Team Project Final Report

‘Lightgunner’

Team C4:
Yohanan Ben-Gad 134428
Frederic Evans 1403746
James Davis 1359799
Joshua Holmes 1333205
Negrita Kalcheva 1190695
Joseph Thomas 1345298

Table of Contents

1. Introduction

Following is the final project report, accurately describing the development process, of the 2D arena, competitive, co-operative, fast paced, shoot-out game that is Lightgunner.

The premise of the game is taking two players, who could potentially be extremely different in terms of skills, and put them together to form a team that can utilise each others strengths, and solid teamwork, to achieve a common goal. We achieved this in the form of two players, who are caught in a fight to the death against other teams. One player acts as a Light source, has no offensive capability, and is completely dependant on his teammate to achieve victory. He can guide the shooter around the map, with his strong vision, in the hopes of eliminating the other team. So who is the shooter? This character is the offensive capability of the team, although, as the game is so dark, he must use the skills of his lighguy to navigate the map, find targets and avoid threats.

This report will cover the whole development of the product, including; how we approached to software and game design, how we structured the project, and a thorough description of the technical aspects of the game. This document also covers our Software Engineering approach to the game, in conjunction with our risk management. Finally we have an Evaluation of the project, including a project wide evaluation, a reflection on team work, and our individual experiences with the project. We have also produced a test report that explains how we tackled testing our code throughout development.

2. Software Design

2.1 Game model

The game consists of two components, a dedicated server program that supports networked play between multiple clients, and a client program that connects to a server specified by the user.

Both the server and client run concurrently, and employ the concept of an update loop to handle real-time updates to the game. One iteration of the loop takes 40 milliseconds, meaning that the game updates at a constant pace, and does not run at a different rate depending on the speed of the processor. This is done by pausing the program after each iteration of the loop for the amount of time needed to reach a multiple of 40ms, i.e. if processing the current loop takes 16ms, then the program will wait for another 24ms before the next loop begins.

Each execution of the loop is regarded as a ‘frame’, and the game processes 25 frames per second, which simulates real-time gameplay in the GUI of the client. The server and client run these loops simultaneously, which allows for synchronised real-time communication between the two.

The client:

The client acts as an interface between the user and the server, where the majority of the game’s processing is handled. This minimises redundant processing, as the server only executes a process once, instead of running the same process on every client. The client still handles some processes, such as connecting to the server, and converting data from the server into output on the screen.

The client contains an enumeration called currentState, that changes depending on data from the server, and the client updates variables and displays different GUI screens depending on the state. There are four possible states for the client: ENTER_IP, IN_LOBBY, IN_GAME and SCOREBOARD. The IN_LOBBY, IN_GAME and SCOREBOARD states are reached when a packet from the server contains one of these state enumerations. ENTER_IP is a special state, in that the client program can switch to it without any feedback from the server.

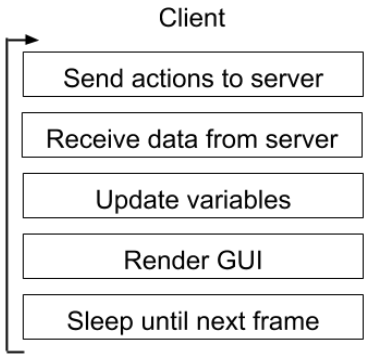


Figure 1. The update loop in RunClient

As the client loops, listeners in the client GUI record different inputs from the user, including key presses, Swing actions, and the mouse position, which are stored in *currentLoopInput*, a queue of player actions. At the start of each loop, the queue is sent to the server. Then the client reads the most recently received packet from the server, updates itself accordingly based on the data in the packet, and renders the updated GUI on the screen.

The server:

The server uses several classes that provide functionality for networking, player physics, map generation, and AI. Like the client, it also uses game states, to determine which methods to run. The possible server states are IN_LOBBY, IN_GAME and SCOREBOARD. During runtime, the server sends constant data to every connected client, including its own state. As the client state changes depending on the state in every packet it receives, the two are always synchronised.

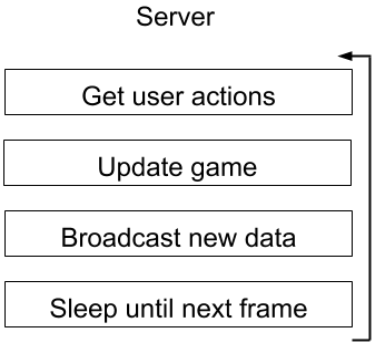


Figure 2. The update loop in RunServer

Initially, the server fetches all actions received from every client during the last frame. It iterates through the queue and, depending on its state, updates the game accordingly.

It then creates a packet from the new data, and sends the packet to every connected client.

updateGame():

This method is called continuously in the IN_GAME state, and updates all variables related to gameplay. It contains the following series of operations:

1. Add moves from AI players to queue
2. Iterate through queue and execute actions (if ‘shoot’ emit bullet, if ‘jump’ add velocity, etc.)
3. Apply gravity and deceleration
4. Check player/wall collisions
5. Update player positions
6. Check bullet/player collisions
7. Check bullet/wall collisions
8. Update bullet positions
9. Check bot/player collisions
10. Check for ‘dead’ players
11. Check if round has reached end state

This method draws information from and makes changes to playerMap, an ArrayList of Player objects (shown below) that includes positional information for every player on the screen, and bulletMap, an ArrayList of all Bullet objects currently in play.

2.2 Data structures

tileMap : Tile[][]
The map is represented by a 2D array of the Tile object; an enumeration with values for every type of tile: EMPTY, WALL, FLOOR, and SPAWN.

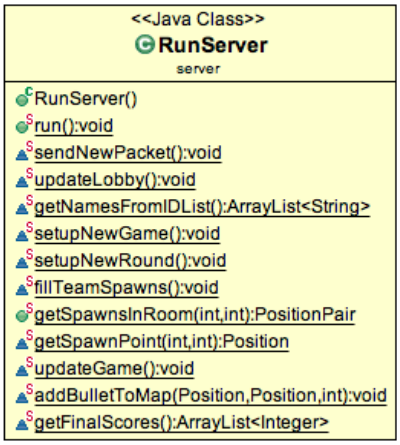
blocked : boolean[][]
Used in collision detection, blocked contains whether a player or bullet can move onto a coordinate position.

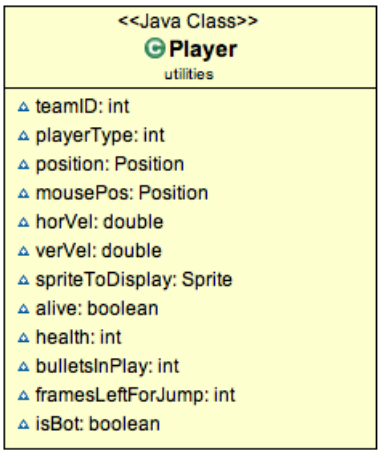
playerMap : ArrayList<Player>
All player information, including position, velocity, and health, is stored in an ArrayList of type Player, where stats for player *n* are found in playerMap.get(n - 1).

