# HEIST.

Final Report
Version 1
20/03/2014

**B1**
Jaeren Coathup
Zoe Kyriakou
Dave McDonald
Jordan Moore

Prepared for
Team Project

# Table of Contents

# Introduction

## Purpose

Our game, named “Heist”, is a wave based game, where the players work in teams of two over the network, and have to survive by killing all of the computer controlled enemies in each level. When all players from both teams have died, the teams scores are shown, and the team with the highest score is the winner.

There are up to four-players over a network. Our multiplayer mode is both cooperative and competitive based. Players work together to survive as long as possible, and collect the most amount of money (which is randomly generated into the game to be picked up by players). The player gains score by both killing enemies and picking up the bags of money. When all of the players have died (on the same wave), the team with the highest overall score is crowned the winner of the game. If a player is killed in a wave, but other players are still alive, they will respawn at the start of the next wave.

Players move with the arrow/WASD keys and click the screen to shoot their weapon in that direction. If a bullet comes into contact with an enemy, that enemy receives damage that is subtracted from its health. If an enemy’s health reaches zero, the enemy is killed and removed from the screen.

There are a number of forms that the enemy comes in, starting with the initial (easy) security guards, (normal) police, and (hard) SWAT. These enemies both have different amounts of health, and you gain different amounts of score depending on their difficulty.

## Setting

The setting of the game is a bank heist. The player controlled characters are the criminals, and the computer controls the law. There are three maps: the bank vault, the bank, and the city streets.

**The Vault**
All of the players will start off in the vault of the bank. There are a couple of corridors surrounding the vault that the players can explore. This vault serves as a tutorial for the players, as this map is the easiest to play. The vault contains a small number of easy to defeat security guards. The players must defeat all of these security guards before being allowed to progress up the stairs, towards the main bank.

**The Bank**
The bank will have been put in lockdown due to the break-in, and the players will be unable to

escape. The players have to defeat  waves of security guards inside of the bank, before progressing outside. After a certain amount of waves, the bank doors will open and the players will walk to the entrance of the bank, and be transported to the city level.

**The City**
The city is the main level, where the players will reside for the rest of the game. The players have to survive a number of waves of increasing numbers of enemies. After completing the completion wave, the players will win the game and the team with the highest score will be declared the winner.

## Scope

### Heist Game

This is our main software product, which is the game that the end user uses to play the game. The benefits of this product are to provide an alternative story with cooperative and online aspects to the traditional wave based games that require you to defeat enemies.  This game should be fun to play, as it features both teamwork and cooperative aspects, but has a challenge aspect in that it can actually (with difficulty) be completed.

### Map Creator

We have also created a map creator, that can be used by the developers to create new maps. Once the map creator is loaded you can select any image on your computer and it creates a collision map for you, where you can select squares which are solid and those that are not.  You can also load a B&W image into the map creator and have it generate a collision map for you. Once you are happy with the collision map you can then save it to a text file, where it will be read in to be used for the map for the game.  We have currently created three maps, but in the future we can easily create more with our map creator.

# Software Design

The team initially began the design for the game by developing prototypes. This prototyping led us to be able to accurately design a class diagram for our game; the prototyping gave us experience with unfamiliar areas of development, to design them properly.

**Game Model**
There were specific design requirements that our game model required. These revolved around the following:

1. **Multiple maps & AI pathfinding**
   This meant that we needed to design our game in such a way that maps could be changed during game play, and so that the AI was dynamic enough to work on whatever environment it was placed inside.

   We solved this by creating a super type Map, which each map extended. Each map implemented a byte array collision map. The AI, which must have access to the map object at all times, can use the collision map to determine walls, rather than hard-coding walls into the algorithm.

2. **Network players being displayed in real time**
   This meant that we needed some way of sending details about our own game to all other games, and a way of updating our game based on details received from other games.

   We implemented this using a network manager. On each game update, packets for each part of the game state would be generated, and then the network manager would send out these packets to all peers. The network manager has an interface connecting it to the game object, so that it can notify the game object of network changes. Using an ID and the new details, it’s possible to update each game based on network data.

