

Iterative Development

Iteration 1 - *Date August - October*

Aims for this iteration

The main aim is to create a working prototype for a system to solve a simpler problem that can be tackled in a similar way to chess.

I intend to create some code to run as a server and some code to run in the user's browser. They must be able to communicate using some form of API.

This prototype should act as a stepping stone to help me tackle the larger problem of chess.

It should use a simple version of minimax with alpha beta pruning to determine the best move in tic tac toe. This is over-engineering for tic tac toe as a simple set of 'if-then' rules would suffice. However if I use the approach of the British Museum Algorithm to make a working version of minimax, then this will help me with chess in future.

Another main aim is to create a simple skeleton architecture that can be built upon. This involves:

- using python to write the code for the webserver
- writing the JavaScript code to validate input and manage the game client side
- Using a REST API or WebSocket to create a connection between the frontend and the server.

In terms of the aims of the prototype from general perspective:

The prototype will have only one page/window and will allow the user to play tic tac toe vs the computer. This is a simpler problem to tackle as tic tac toe is a solvable game. This means that there is a best strategy that leads to a draw or a win and it can be determined. I will go into more specifics on why I have chosen tic tac toe and why it is a simpler problem to tackle in the research section.

The primary goal of this prototype is not to perfect the frontend or to perfect the backend for my final chess program. Instead It will focus on developing the backend and the connection between the frontend and backend. Multiple prototypes will provide a stepping stone toward the completion of each element of the project. In this case the primary goal

will be met so long as the user can play noughts and crosses, even if the user interface is not the most easy to use. This should be a test that I have successfully implemented the webserver, minimax algorithms and the connection between the frontend and backend.

With regards for the macro plan form my prototypes:

The aim for prototype 2 is to create a working interface and game engine to play a simplified version of chess where there are only pawns. I will then add in other pieces and the idea of checkmate in subsequent iterations. I will use as many prototypes as need be to get to full chess. I will then refine by adding a database and difficulty levels. I will then add additional features if I have time.

Functionality that the prototype will have

In terms of the prototypes functionality:

- It should be a single webpage that consists of a tic tac toe board, a reset button as well as a title and some text.
- Text should prompt the user that it is there go
- The user should then be able to click a square to put a cross 'X' there.
- The computer should then respond (near instantaneously) by putting a naught 'O' in another square
- The user should then be able to play the game out until a draw or loss (or in theory win).
- Then an alert should appear to explain how the game has ended
- The user should be able to reset the game at any point to play again

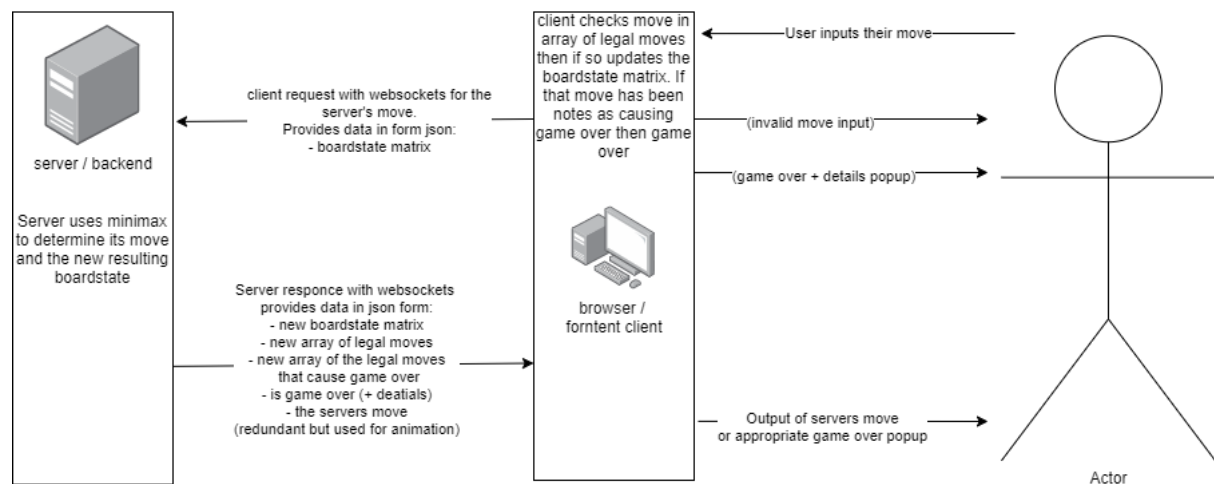
I recognise that either I will fail to correctly implement the minimax function, in which case I will not meet all my technical goals for this prototype, or the user will never be able to win. This shouldn't be a problem with chess as the computer cannot play perfectly and there will be different difficulty settings available.

I believe that the added difficulty of making the computer play sub optimally is greater than the benefit to the user experience in this prototype if the user can win. This is after all only a prototype and the user experience is not its primary goal. This sounds silly but my end product isn't tic tac toe so refinement of the user experience would experience diminished returns.

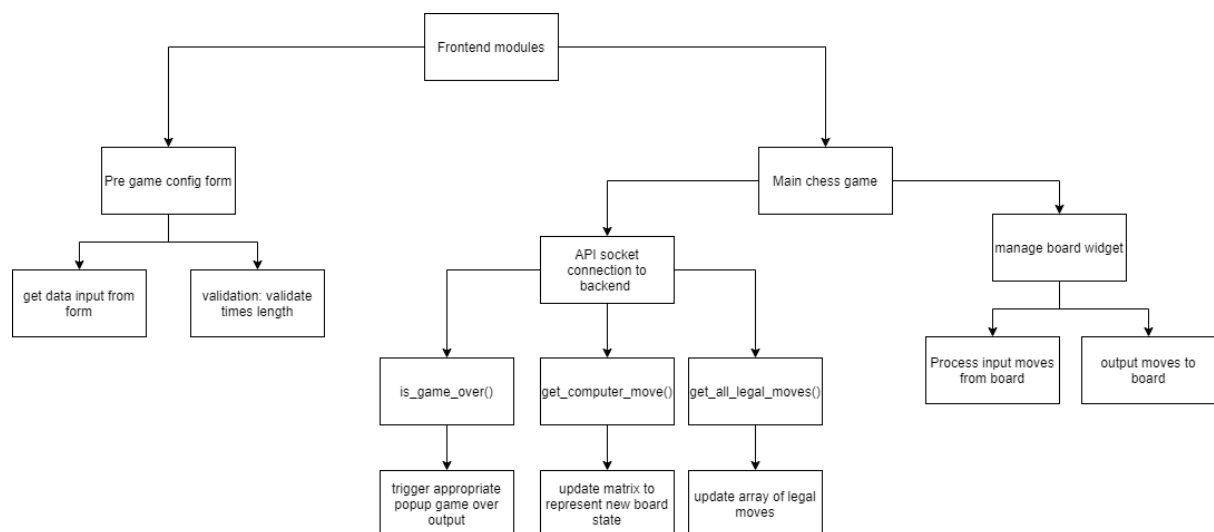
Due to the simplicity of this game compared with chess, not all of the modules, in my plan for the final product, are needed / applicable. I have only in a loose sense,

created a portion of my modules for the front and backend. Instead tic tac toe has some different modules that are simpler.

From by overall plan this iteration includes the following functionality



The API connection is similar. The main difference is that the frontend determines if a move is legal on its own



Of the frontend module plan, I have created the get computer move function and the board widget (a simpler version) but not the rest. The API connection has been simplified and the menu wasn't needed for this prototype

However the functions developed on the backend (including the api connection) are the bulk of this project. I have created:

- simpler version of the minimax function (no static evaluation)
- a simple game state class with child generator and other methods
- I have created a WSGI webserver that host the webpage (simpler as only one webpage)
- I have created the socket connection (simpler as less data transmitted between)

Annotated code screenshots with description

First I will show screenshots of the code with explanation beyond comments when relevant (but comments are extensive). Then I will show screenshots of it working

Code Editor:

I used visual studio code for the editor and IDE. I didn't 'set it up' as I used my normal programming configuration. The relevant extensions I am using in vs code include:

- Microsoft IntelliCode
- Microsoft Jupyter
- Microsoft Python
- Sourcery
- Ventur
- VUE language features

These extensions are mostly for syntax highlighting, 'intelli-sence' and jupyter notebook compatibility.

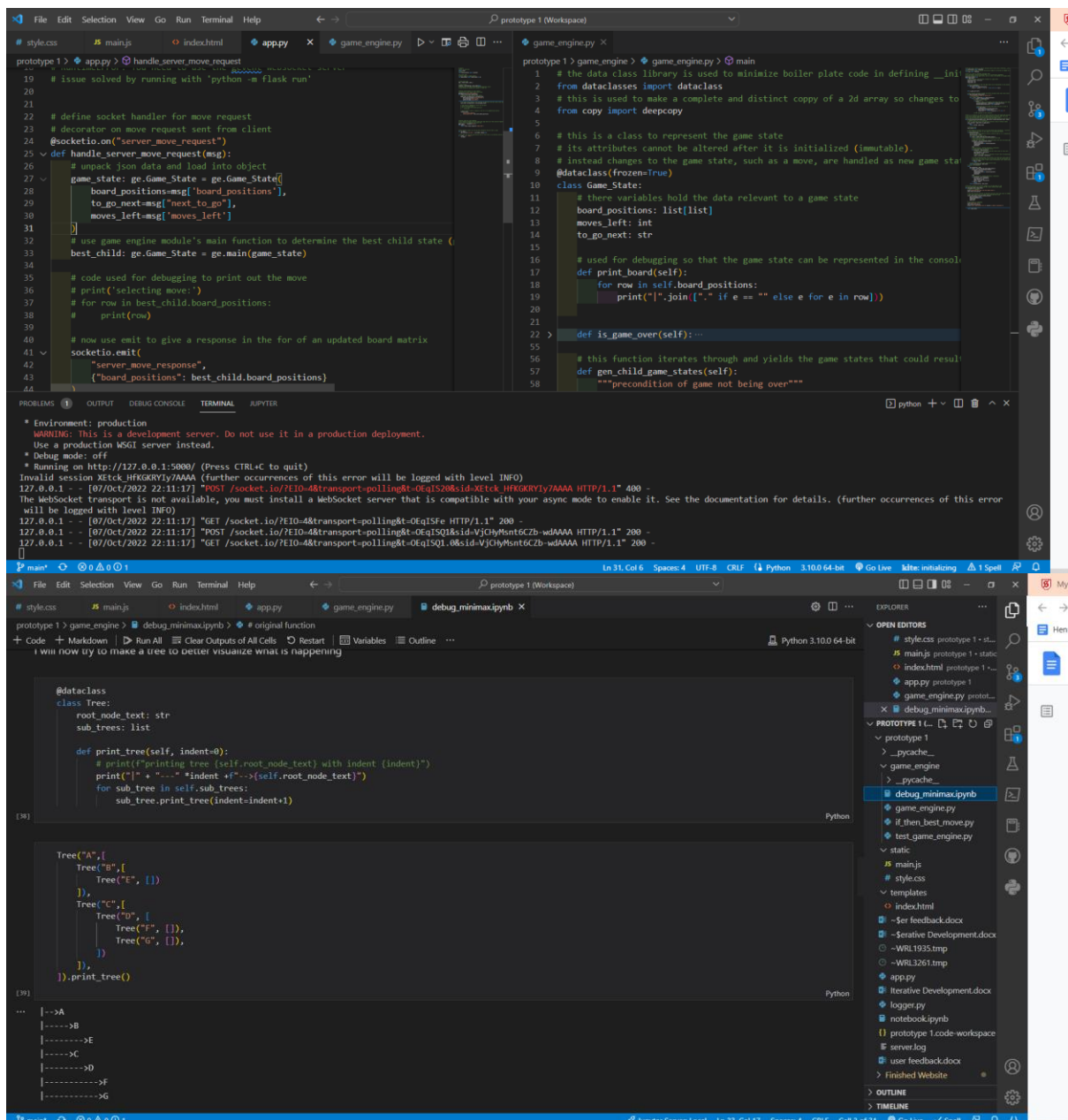
I used the console to run my program mostly but made some use of breakpoints for debugging.

My main tool in debugging was jupyter notebook which allowed me to import the function from a python file I was developing and see how the functions behaved with sample inputs.

The reason I used Visual Studio Code is that it was the editor I was familiar with and had already configured. Over time I had tried other editors like 'Notepad ++' and 'Atom' but I preferred VS code. This was for the following reasons:

- It provides easy navigation of a folder full of files by allowing you to access them with the keyboard only and have multiple windows at once
- It allow you to run your code using the integrated PowerShell terminal (which I rely on as I later discover that I need to use the terminal to run my code successfully)
- It has integration with git which I will try to setup for version 2 to have some version control
- It has a myriad of different available extensions to help with language syntax
- It has direct support for python and jupyter notebooks unlike other editors.
- All settings and keyboard shortcuts can be changed.

Below are some screenshot examples of me using VS Code.



