This property sets the distance between the top edge of the #
    instructions element and the top edge of its containing element to 50 pixels.*/
  right: 10px; /*This property sets the distance between the right edge of the #
    instructions element and the right edge of its containing element to 10 pixels
    .*/
  padding: 10px; /*This property adds 10 pixels of padding to the content of the #
    instructions element. Padding is the space between the content and the border
    of the element.*/
  background-color: #A6D0DD;
  border: 1px solid #ccc;
  max-width: 300px; /*This property sets the maximum width of the #instructions
    element to 300 pixels. If the content exceeds this width, it will be
    automatically resized to fit within this limit.*/
  margin-top: 30px;
  font-family: 'Times New Roman', Times, serif; /*This property sets the font
    family of the text inside the #instructions element to a series of font options
    . If 'Times New Roman' is not available, it will fall back to the 'Times' font,
     and if that is also not available, it will fall back to the generic serif font
    .*/
  font-size: 15px;
  color: #0A4D68;
  text-align: center;
  transition: transform 0.7s ease; /*This property specifies the transition effect
    that will be applied when there is a change in the transform property of the #
    instructions element. In this case, the transition effect lasts for 0.7 seconds
     and has an easing function that determines the speed of the transition.*/
}

#instructions:hover {
  transform: scale(1.1); /*This property applies a scale transformation to the #
    instructions element when it is being hovered over. The scale(1.1) value scales
     the element by a factor of 1.1, making it 10% larger than its original size.
    This creates an effect of slightly zooming in on the element.*/
}

/*Sets hover attributes for start and reset buttons*/
#startButton {
  display: flex;
  align-items: center;
  justify-content: center;
  margin-top: 20px;
  font-family: 'Times New Roman', Times, serif;
  background-color: #FF78C4;
}

#resetButton {
  display: flex;
  align-items: center;
  justify-content: center;
  margin-top: 10px;
  background-color: red;
  font-size: 20px;
  height: auto;
  width: auto;
  border-width: 3.5px;
}

#startButton:hover {
  transform: scale(1.2);
```

```css
  background-color: #30A2FF;
}

#resetButton:hover {
  transform: scale(1.2);
  background-color: #00C4FF;
  box-shadow: 3px 3px 2px rgba(0, 0, 0, 0.3);
}
/*The @keyframes blink defines a blinking animation that toggles the opacity of an
    element between fully opaque and transparent at regular intervals.*/
@keyframes blink {
  0% {
    opacity: 1;
  }
  50% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
}
/*Sets hover attributes and blinking attributes to reveal button.*/
#revealButton {
  position: fixed;
  top: 50%;
  right: 10px;
  transform: translateY(-50%);
  font-size: 25px;
  width: auto;
  height: auto;
  animation: blink 1s infinite;
}

#revealButton:hover {
  background-color: #19A7CE;
  animation-play-state: paused;
}

/*Sets hover attributes for leaderboard.*/
#leaderboard {
  position: fixed;
  top: 15%;
  left: 20px;
  font-size: 20px;
  font-family: Georgia, 'Times New Roman', Times, serif;
  color: #ECF8F9;
  background-color: #B31312;
  padding: 10px;
  transition: transform 0.7s ease, box-shadow 0.85s ease;

}

#leaderboard:hover {
  transform: scale(1.1);
  box-shadow: 5px 5px 3px rgba(0, 0, 0, 0.3);;
}

/*Sets marquee feature for tag having id=float.*/
#float {
  animation: float 15s linear infinite;
}

@keyframes float {
  0% {
    transform: translateX(300%);
  }
  100% {
    transform: translateX(-275%);
  }
}

#float:hover {
  animation-play-state: paused;
}
```

5.3 JavaScript

```
//calling elements having different ids as a variable in js file
const gridSizeOptions = document.getElementById('gridSizeOptions');
const startButton = document.getElementById('startButton');
const resetButton = document.getElementById('resetButton');
const gridContainer = document.getElementById('grid');
const scoreValue = document.getElementById('scoreValue');
const scoreDiv = document.getElementById('score');
const leaderboardContainer = document.getElementById('leaderboard');
const float = document.getElementById('float');
const fieldersCount = 11;

//initialising required variables
let gridSize = 6; // Default grid size
let successfulMoves = 0;
let lives = 0;
let wickets = 0;
let fielderLocations = [];
let score = 0;
let gameOver = false;
let gameStarted = false;
let playerName = '';
let leaderboard = JSON.parse(localStorage.getItem('leaderboard')) || [];