   We were able to use a lobby to create and distribute a list of peers to all peers. We also were able to use the lobby to designate which player would be performing the “host” only jobs, such as pathfinding.

   Because we wanted to send data in real time, we knew that we needed to keep the sent data minimal. To do this we designed a custom packet that would send data using the minimal number of required bytes.

3. **Wave based**
   This meant that we needed a way of generating waves of enemies, knowing when each wave was completed, and how many enemies were required per wave. To do this we decided that we would create a specific wave object. We decided that this object would have a list of enemies inside of it, and that we could monitor this list to decide when the wave had ended.

   We also decided that the wave object should have a method so that when the wave had completed, it would generate the required number of enemies (using an algorithm) for the next wave.

4. **Large maps**
   We decided that we wanted maps that were much larger than the area that the player is able to see. This meant that we needed to implement scrolling maps. This meant that we would need a way to calculate the position of the map and the player on the map, from an absolute position. Most of the time our player is in a fixed position, and the map moves instead, but an illusion is created of the player moving.

5. **Access to keyboard and mouse required in game**
   In order to respond to the user's commands, we would need a way of accessing the keyboard and mouse from inside of the game. We knew that the keyboard and mouse would also be required for the GUI, so knew that we would need to have these items centrally located and accessible.

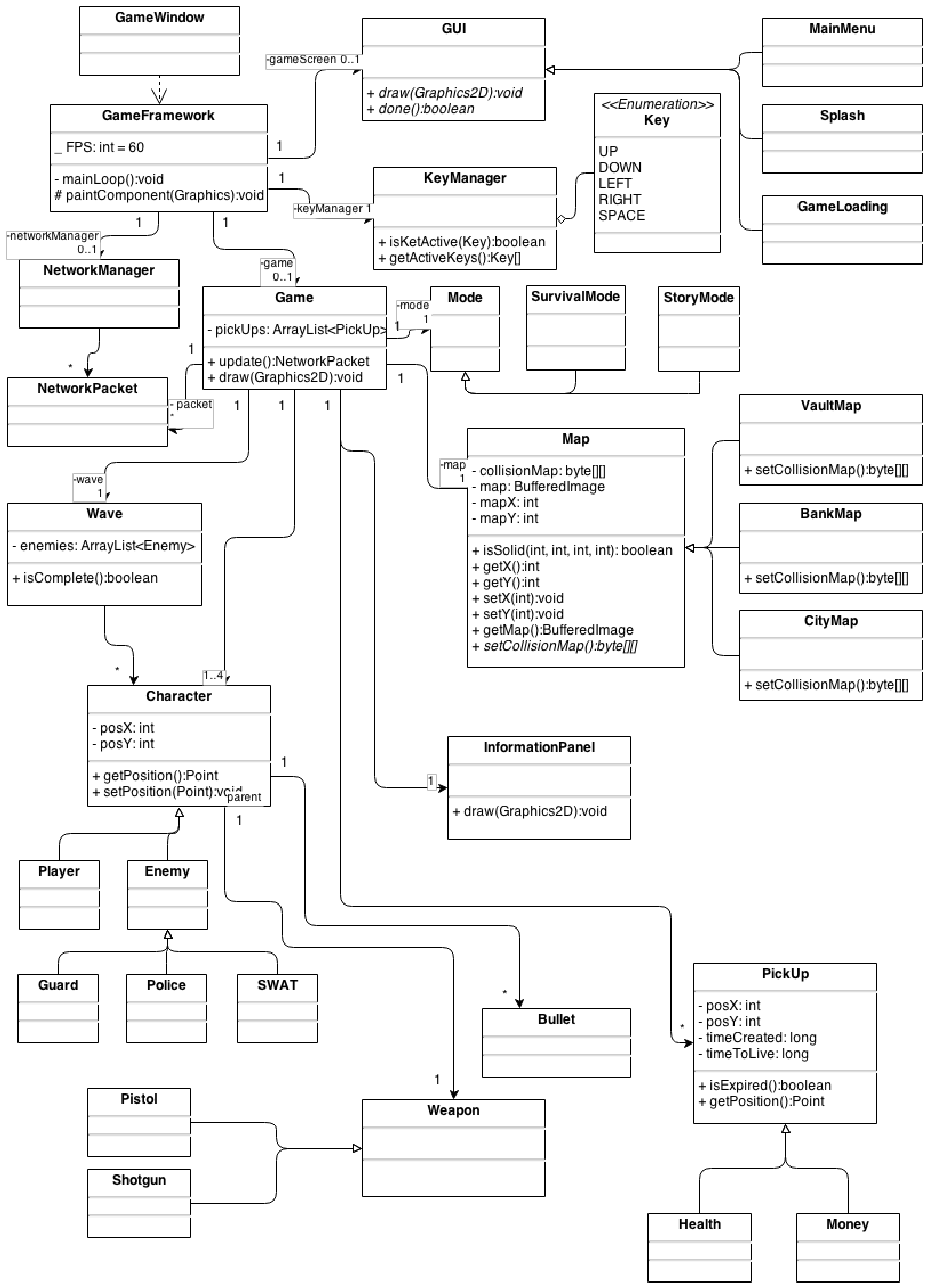
6. **Multiple enemies, and multiple types of enemies**
   Knowing that each map would require multiple enemies, and the ability to display multiple types of enemies, we knew that we would require an enemy superclass. We also decided that we would want 3 types of enemies, so these would subclass the supertype of enemy.

