SDD MAJOR PROJECT

Alex Tsai

ALL PARTS

TABLE OF CONTENTS

Problem Statement

Adobe Flash Player was a freeware computer software widely used through a browser plugin to view interactive websites, stream videos and run web games. At the peak of Adobe Flash's popularity, it was revealed that more than 400 million out of over 1 billion connected desktops had updated to new versions of Flash Player within six weeks of release. However, the presence of maturing open source alternatives like HTML5, WebGL and WebAssembly led to the decreasing relevance of Flash Player. Ultimately, this led to Adobe ending support for Flash Player in 2020.

In combination with numerous security flaws discovered in Flash, many browsers ended support for Flash Player. Since then, many web game developers have switched to the mobile platform for both its lucrative market and lack of performance issues inherent to browser games. This has contributed to a slowdown of web game releases in recent years.

Web games are a special genre of game in that they are more accessible than any other game genre. This is because the only requirement to play a web game is a browser, while other games may require lengthy installations involving large volumes of dependencies and consequently high storage space demands. The slowdown of web game releases represents the loss of a convenient alternative to traditional application games.

Of the web games that have been released in recent years, the 'io' genre of games has become popular, characterised by being free-to-play, multiplayer, mechanically simple, and graphically minimalist. However, the difficulty of writing good multiplayer netcode has made releases in this genre limited.

To support new web games and prevent the extinction of web games as a genre, new solutions need to be developed.

Initial Design Specifications

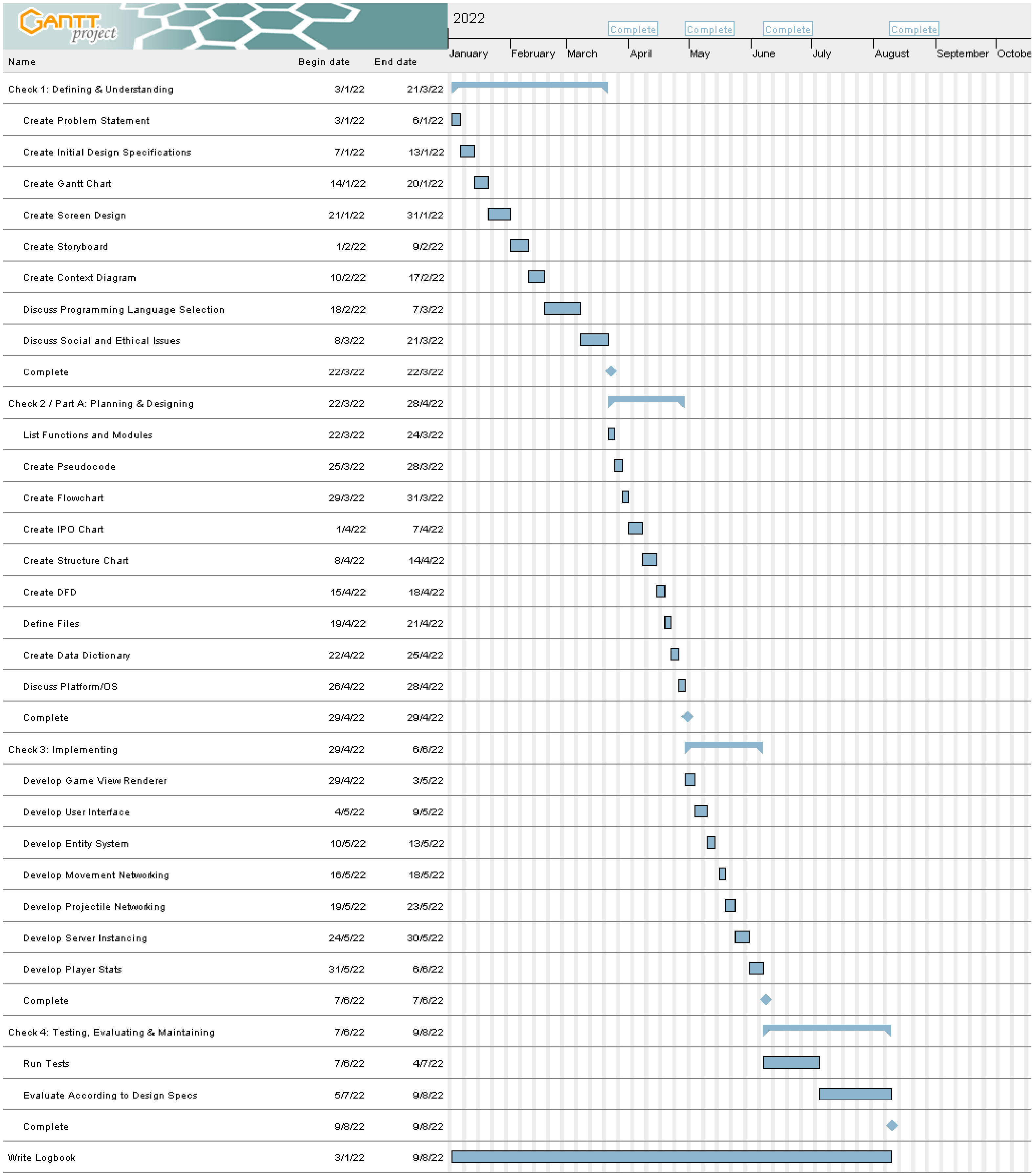
User Specifications

Ergonomic and intuitive controlsh
Simple and uncluttered interface
Clear and integrated tutorial
Judicious use of colours in interface
Compatibility with the user’s browser plugins
Compatibility with the user’s browser settings
Preserves the user’s privacy and security and asks for consent in data collection

Developer Specifications

Developed with structured development approach
Versatile and efficient data structures
Minimised errors, with useful error messages printed to browser console when needed
Efficient algorithms with good performance and no browser freezing
Code is internally documented well and input files are externally documented well
Variables have consistent and precise naming with appropriate data types and scope
Users are surveyed to improve quality and usability of system
System modelling tools are used to clarify the function of complex modules
Javascript ES6 classes are used to organise code.

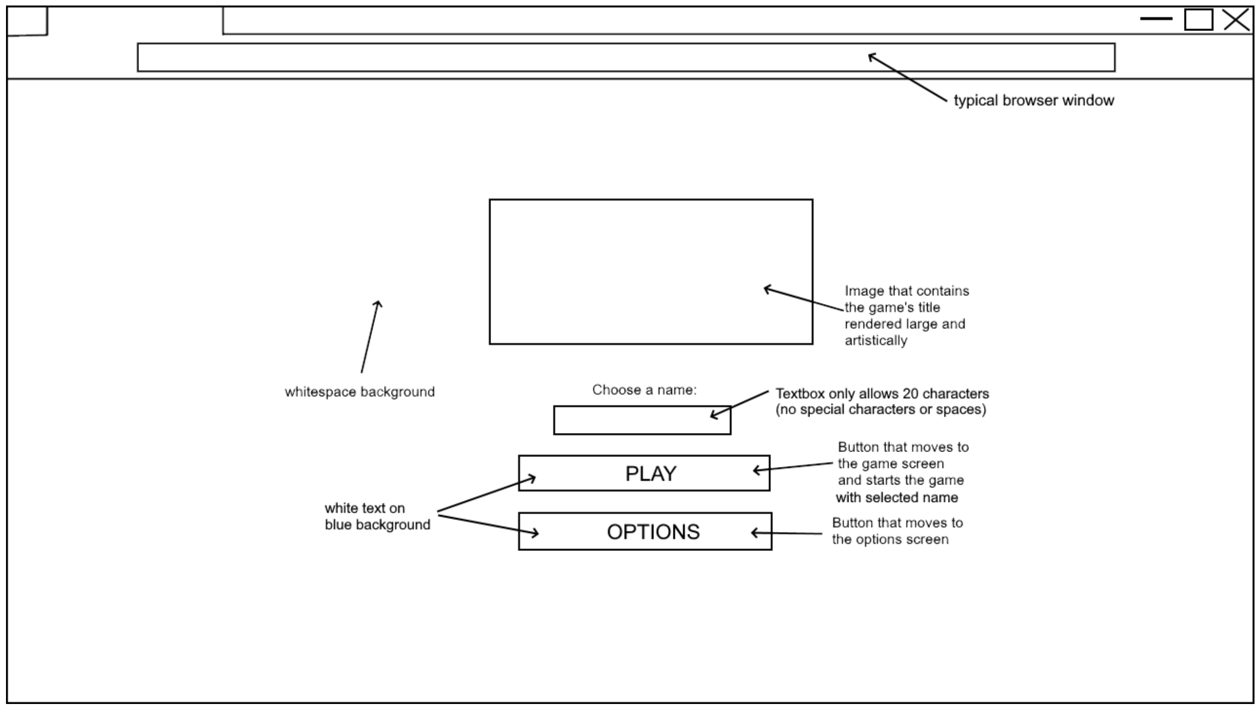
Gantt Chart



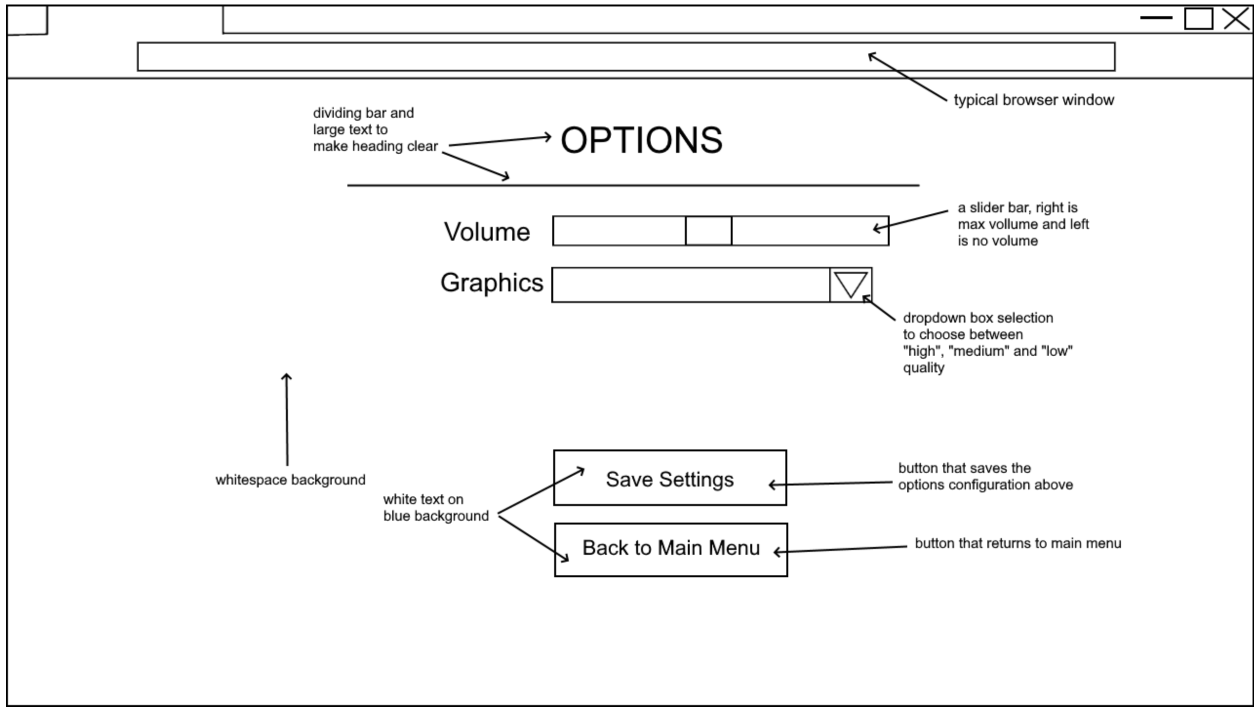
Name	Begin date	End date
Check 1: Defining & Understanding	3/1/22	21/3/22
Create Problem Statement	3/1/22	6/1/22
Create Initial Design Specifications	7/1/22	13/1/22
Create Gantt Chart	14/1/22	20/1/22
Create Screen Design	21/1/22	31/1/22
Create Storyboard	1/2/22	9/2/22
Create Context Diagram	10/2/22	17/2/22
Discuss Programming Language Selection	18/2/22	7/3/22
Discuss Social and Ethical Issues	8/3/22	21/3/22
Complete	22/3/22	22/3/22
Check 2 / Part A: Planning & Designing	22/3/22	28/4/22
List Functions and Modules	22/3/22	24/3/22
Create Pseudocode	25/3/22	28/3/22
Create Flowchart	29/3/22	31/3/22
Create IPO Chart	1/4/22	7/4/22
Create Structure Chart	8/4/22	14/4/22
Create DFD	15/4/22	18/4/22
Define Files	19/4/22	21/4/22
Create Data Dictionary	22/4/22	25/4/22
Discuss Platform/OS	26/4/22	28/4/22
Complete	29/4/22	29/4/22
Check 3: Implementing	29/4/22	6/6/22
Develop Game View Renderer	29/4/22	3/5/22
Develop User Interface	4/5/22	9/5/22
Develop Entity System	10/5/22	13/5/22
Develop Movement Networking	16/5/22	18/5/22
Develop Projectile Networking	19/5/22	23/5/22
Develop Server Instancing	24/5/22	30/5/22
Develop Player Stats	31/5/22	6/6/22
Complete	7/6/22	7/6/22
Check 4: Testing, Evaluating & Maintaining	7/6/22	9/8/22
Run Tests	7/6/22	4/7/22
Evaluate According to Design Specs	5/7/22	9/8/22
Complete	9/8/22	9/8/22
Write Logbook	3/1/22	9/8/22

Screen Design

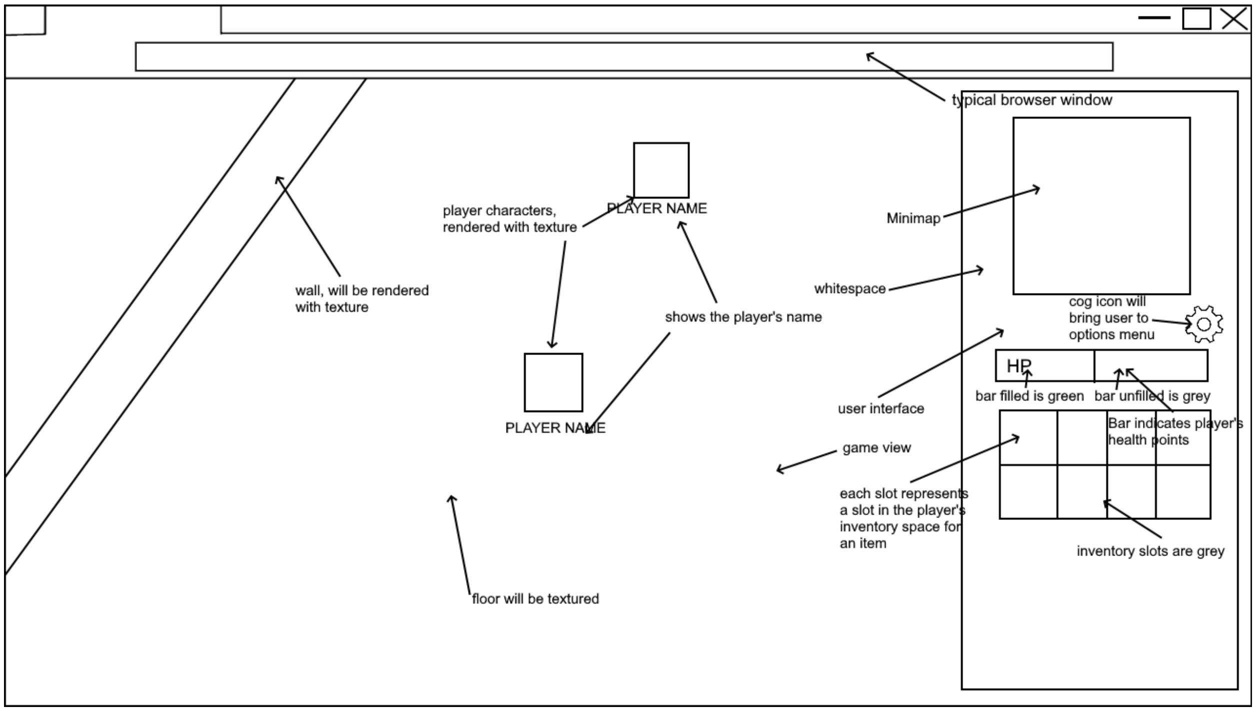
Main Menu


typical browser window
Image that contains the game's title rendered large and artistically
whitespace background
Choose a name:
Textbox only allows 20 characters (no special characters or spaces)
PLAY
Button that moves to the game screen and starts the game with selected name
white text on blue background
OPTIONS
Button that moves to the options screen

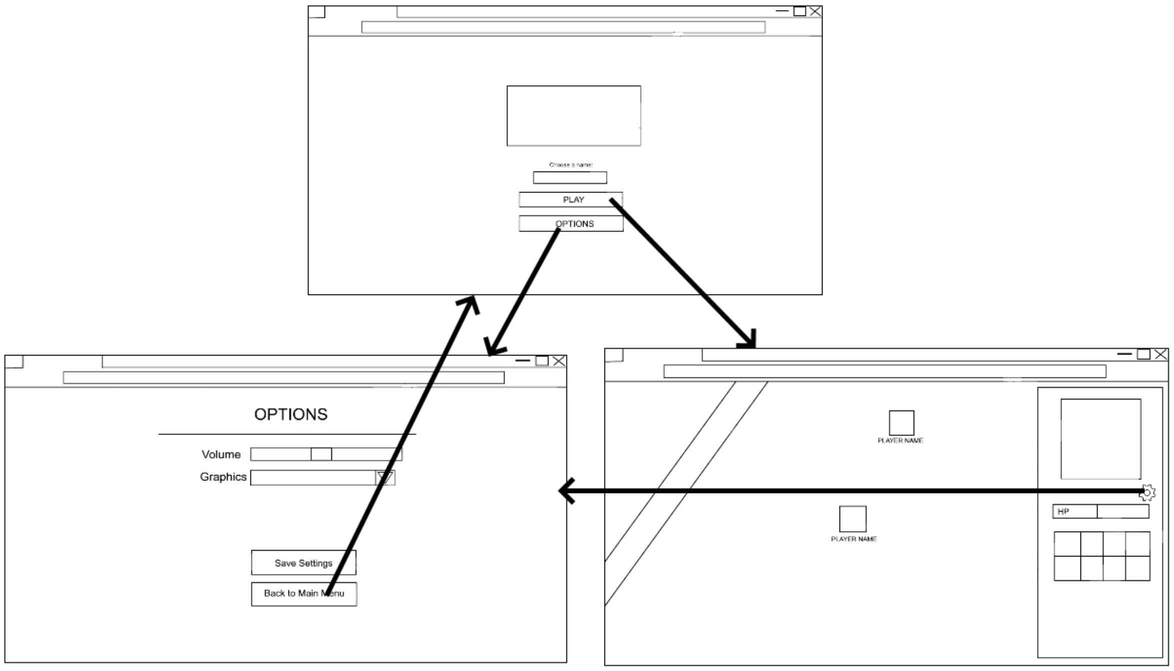
Options Menu


typical browser window
dividing bar and large text to make heading clear
OPTIONS
Volume
a slider bar, right is max vollume and left is no volume
Graphics
dropdown box selection to choose between "high", "medium" and "low" quality
whitespace background
white text on blue background
Save Settings
button that saves the options configuration above
Back to Main Menu
button that returns to main menu

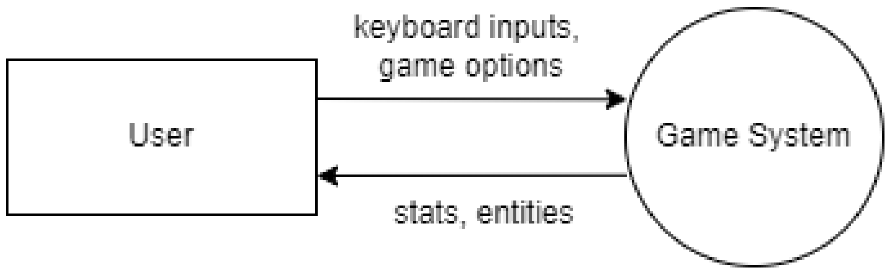
Game View Menu


typical browser window
player characters, rendered with texture
PLAYER NAME
Minimap
wall, will be rendered with texture
whitespace
shows the player's name
cog icon will bring user to options menu
HP
bar filled is green
bar unfilled is grey
Bar indicates player's health points
user interface
PLAYER NAME
game view
each slot represents a slot in the player's inventory space for an item
inventory slots are grey
floor will be textured

Storyboard



Context Diagram



Discussion of Programming Language

Discussion of Programming Language

The programming languages used will be Javascript, and SQL.

