Project CS104

Prasoon Badjatya

April 2025

1 Introduction

For my CS104 project, I developed Howzatt! Cricket Scorekeeper, a web-based cricket scoring application. While fulfilling all the core requirements of the assignment, I have also implemented several enhancements that extend the functionality and improve the user experience. These customizations include support for multiple types of deliveries, match history storage, randomized match simulations with adjustable speeds, a commentary section with searchable player-specific logs, and additional features discussed in detail throughout this report.

2 HTML Files

The application consists of the following HTML files:

1. `setup.html`
2. `live.html`
3. `scoreboard.html`
4. `summary.html`
5. `random.html`
6. `history.html`

2.1 setup.html

This is the landing page of the application. Users can either configure a manual match or initiate a randomized simulation, which redirects to `random.html`. Additionally, users can view the history of previous matches. Inputs for a manual match include team names, toss winner, toss decision, and number of overs. The logo displayed on the top of the page is hand-drawn. The visual theme across all pages is inspired by retro pixelated games, offering a consistent and engaging aesthetic.

2.2 live.html

This page enables manual match simulation. Upon initialization, the user is prompted to enter the striker, non-striker, and bowler in sequence. To streamline the interface, input fields disappear once data is entered. The user can simulate the following delivery types:

1. 0 runs
2. 1 run
3. 2 runs
4. 3 runs
5. 4 runs
6. 6 runs

7. Wicket
8. No ball
9. Bye
10. Leg-bye
11. Wide

These delivery types reflect real-life cricket rules. Some options, such as no-ball and bye, are implemented as checkboxes to allow combinations (e.g., scoring runs off a no-ball). The page features a progress bar indicating the match's current status, a commentary box, and a dynamically updating scoreboard card that shows current batters and bowlers with their respective statistics. A button is provided to navigate to the full scoreboard.

2.3 scoreboard.html

This page displays two tables: one for batters and one for bowlers, showing their performance data. Upon navigation from `live.html`, the match state is automatically preserved and resumed. Players are grouped by team, and the interface ensures clarity and completeness of information.

2.4 summary.html

Once a match concludes, either manually or through simulation, users are redirected to `summary.html`. This page announces the winner, margin of victory (runs or wickets), or if the match ended in a draw. A search feature allows users to retrieve commentary related to specific players. Three action buttons are provided: Save Match, View History, and Start New Match (the last also saves the current match automatically).

2.5 random.html

This file handles simulated matches with predefined logic. Team and player names are drawn randomly from a file, and the number of overs is fixed at two. The page includes a live commentary section, progress bar, and the option to choose from four match speeds: Slow, Normal, Fast, and Very Fast. The layout mimics `live.html` with real-time updates. After completion, the simulation redirects to `summary.html`, where commentary remains accessible.

2.6 history.html

This page provides a record of all previously played matches, showing minimal yet essential details: the teams involved, winner, and final scores. Users may also clear the history, which removes all locally stored match data. A button is provided to initiate a new match, redirecting to `setup.html`. This functionality is driven by an internal JavaScript file that reads from and writes to the browser's local storage.

3 CSS Files

The project includes the following CSS files:

1. `basic_ui.css`
2. `setup.css`
3. `live.css`
4. `scoreboard.css`
5. `summary.css`
6. `random.css`
7. `history.css`

Each file is dedicated to styling a specific page of the application. A consistent retro game aesthetic is maintained throughout, incorporating custom animations, glow effects, and pixel-art-style images. Notably, a hand-designed logo reinforces the visual identity of the application. The styling prioritizes clarity, visual consistency, and user engagement.

4 All js files

The js files are:

1. score.js
2. random.js
3. summary.js
4. match.js

4.1 score.js

This is the main js file where the main game logic is implemented. This contains many variables and classes. When I first was making the project I made the classes inside the file only but when I felt the need for the classes to be exported to other files also I made a seperate file called match.js which contains eminent functions and classes. I have listed all the variables, functions and classes with their functionalities at the end of this section. This mainly works for setup.html, live,html, scoreboard.html, and summary.html.