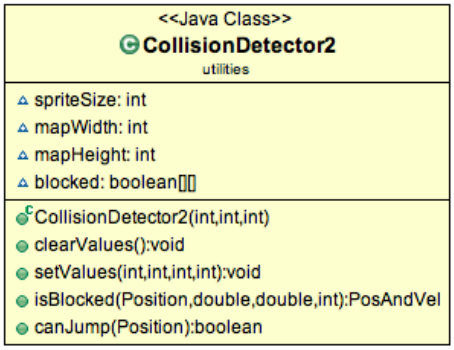
bulletMap : ArrayList<Bullet>
When a bullet is in play, it is stored in an ArrayList of type Bullet, which contains a bullet’s position. A bullet is added to the map when a player shoots in a direction, and removed when it collides with a wall or a player.

2.3 Main classes

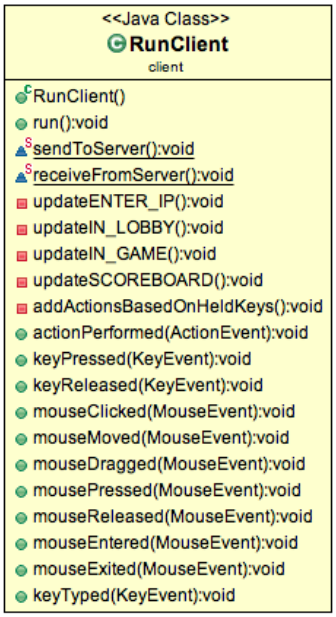
Server side:

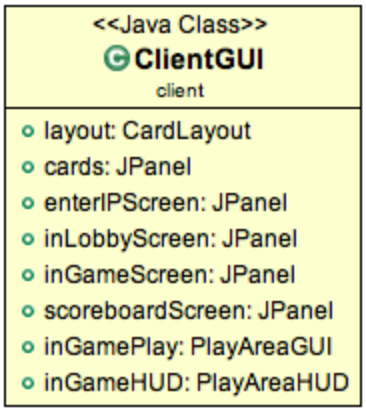






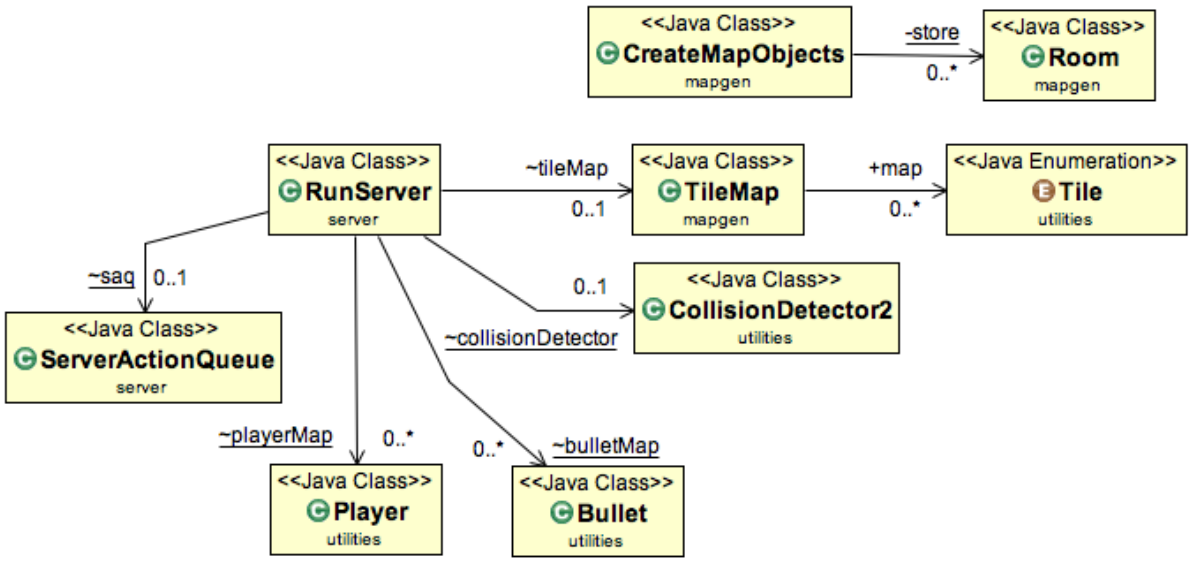
Client side:



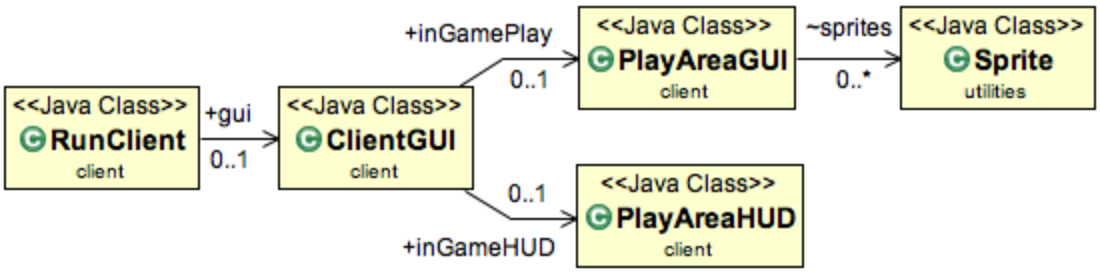


2.4 High-level class diagram

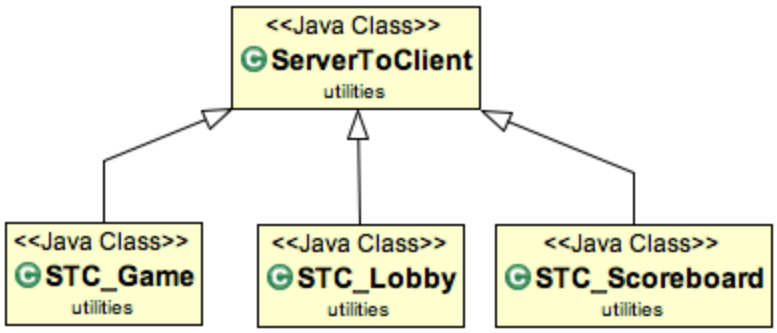
Server side:



Client side:



Objects sent over network:



3. Game Design, GUIs and HCI

3.1 User Interface

3.1.1 General Design

The game's user interface consists of four main swing frames – a start frame, a lobby screen, a screen for the “playing mode” states of the game and a scoreboard screen.

3.1.2 The Start Screen

The start frame contains two editable areas allowing the user to enter the server IP and the port number which are required in order that the user is able to reach the lobby screen of the system. When the user enters valid details in the areas provided and when they click “Join Server” they will be redirected to the lobby screen of the software.