Libraries, Languages and other assets:

I decided to use python and JavaScript for this project as I am already familiar with both. I know how to code many more complex technical tasks in python that I will need to complete in later prototypes. For example maintaining a SQL database. I also think that there is no real alternative to JavaScript for client side logic and validation. Therefore these languages were familiar and a good fit for the project, so I choose to use them.

For this iteration I use very few libraries and external assets. I only used VS Code for the editor and only used code I had written. No elements of my solution (including the html and UI) were produced using a forms designer.

I used libraries minimally, here are all the libraries I used:

Python:

- Flask: used to create a WSGI HTTP web server
- Flask_socketio: used to create a WebSocket server to connect the front and backend

- Dataclasses: used to avoid boiler plate code (which can be quite a lot) in class definitions for the `__init__` and `__eq__` (comparison) method. It also easily allowed me to make my classes immutable
- Copy: I used the `deepcopy` method to make a duplicate copy of a 2 dimensional array in python. Without it changes to the original array change the new copy as they use the same memory location

JavaScript:

- `Socket.io.js`: used to allow the client side JavaScript to connect and communicate with the `socket.io` server

WSGI stands for Web Server Gateway Interface. It allow the python framework that I am using (flask) to give instructions in a standardised way specialist webserver software like waitress. I can still run the program locally for debugging without this software in development mode. When I am finished I will need to use Waitress as an additional piece of necessary software to deploy my project.

I decided to use WebSocket to make my API over a traditional REST approach using HTTP. This is because WebSocket creates a full duplex TCP/IP connection. This leads to lower latency (allowing for more complex validation to be server side without delays for the user) and it allows for messages to be 'broadcast' to all clients or just 'emitted' to one. This is important if I want to allow for a leader board or player vs player matches in future.

The code:

Starting with the HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- standard metadata to help browsers -->
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Tic Tac Toe</title>

  <!-- import javascript -->
  <!-- <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js" integrity="sha512-hVXv7iRa0qZWZpSrfI32tWYG60wEgAyHuK/fMV502zySt1u9PqFAu75X36qvK+0I9/8p2rvkjx8932pm" crossorigin="anonymous"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.1/socket.io.js" integrity="sha512-wsL4T62BjF0p53p+3q6B1zDzO/HgOxM5KpbT5q4kW+hG/KFsWYICGp8MVNIDu76/W0eKTK1A7Yfpq" crossorigin="anonymous"></script>

  <!-- load in the javascript from a url that the server will determine as the html is served -->
  <script src="{{ url_for('static', filename='main.js') }}"></script>

  <!-- import css styles -->
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>|...
</body>
</html>
```

Here is my head tag. It contains boilerplate code to provide browsers with basic metadata. It also fetches the WebSocket library which is how the frontend and the backend will communicate. A CSS file and JavaScript file are also loaded in from the server.

```

<body>
  <!-- i used the center tag to make the CSS simpler -->
  <center>
    <h1>Tic Tac Toe</h1>
    <h2>You go first:</h2>
    <!-- this table will be the 3x3 board -->
    <table id="board">
      <tbody>
        <tr>
          <!-- each square has an ID so I can see which one is clicked -->
          <td id="sq_11"></td>
          <td id="sq_12"></td>
          <td id="sq_13"></td>
        </tr>
        <tr>
          <td id="sq_21"></td>
          <td id="sq_22"></td>
          <td id="sq_23"></td>
        </tr>
        <tr>
          <td id="sq_31"></td>
          <td id="sq_32"></td>
          <td id="sq_33"></td>
        </tr>
      </tbody>
    </table>

    <!-- this is the button for resetting the game -->
    <button id="reset_btn">Reset Game</button>
  </center>
</body>

```

Here is the body tag and its contents. The individual squares of the board and the reset button are given ids so that I can give them a click event in JavaScript later.

The JavaScript then consists of one file with many functions which I will add screenshots of here. I will only add explanation on top of the comments if I think it is necessary. I am not necessarily explaining the JavaScript in which it appears in the file as the order how the functions are defined is not relevant or the best way to explain it.

Here I define my global variables and setup WebSocket. I initially tried to make a class to represent the noughts and crosses game that contained attributes and methods that used and changes these attributes. I then realised that the class would be all the code and that it didn't make sense as I would only ever have one instance. Due to the small size of the code, this lead me to decide to use global variables to contain the data about the current game.

```
// create a socket object form the online socket library
let socket = io();

// global constants to show game state
let moves_left = 9;
let next_to_go= "X";
let board_positions= [
  ['', '', ''],
  ['', '', ''],
  ['', '', '']
];
```

I then I add click events to the squares and the reset button

```
// code to be executed when the page loads
window.addEventListener('load', function(){
  // for testing (if loading with a non blank game state)
  // update_widget();

  // the following blocks of code bind the appropriate handler method to each square
  // I tried to use iteration but it results in a logic error that I struggled to debug
  // I therefore adopted the the less elegant manual approach
  // the looping code is still here commented out as I have tried to make it work

  // for(i=1; i<=3; i++){
  //   for(j=1; j<=3; j++){
  //     get_square(i, j).addEventListener('click', function () { square_click(i, j) });
  //   }
  // }
  get_square(1,1).addEventListener('click', function(){square_click(1, 1)});
  get_square(1,2).addEventListener('click', function(){square_click(1, 2)});
  get_square(1,3).addEventListener('click', function(){square_click(1, 3)});
  get_square(2,1).addEventListener('click', function(){square_click(2, 1)});
  get_square(2,2).addEventListener('click', function(){square_click(2, 2)});
  get_square(2,3).addEventListener('click', function(){square_click(2, 3)});
  get_square(3,1).addEventListener('click', function(){square_click(3, 1)});
  get_square(3,2).addEventListener('click', function(){square_click(3, 2)});
  get_square(3,3).addEventListener('click', function(){square_click(3, 3)});

  // tie the reset the game handler function to the reset button on click
  document.getElementById('reset_btn').addEventListener('click', reset_game)
});
```

I will not show you the functions that I defined. In theory many of them are actually procedures as they don't return anything. However they behave in a similar way as they use global variables.

```
// this function takes a row and column 1 to 3 and gives the html element
function get_square(row, col){
  // console.log('get_square called');
  // return document.querySelector(`#sq_${row.toString() + col.toString()}`);
  return document.getElementById(`sq_${row.toString() + col.toString()}`);
};
```



```

// this is the method that is run whenever a square is clicked
function square_click(row, col){
    // console.log("square_click called");

    // validate that the square clicked is a legal move for the user to make
    // if the game isn't over
    if (!(is_game_over())){
        // and if the next to go is X, the user
        if(next_to_go === "X"){
            // and if clicked square is a legal move
            if(is_legal_move(row, col)){
                // run the make move function
                make_move(row, col);
            };
        }
        // else do nothing and ignore the click input as if the square tag were disabled.
        // else {
        //     console.log({next_to_go})
        //     console.log("user move ignored as server next to go")
        // }
    };
};

```

```

// checks if the target square is empty and returns a boolean
function is_legal_move(row, col){
    // console.log("is_legal_move called");
    // console.log({row})
    // console.log({col})
    i=row-1; j=col-1;
    return board_positions[i][j] === ' ';
};

```

```

// this function is responsible for determining if the current game is over.
// this is done by examining moves left to detect draws and triplet sequences to detect wins
// this function only returns a boolean value and not how the game is over
function is_game_over(){
    // console.log("is_game_over called");
    // if no moves left then the game must be over
    if(moves_left === 0){
        return true;
    };

    // get the relevant triplet sequences
    triplets = get_triplets();
    console.log({triplets})

    // iterate through the 8 triplets
    // for(i=0; i<=8; i++){
    for(i=0; i<=7; i++){
        triplet = triplets[i];
        // using 2 equals comparison on purpose
        // i am using JSON.stringify to compare arrays to see if they are 3 in a row
        if (JSON.stringify(triplet) == JSON.stringify(["X", "X", "X"]) || JSON.stringify(triplet) == JSON.stringify(["O", "O", "O"])){
            return true;
        };
    };
    // if not already determined that the game is over then it must not be over
    return false;
};

```

```

// this returns an array of all rows, columns and diagonals on the board for examination
function get_triplets(){
    triplets = []
    // push all rows
    for(i=0; i<=2; i++){
        triplets.push(board_positions[i])
    }
    // push all columns
    for(j=0; j<=2; j++){
        column = []
        for (i = 0; i <= 2; i++) {
            column.push(board_positions[i][j])
        }
        triplets.push(column);
    }
    // add diagonals
    triplets.push([board_positions[0][0], board_positions[1][1], board_positions[2][2]])
    triplets.push([board_positions[0][2], board_positions[1][1], board_positions[2][0]])
    return triplets
}

// this function is run once the users move has been validated and is responsible for executing
// it orchestrates the various functions that must be called in the process of making a move
function make_move(row, col){
    // console.log("make_move called");
    i=row-1; j=col-1;
    // alter board position array to make the move
    board_positions[i][j] = next_to_go;
    // change who is next to go
    toggle_next_to_go();
    // console.log({next_to_go})
    // there is now one less move left in the game
    moves_left += -1;
    // update the visual widget in html
    update_widget();

    // if the game is over handle else request the server's move
    if(is_game_over()){
        handle_game_over();
        // alert("game over")
    }
    else {
        request_server_move();
    };
};

```

```

// changes the next to go to the next player based on current player
function toggle_next_to_go(){
    // console.log('toggle_next_to_go called');
    if(next_to_go === "X"){
        next_to_go = "O"
    }
    else {
        next_to_go = "X"
    }
};

```

```

// this function updates the table tag in html so the contents reflects the board positions array.
function update_widget(){
    // console.log("update_widget called");
    // console.log({board_positions})
    var row; var col;
    for (let i = 0; i <= 2; i++) {
        for (let j = 0; j <= 2; j++) {
            row=i+1; col=j+1;
            square = get_square(row, col);
            // console.log("changing content of this tag")
            // console.log(`square ${row.toString()} ${col.toString()}`)
            // console.log({square})
            // console.log(`to ${board_positions[i][j]}`)
            // square.innerHTML = board_positions[i][j];
            square.textContent = board_positions[i][j];
            // square.textContent = "Test";
        }
    }
};

```

```

// this function is run once the game is over
// it determines how the game is over and takes displays the appropriate message
function handle_game_over(){
    // precondition game is over so function is_game_over returned true
    // determine who won or if it was a draw
    // 1 is user (X) won, 0 is draw, -1 is user (X) lost
    // console.log("handle_game_over called");

    // variable initialized with 0 to represent draw
    winner = 0;

    // get triplets and iterate through them
    triplets = get_triplets();
    // for (i = 0; i <= 8; i++) {
    for (i = 0; i <= 7; i++) {
        triplet = triplets[i];
        // for each triplet make a comparison to check if the triplet means that the user has won or lost
        // using 2 equals comparison on purpose
        if (JSON.stringify(triplet) == JSON.stringify(["X", "X", "X"])) {
            winner = 1;
            // no real need to break out of the loop as there can only be one or zero winning or losing sequence
            // but it makes the code more efficient and readable
            break;
        }
        else if (JSON.stringify(triplet) == JSON.stringify(["O", "O", "O"])){
            winner = -1
            break;
        }
    };
    // if there were no winning sequences then the outcome will remain the default: 0 for draw

    // display the appropriate message to the user in each case
    if(winner === 1){
        alert("congratulations you won");
    }
    else if(winner === 0){
        alert("the game was a draw");
    }
    else{
        alert("unfortunately you lost");
    }
    };
    // no need to disable the game as the is_game_over check will fail whenever the user clicks a square
};

function reset_game(){
    // reset globals
    // if the user has just in the last second moved and then clicked reset before the computers move
    // | a logic error could occur where the game resets and then the server responds with its move
    // this is prevented here by ensuring that a server move isn't pending
    if(next_to_go === "O"){
        return false
    }

    moves_left = 9;
    next_to_go = "X";
    board_positions = [
        ['', '', ''],
        ['', '', ''],
        ['', '', '']
    ];
    // update the html to reflect the board position matrix
    update_widget();
}

```

The following functions are used to request and handle the server's move. They require a little more explanation

```
// this function emits a request to the server for its move
function request_server_move(){
    // console.log("request_server_move called");
    // it uses socket emit
    // is provides the server with the relevant game data
    socket.emit(
        'server_move_request',
        {
            board_positions,
            next_to_go,
            moves_left
        }
    );
};
```

This function sends a request to the server for its move using the socket.emit function. The server will process the request and then respond with a new game state, that is where its move is added.

The following event handler ties ques a relevant handler function to run when this happens.