   This would then give us the ability to store the enemies inside of the wave object, and create/destroy undefined amounts of them very easily.

7. **A way to link components together**
   As mentioned in the "access to keyboard" section, we knew that we needed a way to link components together. Ideally we also wanted a very easy way to transfer data between areas, and switch states of the game.

   To do this, we decided to create a "GameFramework". This framework would serve the purpose of creating objects, linking created objects together via interfaces, maintaining the game loop, etc.

As a team, we sat down and created the following class-diagram, that we followed for the entire creation of the project. We agreed upon, and collaboratively worked upon, this diagram before beginning to work on the project itself. This diagram is the diagram created in the 2nd week of the project, and has changed very little. Since its creation, it has been extended, and differed very little from.


GameWindow
GUI
-gameScreen 0..1
+ draw(Graphics2D):void
+ done():boolean
MainMenu
Splash
GameLoading
GameFramework
_ FPS: int = 60
- mainLoop():void
# paintComponent(Graphics):void
1
1
-keyManager 1
KeyManager
+ isKetActive(Key):boolean
+ getActiveKeys():Key[]
<<Enumeration>>
Key
UP
DOWN
LEFT
RIGHT
SPACE
-networkManager 0..1
1
1
NetworkManager
-game 0..1
Game
- pickUps: ArrayList<PickUp>
+ update():NetworkPacket
+ draw(Graphics2D):void
-mode 1
1
Mode
SurvivalMode
StoryMode
*
NetworkPacket
- packet *
1
1
1
1
1
1
VaultMap
+ setCollisionMap():byte[][]
Map
-map 1
- collisionMap: byte[][]
- map: BufferedImage
- mapX: int
- mapY: int
+ isSolid(int, int, int, int): boolean
+ getX():int
+ getY():int
+ setX(int):void
+ setY(int):void
+ getMap():BufferedImage
+ setCollisionMap():byte[][]
BankMap
+ setCollisionMap():byte[][]
CityMap
+ setCollisionMap():byte[][]
-wave 1
Wave
- enemies: ArrayList<Enemy>
+ isComplete():boolean
*
1..4
Character
- posX: int
- posY: int
+ getPosition():Point
+ setPosition(Point):void
parent
1
1
InformationPanel
1
+ draw(Graphics2D):void
Player
Enemy
Guard
Police
SWAT
*
Bullet
*
PickUp
- posX: int
- posY: int
- timeCreated: long
- timeToLive: long
+ isExpired():boolean
+ getPosition():Point
1
Pistol
Shotgun
Weapon
Health
Money

**Main Classes**

The GameFramework has many important roles, including

- ensuring a steady 60FPS is maintained in the main loop,
- updating the GUI/Game via the main loop,
- repainting the GUI/Game via the main loop,
- instantiating the Resource Manager,
- instantiating the Network Manager,
- instantiating the Mouse Manager,
- sending network packets generated during game update,
- and linking components together via interfaces.