Javascript is a high-level dynamically-typed programming language available in many implementations. The implementations used in my project will be browser javascript and NodeJS.

Browser javascript is a frontend programming language widely supported by web browsers to run interactive and dynamic websites. Browser javascript will be used to write the entirety of the client. This is because browser javascript is highly compatible with the HTML I intend to use on the web page and javascript's large userbase will guarantee the existence of extensive online documentation, preventing undocumented errors from impeding development.

NodeJS is a backend implementation of javascript used for writing servers and other standalone applications. NodeJS will be used to write the entirety of the server. This is because utilising the same language and syntax on the server and client will make code more reusable and reduce redundancy. In addition, NodeJS has networking support in the form of the NodeJS socket-io module, reducing the need to deal with complicated low-level networking.

SQL will be used to interface with the server database, which will be made using the relational database system 'PostgreSQL'. This is because SQL is widely popular and hence both extensively documented on the internet and optimised. PostgreSQL also incorporates a concurrency management system that will allow multiple clients to write and read from the database's shared data simultaneously, a significant contribution to a multiplayer game.

Social and Ethical Issues

Copyright

The project will have to use assets that are not copyrighted. In addition to the copyrightability of the images and audio used, this also includes code libraries. The used code libraries should be checked to ensure that the project is abiding by the permissions granted by their software licences.

In addition, the project will need to preserve its own copyright– which will be inherently problematic as browsers expose the code behind a website to any user. This will make it easy for individuals to reverse-engineer the program and make potentially harmful modifications to either distribute to users as malware or exploit server logic to cheat the game. Use of code obfuscation and encryption of network traffic to manage this issue could also result in compatibility issues with browser plugins, while only delaying users from reverse engineering the project.

Security

As the project is network-based and requires users to transmit potentially sensitive account data over the internet to a central server, security of transmission is essential. If security is not used, it may be possible for malicious hackers to intercept sent packets of data and figure out a user's sensitive information. Hence, encryption will have to be used to protect users from the potential of this occurring.

Another potential security flaw is that the server will have to be hosted on a third-party server, given cost and tech issues. This means that a third-party could potentially have access to all data stored on the database of the game. The solution to this is also encryption, to prevent sensitive data from being stored as plaintext on the database.

Accessibility

Because of the long legacy of javascript as the programming language for the web, some web browsers do not have support for modern javascript features. This is problematic as the project uses some javascript features only released in 2015. Hence, the program's accessibility will necessarily be limited to an extent, although efforts can still be made to not use features that are deprecated and widely unsupported, or recently released.

Privacy

To make account logins more convenient, many websites including this project use ‘cookies’ to store an automatically expiring login session. In addition to storing, websites can also read what cookies are present on the user’s computer for the sake of collecting the user’s data for use in tracking and analytics. However, usage of the user’s cookies should require clear communication and the asking of consent so as to not be a breach of an individual’s privacy.

Modules

Name	Description
Controller	client module that contains methods used for the receiving of input from the browser, the storage of options relating to input, and the sending of input to the server.
ClientManager	client module that contains methods used for the updating of game entities, storage of existing game entities and the storage of a player id used to identify the player entity.
Renderer	client module that contains methods used to update the game canvas.
Database	server module that contains methods used to access the database for reading and writing.
World	server module that contains methods used to create and remove enemy AIs and entities within server ‘instances’.

Functions

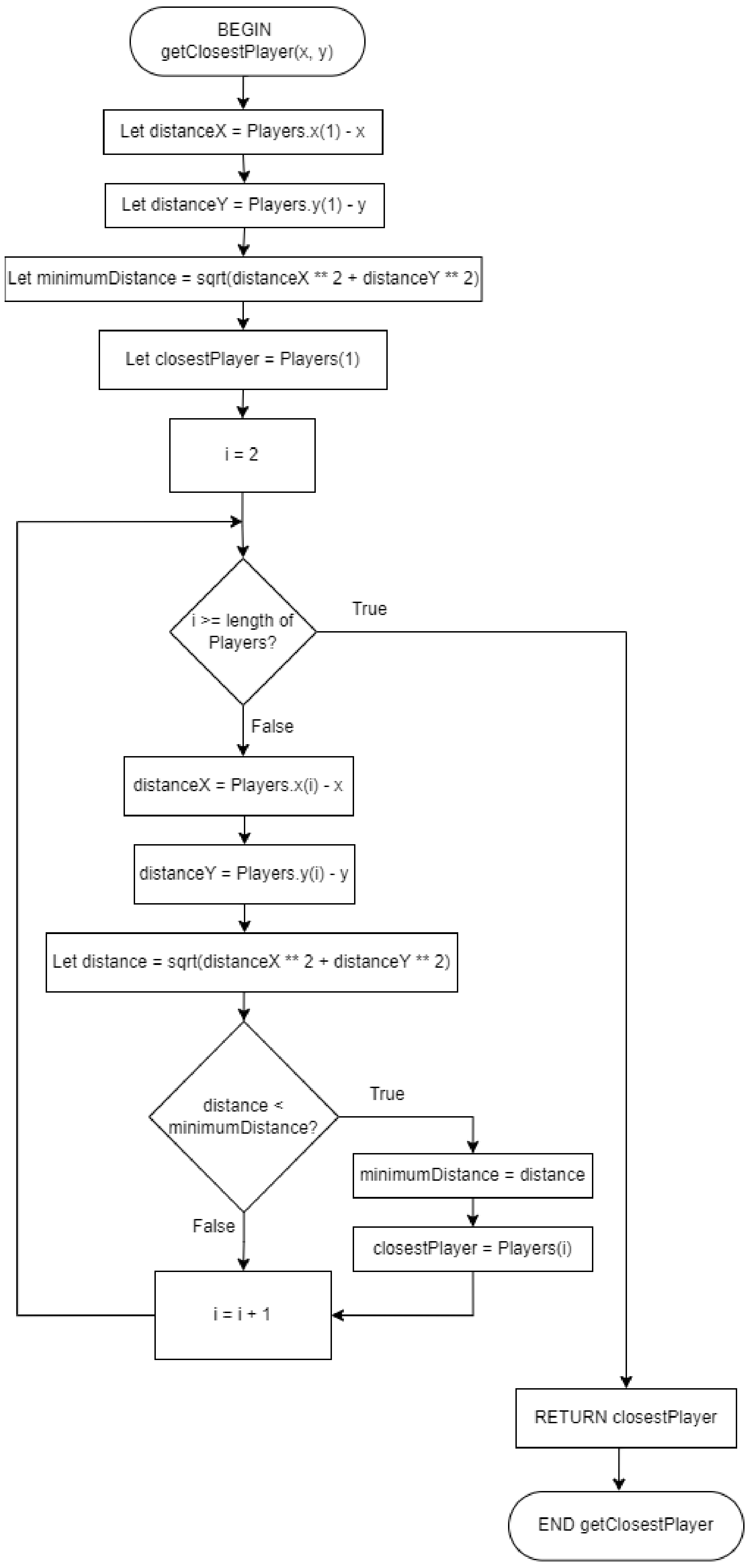
Name	Module	Description
getClosestPlayer	Enemy	given a position as a parameter, this function calculates the distances to each player entity and then finds the minimum distance to return the closest player.
fromDirection	Vector2	given an angle and direction, this function returns a 2d vector represented by the Vector2 class

rotate	Point	given an angle, this function rotates a 2d point about the coordinates (0,0) and returns the resulting point represented by the Point class
performServerReconciliation	Networking	applies given inputs to the player that haven't been processed by the server returning the resultant player entity represented by the Player class.
applyInput	Player	applies given inputs to a Player and returns the resultant player entity represented by the Player class.

Pseudocode for getClosestPlayer

```
BEGIN getClosestPlayer(x, y)
   Let distanceX = Players.x(1) - x
   Let distanceY = Players.y(1) - y
   Let minimumDistance = sqrt(distanceX ** 2 + distanceY ** 2)
   Let closestPlayer = Players(1)

   FOR i = 2 to length of Players STEP 1
      distanceX = Players.x(i) - x
      distanceY = Players.y(i) - y

      Let distance = sqrt(distanceX ** 2 + distanceY ** 2)
      IF distance < minimumDistance THEN
         minimumDistance = distance
         closestPlayer = Players(i)
      ENDIF
   NEXT i

   RETURN closestPlayer
END getClosestPlayer
```

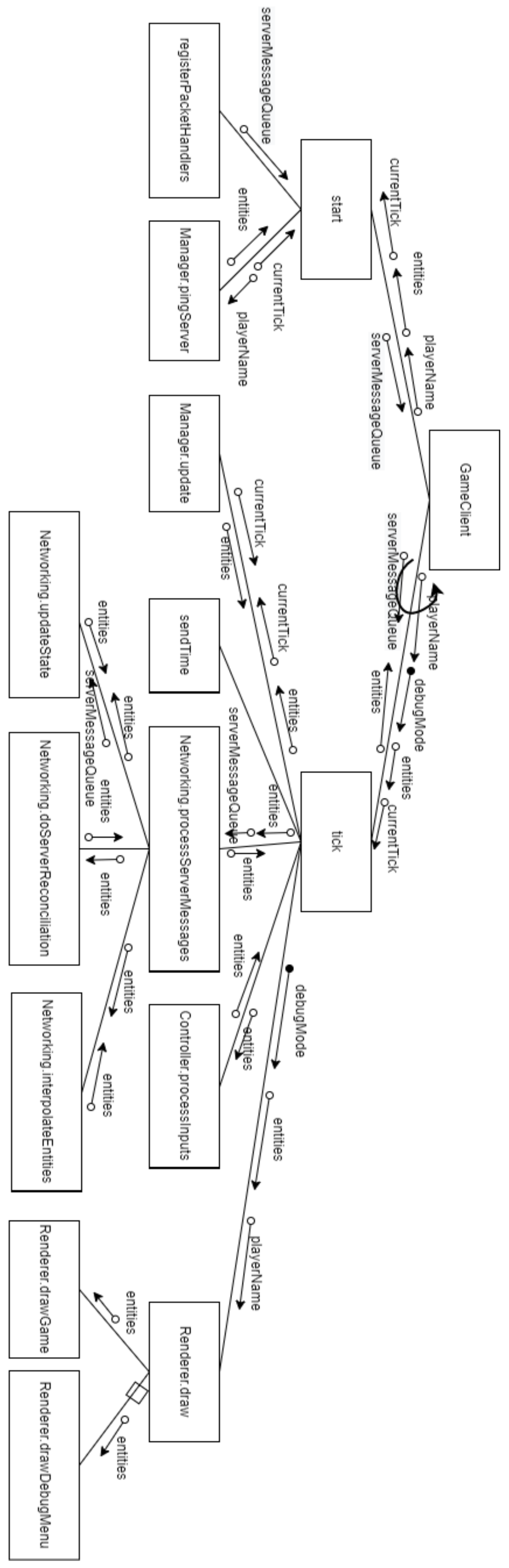
Algorithm Flowchart for getClosestPlayer


BEGIN getClosestPlayer(x, y)
Let distanceX = Players.x(1) - x
Let distanceY = Players.y(1) - y
Let minimumDistance = sqrt(distanceX ** 2 + distanceY ** 2)
Let closestPlayer = Players(1)
i = 2
i >= length of Players?
True
False
distanceX = Players.x(i) - x
distanceY = Players.y(i) - y
Let distance = sqrt(distanceX ** 2 + distanceY ** 2)
distance < minimumDistance?
True
minimumDistance = distance
closestPlayer = Players(i)
False
i = i + 1
RETURN closestPlayer
END getClosestPlayer

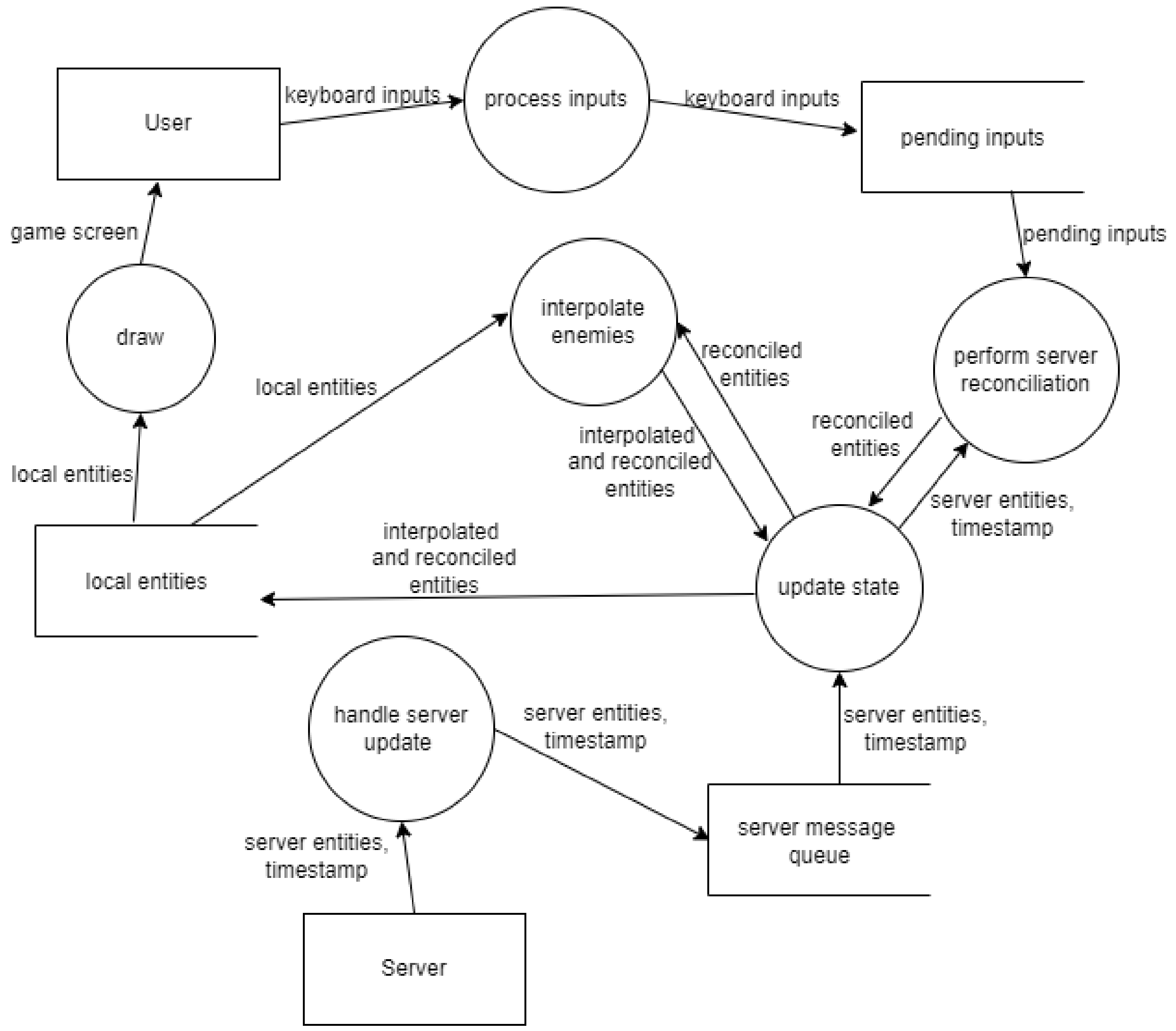
IPO Chart of Main Module

Input	Process	Output
‘GET’ request to website host	1. server responds with the website files stored locally on the server machine 2. a dictionary stored in the website’s code is used to determine the position, action and appearance of buttons for rendering	menu screen
player name, ‘start game’ request	1. ‘start game’ request is sent to server along with player name 2. server initiates a persistent connection with the browser client and sends world data 3. data is used to render entities on screen at their respective positions with their respective textures	game screen
keyboard inputs	1. browser calculates how long keys are pressed 2. browser processes keys that are only relevant to the client (like opening the options menu) 3. browser sends key press data to server 4. server processes keys and updates world state 5. server sends the updated world state as world data back to client 6. data is used to render entities on screen at their respective positions with their respective textures	game screen

Structure Chart



Data Flow Diagram for Client Networking Module



Files

Name	Type	Description

map-data.txt	TXT file	This input file is contained within a “world” folder and works in conjunction with world-data.json and enemy ai files located in the enemyAI directory to represent a loadable game world.
		
		The symbols stored plaintext in a map-data.txt file are used by the server to indicate the position and type of entities created in a particular game world upon loading.
		