//function is responsible for toggling the visibility of an instructions div and
    updating the text of the corresponding button.
function toggleInstructions() {
    const instructionsDiv = document.getElementById('instructions');
    const insbtn = document.getElementById('instructionsButton')
    //checks the current value of display of instructionDiv
    if (instructionsDiv.style.display === 'none') {
        //if nstructionsDiv.style.display = 'none'
        insbtn.innerHTML = "Hide"; //This updates the text displayed on the button
    to indicate that clicking it will hide the instructions.
        instructionsDiv.style.display = 'block'; //This makes the instructions div
    visible by changing its display style to block-level.
    } else {
        //if above condition is false
        insbtn.innerHTML = "Instructions" //This updates the text displayed on the
    button to indicate that clicking it will show the instructions.
        instructionsDiv.style.display = 'none'; //This hides the instructions div
    by changing its display style to none, effectively removing it from the visible
     layout.
    }
}

//function is responsible for setting the grid size and adjusting the game elements
     accordingly. The function takes a parameter 'size' which represents the
    desired grid size.
function setGridSize(size) {
    gridSize = size;  //determines the number of rows and columns in the grid
    gridSizeOptions.style.display = 'none'; //This hides the grid size options from
     the user interface.
    startButton.style.display = 'block'; //This makes the start button visible to
    the user.

    // Set the number of lives based on grid size
    if (size <= 6) {
        lives = size - 2;
    } else {
        lives = 5;
    }
}

//The createGrid() function is responsible for dynamically creating a grid of
    blocks based on a selected grid size.
function createGrid() {
    gridContainer.style.gridTemplateColumns = `repeat(${gridSize}, 1fr)`; //sets
    the number of columns in the grid container using a CSS grid template. The
    gridSize variable determines the number of columns, and the 1fr value ensures
    that each column occupies an equal fraction of the available space.
    //intializing for loop
    for (let i = 0; i < gridSize * gridSize; i++) {
        const block = document.createElement('div'); //a new div element is created
        block.classList.add('block'); //The block element is assigned the CSS class
     "block" using
        block.dataset.index = i;
        block.addEventListener('click', blockClick); //An event listener is added
    to the block element using block.addEventListener('click', blockClick). This
    means that when the block is clicked, the blockClick function (which is not
    shown here) will be executed.
        gridContainer.appendChild(block); //he block element is appended to the
    gridContainer, which is presumably a container element in the HTML where the
    grid of blocks will be displayed.
    }
}

//function is responsible for randomly generating the locations of fielders in the
    game
function generateFielders() {
    fielderLocations = []; //initializes an empty array fielderLocations which will
     store the indices of the fielder positions.
    let fielderCount = 0;
    //determines the number of fielders based on the grid size
    if (gridSize <= 6) {
        fielderCount = gridSize;
    } else if (gridSize === 7) {
        fielderCount = gridSize + 2;
    } else {
        fielderCount = 11;
    }
    //enters a loop that iterates fielderCount times to generate the fielder
    positions.
    for (let i = 0; i < fielderCount; i++) {
        let randomIndex = Math.floor(Math.random() * gridSize * gridSize); // a
    random index is generated using Math.random() and multiplied by gridSize *
    gridSize to ensure it falls within the range of the grid size.
        //checks if the generated randomIndex is already present in the
    fielderLocations array using fielderLocations.includes(randomIndex). This
    ensures that the same index is not selected multiple times.
        while (fielderLocations.includes(randomIndex)) {
            randomIndex = Math.floor(Math.random() * gridSize * gridSize); //If the
     randomIndex is already present in fielderLocations, a new random index is
    generated until a unique index is found.
        }
        fielderLocations.push(randomIndex); //Once a unique index is obtained, it
    is added to the fielderLocations array using fielderLocations.push(randomIndex)
    .
    }
}

//function is called to initiate the game and set up the initial game state
```

```
function startGame() {
    gameStarted = true; //sets the gameStarted variable to true, indicating that the game has started.
    gameOver = false; //sets the gameOver variable to false, indicating that the game is not over.
    score = 0; //initializes the score variable to 0, representing the player's initial score.
    scoreDiv.style.display = 'block'; //displays the score div element on the screen. This element is responsible for showing the player's score during the game.
    gridContainer.innerHTML = ''; //clears the contents of the gridContainer element. This ensures that any previous grid elements are removed before creating a new grid.
    createGrid(); //calls the createGrid()
    generateFielders(); //calls the generateFielders()
    leaderboardContainer.style.display = 'block'; //displays the leaderboard container on the screen. This container holds the leaderboard information.
    gridContainer.style.display = 'grid'; // Display the grid after starting the game
    startButton.style.display = 'none'; // Hide the start button after starting the game
    resetButton.style.display = 'block'; // Show the reset button after starting the game
    float.style.display = 'none'; //hides the float element
    successfulMoves = 0; //initializes the successfulMoves variable to 0, representing the number of successful moves made by the player.
    revealButton.style.display = 'none'; //hides the reveal button. The purpose and behavior of this button are not clear from the provided code snippet.
    loadLeaderboard(); //calls the loadLeaderboard() function to load the leaderboard data and display it on the screen.
}