```
// tie the handler method to the socket event.
socket.on('server_move_response', function(msg){
    // console.log({msg})
    // update global for board positions with the new one that the server has sent
    board_positions = msg.board_positions
    // call handler function
    handle_server_move();
    // i believe all the handler function have to return false
    return false;
});
```

Here is the handle_server_move function

```
// this function is called to handle a response from the server and execute the server's move
// the global board positions will have already been updated with the ones returned by the server
function handle_server_move(){
    // update the html to show the user the updated game state
    update_widget();
    // decrement the moves left
    moves_left += -1;
    // toggle who is next to go to the user
    toggle_next_to_go();
    // check if the servers move is a winning one and if so handle appropriately
    if (is_game_over()) {
        handle_game_over();
        // alert("game over")
    };
    // no else needed to enable the user to have a turn as game over validation included before user's turn
};
```

This is all the client side JavaScript that I used to make the interface work. The rest of the program is python based. It is responsible for hosting the website and determining the best move.

Here is the code for app.py:

This is a python program that acts as the webserver and socket server for the frontend website. It is a relatively small program

```
# imports
# built in libraries
import flask
# import os
from flask_socketio import SocketIO

# my local code
# import logger as logger_module
from game_engine import game_engine as ge
```

To start with I Imported 2 libraries to help provide the WSGI and WebSocket server functionality. I also imported the functions from a program I write called game_engine. This contains the code for my minimax function.

```
# setup flask app
app = flask.Flask(__name__)

# setup sockets
socketio = SocketIO(app, async_mode=None)
# RuntimeError: You need to use the gevent-websocket server
# issue solved by running with 'python -m flask run'
```

I then setup my app object and socketio object. This is boilerplate code to use the relevant libraries and setup my WSGI and socket server.

```
# define flask api route handlers
# this endpoint responds with the index.html page which will then load and request files from the static folder.
# this is the main endpoint that hosts the
@app.route('/', methods=['GET'])
def index():
    return flask.render_template('index.html')
```

I then used my app object to setup a handler for what happens if a HTTP get request is made to the root. This method returns the html page. I use the flask render template method so that URLs to external CSS and JS files are substituted into the html code where {{ }} double curly brackets are used.

```

# define socket handler for move request
# decorator on move request sent from client
@socketio.on("server_move_request")
def handle_server_move_request(msg):
    # unpack json data and load into object
    game_state: ge.Game_State = ge.Game_State(
        board_positions=msg['board_positions'],
        to_go_next=msg["next_to_go"],
        moves_left=msg['moves_left']
    )
    # use game engine module's main function to determine the best child state (game state following best move)
    best_child: ge.Game_State = ge.main(game_state)

    # code used for debugging to print out the move
    # print('selecting move:')
    # for row in best_child.board_positions:
    #     print(row)

    # now use emit to give a response in the form of an updated board matrix
    socketio.emit(
        "server_move_response",
        {"board_positions": best_child.board_positions}
    )

```

I then used my socketio object to define a handler function for the server_move_request.

It unpacks data provided into a Game_State class from the game_engine module. It then gets the best child game state using the minimax function. This represents the game state after the server has made the best possible move. The resulting board positions matrix is then sent back to the client by making an emit with the name 'server_move_response'.

```

# this function simply runs the app and configures the port that is used.
def run_app():
    app.run(host='127.0.0.1', port=5000, debug=True)

# this code runs the main app function only if this file is run directly and not if it is imported
if __name__ == '__main__':
    run_app()

```

The above code is then responsible for starting the server when this python file is run directly. It used port 5000 on the local host.

Game engine file

This file contains code that is used to determine the server's best move.

```

# the data class library is used to minimize boiler plate code in defining __init__ and __eq__ methods for classes
from dataclasses import dataclass
# this is used to make a complete and distinct copy of a 2d array so changes to the original don't affect the copy
from copy import deepcopy

```

Here is imported external libraries and gave the reasons for why

```

# this is a class to represent the game state
# its attributes cannot be altered after it is initialized (immutable).
# instead changes to the game state, such as a move, are handled as new game state objects
@dataclass(frozen=True)
class Game_State:
    # there variables hold the data relevant to a game state
    board_positions: list[list]
    moves_left: int
    to_go_next: str

    # used for debugging so that the game state can be represented in the console.
    def print_board(self):
        for row in self.board_positions:
            print("|".join("." if e == "" else e for e in row))

    def is_game_over(self):...

    # this function iterates through and yields the game states that could result form all possible moves on the current game state
    def gen_child_game_states(self):...
        # else continue to next iteration

```

I then use a class to represent a tix tac toe game in a specific state.

```

def is_game_over(self):
    """Returns False, none for still going and True, 1/0/-1 for over.
    1 is user wins, 0 is draw, -1 is user loses"""
    # internal function as it only used here
    # yields all of the sequences of 3 squares
    def gen_triplets():
        # yield rows
        yield from self.board_positions
        # yield columns
        for col in range(3):
            yield [self.board_positions[row][col] for row in range(3)]
        # yield diagonals
        yield [
            self.board_positions[0][0],
            self.board_positions[1][1],
            self.board_positions[2][2],
        ]
        yield [
            self.board_positions[2][0],
            self.board_positions[1][1],
            self.board_positions[0][2],
        ]
    # if there is a sequence that is 3 in a row return True and 1 or -1 for win or loss (user perspective)
    if ["X"]*3 in gen_triplets():
        return True, 1
    if ["O"]*3 in gen_triplets():
        return True, -1
    # note so long as user takes all odd turns 9,7,...3,1 they will have last go so redundant in theory
    # but I left it in incase the code needs to be extended in future
    if self.moves_left == 0:
        return True, 0
    # if not already determined otherwise then the game isn't over
    return False, None

```

I then produced a function to determine if the game is over and if so what the state is


```

# this function iterates through and yields the game states that could result from all possible moves on the current game state
def gen_child_game_states(self):
    """precondition of game not being over"""
    # for each square
    for i, row in enumerate(self.board_positions):
        for j, square in enumerate(row):
            # if the square is empty
            if square == "":
                # make a copy of the board state where the next to move moved there
                new_board_positions = deepcopy(self.board_positions)
                new_board_positions[i][j] = self.to_go_next

                # yield this as a new game state where the user is next to move
                yield Game_State(
                    board_positions=new_board_positions,
                    moves_left=self.moves_left-1,
                    to_go_next="X" if self.to_go_next == "O" else "O"
                )
            # else continue to next iteration

```

I then produced a function to iterate through all empty squares on the board and yield the resulting game state if the server had moved there. This function returned an iterator of child Game_State objects

```

# this is so called as it used the minimax algorithm along with alpha beta pruning to navigate the decision tree
# the british museum algorithm is an approach where you explore the tree fully
# until all terminal nodes are game states where the game is over
# this function returns 2 values, the score and game state object for the best child game state (corresponds to best move)
def british_museum_minimax(game_state: Game_State, maximizing_player: bool, alpha, beta):
    # computer want 0 or -1 so minimizer
    # this is the base case to the recursive function (stop at terminal nodes)
    over, outcome = game_state.is_game_over()
    if over:
        return outcome, game_state

    # recursive case
    best_child = None

    # even though the initial call is for the servers move (minimizer), recursive calls may be from maximizer
    # I will explain the logic in more depth for the maximizing player
    if maximizing_player:
        # similar for minimizer
        else:

```

```

if maximizing_player:
    # meant to represent negative infinity (arbitrary bad so it is replaced by the best evaluation)
    max_evaluation = -100
    # for each possible game state
    for child in game_state.gen_child_game_states():
        # recursive class to function to evaluate child node
        # now minimizing player
        # index for first element as I only care about score not the actual game state
        evaluation = british_museum_minimax(
            game_state=child,
            maximizing_player=not maximizing_player,
            alpha=alpha,
            beta=beta,
        )[0]
        # max_evaluation = max(max_evaluation, evaluation)
        # if this evaluation is the best so far
        if evaluation > max_evaluation:
            # update the max evaluation and best child (only for this call)
            max_evaluation = evaluation
            best_child = child
            # update the alpha value to represent the best that the maximizing player can get
            alpha = max(alpha, evaluation)

    # if the minimizing player can do better (for them less) than alpha then they have a better option
    # this means that that this won't be the best child of the earlier call
    # (the minimizer wouldn't need to give the maximizer such a good score)
    if beta <= alpha:
        break
    # return the best child game state and its score
    return max_evaluation, best_child

```

```

# similar for minimizer
else:
    min_evaluation = 100
    for child in game_state.gen_child_game_states():
        evaluation = british_museum_minimax(
            game_state=child,
            maximizing_player=not maximizing_player,
            alpha=alpha,
            beta=beta,
        )[0]
        # min_evaluation = min(min_evaluation, evaluation)
        if evaluation < min_evaluation:
            min_evaluation = evaluation
            best_child = child
            beta = min(beta, evaluation)
            if beta <= alpha:
                break
    return min_evaluation, best_child

```

The function is explained by the comments. It explores the decision tree down to the terminal nodes where games are over and then decides what the best move is. It returns the score and the object for the best child of the current game state object

```

# this main function is an easier wrapper for the app module to use
# it only determines and returns the best child (servers best move)
def main(game_state: Game_State):
    # output for debugging
    print("selecting best move from this game state:")
    game_state.print_board()

    # call the recursive function to determine the best child (and score for debugging)
    # start with arbitrarily low alpha value and high beta value
    score, best_child = british_museum_minimax(
        game_state=game_state,
        maximizing_player=False,
        alpha=-100,
        beta=+100
    )
    # more output for debugging
    print("best move is:")
    best_child.print_board()
    print(f"with a guaranteed score of {score} or better (minimizer)")