		Example:
		`######`
		`#    #`
		`######`
		This would create entities in the pattern of a box.

world-data.json	JSON file	This input file is contained within a “world” folder and works in conjunction with map-data.txt and enemy ai files located in the enemyAI directory to represent a loadable game world.
		
		The world-data.json file has two keys “entityDict” and “spawn”.
		
		The “entityDict” key accesses a dictionary used to associate symbols in map-data.txt to entity types along with necessary parameters.
		
		The “spawn” key accesses the position at which the player entity is created upon loading in, represented as a two element array containing x and y.
		
		Example:
		{
		    "entityDict" : {
		        "#" : {
		            "entityType" : "Wall"
		        },
		        "/" : {
		            "entityType" : "Enemy",
		            "ai" : "chaser",
		            "size" : 32,
		            "speed" : 400
		        }
		    },

		"spawn" : [100, 100] } *If used in conjunction with the example in map-data.txt, that map would create a box of walls.*

<enemy ai name>.json	JSON file	There can be many inputted enemy ai files contained in the “enemyAI” folder of a “world” folder, working in conjunction with map-data.txt and world-data.json to represent a loadable game world.
		
		An enemy ai file is referred to by name in a world-data.json file under the ‘ai’ parameter of an entity type with the type “Enemy”.
		
		The enemy ai file contains ‘behaviours’ that abstractly describes an enemy’s ai by a declarative programming language that expresses the logic of an enemy without describing the exact computational steps.
		
		An enemy ai file has three keys “states”, “defaultState” and “projectiles”.
		
		The “states” key accesses a dictionary associating states to their ai behaviours. These behaviours are stored in an array and processed iteratively when an enemy has the respective state.
		
		The “defaultState” key accesses an enemy’s ‘default state’ upon creation.
		
		The “projectiles” key accesses a dictionary associating the names of an enemy ai’s projectile shooting patterns to a description of the projectile pattern. This description details the pattern in which projectiles are fired and the characteristics of the projectiles fired. These projectiles can be referred to in an enemy’s behaviour.
		
		Example:
		{
		  "states" : {
		    "stationaryShoot" : {
		      "behaviour" : ["shoot radialShotgun 2", "stateChangeTime waitTimer 6 wait"]
		    },
		    "wait" : {
		      "behaviour" : ["stateChangeTime waitTimer 6 stationaryShoot"]

		```
		        }
		    },
		    "defaultState" : "stationaryShoot",
		    "projectiles" : {
		        "radialShotgun" : {
		            "type" : "Radial",
		            "shotCount" : 1,
		            "direction" : 90,
		            "projDesc": {
		                "target" : "players",
		                "speed" : 20,
		                "size" : 16,
		                "lifetime" : 3000,
		                "damage" : 5
		            }
		        }
		    }
		}
		```
		 This describes a basic enemy that shoots 3 times over 6 ticks and then waits 6 ticks before repeating.

Data Dictionary

Data item	Data type	Format	Number of bytes required for storage	Size for display	Description	Example	Validation
playerName	string		20	20	A unique player-given name	Name123	No special symbols or spaces. Name must not exist already.
id	integer	NNNNNN	6	6	A unique integer used to identify entities instances.	000123	Id must not exist already.
x	floating point	NNNN.N	5	6	Represents an entity's position on the x axis	1234.5	Must be greater than 0 and less than 10000.
y	floating point	NNNN.N	5	6	Represents an entity's position on the y axis	1234.5	Must be greater than 0 and less than 10000.
debugMode	boolean	X	1	1	Whether to draw a debug menu	true	
svKeys	Array(string)		10 * number of keys	10 * number of keys	The names of keyboard inputs that are sent to the server	KeyW KeyA KeyS KeyD	
mouseHolding	boolean	X	1	1	Whether the mouse is being held	true	
startTime	integer		13	13	milliseconds since unix epoch at time of starting the game	1650889296211	Must be greater than 0.
Profile	record		128	128	server record containing information about a player		

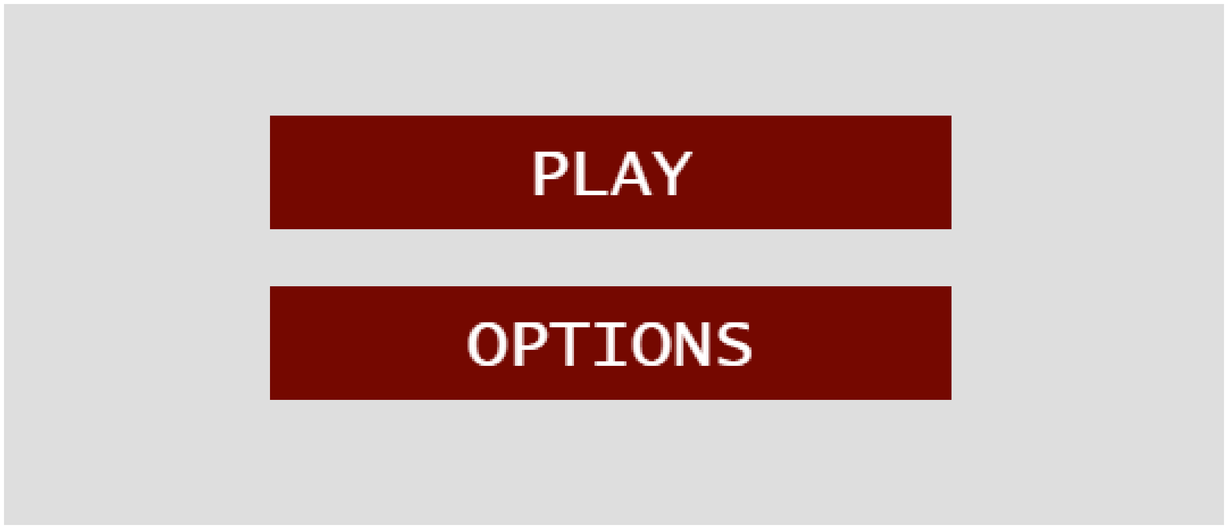
Platform/OS Considerations

The game's primary requirement is the ability to run modern javascript through a browser. By modern javascript, this project refers to ES6 javascript, a major revision to the programming language in 2015. According to the documentation for ES6 javascript, the browser requirements are as follows: Google Chrome 58, Edge 14, Firefox 54, Safari 10 and Opera 10. Looking at the system requirements for Google Chrome 58, the earliest Windows version required is Windows 7, the earliest Mac version is OS X El Capitan 10.11, and the earliest versions of linux required are 64-bit Ubuntu 18.04+, Debian 10+, openSUSE 15.2+, or Fedora Linux 32+. These operating systems should all be capable of running the game.

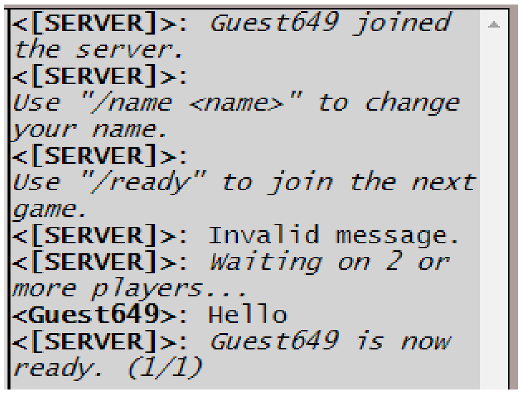
Another requirement is the memory to store the game's code. As a browser game, the browser needs to acquire the game's files from the web server and store the code in memory for execution. According to the inbuilt browser task manager on chrome, this takes roughly 32 megabytes of memory. Given that most modern computers have at least 4GB RAM, this should be a negligible amount of memory for most users and not a significant system requirement.

Additionally, the game requires a network connection. Given the use of a free unstable hosting platform, latency has been taken into account when designing the game's netcode– up to around 500ms of delay, the game is smooth enough to be playable. However, high network jitter (volatile latency) can result in unexpected behaviours with multiple packets being received at the same time or received out of order. The average Australian internet speed is 58.83Mbps download and 21.44Mbps upload, which should be adequate for playing the game with around 300ms of delay. Hence, the main system requirement is decent network quality to prevent high net jitter.

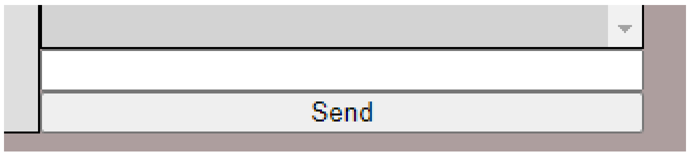
Choice of User Interface Items Selected



In the main menu, I utilise buttons as the user interface item to navigate into the different screens of the program. This is because the simplicity and ease of styling a button assists in creating an intuitive user interface.

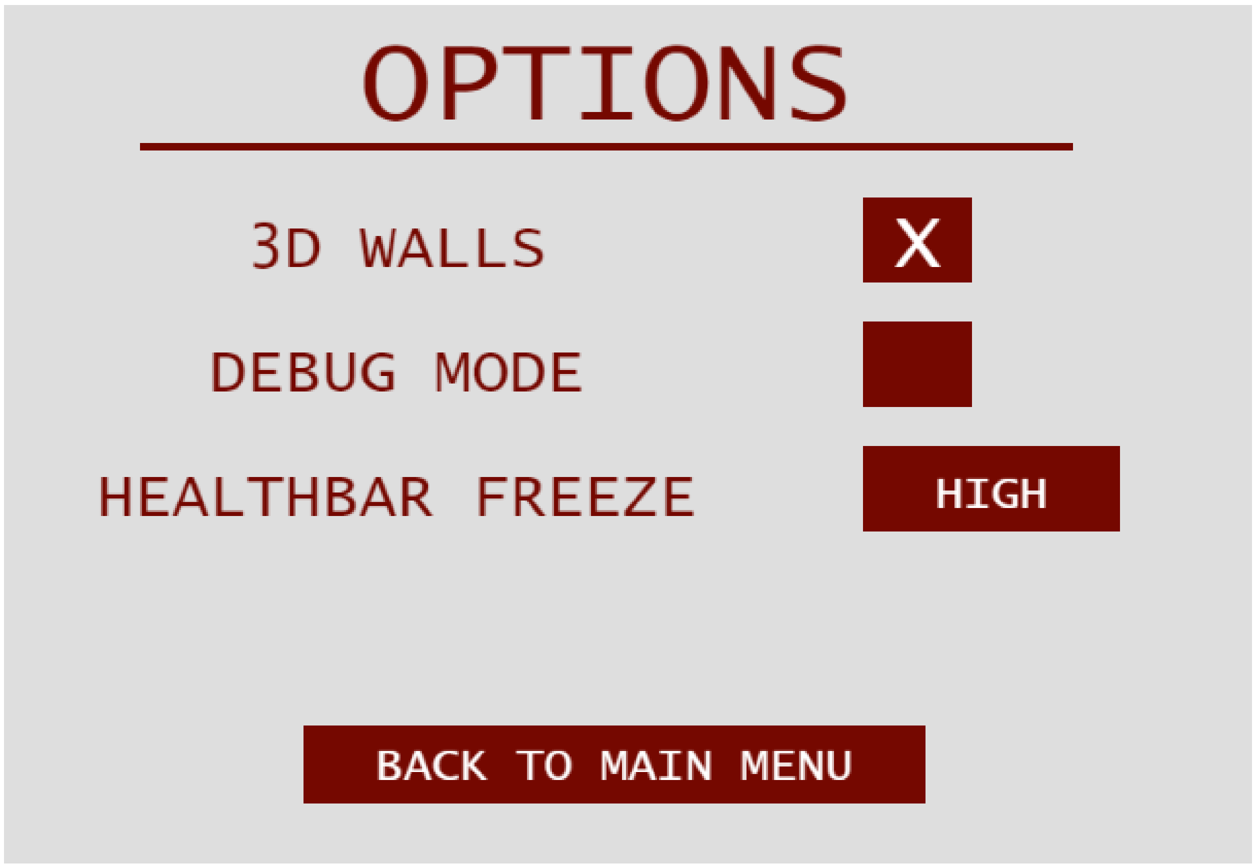


```
<[SERVER]>: Guest649 joined the server.
<[SERVER]>:
Use "/name <name>" to change your name.
<[SERVER]>:
Use "/ready" to join the next game.
<[SERVER]>: Invalid message.
<[SERVER]>: Waiting on 2 or more players...
<Guest649>: Hello
<[SERVER]>: Guest649 is now ready. (1/1)
```



The chat menu in the game screen incorporates a list of messages, a textbox and a button titled 'send'. The usage of this chat menu should be intuitive to users because it is a common design pattern. The user utilises this chat menu to issue commands, communicate with other players, and receive information from the server. This will reduce the need to create additional user interface elements such as textboxes and buttons, which may contribute to crowdedness of the screen.

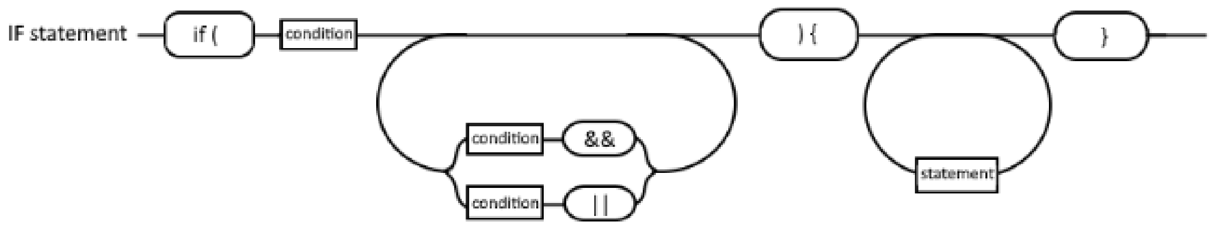
The name of the sender is enclosed within the angle brackets ("<name>"). Names of users can only allow for alphanumeric characters, so the "[SERVER]" message sender and italicised text immediately distinguishes special system messages that convey information about the game.

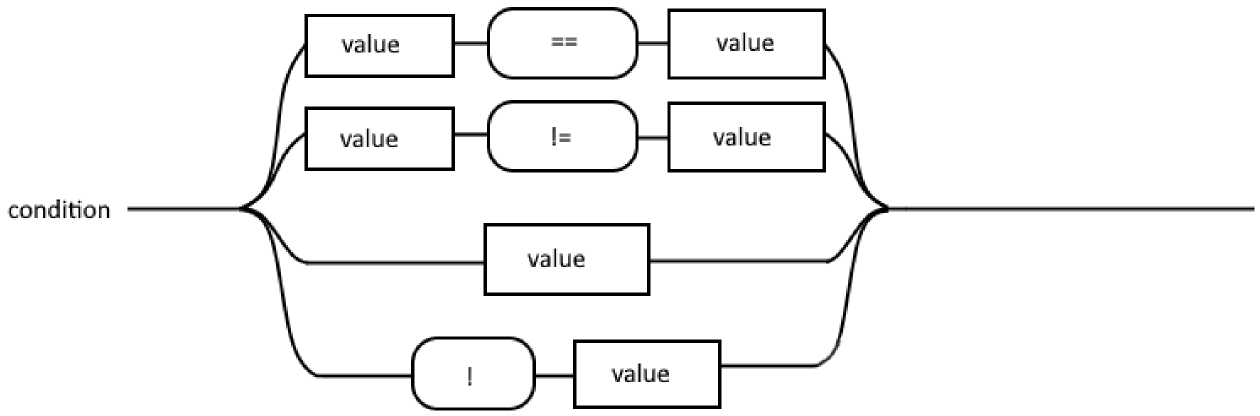


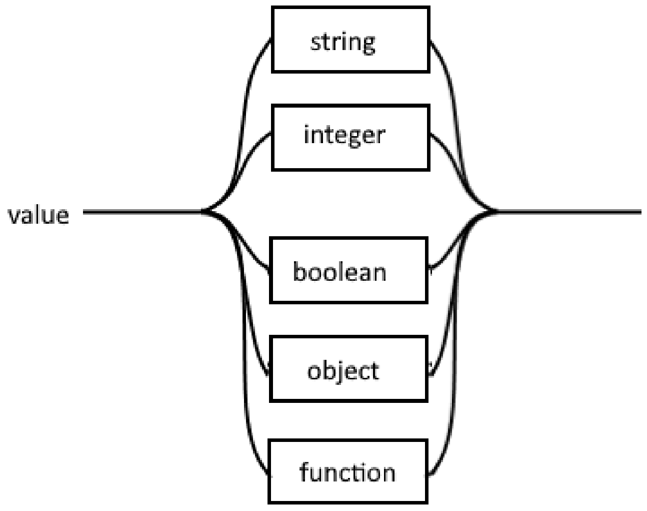
In the options menu, the user interface element used for each setting is dependent on what is most suitable for the data type required. For boolean settings, checkboxes are used, as seen in the above screenshot for the ‘3D walls’ and ‘debug mode’ options. This is because of the checkbox’s simplicity and intuitive usage. For multi-value settings such as the ‘health bar freeze’ option above, a button toggles between ‘high, medium, and low’ on every click. This is better than a dropdown menu because the low amount of selectable values is easily navigated using purely clicks, while a dropdown menu requires clicking to open the dropdown and then using mouse movement to select the desired option.

