Department of Computer Science

Individual Project - CS3IP16

RenegadeSurvivor.co.uk - an online, text based, multiplayer game (MUD)

Luc Dexter

23021793

Supervisor: Patrick Parslow

Date of SUBMISSION

Abstract

Glossary of Terms and Abbreviations

Table of Contents

Contents

[1. Introduction 3](#_Toc511579549)

[2. Problem Articulation / Technical Specification 4](#_Toc511579550)

[3. Literature Review 6](#_Toc511579551)

[3.1 Delivery 6](#_Toc511579552)

[3.1.1 Flash Client 6](#_Toc511579553)

[3.1.2 Telnet 6](#_Toc511579554)

[3.1.3 MUD Client 6](#_Toc511579555)

[3.1.4 Web Client 6](#_Toc511579556)

[3.2 Connecting Players 7](#_Toc511579557)

[3.2.1 Socket.io 7](#_Toc511579558)

[3.2.2 Pusher 7](#_Toc511579559)

[3.2.3 HTML5 Custom WebSocket Solution 7](#_Toc511579560)

[3.3 Updating Client Content 7](#_Toc511579561)

[3.3.1 Angular 4 8](#_Toc511579562)

[3.3.2 ReactJS 8](#_Toc511579563)

[3.4 Running the Game 8](#_Toc511579564)

[3.4.1 NodeJS and PM2 8](#_Toc511579565)

[3.4.2 MUD Runner Software 8](#_Toc511579566)

[3.5 Path Finding 8](#_Toc511579567)

[3.5.1 Dijkstra’s Algorithm 9](#_Toc511579568)

[3.5.2 A\* Search Algorithm 9](#_Toc511579569)

[4. The Solution Approach 9](#_Toc511579570)

[4.1 Browser or telnet delivery 9](#_Toc511579571)

[4.2 Connecting players in browsers 10](#_Toc511579572)

[4.3 Updating client content in browsers 11](#_Toc511579573)

[4.4 Running the game 11](#_Toc511579574)

[4.5 The Solution 12](#_Toc511579575)

[5. Implementation 13](#_Toc511579576)

[5.1 RenegadeSurvivor.co.uk 13](#_Toc511579577)

[5.2 Design 14](#_Toc511579578)

[5.2.1 Use Case 14](#_Toc511579579)

[5.2.2 System Design 14](#_Toc511579580)

[5.2.3 Activities 15](#_Toc511579581)

[5. 15](#_Toc511579582)

[5.2.4 Classes 15](#_Toc511579583)

[5.2.5 Entities 15](#_Toc511579584)

[5.2.6 Wireframes 15](#_Toc511579585)

[5.2.7 Algorithms 15](#_Toc511579586)

[6. Testing: Verification and Validation 16](#_Toc511579587)

[6.1 As active character 16](#_Toc511579588)

[6.2 As a character observing another player 22](#_Toc511579589)

[7. Discussion: Contribution and Reflection 26](#_Toc511579590)

[8. Social, Legal, Health & Safety and Ethical Issues 26](#_Toc511579591)

[9. Conclusion and Future Improvements 26](#_Toc511579592)

[10. References 26](#_Toc511579593)

[11. Appendices 28](#_Toc511579594)

# Introduction

A MUD (multi user domain) [1] is a text based, interactive, online game. Multiple players can create characters and join a real-time virtual world. In it, they complete quests, role-play, fight other players and find AI characters to interact with. It’s a genre of gaming that has been around since the 1990s.

MUD worlds consist of a collection of interconnected rooms that form a virtual space. Players move between these rooms and traverse the world by typing commands with the keyboard. The world usually has a theme or setting which informs gameplay, characters and goals. Their purposes have been both educational and for the sake of entertainment.

So why MUDs? The simplicity of the interface makes them great spaces for imagination, diversity and quantity of content. In a graphical game, only worlds or gameplay that have been graphically produced can been shown. In a MUD however, the world is generated through imagination inspired by the text. This means that different MUDs can have the similar code and still be radically different in the expression of their worlds. In one, player’s might throw fireballs, in the other they might fire arrows, in another they might fire a gun. The code would look very similar; get strings from client to server and vice versa. The player’s experience would be radically different.

An educational MUD called ‘Diversity University’ [2] was created in 1993 by Jeanne McWhorter. Players could access a virtual university and even attend lectures and classes that were ‘held’ in its virtual world. Students even created Dante’s Inferno [3] inside the MUD to learn about it.

Complex human interaction and systems can be produced by the community of players in ways that graphical games cannot replicate. An example of this is Achaea [4] a MUD that has six player run cities with their own councils, governments, ministries and cultures. The social interaction that this game allows players to engage is only possible because the medium allows such freedom.

Freedom of imagination is the strength of all MUDs, the freedom to create the virtual world in the minds of the players.

From Mume [5] which represents Tolkien’s Middle Earth, to Discworld Mud [6] which depicts Terry Pratchett’s Discworld, to the many other virtual worlds [7] [8] [9] [10] created in MUDs, the complexity, detail and size of the worlds is phenomenal. Players can explore, socialise, build and archive to a serious degree of immersion and detail. From complex AI interaction to player run city states to educational literary works, the freedom with which MUDs can express virtual worlds is great.

However, MUDs have some common limitations, which inhibit that freedom of imagination and constrain a player’s sense of immersion in the virtual world, their sense of ‘being there’.

RenegadeSurvivor.co.uk has been built with the sole purpose of overcoming one of these limitations.

# Problem Articulation / Technical Specification

This problem is that in MUDs, the interactivity between rooms is severely limited. When one player is in one room, they have absolutely no indication of what is going on in another room.

Sometimes this makes sense. If the rooms are supposedly miles apart in the virtual world then players shouldn’t know what’s going on in the other room. But at other times it makes no sense at all. What if the rooms were adjacent, separated only by a plasterboard wall? A player in room A wouldn’t be able to see into room B, but they might be able to hear what’s going on. What if the wall had a window? A player could see into room B from room A but not access it. The lack of interactivity between rooms limits a worlds sense of reality.

The people who are affected by this problem are the players and the world builders. The players are affected because their sense of being in the world is limited. World builders are affected by this because the worlds they can create are limited to a series of connected capsules.

One example of this, is in Achaea [4]. A player can become a dragon when they reach a certain level. This is a very exciting feature. This means that in one room a dragon can be fighting a horde of AI agents in the world with its fire and claws. If another player were in an adjacent room, surely, they would see the battle going on, hear its roars, smell its fire. But that’s not the case. Their only indication that something is going on is that another player is in the room because of the map. Player’s abilities to sense the world around them is constrained. Their imaginations are only ever informed that they need to render the game, at the point of entering a room.

