CS5003 — Masters Programming Projects

Assignment: P2 – Project 2

Deadline: Wednesday 3rd April (Week 8) 21:00
Credits: 33% of the overall module grade

NOTE: MMS IS THE DEFINITIVE SOURCE FOR DEADLINES AND CREDIT DETAILS

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Aims

The main aim of this project is to teach you to write a complete web application including both client and server components. More specifically, it will involve designing and testing an API, implementing the front-end and back-end in Javascript which communicate through this API, and choosing the right data representation. You will be using Node.js packages such as Express. The use of git version control system is mandatory for this practical.

Overview

You will create an online game that allows different users to connect and to play against each other. A basic implementation should implement a simplified game of *battleship*[1], where players take it in turns guess where their opponent's ships are on a 10 x 10 grid. The game is over when one player has correctly guessed where all their opponent's ships are (i.e. when a player has targeted and sunk all their opponent's ships).

Working in pairs, you will implement both the client side (HTML+CSS+JS) and the server side (JavaScript based on Node.js), The server side will implement a RESTful API for exchanging data with the client. Your webpage will contain client-side JavaScript which makes HTTP calls to the API, and exchanges data with the server using JSON.

The client should be able to let a user connect to the server, place their ships on their own grid, target cells on their opponent's grid, and display the current state of both their own and their opponent's grids. Information about which cell has been targeted should be sent to the server via the API, and the game state will be held centrally on the server. Your solution should allow two people to play against each other from two separate computers.

The API should provide services needed to make this happen including, but not limited to:

- starting a new game
- configuring the game (player name, size of grid, number/type of ships, game variant, etc. . .)
- placing ships on the player's grid
- targeting a cell on the opponent's grid

[1] `https://en.wikipedia.org/wiki/Battleship_(game)#Description`

- detecting hit/miss/victory conditions

This is a fairly open-ended project, but you should ensure your application provides certain core functionality as described Section Requirements.

Requirements

Your application should provide the functionality described below. You should make sure you have completed all the requirements in each section before moving onto the next.

Basic

Your application should provide the following:

- Allow a player to start a new game against another player.
- Allow a player to place their ships on their own grid. Each ship occupies a number of consecutive squares on the grid, arranged either horizontally or vertically. The number of squares for each ship is determined by the type of the ship:

#	Class of ship	Size
1x	Aircraft Carrier	5
1x	Battleship	4
1x	Cruiser	3
2x	Destroyer	2
2x	Submarine	1

- Allow a player to target a cell on their opponent's grid. If the opponent has part of a ship in the cell, it counts as a 'hit', if not, it counts as a 'miss'. This should be displayed appropriately on each player's grids.
- Finish the game appropriately when all parts of all of a player's ships have been 'hit'.

Intermediate

Your application should provide all the requirements described in Section Basic, plus the following:

- Allow a player to enter their name.
- Allow a player to choose whether each ship is placed horizontally or vertically on the grid.
- Ensure a player's ships do not overlap (i.e. only one ship can occupy any cell on a player's grid).
- Ensure a player's ships remain within the bounds of the grid.
- Detect when all parts of an individual ship have been 'hit' and announce that it has been 'sunk'.
- Keep track of a player's score (e.g. how many games played/won).
- Allow a player to stop or restart a game. Deal with this appropriately for a player's score.

Advanced

Your application should provide all the requirements described in Sections Basic and Intermediate, plus the following:

- Integrate media such as images, animations and sounds (easy — hard).

- Allow the player to configure the game rules, e.g. size of grid, number/type of ships, specific game variants such as *Salvo*[2] (medium)
- Provide database connectivity to store previous games (medium).
- Provide real-time interaction using Web Sockets (medium).
- Allow games to be recorded and replayed (hard).
- Provide user authentication (hard).

Deliverables

A single `.zip` file must be submitted electronically via MMS by the deadline. It should contain:

- The entire working copy of your repository, containing the source code and revision history
- A **joint** report (around 2000 words), in `PDF` format detailing the design of your solution, discussing any requirements and design decisions taken, your approach to testing, your approach to team work (e.g. use of tools such as Trello, or techniques such as pair programming), and reflecting on the success of your application and development process. Try to focus on the reasons for your decisions rather than just providing a description of what you did. **This will be joint work of both partners.**
- An **individual** report (around 1000 words), in `PDF` format describing your own contribution and your own challenges.
- A short README file describing how to run your server and listing any node packages which need to be installed.

Submissions in any other formats may be rejected.

Marking Criteria

Marking will follow the guidelines given in the school student handbook:
`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptor`

Your submission will *not* be evaluated based on aesthetic appeal (this is not a visual design course), but use of CSS and DOM scripting which enhances the experience and interactivity will be rewarded.

There will be a group mark (based on the quality of the final application and joint report) and an individual mark (based on your individual contribution and individual report). The git repository will be used during marking to evaluate individual contribution.