EBNF and Railroad Diagrams for the Javascript If Statement

if statement = if "("<condition>{(|| <condition>)|(&& <condition>)}")" "{"{<statement>}"}"
condition = (<value> == <value>) | (<value> != <value>) | <value> | !<value>
value = <string> | <integer> | <boolean> | <object> | <function>







Test Data for the applyInput module

Code for applyInput:

```
applyInput(rot, inputs, wallEntities) {
    let movementVec = Vector2.ZERO();
    console.log(inputs);
    if (inputs['KeyW'] && inputs['KeyW'] <= 1) {
        movementVec.add(Vector2.fromDirection(-rot+90, -inputs['KeyW']))
    }
    if (inputs['KeyS'] && inputs['KeyS'] <= 1) {
        movementVec.add(Vector2.fromDirection(-rot+90, inputs['KeyS']))
    }
    if (inputs['KeyD'] && inputs['KeyD'] <= 1) {
        movementVec.add(Vector2.fromDirection(-rot, inputs['KeyD']))
    }
    if (inputs['KeyA'] && inputs['KeyA'] <= 1) {
        movementVec.add(Vector2.fromDirection(-rot, -inputs['KeyA']))
    }
    movementVec.normalize(this.speed).divide(100).fix();
    this.x += movementVec.x;
    for (const wallEntity of wallEntities) {
        if (this.detectEntityCollision(wallEntity)) {
            if (movementVec.x > 0) {
                this.right = wallEntity.left
            }
            if (movementVec.x < 0) {
                this.left = wallEntity.right
            }
        }
    }
    this.y += movementVec.y;
    for (const wallEntity of wallEntities) {
        if (this.detectEntityCollision(wallEntity)) {
            if (movementVec.y > 0) {
                this.bottom = wallEntity.top
            }
            if (movementVec.y < 0) {
                this.top = wallEntity.bottom
            }
        }
    }
}
```

Input	Expected Output	Actual Output	Correct?
Send a W key input with time held as 0.05 seconds, 0 rotation and player speed set to 500.	from x=0, y=0, the player entity moves 0.05 * 500 = 25 units to x=0 y=25	From x=0, y=0, the player entity moves forward to x=0 y=25	correct
Send a W key input with time held as 0.05 seconds, 90 rotation and player speed set to 500.	from x=0, y=0, the player entity moves 0.05 * 500 = 25 units to x=-25 y=0	From x=0, y=0, the player entity moves forward to x=-25 y=0	correct
Send a W key input with time held as 0.05 seconds, 45 rotation and player speed set to 500.	from x=0, y=0, the player entity moves 0.05 * 500 = 25 units to x=sqrt((25^2)/2) y=sqrt((25^2)/2), or x=-17.6, y=-17.6	From x=0, y=0, the player entity moves forward to x=-17.6 y=-17.6	correct
Send a W key input with time held as 0 seconds, 0 rotation and player speed set to 500.	Nothing happens.	Nothing happens.	correct
Send a W key input with time held as -0.05 seconds, 0 rotation and player speed set to 500.	from x=0, y=0, the player entity moves -0.05 * 500 = 25 units to x=0 y=-25	from x=0, y=0, the player entity moves -0.05 * 500 = 25 units to x=0 y=-25	correct
Send a W key input with time held as a string, 0 rotation and player speed set to 500.	Nothing happens.	Nothing happens.	correct
Send a W key input with time held as 0.05 seconds,	Nothing happens.	player entity's location set to x=NaN, y=NaN.	correct

rotation as a string and player speed set to 500.			
Send a W key input with time held as 0.05 seconds, 0 rotation and player speed set to 10000.	player moves on their screen, but is quickly rebounded to where they should be if their speed was accurate: 0.05 * 500 = 25 units to x=0 y=25	player moves on their screen, but is quickly rebounded to where they should be if their speed was accurate: 0.05 * 500 = 25 units to x=0 y=25	correct
Send a W key input with time held as 0.05 seconds, 0 rotation and player speed set to -500.	player moves on their screen, but is quickly rebounded to where they should be if their speed was accurate: 0.05 * 500 = 25 units to x=0 y=25	player moves on their screen, but is quickly rebounded to where they should be if their speed was accurate: 0.05 * 500 = 25 units to x=0 y=25	correct
Send a W key input with no parameters.	Nothing happens.	Error produced in browser console. 'Uncaught ReferenceError: inputs is not defined'	incorrect

Error Checking

Stubs

In the following code, the “detectCircleCollision” is a stub used to help the developer to verify the flow of execution in the “tick” subroutine while the “detectCircleCollision” subroutine is being coded.

```
detectEnemyCollision(entity)
    return false;
}
```

```
tick() {
    ...
    for (const projectile of this.projectiles) {
        ...
        for (const enemy of Object.values(this.enemies)) {
            ...
        }
        if (projectile.detectCircleCollision(this.player) &&
            projectile.target == ENTITY_CATEGORY.players) {
            ...
        }
    }
    ...
}
```

Flags

In the following code, the self.gameStarted flag is used to confirm that the startGame subroutine has been executed.

```
startGame(self) {
    clearInterval(self.interval)
    self.game.start();
    self.gameStarted = true;
    self.renderLoad(self)
}
```

Debugging Output Statements

In the following code, a debugging output statement is used to ensure that the onWorldChange function is called, the message ‘hi’ being printed to the browser console if the call succeeded.

```
const onWorldChange = function(playerId) {
    console.log('hi')

    manager.totalBossHp = 0;
    manager.damageDone = 0;
    networking.svMsgQueue = []
    manager.clearEntities();
    manager.playerId = playerId;
}
```

Desk Check for Entity.closestPlayer

Code for Entity.closestPlayer:

```
closestPlayer(playerEntities) {
    let min = Infinity
    let closestPlayerObj;
    for (let player of playerEntities) {
        let dist = Math.sqrt((player.x-this.x)**2+(player.y-this.y)**2);
        if (dist < min) {
            min = dist;
            closestPlayerObj = player;
        }
    }
    if (!closestPlayerObj) return;
    return {
        distance: min,
        closestPlayer: closestPlayerObj
    };
}
```

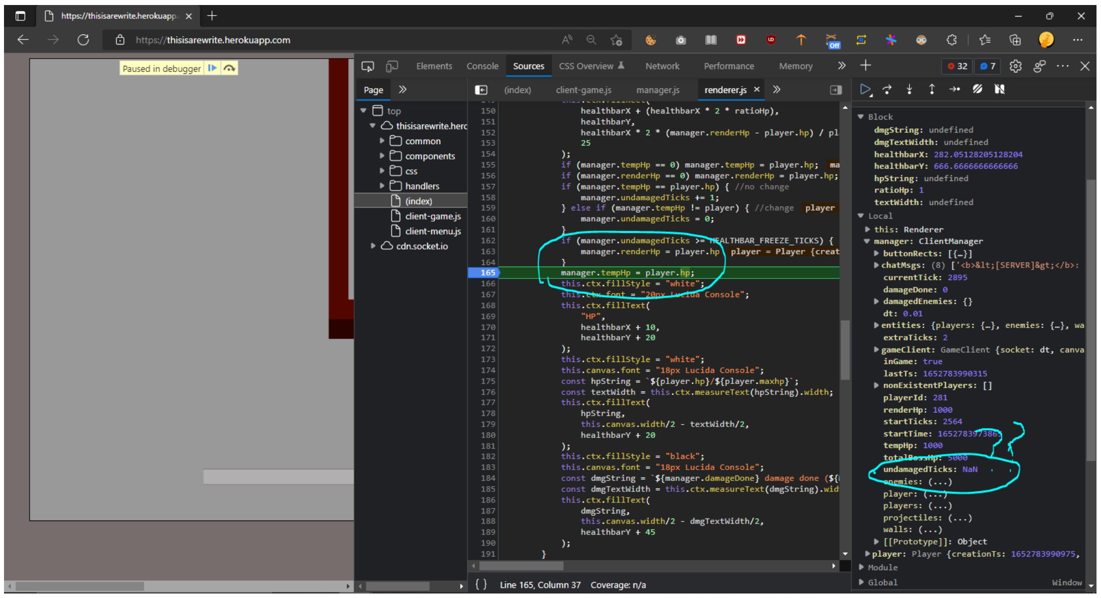
Suppose an array of two player objects is passed as the playerEntities parameter. The first player object Player1 is at x=0 y=0, the second Player object Player2 is at x=20 y=20 and the local Entity object is at x=9, y=9.

min	closest Player Obj	Entity.x	Entity.y	Player	player.x	player.y	dist	return
Infinity		9	9					
				Player1	0	0	12.727	
12.727	Player1							
				Player2	20	20	28.284	
								{ distance: 12.727, closestPlayer: Player1 }

Use of Breakpoints, Traces, Single-line stepping,

Breakpoints

The chrome debugger can be used to set a breakpoint, which will stop program execution upon arriving at the breakpoint. By setting a breakpoint on "manager.tempHp = player.hp", I was able to discover an error in the code that can lead to a NaN value for the variable "manager.undamagedTicks".



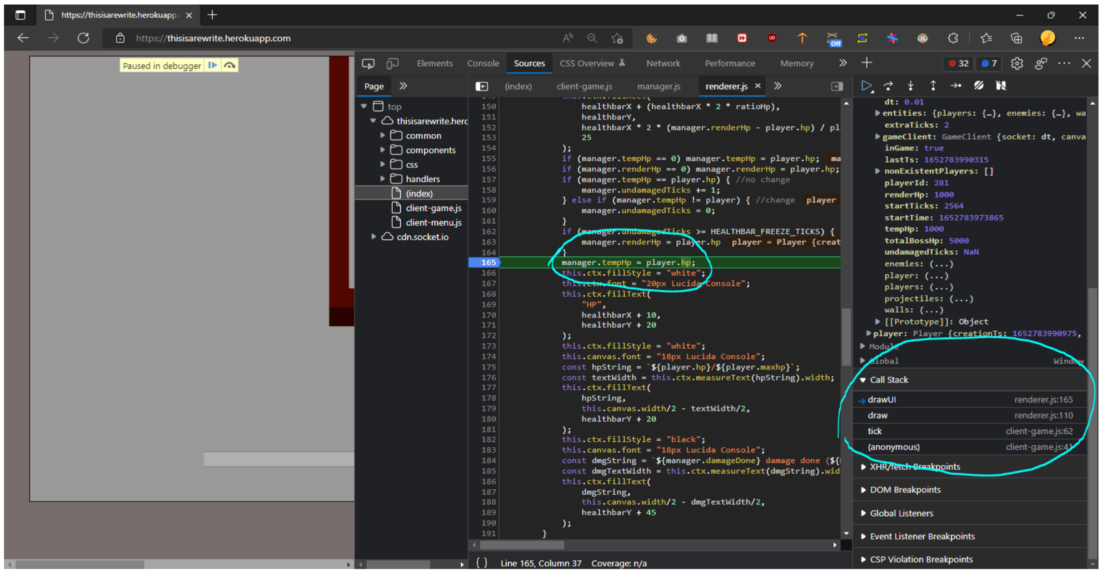
```
        healthbarX + (healthbarX * 2 * ratioHp),
        healthbarY,
        healthbarX * 2 * (manager.renderHp - player.hp) / pl
        25
    );
    if (manager.tempHp == 0) manager.tempHp = player.hp;  ma
    if (manager.renderHp == 0) manager.renderHp = player.hp;
    if (manager.tempHp == player.hp) { //no change
        manager.undamagedTicks += 1;
    } else if (manager.tempHp != player) { //change  player
        manager.undamagedTicks = 0;
    }
    if (manager.undamagedTicks >= HEALTHBAR_FREEZE_TICKS) {
        manager.renderHp = player.hp  player = Player {creat
    }
    manager.tempHp = player.hp;
    this.ctx.fillStyle = "white";
    this.ctx.font = "20px Lucida Console";
    this.ctx.fillText(
        "HP",
        healthbarX + 10,
        healthbarY + 20
    );
    this.ctx.fillStyle = "white";
    this.canvas.font = "18px Lucida Console";
    const hpString = `${player.hp}/${player.maxhp}`;
    const textWidth = this.ctx.measureText(hpString).width;
    this.ctx.fillText(
        hpString,
        this.canvas.width/2 - textWidth/2,
        healthbarY + 20
    );
    this.ctx.fillStyle = "black";
    this.canvas.font = "18px Lucida Console";
    const dmgString = `${manager.damageDone} damage done (${
    const dmgTextWidth = this.ctx.measureText(dmgString).wid
    this.ctx.fillText(
        dmgString,
        this.canvas.width/2 - dmgTextWidth/2,
        healthbarY + 45
    );
}
```

Line 165, Column 37   Coverage: n/a

```
▼ Block
    dmgString: undefined
    dmgTextWidth: undefined
    healthbarX: 282.05128205128204
    healthbarY: 666.6666666666666
    hpString: undefined
    ratioHp: 1
    textWidth: undefined
▼ Local
  ▶ this: Renderer
  ▼ manager: ClientManager
    ▶ buttonRects: [{…}]
    ▶ chatMsgs: (8) ['<b>&lt;[SERVER]&gt;</b>:
      currentTick: 2895
      damageDone: 0
    ▶ damagedEnemies: {}
      dt: 0.01
    ▶ entities: {players: {…}, enemies: {…}, wa
      extraTicks: 2
    ▶ gameClient: GameClient {socket: dt, canva
      inGame: true
      lastTs: 1652783990315
    ▶ nonExistentPlayers: []
      playerId: 281
      renderHp: 1000
      startTicks: 2564
      startTime: 1652783973865
      tempHp: 1000
      totalBossHp: 5000
      undamagedTicks: NaN
      enemies: (...)
      player: (...)
      players: (...)
      projectiles: (...)
      walls: (...)
    ▶ [[Prototype]]: Object
  ▶ player: Player {creationTs: 1652783990975,
▶ Module
▶ Global                                  Window
```

Traces

By using the chrome debugger to set a breakpoint, a developer can also view the call stack, which acts as a trace of function calls that have occurred to arrive at the breakpoint. In this case, I have determined that the flow of execution is correct. The client tick function calls the draw function which then calls the drawUI function.



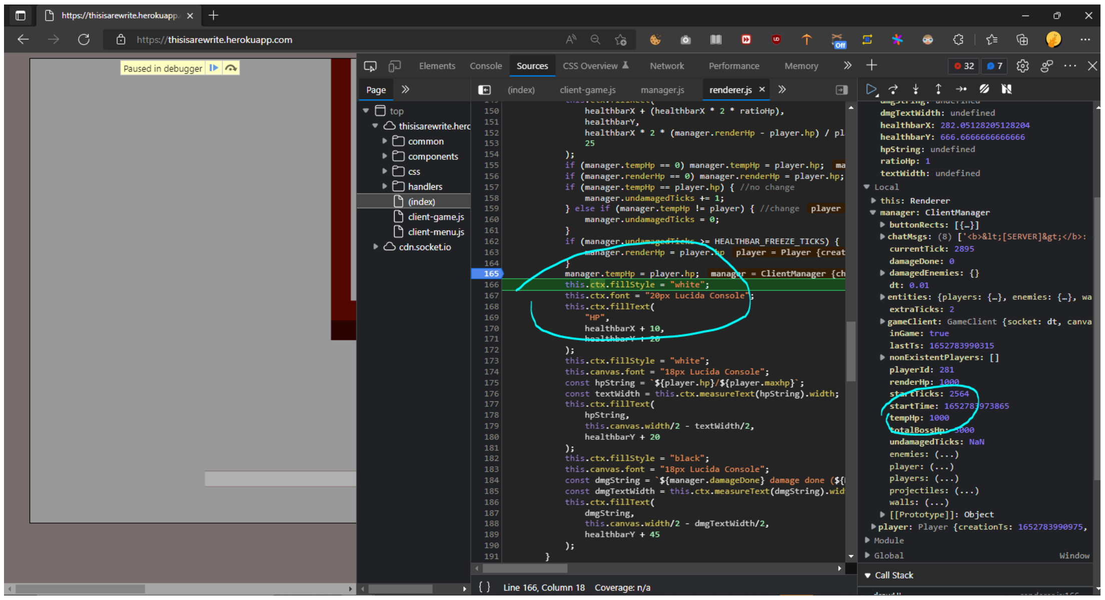
```
        healthbarX + (healthbarX * 2 * ratioHp),
        healthbarY,
        healthbarX * 2 * (manager.renderHp - player.hp) / pl
        25
    );
    if (manager.tempHp == 0) manager.tempHp = player.hp;  ma
    if (manager.renderHp == 0) manager.renderHp = player.hp;
    if (manager.tempHp == player.hp) { //no change
        manager.undamagedTicks += 1;
    } else if (manager.tempHp != player) { //change  player
        manager.undamagedTicks = 0;
    }
    if (manager.undamagedTicks >= HEALTHBAR_FREEZE_TICKS) {
        manager.renderHp = player.hp  player = Player {creat
    }
    manager.tempHp = player.hp;
    this.ctx.fillStyle = "white";
    this.ctx.font = "20px Lucida Console";
    this.ctx.fillText(
        "HP",
        healthbarX + 10,
        healthbarY + 20
    );
    this.ctx.fillStyle = "white";
    this.canvas.font = "18px Lucida Console";
    const hpString = `${player.hp}/${player.maxhp}`;
    const textWidth = this.ctx.measureText(hpString).width;
    this.ctx.fillText(
        hpString,
        this.canvas.width/2 - textWidth/2,
        healthbarY + 20
    );
    this.ctx.fillStyle = "black";
    this.canvas.font = "18px Lucida Console";
    const dmgString = `${manager.damageDone} damage done (${
    const dmgTextWidth = this.ctx.measureText(dmgString).wid
    this.ctx.fillText(
        dmgString,
        this.canvas.width/2 - dmgTextWidth/2,
        healthbarY + 45
    );
}
```

Line 165, Column 37   Coverage: n/a

```
      dt: 0.01
    ▶ entities: {players: {…}, enemies: {…}, wa
      extraTicks: 2
    ▶ gameClient: GameClient {socket: dt, canva
      inGame: true
      lastTs: 1652783990315
    ▶ nonExistentPlayers: []
      playerId: 281
      renderHp: 1000
      startTicks: 2564
      startTime: 1652783973865
      tempHp: 1000
      totalBossHp: 5000
      undamagedTicks: NaN
      enemies: (...)
      player: (...)
      players: (...)
      projectiles: (...)
      walls: (...)
    ▶ [[Prototype]]: Object
  ▶ player: Player {creationTs: 1652783990975,
▶ Module
▶ Global                                  Window
▼ Call Stack
→ drawUI                        renderer.js:165
  draw                          renderer.js:110
  tick                          client-game.js:62
  (anonymous)                   client-game.js:41
▶ XHR/fetch Breakpoints
▶ DOM Breakpoints
▶ Global Listeners
▶ Event Listener Breakpoints
▶ CSP Violation Breakpoints
```

Single Line Stepping

By using the chrome debugger to set a breakpoint, a developer is able to then step over lines of code to observe changes in the program's state. In this example, I can see that after stepping, the tempHp variable is still set to the expected value of 1000. Hence, I can determine that the code is functioning correctly.



```
            healthbarX + (healthbarX * 2 * ratioHp),
            healthbarY,
            healthbarX * 2 * (manager.renderHp - player.hp) / pl
            25
        );
        if (manager.tempHp == 0) manager.tempHp = player.hp;  ma
        if (manager.renderHp == 0) manager.renderHp = player.hp;
        if (manager.tempHp == player.hp) { //no change
            manager.undamagedTicks += 1;
        } else if (manager.tempHp != player) { //change  player
            manager.undamagedTicks = 0;
        }
        if (manager.undamagedTicks >= HEALTHBAR_FREEZE_TICKS) {
            manager.renderHp = player.hp  player = Player {creat
        }
        manager.tempHp = player.hp;  manager = ClientManager {ch
        this.ctx.fillStyle = "white";
        this.ctx.font = "20px Lucida Console";
        this.ctx.fillText(
            "HP",
            healthbarX + 10,
            healthbarY + 20
        );
        this.ctx.fillStyle = "white";
        this.canvas.font = "18px Lucida Console";
        const hpString = `${player.hp}/${player.maxhp}`;
        const textWidth = this.ctx.measureText(hpString).width;
        this.ctx.fillText(
            hpString,
            this.canvas.width/2 - textWidth/2,
            healthbarY + 20
        );
        this.ctx.fillStyle = "black";
        this.canvas.font = "18px Lucida Console";
        const dmgString = `${manager.damageDone} damage done (${
        const dmgTextWidth = this.ctx.measureText(dmgString).wid
        this.ctx.fillText(
            dmgString,
            this.canvas.width/2 - dmgTextWidth/2,
            healthbarY + 45
        );
    }
```