function blockClick(event) {
    //This condition checks if the gameOver flag is true. If the game is already over, the function returns early and does not execute the remaining code.
    if (gameOver) {
        return;
    }
    successfulMoves++; //Increments the successfulMoves variable by 1, representing the number of successful moves made by the player.
    //Checks if the number of successful moves is a multiple of 5. If it is, the reveal button is displayed by setting its display property to 'block'.
    if (successfulMoves%5 === 0) {
        revealButton.style.display = 'block';
    }
    const block = event.target; //Retrieves the clicked block element from the event object.
    const blockIndex = parseInt(block.dataset.index); //Retrieves the index of the clicked block from its data-index attribute and converts it to an integer.
    //Checks if the clicked block index is included in the fielderLocations array, indicating that it contains a fielder.
    if (fielderLocations.includes(blockIndex)) {
        //If there are remaining lives (lives > 0), the block's background image is set to 'out.png' to indicate that a wicket has fallen, and the wicketfall function is called.
        if (lives > 0) {
        block.style.backgroundImage = 'url(out.png)';
        wicketfall();
        }
        //If there are no remaining lives (lives === 0), the game is set to over (gameOver = true), the block's background image is set to 'out.png', the displayGameOver function is called to handle the game over scenario, and the updateLeaderboard function is called to update the leaderboard.
        if (lives === 0 ){
        gameOver = true;
        block.style.backgroundImage = 'url(out.png)';
        displayGameOver();
        updateLeaderboard();
        }
    }
    //If the clicked block does not contain a fielder:
    else {
        let scoreV = 0;
        //The score value (scoreV) is determined based on the grid size. A random number is generated, and based on probability ranges, a score value is assigned.
        if (gridSize <= 7) {
            const prob = Math.random();

            if (prob <= 2 / gridSize) {
                scoreV = 1;
            } else if (prob <= (2 / gridSize) + (1 / gridSize)) {
                scoreV = 2;
            } else if (prob <= (2 / gridSize) + (1 / gridSize) + (5 / (3 * gridSize))) {
                scoreV = 4;
            } else {
                scoreV = 6;
            }
        } else {
            const prob = Math.random();

            if (prob <= 0.2) {
                scoreV = 1;
            } else if (prob <= 0.2 + 0.1) {
                scoreV = 2;
            } else if (prob <= 0.2 + 0.1 + 0.05) {
                scoreV = 3;
            } else if (prob <= 0.2 + 0.1 + 0.05 + 0.35) {
                scoreV = 4;
            } else {
                scoreV = 6;
            }
        }

        score = score + scoreV; //The score variable is updated by adding the scoreV.
        scoreValue.textContent = score + "/" + wickets; //The scoreValue element's text content is updated to display the updated score and wickets.
        block.textContent = scoreV; //The clicked block's text content is set to the scoreV, displaying the score on the block.
        block.style.backgroundColor = '#DB005B'; //The clicked block's background color is set to #DB005B.
        block.removeEventListener('click', blockClick); //The click event listener is removed from the block, preventing further clicks on the same block.
        block.style.transform = 'rotateY(360deg)'; //The clicked block is rotated 360 degrees using the transform property.
    }
}

//function reveals a randomly chosen fielder tile for a short duration
function revealFielder() {
    const fielderIndex = fielderLocations[Math.floor(Math.random() * fielderLocations.length)]; //Generates a random index from the fielderLocations array using Math.random() and Math.floor(). This index corresponds to a randomly chosen fielder block.
    const fielderBlock = document.querySelector(`[data-index="${fielderIndex}"]`); // Selects the fielder block element based on the randomly chosen index using a CSS attribute selector.
    fielderBlock.style.backgroundImage = 'url(fielder.png)'; //Sets the background image of the fielder block to 'fielder.png', revealing the fielder.
    //Sets a timeout function to execute after 1000 milliseconds (1 second).
    setTimeout(function () {
        fielderBlock.style.backgroundImage = 'none'; //Removes the background image from the fielder block, hiding the fielder.
        fielderBlock.style.backgroundColor = '#E8AA42'; //Sets the background color of the fielder block to #E8AA42, restoring its original appearance.
    }, 1000);

    revealButton.style.display = 'none'; //Hides the reveal button after it has been clicked.
}