3.1.3 The Lobby Window

The lobby window displays the Server IP below the “LOBBY” label on top . The lobby window also contains a list of players who have already joined the server. If the user is the first one to join the server they become the “Party Leader” of the game and are able to start a new game given that there are three more players in the lobby so that two teams of two players can be formed. The “Party Leader” of the game can also add Bots/AI players/ to the list of players in the lobby via two buttons located in the lower right corner of the lobby window . The maximum number of players and/or Bots which can be added to the list of players in the lobby is 8 (equivalent to 4 teams of 2 players). A user who is not the “Party Leader” does not see a “Start Game” button on their lobby screen. The buttons for adding and removing AI players to the lobby are also not visible for them. All users (including the “Party Leader”) are able to leave the lobby screen at any time via the “Leave” button. When the players' list in the lobby has at least four players/Bots and a game's “Party Leader” clicks the “Start Game” button that is usually displayed in the center of the last menu line on the lobby screen all players are being redirected to the game's playing mode.

3.1.4 The Playing Mode GUI

The game's “playing mode” window consists of a HUD located at the top of the window. The HUD displays the team's current “Health” state in the form of a red health bar. The teams' current scores are also displayed in the center of the same menu. The current round of the game is located on the right of the HUD. The players' actions within the playing area and the game's main graphics are displayed below the menu mentioned above.

3.1.5 The Scoreboard Screen

At the end of the game the scoreboard screen displays which team won and which team lost. Then the user is redirected to the lobby screen.

3.2 Game Design and HCI

3.2.1 General HCI Techniques

- Interactive GUI which allows up to 4 teams made of 2 players/AI components to be accommodated in the game’s HCI Framework
- User-Centered Design - the user is able to navigate through the system via clear and responsive menu framework which handles mouse and keyboard actions appropriately and in relation to the user’s character role in the game (the shooter can shoot via the mouse and the keyboard while the light guy can light their teammate’s way in a dark environment of the game via mouse movements)
- The players interact with the software through the mouse and the keyboard - the players can use the keyboard to navigate their way in the maze of a single randomly generated game map. The players (specifically light guys) can use the mouse to change the mode of the environment from light to dark. They can perform mouse movements over the game map in order to set a particular area of the map light or dark with the purpose of helping their teammate (the team’s shooter) to find their way ahead. Exactly this particular design feature and its HCI extensions make the main ideas for our game work in the way we planned them.
- Handling errors is accomplished by displaying appropriate error/warning messages (e.g. when the number of players in the game lobby is uneven the ‘Party Leader’ is not able to start a new game)
- Informing the user of the current, future and previous states of the system is accomplished by displaying appropriate windows on the screen or simply by

displaying nothing when feedback is self-explanatory (e.g. when the “Party Leader” starts a new game all players in the lobby are redirected to the “In Game” mode of the game and the lobby screen is hidden from view)

- The system’s design constraints don’t allow two games running at the same time

3.2.2 Design Consistency

The different colors in the Game Menus are used in such a way so that it is clear, readable and understandable for the users what the current stage of the system's progress is.

The interesting background for the menu of the playing mode screen makes the game's user interface not only likable but also allows the other components of this particular frame (the main game environment and also the informative components of the menu) to stand out as well. This is due to the careful implementation of layouts combined with proper usage of background and foreground colors.

The game’s main graphics make good use of different textures, colour combinations and movement simulation in order to achieve a very pleasant experience in the eyes of the users. The implementation of a complex random map generation algorithm makes the game’s design even more interesting and appealing for the users. In addition, the graphics of this game software are a good basis for thought provocation when the teams are playing the game.

3.2.3 Extending the Game's HCI Framework

Features like sound mode and colour blind mode could be added very easily to the game as usability techniques such as code refactoring and user-centered usability tests have been implemented throughout the development of the software. Furthermore, instructions for the game can be loaded from a file in a separate text area which can be very easily added to the game’s start frame design in order to make the game design even more fluent and even more easily adaptable for the users. Probably another refinement of the HCI for the game could be adding “keystroke detection” for all “Yes”, ”No”, “OK” type of buttons of our game’s user interface design framework. This would mean detection of “Enter”, “Esc” and arrow commands from the keyboard in addition to the mouse clicks over buttons which are already implemented in our GUI frameworks (e.g. in the same way in which we have made one of our prototype frames - the start

frame - have a button to detect both mouse clicks and “Enter” commands from the keyboard)

3.2.4 HCI additions implemented but not integrated

- Interactive main frames with interactive error and notification windows (e.g. when an error message is displayed the previous window becomes hidden and only the error message is displayed; when the user clicks the “OK” button over the frame displaying the error message the error message frame disappears and the state in which the GUI was in before the error occurred is displayed again)

- Non – standard choice of fonts for the GUI components (e.g. “URW Chancery L” , “Freestyle Script”)

3.3 Game Design

Although our aim was to show off our skills as programmers throughout the project, we were also grounded in an attempt to create an entertaining and balanced game. We made some decisions throughout the project that meant that parts of our code were dropped, in favour of other approaches in an attempt to keep the game fun and fair. Some of these are detailed below

3.3.1 Map Generation

Map Generation was the focus for the exploration portion of LightGunner, randomised maps would make every round different, and exciting.
In an early prototype of the code we had some rooms with closed off exits. These rooms would be placed similar to how they are now, lining up the exits so that the rooms were pathable between each other. The maps were also planned to be placed to make sure that players could always path between them, with no room tiles being unpathable by players. This feature was dropped as the tendency for the map placement to create dead-ends, which promoted slower paced game play, which we didn’t feel fit the aim of our game.

3.3.2 Power-ups and Weapon Upgrades

This feature was cut due to a lack of time, however, this feature was very debatable in the progress of thinking about how the game played out. Lightgunner is a lot about exploration, and rewarding players who explore through things like power-ups was a great way to enforce that dynamic. However, if players are making decisions based on their inventories, that slows down the pace of the game. If a player sees a stronger enemy, they may decide to run away until they can match the strength of an opponent. This leads to a possibility of longer, more drawn out rounds, as opposed to a more intense and exciting reaction based brawls.

3.3.3 Map Size & Spawning

In order to promote exploration, we needed the maps to be large enough for players not to always run into each other straight away. However, as the spawning was randomised, this could lead to players starting in adjacent rooms. We decided to keep this feature, as the occasional round would be a terrifying instant battle before the players had really got their bearings.

We toyed around with maps that were a lot smaller than the current system. The smaller the map, the less game time there was, and the less possible spawn points. It seems that the more rooms we had, the less predictability the game had, it essentially higher entropy, it was more random. However, it got to a stage with bigger rooms that the game field was too big, and walking around the area, it could take far too long to find other players, which started to turn the game a little boring.

3.3.4 Lighting Engine

The showcasing point of Lightgunner is how it's played with strict vision limitations. Not only does this promote exploration, but also tension. When played, we found it was often quite amusingly scary to be suddenly jumped by players of the opposing team. We feel like this was our greatest game design success. How the atmosphere of the game promoted this 'jumpy tension' was something extremely entertaining that we'd formed.

3.3.5 Co-operation/Competition

Lightgunner is all about co-operation, two characters with very different skills, one useless without the other, while still remaining balanced. This is a great success of the project, bringing everything together, taking a player through a myriad of emotions, while keeping them utterly reliant on another actor in the game. The game also remains to be a test of skill between the two teams. New players to the game will quickly find that poor teamwork will lead to embarrassing losses, and experienced players will be rewarded for their sharp-shooting securing victories.