    # return the best child
    return best_child

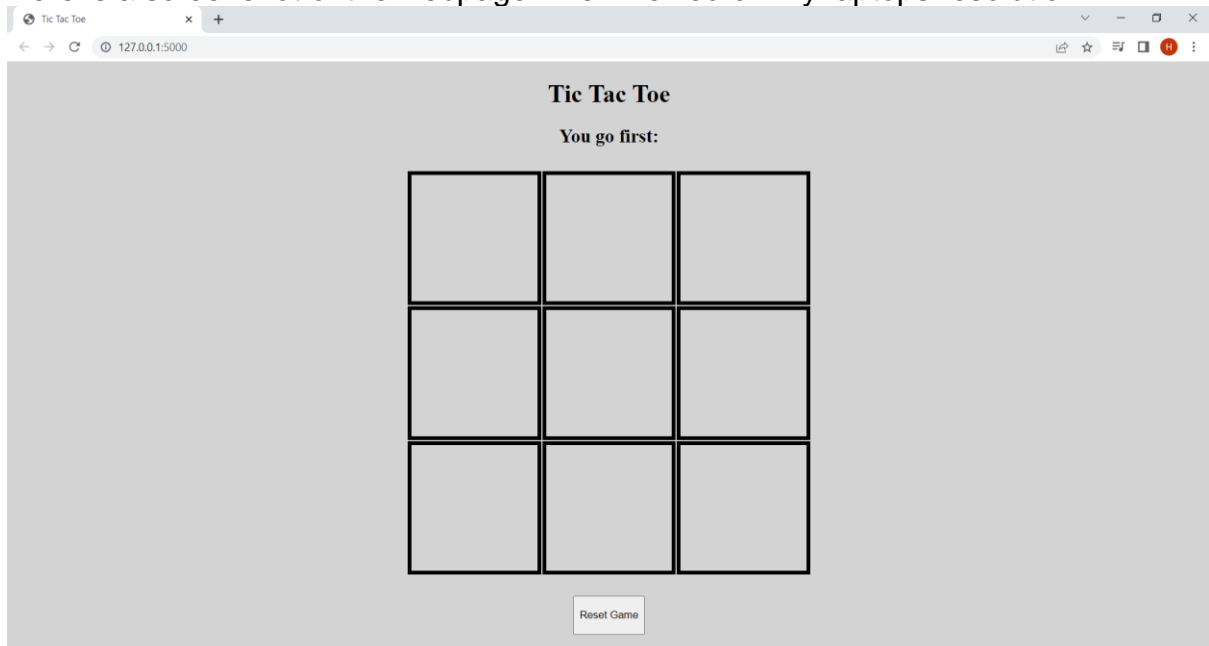
```

I then made a main function for the whole module that provides abstraction from the minimax function. It takes a Game_State and returns the child game state corresponding to the best move. This is the function that the app.py program uses.

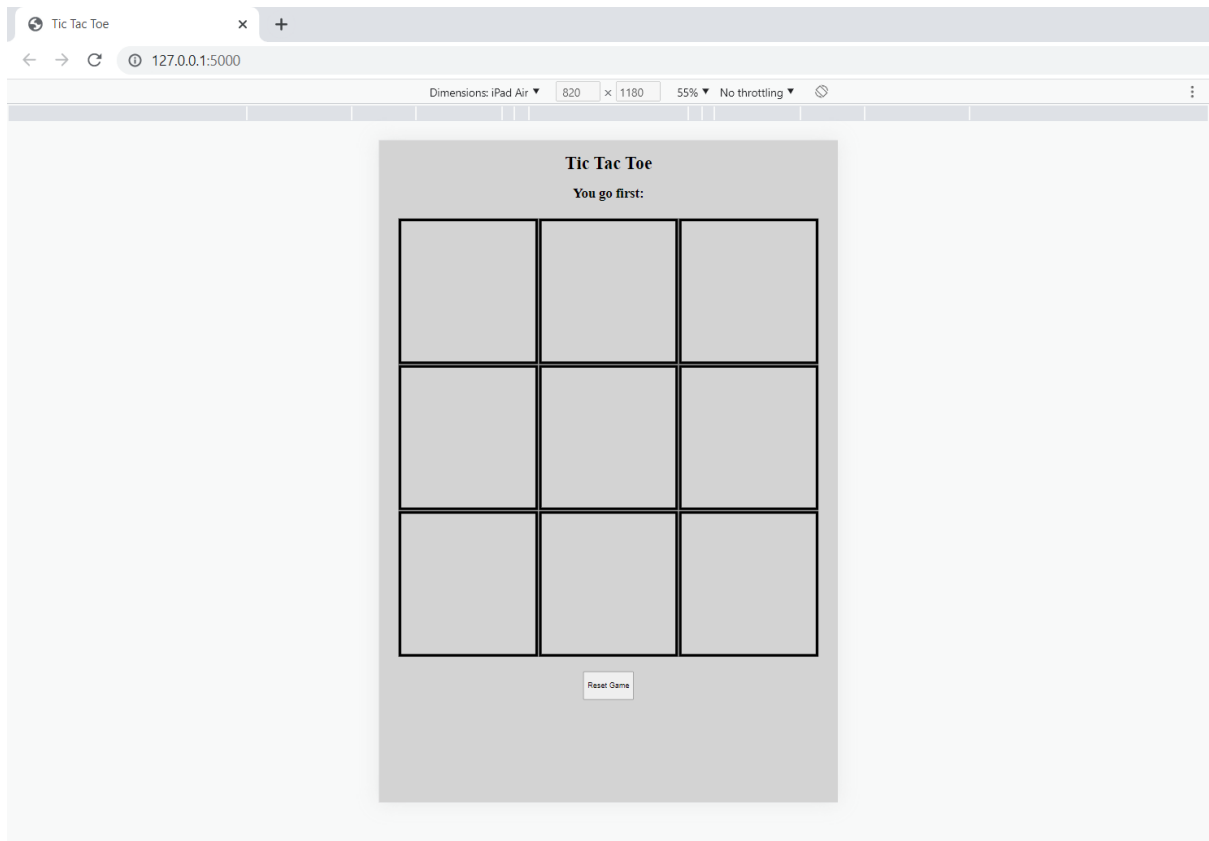
This is all the code used in the project.

Here are some screenshots of the final product:

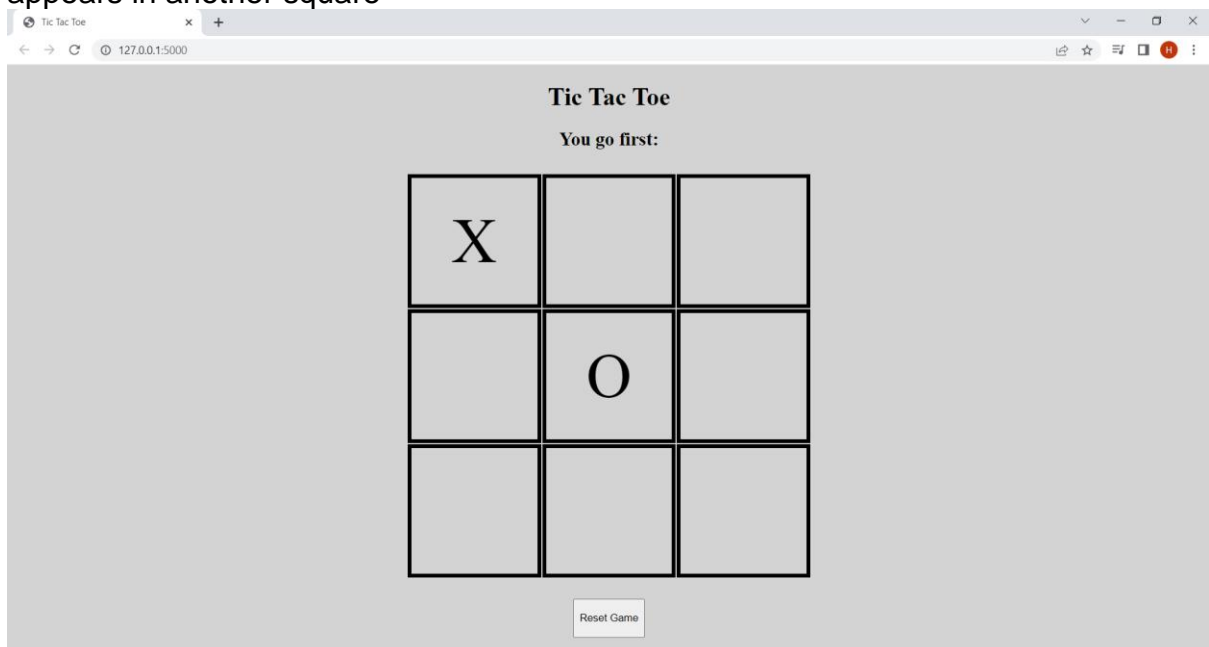
Here is a screenshot of the webpage when viewed on my laptops resolution



And here is a screenshot of the website from the resolution of the ipad

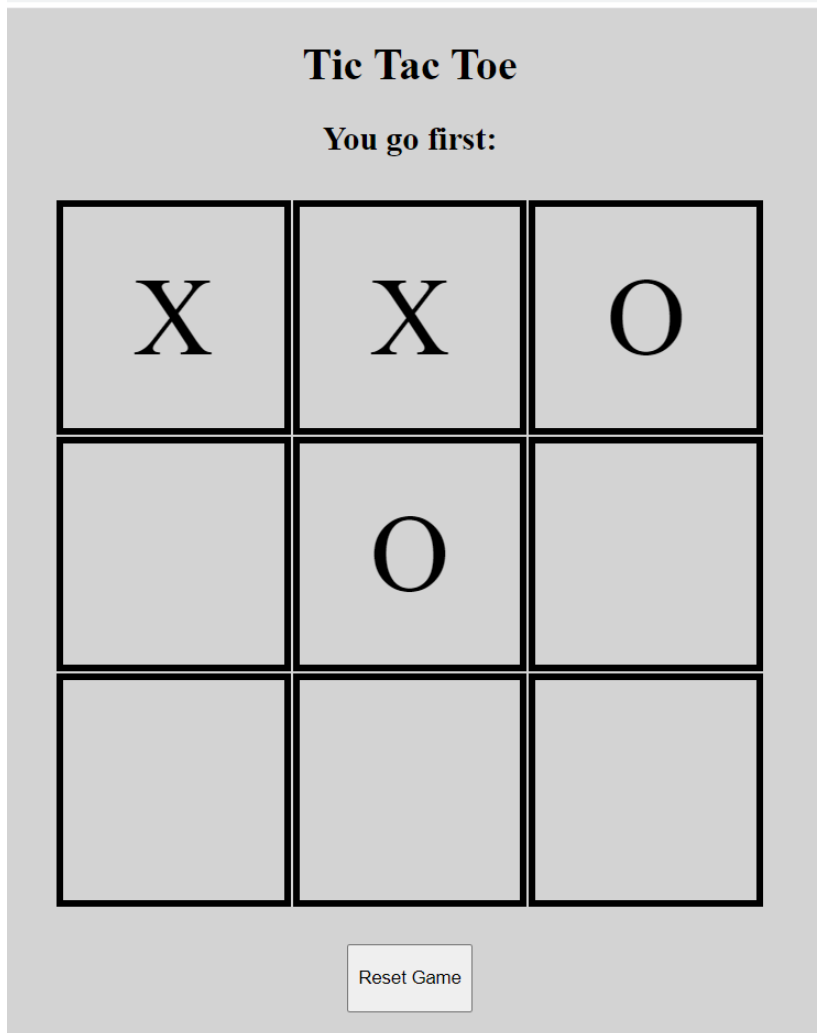


When I click on a square a 'X' Appears in it. Then a fraction of a second later a 'O' appears in another square

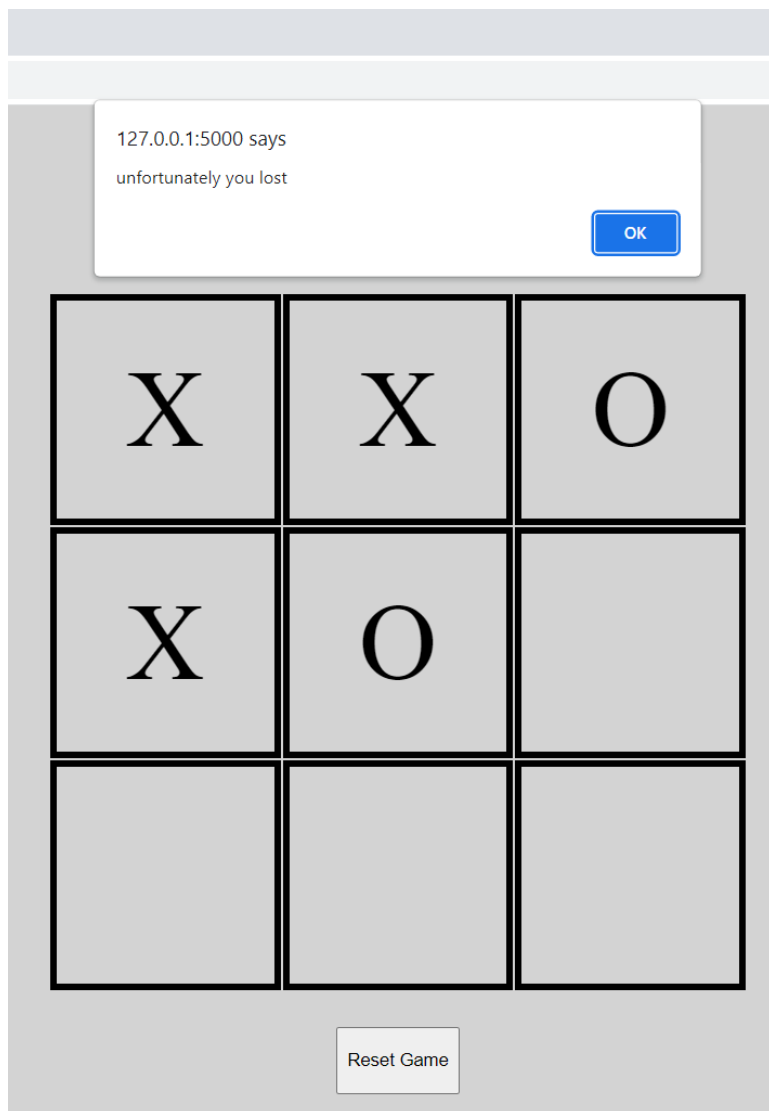


If I click one of the squares containing the 'X' or 'O', nothing happens due to client side validation

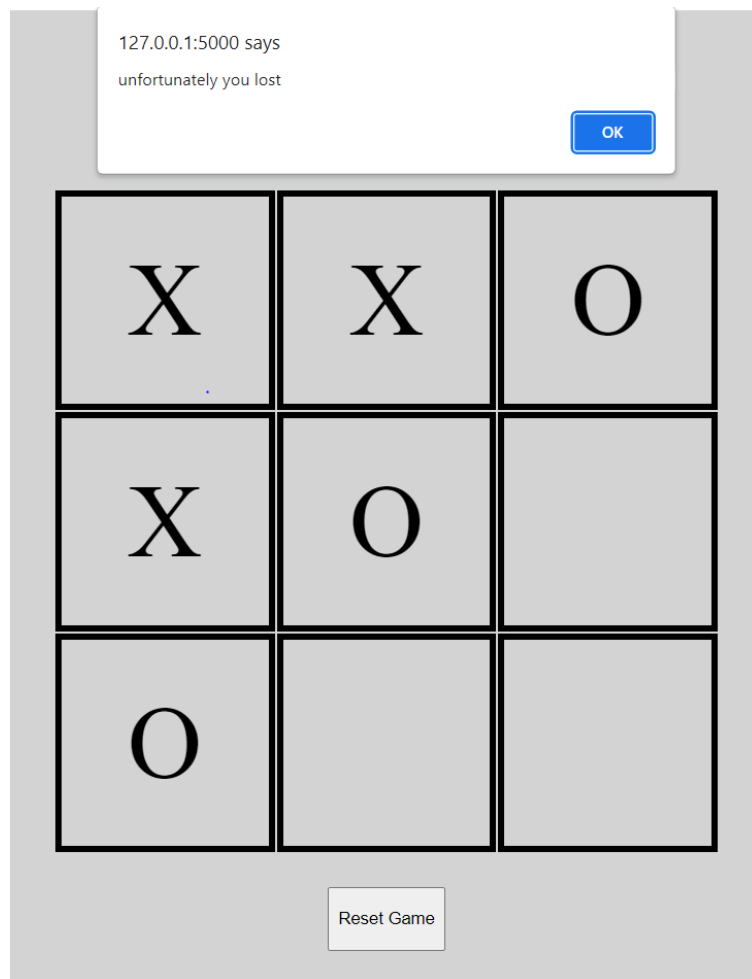
Below are a series of screenshots of a game where I allow the computer to win. (I will not show all of the webpage just the relevant part like the board as the rest is clear)



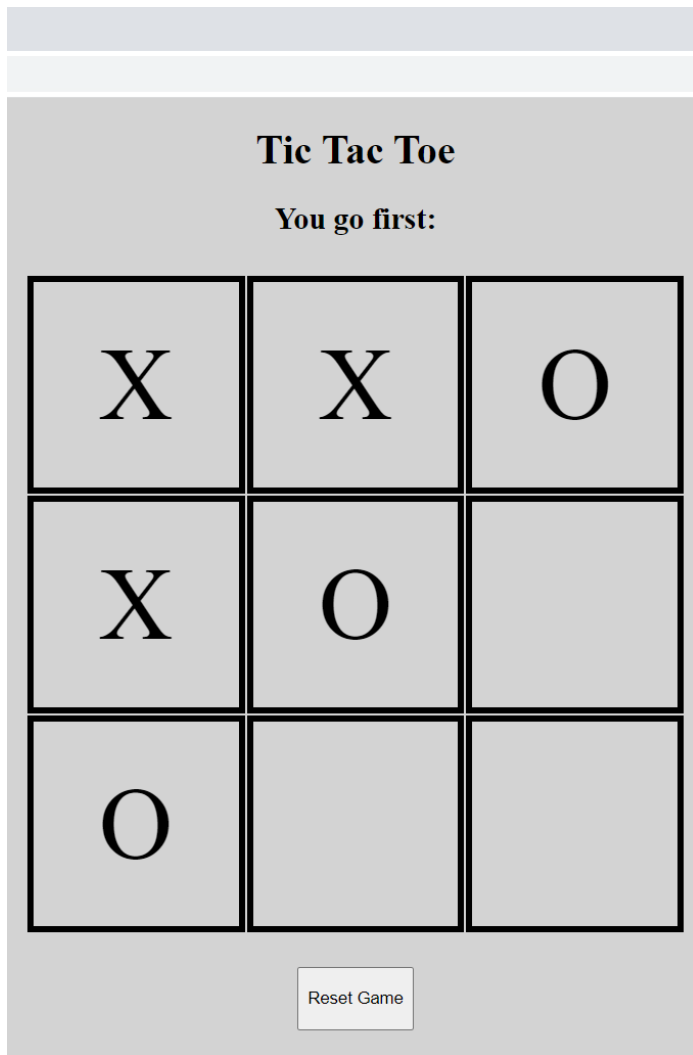
I then clicked the square in the 1st column and second row and this was the result



This shows a logic error that is minor but hard to deal with. It is where the alert to say that the game is over comes before the winning move (in this case a 'O' in the bottom left).



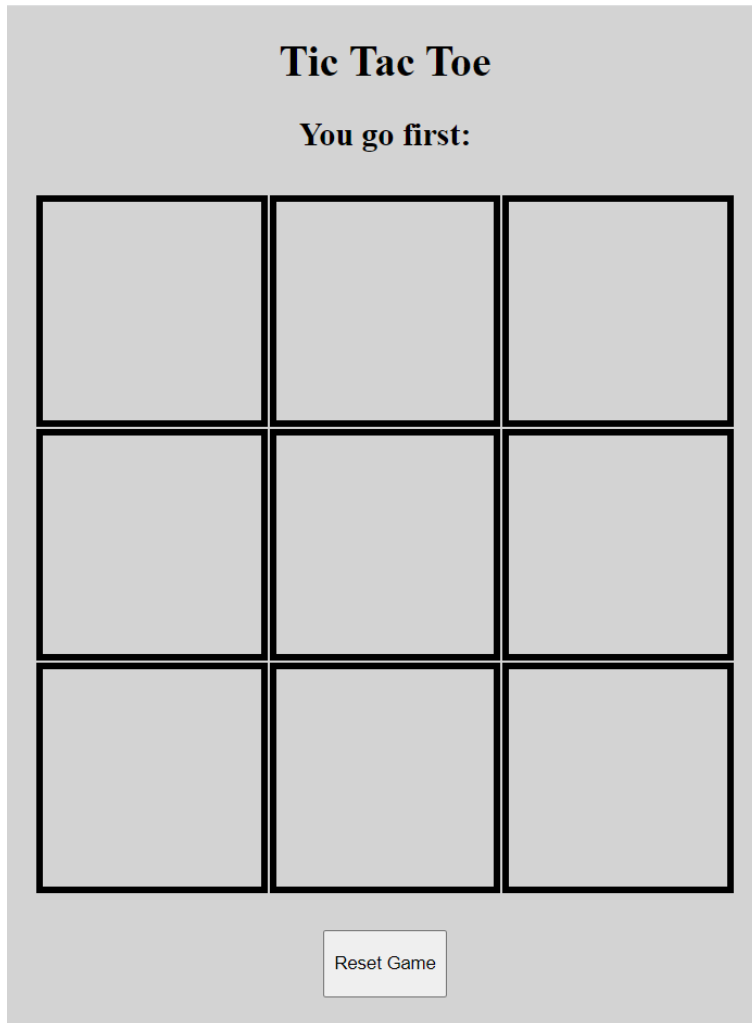
I noticed that if the window is minimised and then clicked again the winning move is shown



The winning move is also shown either way when the user clicks 'ok' on the alert. This is an issue but it doesn't prevent the program from functioning, it is more of a ruff edge.

Now if I click any square nothing happens as the game is over

If I click the reset button the board resets as shown



Problems throughout and how I tackled them:

While I did have to use a bit of trial and error to get my CSS to work the main problem I had in development was with my minimax function.

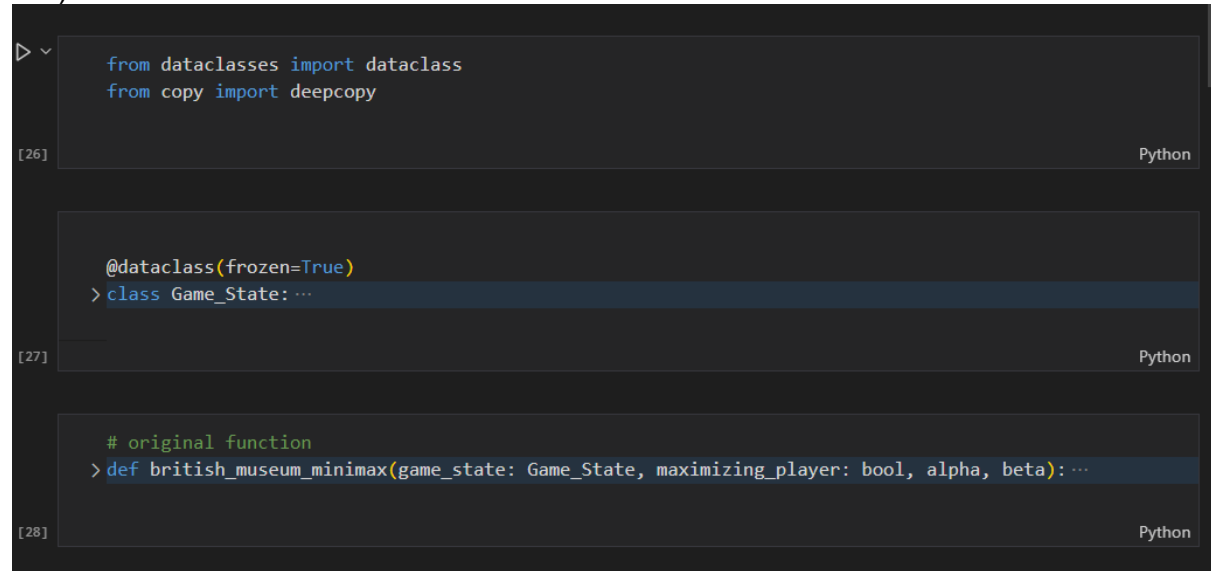
The process of developing this prototype was to make a rudimentary server that couldn't decide the best move. I then created the frontend and tested the program. The server would use command line input to allow me to input a move and simulate the minimax process before I wrote it. I then wrote my minimax function and found that it wasn't working.

I found that it was returning a move but not always the best move. It was strangely inconsistent and missed opportunities to win or stop the user winning at times. This was a challenge to debug as the minimax function is complex and has many recursive calls. For instance, for the first computer move it could have (without alpha beta pruning) $8!$ or 40320 recursive calls (total calls not depth). The challenge therefore was to try to make the function more transparent so I could see how it was behaving.

To do this I tried using breakpoints and stepping but this was not fruitful as it was hard to keep track of what the correct behaviour of each recursive call should be and I was unsure where the problem was. So I created a jupyter notebook which is a python

environment where I can define cells which are blocks of code I can run manually. This allowed me to import the function and to see what the results were for a variety of outputs. Here is what I did:

I started by importing some libraries and making a copy of the invalid code (which I ran)