Another example, is in Mume [5]. Players can aspire to become wizards of great power. The level of their spells increases as they level up which means at high levels their spells can be very impressive. Imagine there was a fire storm spell that created a fierce storm of flames. In terms of a player’s imagination, that spell’s description would really fire the motors of their mind, painting an elaborate picture of them using the spell. That picture is somewhat tainted when the effects of such a spell are limited to one room. The spell might conjure a tornado of fire thirty feet wide and as hot as volcano, but if it only effects the small wooden shed the player casts it in, the sense of grandeur is somewhat deadened, especially if the wooden walls of the shed survive the scorching. All that power and it couldn’t even breach the walls of a wooden shed.

In terms of educational uses, a space where learners can watch each other learning is very important. Let’s imagine an educational MUD like Diversity University [2]. If players were learning how to undertake some task, let’s say build a motor engine, watching other player’s progress is a part of that learning. One player might already have an engine running in one room, but an ‘instructor’ play can’t hear it they will never know unless they check on that room.

The following criteria has been written to validate whether this problem has been solved by a proposed solution.

1. Players should have a range of senses which they can use to detect the ongoing action in other rooms. Whether they can sense something in other rooms should be determined by distance from the room, the world and the strength of the action being sensed.
2. The MUD framework should allow world builders enough freedom to create rooms that allow and block these senses to varying degrees, regardless of the flavour of theme of the world.
3. Players should have commands available to them which have cross-room impacts. When they perform such actions, the consequences should be felt across the other rooms, changing a rooms state or effecting detectability.
4. Players should be able to interact with the objects in the world in such a way that the objects impact their likelihood of being detected by other characters. These should both increase and decrease their detectability.

The vision is that a solution would provide techniques or a framework by which the interactivity between rooms can be increased. Where before a player could not detect the raging battle going on ahead in their world until it is too late, a solution would give them means of doing so, enhancing the freedom of their imagination. They might rush in a join the fight, they might hide in the shadows and pick off assailants from cover. Where before world builders were limited to creating a connected world of capsules, a solution would break down the walls between these capsules, meaning more object properties can be simulated, allowing for objects such as windows or thin walls.

# Literature Review

Surrounding the question of how such a system might be implemented, lies a few core factors. Each factor is evaluated below along with the potential technologies that could be used in the final implementation.

## Delivery

The question of how a player gains access to the MUD is very important when considering the problem. The wrong kind of method of delivery will affect the limitations on how cross-room interaction can be implemented. Options were considered by researching existing MUDs.

### Flash Client

Some MUDs such as Aardwolf [9] use flash clients. This means that on the play page of the website the game runs in one flash content section. Flash is a mature technology and there are some drawbacks and showstoppers when considering its usefulness in solving the problem. The main one is longevity. The number of websites using flash is going down [11]. Why use a declining technology? In an article by Apple’s former CEO Steve Jobs [12] highlighted his reservations about flash and how Apple does not support flash on their smaller devices.

### Telnet

Telnet is a mature internet protocol which users can use on UNIX systems to access remote networks and machines [13]. Legendsofkallisti [14] is a MUD which supports this method of delivery. The problem with Telnet is that has no graphic or colour support [15]. While for a MUD server it is not a disaster because the main interface is an input box with a larger output text section, for cross-room interaction it’s a big issue. If all content is limited to the output box there is no way of displaying a map or having other sections to display what might be occurring in other rooms.

### MUD Client

Mume [5] and many other MUDs use MUD clients. These are programs that players install onto their machines that allow them to access a variety of MUDs. The player enters the web address and port and then the software handles the connection. However, these are really just glorified telnet programs [16]. They may have certain MUD tailored features, but the technology is the same. In terms of cross-room interaction the ability to make headway is limited by the technology and features of each MUD client.

### Web Client

Most MUDs running today use a web client such as Achaea [4] and Forgotten Kingdoms [8]. A player accesses the website, creates a character and is then redirected to a play page within their browser. The play page is just a webpage and therefore can support other content around the main game interface. This is by far the best delivery method in terms of player effort. The system handles all the client server configuration on the site. In terms of influencing the cross-room interaction problem this option gives the most freedom to display additional information to players relating to other rooms.

## Connecting Players

The different delivery options create certain question regarding how players connect to each other. For telnet and MUD clients this is already handled by the delivery method. For flash and web clients and solution to how players connect to the server is required. When one player leaves a room on their client, another player’s client needs to receive that information from the server and update accordingly. To solve this, three solutions were considered.

### Socket.io

Socket.io [17] is real time engine that supports client-server connections. Its main advantage over technologies such as AJAX is that it removes long-polling [18] which means sending HTTP data requests regularly. It provides a server and client API which allows connections to a server to be managed. In terms of enabling a web client of flash client this technology provides a means for players to connect to the host server within their browsers. Both clients and server can emit event between each other allowing for the creation of one consistent world.

### Pusher

Pusher [19] performs the same function as Socket.io. It provides an API a triangular architecture [20] where the whenever the server emits a message, it uses the Pusher API to send it to Pusher. Pusher uses WebSockets to update client browsers. Client browsers use existing HTTP to update the server. Its purpose in this project would be to provide the vehicle by which a consistent world can be created through client server connections for web clients and flash clients.

### HTML5 Custom WebSocket Solution

Another solution would be to write custom server client software that utilises WebSockets [21]. This would mean writing a TCP application in a server-side language to handle client’s connections to the server. The main downside to this approach is that while it gives freedom to the developer of the project in terms of how connections are managed, that freedom is meaningless in terms of cross-room interaction in MUDs. It doesn’t matter how the data gets from sever to client and back. What matters is how the data is handled when it arrives.

## Updating Client Content

Therefore, for web clients, the question of how to update the clients content within the browser is raised. When the data is sent to the client, how the browser displays the data in real time must be answered. For example, it would be unacceptable to reload the browser every time it receives from the server. To solve this issue for web clients, two JavaScript frameworks were reviewed.

Both are used to make single page web apps. This means all content is loaded onto one page and updated dynamically [22]. When one part is accessed by a user, only the content for that part is loaded, meaning no page refresh, because there is only one page.

### Angular 4

Angular 4 [23] has more features. These include dependency injection, templates, routing, forms, component CSS encapsulation, XSS protection and utilities for unit-testing components [24]. In terms of creating an interface that can deal displaying a UI that informs the user of their characters senses the strong component model provides a good choice.