4. Technical aspects

4.1 Networking

A combination of UDP and multicast sockets are used to send data between the server and the client. UDP is used to establish the initial connection packages, and let the server know what IP address the client is running on. Once a connection has been established, the client begins to receive packets from the server on it's multicast socket, simultaneously along with all of the other clients.

Information is sent and received in real time. During game play there is a constant stream of packets between the server and the clients. Due to the inconsistencies of some networks, packets can be lost in transmission. Heavy packet loss can result in a lagging game on the client end.

ServerActionQueue runs as a thread on the server, and is constantly receiving ClientToServer packets from the clients in the game. Before data can be sent, and after it is received, packets must be serialised and unserialised respectively. Serialisation involves converting the java objects into an array of byte, which can be transmitted with sockets over a network. Buffers are used in various networked components in the system to ensure that differences in the rate of data transmission and data processing do not lead to errors in functionality elsewhere in the game.

PlayerIDs are used on the server as a unique identifier for each client. When a client connects to the server they can specify the address of the server and the port they wish to send on. Allow the clients to specify the port they send data on enables multiple client programs to be run from the same computer. When a client submits the connection information, the server receives a join request from the client. If the client is not already present in the server, the client will be added to the list of PlayerIDs in the lobby, along with their IP address.

Depending of the state of the system, the server will broadcast different types of ServerToClient packets. For example, packets broadcast during the lobby state contain the current list of clients in the lobby, packets sent during the game state will contain lots of information about the position and velocities of players and projectiles.

4.2 Physics

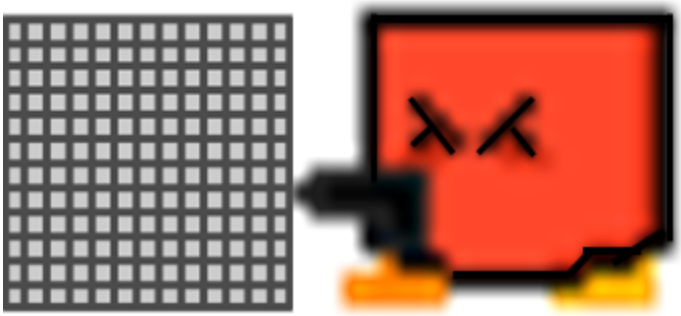
4.2.1 Player movement

Each time the server is updated, it will check to see which, if any, of the keys used for moving were being pressed. If the right or left key is being pressed (in our game A and D), the server would add a set horizontal velocity of 8 or - 8 in correspondence with the key being pressed to the total velocity of the player's sprite unless the sprite has reached its upper limit. So if, for instance- the sprite is traveling with a velocity of 20 (i.e. 20 tiles per frame) and the right key is pressed, the next time the server updates itself, the sprite will jump forward 28 tiles to the right. When the player removes his or her finger from the key, the sprite will very quickly decelerate to a hault. Jumping in the game is slightly more complicated. When the up (W) key is pressed, the server will check if the player is able to jump, that is, if he or she is standing on a platform or the floor or not. If the player is, an upwards velocity will be applied to the sprite which is then set to slowly decrease until it reaches zero. At this point, the player will begin to fall. Falling in the game is generated similarly, with a starting velocity applied and programmed to increase until reaching a maximum.

By incorporating aspects of acceleration and deceleration into the game as opposed to just having the players travel with constant speeds, a layer of realism is added to the mechanics that makes the game not only funner to play, but also easier for players to use and adapt to.

4.2.2 Collision detection

As a player moves, a bounding box is cast in front of the sprite in whichever direction it is going. This is like a virtual net, which will go through each tile between and the sprite and its next jumping location (i.e. the point it will move to the next time the server is run) and check which ones are blocked.



Obviously, it doesn’t really matter which tiles are blocked if they aren't the closest one. The key is to find the nearest blocked tile, or one of the nearest column of blocked tiles. In the diagram above, the program will start by checking the tile on the upper left. It will then check the tile below it and continue downwards until reaching the bottom of the box. It will then move on to the adjacent column and repeat. Whenever the program finds a blocked tile, it will remember its value as the last blocked tile (and discard the one previously holding this position). When the program finishes traversing the box, the tile currently marked as the last blocked tile is the one we care about. The sprite will then be moved forward to the column (or row, if the user is moving vertically) adjacent to the column featuring the nearest blocked tile. Thus, the sprite has been brought as close as possible to the blocked wall or platform, creating the illusion that it has been physically obstructed.

Obviously, this can be applied to the sprite when it is traveling in any direction. Ultimately, in whichever direction the sprite is traveling, the bounding box must always start by checking the row or column farthest away from it and then makes its way towards the sprite. The size of the bounding box depends directly on the speed the sprite is traveling. The faster the sprite, the longer the bounding box.

4.2.3 Bullet firing

When the user left clicks on the mouse, the server finds the position of the user’s sprite and cursor and calculates the vertical and horizontal distances between them. The server then uses these distances (as well as the direction of the cursor in relation to the sprite) to find the firing angle. The cos and sin values of this angle are then taken to give

the vertical and horizontal velocity of the bullet. The angle is converted to degrees and either 90 or 270 degrees are then added depending on the general direction the bullet is being fired. This is the exact bearing the bullet will be fired at. A new bullet is then created, with the value of it's bearing, it's horizontal and vertical velocities, the team member's Team ID, and its initial position stored. The bullet is added to an arrayList of bullets that are on the map, which the server will loop through each time it is updated. When it reaches this bullet, its coordinates will be updated by adding the vertical and horizontal velocities multiplied by the scaled bullet speed to the current positions. Thus, the simulation is created of bullet traveling in a straight line at a constant speed. With each frame, the server will also check if the bullet has intersected any blocked tiles or users not on the team that fired the bullet. If it has, the bullet will be removed from the arrayList, causing it to disappear from the map. If it has intersected a user, the user will lose a health point.

4.3 Map Generation

Due to the nature and focus on exploration of light gunner, the same map every game would quickly make the game stale and uninteresting. To counter this we created a system that would randomly generate maps that the game could be played through. These maps were made up of a 4x3 grid of rooms. A room was made up of a 20x20 grid of individual tile elements. At the start of every round, predefined rooms are randomly selected and ordered to create the game map, to create different game maps every time the game was launched.

Starting at the basics, the game is made up of an array 2D of tiles. Each tile is 16x16 pixels in area, and can either be an empty space, that the player can move through, or a filled in wall area that the player collides with.

To make sure the game is actually fun and playable, rather than generating maps based off something like randomly generated noise, we wrote predefined rooms that would be placed in a random order. The predefined rooms all have exits in all four directions, so the placement of them will not produce any dead-ends or unpathable areas. After the rooms have been played the map is then bordered so that players cannot accidentally fall out of the play area.

Spawn points are handled server side, at the start of a game, the server randomly selects separate rooms that each team can spawn in, based on the spawn points that are stored in the map data.