When a new game is created, the GameFramework generates a new instance of Game. On each loop of the main loop, the GameFramework will now update the Game via the update method, then paint it using the draw method.

The Game object has many important roles, including

- updating the player/map position
- updating enemy search path/position
- updating player/enemy bullet positions
- updating/maintaining/checking the collision map
- generating ingame pickups
- updating network based players/enemies
- generating network packets for the current game state
- providing a method to paint the current graphics of the game

We made a conscious effort, as a team, to ensure that the entire game functionality was not located inside of Game. Rather, we modulated the game, so that functionality was provided by multiple classes. We also made extensive use of subtype polymorphism to ensure that no redundant code was generated. Given this style of design, adding new enemies or weapons to the game, for example, is a very simple and quick job. It also meant that adding the Bank and City maps for the game was a very simple job.

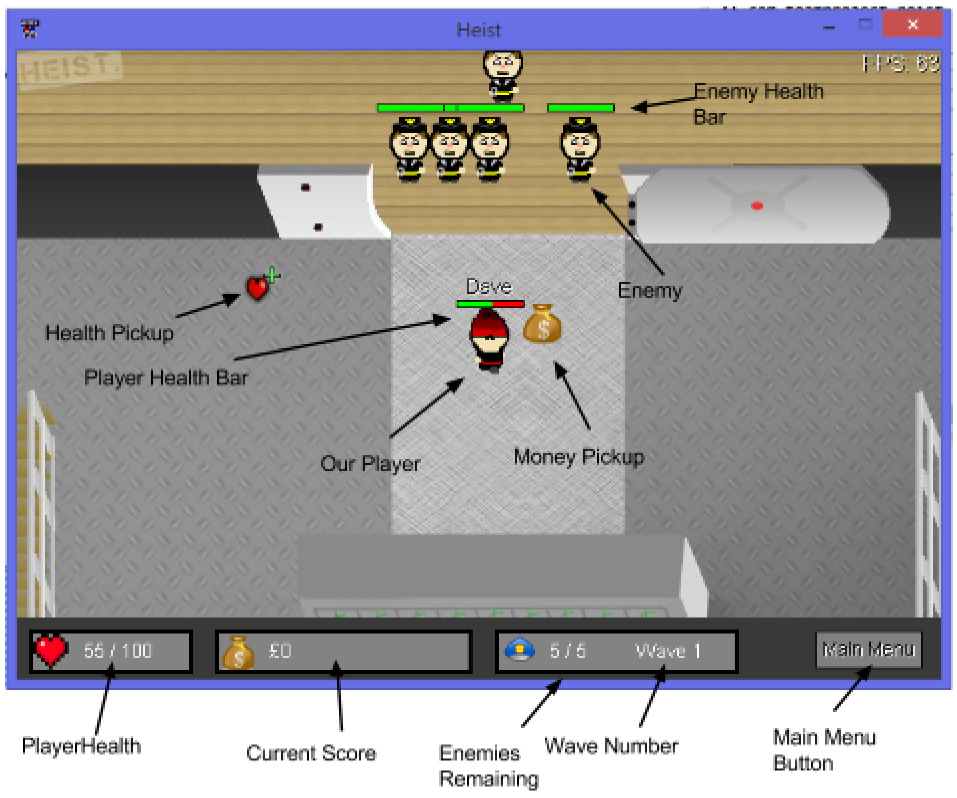
Designing the project early on, as above, led to an easy to work on project. Our project now features almost 20 packages, over 100 classes, and close to 13,000 lines of code, however, it is still very simple to develop for and debug, due to the early and well designed design.*

*19 packages, 109 classes, 12782 lines of code. Generated by State Of Flow Eclipse Metrics on March 23 at 15:19.