### ReactJS

ReactJS [25] has a weaker component model and less features [24]. This means there is greater flexibility in development because more code must be written alongside it, but also more effort.

## Running the Game

To run the game, the server must execute the game code continually. For mud clients and telnet clients, this means connecting to the program port on the server, where the program is running. The telnet/mud connection handles the handshake and allows the client to interact with the game code already running. For web clients the browser sends a HTTP request to the port on the server machine to form a connection. The server machine runs code that manages the client to server connection. Similarly, the player connects to the server’s port where the game is running.

Two options were investigated when looking into how to run the game on the server.

### NodeJS and PM2

The first was NodeJs [26]. NodeJs runs JavaScript. It has an extensive library of packages that it can call and run. One of these is PM2 [27]. PM2 is a production process manager for Node which continually runs JavaScript code. When there is an issue or a drop out it restarts the JavaScript so that it runs continually. The usages of this in MUDs lie in the running of the game.

### MUD Runner Software

The second option was to use a pre-existing MUD runner [28]. This would mean using a prebuilt software to run the game. One of these is called CircleMUD [29] a system written by Jeremy Elson. This would mean that a lot of the MUD ground work will have already been written and therefore save time.

## Path Finding

The problem of characters having senses that detect things in other rooms requires a path finding algorithm. Whether a character can hear a sound in another room for example, depends on several factors, such as loudness, distance and anything blocking the path between origin and receiver. Whether or not the sound can be detected might depend on its loudness versus the distance of the path back to the hearer.

### Dijkstra’s Algorithm

Dijkstra’s shortest path algorithm [29] runs on a weighted graph. It starts with an initial node and goal node and finds the shortest path between them. It uses a visited set and an unvisited set. Every node has a value that defines how long it took to reach that node. These values are initially set to infinity but as the algorithm progresses they are updated with the lowest possible value. This gives a selection of lowest paths each with an overall value at every stage of the algorithm. When there are no more nodes on the unvisited list, the lowest value path is chosen. Therefore, it always provides the best solution.

### A\* Search Algorithm

A\* search algorithm [30] is a form of Dijkstra’s algorithm but improves it by using a heuristic. The heuristic could be Euclidean distance, Manhattan distance, Chebyshev distance or any other. The function is:

G(n) is the value of the route between two nodes and h(n) is the heuristic that estimates the overall cost of the node to the goal node. This means that where Dijkstra’s algorithm will check all possibilities, A\* search eliminates paths from consideration based on the heuristic.

# The Solution Approach

The problem is that cross-room interactivity in MUDs is severely limited. A solution needs to provide players with a range of cross-room senses, commands and object interactions and world builders with tools to allow their worlds to make this possible.

## Browser or telnet delivery

When looking at the options two branches appear very early on regarding delivery. Should players connect using telnet or in a browser? These are compared in the following table.

|  |  |  |
| --- | --- | --- |
| Client Solution | Advantages | Disadvantages |
| Telnet  *(mud client or telnet)* | It solves a lot of problems in terms of connecting players and how the game is displayed to a player.  MUD Clients already exist so time can be saved in development. | There is only one space given to the player, one text input and output. Questions such as how to properly display senses are raised. |
| Browser  *(flash or web client)* | The web page on which the game runs provides space to give the user additional information. | More development effort is required to build the framework by which the game will be delivered. Questions such as how to connect players or update connect on the page are raised. |

When comparing these options each solution raises issues. The most pertinent of these are seen in the disadvantages column.

Firstly, how can a telnet solution provide enough interface to display senses without overload? Should text be displayed that informs the user what they are sensing in other rooms? Surely the amount of data would overload the player. Line after line of text displaying other room data would detract from the experience within the current room. If a player is fighting in their current room, then most of the time they don’t care about a vague sound two rooms away. Using line after line to display that information would destroy the experience in the current room. However, if for some reason they do care about the ongoing in another room, maybe for game related reasons, maybe a target is escaping, then they need that information. In short, they need to be given the tools to choose to sense in other rooms.

But the only tool that telnet could offer would be a command to sense, for example a listen command. The issue with this is that our own human senses are not triggered by commands, they are always functioning. We can block out or miss some senses when another sense overpowers, but we don’t turn them on an off like commands. The system needs to make it so that a player is always listening but based on relevance can block out data.

In direct contrast to telnet, using a browser can solve this problem. Around the input and output text, the focus of the player’s attention, additional information can be displayed. Thus, senses can be given to the player which are always there but can be ignored when the user deems them irreverent. The player would have the power to decide what is important by what they look at on the web page. If they are fighting in a room, they might miss data displayed on a map. Equally, they could deem that data important enough to check the sense data.

Therefore, telnet cannot solve the senses issue in such a way that a browser delivered MUD can.

However, the second issue raised in the table is that while a browser-based delivery method solved the senses problem, it means that other problems need to be solved, such as connecting players and updating the web page content.

## Connecting players in browsers

Using browser delivery may solve the senses issue but it means a solution to how to connect players is required.

Three were analysed, Socket.io, Pusher and creating a custom solution using HTML 5 WebSockets. Very soon the latter was discounted because the prior two accomplish everything that needs to be accomplished and there’s no reason to reinvent the wheel. Also, the decision about how players are connected in browsers does not directly affect the problem of cross-room interaction. Both socket.io and pusher handle getting data from clients to the server and back again. All that matters is flexibility in tying in their solution to the MUD.

## Updating client content in browsers

Using browser delivery also means a solution for how to update a web page is required. Flash can update itself, so running the client using Flash player goes some way to solve this issue. But is does not go the whole way. For example, when talking about the senses issue, providing the player with a range of senses that doesn’t overload them means having information outside the main text box. Using Flash would mean that a solution would be required to comprehend what was happening inside the flash player and then feeding that back to the web page, which is a problem that is not worth solving. Instead a web client with a HTML text box is much better.

However, this means that a solution for updating content on the web page is required. Therefore, two single page web app frameworks were considered. These were ReactJS and Angular 4. They both solve the same problem, how to dynamically update web page content. There is not much between them, but one main factor is the component model. Angular 4 has a strong component model [24] which means that pieces of content can componentised and the reused. For displaying additional senses information this is very important. If the player has hearing data in one component, sound data in another etc. during development or on user request, the locations of these on the page might change. Angular 4’s components can be easily moved on from one part of a web page to another. As the method of displaying senses data is crucial to the solving of the main problem, freedom to provide different layout is paramount.