```
[26] from dataclasses import dataclass
     from copy import deepcopy
Python

[27] @dataclass(frozen=True)
     > class Game_State: ...
Python

[28] # original function
     > def british_museum_minimax(game_state: Game_State, maximizing_player: bool, alpha, beta): ...
Python
```

(code is hidden to save space as it is repeated from earlier)

I did this rather than import from the file so that my initial tests wouldn't change if I tried to fix the version in the file.

I then picked an example game state to try to capture the issue

```
▶ score, best_child = british_museum_minimax(  
    game_state=Game_State(  
        moves_left=6,  
        to_go_next="O",  
        board_positions=[  
            ["O", "X", ""],  
            ["", "X", ""],  
            ["", "", ""],  
        ]  
    ),  
    maximizing_player=False,  
    alpha=-100,  
    beta=+100  
)  
[29] Python
```

```
score  
[30] Python
```

```
... 1
```

```
▶ best_child.print_board()  
[31] Python
```

```
...  O|X|O  
    .|X|.   
    .|.|. 
```

This shows a game state where the computer could prevent the user from winning by placing an O in the bottom row and middle column. As we can see there is an issue as it fails to do this. The score of 1 shows that the minimax function believes that no matter what it does, from here if the user plays optimally it has lost (computer is minimiser).

```
> ✓
score, best_child = british_museum_minimax(
    game_state=Game_State(
        moves_left=2,
        to_go_next="O",
        board_positions=[
            ["O", "X", "X"],
            ["O", "X", "O"],
            ["", "", "X"],
        ]
    ),
    maximizing_player=False,
    alpha=-100,
    beta=+100
)

32] Python

score

33] Python

.. -1

best_child.print_board()

34] Python

.. O|X|X
   O|X|O
   O|. |X
```

A similar test showed that sometimes the algorithm works. Here it successfully determines that it can win by putting an O in the bottom left

This was helping me to build up a picture of when it fails

Perhaps the issue is with the game over function incorrectly determining who wins

+ Code

+ Markdown

```
Game_State(  
    moves_left=0,  
    to_go_next="O",  
    board_positions=[  
        ["O", "X", "X"],  
        ["O", "X", "O"],  
        ["X", "O", "X"],  
    ]  
)  
.is_game_over()
```

[35]

Python

... (True, 1)

Thats correct

```
Game_State(  
    moves_left=0,  
    to_go_next="O",  
    board_positions=[  
        ["O", "X", "X"],  
        ["X", "O", "O"],  
        ["X", "O", "X"],  
    ]  
)  
.is_game_over()
```

[36]

Python

... (True, 0)

thats also correct

```
Game_State(  
    moves_left=0,  
    to_go_next="O",  
    board_positions=[  
        ["X", "O", "O"],  
        ["X", "O", "X"],  
        ["O", "X", "O"],  
    ]  
)  
.is_game_over()
```

Python

(True, -1)

this is also correct

I then investigated to check that the game over function was working as it should or if it was getting confused as to who won. It was working.

I then tried to visualize the decision tree that the minimax function was exploring


I will now try to make a tree to better visualize what is happening

```
@dataclass
class Tree:
    root_node_text: str
    sub_trees: list

    def print_tree(self, indent=0):
        # print(f"printing tree {self.root_node_text} with indent {indent}")
        print("|" + "---" * indent + f"-->{self.root_node_text}")
        for sub_tree in self.sub_trees:
            sub_tree.print_tree(indent=indent+1)
```

[38]

Python



```
Tree("A",[
  Tree("B",[
    Tree("E", [])
  ]),
  Tree("C",[
    Tree("D", [
      Tree("F", []),
      Tree("G", [])
    ])
  ])
]).print_tree()
```

```
.. |-->A
   |----->B
   |----->E
   |----->C
   |----->D
   |----->F
   |----->G
```

that isn't a great tree

This didn't work out and so I tried a different approach

maybe I could just add a load of print statements

```
def print_dec(func):
    func_name = func.__name__
    def wrapper(*args, **kwargs):
        print(f"started: {func_name}(args={args}, kwargs={kwargs})")
        result = func(*args, **kwargs)
        print(f"finished: {func_name}(args={args}, kwargs={kwargs}) return value: {result}")
        return result

    wrapper.__name__ = func_name
    return wrapper
```

[40]

Python

```
# new transparent version
@print_dec
> def british_museum_minimax(game_state: Game_State, maximizing_player: bool, alpha, beta):...
```

[41]

Python

I then created a decorator to augment the function's behaviour. It caused the function to print out its arguments when called and the result when it finished.