HCI was even implemented in Map Generation. In attempt to make the game more aesthetically pleasing, if there was no wall tile above a wall tile, the wall tile will be

turned into a grass tile. These tiles function essentially the same, but the way they are displayed on the map shows attention to HCI components of the game.
Finally, because the rooms of the game needed to be predefined, they needed to be stored somewhere. Map data is stored in an xml file in the game code. This is a really interesting way of doing it as it's easy to write programs that can write xml files themselves. So, we did that, and created a map editor that us, the developers, and advanced users can use to create more versions of the rooms, therefore their own versions of the game.

4.4 AI Components

The AI players in Light Gunner work fairly differently to the standard player controlled units. An AI bot team consists of only one player, that is an amalgamation of the two units, that are the light guy & the shooter. To balance this the AI is a melee attacker, upon colliding with an enemy, they'll kill them, and take them out of the game.
The priority queue of the AI player is a list of targets. At the start of a round, the bot creates a copy of the player list, and then shuffles it to randomise the order of it's targets. Everytime the bot makes a decision, it starts with that player list, gets the first alive player, and they are set as the bots target.
In the game code, there is a working implementation of Depth First Search based movement for the bot players. If a player came into the proximity of a bot, it would begin pathfinding towards that player, using depth first search, chasing them until one of them died, or the player somehow got away from the proximity of the bot. Because the bot would only be searching a relatively small area, this made Depth First Search a good pathfinding algorithm to use.
However, it was later decided that the proximity detection of the bot didn't really fit the fast paced nature of the game, and that doing any more pathfinding, over such a large area, with multiple bots, would cause further problems in the game. For this reason, the bots were changed to act on movement based on the player position, walking through walls, fitting their ghostly texture, to search & destroy the enemy teams.
The AI's movements are all calculated server side, in a separate thread that returns the bot actions, as a list, to the main server thread for it to process.

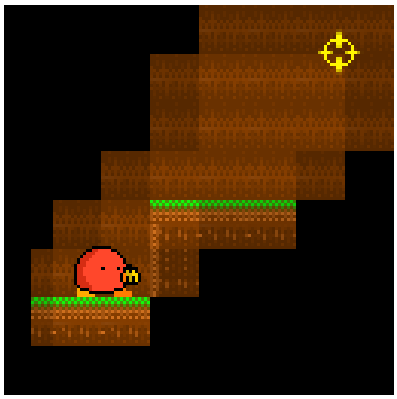
4.5 Lighting

As the game is played in darkness lighting areas around them is the only way for players to navigate the map or spot enemies. The Shooter has a small radius of light around them so they can see what is in their immediate vicinity, such as if they're on a platform or by a wall. The Light Guy has this radius as well as a cone of light which illuminates a lot more of the Light Guy's surroundings that is controlled by the player's mouse. Players cannot see their enemies lights.

Bullets travelling across the map also produce a small radius of light around them. Bullet lights can be seen by all players regardless of who fired them.

Lighting like this is possible thanks to the tile map system, that is that maps are stored as a 2D array of Tiles. The implementation of lighting meant minimal computational work, as the game would only render the images of tiles that the player could see, rather than rendering the whole map and masking it. The way this works is as follows:

- Iterate through all the Tiles making up the map
- For each Tile check its distance from the Light Guy
- For each Tile check its angle from the Light Guy
- Check the angle of the Light Guy's mouse to the Light Guy
- If the Tile is close enough (small radius) to either player it will be rendered
- If the Tile is close enough (large radius) to the **Light Guy** and its angle is close enough to the angle of the Light Guy's mouse, it will be rendered
- Else the Tile will not be rendered, and instead be filled black



Lighting by just rendering the Tiles that are within player's lights gave the game the sort of blocky style you see here.

5. Software Engineering & Risk Management

5.1 Software Engineering

In order to create our project on time and with minimal complications, it was obvious that we needed to adopt some sort of software engineering technique to work around. The one that we chose is called an Incremental Model. The appeal of the incremental model for us was that provided a good balance between being able to work on the project separately and independently to each other and still frequently making sure that all the separate components functioned together as they were supposed to.

5.1.1 Our initial plan

Our strategy was as follows: we would split into sub-teams of two, each of which would work on a different component of the game. One of these would be the networking aspect of it. This was to be the basis of the process, and when each of us had finished our part and gotten it working independently to the others, we would combine it together with the networking component and, after vigorous testing to make sure that everything works as intended, move on to another task. Gradually, the full game would be assembled one small piece at a time.

5.1.2 The resulting process

Unfortunately, we quickly discovered that this plan was impractical. The reason for this was that the networking process proved to be far more complex and challenging than any other aspect of the game and would only be ready very late in the process. As a result, we chose to keep the networking side completely separate for most of the process and instead apply an incremental model to the rest of the components.

About two weeks before the deadline, this decision started to show its problems. We had an almost fully functioning networking system with no game attached to it, and an almost fully functioning game system that only one person could play- which meant they could pretty much move their figure around the screen while firing bullets until they got bored and quit (we didn't have the AI implemented yet). As a result of this, the next two weeks were hectic, as we frantically tried to transform our complex and extensive collection of classes into something that fit into the networking system set up by the two team members working separately to the rest of us. Ultimately, we did manage to do it

in time, though in hindsight we should have adopted another software engineering model to prevent us from running into this problem from the outset.

5.2 Risk Management

Risk	Strategy	Strategy Type
SVN malfunction	All team members saved their work both on SVN and in their school accounts and home computers. This meant that we would still be able to work on the project and email each-other our updates	Impact reduction
Loss of data on backup server	All team members backed up their work on at least separate locations.	Impact reduction
Networking error during demonstration	Filming ourselves playing the completed game prior to our presentation. As a result, there would still be something to show to the markers.	Risk avoidance
Absence of personal due to illness, other commitments, etc.	Team members constantly filling in those that were working on something similar or with a large degree of overlap on what they were doing. As a result, there was always someone who was able to step in and complete an essential part of the project on time.	Impact reduction
Run out of time before game is completed.	We decided on which elements from our original specification were integral to the game, which were non- essential but ones we wanted, and which ones could be left out. During the first half of the project we worked only on things from the first two categories, and in the latter half switched to only the first.	Probability reduction
Not allowed to use an external library that we have utilised.	Only use external libraries when absolutely necessary.	Probability reduction

6. Evaluation

We were extremely optimistic while writing the specification. Comparing the final project results to what was achieved in the specification we can make a few observations.

Functionally, we achieved a fair amount that was wrote in the specification. We had a well made map system, that is exactly what is described in the specification. In terms of units, both of the player units were successfully implemented, as well as the AI component. They are able to move, shoot and were well implemented. The AI component of the game was also changed, this was also due to game design reasons. The lighting system was also well implemented, however, there are a few changes to it, based on the specification. Light is displayed differently to what was originally expected, this change was made due to game design, allowing players to see through walls produced better game play. Sadly, weapons, traps, teleporters & powerups were all dropped from development, due to a lack of time. Although these were implemented in some early versions of the game & map generation, full integrating them with the networking was placed at a lower priority during the end of the project. Networking was working in the final version of the game.

To improve on the project, it seemed that if a networked game and with a working game loop system was placed with a higher priority, features could have been directly added to that code, rather than working on, what were essentially prototypes. Although some features were added in earlier versions of the code, rewriting them in terms of the networked game code, was quite a challenge. However, we showed our ability to adapt by scraping features, and concentrating on perfecting the more important parts of the game.