Line 166, Column 18   Coverage: n/a

```
    dmgTextWidth: undefined
    healthbarX: 282.05128205128204
    healthbarY: 666.6666666666666
    hpString: undefined
    ratioHp: 1
    textWidth: undefined
▼ Local
  ▶ this: Renderer
  ▼ manager: ClientManager
    ▶ buttonRects: [{…}]
    ▶ chatMsgs: (8) ['<b>&lt;[SERVER]&gt;</b>:
      currentTick: 2895
      damageDone: 0
    ▶ damagedEnemies: {}
      dt: 0.01
    ▶ entities: {players: {…}, enemies: {…}, wa
      extraTicks: 2
    ▶ gameClient: GameClient {socket: dt, canva
      inGame: true
      lastTs: 1652783990315
    ▶ nonExistentPlayers: []
      playerId: 281
      renderHp: 1000
      startTicks: 2564
      startTime: 1652783973865
      tempHp: 1000
      totalBossHp: 3000
      undamagedTicks: NaN
      enemies: (...)
      player: (...)
      players: (...)
      projectiles: (...)
      walls: (...)
    ▶ [[Prototype]]: Object
  ▶ player: Player {creationTs: 1652783990975,
▶ Module
▶ Global                                  Window
▼ Call Stack
```

Readability of Code

Meaningful Variable Names

In the following code and throughout my program, I use descriptive variable names. In addition, I utilise a naming convention of camelCase for normal variables and functions, SCREAMING_SNAKE_CASE for constant variables and PascalCase for classes. This improves the ability for future developers to understand the code written and use variables correctly when making modifications.

```
const onTryHit = function({target}) {
    if (target.invincible) return;
    let enemy = manager.worlds[socket.profile.currentWorld].enemies[target.id];
    if (!enemy) return;
    enemy.hp -= PLAYER_PROJ_DESC.damage;
    if (enemy.hp <= 0) {
        enemy.dead = true;
    }
    socket.profile.damageDone += PLAYER_PROJ_DESC.damage;
}
```

White Space

In the following code, I utilise blank lines to create white space between the various steps of the subroutine. This improvement in readability by grouping makes the logical steps of the module easier to decipher. The first section of the code acquires the closest player, the second subtracts from the entity's timers, and the third parses behaviours.

```
tick(manager, worldName) {
    const worldObj = manager.worlds[worldName];
    const playerEntities = Object.values(worldObj.players);
    if (!playerEntities.length) return;
    const closest = this.closestPlayer(playerEntities)

    for (const timer in this.timers) this.timers[timer].timer -= 1;
    if (this.timers.shootTimer.timer <= 0) this.timers.shootTimer.timer = 100;

    for (const bhv of this.ai.states[this.aiState].behaviour) {
        const args = bhv.split(' ');
        this[args[0]](args, manager, closest, worldObj);
    }
}
```

Indentation

In the following code, I use indentation to clarify the scope of arrow functions passed as parameters to other functions. This improvement in readability assists in preventing future developers from referring to variables outside of scope and making a common javascript error.

```
start() {
    waitUntil(() => this.socket.connected, () => {
        …
        this.interval = setInterval(
            function(self) { return function() { self.tick() } }(this),
            CL_TICK_RATE
        );
    })
}
```

Comments

In the following code, I utilise a multi-line comment to provide a brief summary for how a class should be used. In this case, the improvement in readability for the Controller class makes it easier for a developer to determine where to make changes and create new functions.

```
/*
client module that manages the receiving of input from the browser,
the storage of options relating to input, and the sending of inputs
to the server.
*/
export default class Controller {
    ...
}
```

In the following code, I use a multi-line comment to create a “docstring” for the function. This “docstring” summarises the function’s usage, parameters and return variable so that future developers can read the code more easily and utilise this function properly.

In addition, I use comments to provide an abstract description for lines of code that are more difficult to decipher and provide reasons for coding decisions made. This improvement in readability helps a developer in understanding the flow of execution.

```
 static updateState(manager, state) {
    /*
    This is a static function that updates the local state
    of a Manager object by parsing the entities sent by the
    server in an 'update' packet.

    Args:
        manager : Manager object with local entities to be updated
        state : an object dictionary of format { entityid : entity object }
    Returns:
        updated Manager object
    */
    let updatedManager = manager;
    for (let entity of Object.values(state)) {
        //if an entity does not exist locally, create it locally and add a positionBuffer
        if (!updatedManager.entities[entity.category][entity.id]) {
            let newEntity = new entityTypes[entity.objType](entity);
            newEntity.positionBuffer = [];
            updatedManager.entities[entity.category][entity.id] = newEntity;
        }
        //if the entity is the player, update everything
        if (entity.id == updatedManager.playerId) {
            Object.assign(updatedManager.player, entity);
        }
        //if the entity is not the player, exists locally, and does not have the
doNotUpdate flag.
        else if (!entity.doNotUpdate) {
            //get the corresponding local entity by using the id of the server-sent
entity
            let localEntity = updatedManager.entities[entity.category][entity.id];
            //push the server-sent entity's location to the local entity's position
buffer for interpolation
            localEntity.positionBuffer.push({
                ts: Date.now(),
                x: entity.x,
                y: entity.y
            })
            //delete these variables so they are not assigned to the localEntity,
interpolation will assign these variables
            delete entity.x;
            delete entity.y;
            Object.assign(localEntity, entity);
        }
    }
    return updatedManager
}
```

Errors

Syntax Errors

Syntax errors arise when the source code does not adhere to the strict rules of the programming language. In the case of browser javascript, the browser's javascript interpreter will attempt to perform lexical and syntactical analysis of the syntactically incorrect source code according to the rules set forth in metalanguages such as EBNF and railroad diagrams. However, the interpreter will be unable to translate the source code into machine code instructions and ultimately throw an exception and/or crash.

A common syntax error I encountered when writing my code was missing commas (,) when defining a javascript object, as seen below:

```
get state() {
    return {
        x: this.x,
        y: this.y,
        size: this.size //Uncaught SyntaxError: Unexpected identifier
        category: this.category,
        objType: this.objType
    }
}
```

Runtime Errors

Runtime errors occur when the computer is unable to continue executing instructions. In the case of the javascript interpreter, fatal logic errors or software/hardware incompatibilities may lead to the browser freezing and ultimately crashing. However, browser javascript is a language that is generally resilient to errors, opting to print the exception to the console and continue execution in most cases.

I encountered several runtime errors while writing the project's code. The following code contains an error that results in the message box rapidly doubling its text contents until an eventual browser freeze and crash:

```
for (const msg of manager.chatMsgs) {
    msgBox.innerHTML += msg;
}
```

Logic Errors

Logic errors occur from programs that are written incorrectly. Often, this type of error is derived from mistakes made developing algorithms before implementation into the final source code. Logic errors manifest in either a runtime error, or continued execution of incorrect code.

I made several logic errors throughout development. In the following code, I make the mistake of not removing a message from the server message queue after parsing the message, which led to the server's first sent message being replayed infinitely.

```
processServerMessages(manager) {
    while (true) {
        const msg = this.svMsgQueue[0];
        if (!msg) break;

        Networking.updateState(manager, msg.state);
        this.lastAckNum = msg.inputAckNum;
        this.performServerReconciliation(manager);
    }

    Networking.interpolateEntities(manager)
}
```

Efficiency and Elegancy of Code

Object Oriented Programming

I utilise object oriented programming to improve the elegance of code and the speed of development. For example, entity types within the game inherit from a base class to share a common interface in their attributes and functions, a concept known as polymorphism. This shared interface allows for the writing of code able to handle any entity type. In the following code, the definition of the Entity base class is shown, along with an example entity type that inherits from this base class and example code that processes entities derived from BaseEntity.

The base class.

```
export class Entity {
    creationTs = Date.now();
    rotate = false;
    doNotUpdate = false;
    category = None;

    constructor(x, y, size, identifier) {
        this.x = x;
        this.y = y;
        this.size = size;
        if (identifier) {
            this.id = identifier;
        } else {
            this.id = id;
            id++;
        }
    }

    detectEntityCollision(entity) { … }

    get chunkLoc() { … }

    get loc(). { return new Point(this.x, this.y) }
    get top() { return this.y - this.size/2; }
    get bottom() { return this.y + this.size/2; }
    get left() { return this.x - this.size/2; }
    get right() { return this.x + this.size/2; }

    set top(y) { this.y = y + this.size/2; }
    set bottom(y) { this.y = y - this.size/2; }
    set left(x) { this.x = x + this.size/2; }
    set right(x) { this.x = x - this.size/2; }
}
```

An entity type that inherits from the base class. By inheriting from the base class with 'class Player extends Entity', a significant amount of 'boilerplate code' is removed.

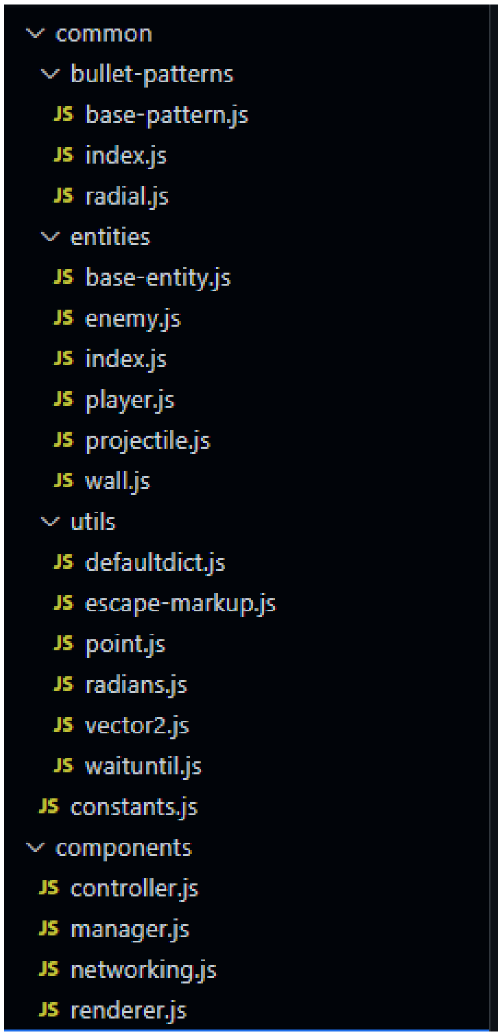
```
export class Player extends Entity {
    category = ENTITY_CATEGORY.players;
    objType = this.constructor.name;
    dead = false;
    damaged = false;

    constructor({x, y, size, speed, socketId, hp, id=null}) {
        super(x, y, size, id);
        this.speed = speed;
        this.hp = hp;
        this.maxhp = hp;
        this.socketId = socketId;
    }

    applyInput(rot, inputs, wallEntities) {
        …
}
```

An example of using the base class interface in the networking code.

```
static interpolateEntities(manager) {
    const renderTs = Date.now() - SV_UPDATE_RATE;
    for (const entityType of Object.values(manager.entities)) {
        for (let entity of Object.values(entityType)) {
            if (entity.doNotUpdate) continue; //a base class attribute
            const b = entity.positionBuffer;
            while (b.length >= 2 && b[1].ts <= renderTs) {
                b.shift();
            }
            if (b.length >= 2 && b[0].ts <= renderTs && renderTs <= b[1].ts) {
                entity.x = b[0].x + (b[1].x - b[0].x) * (renderTs - b[0].ts) /
(b[1].ts - b[0].ts); //a base class attribute
                entity.y = b[0].y + (b[1].y - b[0].y) * (renderTs - b[0].ts) /
(b[1].ts - b[0].ts); //a base class attribute
            }
        }
    }
}
```

Modular Design - one logical task per subroutine

Rather than writing long functions that contain multiple, disjointed processes, I apply the tenet of ‘one logical task per subroutine’ to keep code organised and easily reused. This is furthered by the object oriented programming approach, which inherently organises subroutines into objects. Functions and objects are also separated into individual files where appropriate, so as to minimise the length of individual files and more elegantly structure the program. The file directory of my program is shown in the below screenshot.



```
common
  bullet-patterns
    base-pattern.js
    index.js
    radial.js
  entities
    base-entity.js
    enemy.js
    index.js
    player.js
    projectile.js
    wall.js
  utils
    defaultdict.js
    escape-markup.js
    point.js
    radians.js
    vector2.js
    waituntil.js
  constants.js
components
  controller.js
  manager.js
  networking.js
  renderer.js
```

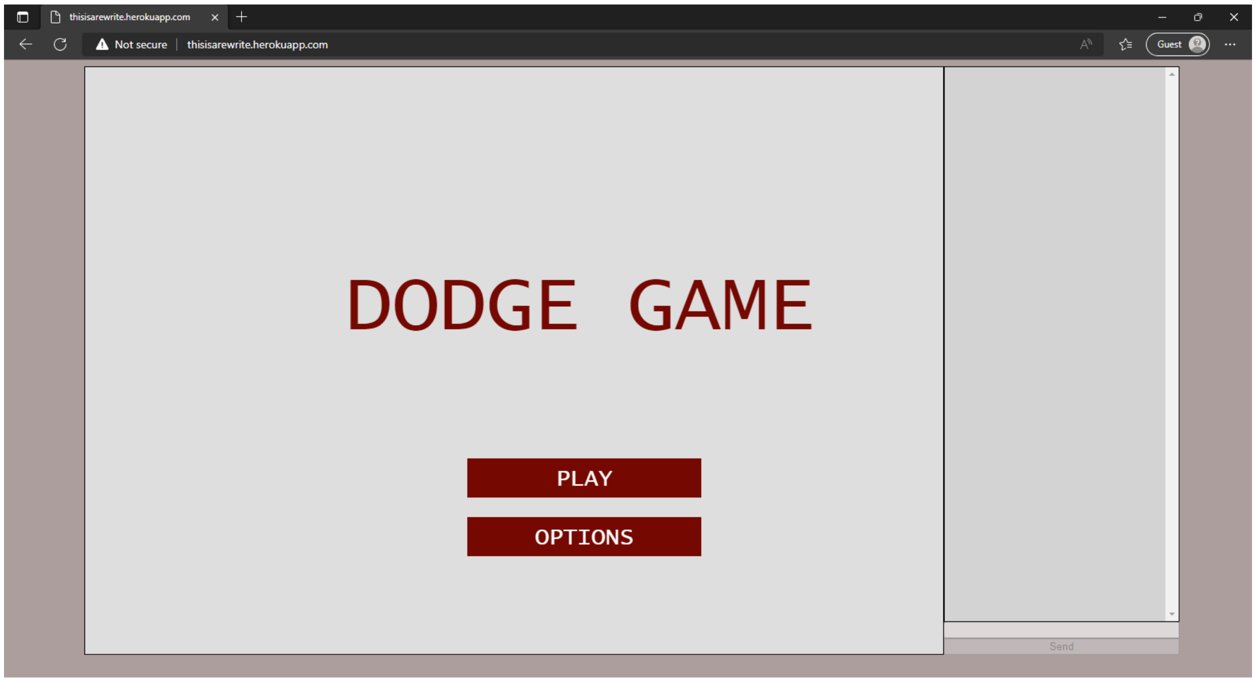
Clear and Uncluttered Mainline

A clear and uncluttered mainline is maintained throughout the code to improve its elegance. A clear mainline makes it easier to follow and identify calls to subroutines of the program. In addition, it facilitates a modular structure within the program, the mainline calling subroutines which cascade into further subroutine calls. In the following code, the mainline is shown.