```
british_museum_minimax(
    game_state=Game_State(
        moves_left=2,
        to_go_next="0",
        board_positions=[
            ["0", "X", "X"],
            ["0", "X", "0"],
            ["", "", "X"],
        ],
    ),
    maximizing_player=False,
    alpha=-100,
    beta=+100
)
```

[5]

✓ 0.5s

Python

```
... started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', 'X'], ['0', 'X', '0'], ['', '', 'X']],
moves_left=2, to_go_next='0'), 'maximizing_player': False, 'alpha': -100, 'beta': 100})
started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', 'X'], ['0', 'X', '0'], ['0', '', 'X']],
moves_left=1, to_go_next='X'), 'maximizing_player': True, 'alpha': -100, 'beta': 100})
finished: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', 'X'], ['0', 'X', '0'], ['0', '', 'X']],
moves_left=1, to_go_next='X'), 'maximizing_player': True, 'alpha': -100, 'beta': 100})
return value: (-1, Game_State(board_positions=[['0', 'X', 'X'], ['0', 'X', '0'],
['0', '', 'X']], moves_left=1, to_go_next='X'))
finished: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', 'X'], ['0', 'X', '0'], ['', '', 'X']],
moves_left=2, to_go_next='0'), 'maximizing_player': False, 'alpha': -100, 'beta': 100})
return value: (-1, Game_State(board_positions=[['0', 'X', 'X'], ['0', 'X', '0'],
['0', '', 'X']], moves_left=1, to_go_next='X'))
```

This worked as I used a near completion game so that there were fewer recursive calls. Due to pruning there is only one recursive call as it successfully identifies the winning move.

```
[43] best_child.print_board()
Python

... 0|x|x
    0|x|0
    0|. |x

[44] score
Python

... -1

that makes sense, lets try to do this for one that doesn't work
```

This example seemed to all be in order

```
[45] score, best_child = british_museum_minimax(
    game_state=Game_State(
        moves_left=6,
        to_go_next="0",
        board_positions=[
            ["0", "X", ""],
            ["", "X", ""],
            ["", "", ""],
        ]
    ),
    maximizing_player=False,
    alpha=-100,
    beta=+100
)

best_child.print_board()
print(score)
Python

... Output exceeds the size limit. Open the full output data in a text editor
started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', ''], ['', 'X', ''], ['', '', '']], moves_left=6,
to_go_next='0'), 'maximizing_player': False, 'alpha': -100, 'beta': 100})
started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', '0'], ['', 'X', ''], ['', '', '']], moves_left=5,
to_go_next='X'), 'maximizing_player': True, 'alpha': -100, 'beta': 100})
started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', '0'], ['X', 'X', ''], ['', '', '']], moves_left=4,
to_go_next='0'), 'maximizing_player': False, 'alpha': -100, 'beta': 100})
started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', '0'], ['X', 'X', '0'], ['', '', '']],
moves_left=3, to_go_next='X'), 'maximizing_player': True, 'alpha': -100, 'beta': 100})
started: british_museum_minimax(args=(), kwargs={'game_state':
Game_State(board_positions=[['0', 'X', '0'], ['X', 'X', '0'], ['X', '', '']],
```


I then tried to look at the output for a game where the function had failed earlier. This was not fruitful as the output was too large to successfully analyse (6! Or 720 calls without pruning).

This is the end of my use of Jupyter notebook. It did however help me understand a little more what was happening when the program failed.

From coming back to the problem at a later date I realised that the client side JavaScript couldn't recognise a win when it was a diagonal in a row. I then fixed this on the frontend and backend by adding to the 'triplets' function. This still didn't account for some of the errors and it still wasn't working. From looking at my research into the minimax function I realised my mistake.

```
beta = min(alpha, evaluation)
```

the above line is the source of my logic error. It is contained in the else (so if minimiser) part of the algorithm. It is supposed to update beta with the best result the minimiser can so far obtain (to allow for pruning of clearly worse branches of the decision tree in future).

The correction is this:

```
beta = min(beta, evaluation)
```

With the help of my debugging and jupyter notebook this unassuming logic error was found. The correction allowed the program to work in all scenarios and play tic tac toe optimally. It is worth mentioning that I, nor my stakeholders, never beat this version of the minimax function at tic tac toe.

The other significant error I encountered was the following

```
PS C:\Users\henry\Documents\computing coursework\prototype 1> python app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 713-354-091
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Oct/2022 22:03:01] "GET /socket.io/?EIO=4&transport=polling&t=0EqGZPT HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2022 22:03:01] "POST /socket.io/?EIO=4&transport=polling&t=0EqGZQ&sid=JgJR_iMr1EGYccaQAAAA HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2022 22:03:01] "GET /socket.io/?EIO=4&transport=polling&t=0EqGZR2&sid=JgJR_iMr1EGYccaQAAAA HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2022 22:03:01] "GET /socket.io/?EIO=4&transport=websocket&sid=JgJR_iMr1EGYccaQAAAA HTTP/1.1" 500 -
Traceback (most recent call last):
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\flask\app.py", line 2091, in __call__
    return self.wsgi_app(environ, start_response)
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\flask_socketio\__init__.py", line 43, in __call__
    return super(SocketIOMiddleware, self).__call__(environ,
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\engineio\middleware.py", line 63, in __call__
    return self.engineio_app.handle_request(environ, start_response)
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\socketio\server.py", line 604, in handle_request
    return self.eio.handle_request(environ, start_response)
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\engineio\server.py", line 411, in handle_request
    packets = socket.handle_get_request(
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\engineio\socket.py", line 103, in handle_get_request
    return getattr(self, 'upgrade' + transport)(environ,
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\engineio\socket.py", line 158, in upgrade_websocket
    return ws(environ, start_response)
  File "C:\Users\henry\AppData\Local\Programs\Python\Python310\Lib\site-packages\engineio\async_drivers\gevent.py", line 35, in __call__
    raise RuntimeError('You need to use the gevent-websocket server.')
RuntimeError: You need to use the gevent-websocket server. See the Deployment section of the documentation for more information.
```

I struggled to solve this using Stackoverflow as their solution didn't work for me. I found that the program works if I run it in a different way using the following

command

```
PS C:\Users\henry\Documents\computing coursework\prototype 1> python -m flask run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
The Websocket transport is not available, you must install a Websocket server that is compatible with your async mode to enable it.
will be logged with level INFO)
127.0.0.1 - - [07/Oct/2022 22:05:05] "GET /socket.io/?EIO=4&transport=polling&t=0EqH1R8 HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2022 22:05:05] "POST /socket.io/?EIO=4&transport=polling&t=0EqH1gw&sid=XEtck_HfKGKRYIy7AAAA HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2022 22:05:05] "GET /socket.io/?EIO=4&transport=polling&t=0EqH1gx&sid=XEtck_HfKGKRYIy7AAAA HTTP/1.1" 200 -
```

In future, rather than circumvent the error (as this still gives a warning), I will try to alter the async parameter of my socket object to fix this.

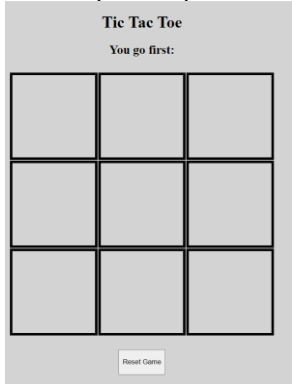
```
# setup sockets
socketio = SocketIO(app, async_mode=None)
# RuntimeError: You need to use the gevent-websocket server
# issue solved by running with 'python -m flask run'
```

This does mean that I use an integrated terminal to run my program as running the python file directly doesn't work.

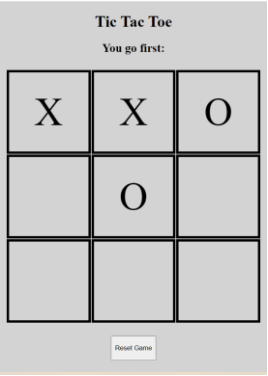
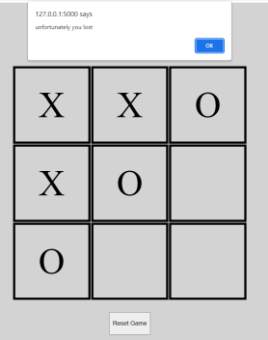
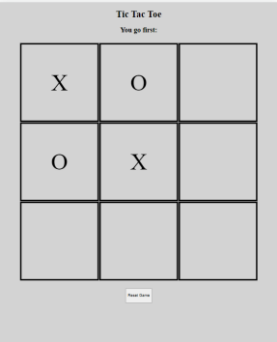
Test Plan for this version

The test plan for this program has 2 prongs. One set of tests will be testing that when I click on a specific square or button on the website, the expected behaviour happens. Another set of tests will be that in specific cases the tic tac program can see the correct move to win. These tests will be non-exhaustive as there are 9! Possible games which is too many to test.

The below tests are shown in a table. I use numbers to indicate what screenshots are evidence to what test (usually 2 for before and after)

Description	Expected behaviour	Success ?	Evidence
When the game starts I will click one of the squares	That square should fill with a 'X' then another square should fill with a 'O'	Y	<div>Tried top left square. Before</div> <div></div> <div>After</div>

			<div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div>	X				O																						
X																														
	O																													
2: I will then try to click both of the filled squares	Nothing should happen as validation should stop me moving there	y	<div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div> <p>Nothing to evidence but nothing did happen when I clicked the top left X or the centre O. the webpage still looked like this</p> <div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div>	X				O					X				O													
X																														
	O																													
X																														
	O																													
3: If I click on another empty square I can move there	An X should appear in that square and then an O should appear in another square	y	<div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div> <p>I moved to the top middle</p> <p>Before:</p> <div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div> <p>After</p> <div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div>	X				O					X				O					X	X	O		O				
X																														
	O																													
X																														
	O																													
X	X	O																												
	O																													

4: It should successfully recognise a loss	I should be able to click on another square and then the computer should move into the winning square. An alert should show that the computer has won	Y (but sometime s the alert happens before the move as already discussed)	<div>Before:</div> <div></div> <div>After:</div> <div></div>
5: The program should successfully recognise when the user wins. (Note this doesn't ever happen so I changed the JavaScript so that the game starts with the user a move away form winning and the user moves next)	The user should be able to move into the winning position. There should be no subsequent server move. An alert should show that the game is over	y	<div>Global variables changed:</div> <div><pre>// global constants to show game state // let moves_left = 9; // let next_to_go= "X"; // let board_positions= [// ['', '', ''], // ['', '', ''], // ['', '', ''] //]; let moves_left = 5; let next_to_go= "X"; let board_positions= [['X', 'O', ''], ['O', 'X', ''], ['', '', '']];</pre></div> <div>Update widget to array added on load</div> <div><pre>// code to be executed when the page loads window.addEventListener('load', function(){ // for testing (if loading with a non blank game state) update_widget(); });</pre></div> <div>Before:</div> <div></div> <div>After</div>

			<div><div>127.0.0.1:5000 says congratulations you won</div><div>OK</div><table><tr><td>X</td><td>O</td><td></td></tr><tr><td>O</td><td>X</td><td></td></tr><tr><td></td><td></td><td>X</td></tr></table><div>Reset Game</div></div>	X	O		O	X				X									
X	O																				
O	X																				
		X																			
6: The program should handle a draw correctly	If the user plays appropriately the game should continue until the user fills the 9 th square (none left empty). Then the game should end with an appropriate alert	y	<div>Before:</div> <div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td></td></tr></table><div>Reset Game</div></div> <div>After:</div> <div><div>127.0.0.1:5000 says the game was a draw</div><div>OK</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table><div>Reset Game</div></div>	X	X	O	O	O	X	X	O		X	X	O	O	O	X	X	O	X
X	X	O																			
O	O	X																			
X	O																				
X	X	O																			
O	O	X																			
X	O	X																			
7: once the game is over the board should be unresponsive	Once the game ends the user can click 'ok' on the	Y	Before:																		

	alert and then clicking any square on the board should do nothing		<div><div>127.0.0.1:5000 says the game was a draw</div><div>OK</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table><div>Reset Game</div></div> <p>After: (clicking did nothing)</p> <div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table><div>Reset Game</div></div>	X	X	O	O	O	X	X	O	X	X	X	O	O	O	X	X	O	X
X	X	O																			
O	O	X																			
X	O	X																			
X	X	O																			
O	O	X																			
X	O	X																			
8: the reset button should work once the game is over or halfway through	Clicking reset button should in effect reload the page to an empty board.	Fail (only worked in some cases). I address the correction as a result after the tests	Case 1: the game is over loss Before:																		

Tic Tac Toe

You go first:

X	X	O
X	O	
O		

Reset Game

After:

Tic Tac Toe

You go first:

Reset Game

Before:

		<div><div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table><div>Reset Game</div></div><div>After: it didn't clear, the test failed. I reloaded the window and was able to recreate the issue</div><div>Case 2: The game is still going: Before</div><div><div><div>Tic Tac Toe</div><div>You go first:</div><table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td></td><td>O</td><td></td></tr><tr><td></td><td></td><td></td></tr></table><div>Reset Game</div></div><div>After:</div></div></div>	X	X	O	O	O	X	X	O	X	X	X	O		O				
X	X	O																		
O	O	X																		
X	O	X																		
X	X	O																		
	O																			

			<div><div>Tic Tac Toe</div><div>You go first:</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div>Reset Game</div></div>
--	--	--	--

Changes to code made after testing:

So test number 8 or the reset button failed specifically when the game was over and it was a draw. I had recently added to that function and so immediately knew what the issue was. I found a logic error where a user makes a move and then immediately clicks reset, resulting in the server then responding with its move after the restart. I added an if statement to check if the program was waiting for the server's response and if so not to reset the board as highlighted here