We were challenged to not use any external libraries in our code. We rose to this challenge, and even opted not to use a simpler networking library that was allowed. Although this was a great success, this links back to the issue we had with the networked version of the game not being placed with high enough priority. Overall, we decided this was still the best decision, as implementing the Networking code ourselves seemed to be a more commendable notion.

In terms of non functional requirements, we believed we achieved these quite well. The performance of the game is good, our lighting engine was designed with this in mind, and a computationally efficient option was chosen with this in mind. We’re sure we have hit maintainability standards. Our code is well commented, and includes rigorous testing.

Finally, we believe we met the user characteristic specification. Our game is simple to understand, the lack of power-ups, weapons and traps actually helps the game by giving it a much more gentle learning curve, making to more fast paced, as opposed to making the game more about making strategic decisions based on the state of your inventory.

7. Teamwork

7.1 Organisation

As one of the larger teams, good organisation was essential to ensure effective and productive teamwork. We were able to stay organised through a number of methods.

Firstly, meeting early on and establishing regular weekly slots in which everybody was available to meet. Secondly, we created a Google Drive for the project which we used to write the Specification, Prototype Presentation, Final Presentation and the Final Report. Organising documentation like this meant we were optimising our efficiency as at the start of any new piece we would all assign ourselves to one or more sections to write. Everyone could see what everyone else was working on and there was no overlap, leading to focussed work. After a piece was written we would proof read it for mistakes and to guarantee a consistent style throughout.

Once the coding part of the project began it became obvious that coordinating 6 people work on one project would be more challenging than it had to be. Our solution to this was to divide into three 2 person sub teams, each being responsible for beginning a different section of the game. Full team meeting still took place to update everyone on progress and obstacles. This proved extremely effective as within a week we had three different aspects of the game working in some form or another, those being player movement, networking and map generation. The teams weren't so strict as to be a hindrance. For instance James from team Map Gen and Joe from team Networking worked together to and essentially pair programmed the first iteration of the tile map system in which map information was stored.

As the project progressed, the non networking teams divided again and individuals would work on different parts of the game, such as the AI, GUI and shooting mechanic. Constant communication was especially important at this stage to ensure that everyone was clear on what they were doing and were able

to outsource any problems they faced. Although we were working on separate components we found it useful to work side by side in the labs to keep comradery high.

This approach of parallel working meant the final part of the project required the most organisation. This was the integration stage. The player mechanics, map generation, AI and menu system had to be integrated within the networking structure. A problem we faced was that the way player movement and collision was originally implemented didn't translate easily to the networked version of the game and ended up taking an unanticipated number of man hours to resolve.

7.2 Teamwork Resources

7.2.1 SVN

Although using subversion was mandatory for this project the team definitely found that regular and frequent commitments was optimal when working as a team; it allowed all team members to be up to date with new developments in code. Occasional problems emerged, such as lack of comments in commits, but thanks to the open line of communication these were addressed quickly.

7.2.2 Facebook Chat

This was possibly the most useful line of communication during the project. It was popular because it was informal, immediate and easy to set up. It was a constant line of communication to the rest of the team if you needed anything, big or small. By the end of the project over 600 messages had been sent over the chat. One significant issue here was that one of our team didn't have a Facebook account and so risked isolation from the majority of team discussion. However, this problem was minimised by our committed team members who would email the other member with updates and even screenshots from the Facebook chat.

7.2.3 Trello

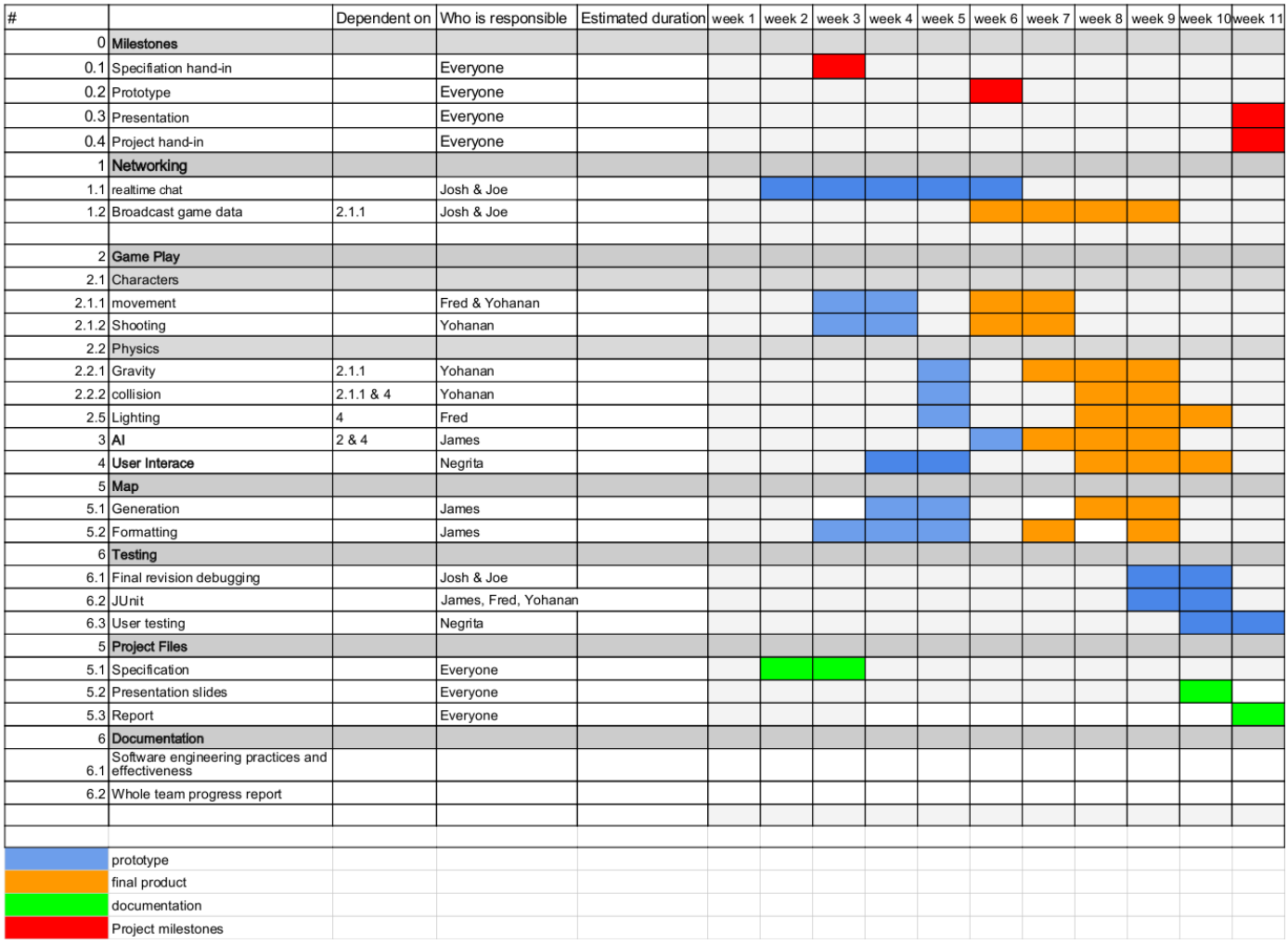
Trello had obvious advantages as a well made piece of teamwork software that everyone had access to, and it was utilised in the first few weeks. However, later

it was abandoned as there were easier ways of communicating tasks, either in person or by a simpler online messenger.