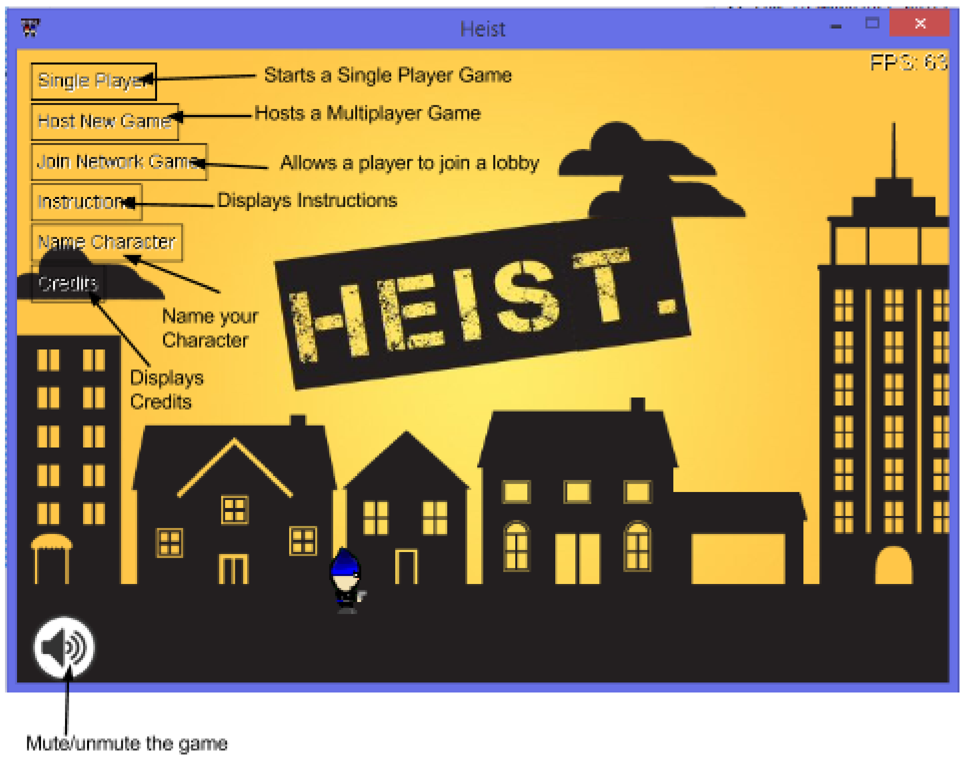
# The GUI, & HCI

## GUI

In game screen:



Main Menu:

