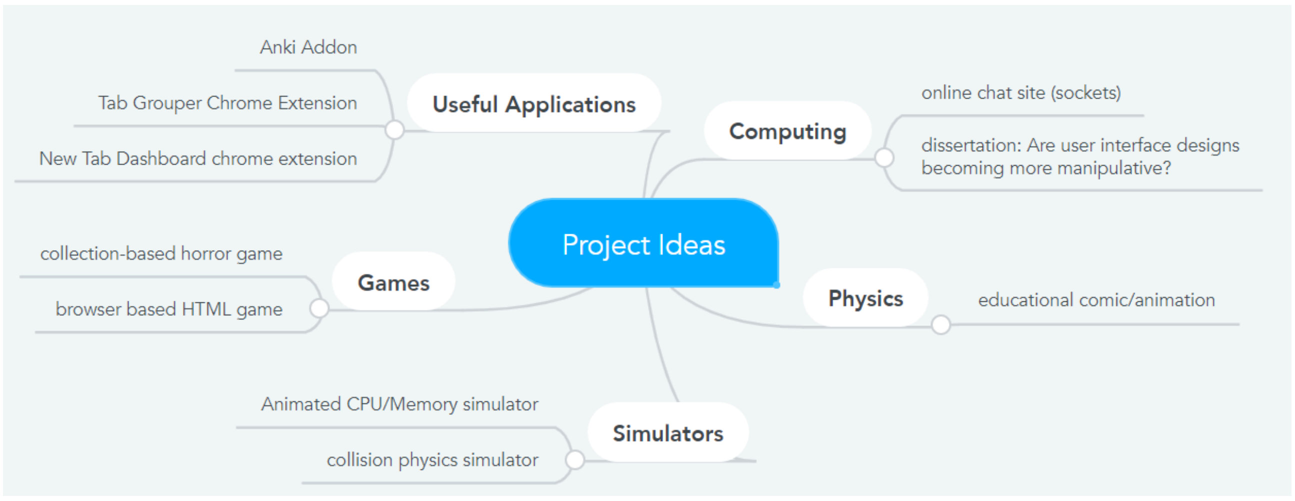
A MULTIPLAYER BROWSER-BASED GAME

Michelle Adetunji

Contents

Ideas


Project Ideas
Useful Applications
Anki Addon
Tab Grouper Chrome Extension
New Tab Dashboard chrome extension
Computing
online chat site (sockets)
dissertation: Are user interface designs becoming more manipulative?
Games
collection-based horror game
browser based HTML game
Physics
educational comic/animation
Simulators
Animated CPU/Memory simulator
collision physics simulator

Aside from the dissertation, I am eager to create a usable application or game that can be published online and viewed by anyone. This would require me to learn about app deployment and decide on specific platforms. I would also like to create an internet-based project like a Chrome extension or a flashcard app modification that makes use of APIs, so I can learn the fundamentals of networking and how I can apply them to a program.

Rationale

The aim of my project is to create a HTML, browser-based multiplayer game, with a focus on the design of the server and networking. The outcome of my project will be a multi-user app that can be accessed by anyone via a browser.

I will need to learn JavaScript to program the logic of the clients and server, and HTML and CSS to lay out and design the actual website. I will also need to learn about multiplayer server management and online communication between multiple devices.

My research will be largely practical – I will follow tutorials and create smaller practice projects to learn the processes required for my own game. I will create several prototypes to separate the development of different mechanics.

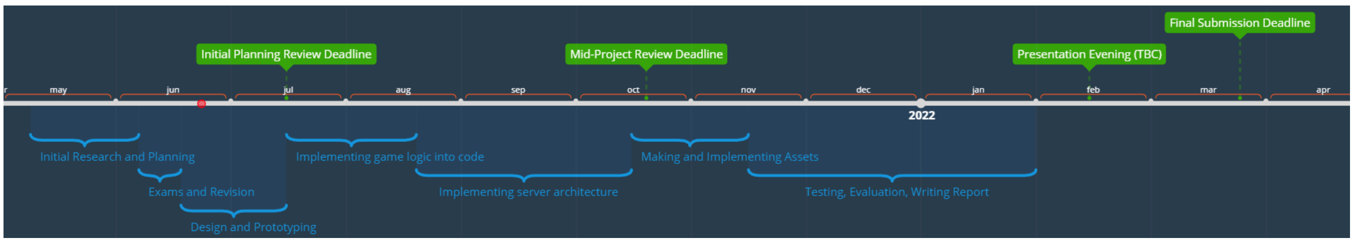
Planning

Game Design Document

- Players:
    - move with WASD/arrow keys
    - can tag other players by touching them
- Field:
    - the space where players move
    - does not scroll and remains a fixed size
- Visuals:
    - greyscale background with colourful players
    - background is a chessboard style pattern

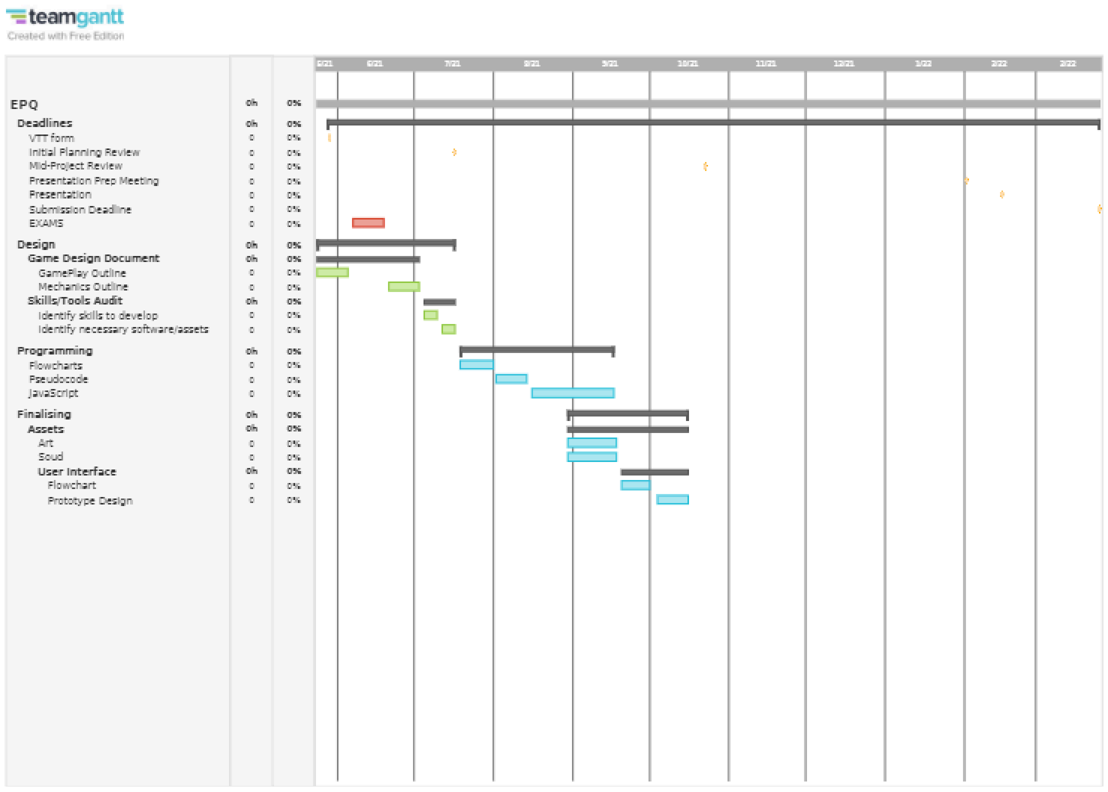
- audience: older children and young teens (10-15 years)
    - colourful and appealing visuals
    - the PC platform might be less accessible to younger children

Timeline:

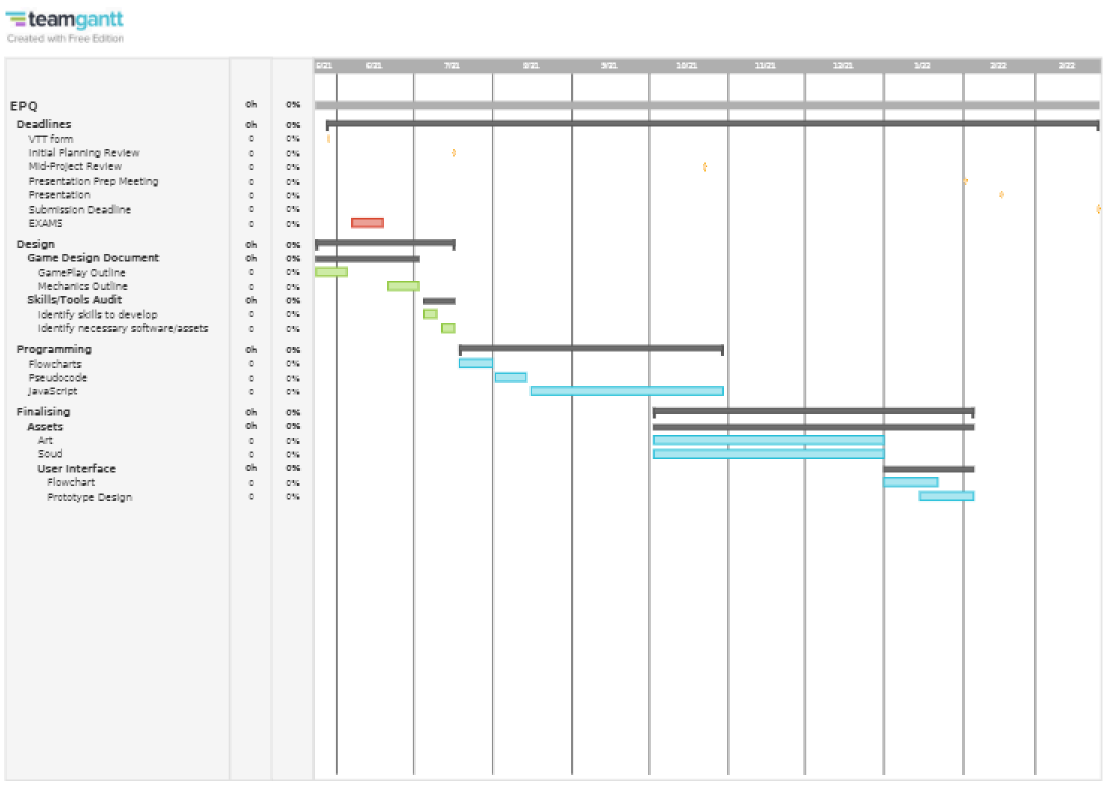


Gantt Chart Plans:

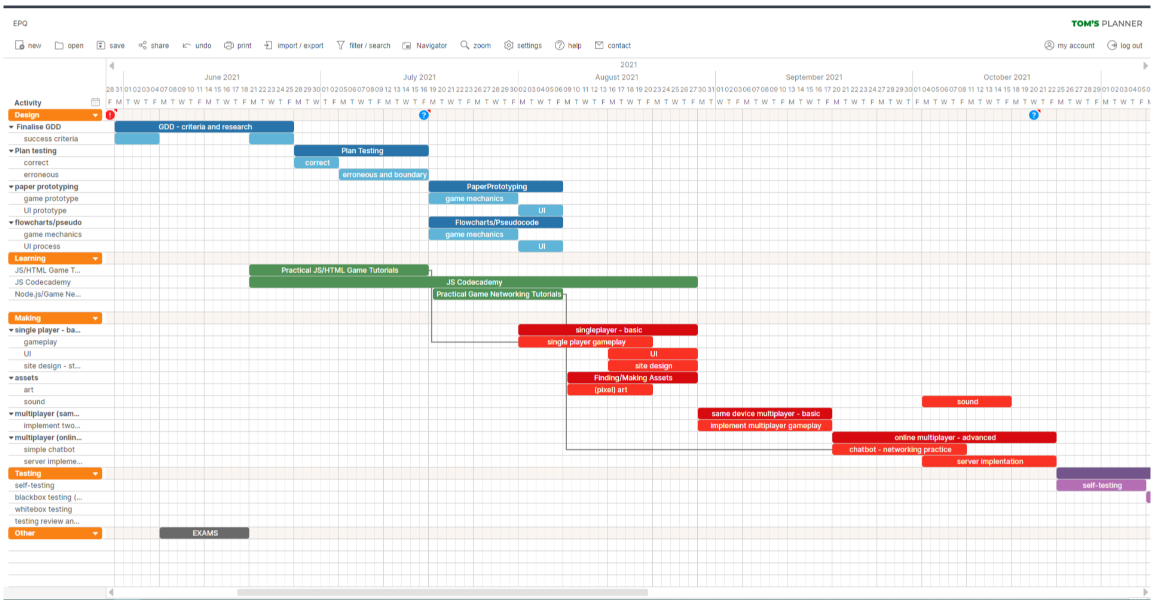
Version 1 – For my first draft I laid out every task according to the task breakdown and gave an estimated time of completion for each one.