7.2.4 Google Drive

As discussed earlier Google Drive was a central tool used for documenting the project and preparing presentations and reports. The real time editing and ability to add notes everyone could lay claim to a their part of a task and no one was left with an older version of a document.

7.3 Revised Gantt Chart



8. Summary

Overall, we are extremely proud of the game we have made. We feel that it is challenging, fun to play, and original in its ideas. The necessity for the user to think and act quickly at points while in others carefully strategize is not something present in most 2D platform games, and as a result adds a special level of depth to the playing experience. We feel that the game is very easy for players to use and learn, with the movement, lighting and shooting mechanics working in such a way that is easy for users to control and familiarise themselves with. In addition, we feel that the game is easy to access, with clear, easy-to-use GUI's that are aesthetically simple but eye-catching.

The process by which we made the game however, could have been vastly improved. As a result of our underestimating the importance of having the game mechanics integrated into the game from the start, the game ended up not being as complex or sophisticated as we had envisioned and a lot of time and energy was wasted on creating features that didn't make it to the final product. Outside of this huge, gaping error, our risk analysis was very good and well thought out, with no other significant problems being encountered throughout the project.

In regards to software design, the structure and systems that we used in our program effectively facilitated real-time gameplay over a network. The decision to implement an update loop part way through the project made synchronising the server and client easier, but not doing this at the beginning meant that we had to rethink the way we implemented some parts of the game, such as collision detection and player movement. By putting a lot of thought into the class structure before starting, and refactoring parts of the code throughout development, our final code is well-structured and concise, and relatively easy to comprehend.

Were we to do this project again, we would spend more time considering other networking techniques that would allow us to spend more time developing other aspects of the game. We chose to implement the network using a combination of UDP and multicast sockets which unfortunately turned out to be difficult and time consuming. Next time we would consider making use of technologies such as Kryonet.

9. Individual Reflection

9.1 Yohanan Ben-Gad

Overview

This team project has, without a doubt, been the most fun and rewarding module I have taken thus far throughout the entire course. I loved creating and implementing the physics for the game (as well as several other features that didn't make it into the final version due to time constraints) and although the JUnit testing was exhausting, I can't honestly say that I didn't learn a lot from it. My team were great to work with, their enthusiasm for the project managing to make even the long weekends spent in the computer labs just about bearable. I hope to work with them again at some point in the future and hope that this is not the last computer game I will have a hand in making.

Working as a team

Having never done any sort of team-work on this scale before, particularly with people I didn't already know- I was unsure of what to expect going in. The fact that things worked out as well as they did was a pleasant development to say the least. The people I ended up with turned out to be a group of intelligent, hard working individuals, each of which was able to bring a different set of skills to the table. Looking back, I realise that I would never have been able to do a project like this on my own, not even with six times the time to do it in. There simply would not be the diversity of thought that is needed.

I particularly enjoyed enjoyed it whenever I collaborated on a specific piece of code with another team-member. Even when I was on my own, it was great to start a session by seeing all the additions that had been made to the code in my absence- the various ideas I hadn't thought of and ways of doing something different to my own. Then I had the challenge of trying to think of how I could push and improve it even further. Going forward, I feel far more confident about the prospect of taking part in another project like this one than I ever had prior to this.

9.2 Fred Evans

Overview

Working on the team project has been an overall enjoyable experience. The requirements of creating a cooperative game gave a lot of freedom. We came up with the concept, made the design and build the game based on that design. This project let me experiment with things that ordinarily I wouldn't get the chance to during a computer science degree, such as designing sprites and their animations. It gave me the opportunity to decide how I wanted to tackle tasks. Rather than asking me to write code that performs a specific function, team project meant that I could choose exactly what that function was. There are less restrictions. When I was implementing the lighting mechanic I learned that things will change dramatically from your original ideas, often for the better. There were stages where I thought it wouldn't be possible for the player to have a mouse directed torch beam, but playing around with different methods eventually lead to the final design.

This project definitely help me improve my coding, as well as how I learn to code. Although you could find tutorials online similar to what I was doing, they were only the building blocks with which start. I would learn online how to do something, such as simulating gravity, then I'd adapt that so it fit the functionality of the game. Doing this reinforced the lesson and helped me understand what was actually happening in the code.

Working as a team

The fact that we were given little guidance meant that good teamwork was imperative during this project, and looking back I would say that we worked well as a team. Although I didn't know any of my teammates prior to the project, it quickly became evident that they were all pleasant, competent and hard working people. Having heard some horror stories from other teams, I feel lucky to have a team which worked who worked together with general ease.

What I found most useful when working in the team was regular meetings and communication. This made sure we were all aware of what was happening in the project and it also meant progress was always being made. I didn't want to show up to a meeting if I hadn't made any headway, so it forced me to work. Good communication meant we were available to help each other at most any time.

Probably the biggest challenge for me when working in a team to create such a large piece of software was understanding other people's code. If it wasn't commented the it ran the risk of being difficult to decipher. Luckily teammates were always willing to sit down and explain their code if need be.

9.3 James Davis

Overview

Being a huge fan of many styles of video game, and a huge follower of the complex industry that is the video games market, I found this project exceptionally exciting and rewarding to be a part of. Truthfully, my enthusiasm for the industry was not what made this project what it was, being placed with an amazing, diverse and very committed group of people made this project exceptional.

We all started with different ideas of what we wanted to make, boiling down into a fantastic idea for a game. We incorporated our different ideas in an extremely balanced fashion, manifesting into a product that seemed to represent all of our values, without being overloaded and chaotic.

This project has given me a passion for this kind of work, and I'm sure this won't be my last experience in game development, and whether I attempt to do this on my own, with a group of friends, or even an entirely different piece of software altogether, the experiences I have gained from this module will be invaluable.

Working as a Team

An extremely daunting part of this task was working with people I'd never met before. We didn't know each others strengths, weaknesses, work patterns, or senses of humour. Our team was extremely rewarding to work with, we had regular meetings that were always well attended. Everyone seemed motivated to do the work we set each other each week, a week never went by when nothing was achieved, even though we were all working to other schedules and had different priorities and motivations outside the project.

It was great working with people on such a big project, seeing large amounts of made in a few days, and the satisfaction of what we could achieve as a team, when we all put the hours in was exceptional when viewed as a whole.

On a smaller scale, if there was a part of my code I was struggling with, whether it be relating to game design, a functional problem, or making sure I was returning object that could be used by our networking on a higher level, we worked well to make sure we always had time to help each other out on any difficulties that cropped up.

I really look forward to future, similar, tasks. I feel I've learnt valuable teamwork skills during the project, that I could only have been achieved though a large scale project.