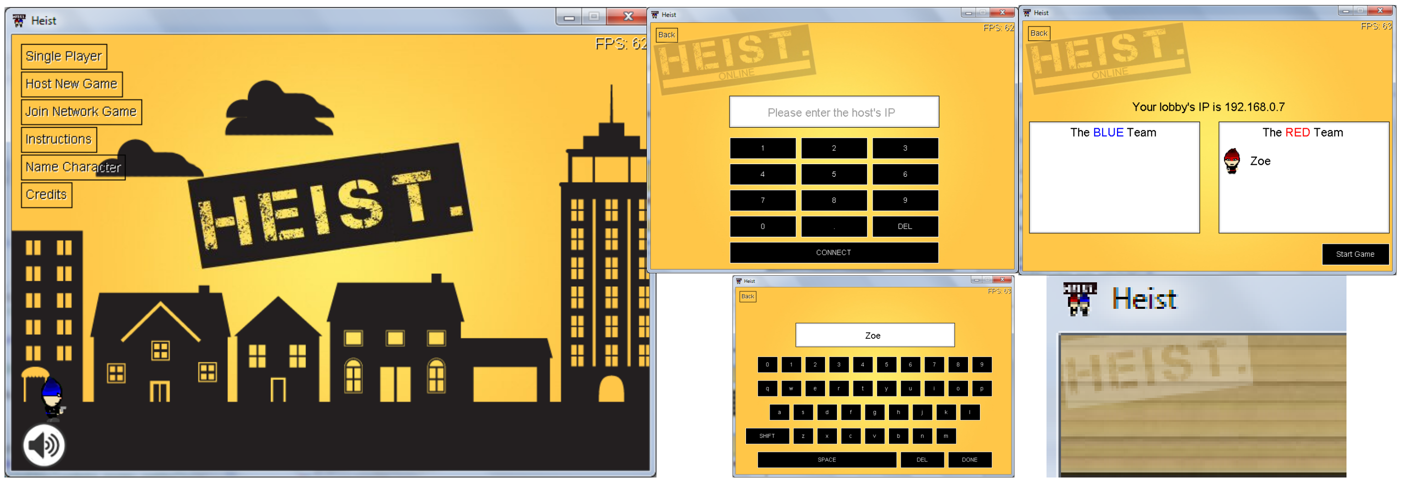


## HCI

When discussing what our game should look like, we knew we had to consider Human-Computer Interaction as a priority. We believe our game is both easy and intuitive to use. We ensured this was the case in the following ways:

### Consistency

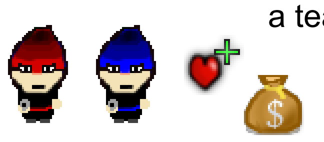
The GUI is consistent in many ways. First of all, throughout all our menus we have gone with a colour scheme of orange and black and used the same font. This gives our game a specific identity and image which is easily recognisable.



The logo appears on most screens, including within the actual gameplay. We also have a consistent use of buttons, with the main menu button appearing in the same place during gameplay and completion.

### Intuitive Play

We have used obvious, intuitive symbols and colours over plain text where possible. This has resulted in our game not just becoming much more user-friendly, but also being more pleasing to the eye. Teams are displayed as two different colours, red and blue, and all information linked to a team, such as score, is displayed in their respective team colours. Health and money pickups are also intuitive, with their images being self explanatory.



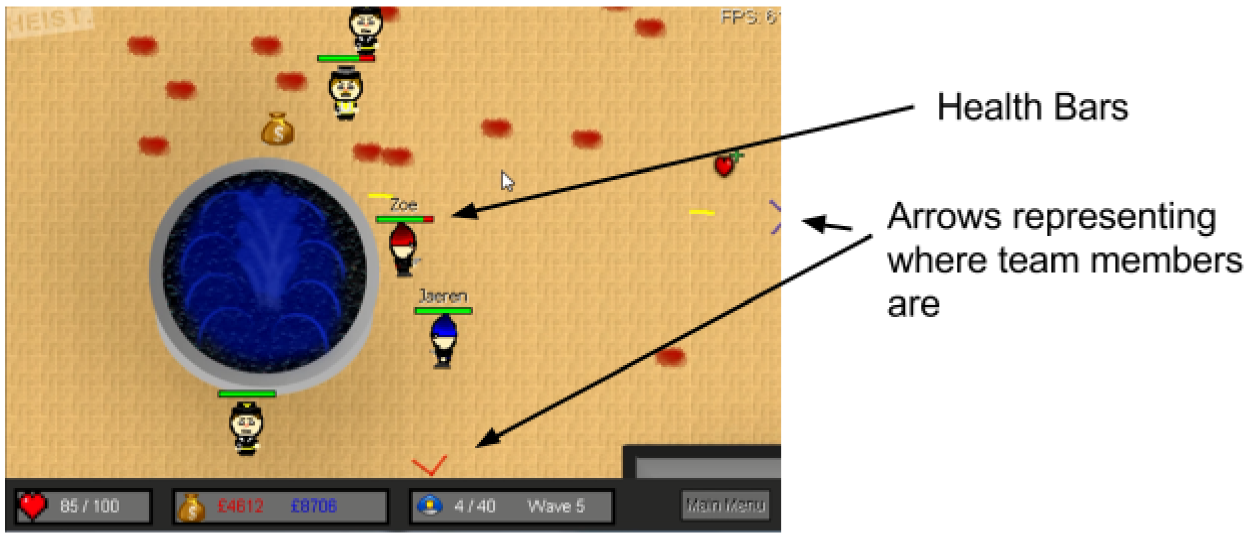
There are three different types of enemies within the game, each increasing in difficulty to remove. The images of these guards depict their difficulty, with the guard wearing the least amount of protection, police with medium protection and SWAT in full armour.