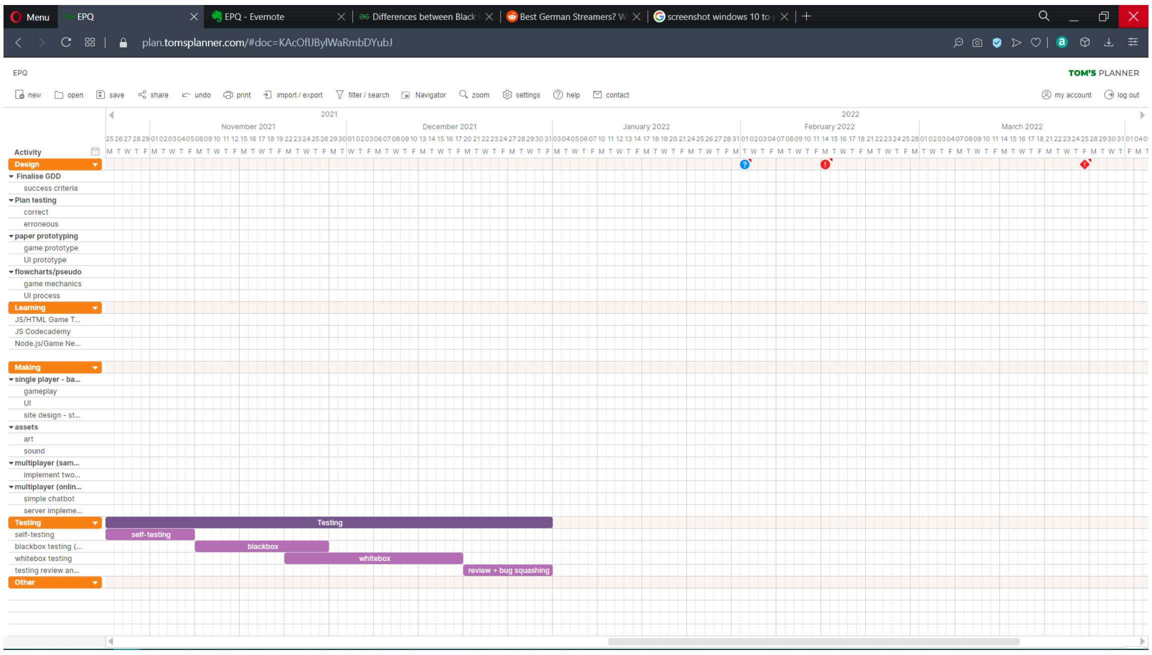


Version 2 – Taking feedback from my supervisor, I included more time for programming to account for potential delays.

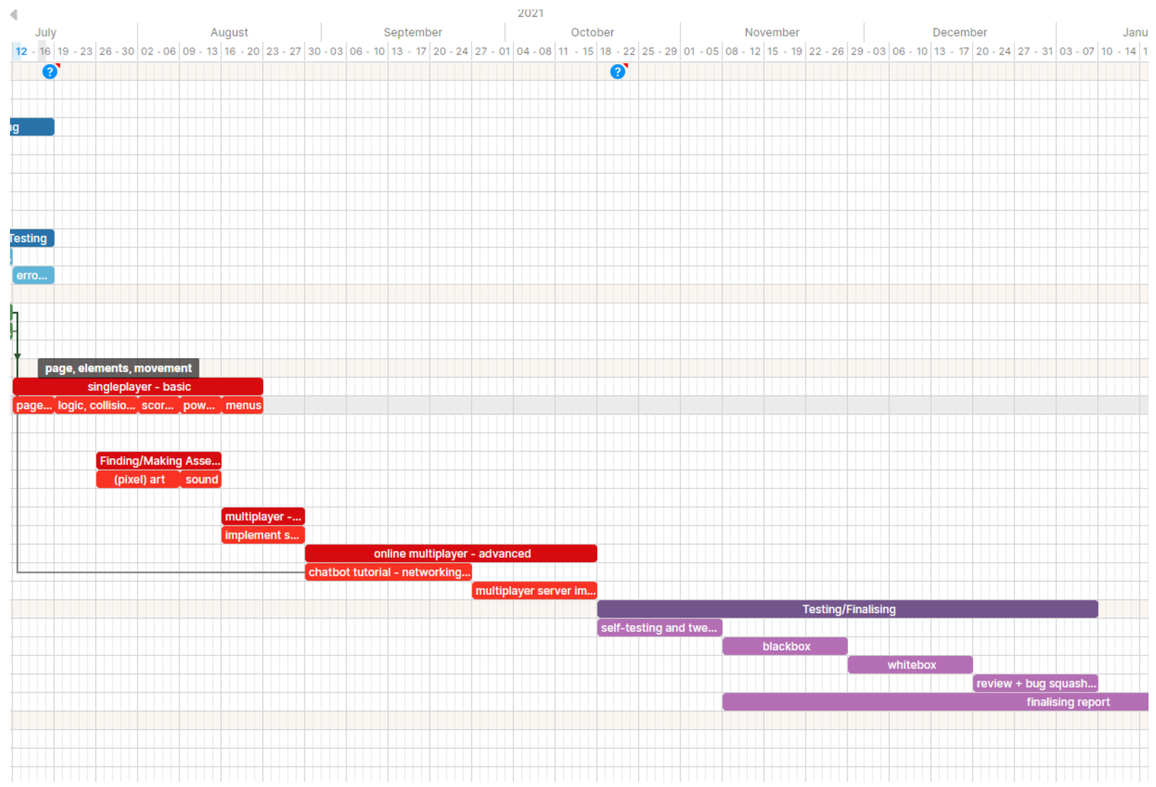


Version 3 – I switched to using Tom's Planner, as it allowed for scheduled activities to be broken up into specific tasks. I also included an extensive testing phase, during which I will examine each mechanic to ensure my game meets my success criteria.

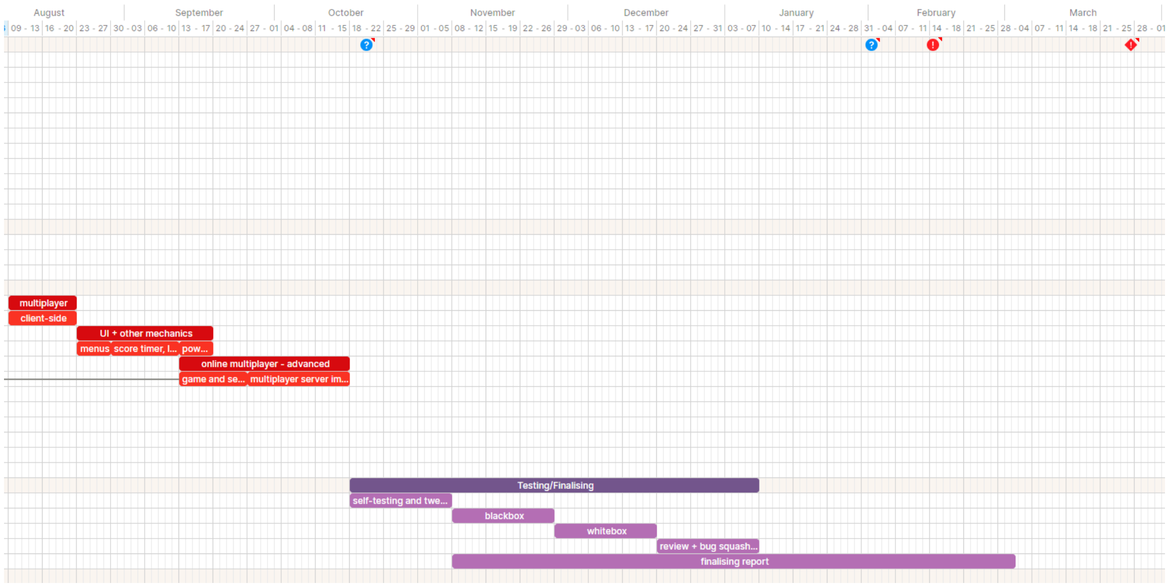




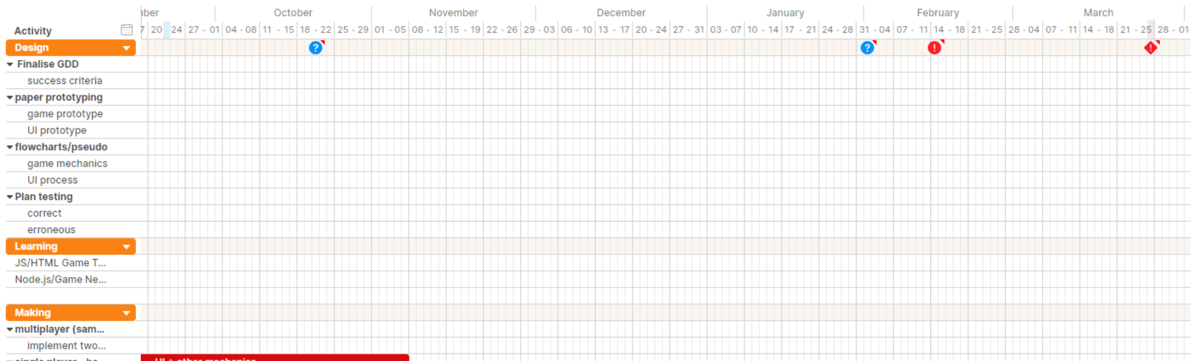
Version 4 – After programming practice games and learning the processes and tools required for a multiplayer server, I was able to further specify the programming section of my plan.



Version 5 – I struggled with getting the client-side code to work as expected when getting my programs online, so I re-ordered the networking tasks to allow more time for this.



Version 6 – Since the tagging mechanics were still buggy, I decided to simultaneously bugfix while getting the game on an online platform.



Diary

Date (week beginning Mon):	Activity	Summary	Reflections

10/05/2021	Researching Potential Topics	I have created a mind-map that displays various topics I am interested in and how they could be further studied in an EPQ. I chose to program a simple browser game in JavaScript and HTML, with the focus being on learning multiplayer networking using the libraries node.js and socket.io.	Some of my early ideas, namely the Chrome extension and educational animation, would be very complex and time consuming. I decided on an artefact that would be achievable by the deadline, while still allowing the development of new skills.
17/05/2021	Writing Success Criteria	I have written a list of success criteria for my project.	The main objective of my project is to create a browser-based multiplayer game.
24/05/2021	Researching Game Design	I have read example game design documents and articles on game project planning.	Although the focus on my project is the networking aspect and not the game design, learning how to design a game properly will help with initial planning and later documentation.
31/05/2021	EXAMS		
07/06/2021	EXAMS		
14/06/2021	Writing Game Design Document	I have written a simple game design document, outlining my game's objectives.	I have also identified my target audience and made my main objective more specific - the goal is to create a multi-user app that can be accessed from an online browser by any user.

21/06/2021	Filling VTT Form	I have filled out the Verification of Topic and Title form (VTT) and solidified the direction for my project.	I have decided on a simple "Tag" game to implement the server architecture to. I have also determined how I will measure the success of my project - if the moderator or anyone I give the link to can open the game, join the server, and see other players' movements in real-time I will consider my main objective complete.
	Preparing for Initial Planning Review	I have filled out the first section of the PPR with my project title and rationale.	
28/06/2021	Initial Planning Review	During this meeting, my supervisor advised me to ensure my goal wasn't vague and to make it clear in the project title. I was also advised to account for barriers to progress, for example a difficult-to-learn skill or a section of programming that takes longer than expected to complete.	Following this advice, I have created a more specific list of objectives, with identifiable and achievable goals that I can refer to as I progress through my project. My plan is flexible as the time limits for tasks are fairly long. When I reach steep learning curves that need more time, tasks can be easily modified and reordered in my plan. This will ensure that progress never stops when I hit roadblocks.

05/07/2021	Researching Time-Manage ment Software	I have looked at different software and decided to use the digital Gantt chart maker "TeamGantt" to manage my project and its deadlines.	Although it is difficult to read clearly at times, TeamGantt is helpful as it helps to visualise the planning process. I now feel much more confident in how I to effectively organise and document my project.
	Drafting Plan	I have continued to research and gather potential learning resources, while documenting and evaluating each one. Since I believe the networking aspect will be the most difficult, I plan to focus on that first.	As I began decomposing the process of making a game, I realised that there were some extra steps I had not considered before. For example, assets such as art and sound could be challenging to create from scratch, so I may need to license pre-made assets or utilise free and open-source resources online.
12/07/2021	Completing First Plan	I have completed an initial written plan for my entire project's process, and also listed the skills I will need to learn and develop.	I believe it will be easier to apply the network server code to pre-defined logic, so I decided to start with the game's logical design before moving on to the network implementation, instead of the reverse as I had originally planned.

	Putting Plan into Gantt Chart Software	I have created a digital plan using Tom's Planner.	I switched to using Tom's Planner because it had a simpler interface that I found easier to navigate than TeamGantt's. The task of planning months in advance was initially quite daunting, but after watching video guides and reading articles on project planning and management I feel more confident.
19/07/2021	Updating Plan and Game Design Document	I have updated my plan and specified how my game will appeal to my target audience.	I now have a clear target player, which will help to keep the design specific to them.
26/07/2021	Contacting professionals	I have contacted two industry professionals programming management advice.	Both of my contacts gave similar advice: the most vital step in the plan should be the creation of multiple prototypes in order to develop mechanics individually. Joe Williams also confirmed that gameplay mechanics are best implemented before art and assets are finalised, which I agreed with.
	Editing Plan and Objectives	I have reviewed and updated my plan based on the responses I received.	Following Joe Williams' advice, I have moved the prototyping section of my plan to earlier in the timeline. After this I can then create a detailed test plan, as I will have my prototypes as models to guide the evaluation process.

02/08/2021	Creating Practice JavaScript Projects	I have started following tutorials and experimenting to create 3 simple JavaScript games.	I have used JavaScript in the past for website building, so these tutorials were a great way to refresh my knowledge. They were also incredibly useful for teaching me how to format a webpage for a game. I initially thought that many components would be controlled via HTML, but I learned that it's possible and much more efficient to create and manage HTML elements using exclusively JavaScript.
09/08/2021	Continuing Projects and Refining Plan	I have continued the JavaScript games and updated my project plan.	I took notes while programming, which I then used to refine my plan. I also took notes on common mistakes or errors I made, so that I can easily find my previous solutions to them when programming my own game.
16/08/2021	Creating Layout Prototype	I have finished the practice games and moved on to programming the layout of my game.	I have structured my code with frequent comments to allow for easier maintenance later, although they clutter the code somewhat.