4.2 random.js

This file contains the logic for the simulation. It is completely seperate from score.js and only uses the classes and different functions from match.js. This also handles the different speed selected for the match in a clever way, By just pausing and starting the match at different intervals! (I asked ChatGPT for this :)) . It also calls the functions recursively so that the matches goes on. Note that it does not contain the different types of balls, just the same ol' runs and a wicket. It also picks the team who won the toss and what they chose randomly. Moreover the players and their runs scored are also randomized. It then automatically redirects to summary.html after the match is over and saves it. The different functions included are

- **initSimulation()**: Initializes a new random match by randomly selecting team names, assigning batting order based on a simulated toss, and creating initial batters and bowlers.
- **createPlayers(team, type)**: Adds a random player (either a `Batter` or a `Bowler`) to the specified team.
- **startSimulation()**: Starts the ball-by-ball simulation by running the `simulateBall()` function at regular time intervals based on the selected speed.
- **pauseSimulation()**: Pauses the ongoing simulation by clearing the interval timer and toggling button visibility.
- **resetSimulation()**: Resets the match to its initial state, reinitializing teams, players, and match data, and prepares for a fresh simulation.
- **simulateBall()**: Simulates the outcome of a single ball by randomly selecting a result (run, wicket, wide, etc.) and updating match progress.
- **processBall(run)**: Updates the match data based on the outcome of a ball, including runs scored, wickets, overs completed, commentary updates, and checking for innings or match end.
- **changeInnings()**: Swaps the active and inactive teams when an innings is completed, resets relevant counters, and initializes new batters and bowlers.
- **endMatch()**: Finalizes the match, declares the winner or a draw, saves the match summary into Local Storage, and redirects to the history page.
- **updateTeamInfo()**: Updates the displayed team names, scores, and current innings information on the webpage UI.
- **updatePlayerStats()**: Updates the batter and bowler statistics displayed on the webpage for the current players.

4.3 summary.js

This js script added only for the customizations part the basic part is already handled by score.js. The script handles the saving of the match and fetching the commentaries related to each player based on their team. So it does not really matters if there are players with the same name! Moreover you can view the history or directly start a new match. The page also saves the matches history in the Local Storage.

4.4 match.js

This js contains classes and functions that are required to all the other js files. The classes are - Matches, Team, Batter and Bowler. The functions include - addCommentary, clearLocalStorage, won and reviveTeam. More about them in the next section.

5 All parameters used

5.1 Classes

5.1.1 Match

The Match class represents a cricket (or any sport) match. It stores basic information about a match — like the two teams that played, who won, their scores, and when the match happened. The parameter are as following -

1. id
2. team1
3. team2
4. winner
5. team1 score
6. team2 score
7. timestamp

A particular id is given to each match saved the timestamp is made. The id is generated randomly using todays date and picking a number random from 0 to 1000. This is done by the generateID function.

5.1.2 Team

The Team class models a cricket team during a match, keeping track of:

1. Players (batters + bowlers)
2. Score (runs, wickets, balls)
3. Whether they are currently batting
4. Whether they have already played their innings

The last 2 are of boolean type (0 or 1). The Players is an array and the score gets updated using the different methods as mentioned below.

1. addBatter(batter)
2. addBowler(bowler)
3. addBall()
4. addRun(run)
5. addWicket()
6. getScore()
7. crr()
8. rrr()

All of them works like their name.

5.1.3 Batter

The Batter class represents a single player who is batting. It tracks personal performance stats and commentary during their innings.

- **name**: Name of the batter.
- **runs**: Runs scored by the batter.
- **balls**: Balls faced by the batter.
- **fours**: Number of boundaries (4 runs) hit.
- **sixes**: Number of sixes (6 runs) hit.
- **team**: Team name the batter plays for.
- **commentary**: (Optional) An array of commentary text.