## Running the game

The previous sections have looked at the running of the client. This section looks at the running of the server. Crucially, the vehicle for the running of the game needs to allow for greater cross-room interaction. Two aspects of the problem are important here. Firstly, the outworking of the results of commands that effect multiple rooms. Secondly, a framework that give world builders tools to create worlds that allow greater interaction with regards to senses and detection. The two options considered were NodeJS and using MUD runner software [28].

|  |  |  |
| --- | --- | --- |
| GAME RUNNER | advantages | disadvantages |
| NodeJS | There is complete freedom to create the core functionality to solve the problem using JavaScript.  Socket.io is installed via NodeJS so would require no extra code to connect it to the game. | MUD needs to be built completely from scratch, meaning less time to work on the world itself. |
| MUD RUNNer software | Much of the core functionality is already done meaning development effort can be focussed elsewhere. | The MUD may not be able to solve the problem. Core functionality may be restricted or limited to the norms of the MUD genre meaning cross-room interaction improvements cannot be built in. |

The crux of the issue is that while using NodeJs will be more effort, the problem can definitely be solved. Whereas using a MUD runner would being tied into design decisions that may affect whether the runner can support cross-room interaction to sufficient standards. One main bone of contention is what the room object contains and how modifiable it is. Finding a suitable enough MUD runner may not even be possible.

## The Solution

Therefore, with all these things considered the chosen approach will be as follows. The game will be run using NodeJS to give freedom to the solution during development. A web-based client will be used as a method for players to access the MUD so that a range of sense data can be displayed. This web-based client will connect to the server using Socket.io because socket.io runs on NodeJS. Updating the clients content will be managed by Angular 4 because of its strong component model [24] which gives it its flexibility in creating UI.

The criteria by which this solution will have succeeded is as follows:

1. The range of senses will be displayed on an angular 4 site that places sense data next to the main play interface. This will allow players to ‘sense’ cross-room activity.
2. The game runner will allow a world to be loaded that has means of blocking and allowing players to sense activity in other rooms based on terrain.
3. Players will be able to input commands that have cross-room impacts, changing other room’s states and effecting detectability.
4. Objects will allow players to hide in/under/behind them to affect their cross-room detectability.

# Implementation

## RenegadeSurvivor.co.uk

RenegadeSurvivor.co.uk is a solution to the problem of limited cross-room interaction in MUDS. The premise of the world is that an overlord and their bandits roam a warehouse area. Players must enter the world and defeat the overlord to win. Its features are designed to solve the problem.

These features are:

|  |  |
| --- | --- |
| Feature | Description |
| Face direction | Characters have a direction that they are facing, i.e. north, east, south or west. This impacts a character’s sight sense, enabling them to be snuck up on by others. |
| sight sense | Characters can see what other characters are doing in rooms ahead of them and in their peripheral vision. |
| Sound sense | Characters can hear characters in other rooms without seeing them depending on loudness. |
| smell sense | Characters can leave smells after eating which make them detectable to other players even if they are not heard or seen. |
| world that allows sense interaction | Some walls in a room allow sight and others block sight. Some walls in a room allow sound and others block sound. Some walls in a room allow smell and others block smell. |
| hide-in objects | Characters can hide in or under or behind objects, making themselves less detectable to others. |
| ai behaviour responding to senses | The AI bandits use the sound of characters actions to plan their routes if they can hear them. The loudness and distance impact whether they can. |
| interactive lighting | Characters can impact the visibility of rooms by turning the lights on and off. Rooms are linked on ‘circuits’ so one switch can impact many rooms. |
| time simualtion | Every action has a time value, meaning they take a certain time to execute. |
| room imapcting commands | In every game there is a bomb which can be detonated. This detonation causes the rooms to change their sight, access, smell and sound barrier qualities. A explosion can cause a storeroom with four sight blocking walls to become an open space full of rubble. |

## Design

The MUD consists of two parts. There is a client game interface which functions as the access point for players to join the game. There is a server which runs the game.

### Use Case

Players log into the site to access the game interface through the client, creating a character if required. The server runs the game. Players input commands which are given to the server. The server resolves the game and the results are displayed to players. The server also updates the game regularly and gives the results to the client to display.

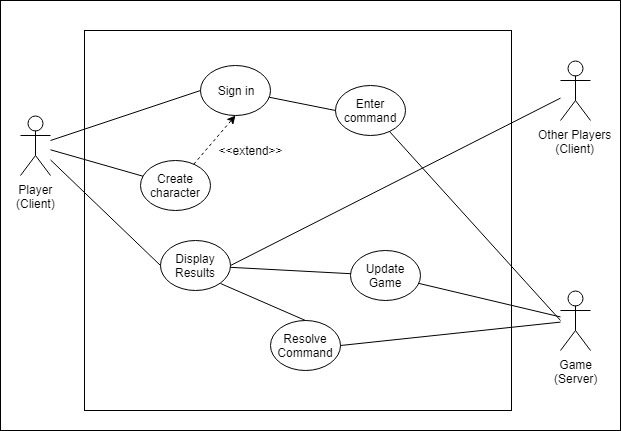


Figure 1: *Use Case Diagram*

### System Design

The server uses NodeJS [26] to install packages and allow the executing of JavaScript. Git [32] is used to update the client and server code. PM2 [27] is a package that runs the JavaScript game code without interruption. Socket.io [17] is installed via the node package manager on both the game and client code to handle communication between them. The client code is built using the Angular[22] build commands. Nginx [33] is a HTTP server that is used to host the angular web app/RenegadeSurvivor.co.uk. Players use the client to access the MUD which uses socket.io to communicate with the game running on the server.

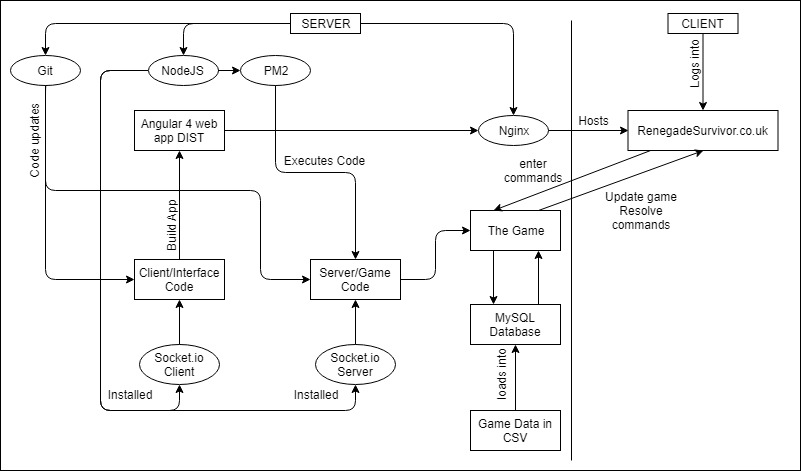


Figure 2: *System Design Diagram.*

### Activities

The activities displayed on the use case diagram were used to create activity diagrams. An example is shown below.