23/08/2021	Programming Movement	I have programmed the controls for one player using the arrow keys.	Initially the controls only allowed for vertical and horizontal movement, but after researching different arrow-key control schemes, I learned that this could be solved by adding all currently pressed keys to an array, so multiple directions can be specified at once.
	Programming Collisions	I have programmed the player-to-border collision mechanics.	Some of the systems I've devised, particularly the player collision mechanic, are very repetitive and tedious. My current method of checking for collisions involves looping through every object in the play area and checking whether it overlaps with any other object (this is a nested loop, so it has a time complexity of O(n^2).

06/09/2021	Programming Controls for Multiple Players	I have completed the setup and layout of the game and have begun adding two-player functionality.	I will start with only programming multiplayer for the same device, so that later I can test gameplay mechanics and the server code separately. While reading MDN's official JavaScript documentation, I discovered that different browser versions use different JavaScript codes to refer to each keyboard key. Since my success criteria states that my game should be playable on most modern browsers, I may need to allow the use of dynamic codes to account for multiple specifications.
	Completing Multiplayer Prototype	I have completed the first playable prototype for same-device multiplayer.	I learned that detecting collisions of circles is much easier than rectangles, as the only values that need to be checked are the circles' radii and their centre positions. I have changed the player objects' shapes and modified their component constructor to include a radius value. My initial code could definitely be improved, but the main gameplay mechanics are now working as intended.

	Improving Prototype	I have improved my prototype's code by writing functions to reduce repetition and improve maintainability.	To help with readability and navigation, I separated the main logic, canvas drawing and object constructors into separate JavaScript files. It would have been helpful to detail how these would be organised in my plan.
13/09/2021	Creating Client and Server Code	I have started networking by separating my program into client and server sections. The client code will be run by the players' browsers and will control the webpage setup, the game area display and the logic. The server code will manage movement and will ensure that each client can see the current position of other clients.	The server is currently only running locally, so for now only different browser sessions on my device are able to communicate. I know that most multiplayer servers rely on game hosting companies which can be expensive - so I will use one of the costless deployment options I have already learned of from *The Node.js Handbook*.
	Implementing Multiplayer Movement and Display	I have completed the server-side code for my game, and two clients can now successfully receive information about each other and display themselves on the canvas.	My game's tag mechanics are quite specific, so the official documentation hasn't been too helpful in fixing any bugs in it. I may have to move on to another section and return to bug fixing later if it requires more time.

20/09/2021	Completing Multiplayer	I have finished programming the client-server communications, and multiple clients can now successfully communicate with each other over a local network	With this I have completed the main objective of my project, in slightly less time than I had planned. I will now work on the game mechanics and ensure the app continues to be functional even if the mechanics don't work properly.
	Updating Plan	I have modified my plan by giving myself more time to try to fix the display bug. I have finished the prototype and started working on the server code earlier than intended, so I won't fall behind schedule. After this designated time, I will move onto the UI and menu, regardless of whether I have fixed the issue. I will return to work on it later during the testing and refining phase.	Since I didn't know how I would implement the server at the beginning of my project, the plan for this section was very vague. A detailed network design would have helped me organise the server development process and removed the time spent through trial and error. Despite this, I thoroughly enjoyed learning how to program this myself without any prior knowledge, and finding solutions to new problems. Now that I understand what is needed, I could design and develop another similar mechanic in much less time.
27/09/2021	Programming Individual Timing Mechanics	I have programmed the timing mechanics for each client.	I was able to make use of one of my practice games, Whack-A-Mole - I adapted the timer I used in that game to make one that counted in the other direction.

04/10/2021	Preparing for Mid-Project Review	I have filled out the next section of the PPR form and evaluated my current progress.	
11/10/2021	Mid-Project Review	I met with my supervisor to discuss my updated PPR.	I am on track to complete my project before the deadline.
18/10/2021	Programming Server Leaderboard	I have begun programming the server-side timing mechanics.	Following my plan, I have started on the timing and scoring after the movement and game mechanics. An improvement could be made to this by including the timing information when I first made the client-server communication, to prevent unnecessary backtracking.
25/10/2021	Continuing Leaderboard and Timing Mechanics	I have continued working on the timing and scoring mechanics.	I have encountered more difficulties with the server-side timing. None of these have been critical issues so my game still functions, however I will have to postpone the fixing of some of these issues to the testing and bug fixing phase.
	Updating Plan	I have updated my plan to include the scheduling changes.	
01/11/2021	Designing UI	I have begun designing the user interface.	I took a course on web design with CSS over the summer, so I was already familiar with how to set up a stylesheet.

08/11/2021	Creating Interface	I have created the visuals and interface.	I kept the visuals simple, as I want to keep the focus of my project on the networking and not the graphics.
15/11/2021	Continuing Bug fixing	I have continued fixing minor errors and improving the maintainability of my code.	
22/11/2021	Implementing Git	I have created a git repository for my project and continued to bugfix and improve the program.	Git will allow me to easily manage different versions and deploy the app.
	Deploying App	I have begun modifying the server-side code for deployment online, via the cloud-based Heroku platform.	I enjoyed experimenting with git and Heroku's CLI and learning useful functions.
29/11/2021	Completing Deployment	I have completed deployment of my game onto a public platform.	I enjoyed experimenting with git and Heroku's CLI and learning useful functions.
06/12/2021	Setting Up Documentatio n	I have created the layout for the final documentation.	I determined the section headers and their order based on guidance from the taught sessions.
13/12/2021	Researching Harvard Referencing	I have made notes on how to properly reference sources in various formats.	Due to the practical nature of my project, I have not directly cited many sources in my projects. However, my bibliography will contain all the sources I used to come to the finished product.

27/12/2021	Writing References in the Harvard Style	I have referenced every research source I used and added them to the bibliography section of the documentation.	I could not reference Stack Overflow as a reliable source, despite referring to it frequently during the programming process. This is because the responses are written by non-verifiable users, so are not guaranteed to be accurate.
03/01/2022	Updating Plan	I have modified the final stages of my plan.	I have now received my exam timetable, so the time for testing and bug fixing of the app will be pushed forward until afterwards.
10/01/2022	EXAMS		
17/01/2022	EXAMS		
24/01/2022	EXAMS		
31/02/2022	Testing and Bug fixing	I have made the final touches to the online app.	The app itself and the networking mechanics are all working as intended, and clearly demonstrate the attainment of my project's objective.
	Continuing Documentatio n	I have continued the documentation by including screenshots of my code.	It was difficult to decide how to show the evidence of my programs in the documentation, as I wanted it to be readable and easily understood by someone without the technical knowledge. I settled on showing screenshots of short sections code and providing a more detailed explanation in plain English alongside it.

07/02/2022	Planning Report	I have created a list of sections to cover in my report.	
	Drafting Report	I have written a draft for my report and overall evaluation.	
	Continuing Documentatio n	I have continued the documentation by updating the source evaluation sheets.	
14/02/2022	Preparing Presentation	I have created my presentation.	I chose to make this in the form of a digital A1 poster, which seamlessly details the entire journey of my project on one page. This way, the links between the different aspects of my project will be clearly illustrated to my audience. I will also provide a link to the app, so they can see my project's outcome for themselves.
28/02/2022	Presentation	I have conducted my presentation and received feedback from attendees.	I believe this event was very successful - My audience thoroughly enjoyed engaging with my project and trying out the app.
07/03/2022	Writing Evaluation and Report	I have written my presentation and whole project evaluations and updated the report.	
14/03/2022	Finalising Document Layout	I have continued the documentation by improving the page layout and ensuring everything reads smoothly.	

	Finishing Documentation for Submission	I have reviewed my project and made the final changes before finally printing and assembling my portfolio.	

PPR

Research Sources Evaluation

Name of Source:

The Node.js Handbook by Flavio Copes

Type/Format:

eBook

Summary and Relevance:

This source provided a well-rounded overview to Node.js and its uses. This was valuable as a first source, as from it I was able to gain a broader understanding of browser server-side code and how I could implement it.

Reliability and Supporting Material:

This book was written by Flavio Copes, who is known for writing JavaScript and web development tutorials. He cross-references X, particularly when Y, so his handbooks are well supported by other material

---

Name of Source:

The Express.js Handbook by Flavio Copes

Type/Format:

eBook

Summary and Relevance:

Following on from *The Node.js* Handbook, this source helped me determine the required software and libraries I would need for my project.

Reliability and Supporting Material:

This detailed only the required software, which was commonly used in most of the code examples I read in other sources.

Name of Source:

Node.js Notes for Professionals

Type/Format:

eBook

Summary and Relevance:

This source consisted mostly of practical examples on how to apply node.js to applications. As a beginner resource it wasn't very useful, because the code was quire complicated to understand without prior knowledge.

Reliability and Supporting Material:

The code examples are supported well by *The Node.js Handbook*, as the processes used to set up a node application in both sources are similar.

This eBook is a compilation of documentation written by users on the programming-focused question-and-answer community website, Stack Overflow. Many users may not be verifiable as industry professionals. Due to this and the fact that online sites are prone to misinformation, this source cannot be guaranteed to be accurate.

Name of Source:

Node JS Tutorial for Beginners by The Net Ninja

Type/Format:

Video Series

Summary and Relevance:

This was a useful beginner resource, as it explained the basics in great detail and highlighted all the software needed. It was also more recent and relevant as it was written in 2017. It was written in 2017, so it more recent and relevant compared to the older articles I read

Reliability and Supporting Material:

The channel is owned and managed by a single person. While they are very experienced and their tutorials line up with other sources, their own wording has not been cross-referenced so may contain errors.

Name of Source:

How to Build a Game Design Document by Dustin Tyler

Type/Format:

Article

Summary and Relevance:

While this source focused on documentation for team projects, it was useful to understand the format and how to define a game's scope.

Reliability and Supporting Material:

This was valuable to help me plan the specifics of my game

Name of Source:

Online Post Response from Joe Williams and Justin Nolan

Type/Format:

Communication with professionals via a GameDev forum

Summary and Relevance:

Question: How do you plan a game project from start to finish?

Joe Williams (thehumanidiot – developer of the indie videogame “Who’s Your Daddy?”):

“Prototype, prototype. prototype. Make something with minimal visuals to create a base ground for your core mechanics. Let others try out this prototype and see what feedback you can get.

Everything about game development should be kept pretty modular, meaning you can generally decide on the order yourself. Every game has different needs. That said, finalized art assets should be the last thing you worry about. Focus on getting gameplay mechanics down with prototyping, then write up a design doc of everything you want the game to have. Take sometime to write down a synopsis and the core themes of the game that you want to carry through development. Break it down into chapters/levels/areas and make a specific plan for each one as you get to it. Don't be afraid to let people test the game at each and every stage.

Good luck with your project!”

Justin Nolan (FakeByte – owner of the largest game server host in Australia and Asia Pacific):

“Try to build your game step by step. Your first version might contain a bit UI, a bit gameplay code and some other basic things. Then each version extend those basic systems farther. For single developer projects this is a much better approach then to build out system by system.”

Reliability and Supporting Material:

They both gave similar advice to program in a modular way, writing entire mechanics individually, and are also supported by the Game Document Article, as they advised laying out the basics in a document before starting programming. Joe Williams also suggested making multiple prototypes to ensure these mechanics line up with the game’s synopsis. This is similar to the approach used by Ania Kubów in her JavaScript game walkthroughs.

Name of Source:

Race'n'Chase Game Design by DMA Design Ltd

Type/Format:

Online Document

Summary and Relevance:

This source was an example game design document for the first *Grand Theft Auto* game. It gave me valuable insight into how professional companies design their games.

Reliability and Supporting Material:

This was supported by the article I read on how to write a game design document, as it was in a similar format.

Name of Source:

JavaScript Game Walkthroughs by Ania Kubów

Type/Format:

Video Series

Summary and Relevance:

I used these to practice JavaScript and become more familiar with the object-oriented approach to game programming.

Reliability and Supporting Material:

These walkthroughs were written by several experienced individuals from the free programming learning website "freeCodeCamp", so were reliable and also supported by the tutorials on W3Schools.

Name of Source:

W3Schools

Type/Format:

Website

Summary and Relevance:

This site contains interactive web development tutorials in various languages including JavaScript, HTML and CSS. I referred to this more when programming the mechanics, as it does not provide guides for writing server-side code.

Reliability and Supporting Material:

The tutorials here are written by experts with decades of professional web development experience, so are very reliable. The writers also constantly review their tutorials to avoid errors, so users can be confident in their accuracy.

Name of Source:

Mozilla Development Network (MDN) Web Docs

Type/Format:

Website

Summary and Relevance:

This was an excellent source for learning content and specific JavaScript guides. Solutions for different problems often contained multiple techniques, so it was useful in understanding the different methods I could use when programming.

Reliability and Supporting Material:

Though it is open-source, any changes made by contributors is checked and reviewed by multiple other users.

Practice Projects

As part of my research, I followed online guides and eBook tutorials to create the following simple browser-based games in JavaScript. I used them to learn of and practice the skills I would need.