The methods now to update the respective objects -

- `addRun(run)`: Adds the number of runs. Updates fours and sixes count based on the run value.
- `addBall()`: Increments the number of balls faced by 1.
- `sr()`: Calculates and returns the strike rate as Strike Rate $= \frac{\text{runs}}{\text{balls}} * 100$.
- `addComment(text)`: Adds a commentary line to the commentary array.

5.2 Bowler Class

The `Bowler` class models a cricket bowler, tracking their bowling statistics and commentary.

5.2.1 Constructor

- **name**: Name of the bowler.
- **balls**: Number of balls bowled.
- **overs**: Completed overs bowled.
- **runs**: Runs conceded.
- **wickets**: Wickets taken.
- **team**: Team name.
- **maidens**: Maiden overs bowled (initialized to 0).
- **currentOverRuns**: Runs conceded in the current over (initialized to 0).
- **commentary**: (Optional) An array of commentary entries.

5.2.2 Methods

- `addRun(run)`: Adds runs conceded, both to total and current over.
- `addWicket()`: Increments wicket count by 1.
- `addBall()`: Increments balls bowled by 1.
- `addOver()`: Increments overs bowled by 1. If no runs conceded in the over, increments maiden count. Resets current over runs.
- `addComment(text)`: Adds a commentary line to the commentary array.

5.3 addCommentary Function

The `addCommentary(text)` function handles updating the live commentary section during the match. It performs the following actions:

- Creates a new `<div>` element with the class `commentary-entry`.
- Sets the text content of the new element to the given commentary text.
- Appends the newly created commentary entry to the `commentaryBox` element.
- Automatically scrolls the `commentaryBox` to the bottom, ensuring that the most recent commentary is visible.

Earlier, there was an attempt to also store commentary entries in `Local Storage` for persistence, but this functionality is currently commented out.

6 Game Logic Explanation

The core gameplay logic for Howzatt! Cricket Scorekeeper is implemented in `score.js`. The match progresses through the following structured flow:

6.1 Match Setup

- The user enters both team names, toss winner, toss decision (bat/bowl), and overs.
- Based on the toss decision, the batting and bowling teams are determined.
- The setup information is saved to `Local Storage`, and the game navigates to `live.html`.

6.2 Match Progression

- The batting side (`active`) and bowling side (`inactive`) are set according to the toss results.
- Two batters (striker and non-striker) and one bowler are set at the beginning of the innings.
- Different types of balls (runs, wicket, no-ball, bye, leg-bye, wide) are handled when a user selects an outcome and clicks the `Play Ball` button.

6.3 Ball Outcome Handling

- For each ball:
  - Runs and balls are updated for batters and bowlers.
  - Special cases like **no-ball**, **bye**, **leg-bye**, and **wide** are treated separately, modifying the rules accordingly (e.g., extra runs, no valid ball counted).
  - For a **wicket**, a new batter is prompted immediately.
- Commentary is dynamically generated for each event.

6.4 Over Handling

- After every six valid balls:
  - A new bowler must be chosen.
  - Striker and non-striker batters are swapped automatically.

6.5 Innings Change

- An innings is considered complete if:
  1. All 10 wickets fall, or
  2. The allotted overs are completed.
- When an innings ends:
  - Teams swap roles (batting ↔ bowling).
  - Batters and bowlers are reset.

6.6 Match End

- The match ends if:
  - Both innings are completed, or
  - The chasing team overtakes the target score.
- The result (win/loss/draw) is displayed and the match automatically redirects to `summary.html`.

6.7 State Persistence

- The current state (teams, players, scores, commentary, etc.) is saved continuously in `Local Storage`.
- This allows users to navigate between `live.html` and `scoreboard.html` without losing progress.

6.8 User Interactions and Navigation

- Users can manually add batters and bowlers during the game.
- Buttons are provided for moving between live scoring and scoreboard views.
- Proper validations ensure duplicate or invalid entries are not allowed.