#### Enter commands

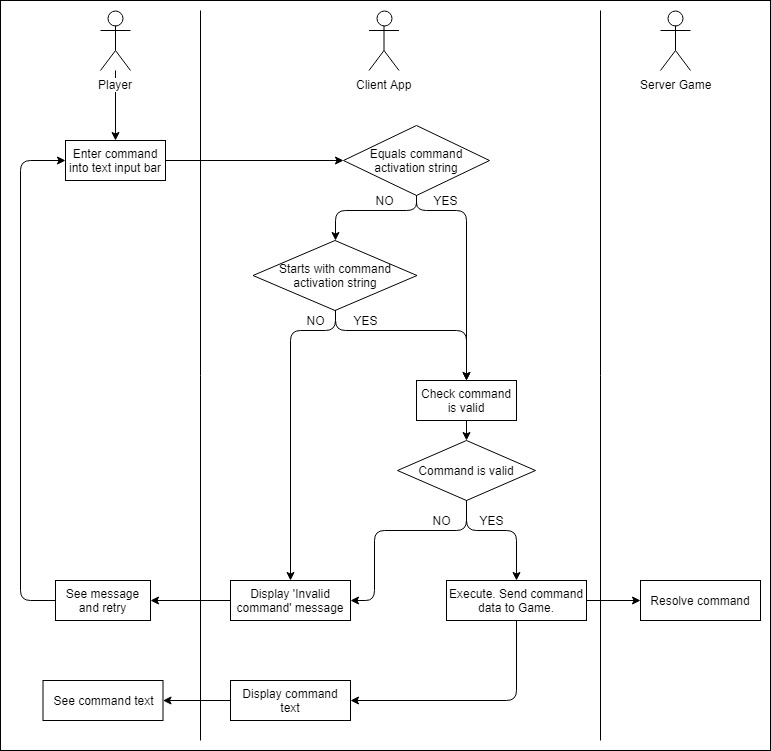


Figure 3: *Enter Command activity diagram*

### Classes

The game/server code is run using NodeJS/PM2. The classes used are displayed below.

A close up of text on a white background

Description generated with very high confidence

Figure 4: *Class Diagram. Overview of Game/server code*

A close up of text on a white background

Description generated with very high confidence

Figure 4 continued

A close up of text on a white background

Description generated with very high confidence

Figure 4 continued

|  |  |
| --- | --- |
| class | purpose |
| game | Run the game.  Contain all relevant data pertaining to the game. |
| map | Retrieve map data from the database and create a list of room objects. This list of room objects is then stored in the Game object. |
| sound | Hold data of a sound created in the world. |
| action | Hold data of an action being undertaken in the world. |
| room | Represent a room the user enters. |
| exit | Form a bridge between two rooms by which access can be gained. |
| boundary | Represent the boundary of a room, north, south, east or west. These boundaries can simulate a range of objects such as windows, nothing, solid walls. |
| Light | Represent a light switch in the game, for toggling all the rooms on the circuit on and off. |
| character | Represent a character with all their stats and game status. |
| item | Represent an item a room. |
| commandlist | Give the character a list of commands that they can be uses in the interface to check against input. |
| command | Represent an abstract, generic command that can be called. |
| Targetcommand | Inherits from Command.  Useful for commands that have a target associated with them, such as ‘shoot Brian’. The target could change. |
| statuscommand | Inherits from Command.  Useful for commands that have a status associated with them, such as ‘turn lights on’. The status ‘on’ or ‘off’ could change or be different depending on the command. |
| targetandstatuscommand | Inherits from Command.  Useful for commands that have a status and a target associated with them, such as ‘hide in the cabinet’. The target can change, and the status might relate to the target such as ‘in’ or ‘under’. |
| outlaw | Inherits from Character.  Represent an AI character and manage their behaviour |
| overlord | Inherits from Outlaw. |
| bomb | Inherits from Item. |
| hide | Inherits from TargetAndStatusCommand. |
| LightCommand | Inherits from StatusCommand. |
| pickup | Inherits from TargetCommand. |
| search | Inherits from TargetCommand. |
| shoot | Inherits from TargetCommand. |
| detonate | Inherits from TargetCommand. |
| eat | Inherits from TargetCommand. |
| drop | Inherits from TargetCommand. |
| server | Set up site on port 3000.  Connect to the database.  Build the game and map.  Manage when clients connect to the game via socket.io |
| astarsearch | Find the quickest route on between two points on the map |
| create-character | Set up and add a Character to the database |
| check-sign-in | Verify that the user login details are correct. |
| walk | Inherits from Command.  Enable character to move between rooms. |
| run | Inherits from Command.  Enable character to move between rooms. |
| sneak | Inherits from Command.  Enable character to move between rooms. |

### Angular 4 Component Architecture

The client/interface is an angular 4 web app. Each component fits into the component tree and is shown below. The top component Is AppRoot which is contained in the index page. That calls AppComponent which contains the navigation bar and handles the navigation through router outlets. The rest are shown below.

A screenshot of a cell phone

Description generated with very high confidence

Figure 5: *Angular 4 component tree*

|  |  |
| --- | --- |
| Component | purpose |
| home | Acts as a landing page to describe features of the MUD. |
| play | Determines if the player is signed in and displays sign in or play screen depending on that outcome.  Connects to Socket.io. |
| Sign in | Displays form to allow user to sign in |
| create character | Displays form to allow user to create character |
| play screen | Acts as the container for the two parts of the interface game interface and game details. |
| game interface | Handles the text box input and output.  Displays character details.  Verifies that commands are valid, in terms of wording and the game. |
| game details | Displays room, player and visible lists. |
| map | Displays the map. |
| additional info | Displays game events. |

### Angular 4 Services

The angular app has a collection of services which contain data that persist while the app runs, whereas component data only persists if the components are visible.

|  |  |
| --- | --- |
| service | purpose |
| astar-search | Contains a method to find the fastest route between two points on the map.  Used to calculate if a character can sense a sound or smell based or distance and loudness. |
| current-room | Used to store the data of the room the character is in. |
| playerlist | Used to store a list of players who are online. |
| sign-in | Used to store the characters stats and game data. |

### Entities

A screenshot of a social media post

Description generated with very high confidence

Figure 6 : *Entity Relationship Diagram*

### Wireframes

The design the interface angular app wireframes were produced.