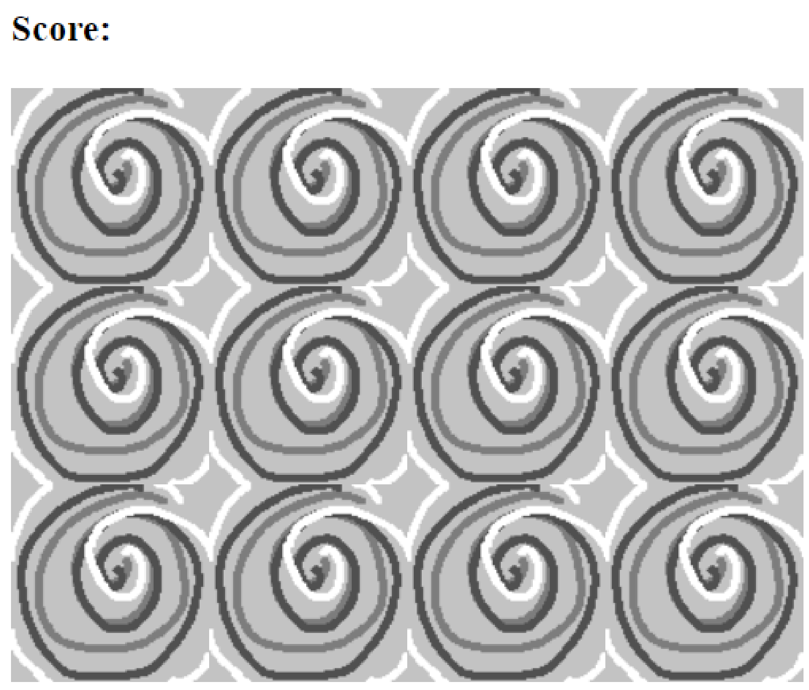
Memory Game:

Project Overview:

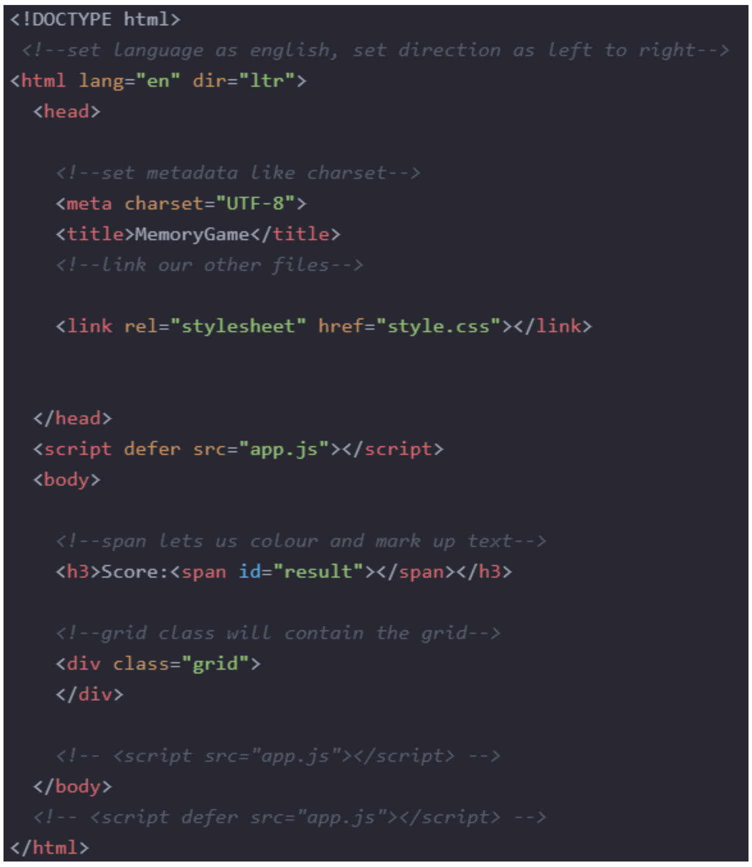
First, I created a simple picture-based memory game. A 4x3 grid of tiles is displayed to the player, who can click on a tile using the cursor to reveal the image on the other side. The player may flip two cards in this way, and if they are a matching pair, they disappear, and the user scores a point. If they do not match, then they are hidden, and the player may guess again. The game ends after all the matching pairs have been found, after which a final results page is shown to the player.

Programming Process:

I began by making the visuals, including drawing the pictures themselves and designing the layout of the webpage. Using Windows 10's built-in image editor *Paint,* I drew six simple animal pictures and a card back, each being a 100px-by-100px PNG file.

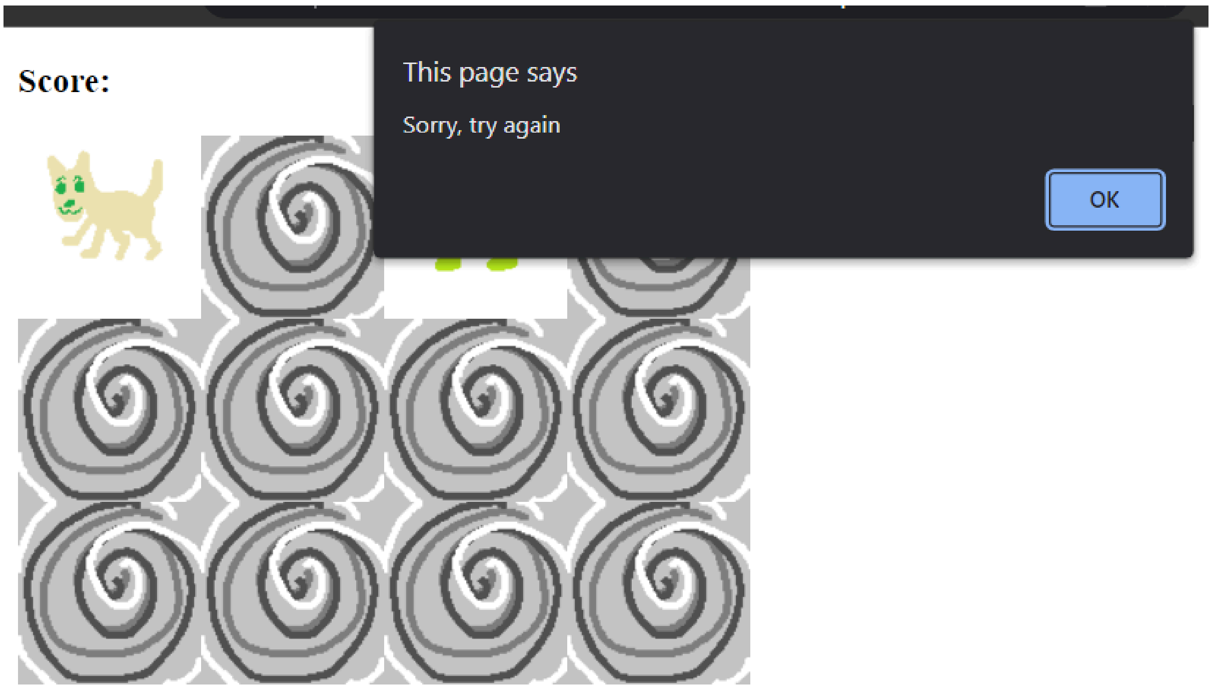


I then added the pictures to the project directory and created an array containing two copies of each image, so the player could pair them up. I decided I would program the game's mechanics in the app.js file rather than the index.html file, so that the index file would contain just the page's layout and no logic. This would allow elements like the cards grid to be flexible and easy to modify.



```
<!DOCTYPE html>
 <!--set language as english, set direction as left to right-->
<html lang="en" dir="ltr">
  <head>

    <!--set metadata like charset-->
    <meta charset="UTF-8">
    <title>MemoryGame</title>
    <!--link our other files-->

    <link rel="stylesheet" href="style.css"></link>


  </head>
  <script defer src="app.js"></script>
  <body>

    <!--span lets us colour and mark up text-->
    <h3>Score:<span id="result"></span></h3>

    <!--grid class will contain the grid-->
    <div class="grid">
    </div>

    <!-- <script src="app.js"></script> -->
  </body>
  <!-- <script defer src="app.js"></script> -->
</html>
```

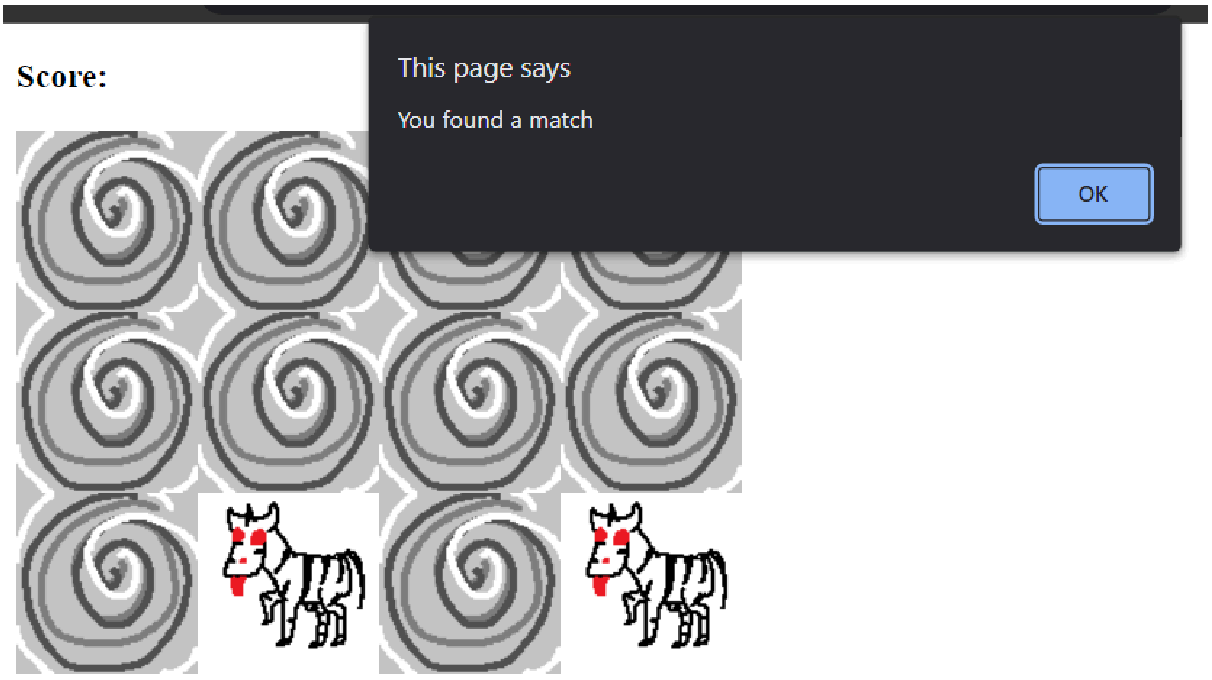
To communicate the game's state to the player, I used the alert() method to give them a browser pop-up:



Score:

This page says

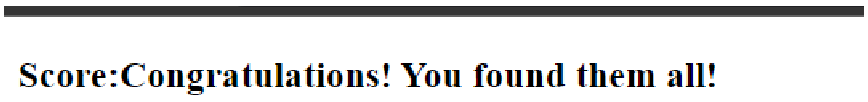
Sorry, try again


Score:

This page says

You found a match

At the end of the game, a congratulations message is displayed next to the score at the top of the screen.


Score:Congratulations! You found them all!

What I learned from this project:

I didn't actually use the score and it wouldn't have been useful, as every player would have the same score once they finish the game anyway. However, including this allowed me to learn how to format persistent text, which I used later to create a point counter.

I also learned how to use event handlers in JavaScript to wait for a specific input from the user, for example a click on an element or a keypress.

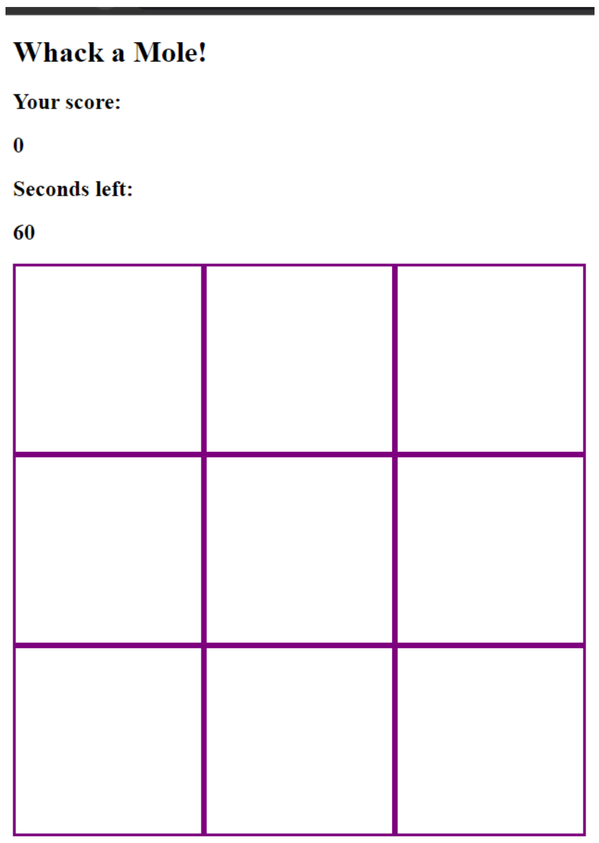
Whack-A-Mole:

Project Overview:

Next, I created a replica of the carnival game whack-a-mole. A 3x3 grid and a 60-second timer are displayed to the user. An image (the 'mole') is displayed in one of these squares and moves to another random square after a set interval (e.g., 1 second). The player can click on a square when the image is displayed on it to gain a point. After the time is up, the player's final score is displayed.