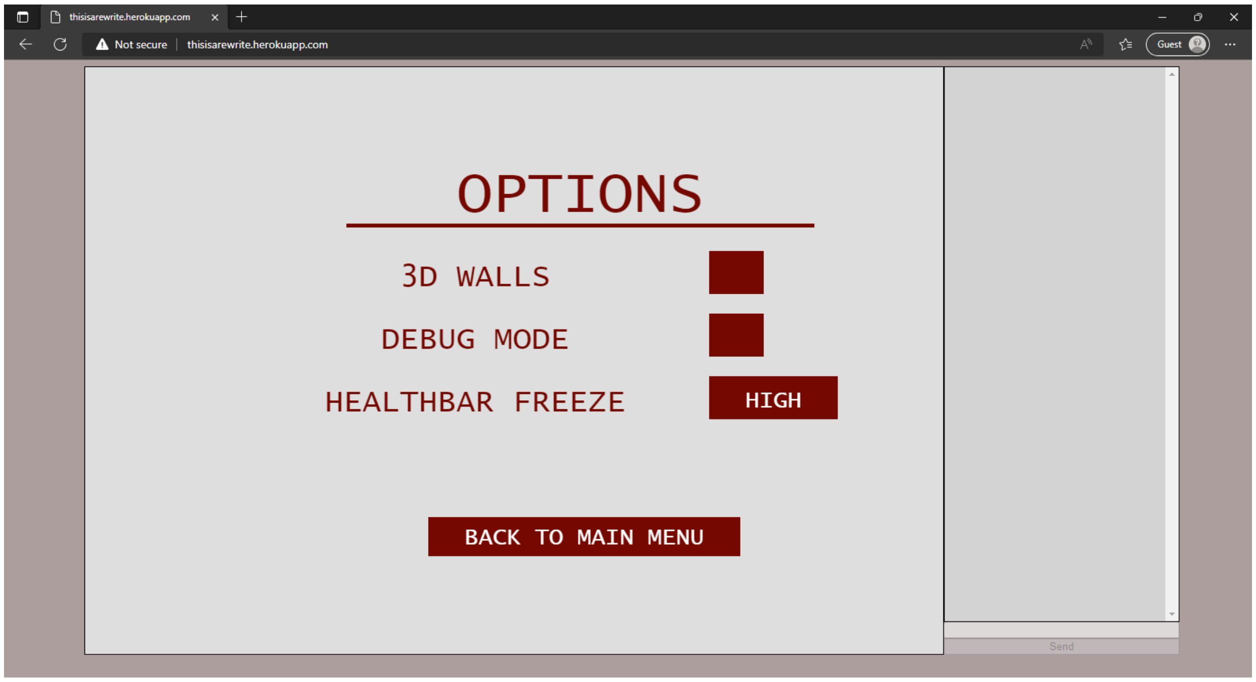
```
tick() {
    this.manager.tick(this.networking.socket);
    this.sendTime(this.manager);
    this.networking.processServerMessages(this.manager);
    this.controller.processInputs(this.manager, this.networking);
    this.renderer.draw(this.manager, this.controller.rotation);
}
```

User Interface

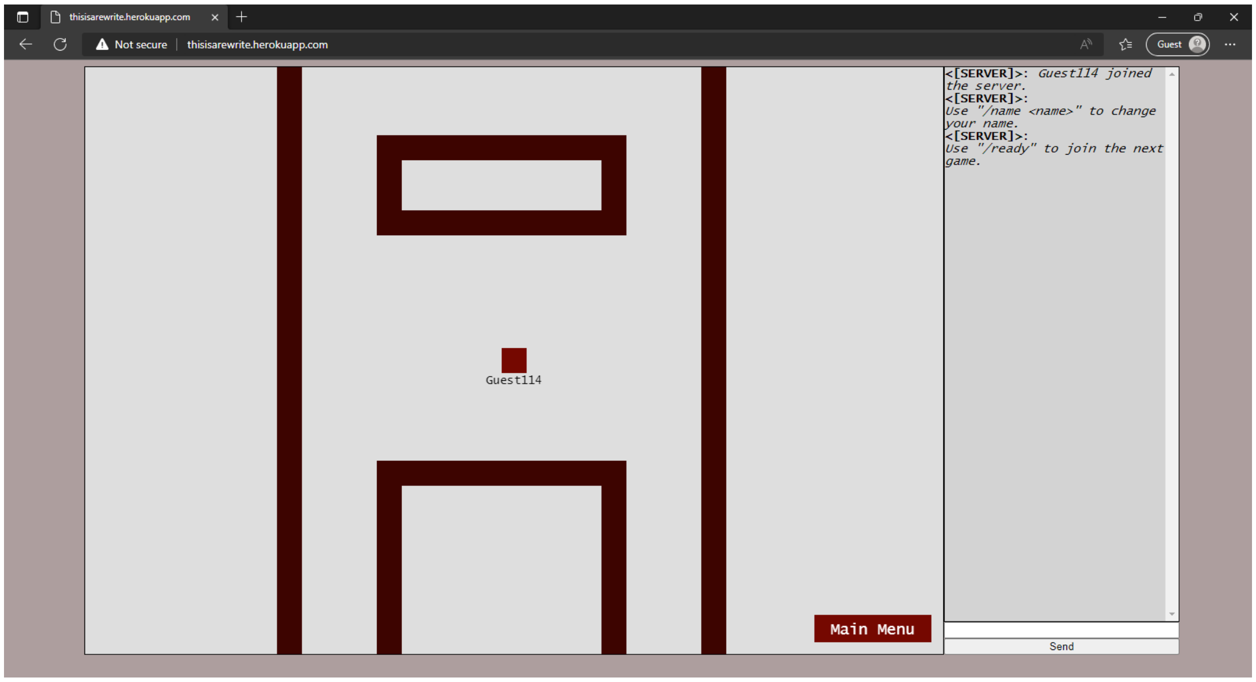
main menu



options menu

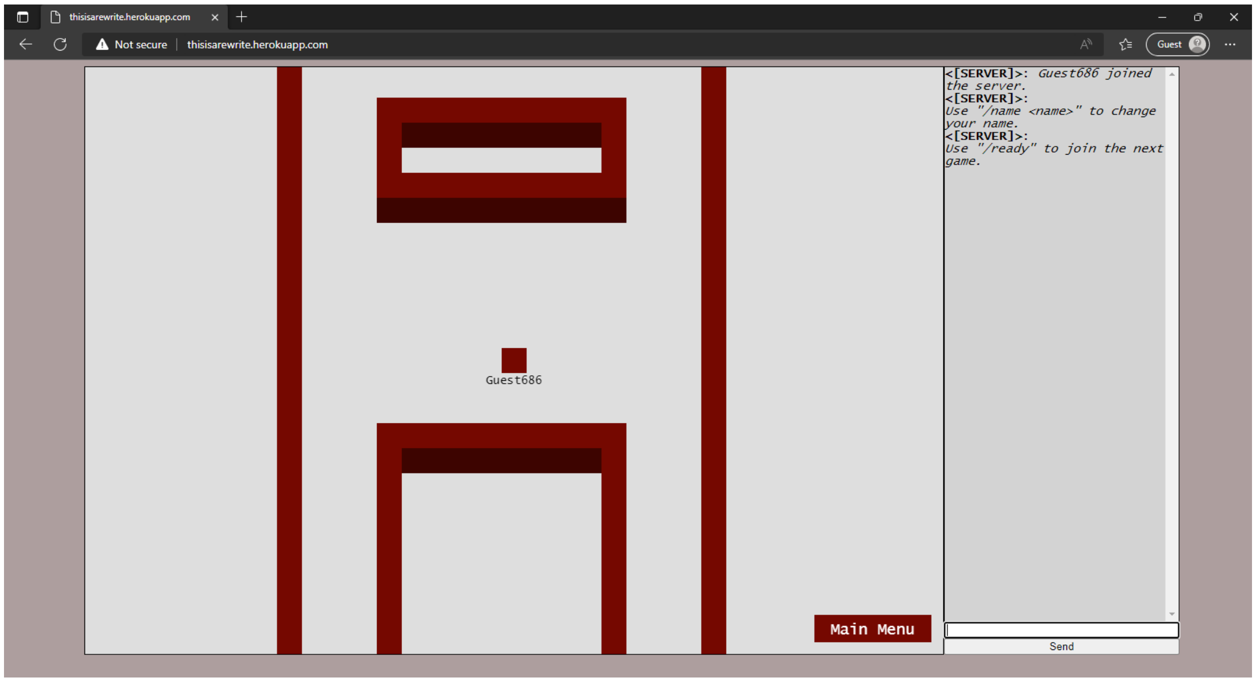


game view (lobby)



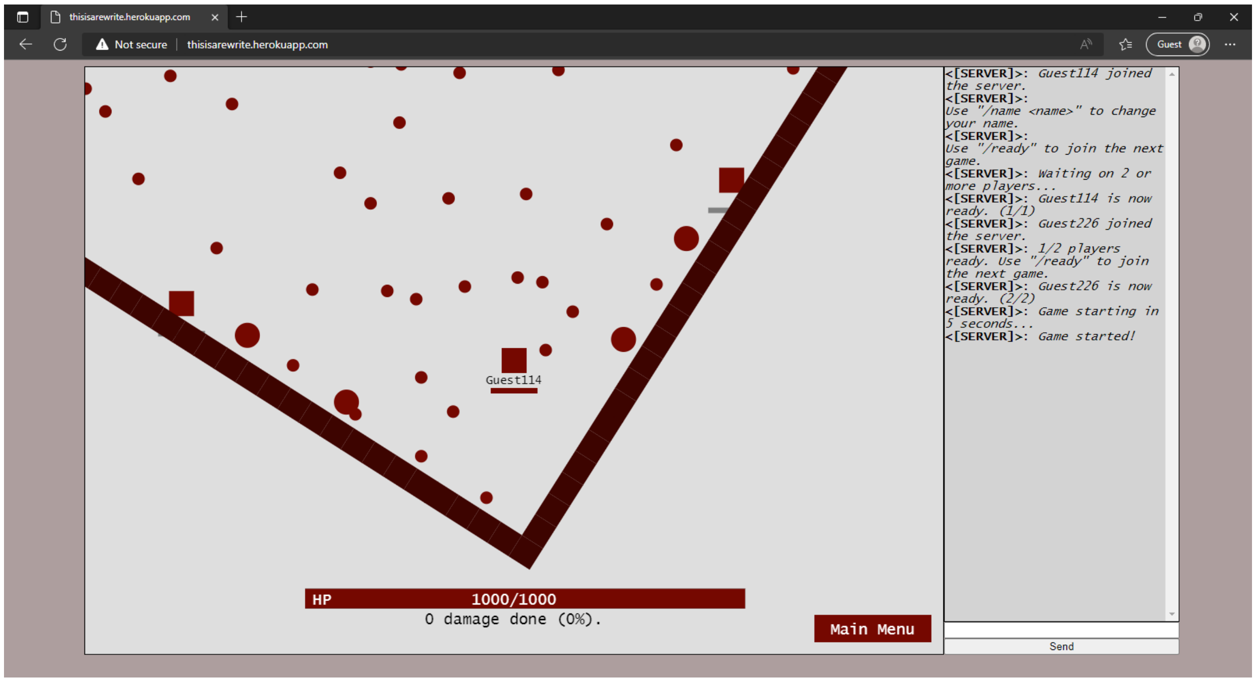
```
<[SERVER]>: Guest114 joined
the server.
<[SERVER]>:
Use "/name <name>" to change
your name.
<[SERVER]>:
Use "/ready" to join the next
game.
```

game view (lobby, 3d walls)



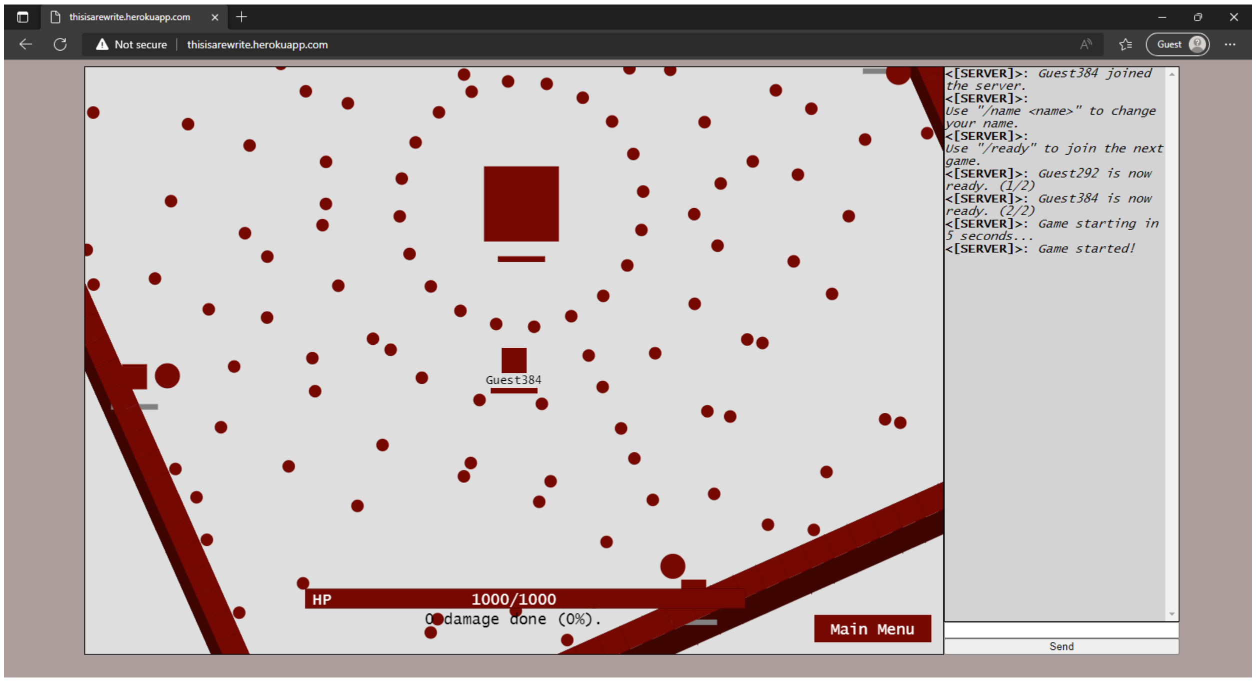
```
<[SERVER]>: Guest686 joined
the server.
<[SERVER]>:
Use "/name <name>" to change
your name.
<[SERVER]>:
Use "/ready" to join the next
game.
```

game view (in-game)



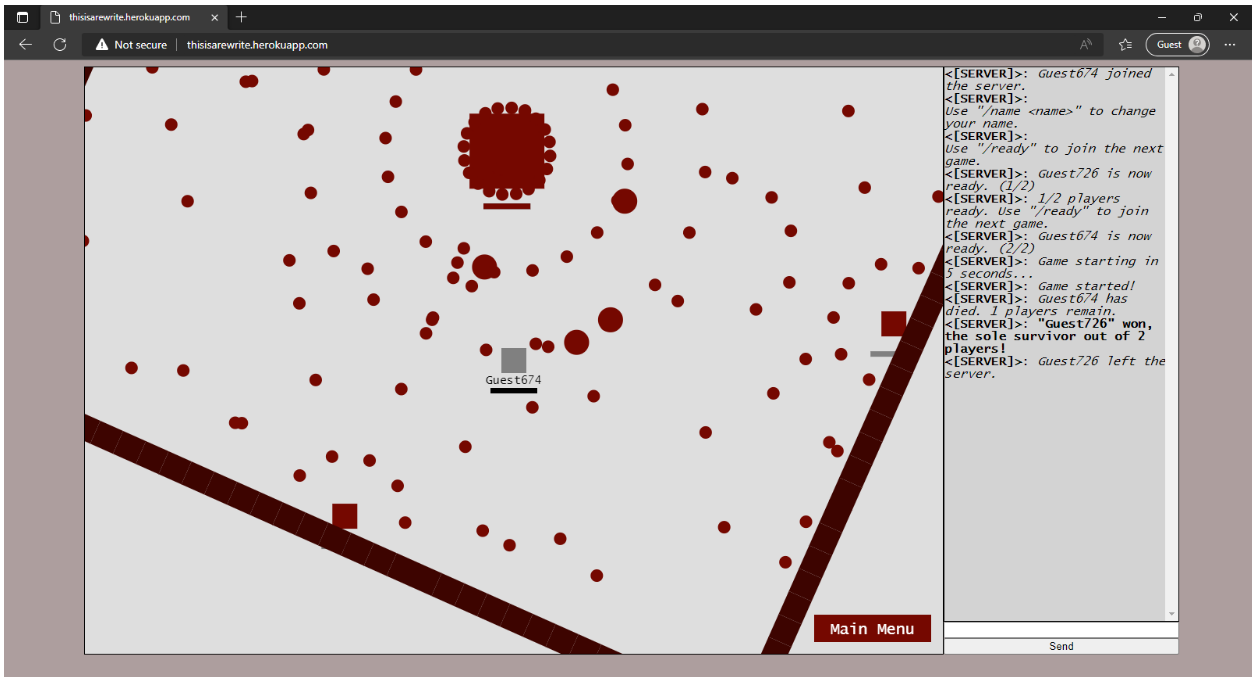
```
<[SERVER]>: Guest114 joined the server.
<[SERVER]>:
Use "/name <name>" to change your name.
<[SERVER]>:
Use "/ready" to join the next game.
<[SERVER]>: Waiting on 2 or more players...
<[SERVER]>: Guest114 is now ready. (1/1)
<[SERVER]>: Guest226 joined the server.
<[SERVER]>: 1/2 players ready. Use "/ready" to join the next game.
<[SERVER]>: Guest226 is now ready. (2/2)
<[SERVER]>: Game starting in 5 seconds...
<[SERVER]>: Game started!
```

game view (in-game, 3d walls)



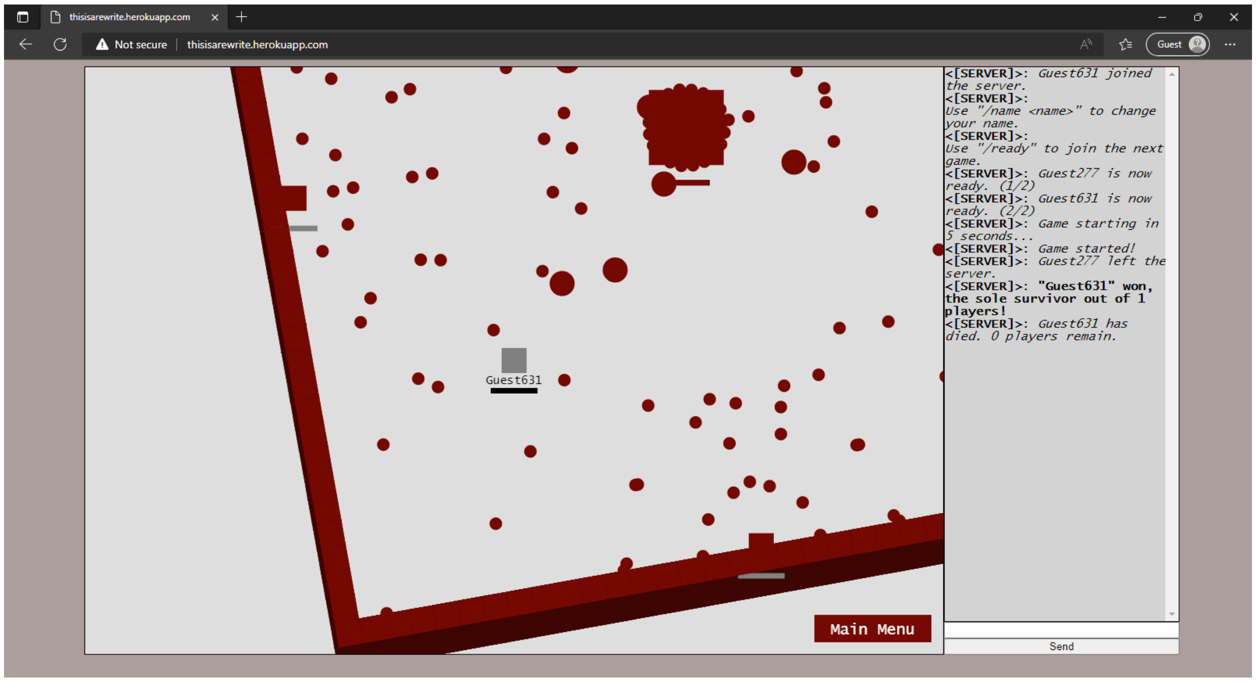
```
<[SERVER]>: Guest384 joined the server.
<[SERVER]>:
Use "/name <name>" to change your name.
<[SERVER]>:
Use "/ready" to join the next game.
<[SERVER]>: Guest292 is now ready. (1/2)
<[SERVER]>: Guest384 is now ready. (2/2)
<[SERVER]>: Game starting in 5 seconds...
<[SERVER]>: Game started!
```

game view (in-game, 0 hp)



```
<[SERVER]>: Guest674 joined the server.
<[SERVER]>:
Use "/name <name>" to change your name.
<[SERVER]>:
Use "/ready" to join the next game.
<[SERVER]>: Guest726 is now ready. (1/2)
<[SERVER]>: 1/2 players ready. Use "/ready" to join the next game.
<[SERVER]>: Guest674 is now ready. (2/2)
<[SERVER]>: Game starting in 5 seconds...
<[SERVER]>: Game started!
<[SERVER]>: Guest674 has died. 1 players remain.
<[SERVER]>: "Guest726" won, the sole survivor out of 2 players!
<[SERVER]>: Guest726 left the server.
```

game view (in-game, 0 hp, 3d walls)



```
<[SERVER]>: Guest631 joined the server.
<[SERVER]>:
Use "/name <name>" to change your name.
<[SERVER]>:
Use "/ready" to join the next game.
<[SERVER]>: Guest277 is now ready. (1/2)
<[SERVER]>: Guest631 is now ready. (2/2)
<[SERVER]>: Game starting in 5 seconds...
<[SERVER]>: Game started!
<[SERVER]>: Guest277 left the server.
<[SERVER]>: "Guest631" won, the sole survivor out of 1 players!
<[SERVER]>: Guest631 has died. 0 players remain.
```