```

function reset_game(){
  // reset globals
  // if the user has just in the last second moved and then clicked reset before the computers move
  // a logic error could occur where the game resets and then the server responds with its move
  // this is prevented here by ensuring that a server move isn't pending

  // LOGIC ERROR HERE, FAILS TO RESET WHEN DRAW
  if(next_to_go === "O"){
    return false
  }

  moves_left = 9;
  next_to_go = "X";
  board_positions = [
    ['', '', ''],
    ['', '', ''],
    ['', '', '']
  ];
  // update the html to reflect the board position matrix
  update_widget();
}

```

This caused the logic error. This is because there are 9 turns and the user makes all odd numbered turns, including the last turn which is 9. After the last turn in the case of a draw the game is over but the next to go has been changed to "O" after the move.

I initially thought I would correct this by adding an AND statement to the condition
 If next_to_go == "O" AND moves_left > 0

However in the theoretical case where the user has won, the user would have made the winning play on their go and so next to go would be "O" and there would be moves left. Therefore I need to more generally check that next to go is "O" and that the game isn't over. Here was my corrected code

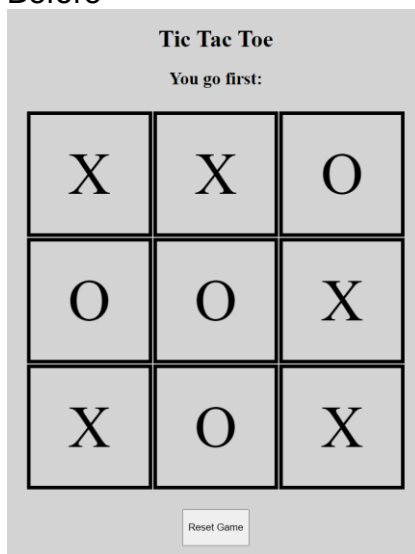
```
function reset_game(){
    // reset globals
    // if the user has just in the last second moved and then clicked reset before the computers move
    // a logic error could occur where the game resets and then the server responds with its move
    // this is prevented here by ensuring that a server move isn't pending

    // // LOGIC ERROR HERE, FAILS TO RESET WHEN DRAW
    // // if(next_to_go === "O"){
    // //     return false
    // // }

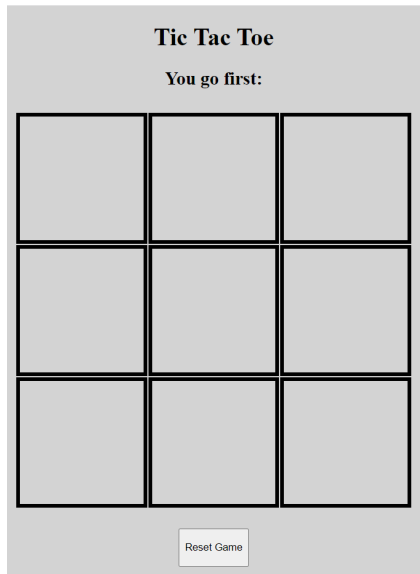
    // corrected code after reset button test
    if(next_to_go === "O" && !(is_game_over())){
        return false
    }

    moves_left = 9;
    next_to_go = "X";
    board_positions = [
        ['', '', ''],
        ['', '', ''],
        ['', '', '']
    ];
    // update the html to reflect the board position matrix
    update_widget();
}
```

The test (test 8) now is successful. The reset button now works in all cases
Before



After:



Validation:

The validation is limited in this early prototype as there is a limited number of user inputs. Due to the lack of menus there are only really 2 types of input

- The user clicks the reset button
- The user clicks a square in the 3x3 grid

I have used validation to decide whether or not an action should be taken based on the current game state.

For the square click input:

The relevant action is to call the `make_move` function. The validation before-hand is that all the following conditions are met:

- The game isn't over
- It is the User's turn to make a move
- The move is a legal move (the target square is empty)

For the reset button click input:

The relevant action is to reset the global variable values and update the board in HTML to show this. The validation consists of a condition which if met causes the restart to be ignored. This check if both:

- It is the server's turn to move
- and the game isn't over

this prevents the user clicking to input as move and then immediately restarting. This would cause a logic error when the server's move is received and shown. I was not able to create this but this could happen if the server traffic or latency (network speed) meant that the server response time was slow. (This validation is the corrected version after a failed test).

Feedback from Stakeholder

I then received feedback from my stakeholders (namely my parents and friends). The feedback broadly focused on the user interface.

My parents told me midway through the process that the board should be larger so it fills the screen. I also increased the font size of the 'X' and 'O' characters in the squares so that they fill the boxes and are more obvious.

Then when the program was finished I got more feedback from friends:

Regarding the alerts

- They told me not to use alerts as they were too jarring. They interrupted the flow of the webpage and were not what they (the stakeholders) were used to.
- They also recognised the issue where sometimes the alert for the 'game is over' occurs before the last move is shown

Regarding the computer's move:

- They thought it would be a good idea to add a small delay before the computer moves so that the user has time to finish making their move, see the computer's move, and react. They said that this would also make it seem more natural.
- They also said that it would be good if the text saying 'it's your turn' was dynamic and changed for when it was the computer's turn.

Other feedback included:

- Using a different background colour other than grey. This would help make the webpage more visually appealing and help differentiate the board in the foreground from the background
- There should be more of an effort to explain what the user should do when the page loads as it isn't necessarily intuitive.

The final piece of feedback was echoing my earlier concerns, as they said that it wasn't really a game as there was no way for them to win. In this way it wasn't fun.

Hopefully this issue will clear itself up when I move to a chess program. While it is an important technical achievement that my tic tac toe program is unbeatable, my chess program will be beatable. Not only will my algorithm not generate the theoretical best move but it will also be restricted in its thinking time as part of the difficulty settings.

It is worth also mentioning that I allowed my friends and parents to play with the program to help me create this feedback. In doing so it is important to recognise what they *didn't* say and what *didn't* go wrong.

- For one they at no point had any issue with the validation. Note this was before I accidentally created my reset button issue while

trying last minute to tackle another logic error. This means that the validation was working (at least it was when they used it)

- They also didn't have any problem with the page loading or the server response times which means the WSGI HTTP and WebSocket servers were working.
- They also didn't have to deal with any crashes on the server side, resulting in the server failing to respond with a move, or any errors client side that prevented them from playing the game. This is good as it gives me confidence that the code is robust.

Planned changes after stakeholder feedback

There are a variety of changes that I plan to either implement in this prototype or (more likely) take into account when building prototype 2.

- I will not use alerts. In this case I could have used a hidden HTML element containing dynamically altered text to show that the game was over. This would also deal with the alert before last move problem.
- I will also try to have some consideration for colour. While it is a weak point of mine I can get feedback as I develop to make a less starch looking website.
- I will add a delay (if needed on top of thinking time) to the chess program so that it feels like the computer is thinking and doesn't break immersion

Evaluation

Overall I am happy with the prototype as I think that this iteration has been a useful stepping stone. Its primary goal was to get me started with the connection between the front end and backend (WebSocket) as well as the backend logic. This included the WSGI HTTP server and a basic version of the minimax function. It would have been exceedingly hard to tackle the problem of chess without tackling this simpler problem first. This prototype has been a success because it has made me feel more able (on a technical level) to tackle a simplified version of chess form my second prototype.

The feedback from stakeholders has also been clear and useful. My program has a rugged and industrial look as the user interface was not the primary goal. However despite this, the scrutiny of stakeholders as to how specifically it could be better has helped me to learn more about how to make my program more usable. I will add what I have learned to my intentions surrounding usability in my planning document. So the prototype met its primary goal of helping me to develop the backend but, to be clear, the front end user interface failed to be user friendly.

I accept that this prototype looks very basic and tic tac toe is far from chess. However another factor that has justified such 'small' steps to chess has been the complexity of learning full stack web development (front and back end integrated) and the heuristics used to tackle chess. This is evidenced in the informal and 'rough' research and planning into how the minimax algorithm and WebSocket worked (I have included it at the end).

One of my aims in the analysis and planning stage was to make my program suitable for a variety of users. I would do this by making it work for a variety of aspect ratios.

I used the chrome dev tools to change the aspect ratio between my laptop and an Ipad Air and decided that there were differences in the dimensions I preferred.

```
td{  
  border: 5px solid black;  
  overflow: hidden;  
  
  /* laptop */  
  width: 21vh;  
  height: 21vh;  
  