Programming Process:

I first created the visuals and the webpage's layout, like before. Since I had set up the HTML before, I knew exactly what elements I needed so was able to create a starting layout much quicker than before.



Whack a Mole!

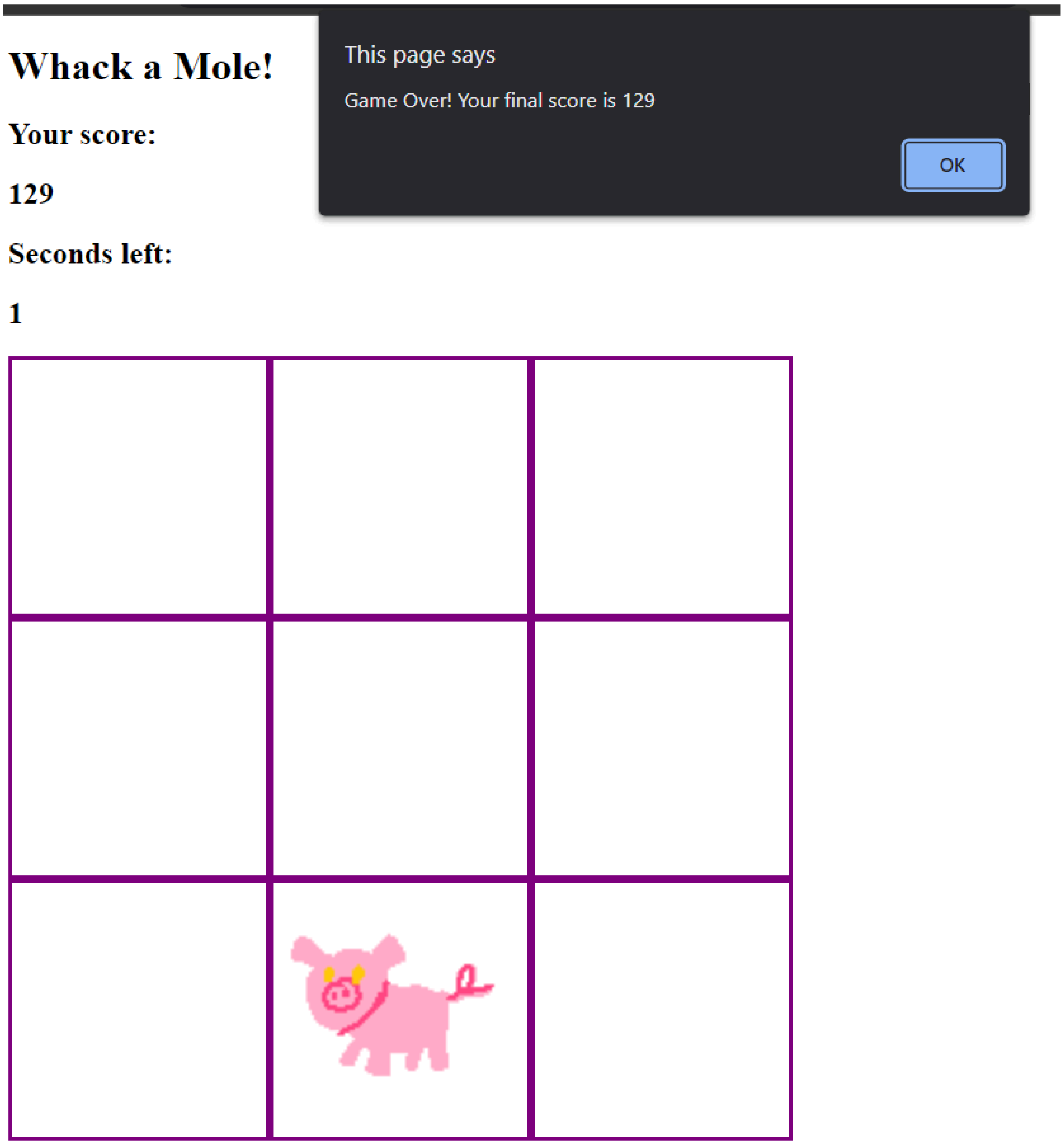
Your score:

0

Seconds left:

60

For the 'mole' I reused an earlier picture. The logic for this game was similar to the memory game but slightly more complicated - each click on the image gives points to the player, and after the time is up the final score is shown via a pop-up message.



Whack a Mole!

Your score:

129

Seconds left:

1

This page says

Game Over! Your final score is 129

To implement the scoring mechanic, I added an eventListener method to each grid square, that waited for the player to click a square, then gave them points if the clicked grid's ID was the same as the moving image's ID.

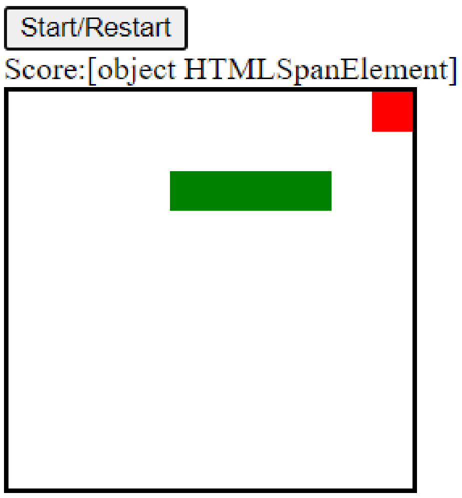
What I learned from this project:

I learned how to implement timing mechanics into a JavaScript program and display a countdown using HTML. I also learned that a "start game" button would be useful, so the timer doesn't immediately begin when the page is opened.

Snake:

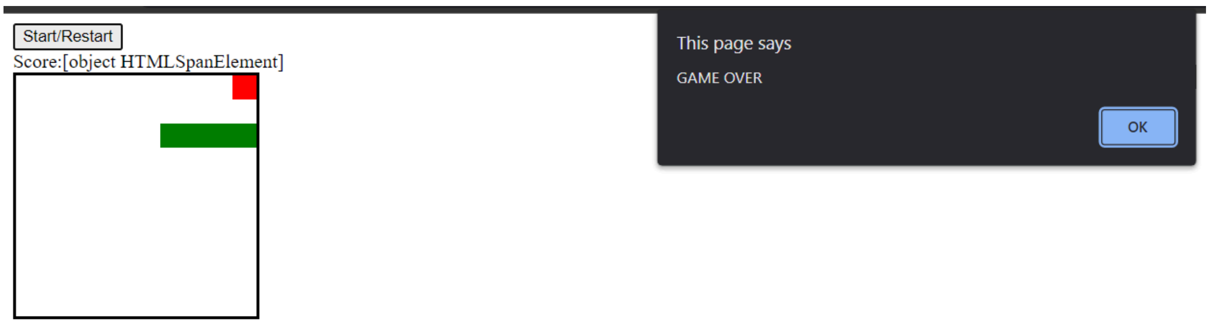
Project Overview:

A blank canvas is displayed until the user clicks the start button.



```
Score:[object HTMLSpanElement]
```

When the user hits an edge, the game ends with an alert message displayed to the player, and can be restarted with the same button.



```
Score:[object HTMLSpanElement]
```

This page says

GAME OVER

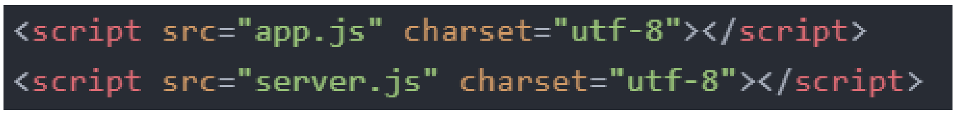
What I learned from this project:

I learned how to program endless gameplay that ends after a specific condition has been met. I also learned how to include a starting menu and allow the player to begin the game manually.

Programming Process

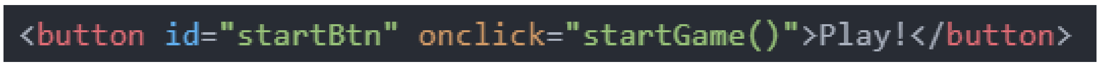
Webpage Layout

First, I created the page's layout in HTML. This included references to the external JavaScript files, app.js and server.js:



```html
<script src="app.js" charset="utf-8"></script>
<script src="server.js" charset="utf-8"></script>
```

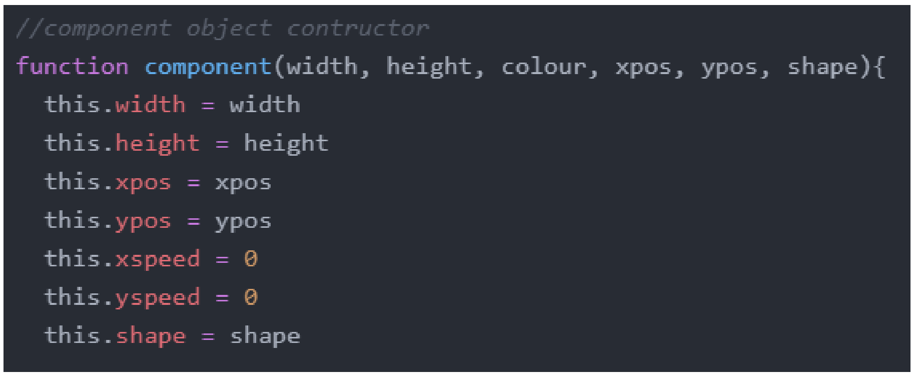
I wrote the initial function startGame(), which is triggered when the user clicks on the HTML button element with id 'startBtn':



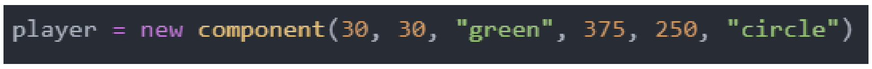
```html
<button id="startBtn" onclick="startGame()">Play!</button>
```

Object Constructors

Next, I created the constructor for the player object. Each component holds attributes that are given values on creation, including the player's x and y coordinates and their shape dimensions:



```javascript
//component object contructor
function component(width, height, colour, xpos, ypos, shape){
  this.width = width
  this.height = height
  this.xpos = xpos
  this.ypos = ypos
  this.xspeed = 0
  this.yspeed = 0
  this.shape = shape
```



```javascript
player = new component(30, 30, "green", 375, 250, "circle")
```

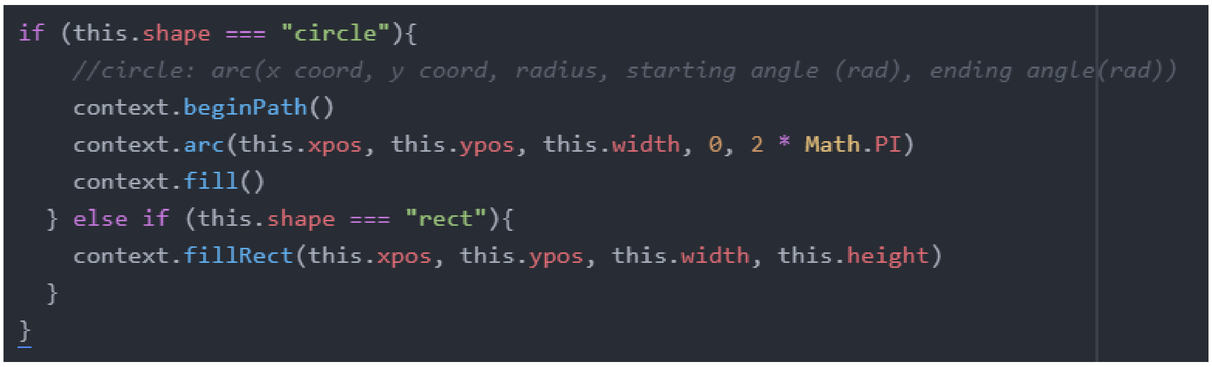
(an example component being created)

Display and Canvas Updates

From MDN's Documentation, an arc can be drawn in HTML via JavaScript by using the context.arc() function, which "draws an arc which is centred at (x, y) position with radius *r* starting at *startAngle* and ending at *endAngle*" (Mozilla, 2022). I used this to draw a circle by making the starting angle 0 and the final angle 2π radians.

A rectangle can easily be drawn by passing in the object's width and height to context.fillRect().

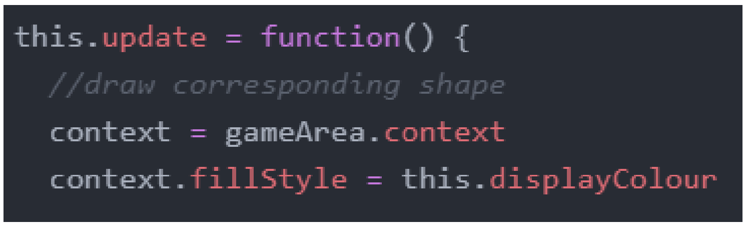
The following code within the constructor draws the object to the canvas based on the 'shape' attribute:



```
if (this.shape === "circle"){
    //circle: arc(x coord, y coord, radius, starting angle (rad), ending angle(rad))
    context.beginPath()
    context.arc(this.xpos, this.ypos, this.width, 0, 2 * Math.PI)
    context.fill()
  } else if (this.shape === "rect"){
    context.fillRect(this.xpos, this.ypos, this.width, this.height)
  }
}
```

I then created an updateGameArea() function that clears the canvas and re-draws all the components in their new positions if they have moved. This function is executed every 20 milliseconds (0.02 seconds).