9.4 Joshua Holmes

Overview

Game development is the area I aim to have a career later in life, so this project was a great opportunity to apply the skills I've learned in an area relevant to the skillset I need, while developing a creative product, and working within a great team of people to do so.
Networking was by far the hardest part of the project to overcome, especially with UDP and the risk of packet loss. We developed the network system manually, as I personally thought it would provide better experience than using an approved library like Kryonet, but it took longer than it should have and held up other aspects of the project. In the future I will aim to use more libraries, even if time has been invested into writing a solution a different way, to save time and unnecessary effort. It also allows more time that can be better spent on improving the quality of the final product.
The real-time aspect of our game meant that the code structure was designed around the networking, and as Joe and I wrote the code for the related classes, it meant that some other team members did not completely understand the entire framework of the code. This meant that integrating code was difficult and created bottleneck in work flow, and in future projects I would ensure that everyone has at least a basic understanding of the overall code structure. We also should have agreed on basic data structures before any coding was done, as some code had to be rewritten when integrating.
The last thing I would do differently is time management and scope. We were ambitious with what we could achieve in the time given, and had to cut a lot of features towards the end of development. This also resulted in some very long days in the lab in the last week, to meet the deadline. That said, our team performed exceptionally and I am very happy with the relationships we built, and the quality of the final product.

Working in a team

We worked well as a team and our diverse areas of expertise meant that we all gravitated towards different areas of development, and splitting up the workload was easy. During the initial meetings, everybody contributed good ideas, and our final concept was interesting and unique because of it. Everyone was enthusiastic about the task that we were presented with, and we got on well.
We initially split into subteams and worked on different areas concurrently, which meant that we developed a lot of code early on, and quickly had a working prototype. This took a lot of pressure off the later stages of development.
We held multiple meetings each week which we all attended, and extra meetings within our subgroups on areas that needed extra time. We all communicated through Facebook chat and Trello, which meant that everyone was reachable at any point.
Overall, I feel that we worked very well as a team, and our product thrived because of it.

9.5 Joseph Thomas

Overview

I greatly enjoyed the team project module this semester. The game we chose to build had some advanced and complex aspects of its design and functionality. Although complexities in the design and functionality of the game made certain parts very difficult to implement, when we finally did manage to complete it, it was more rewarding. One of the ways in which our game was functionally complex was the networking aspect. Early in the project we chose to network our game in real time as opposed to using a turn based method. This increased the functional complexity of the game as we needed to ensure the method of controlling the flow of the game was very robust and would not be extremely sensitive to an inconsistent reception of data. We also chose to create a 2D side scrolling environment for our game to take place in, rather than a top down design. This increased the complexity of the design of the gameplay, as we needed to implement a rudimentary physics system to simulate gravity and generally make the player movement feel natural. After we had presented our prototype midway through the semester we received feedback that though aspects of our game were impressive functionally, we needed to work on the visual appeal. In response to this, we refined the design of the user interface, as well as overhauling the look of game play.

Working as a team

I have found working as a team to be the most rewarding part of this project. Early in the project we were able to the effectively meet as a team, discuss how we wanted to approach different sections of the project together, and divide work between members of the team. The most challenging aspect of teamwork in this project wasn't sharing the work, but communicating the functionally of complicated parts of the system. As an example, Joshua and I formed a sub-team to develop the network of the game. We made the mistake of isolating the development of the network from other sections of the game system. This meant that when it came to integrating the sections that the other team members found it hard to connect the components.

9.6 Negrita Kalcheva

Overview

This project has given me a very different experience from all the work I've done so far in this subject area. The development of the GUI and HCI Frameworks on which I worked on for the most part of the project proved more challenging than I thought.

For the project we had to work in a team of 6 people for 11 weeks in which we had to build fully functional interactive real-time game software playable in teams over a network. The huge group of people in our team were not only very capable technically but were also very well organized and very open - minded people. Our team demonstrated a great amount of responsibility towards this project, keeping in mind that there were six people in our team and more people often means less organization. Our team proved to be entirely different and has nothing to do with this stereotype. This is due to the fact that we made organization plans very early in our project. However, there were several stages during the development of this project, at which I personally didn't pay any attention to anything else in the academic schedule except for this project.

Working in a team

Fortunately, our team has been very cooperative and we managed to arrange times for our team meetings in such a way that they didn't collide with other events in our weekly academic timetable. The team organization and team communication factors became one of our team's greatest strengths. Because of these we were able to integrate our code more easily and fluently. We were having regular weekly meetings. At the end of every team meeting we arranged the time and place for our next meeting. In this way we were always over prepared. We were often setting tasks for each team member at the end of our team meetings - e.g. creating textures, refining our GUIs and HCI techniques etc. As we were quite a lot of people in our team we frequently divided our team into subteams (made of two people in almost all cases). This helped us maintain stable progress in our software development. Our plan was that whenever a team member would not be present at a team meeting they were to be informed about the next meeting's time and place usually either via email, Facebook and/or Trello (the team software which we all used for sharing information on our project's progress). These intensive methods of team communication (both virtual and face-to-face interaction) allowed our team to join efforts, skillsets and experience so that we can develop a fully functional game software with our initial plans and ideas being realised in practice and being implemented in a machine. As soon as we saw some of our code working we

became even more excited to write more code. The diversity of technical practices, the usage of soft skills and coding techniques in our team is quite noticeable and can be observed well in our team's SVN Repository as well as in our team's Trello archives. This diversity was actually a huge plus since every time we met each team member had new ideas and new code to demonstrate and add to our software.

Conclusion

We were further motivated by the fact that we were working in a team and we wanted to work even harder and wiser. We started working intensively as a team very early in the project and we finished our project working even more intensively again as a team. Although this project has probably exhausted all of us I can't deny that what we managed to achieve is quite fascinating considering the short amount of time in which we completed our work.

10. Instruction Manual

Players
Lightgunner can be played with 2-4 teams, each consisting of 2 players. In a team one player is the shooter, the other is the light bearer.

Joining a server
To join a game, first one player must run the server file, and share his or her ip address with the other players who are playing. Those players must then run the LightGunner Client. In the client screen players must enter the IP into the IP field, and the port number (default: 8001) to connect.

Start of the Game
As the game begins, each team spawns at a random set of nearby spawning points in a randomly generated map. The entire map is invisible to all players save for a small circle around the shooter and the shooter's bullets, and a larger circle and rotatable beam coming from the light bearer. Only players from a team can see the light from their team's players. Each player begins with 10 health points.

Objective
To become the last team on the map after all the other teams have been eliminated.

Gameplay
Players can move to the left and to the right using the A and D keys. They can jump up onto higher platforms using the W key and they will fall off of platforms when they move past they're edges.

The shooter can fire by left clicking on the mouse in the direction they want to shoot.
If a bullet hits a player of an opposing team, the health of that player will decrease by one. If the health of either of the players in a team goes down to zero, that team is eliminated from the game.

In order to see where he or she is going, it is advisable for the shooter in each team to stay close to the light bearer.
This game presents a 'cat and mouse' scenario, with each team having to find the other teams and eliminate them while staying on guard from other teams firing on them in the process.