A screenshot of a cell phone

Description generated with very high confidence

Figure 7: *Interface layout wireframe*

### Algorithms

A star search was implemented to calculate whether senses can be detected based of loudness and distance etc. Jeff Allen illustrates the algorithm on his site[34].

A close up of a map

Description generated with high confidence

Figure 8*: A\* search [34]*

In RenegadeSurvivor.co.uk the heuristic used is the Manhattan distance. This means calculating the distance up and then across, ignoring terrain.

## Implementing the designs

These are some examples of how the designs were implemented in code.

### Interface

Below is what the interface looks like.

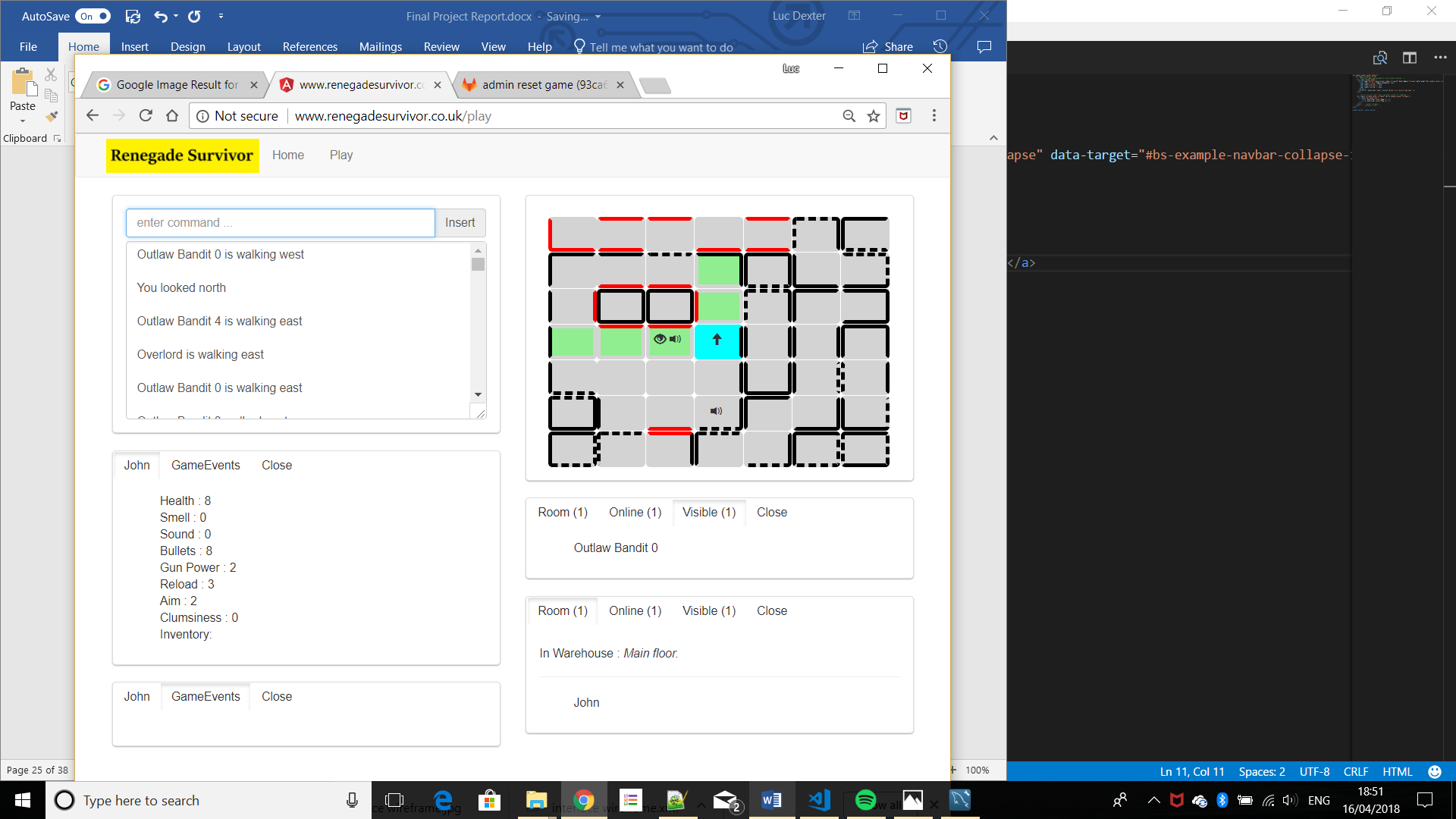


Figure 9: *the client/interface*

### Handling command input on the client

Explanation of what makes up a component

### Resolving commands on the server

Explanation of what makes up a component

### Run

Explanation of what makes up a component

### Building command list

Explanation of what makes up a component

### Updating the map

Explanation of what makes up a component

### Displaying results of other characters actions

Explanation of what makes up a component

# Testing: Verification and Validation

Tests were carried out manually. They were Alpha tests. These were the test cases by which the solution was validated against. As this is a multiplayer world some tests were conducted as the active player while others were conducted as a watching player.