The component objects also contain an update() method that is called when updateGameArea() is called, and draws itself to the canvas:

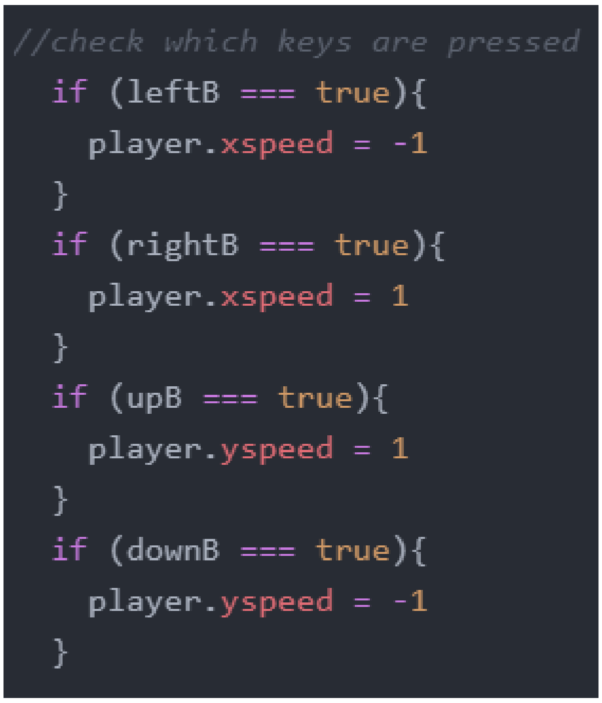


```
this.update = function() {
  //draw corresponding shape
  context = gameArea.context
  context.fillStyle = this.displayColour
```

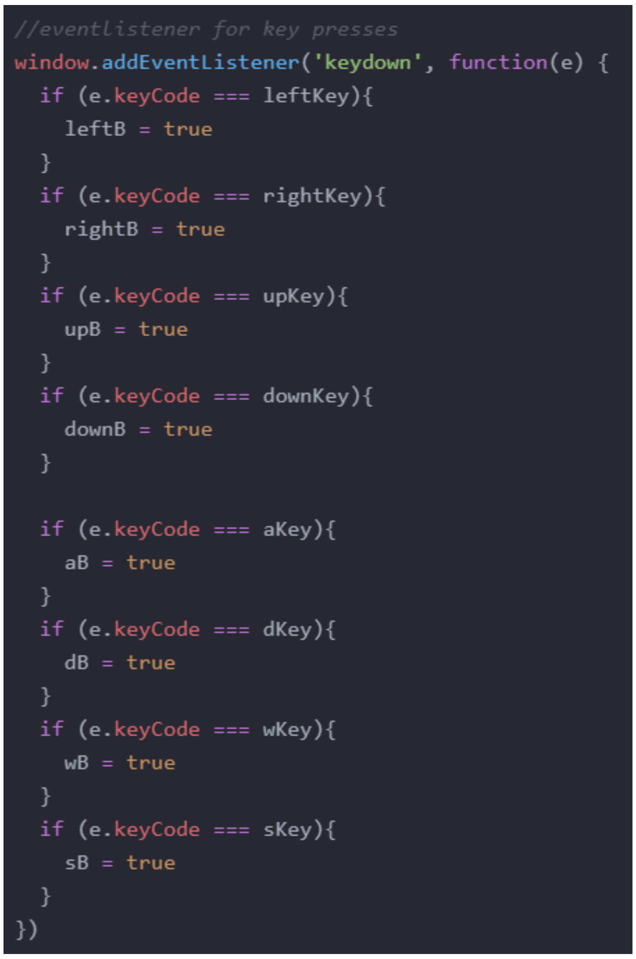
As shown in the object constructor above, each component also contains xspeed and yspeed properties. These will be updated by the player's key presses and will change the xpos and ypos attributes accordingly.

Evaluating User Input

For this prototype, I created controls for multiple players on the same device, so one player moves using the arrow keys and the other uses the WASD keys. I added an eventListener to the gameArea object that listens for key presses and sets the key's corresponding variable to true. This is then checked every game update (20 milliseconds) to change each player's xspeed and yspeed:


```
//check which keys are pressed
  if (leftB === true){
    player.xspeed = -1
  }
  if (rightB === true){
    player.xspeed = 1
  }
  if (upB === true){
    player.yspeed = 1
  }
  if (downB === true){
    player.yspeed = -1
  }
```

However, this resulted in stuttering, orthogonal-only movement. I found a solution from a video guide – the key variables should instead be booleans that are set to true for as long as the key is pressed (Dobervich, 2018). This allowed for multiple directions to be specified at once so players could move smoothly and diagonally:

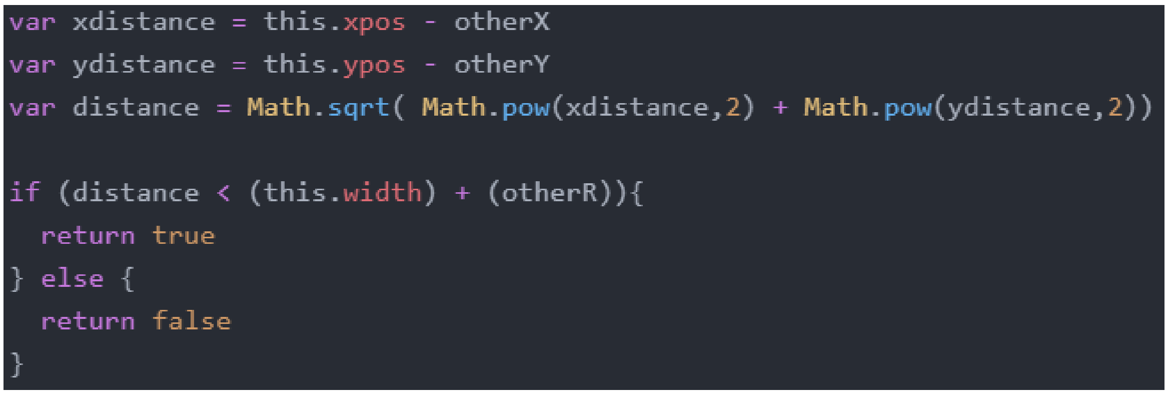


```
//eventlistener for key presses
window.addEventListener('keydown', function(e) {
  if (e.keyCode === leftKey){
    leftB = true
  }
  if (e.keyCode === rightKey){
    rightB = true
  }
  if (e.keyCode === upKey){
    upB = true
  }
  if (e.keyCode === downKey){
    downB = true
  }

  if (e.keyCode === aKey){
    aB = true
  }
  if (e.keyCode === dKey){
    dB = true
  }
  if (e.keyCode === wKey){
    wB = true
  }
  if (e.keyCode === sKey){
    sB = true
  }
})
```

Collision Mechanics

In the player constructor I included the option to draw objects as either circles or squares. I then learned it would be easier to calculate overlapping objects if they were all the same shape, so I decided to only let player object be circles. After consulting MDN's documentation again, I learned of the mathematical method for determining whether two circles are overlapping. We can check if two circles are not overlapping by "taking the centre points of the two circles and ensuring the distance between the centre points are less than the two radii added together." (Mozilla, 2022)

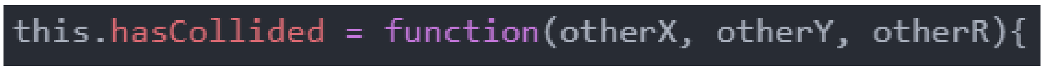
i.e., A collision is detected if the distance between the two circles' centre points is less than the sum of their radii. The distance between the two points can be easily calculated using Pythagoras' Theorem. This is implemented as follows:



```
var xdistance = this.xpos - otherX
var ydistance = this.ypos - otherY
var distance = Math.sqrt( Math.pow(xdistance,2) + Math.pow(ydistance,2))

if (distance < (this.width) + (otherR)){
  return true
} else {
  return false
}
```

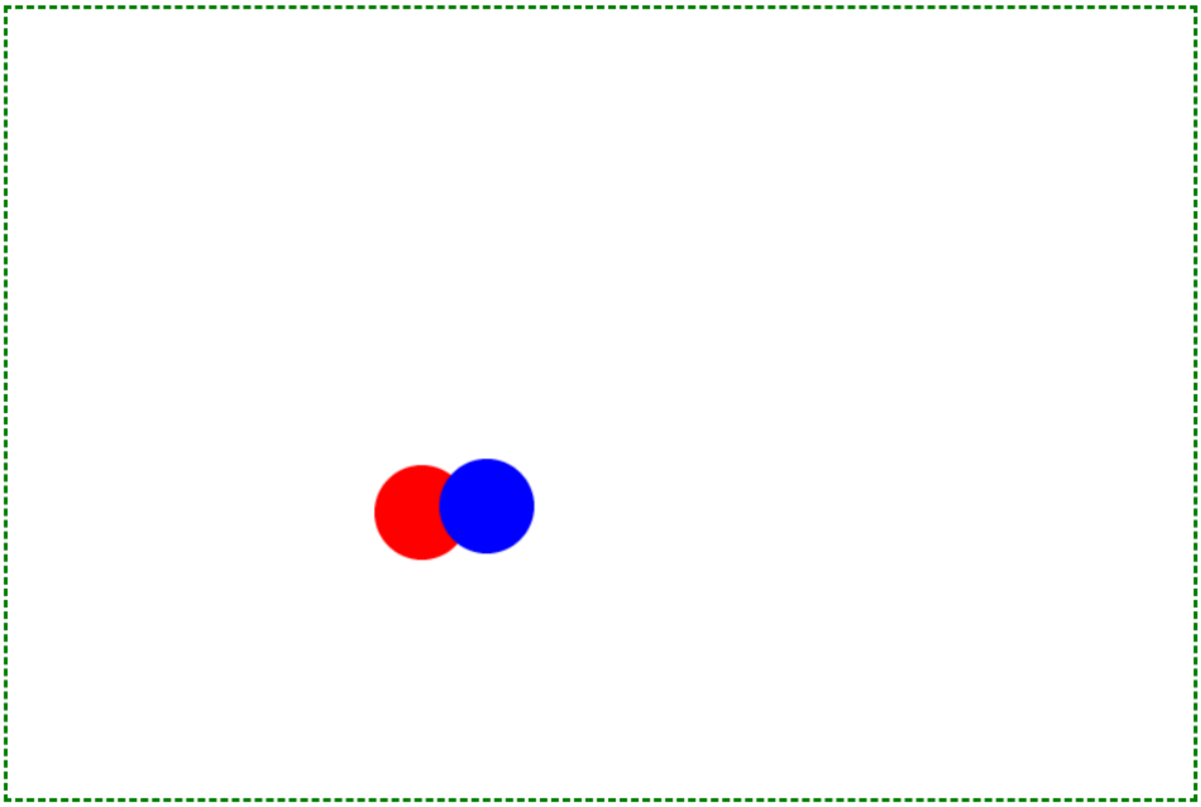
Initially I made the common error of mistaking the diameter for the radius. This caused collision to only be detected when players were past the centre point of each other. To tidy up my code and make the width attribute less ambiguous, I removed the extra height attribute from the player object, as I had no need for a square's dimensions anymore.

I created a method in the player class called hasCollided(), which takes in another object's x and y positions and its radius. It then runs the calculations above and returns true or false depending on the result:



```
this.hasCollided = function(otherX, otherY, otherR){
```

To test that collision detection was working correctly, I set the player to change to a different colour (blue) for each frame that they are in collision with another player.



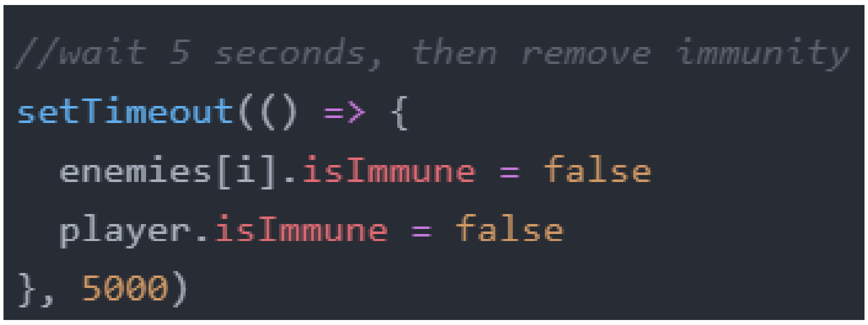
Collision detection was working as intended, so I moved on to programming the game's tagging mechanics.

Tagging Mechanics

This mechanic only required the collision to be working, so implementing it was as simple as adding an isTagged boolean attribute to the components. This resulted in both players flickering between being tagged and not tagged every 20ms. I had anticipated this, so had already planned to include an 'immunity' period.

Once a player tags another player, they are immune to being tagged for 5 seconds. This prevents the flickering display and also makes the game fair.

I added an isImmune boolean attribute to the components, which is set to true whenever a player tags someone or is tagged themselves. This is then set to false again after 5000 milliseconds (5 seconds) have passed:



```
//wait 5 seconds, then remove immunity
setTimeout(() => {
  enemies[i].isImmune = false
  player.isImmune = false
}, 5000)
```

I now had a working game prototype that allowed two players to play a game of tag on the same machine. Using this as a foundation will make the networking section easier, as I will already have the mechanics laid out.

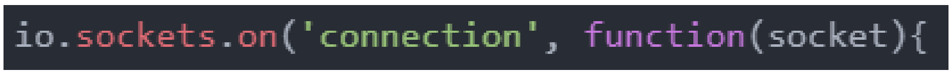
Improving Prototype

Next, I worked on optimising the prototype. I separated the main program from the player component to improve readability, and also organised some repeated sections of code into easier-to-read functions.