  /* ipad air */  
  /* width: 24vw;  
  height: 24vw; */  
  font-size: 80px;  
  text-align: center  
}
```

There are diminishing returns to refining a prototype when I could just bear the refinements in mind for the next prototype. One such refinement I would make if I had more time would be to add to the CSS so that it automatically used different styles for different aspect ratios.

Another change that I would have added had I had more time to refine is server side validation. Due to browsers having different versions of JS installed or users being able to circumvent client side validation, I would like to add server side validation. This could for example ensure that the client js wasn't miss reporting the current game state to be more favourable to the user. This could be done with sessions, ensuring that the client game state is a legal move away from what it was after the server last moved. This would be of particular importance if the incentive to cheat existed. This could occur if I added a trophy / rank system or user vs user matchups (both of which are low priority features I decided weren't suited to my target user in my analysis).

Research and planning (please do tell me if this shouldn't be here)

(Minimax algorithm research to be included in backend planning document)

First Prototype:

My first prototype should be very simple and won't fully achieve all the design specifications. It is important however as it will aim to create some of the main mechanisms that will be needed by later versions. It should also be quick to develop so that I don't lose motivation or bite more than I can chew (try to do too much of the development at once). For this reason reusing code from other projects is essential. I also want to select a more simple game than chess.

Here is the idea:

I will make a tic tac toe / noughts and crosses game that will allow a user to play against a computer. The user interface will not be the focus of attention and so it doesn't need to be overly complicated.

Here is my first and then my improved second iteration of the user interface:

Done on paper insert images here

I will implement this 2nd version with HTML and JavaScript with very little in the way of CSS

The current board will be stored as a 3x3 array in JavaScript.

The user will always go first and will always be X

The board widget will be implemented as a table tag with 3 rows and 3 columns, this allows each individual cell to be a separate tag allowing me to see which cell has been clicked.

Validation will be needed when a square is clicked:

- It is impossible to give as input a square that is not one of these 9
- If game over do nothing
- Else if filled do nothing

To make a move this is what will happen (post validation):

- Disable the board (make it ignore further user input)
- Update the 3x3 2d array in js
- Update the board the user sees
- Make a request to the server to identify if the user has won
 - If user has won or the game is a draw do nothing else, leave the board disabled, make a message to the user accordingly
- Else request to the server providing the board currently. The server will respond with the new board after it has moves and whether or not it has won. Moves left are also given and if 0 the game is won by the computer.
 - If user has lost or the game is a draw do nothing else, leave the board disabled, make a message to the user accordingly
- Else update the 3x3 array stored in js
- Update the board widget
- Change the message to show it is the user's turn
- Re-enable the board widget to allow for the users input

Note how some of these operations like determining if the game is won could easily be done in client side JavaScript. I have chosen not to do this as in chess that is a more complex operation that the backend will handle. It is overkill to have the backend decide if a move is valid so this will be done in the client side JavaScript. In the final thing I intend for the server to give the frontend an array of all the users legal moves after it makes a move to avoid unnecessary api calls but to also avoid logic being done in the front end.

A full duplex bi-directional connection is not required and so a form of WebSocket is not required. Instead a restful API will be used using the HTTPS protocol.

The backend (server side code) will be written in python. It will setup a restful API and host the websites files using a simple flask server.

It would be easy to have the backend memorise a process of steps that would guarantee a win or draw but instead I will use the British Museum algorithm to generate a complete tree of depth 9. Minimax will be used to navigate this tree but no static evaluations are needed as all leaf nodes represent a win (score of 1) a draw (score of 0) or a loss (score of -1).

A tree of depth 9 with a branching factor of 9 decreasing by 1 each time will have a total number of static evaluation of approximately 360,000. However as the user is starting only a depth of 8 is required reducing static evaluation to 40,000.

So to be clear I will need to perform minimax at a depths (d) of: 8, 6, 4 and 2 and the branching factor (b) is decreasing by 1 each time.

This means the total computation (measured by total leaf nodes in each minimax call) will be:

$8! + 6! + 4! + 2! = 41066$ which is mostly calculating the first move (first move is 40320 steps).

The backend will require these functions:

(note I am not including the actual code to make the HTTPS restful API as this is very library and language specific, instead I am including all the logical functions)

- Determine if a game is over, still going, won or lost
- A representation of the board using a 3x3 2d array
- Minimax recursive function with alpha beta pruning to determine the computers best move
 - A method to determine all legal moves from a given game state (empty squares in array)
 - A method to make a copy of the current game state which has been changed to reflect a move being made

I want to follow the industry standard of test driven development for the chess engine. I will write a unit test for the various internal functions that the program uses: **DESIGN**

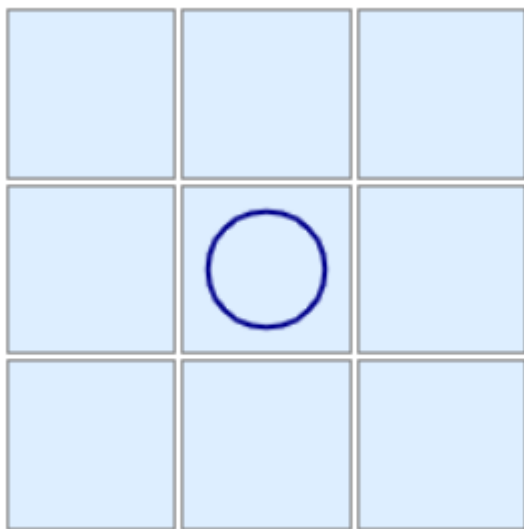
- Test if various boards are correctly determined to be wins, losses draws or unfinished
- The thing that is less obvious is how to test its tic tac toe ability. Here are the options I thought of:
 - Test that when random inputs are given the program is able to win and test that when it is against a tic tac toe algorithm (meant to represent optimal play) it can draw.
 - I could manually ply a bunch of games with it and ensure it is working then write the array of moves I made down and use it as a test to ensure the program makes the same moves as before

The best option I can think of is to make a function to score each possible legal move in a given game state. Therefore I can ensure that the minimax function always returns a move that is optimal or draws with other equivalent moves (due to symmetry)

I did also consider exploiting the symmetry of tic tac toe in order to reduce the total number of legal moves down to a subset of distinct moves.

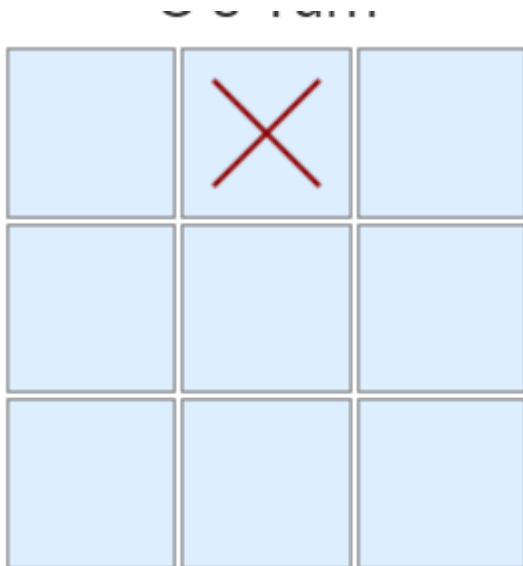
For example let's number squares from top to bottom left to right with numbers 1 to 9 (top left is 1, middle is 5, bottom right is 9).

In this game state:



Only squares 1 and 2 or equivalent need to be examined. This is because there is rotational, vertical, horizontal and diagonal symmetry.

In this game state:



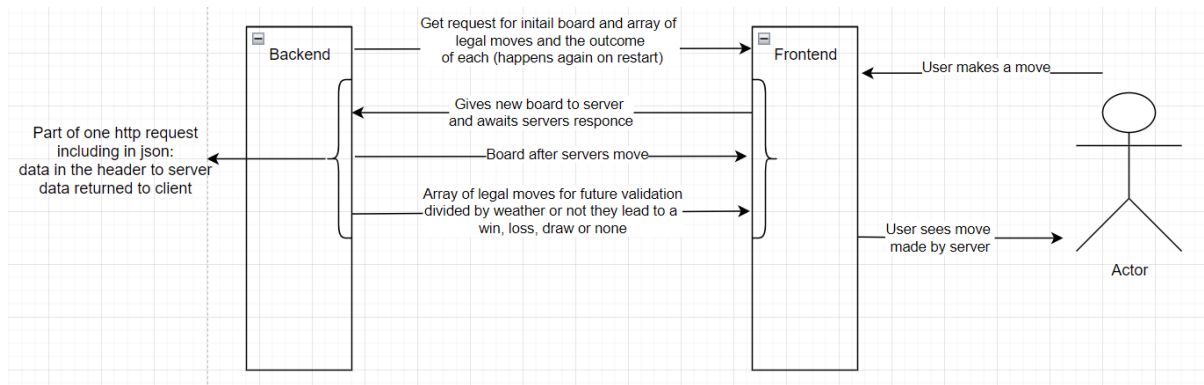
Only 1, 4,5,7 and 8 need to be examined due to vertical symmetry. I have decided to ignore this as 8! Is a manageable number for a computer to compute and in chess I cannot take advantage of symmetry at any point due to the king and queen.

Therefore I have the freedom to make the play experience more enjoyable by having the computer take a random move if there are multiple that are equivalent.

I have created a logging module to increase the transparency of my code. It allows me to log to both the console and a log file. It also includes a decorator that I can use to easily augment my functions to add logging to them to show their behaviour. I have optionally turned off the logging to console part to avoid filling the console with noise as I am unable to filter by importance level so that debug and info logs don't make it to the console, only warning error or critical.

I also invested a small amount of time trying to get web sockets working. I found it difficult as it is less commonly used than alternatives and so the documentation and online troubleshooting is less helpful. In doing so I also realised that I only ever need this communication to happen between the front and backend

My experiment with WebSocket was fruitful as I determined the exact nature of the connection between the front and backend



This is over the top for noughts and crosses as it is more complex due to all the logic being done by the backend including validation. This is unnecessary here as the logic is simple enough that it can be done client side. In my chess website I want to ensure that all the more complex logic is the sole responsibility of the backend.

I am still unsure about where to store the information of what the initial board state is. This could be in a json file that the frontend requests or it could be hard coded in variables in the JavaScript. Either way the client side JavaScript will need to know what legal moves the user can make and that they all result in the game continuing. Each of the 9 possible moves the user could try to make (by clicking the square) has one of these properties:

- Is illegal as the square is full
- Results in the game continuing (the only outcome that requires the server to be queried for its move)
- Results in the user winning
- Results in the user losing
- Results in a draw

This property will be stored for each move in a 3x3 array. It will start off before the user's first move as all moves resulting in the game continuing. Then the user will make a move. The client side JavaScript will determine that move is valid so it will be displayed and added to its copy of the board, it will also determine that the game should continue. It will send the server a copy of the board showing the user's move. The server will then respond with a new board showing its counter move and a new array of the properties of the 9 tiles that the user could try to move to (as listed above) this would be used for the validation of the user's next move.

I have created the webserver and I have it hosting 3 near empty files. A html file with the line 'hi from html' a css file that makes the background colour red and a JavaScript file that creates an alert "hello from JavaScript". Currently everything is loading and working.

I have had some time to think it through and even just the tiny start on this prototype has helped me see what logic should be completed on the frontend and the backend and how to connect them.

My initial idea for the communication between the frontend and the backend has some benefits. For instance it means that the frontend can validate a move without having to know anything about how the game of chess works. However the idea of the frontend being provided with a list of legal moves the user could make upon the computer making its move doesn't make much sense. For one in chess there could be multiple pieces that can move to a given square and there are all together more legal moves. Looking back I do think that a bidirectional communication would be better than trying to do all the information exchange in one http request

For example I could use some kind of socket setup to have it work this way:

User makes move (trigger / event) →

client asks server if it is valid (could include the from and to square) → server responds with whether or not it is valid

→ client asks if the game is over after the users move → server responds with details

→ client requests servers counter move → server responds with counter move

(→ client asks if game is over → server responds with details)

(repeat)

Perhaps this could be simplified so that there are less distinct requests and more is done in each request. The obvious alternative is to change my original idea so that more of the logic is done client side (validation of legal moves and determining if game over etcetera). For now I will try to stick with this (trying to imagine what is the best API for chess and not tic tac toe). At least the is game over can be one end point.

After my first attempt where I got stuck I will try again with web sockets but I might use a different library. This website covered a variety of other sources of how to use web sockets (<https://www.fullstackpython.com/websockets.html>)

I looked at this repository and will try to use it to help me get started:

<https://github.com/miguelgrinberg/Flask-SocketIO/tree/main/example>

This GitHub code repository is very useful as it provides examples of using web sockets including some more advanced features. It shows how the backend and the frontend can exchange data in json form with web sockets and it includes more complex concepts like sessions, emitting vs broadcasting and rooms. I could use rooms to allow 2 online players to

play chess. The example shows jQuery and a sockets library on the frontend and a flask WebSocket backend

So I will retry using WebSocket. The incentive to doing so is it will allow for a lower latency connection as a persistent full duplex direct TCP/IP connection is created after a HTTP handshake. This allows for low latency bi directional messages to be shared. Low latency means that I can have the frontend and backend make many requests to each other without performance lags (the alternative would be many https request which is slow or doing it all in one http request).

I have decided it wouldn't be worth the effort to create a test to see how well it plays noughts and crosses. Instead I will play the minimax computer and check that even if I play optimally it cannot lose. I will also not worry about the api connection between front and backend being similar enough to how I intend to do it with chess. What's important is that I can get it working with a similar approach to the one I intend to use for chess.

I have completed my frontend. It allows the user to go first as X and the server is O. It includes validation and a reset button. All the validation and determining if the game is over is completed by client side JavaScript. I am using WebSocket only to allow the server to communicate its move to the client. The server hosts the website and allows this WebSocket communication but doesn't yet have the ability to determine the appropriate move. At the moment it waits for me to type in a move before responding with it. I am currently using global variables and some pretty basic functions. This is ok as the first prototype only needs to be functional. These improvements can be made later.

I will now try to create a version of the British Museum algorithm in order to have the computer determine its own moves. It should in theory start by picking a path that guarantees is a draw or better. Then if the user slips up and it has an opportunity to guarantee a win it should be able to win.

I have written my min max function to allow the server to decide the best move. It is working but not picking an optimal move. It can be beaten which shouldn't be possible. In order to debug it I need to find ways to make the highly complex recursive function more transparent. One way to do this is to start with a half finished game where there are less moves left and run British museum minimax on this. I can use a modified copy of the function in a Jupiter notebook so that it generates a tree during execution. This will allow me to better see the logic error.

I have created a jupyter notebook and began to better understand the problem. I have discovered an issue that is likely not the only issue but needs to be fixed. The `is_game_over` functions in the server side python code and the client side JavaScript is flawed as I forgot to add a check for winning by diagonals. I will now fix this.

I thought that perhaps even after this fix there was still an issue with the server's game over function. I thought it may be inaccurately scoring game states that are over (incorrectly determining a win or loss). This function appeared to be working when tested for some sample inputs in the notebook. I will now try to see the functions decision making by visualising the decision tree using a library for graphs.

I have tried adding a decorator to the function to add print statements to illustrate when functions are called and when they finish, it is still hard to tell what is happening. From this I noticed that the alpha values were changing in a strange way but the beta values never were. I then checked the code and found this logic error:

```
beta = min(alpha, evaluation)
```

where the first argument to the min function should be beta not alpha. Let me try this fix.

This seems to have been the issue. It now seems to always win or draw when I play it now.

It is all fully working now, I had to modify some of the frontend validation as it was still failing to recognise diagonals but not it works.

First Prototype Done:

I have now completed my first prototype. It clearly doesn't meet all the success criteria and it clearly isn't chess. I think it was necessary as it is better to do more prototypes with a shorter gap / jump in progress between them. This ensures that motivation is maintained and ensures that I don't get stuck feeling like I have bitten off more than I can chew.

I used an algorithm to determine the best possible move from a given game state. I called it British Museum minimax. This is because it featured the approach of fully searching the decision tree all the way down to wins and losses (no static evaluations needed). This general approach is called the British Museum algorithm. To then analyse the tree I had constructed efficiently I used a variation of minimax to back propagate for the lead nodes to determine the best move. This is the reason for the name. Modifications on the standard minimax included alpha beta pruning, no depth argument or static evaluations and most importantly the algorithm actually returned both the score of a given game state and the child game state with the best score (gives you the best move as an output to the function).

~~Iteration 2 - Date xxxx~~

~~Aims for this iteration~~

Functionality that the prototype will have

Annotated code screenshots with description

Test Plan for this version

Test Results / Evidence

Feedback from Stakeholder

Changes/Fixes that I now plan to make to the design or code as a result
of testing and feedback

Evaluation

Iteration 3 – *Date xxxx*

Aims for this iteration

Functionality that the prototype will have

Annotated code screenshots with description

Test Plan for this version

Test Results / Evidence

Feedback from Stakeholder

Changes/Fixes that I now plan to make to the design or code as a result of testing and feedback

Evaluation