Your final mark will be the average of the group mark and your individual mark.

Some specific descriptors for the **group component** of the assignment are given below:

- A *poor implementation* **in the 0–7 grade band** will be missing nearly all required functionality. It may contain code attempting a significant part of a solution, but with little success, together with a report describing the problems and the attempts made at a solution.
- A *reasonable implementation* **in the 8–10 grade band** should provide some of the functionality described in Section Basic, and demonstrate reasonable use of HTML, Javascript and Node.js. The code should be documented well enough to allow the marker to understand the logic. The report should describe what was done but might lack detail or clarity.

[2] `https://en.wikipedia.org/wiki/Battleship_(game)#Variations`

- A *competent implementation* **in the 11–13 grade band** should provide all the functionality described in Section Basic, demonstrate competent use of HTML and CSS (e.g. for layout and positioning), allowing players to play a complete game against an online opponent. The code should be documented well enough to allow the marker to understand the logic and should have a modular design. The report should describe clearly what was done, with good style.

- A *good implementation* **in the 14–16 range** should provide all the functionality described in Section Basic and some or all of the functionality from Section Intermediate. It should demonstrate good code quality, good comments, modular design, and proper error handling. All of the JSON objects passed through the API must be checked and validated. Good use of CSS for layout and styling is expected for a submission in this range (not unmodified defaults). The report should describe clearly what was done with some justification for decision, with good style, showing a good level of understanding.

- An *excellent implementation* **in the 17 and higher range** should provide all the functionality described in Sections Basic and Intermediate, and some or all of the functionality from Section Advanced. It should demonstrate high-quality code and be accompanied by a clear and well-written report showing real insight into the subject matter.
  Note: For this practical you do not need to invent your own extensions. Concentrate on providing high quality, sophisticated implementations of the requirements in this specification and writing an insightful report demonstrating understanding.

Some specific descriptors for the **individual component** of the assignment are given below:

- A *poor contribution* **in the 0–7 grade band** indicates a weak or missing individual report, demonstrating confusion and misunderstanding of the topic. Little or no contribution to the final software, adding little to no value and focussing on only one part of the program functionality.

- A *reasonable contribution* **in the 8–10 grade band** indicates an individual report lacking detail or clarity, with a small contribution to the final software, which added little value and focused on only one part of the program functionality.

- A *competent contribution* **in the 11–13 grade band** indicates

  - a good individual report, but with a small contribution to the final software, focussing on only one part of the program functionality.
  - OR an individual report lacking detail or clarity, but with regular contributions to the project limited to a few parts of the project.

- A *good contribution* **in the 14–16 range** indicates a good report, regular contributions to the project, but limited to a few parts of the project.

- An *excellent contribution* **in the 17 and higher range** indicates an excellent individual report, regular and sizeable code contributions throughout the project, and contribution to many important parts of the project.

Word Limit

An advisory word limit of approximately 2000 words applies to the joint report for this assignment and an advisory word limit of approximately 1000 words applies to the individual report for this assignment. Word limits exclude references and appendices. No automatic penalties will be applied based on report length but your mark may still be affected if the report is short and lacking in detail or long and lacking focus or clarity of expression.

A word count must be provided at the start of each report.

Lateness Penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

Good Academic Practice

The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Hints and suggestions

You do not have to follow any of these suggestions, but you might find some of them useful.

Start by spending some time on design – what needs to be done to meet the basic specification. What type of data needs to be represented and where (client or server, database or memory). What is the best way to represent the data – objects, classes, JSON objects, etc. Write a basic outline of your object/class structure and RESTful API on paper and think whether it makes sense. It pays to spend time with your partner and discuss this in detail before starting to code.

Divide the work into small chunks and decide how to split the work between the two of you. Agree on a rough work plan, and meet regularly to see how the work is progressing – sometimes you will need to adjust because things turn out to be more difficult than planned.

Have the basic requirements working fully before venturing into intermediate or advanced requirements. A clean, fully functional and well-written basic implementation is better than a buggy mess with extensions.

Try to avoid splitting the work into parts that are mostly separate (like only client-side and only server-side) and working on them independently. Integrating such work at the very end is likely to be difficult. Try to work on related aspects – like splitting the API between the two of you, or working on different aspects of the user interface. Then integrate work at regular intervals so you can look at each other's code from time to time. It will help you understand your partner's thinking, help spot problems, and help you pick up good habits and tricks from each other.

Make use of existing resources where available – but make sure that there are no licensing issues and that you credit external sources. A very useful resource for graphics is `openclipart.org`.

Finally

Don't forget to enjoy yourselves and use the opportunity to experiment and learn! If you have any questions or problems please let me know (`mailto:ruth.letham@st-andrews.ac.uk`) — don't suffer in silence!