Server Setup

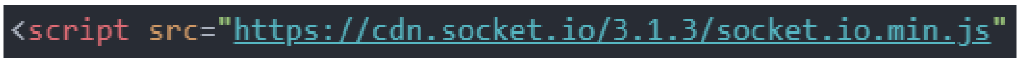
I started programming the client-server communications locally, where my computer acted as the virtual server. To do this, I had the server listen on port 3000, so that clients from multiple sessions on the same browser can access the same instance of the app.

First I created a function to handle new connections. Each new client is assigned a socket ID that is used by the server to identify them:



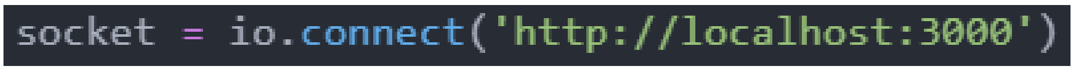
```
io.sockets.on('connection', function(socket){
```

The socket.io library needs to be imported into the clients' HTML code for them to communicate with the server in real-time. I did this by providing a script element with a link to the content delivery network (or CDN):



```
<script src="https://cdn.socket.io/3.1.3/socket.io.min.js"
```

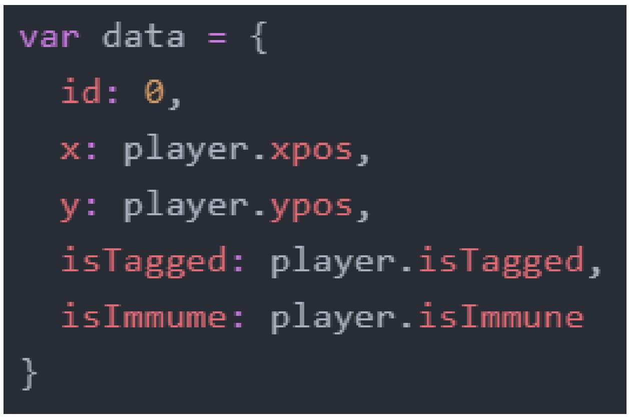
The client also needs to listen on port 3000, so I included this in the setup function (startGame):



```
socket = io.connect('http://localhost:3000')
```

Communication

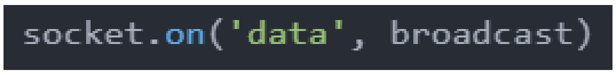
Next, I had to decide exactly what information would be needed by the server. I started by sending a message named "data", which contained the client's x and y positions, their ID, and whether or not they were tagged or immune:



```
var data = {
  id: 0,
  x: player.xpos,
  y: player.ypos,
  isTagged: player.isTagged,
  isImmume: player.isImmune
}
```

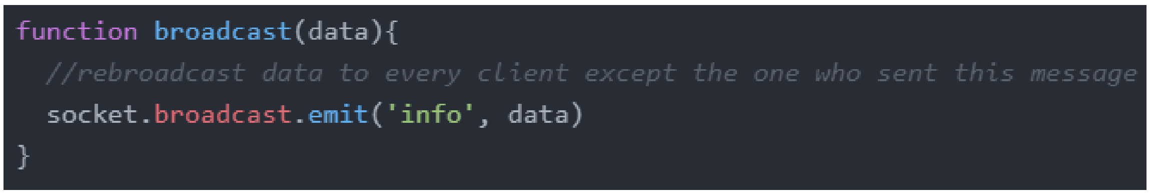
This was included in the gameUpdate function, so the message was also transmitted every 20 milliseconds.

I decided to put the game's mechanics and logic onto the server, so that one client's latency won't affect the other players. I added another socket.on function to the server, this time wating to receive the message named "data":



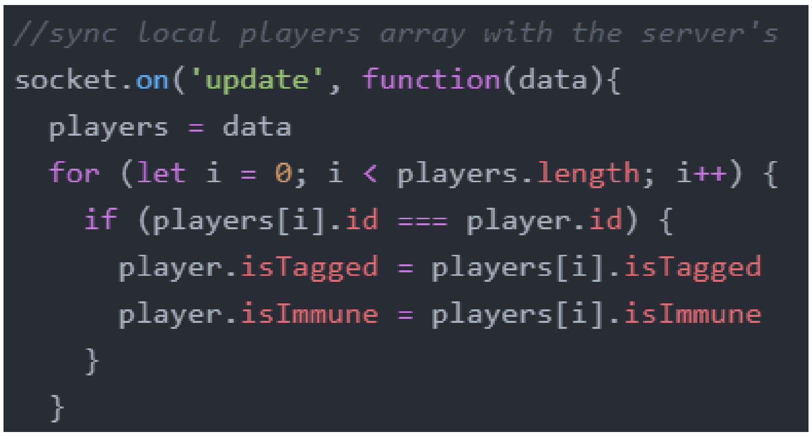
```
socket.on('data', broadcast)
```

The server updates its own array of players, then rebroadcasts its contents to all clients in a message called "info". Socket.broadcast.emit sends the message to all connected clients except the sender (Socket.IO, 2022):



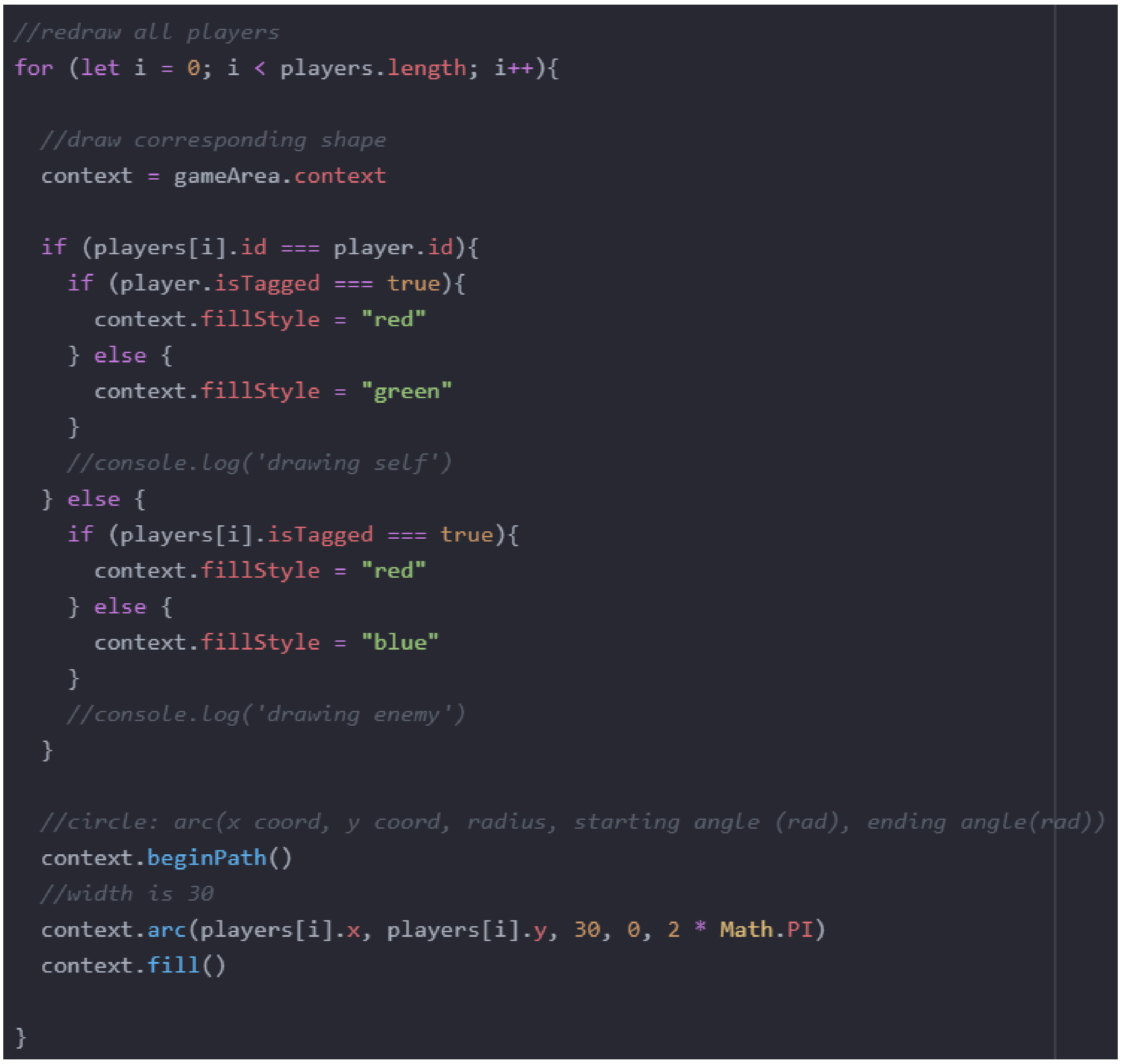
```
function broadcast(data){
  //rebroadcast data to every client except the one who sent this message
  socket.broadcast.emit('info', data)
}
```

On receiving this message, clients then update their own array of players:



```
//sync local players array with the server's
socket.on('update', function(data){
  players = data
  for (let i = 0; i < players.length; i++) {
    if (players[i].id === player.id) {
      player.isTagged = players[i].isTagged
      player.isImmune = players[i].isImmune
    }
  }
```

I modified the gameUpdate function to also draw each object in this array, so the most recent update is continuously drawn to the canvas:

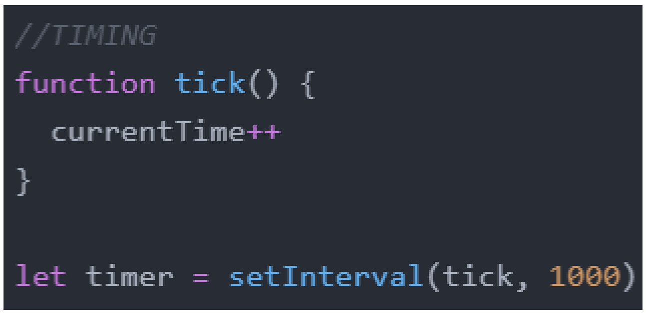


```
//redraw all players
for (let i = 0; i < players.length; i++){

  //draw corresponding shape
  context = gameArea.context

  if (players[i].id === player.id){
    if (player.isTagged === true){
      context.fillStyle = "red"
    } else {
      context.fillStyle = "green"
    }
    //console.log('drawing self')
  } else {
    if (players[i].isTagged === true){
      context.fillStyle = "red"
    } else {
      context.fillStyle = "blue"
    }
    //console.log('drawing enemy')
  }

  //circle: arc(x coord, y coord, radius, starting angle (rad), ending angle(rad))
  context.beginPath()
  //width is 30
  context.arc(players[i].x, players[i].y, 30, 0, 2 * Math.PI)
  context.fill()

}
```

By having the server manage all of the clients' changes, there exists only one correct instance of the game, so any inconsistencies between clients can be easily corrected by receiving an update from the server.

Timing Mechanics

I added a "tick" function to the server that increments an integer by 1 every 1000 milliseconds:



```
//TIMING
function tick() {
  currentTime++
}

let timer = setInterval(tick, 1000)
```

I then included this in the update message to the clients, who would then update their individual timer:

```
socket.on('tick', function(data){
  currentTime++
})
```

This functioned as expected, however it also resulted in excessive messages being sent to clients and a decline in maintainability. Instead, I chose to have clients manage their own timer, and then notify the server of any changes with each update:

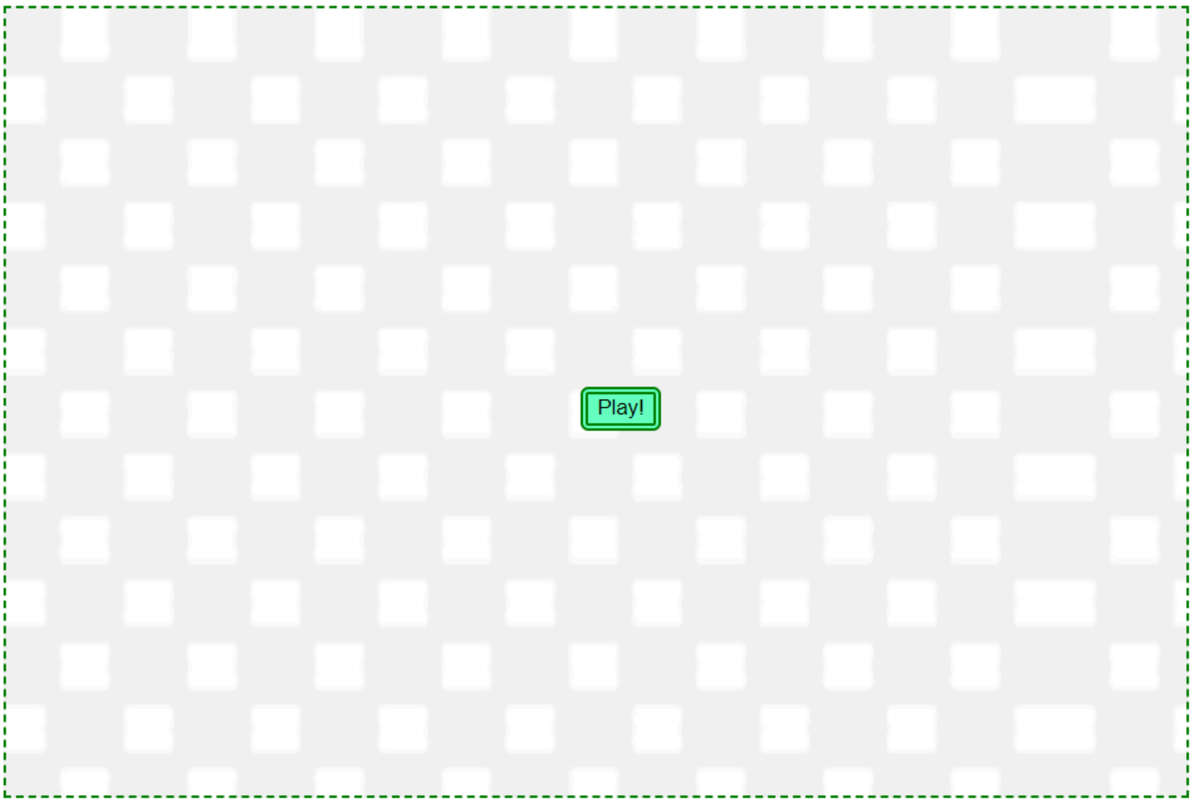
```
//message to send to server
var data = {
  id: player.id,
  x: player.xpos,
  y: player.ypos,
  isTagged: player.isTagged,
  isImmune: player.isImmune,
  time: currentTime
}
```

There were still problems and bugs with the leader board that I would have liked more time to work on, however these mechanics were secondary and not the main focus of my project.