7 Challenges Faced and Solutions

During the development of Howzatt! Cricket Scorekeeper, I encountered several challenges which helped me improve my understanding of JavaScript event handling and data management.

7.1 Handling Player Data

The first major problem I faced was managing the statistics and data for each individual player (batsmen and bowlers). Initially, storing variables separately for each player became messy and unmanageable. To solve this, I created dedicated `Batter` and `Bowler` classes, which made it easier to track performance, maintain consistency, and expand functionality later on.

7.2 Event Listeners Across Pages

Since `score.js` was shared across multiple HTML pages, many elements that the script tried to access did not exist on every page. This caused errors when the script tried to add event listeners to non-existent buttons. I fixed this by adding `if` conditions before attaching event listeners, ensuring they were only set up if the relevant elements were present on the page.

7.3 State Persistence Between Pages

Another major issue occurred when navigating to `scoreboard.html` mid-match. Visiting the scoreboard page would reset the entire match state, causing unexpected errors. To address this, I implemented a `saveState()` function that saves the current match state into `Local Storage` before navigation. This allowed smooth transitions between pages without losing any match progress. While implementing this, I also noticed issues when a batsman or bowler prompted would just vanish while returning. So I made necessary to fill out the hanging form first before navigating to scorreboard.html.

7.4 Overs Not Updating in Random Simulation

While developing `random.html`, I noticed that overs were not incrementing properly and innings transitions were not reflected correctly on the UI. I debugged the issue using `console.log()` statements and found that during simulation, two batters (striker and non-striker) were being added simultaneously, causing inconsistencies. Fixing the logic around batter addition solved the problem.

7.5 Other Minor Issues

Some minor inconveniences included event listeners not triggering properly and incorrect matches loading due to improper clearing of `Local Storage`. These were addressed through careful checking, resetting states correctly, and improving overall error handling. So if you see any console.log please ignore them.

A special note

All the basic implemetation is done in score.js itself as told but since I required more js file and it was becoming difficult to manage the file I made some new files. I hope thats okay.

Usage of AI

LLMs were used majorly for the css part and some of the errors or concepts that I didnt knew. The portion written by AI in the simulation was:

```
    const outcomes = [
        { runs: 0, weight: 30 },
        { runs: 1, weight: 25 },
        { runs: 2, weight: 15 },
        { runs: 3, weight: 5 },
        { runs: 4, weight: 15 },
        { runs: 6, weight: 5 },
        { runs: -1, weight: 5 }
    ];


    if (Math.random() < 0.1) {
        outcomes.push({ runs: 7, weight: 5 });
    }

    const totalWeight = outcomes.reduce((sum, outcome) => sum + outcome.weight, 0);
    let random = Math.random() * totalWeight;
    let selectedOutcome = outcomes[0];

    for (const outcome of outcomes) {
        if (random < outcome.weight) {
            selectedOutcome = outcome;
            break;
        }
        random -= outcome.weight;
    }


    processBall(selectedOutcome.runs);


    matchInfo.currentBall++;
    const totalProgress = (matchInfo.currentBall / (matchInfo.totalBalls * 2)) * 100;
    progressBar.style.width = `${totalProgress}%`;
}
```

More features I would like to add

- Players and team cards
- Resuming a game from where you left of
- A light and a dark theme toggle
- More features like the weather controlling, the pitch condition etc.
- And adding more features for random.html like no.overs, a scoreboard etc.
- Adding a coin flip.

References

- https://stackoverflow.com/questions/23113655/clear-local-storage-but-exempt-certain-values
- https://www.youtube.com/watch?v=CJ26NLtdzPA
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/floor
- https://www.google.com/search?q=how+to+see+all+event+listeners+javascript+from+console&oq=how+to+see+all+event+listeners+javascript+from+console&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIHCAEQIRigATIHCAIQIRig sourceid=chrome&ie=UTF-8