## As active character

|  |  |  |  |
| --- | --- | --- | --- |
| feature | test | expectations | outcome |
| Create character | Create character with all fields completed | Filling out fields and pressing play redirects to game.  New character is created. | pass |
|  | Create character without all fields completed | Missing username or password fields are reported to user. | pass |
| sign in | Sign in with correct details | Player is redirected to the game. | pass |
|  | Sign in with incorrect details | Missing username or password fields are reported to user. | No reports to user |
|  | Sign in with correct username but without password | Missing password field is reported to user. | No reports to user |
| look | Look north. | The character arrow is up. | Pass |
|  |  | Green visibility squares update. | pass |
|  |  | The visible list updates. | pass |
|  | Look east. | The character arrow is right. | Pass |
|  |  | Green visibility squares update. | Pass |
|  |  | The visible list updates. | Pass |
|  | Look south. | The character arrow is down. | Pass |
|  |  | Green visibility squares update. | Pass |
|  |  | The visible list updates. | Pass |
|  | Look west. | The character arrow is left. | Pass |
|  |  | Green visibility squares update. | Pass |
|  |  | The visible list updates. | pass |
| WALK | Walk north. | Can walk north. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Walk east. | Can walk east. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Walk south. | Can walk south. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Walk west. | Can walk west. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Walk through door. | Can walk through a door into another room. | pass |
|  | Attempt to walk through wall. | No exit that direction is displayed. | Pass |
|  | Walk through non-visible walls | Can walk through non-visible rooms. | pass |
| sneak | Sneak north. | Can sneak north. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | No sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Sneak east. | Can sneak east. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | No sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Sneak south. | Can sneak south. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | No sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Sneak west. | Can sneak west. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | No sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Sneak through door. | Can sneak through a door into another room. | Pass |
|  | Attempt to sneak through wall. | No exit that direction is displayed. | Pass |
|  | Sneak through non-visible walls | Can sneak through non-visible rooms. | Pass |
| run | Run north. | Can run north. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Run east. | Can run east. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Run south. | Can run south. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Run west. | Can run west. | Pass |
|  |  | Room text is updated. | Pass |
|  |  | Visible list is updated. | Pass |
|  |  | Room list is updated. | Pass |
|  |  | A sound is produced on map. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  | Run through door. | Can run through a door into another room. | Pass |
|  | Attempt to run through wall. | No exit that direction is displayed. | Pass |
|  | Run through non-visible walls. | Can run through non-visible rooms. | Pass |
| shoot overlord | Shoot Overlord when out of view. | Invalid action is displayed when Overlord is not in sight. | Pass |
|  | Shoot Overlord when in view. | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Bullets are decreased by one. | Pass |
|  |  | A sound is produced on map. | Pass |
|  | Shoot Overlord without any bullets. | ‘No bullets’ is displayed and shot not fired. | Pass |
| shoot character | Shoot character when out of view. | Invalid action is displayed when character is not in sight. | Pass |
|  | Shoot character when in view. | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Bullets are decreased by one. | Pass |
|  |  | A sound is produced on map. | Pass |
|  | Shoot character without any bullets. | ‘No bullets’ is displayed and shot not fired. | Pass |
| search object | Search an object when it is not in room. | Invalid command is displayed. | Pass |
|  | Search an object that cannot be searched. | Invalid command is displayed. | pass |
|  | Search an object that can be searched and is in room. | When object is present and searchable, ‘found bullet’ is displayed. | Pass |
|  |  | Bullets are increased by one. | Pass |
|  |  | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | If bomb is present in room bomb is revealed instead. | pass |
| hide in an object | Hide in object when it is not in room. | Invalid command is displayed. | pass |
|  | Hide in an object that cannot be hid in. | Invalid command is displayed. | pass |
|  | Hide in an object that can be hid in and is in room. | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Green visibility squares are updated. | Pass |
|  |  | Room list is updated. | pass |
| toggle lights | Turn off lights when they are on. | All rooms on the circuit are turned off. | Pass |
|  |  | Map updates. | Pass |
|  |  | Visible list updates. | pass |
|  | Turn off lights when they are off. | displays lights already off. | Pass |
|  | Turn on lights when they are off. | All rooms on the circuit are turned on. | Pass |
|  |  | Map updates. | Pass |
|  |  | Visible list updates. | pass |
|  | Turn on lights when they are on. | displays lights already on. | pass |
| pick up an object | Pick up an object when it is not in room. | Invalid command is displayed. | Pass |
|  | Pick up an object that cannot be picked up. | Invalid command is displayed. | Pass |
|  | Pick up an object that can be picked up in and is in room. | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Object is removed from room description text. | Pass |
|  |  | Object is placed in inventory. | pass |
| Drop object | Drop an object when it is not in inventory. | Invalid command is displayed. | Pass |
|  | Drop an object that cannot be picked up or dropped. | Invalid command is displayed. | Pass |
|  | Drop an object that can be dropped in and is in inventory. | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Object is added from room description text. | Pass |
|  |  | Object is removed from inventory. | pass |
| detonate bomb | Detonate bomb in original room | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Countdown begins. | Pass |
|  |  | Rooms are impacted by explosion. | pass |
|  | Detonate bomb in inventory | Executing text is displayed. | Pass |
|  |  | Completed text is displayed. | Pass |
|  |  | Countdown begins. | Pass |
|  |  | Rooms are impacted by explosion. | pass |
|  | Detonate bomb after dropping it in a different room. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | Countdown begins. |  |
|  |  | Rooms are impacted by explosion. |  |
| eat | Eat an object when it is not in room. | Invalid command is displayed. |  |
|  | Eat an object that cannot be eaten. | Invalid command is displayed. |  |
|  | Eat an object that can be eaten and is room. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | One health points are added. |  |
|  |  | Two smell points are added. |  |
| DIE AND RESPAWN | Die and respawn after being shot | Text is displayed that informs player. |  |
|  |  | Respawned back to start room. |  |
|  |  | Scores set back to default values. |  |
|  | Die and respawn after being in explosion | Text is displayed that informs player. |  |
|  |  | Respawned back to start room. |  |
|  |  | Scores set back to default values. |  |
| WIN GAME AND RESET | Player defeats the Overlord | Win game text is displayed with character’s name |  |
|  |  | Character respawns in start room. |  |
|  |  | Explosion damage is repaired. |  |
|  |  | Bomb is replaced in game. |  |
|  |  | Characters scores are set to default. |  |