User Interface

I created a simple grid background using the digital paint software *Krita*. I chose to use a grid as it allows the relative positions of players to be easily seen without cluttering the screen.

I also added a start button that runs the setup function startGame, so that clients can enter the session manually:

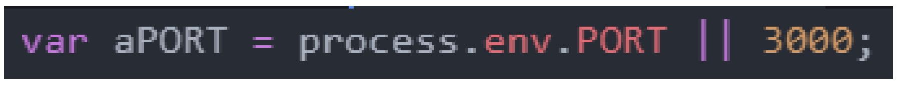


Deployment

I chose to use the cloud platform Heroku with Git for deployment as it provides a free and easily manageable host for node.js apps. (Copes, 2018)

Another option I considered was Glitch, but it doesn't allow for custom domains, which was important as I wanted my project to be easily accessible.

I accessed and pushed changes to Heroku via the command line. I then had to modify the server so it listened on the port given to it by Heroku, or otherwise default to port 3000 again:



```
var aPORT = process.env.PORT || 3000;
```

By using Git, I had access to a centralised version control system. Any changes I made to the root directory could be uploaded safely and update my app on Heroku

I had now completed the goal of my project by publishing a multi-user app than can be accessed by anyone online. I continued to update and make minor improvements to the code through the rest of my project.

Presentation & Feedback Sheets

The format I chose for my presentation was a digital A1 poster illustrating the entire journey of my project in one document. I believe this was effective as it demonstrated the links between each area of my project in a visually pleasing display.

As part of the presentation, I included a link to my project's website. This was very effective as it provided enjoyable engagement for my audience and allowed them to try the app's features for themselves.

I think the presentation could have been improved by talking my audience through my project's process in more detail. I mostly relied on visuals and general descriptions of the software I used, but I think my audience would have appreciated some more insight into my technical solution. I also could have made it easier for them to access the site by providing a QR Code.

Evaluation

Overall, my project was a success, and I learned how to practically apply networking to a JavaScript app. I also learned how to design and implement full-duplex communications between clients and a server from scratch. My game itself is simple and not very appealing, and the main game mechanics are bugged and don't always work as intended. The app also tends to slow down or freeze whenever a new client joins. However, the main objective of publishing an app has been met as communication between clients and the server is otherwise functional. By making these I developed my programming and problem-solving skills, which I could apply to the practical sections of A Level Computer Science.

My time management, however, could have been improved. Because I had so little knowledge on the topic before starting, my initial plans were vague and poorly scheduled. For example, in my first Gantt chart I had planned to finish all the JavaScript programming within a month, after only a few weeks of preparation. After researching and programming some smaller projects of my own, I realised that this was unrealistic and modified my plan to allow sufficient time.

Links to Higher Education - **Personal Statement:**

For my EPQ I decided to make a multiplayer browser-based game using JavaScript and HTML. I chose to write this ambitious project in two languages I am not very familiar with, so I could challenge myself to gain experience in programming languages I hadn't learned at school. I self-learned the skills needed to create a number of small games in preparation for my more complex project, including whack-a-mole and snake. I also taught myself networking using socket.io and node.js. I was able to use a variety of resources, including video tutorials, articles and Mozilla's documentation to learn how to use these libraries and apply them to my game. As I had to plan my project over the course of 10 months, I also developed my planning and project management skills and my ability to organize events over an extended period.
In my Computer Science NEA, I have been able to apply the skills learnt from my EPQ. I am in the process of designing a procedurally generated dungeon game using Unity with C#. With this I have been able to improve my C# proficiency beyond what I have learned in the classroom and gain valuable practice for the programming section of my exams. Learning to use Unity has been very rewarding, as I've been able to practically apply C# to a usable product.
I participated in CyberFirst's online Discovery Program, which involved solving a variety of difficult yet engaging cyber security activities. These included finding weaknesses in fictional webpages to access private information, modifying web server requests and solving computer science related problems by modifying a site's JavaScript code. I performed well in the first stage and was invited to CyberFirst Essentials. With this I had access to hours of advanced cyber security and computer science courses.
I frequently complete programming challenges like Project Euler tasks online. These allow me to keep my fluency and confidence in multiple languages and challenge myself. I take advantage of the numerous free courses and tutorials online: I have completed a 300-hour course on Responsive Web Design on freecodecamp.com. This course solidified the basics of HTML and helped me get more familiar with common tags and how to structure a webpage, which I then used in designing the page in my EPQ. Another valuable lesson I learned from this course was accessible web design – structuring web pages and content in a way that is comprehensible to people using screen readers or other assistive technologies.

Over the summer I volunteered at a children's summer camp as an activity leader, which demonstrated my ability to work in a high stress environment and manage a team. Through this I also developed my communication skills: as part of a problem-solving activity, I was able to phrase the rather complex logic in a way that even the youngest children could understand. This has contributed to my understanding of why writing maintainable code is important. I also work as a computer science tutor for primary school students. I have been able to help children develop their programming skills and share my love of the subject.
In the summer school I attended, I was tasked with creating a design brief for a game in under 24 hours. I was able to apply my digital art skills to create a draft of the game's visuals and interface. I learned how to design for and focus on a specific audience, a skill which I used in my NEA to give my project a clearer direction. I also followed a 3D Modelling course, which taught me the basics of using Blender. I participated in the foreign language public speaking competition 'Have Your Say', performing a skit in German to a large audience. I was chosen to represent my school in the East Anglia regional finals, coming in 2nd place overall. I enjoyed the opportunity to develop my confidence and articulation skills.
I am well prepared to face the challenge of university and am eager to apply the knowledge and experience I will gain, while working towards my goal of being a games programmer.

Report

My initial map of project ideas covered a range of subjects and use cases. As a student studying Maths, Physics and Computer Science I knew I wanted to create something technical that could showcase my ability. A practical piece of software such as an Anki addon or Google Chrome extension would have been impressive, but I also had to consider how I would realise the product before the deadline in less than a year. A more art and design focused product like an educational comic or an animated CPU simulator would have been fun to create, but I wanted to challenge myself to learn a new skill rather than focusing on pre-existing strengths.

I chose to focus on networking so I could gain practical programming experience while learning how the multiplayer mechanics in my favourite games are implemented. I also wanted to learn how to program real-time communication between clients and a server and to apply this to a simple game. Due to its pre-defined rules and mechanics, I chose to make a version of the playground game "Tag". This would allow me to focus on developing my programming skills without worrying about complex game features. I decided against using a game engine such as Unity or Unreal Engine, as they would have distanced me from the source code and prevented me from fully understanding the logic behind my final product.

I started my research by reading a number of eBooks on node.js and networking with JavaScript. These were interesting and informative, and *The Node.js Handbook* even detailed the libraries and software I would need to install. Some of these sources went into more advanced networking techniques that weren't relevant, so I only focused on the libraries that I would need.

I also researched how to properly Harvard reference in my documentation. Due to the practical nature of my project, I did not have many citations, but my bibliography was extensive as I referred to various material to come to my final product.

I found article guides helpful as well, and took notes on their processes to formulate my own plan. Videos and official documentation were my most used resources during this project. Many of my sources, such as MDN's documentation, provided code examples that helped me understand new functions or syntax in context. I frequently referred to the forum-like site StackOverflow as well, however since users are not verified and anyone can answer a question, information from it isn't guaranteed to be accurate or reliable.

I used digital Gantt charts and timelines to plan and manage my project. First, I listed all of the required jobs and then broke them into individual tasks. Inputting them to the software was simple, and I then scheduled them so my project would be complete by the deadline. I found these apps very effective as they were easy to modify and reorder to make new versions of the plan as I researched and progressed. I made several iterations of my plan, as my understanding of the programming process improved as I progressed, and I also had to schedule tasks around important events such as exams. I should have been more strict with following the plan, however. A lot of times I deviated from it and worked on different mechanics at different times. This may be due to the fact that my early plans were vague and uninformed, but in future I would stick more reliably to my written plan.

The 3 practice projects were a significant step in learning to create an app in JavaScript from scratch. They helped me become more familiar with webpage layout using HTML. Through them I also developed my confidence in JavaScript, which is a programming language that I had only used a few times. I enjoyed making *Snake* in particular – I was able to move away from the tutorial and work out how to program some of the mechanics myself.

Programming the app was a very enjoyable and rewarding experience. I made multiple prototypes for one player, two players on the same device and several players online. As I did these, I took notes on errors I made and exactly what I did to solve them, to prevent the same mistakes being repeated in my final product. I then used the prototypes as a foundation to deploy my app on the cloud-based platform Heroku. As I programmed, I kept a written log, which detailed every step I took so that I could keep track of where I was when returning to my code. I thoroughly enjoyed implementing the client and server messages, and felt a great sense of achievement when each session was finally able to communicate successfully.

If I were to program a game like this again, I would definitely create class diagrams and structure charts as part of the planning process. I now know what objects, attributes and methods are needed, so I would be able to arrange and implement these more efficiently. I would also write a better Game Design Document that detailed how specific mechanics would work and how graphics would be affected and displayed, so the final product could be more visually appealing.

I know that many computing courses at university start from the beginning and so don't require any previous programming knowledge. My existing coding ability and the skills learned in this project have given me a significant advantage regarding my university applications. My planned university courses were Computer Science for Games, and Computer Games Technology, and I have since received offers for both of them.

The realisation of this project has been an enjoyable and valuable experience that will be useful when doing coursework and other projects at university in the future.

References

Copes F. (2018) *The Node.js Handbook* [Online] Available at: https://flaviocopes.com/page/ebooks/ (Accessed: 29/06/21)

Dobervich D. (2018) *Processing - respond to multiple key presses.* Available at: https://www.youtube.com/watch?v=yKv02lq7JHs (Accessed: 25/09/21)

Mozilla Corporation (2022) *Mozilla Development Network Web Docs.* Available at: https://developer.mozilla.org/en-US/ (Accessed: 01/08/21)

Socket.IO (2022) *Socket.io Docs,* Version 4. Available at: https://socket.io/docs/v4/ (Accessed: 12/09/21)

Bibliography

Copes F. (2018) *The Express.js Handbook* [Online] Available at: https://flaviocopes.com/page/ebooks/ (Accessed: 29/06/21)

Copes F. (2018) *The Node.js Handbook* [Online] Available at: https://flaviocopes.com/page/ebooks/ (Accessed: 29/06/21)

DMA Design Ltd. (1995) *Race n' Chase Game Design*, Version 1.05. Available at: https://www.gamedevs.org/uploads/grand-theft-auto.pdf (Accessed: 25/05/21)

Dobervich D. (2018) *Processing - respond to multiple key presses.* Available at: https://www.youtube.com/watch?v=yKv02lq7JHs (Accessed: 25/09/21)

Kalambay J. (2018) *Deploy your first App with Heroku and Node.js.* 21 July. Available at: https://www.youtube.com/watch?v=MxfxiR8TVNU&t=404s (Accessed: 27/09/21)

Kubów A. (2020) *Build NOKIA 3310 Snake using JavaScript in under 15 minutes.* Available at: https://www.youtube.com/watch?v=rui2tRRVtc0 (Accessed: 09/08/21)

Kubów A. (2020) *Make Memory Game in JavaScript, HTML and CSS for your portfolio.* Available at: https://www.youtube.com/watch?v=tjyDOHzKN0w& (Accessed: 01/08/21)

Kubów A. (2021) *Whac-a-mole in JavaScript (super simple!).* Available at: https://www.youtube.com/watch?v=rJU3tHLgb_c (Accessed: 04/08/21)

Mozilla Corporation (2022) *Mozilla Development Network Web Docs.* Available at: https://developer.mozilla.org/en-US/ (Accessed: 01/08/21)

Refsnes (2022) *W3Schools.* Available at: https://www.w3schools.com/ (Accessed: 01/08/21)

Socket.IO (2022) *Socket.io Docs,* Version 4. Available at: https://socket.io/docs/v4/ (Accessed: 12/09/21)

The Coding Train (2016) *Agar.io - Part 2 (Networking with Socket.IO and Node.js.* 4 September. Available at: https://www.youtube.com/watch?v=ZjVyKXp9hec (Accessed: 26/07/21)

The Net Ninja (2016) *Node JS Tutorial for Beginners.* 25 May. Available at: https://www.youtube.com/watch?v=ZjVyKXp9hec (Accessed: 10/05/21)

Tyler D. (2021) *How to Build Your First Game Design Document.* Available at: https://www.gamedesigning.org/learn/game-design-document/ (Accessed: 24/05/21)