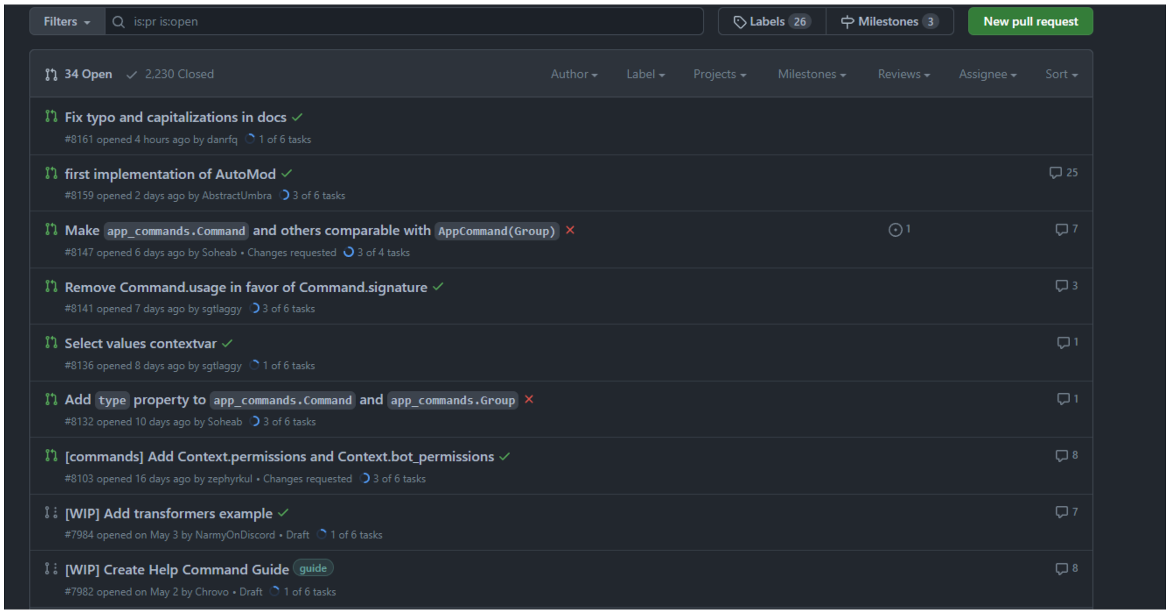
Suitable CASE Tools to Monitor Changes and Version Control

Git

A suitable CASE tool used for the project is the open source software Git, which provides both version control and changes management. This is done through a ‘distributed version control’ system, where the complete codebase located in the main branch of a repository is mirrored onto the computer of a developer as a separate branch for modification until a ‘pull request’ is made to make changes to the main branch. These pull requests often include a description of changes made, reason for making the changes and the identity of the person making the changes. If a pull request is approved by the owners of the repository, the changes on the branch are merged with the main branch of the repository. These merges are either ‘fast-forward’, where the changes are unrelated and able to be automatically combined without concern, or ‘conflicts’, which must be resolved manually by developers. This change management system supports the life cycle of changes from requested inclusion to part of the final product.

The main advantage of Git’s change management system is that it allows for developers to engage in parallel development both offline and online without code conflicts or compatibility issues. Another advantage is that pull requests can be monitored to create a history of changes and record of past versions– this acts as the foundation for Git’s system for version control.

Example: pull requests for the open-source discord.py project, in GitHub



Fix typo and capitalizations in docs ✓
#8161 opened 4 hours ago by danrfq 1 of 6 tasks

first implementation of AutoMod ✓ 25
#8159 opened 2 days ago by AbstractUmbra 3 of 6 tasks

Make `app_commands.Command` and others comparable with `AppCommand(Group)` ✗ 1 7
#8147 opened 6 days ago by Soheab • Changes requested 3 of 4 tasks

Remove Command.usage in favor of Command.signature ✓ 3
#8141 opened 7 days ago by sgtlaggy 3 of 6 tasks

Select values contextvar ✓ 1
#8136 opened 8 days ago by sgtlaggy 1 of 6 tasks

Add `type` property to `app_commands.Command` and `app_commands.Group` ✗ 1
#8132 opened 10 days ago by Soheab 3 of 6 tasks

[commands] Add Context.permissions and Context.bot_permissions ✓ 8
#8103 opened 16 days ago by zephyrkul • Changes requested 3 of 6 tasks

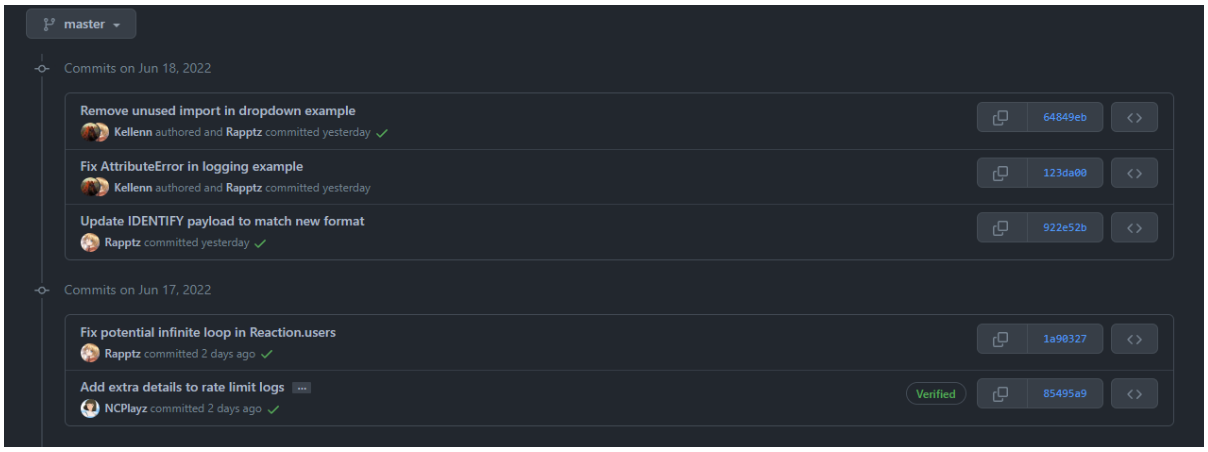
[WIP] Add transformers example ✓ 7
#7984 opened on May 3 by NarmyOnDiscord • Draft 1 of 6 tasks

[WIP] Create Help Command Guide guide 8
#7982 opened on May 2 by Chrovo • Draft 1 of 6 tasks

Git manages which version is officially released by the version set in the main branch. Particular changes are identified by random alphanumeric strings in Git, and are able to be rewinded to reset a project back to a particular version. Version numbers are expected to be maintained by the developer rather than the Git software. A common practice is to utilise the 'git tag' command to assign a version to a project. These version numbers are often in the format of x.y.z, where x is the major version number, y is the minor version number and z is the patch version number.

Git's version control system is useful for the fact that it allows developers to restore an older version if a newer version is broken, or if a developer needs to track down a particular version where a bug was introduced.

Example: commit history for the open-source project discord.py, in GitHub



Commits on Jun 18, 2022

Remove unused import in dropdown example
Kellenn authored and Rapptz committed yesterday ✓ — 64849eb

Fix AttributeError in logging example
Kellenn authored and Rapptz committed yesterday — 123da00

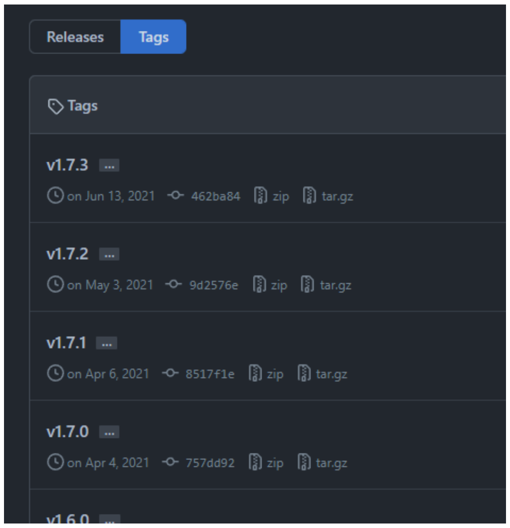
Update IDENTIFY payload to match new format
Rapptz committed yesterday ✓ — 922e52b

Commits on Jun 17, 2022

Fix potential infinite loop in Reaction.users
Rapptz committed 2 days ago ✓ — 1a90327

Add extra details to rate limit logs ...
NCPlayz committed 2 days ago ✓ — Verified — 85495a9

Example: version tags for the open-source project discord.py, in GitHub



Tags

v1.7.3 ...
on Jun 13, 2021 — 462ba84 — zip — tar.gz

v1.7.2 ...
on May 3, 2021 — 9d2576e — zip — tar.gz

v1.7.1 ...
on Apr 6, 2021 — 8517f1e — zip — tar.gz

v1.7.0 ...
on Apr 4, 2021 — 757dd92 — zip — tar.gz

v1.6.0 ...

Git’s change management system for parallel development is not a particularly useful feature for this project given that there is a singular developer. However, the version control system of Git is useful for this project because it provides the security of older versions for debugging issues and rollback in the case that a breaking change is made.

GitHub

An accompanying CASE tool is GitHub, which acts as a code-hosting platform and interface for Git’s version control and change monitoring. By storing old versions and changes on the cloud, the project is able to be developed without concern for significant errors or data loss, always having an old version to fall back on or look at for reference.

There are two GitHub repositories that contain the project’s code. A complete rewrite of the code prompted the creation of another repository. These repositories are publicly accessible, and available via Github’s web interface at github.com/haha-XD/thisisindeedagame and github.com/haha-XD/thisisindeedarewrite

Changing User Requirements

User requirements can change for the purposes of bug fixing, compatibility with new hardware and software, and addition of new features. To handle changing requirements, developers must identify the change, locate the section of code to be altered, note the necessary changes with consideration of dependencies, and then engage in modification of the code with implementing and testing.

Changing user requirements are often communicated via user-submitted support tickets, which can also act as a gauge for the demand for a particular change. In this project, a support ticket command is implemented to send a short, 40 character message to the developer. By looking at these messages, a developer should be able to identify potential changes and their rationale.

After identifying a change in user requirements, the specific section of code that requires changing must be located. This should first require consultation with system modelling tools such as the project's data flow diagrams and structure charts to acquire a thorough understanding of the system's structure. Then, after understanding the system's structure, a developer should be able to locate the module of code to be altered-.

With the module of code to be changed located, a developer must then consider the dependencies and dependents of a module to determine the necessary changes that have to be made. In the case of this project, the modular structure of the code often necessitates changing logic throughout the program if a module is altered. For example, a change in the structure of the Entity object may require an alteration in the networking code to process this new Entity object structure– and potentially other modules.

Once the changes required are determined, the developer should modify the code by engaging in implementing and testing. Often, testing data and CASE tool scripts for the original modules can be reused whilst implementing and testing the changes.

After the changes are made, the new project has to be distributed to users. In the case of this project, the web-based nature of the game means that the first step to update the game is to upload a newer version of the project to the server host. Given that the game is multiplayer and clients must share a protocol, the verification checks of the server must then also be updated to ensure that connecting clients run the latest version of the game.

User Manual

What is Dodge Game?

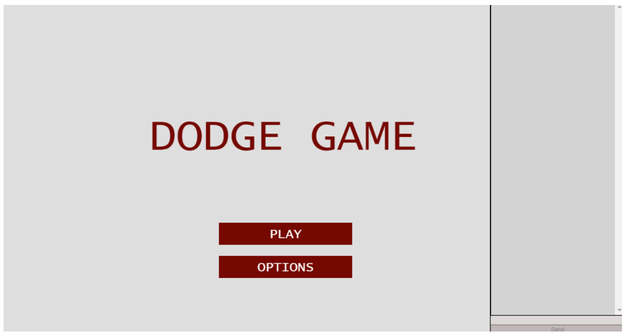
Dodge Game is an online multiplayer action game. Players control squares and compete against each other to dodge for as long as possible.

The game is available at https://thisisarewrite.herokuapp.com.

Getting Started

Upon loading the website, the main menu screen will be visible.

From there, you can either click PLAY to play the game, or click OPTIONS to change game options.



How do I Play?

After clicking PLAY and loading into the game, you will be placed in the main lobby. To join when the next game starts, '/ready' has to be entered into the chat to set your status as ready. A game starts if two conditions are satisfied: there are at least two people who are ready, and at least 75% of users are ready. After a game ends, all players are returned to the main lobby– you must re-enter /ready to join the next game.

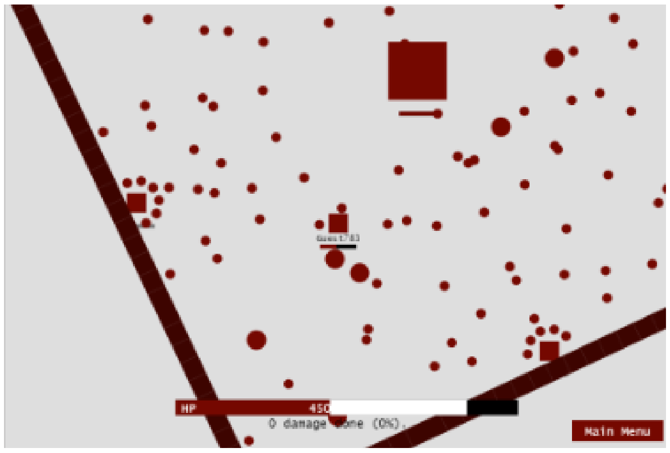
Only one game can be started at a time. If you attempt to mark yourself as ready when players are in-game, nothing will happen.

How does the Game Work?

Upon starting a game, all ready players are moved into the game lobby. All players start with 1000 hp. If your character collides with a projectile, you will lose hp. You must dodge projectiles to preserve your hp.

You are also able to shoot projectiles by left clicking. If your projectiles collide with a damageable enemy, their hp will decrease and your damage dealt will increase.

This information is displayed at the bottom of the screen.



To win at Dodge Game, you must either have the most damage when the enemy's hp reaches 0, or be the only player with positive hp.

Controls

The inputs to control the character are as follows.

- WASD: directional movement
- QE**:** rotation of screen
- T**:** reset rotation of screen
- Left Mouse Click**:** shoot projectile, in direction of cursor

Chat Commands

A number of 'commands' that can be entered into chat to interact with the game. All commands are prefixed with a forward slash ('/').

List of commands:

'/help' : produces a help message in the chat that gives a short rundown on the game and available commands.

'/ready' : marks you as 'ready'-- if a sufficient number of players are ready, a game will begin. If you are not ready and a game begins, you will not join.

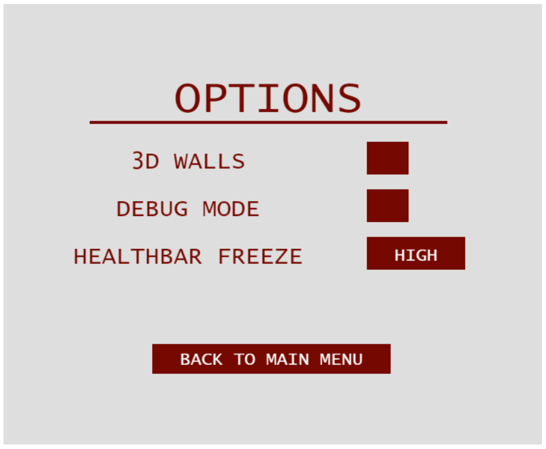
'/name <desired name>': sets your player name to the desired name. Only alphanumeric characters of less than 10 characters are permitted.

'/ticket <message>': sends a message to the developer, should be used for bug reports and feature requests.

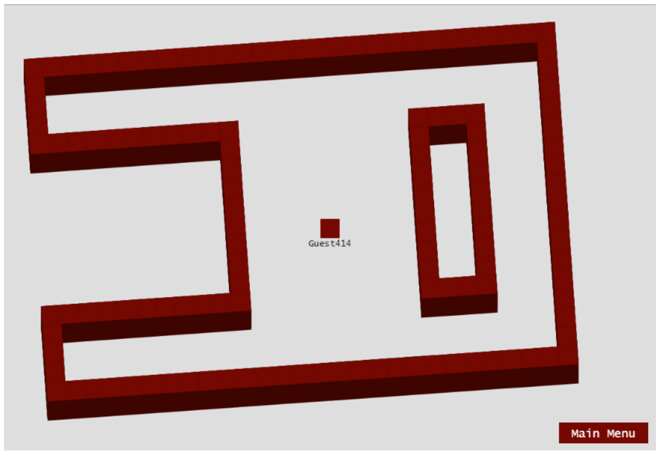
Options

In the options menu, there are three graphical options to alter.

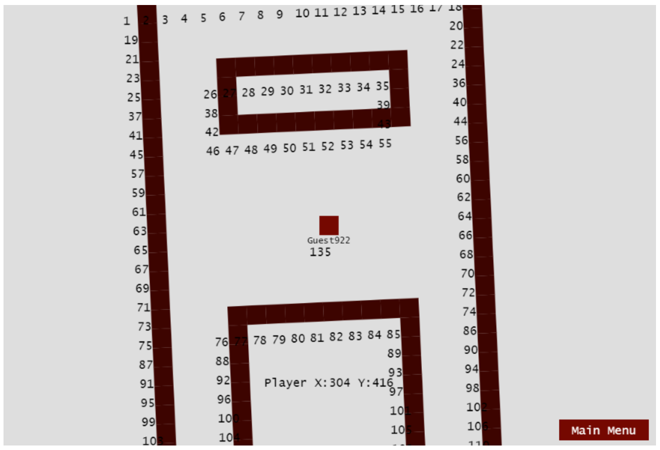
The 3D Walls and Debug Mode options are checkboxes. The Healthbar Freeze option can either be low, medium, or high.