## As a character observing another player

|  |  |  |  |
| --- | --- | --- | --- |
| feature | test | expectations | outcome |
| sign in | Another player logs in and joins same room. | Character is in online list. |  |
|  |  | Character is in room list. |  |
|  |  | Character is in visible list. |  |
|  |  | Green visibility squares show an eye. |  |
|  | Another player logs in and room in visible range. | Character is in online list. |  |
|  |  | Character is in visible list. |  |
|  |  | Green visibility squares show an eye. |  |
|  | Another player logs in and is not in visible range. | Character is in online list. |  |
| WALK | Another character walks north. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character walks east. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character walks south. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character walks west. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character walks when out of sight. | A sound is produced on map. |  |
| sneak | Another character sneaks north. | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character sneaks east. | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character sneaks south. | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character sneaks west. | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character sneaks when out of sight. | No sound is produced on map. |  |
| run | Another character runs north. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character runs east. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character runs south. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character runs west. | A sound is produced on map. |  |
|  |  | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
|  |  | Map and visible list updates correctly. |  |
|  | Another character runs when out of sight. | A sound is produced on map. |  |
| shoot character | Another character shoots another character in view. | A sound is produced on map. |  |
|  |  | Executing text is displayed with both character’s name. |  |
|  |  | Completed text is displayed with both character’s name. |  |
|  | Another character shoots player in view. | Executing text is displayed with both character’s name. |  |
|  |  | Completed text is displayed with both character’s name. |  |
|  |  | Two health points are depleted. |  |
| search object | Another player searches in view. | Executing text is displayed with character’s name. |  |
|  |  | Completed text is displayed with character’s name. |  |
| hide in an object | Another player hides in an object in view. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | Green visibility squares are updated. |  |
|  |  | Room list is updated. |  |
|  |  | Visibility list is updated. |  |
| toggle lights | Another character turns off lights. | All rooms on the circuit are turned off. |  |
|  |  | Map updates. |  |
|  |  | Visible list updates. |  |
|  | Another character turns on lights. | All rooms on the circuit are turned on. |  |
|  |  | Map updates. |  |
|  |  | Visible list updates. |  |
| pick up an object | Another character picks up an object in view. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | Object is removed from room description text. |  |
| Drop object | Another character drops an object in view. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | Object is added to the room description text. |  |
| detonate bomb | Another character detonates bomb. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | Countdown begins. |  |
|  |  | Rooms are impacted by explosion. |  |
| eat | Another character eats an item in view. | Executing text is displayed. |  |
|  |  | Completed text is displayed. |  |
|  |  | Smell is displayed on map. |  |
| WIN GAME AND RESET | Another character defeats the Overlord | Win game text is displayed with character’s name |  |
|  |  | Character respawns in start room. |  |
|  |  | Explosion damage is repaired. |  |
|  |  | Bomb is replaced in game. |  |
|  |  | Characters scores are set to default. |  |

--

Manual testing and test cases. Important feature tests and reports.

# Discussion: Contribution and Reflection

Has it worked? Use images from testing to discuss this. Limitations. Provide gaps for future work.

# Social, Legal, Health & Safety and Ethical Issues

Words:

# Conclusion and Future Improvements

Draw everything together. How can it be improved?

# References

|  |  |
| --- | --- |
| [1] | R. Bartle, in *Designing Virtual Worlds*, New Riders, 2003, pp. 9-10. |
| [2] | S. Carton, in *Internet Virtual Worlds Quick Tour*, Ventana Press, 1995, pp. 138-139. |
| [3] | L. Harris, “WRITING SPACES: USING MOOs TO TEACH COMPOSITION AND LITERATURE,” 1996. [Online]. Available: http://english.ttu.edu/kairos/1.2/binder2.html?coverweb/Harris/contents.htm. [Accessed 04 April 2018]. |
| [4] | “Achaea,” [Online]. Available: https://www.achaea.com/. [Accessed 04 April 2018]. |
| [5] | “Mume,” [Online]. Available: http://www.mume.org/. [Accessed 05 April 2018]. |
| [6] | “Discworld MUD,” [Online]. Available: http://discworld.starturtle.net/lpc/. [Accessed 05 April 2018]. |
| [7] | “Lusternia,” [Online]. Available: https://www.lusternia.com/. [Accessed 05 April 2018]. |
| [8] | “Forgotten Kingdoms,” [Online]. Available: http://www.forgottenkingdoms.org/. [Accessed 05 April 2018]. |
| [9] | “Aardwolf,” [Online]. Available: https://www.aardwolf.com/. [Accessed 05 April 2018]. |
| [10] | “Materia Magica,” [Online]. Available: https://www.materiamagica.com/. [Accessed 05 April 2018]. |
| [11] | w3Techs, “Usage of Flash for websites,” [Online]. Available: https://w3techs.com/technologies/details/cp-flash/all/all. [Accessed 05 April 2018]. |
| [12] | S. Jobs, “Thoughts on Flash,” April 2010. [Online]. Available: https://www.apple.com/hotnews/thoughts-on-flash/. [Accessed 05 April 2018]. |
| [13] | “HISTORY AND FUTURE OF TELNET,” [Online]. Available: https://schoolworkhelper.net/telnet-history-purpose-advantages-disadvantages/. [Accessed 05 April 2018]. |
| [14] | “Legends of Kallisti,” [Online]. Available: http://www.legendsofkallisti.com/. [Accessed 05 April 2018]. |
| [15] | G. Henry, “Advantages and Disadvantages of Telnet,” 29 November 2013. [Online]. Available: https://www.scoop.it/t/internet-tools-by-galesha-henry. [Accessed 05 April 2018]. |
| [16] | J. R. Levine, in *More Internet for Dummies*, IDG Books, 1997, p. 199. |
| [17] | “Socket.io,” [Online]. Available: https://socket.io/. [Accessed 05 April 2018]. |
| [18] | F. Kelleher, “Understanding Socket.IO,” 10 August 2014. [Online]. Available: http://nodesource.com/blog/understanding-socketio/. [Accessed 05 April 2018]. |
| [19] | “Pusher,” [Online]. Available: https://pusher.com/. [Accessed 05 April 2018]. |
| [20] | “What is Pusher,” [Online]. Available: https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html. [Accessed 05 April 2018]. |
| [21] | “Writing WebSocket servers,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API/Writing\_WebSocket\_servers. [Accessed 05 April 2018]. |
| [22] | “Angular Single Page Applications (SPA): What are the Benefits?,” Angular, 25 January 2018. [Online]. Available: https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/. [Accessed 05 April 2018]. |
| [23] | “Angular,” [Online]. Available: https://angular.io/. [Accessed 05 April 2018]. |
| [24] | P. Jelisejevs, “React Vs Angular,” 13 March 2018. [Online]. Available: https://www.sitepoint.com/react-vs-angular/. [Accessed 05 April 2018]. |
| [25] | “React,” [Online]. Available: https://reactjs.org/. [Accessed 05 April 2018]. |
| [26] | “NodeJs,” [Online]. Available: https://nodejs.org/en/. [Accessed 05 April 2018]. |
| [27] | “PM2,” [Online]. Available: http://pm2.keymetrics.io/. [Accessed 05 April 2018]. |
| [28] | “Run Your Own MUD,” [Online]. Available: https://www.livinginternet.com/d/da\_own.htm. [Accessed 05 April 2018]. |
| [29] | T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, in *Introduction to Algorithms*, 2nd edition ed., MIT Press and McGraw-Hill, 2001, pp. 595-601. |
| [30] | P. E. Hart, N. J. Nilsson and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” IEEE, 1968. |

# Appendices