//function is called when a fielder tile is clicked, indicating that a wicket has fallen
function wicketfall() {
    wickets++; // Increments the wicket count by 1, tracking the number of wickets that have fallen.
    lives--; //Decrements the number of remaining lives by 1. Lives represent the number of remaining chances the player has before the game is over.
    successfulMoves = 0; //Resets the successfulMoves variable to 0. successfulMoves keeps track of the number of consecutive successful moves made by the player without losing a wicket.
    scoreValue.textContent = score + "/" + wickets; //Updates the text content of the scoreValue element to display the current score and wicket count.
    revealButton.style.display = 'none';
    alert(`Out!. You lost a wicket at ${score}. Current Score: ${score}/${wickets}`); //Displays an alert dialog box indicating that a wicket has been lost. The alert message includes the score at which the wicket fell and the current score and wicket count.
}

//function is called when the player clicks reset button
function resetGame() {
    successfulMoves = 0; // Resets the successfulMoves variable to 0. This variable keeps track of the number of consecutive successful moves made by the player without losing a wicket.
    revealButton.style.display = 'none'; //Hides the revealButton element. This button allows the player to reveal a fielder block.
    location.reload(); // Reload the page to reset the game and go back to the homepage
}

//function is called when the game is over, meaning the player has lost all their wickets.
function displayGameOver() {
    revealButton.style.display = 'none';
    alert(`All Out! Your final score is ${score}.`); //Displays an alert box with a message informing the player that the game is over and showing their final score.
    //Checks if the player has scored any runs (score > 0). If they have, it calls the promptPlayerName function. This function prompts the player to enter their name for the leaderboard if they have scored any runs.
    if (score > 0) {
      promptPlayerName();
    }
    displayLeaderboard(); //Calls the displayLeaderboard function. This function displays the leaderboard, showing the current high scores achieved by players.
}

//function is responsible for prompting the player to enter their name after the game is over and their score is greater than zero.
function promptPlayerName() {
    playerName = prompt('Enter your name:'); //Displays a prompt dialog box with a message asking the player to enter their name. The entered name is stored in the playerName variable.
    // Checks if the playerName variable is not empty (the player entered a name).
    if (playerName) {
      addToLeaderboard(); //This function adds the player's name and score to the leaderboard. It typically updates the leaderboard data structure or sends the player's name and score to a server for storage.
      displayLeaderboard(); //This function displays the updated leaderboard, including the newly added player's name and score.
    }
  }

// function is responsible for loading the leaderboard data from the browser's local storage and updating the leaderboard display
function loadLeaderboard() {
    const storedLeaderboard = localStorage.getItem('leaderboard'); //Retrieves the leaderboard data from the browser's local storage using the key 'leaderboard'. The data is stored as a string.
    //Checks if there is any stored leaderboard data in the local storage.
    if (storedLeaderboard) {
      leaderboard = JSON.parse(storedLeaderboard); //parses the string data back into a JavaScript object using JSON.parse(). The parsed leaderboard data is stored in the leaderboard variable.
      updateLeaderboard(); //Calls the updateLeaderboard function to update the leaderboard display based on the loaded data.
    }
  }

// function is responsible for adding the current player's data to the leaderboard and storing it in the browser's local storage
function addToLeaderboard() {
    // Creates a new JavaScript object playerData that represents the current player's data.
    const playerData = {
      name: playerName,
      score: score,
    };
    leaderboard.push(playerData); //Adds the playerData object to the leaderboard array. This appends the player's data to the end of the array.
    leaderboard.sort((a, b) => b.score - a.score); // Sort leaderboard in descending order by score
    //Checks if the leaderboard has more than 10 entries.
    if (leaderboard.length > 10) {
      leaderboard.pop(); // Keep only the top 10 scores
    }
    localStorage.setItem('leaderboard', JSON.stringify(leaderboard)); // Store leaderboard data in local storage
  }

//function is responsible for rendering the leaderboard on the webpage
function displayLeaderboard() {
    leaderboardContainer.innerHTML = '<h2>Leaderboard</h2>'; //ets the content of the leaderboardContainer element to include an <h2> heading with the text "Leaderboard". This clears any existing content in the container.
    //Checks if length of array is 0.
    if (leaderboard.length === 0) {
      leaderboardContainer.innerHTML += '<p>No scores yet.</p>'; //appends a <p> paragraph element to the leaderboardContainer with the text "No scores yet.".
    } else {
        //iterates over each player object in the leaderboard using the forEach method
        leaderboard.forEach((player, index) => {
        leaderboardContainer.innerHTML += `<p>${index + 1}. ${player.name}: ${player.score}</p>`; //appends a <p> paragraph element to the leaderboardContainer with the player's rank, name, and score.
      });
    }
  }
```

References

[1] URL: https://www.w3schools.com/html/.

[2] URL: https://www.w3schools.com/css/.

[3] URL: https://www.w3schools.com/js/.