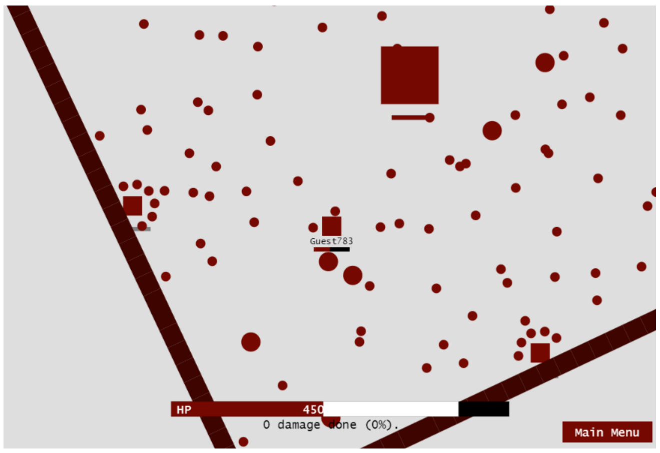
The 3D Walls option toggles three-dimensional rendering for walls. This may be taxing on performance, so leave this option disabled if you experience frame drops.



The Debug Mode option shows additional information about the game. This should only be used for adding detail to bug reports, given that rendering this additional information slows performance and adds a significant amount of visual clutter.



The Healthbar Freeze option determines the duration of the white visual effect on the health bar that occurs when taking damage.



Installation Manual

Requirements

System Requirements:

	Minimum	**Recommended**
CPU	Intel Atom x6-Z8350 CPU @ 1.44 GHz or equivalent	Intel Xeon CPU @ 3.6 GHz or equivalent
GPU (see: 'Enabling Hardware Acceleration on your browser')	Intel HD Graphics 2GB or equivalent	NVIDIA GeForce 1060 3GB or equivalent
RAM	2GB	4GB

Minimum Browser Requirements:

Browser	**Version**
Chrome	51
Firefox	52
Edge	14
Safari	10
Opera	38

Installation Instructions

1. Open your browser
2. Navigate to https://thisisarewrite.herokuapp.com

Enabling Hardware Acceleration on your Browser (optional)

If your browser supports it, enabling hardware acceleration on your browser may increase the performance of the game by shifting processing to your GPU.

Chrome:

1. Navigate to chrome://settings
2. Click **Show Advanced Settings** at the bottom of the page
3. Find the **System** Section
4. Select **Use hardware acceleration when available**
5. Restart chrome.

Edge:

1. Navigate to edge://settings
2. Click on **System** on the left side menu.
3. Under **System**, select **Use Hardware Acceleration when available**
4. Restart Edge

Firefox:

1. Click on the menu button and select **Settings**
2. Select the **General** panel
3. Under **Performance**, select **Use Hardware Acceleration when available**
4. Restart Firefox

Safari: does not support hardware acceleration.

Opera:

1. Click on the menu button and select **Settings**
2. Select **Browser** from the left side menu
3. Find the **System** section
4. Select **use Hardware Acceleration when available**
5. Restart Opera

Project Report

Learning

Through the major project, I learned technical skills in javascript, multiplayer networking and structuring large projects. Over the process of acquiring these skills and developing the program, I also learned about the importance of research and the necessity of clearly defining a project's scope.

Upon deciding that I wanted to make a web game, I chose to use javascript as the project's programming language because it could run on both the browser and the server. From a background of only knowing Python and C++, learning javascript was difficult. Despite the syntax being mostly uncomplicated, javascript's long legacy of changes made learning how to structure a Javascript project difficult. For example, the online resources for object oriented programming in javascript often contained conflicting information. This was because, prior to the release of ES6 javascript, there was no concept of a javascript 'class'. Instead, functions, which also acted as objects, had to be used as 'classes'– this outdated method was shown in many older javascript articles. Following these outdated javascript articles, my code became unnecessarily complex until I finally rewrote the program. In my process of learning javascript, I also learned the value of researching the most optimal way before jumping to development.

Given that the game was multiplayer, I had to develop an understanding of multiplayer networking. Prior to starting the project, I had never done any form of networking. However, with research and practice, I was able to develop my own solutions to the networking problems of the game. I learned the importance of research and practice to understand programming concepts.

The size of the program was the primary difficulty in developing the program. This was due to my lack of experience designing software architecture for large programs, my constantly expanding scope for the program and the time commitment required for development. The combination of these factors eventually led to a complete rewrite of the program. This required even more unexpected time commitment, bringing the total development time to about five months of consistent work (my exact work time is evident from the commit history on my github repositories). However, by reflecting on the failures of my first iteration of the program, I was able to reduce the scope of the program, learn the skill of designing large programs and write code that was far easier and faster to develop.

Future Directions

Future directions for the project should first prioritise fixing of the networking code. In cases where the player or the server are lagging severely, desyncs and disconnects occur. In particular, this occurs when beginning a game. To resolve this, clients should send an acknowledgement packet in response to the ‘begin game’ packet, so that the server can ensure all clients receive the server update prior to beginning the game and disconnect clients experiencing extreme latency. This solution should also be applied to other aspects of the game, such as the chat function.

The second priority should be to rewrite the game’s rendering engine, which uses HTML canvas and javascript. Significant performance improvements can be made. For example, the 3D Walls rendering with HTML canvas requires many more drawings than apparent on the screen. An alternative solution to HTML canvas could be WebGL, which should produce significant performance improvements as it works at a lower level with the GPU and does not require the restrictive abstraction of HTML canvas.

An account system could be implemented to persistently measure a player’s stats, in conjunction with a database hosted on the server to hold this data. This would be a simple addition to improve the game’s player retention.

Comparison with Original Design Specifications

User Specifications

Ergonomic and intuitive controls	**Met** The controls of the program are intuitive, given that WASD and left click are common controls for games.
Simple and uncluttered interface	**Met** The interface is simple and uncluttered. This was achieved by letting chat commands replace a lot of the interface and keeping the visuals minimalist.
Clear and integrated tutorial	**Not met** The /help command provides some level of information, but the help information primarily provides information about chat commands. There is no tutorial and details on the game's mechanics are not found in the game itself, so new users may have trouble understanding the game.
Judicious use of colours in interface	**Met** The colour scheme of the interface is minimalist and consistent– shades of light red to dark red.
Compatibility with the user's browser plugins	**Met** The game shouldn't interfere with the user's browser plugins.
Compatibility with the user's browser settings	**Not met** The game's resolution can't be changed. The user must change their zoom levels to adapt to the game's resolution. This is a limitation of

	HTML canvas, the foundation for the game's rendering engine.
Preserves the user's privacy and security and asks for consent in data collection	**Met, adapted.** This was originally intended as a specification for the game's account system. A reduction of the game's scope removed the account system from the project. Hence, the game no longer requires any user data.

Developer Specifications

Developed with structured development approach	**Not Met** Although the major project documentation is styled around the stages of the structured development approach, the project itself was not developed with this approach. Instead of the phases of understanding the problem, planning and designing, implementing, testing and evaluating and maintaining, implementation took place parallel to the other phases.
Versatile and efficient data structures	**Met** The game was developed with object oriented programming to maximise the versatility of data structures. In addition to that, variables were stored in such efficient data structures as arrays, records, and arrays of records.
Minimised errors, with useful error messages printed to browser console when needed	**Not Met** Errors have been minimised. However, useful error logs in the project have largely been overlooked. Despite this, the base javascript interpreter will still produce errors– albeit, with less detailed information.
Efficient algorithms with good performance and no browser freezing	**Met**

	Browser freezes have been resolved, and the algorithms are optimised to be as efficient as possible while still being readable.
Code is internally documented well and input files are externally documented well	**Met** The code is internally documented well. The code contains both detailed comments where necessary to clarify complex logic and intrinsic documentation such as indentation, meaningful variable names and whitespace. The format of input files and their usage have also been documented in the major project documentation.
Variables have consistent and precise naming with appropriate data types and scope	**Met** The variables are precisely named and follow a consistent naming convention– camelCase for normal variables, PascalCase for classes, and SCREAMING_SNAKE_CASE for constants. Data types are appropriate for all variables. Scope is managed through the ES6 module system and object-oriented programming with classes.
Users are surveyed to improve quality and usability of system	**Not Met** A user survey has not been performed.
System modelling tools are used to clarify the function of complex modules	**Met** The major project documentation contains a data flow diagram and structure chart for the most complex modules of the program.
Javascript ES6 classes are used to organise code.	**Met** ES6 classes are used in order to facilitate object oriented programming.

Bibliography

Client-Server Game Architecture - Gabriel Gambetta. 2022. Client-Server Game Architecture - Gabriel Gambetta. [ONLINE] Available at: https://www.gabrielgambetta.com/client-server-game-architecture.html. [Accessed 21 June 2022].

Bagoum. 2022. Describing Bullet Hell: Declarative Danmaku Syntax | by Bagoum | The Startup | Medium. [ONLINE] Available at: https://medium.com/swlh/describing-bullet-hell-declarative-danmaku-syntax-d912d0a2ac02. [Accessed 21 June 2022].

HTML Canvas Drawing. 2022. HTML Canvas Drawing. [ONLINE] Available at: https://www.w3schools.com/graphics/canvas_drawing.asp. [Accessed 21 June 2022].

Stack Overflow. 2022. How do I escape some html in javascript? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5251520/how-do-i-escape-some-html-in-javascript. [Accessed 21 June 2022].

Stack Overflow. 2022. Get the mouse coordinates when clicking on canvas - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/43172115/get-the-mouse-coordinates-when-clicking-on-canvas. [Accessed 21 June 2022].

Luke Ruokaismaki. 2022. JavaScript ES6: Classes. Objects in programming languages… | by Luke Ruokaismaki | Medium. [ONLINE] Available at: https://medium.com/@luke_smaki/javascript-es6-classes-8a34b0a6720a. [Accessed 21 June 2022].

Stack Overflow. 2022. jquery - Defaultdict equivalent in javascript - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/19127650/defaultdict-equivalent-in-javascript. [Accessed 21 June 2022].

Server API | Socket.IO. 2022. Server API | Socket.IO. [ONLINE] Available at: https://socket.io/docs/v3/server-api/. [Accessed 21 June 2022].

Client API | Socket.IO. 2022. Client API | Socket.IO. [ONLINE] Available at: https://socket.io/docs/v4/client-api/. [Accessed 21 June 2022].

Stack Overflow. 2022. javascript - Using Node.JS, how do I read a JSON file into (server) memory? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/10011011/using-node-js-how-do-i-read-a-json-file-into-server-memory. [Accessed 21 June 2022].

Application structure | Socket.IO. 2022. Application structure | Socket.IO. [ONLINE] Available at: https://socket.io/docs/v4/server-application-structure/. [Accessed 21 June 2022].

pierre. 2022. Beginner's Guide to Game Networking | pvigier's blog. [ONLINE] Available at: https://pvigier.github.io/2019/09/08/beginner-guide-game-networking.html. [Accessed 21 June 2022].

Game Development Stack Exchange. 2022. networking - Networked projectiles in an authoritative server - Game Development Stack Exchange. [ONLINE] Available at: https://gamedev.stackexchange.com/questions/124364/networked-projectiles-in-an-authoritative-server. [Accessed 21 June 2022].

Stack Overflow. 2022. javascript - Simple Button in HTML5 canvas - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/24384368/simple-button-in-html5-canvas. [Accessed 21 June 2022].

CSS-Tricks. 2022. A Complete Guide to Flexbox | CSS-Tricks - CSS-Tricks. [ONLINE] Available at: https://css-tricks.com/snippets/css/a-guide-to-flexbox/. [Accessed 21 June 2022].

JavaScript modules - JavaScript | MDN. 2022. JavaScript modules - JavaScript | MDN. [ONLINE] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules. [Accessed 21 June 2022].

Logbook

Date (of entry)	**Description of Issue/Task (What I've done)**	**Problems encountered /What I should have done (Reflection)**	**What I need to do (Forward planning)**	**Completion sign-off (Issue resolved /Task completed)**
December 12, 2021	Started and finished work on a basic socket-io application (a chat channel) to figure out how the library worked Began work on the game itself, movement of player characters accomplished but without networking tricks so very unsmooth. First attempt at movement networking tricks.	Code structure is probably unsustainable. However, planning out the entire project is a doomed venture so not much to be done. Movement networking is difficult and confusing. Should have made more attempts to understand the theory behind movement networking before attempting.	Should consult with more movement networking articles and look at some example code.	done
December 13th-Jan 8th 2022	Continued to try and debug movement networking, still can't tell why the system is broken.	Despite attempting several different solutions, the game is not working. Possibly should have just rewritten everything and compared it with an example of movement networking.	Need to consult with more movement networking articles and look at some more example code.	done
Jan 9th 2022	Fixed movement networking by creating a queue of server packets and preventing the server from interrupting the client at random points during the game loop. Movement on client-side is now smooth	The solution took a while to figure out. Should have seen earlier that my implementation had to be significantly different from other examples due to socket-io and js being asynchronous.	Continue with movement, which will have to be altered to fit rotation of the camera and collision with entities.	done

Jan 10th 2022	Did reading on javascript conventions, realised that code was written without javascript's typing system (let vs const vs var) or ES6 modules. Rewrote and split up code into multiple files using both features.	Code was written poorly as a result of my lack of knowledge of javascript. Should read more into javascript before continuing the project.	Do reading into javascript, continue with movement.	done
Jan 15th 2022	Added camera rotation to movement and rendering.	Rendering took a while because of a simple error that I overlooked (using degrees instead of radians). Should look over my code before attempting solutions.	Continue with collision.	done
Jan 22th 2022	Added walls & collision, as well as a map-loading system to create these walls in the correct places.	Collision was complicated as a result of non-cardinal movement (because the player could rotate the camera and their movement). Should have attempted to break down this movement into x and y vectors. The entire map is loaded into memory, which has negative impacts on performance because of the amount of data that has to be iterated over.	Change networking to only send chunks of entities so large amounts of walls don't deteriorate FPS Add enemies.	done
Jan 25th 2022	Added chunking of entities so the client no longer has the entire map loaded into memory, only a chunk of the map sent by the server. Normalised movement vectors so that diagonal	Chunking puts significant strain on the server. Should make chunking server-wide so there are pre-prepared chunks ready for each client.	Add enemies.	done

	movement isn't faster than normal.			
Jan 26th 2022	Attempted to write server-side projectiles.	Server-side projectile networking is very undocumented and confusing to understand. Should make a prototype of server-synced projectiles outside of this project just to do rapid testing of possible solutions.	Continue to attempt projectile networking.	done
Jan 27th-Jan 29th 2022	Continuing to attempt server-sided projectiles.	It turns out that computers are incapable of perfectly syncing time over the internet– delays of milliseconds to seconds make server-sided projectiles using client time mixed with server time impossible. Should try to do research in hope someone has figured this out.	Continue to attempt projectile networking.	done
Jan 30th 2022	Halting work on server-sided projectiles. Projectiles are client-sided until the inevitable rewrite. Added enemies and created a declarative programming language for enemy AI.	Declarative programming language is written using a long switch-case statement. This will need to be a consideration in the future rewrite.	Add player shooting projectiles.	done
Feb 3rd-Feb 13th 2022	Because player projectiles are also projectiles, will have to resume work on server-sided projectiles.	Attempting more radical solutions to the projectile problem is impossible given the current codebase. Will have to rewrite the project soon.	Rewrite the entire game.	done

Feb 14th 2022	Did research on javascript programming conventions.		Rewrite the entire game.	done
Feb 15th-20th 2022	Did significant work in rewriting the entire game up to collision and movement, utilising ES6 classes and more splitting up of code via ES6 modules.	Rewriting in significant chunks without testing led to bugs occurring all throughout the program. Should slow down and test individual modules before implementing.	Rewrite the entire game. Figure out server-sided projectiles soon.	done
Feb 28th 2022	Thought up an idea of how to do server-side projectile networking and implemented it– it works. Project rewrite only has enemies left to implement.	The problem was the mixing of server time and client time because of computer desync. Hence, the game now only uses client time via a packet sent regularly containing client time. Clients that do not send this update packet regularly are disconnected.	Rewrite the entire game.	done
March 5th 2022	The old release processed enemy AI scripts with a long switch statement. Rewritten with ES6 modules so the functions can exist in isolation without being connected with a switch statement. Enemies added. Attempted work on database.	Database makes testing more difficult as the database will have to be open every time the game is open. Should make a seperate release without a database for testing.	Add database before attempting player shooting.	done
March 5-22nd	Halted work on database. Attempted work on the menu system.	The game client’s structure is not conducive to a menu system. Should have written the initial game with a user interface	Rewrite sections of code to fit in a menu system.	done
March 22-April 10th	Rewrote the game to incorporate a player name and login system.	The code for starting a game is scattered throughout the project,	Continue with the menu system.	done

		making further development cumbersome. Should have grouped all relevant functions in a module.		
April 10-16th	Achieved basic main menu screen.	Menu system requires a significant amount of ui data that should probably be stored more efficiently– perhaps a json file should have been used.	Continue with database	done
April 16th-22nd	Achieved ‘main menu’ screen in-game	Again, a significant amount of ui data is hardcoded to render the button. A json file may need to be implemented at some point to store these ui components.	Continue with UI	done
April 22nd - 29th May	Work on refactoring UI code to be more manageable.	Rather than a json file, the UI data is stored as a class attribute– json would probably have been more elegant.	Continue with lobby system	done
April 29th-10th May	Abandoned database, started work on server lobby system.	Scope was too big, needed to cut down on features to finish by the due date. The new game is based on a lobby system where players in a lobby dodge projectiles to be the last one standing.	Continue with lobby system	done
May 11th-15th	Finished server lobby system. Finished rendering 3d walls. Finished shooting mechanics.	Should have considered the scope of the project from the beginning, lots of over-engineered code that I thought would be useful.	Start work on options menu	done

	Finished in-game UI design.			
21st-28th May	Finished options menu.	Options menu is reset upon returning to the main screen. I may have to rewrite how that is implemented.	Add chat	done
28th May-5th June	Added chat function	Chat inputs are treated as inputs into the game– for example, typing w in the chat moves the player forward. This needs to be fixed.	add chat commands, fix chat bug	done
5th-12th June	Chat commands added to start game, chat bug fixed	More commands may be needed, potentially one to get a short help message and one to send a support ticket	add new commands	done
12th-16th June	New commands added	Commands are currently structured as a long switch case statement, may need to change.	fix bugs	done
16th-20th June	Bug fixes, networking synced better when joining lobby	Further testing may be needed with internet connections that are particularly slow. Project is effectively complete.	complete.	done