The way users interact with the game is also standard, with moving performed via arrow keys or WASD, and shooting via mouse clicks. We chose to include the WASD option of controlling your character for comfortability of use, as we realised using the mouse to the right of a computer, and using arrow keys also the the right of a standard keyboard can seem unnatural.

**Handling Errors and Reversal of Actions**
Every display on the game (excluding the main menu) contains either a 'Back' button or a 'Main Menu' button. This allows the user to return to the main screen and end their game at any point. All connections are closed upon pressing the main menu button therefore new games can easily be started without having the close the game entirely and run it again.

There are also error screens if something goes wrong, that allow the user to go to the previous step and try again. These occur in situations such as a peer disconnecting from a multiplayer game, or an incorrect IP address being entered.

**Useful Features**
There are additional features within our game to make the user-gameplay experience much more fun and comfortable. An example of this are the arrows to show where other players are on the screen, with the arrow darkening the greater distance the player is away. We also have visual representations of health in the form of health bars for all enemies and players.



# AI, Networking & Collision Detection

## AI

Our game features simple AI for enemies that allows them to target players and find the best path to them, avoiding any obstacles in the map and other enemies. To do this, we implemented a modified A-star search. An enemy is assigned a node representing its starting position and its target destination, and the algorithm performs an A-star search on all adjacent nodes, checking for collision with the current map's collision map. This continues until the algorithm has found the assigned target node. If all possible nodes have been searched, and the target has not been found, then the path is null.

If the path is not null, then a method is then called to generate an ArrayList of nodes, from the target node to the start node, by looking at each node's parent node. This ArrayList is the enemy's path. When an enemy has a path assigned, it finds the first node in its path and begins to move towards it, moving an assigned amount of pixels each time the game updates. When an enemy has reached this node, this node is removed from the enemy's path and the enemy starts to move toward the next node.

Before moving, each enemy checks for a clear line of sight to its target. A line is drawn from the enemy to the target and each point of it is checked against the collision map for an obstruction. If there is no obstruction at any point of the line, then the enemy does not move and, instead, fires at its target. If there is an obstruction, then the enemy moves along its path.

Every enemy's path is updated periodically (once every 50ms) to ensure the path is always up to date with the current collision map. Clearly, this process is very intensive, and so, calculation is done in a separate thread. On an online game, only the host calculates search paths, as this absolutely ensures that each player will have the same path for each enemy.

Enemy's targets are also updated dynamically so that they always target the player that is closest to them. This was implemented so that players could use some strategy to lure enemies away from their teammates. A method is called that iterates through all the players in the players ArrayList and returns the player with the smallest absolute difference in x and y.

Later in the development of the game, we decided that we would like players to automatically walk to the exit of a map, after they have completed the map. To achieve this, we adapted the A-star algorithm to work for all classes that extend character, instead of just enemies. Every player in an online game is assigned a designated end location for each map, depending on their networkAssignedID. When all players have completed the wave specified as the last wave of a map, a path is generated from the player's current position to their designated end position and they begin to move towards it. The keyboard and mouse is disabled at this point, so that players cannot stray from their path. When all players have reached their end position, every player is taken to the next map.

We found this challenging to network, and unfortunately due to time constraints, were not able to implement this functionality successfully for online games. For our final submission, we have disabled this functionality, and simply switch waves without animation.

## Networking

The networking for this game in doing in 4 main sections, one of which splits into 2 main sub-sections. The abstract view is that each update of the game marks its changes and sends these updates to all peers that are playing the game. Each peer receives updates from other peers, and updates their game according to the received data. The game uses both UDP & TCP traffic, but due to TCP induced packet loss on a UDP connection, we use minimal amounts of TCP during an actual game, but TCP exclusively during the game lobby.

### Network Manager

The network manager is the heart of the networking. Initially this is responsible for creating and accepting connections. It does this by creating a socket that will accept new sockets, and requesting the IP address from and UDP data that it receives. Then, using the IP from the socket and/or UDP, the network manager will create a dedicated receiver that will be in charge of receiving all of the data for a particular peer.

Having these peer receivers is very beneficial, because it allows us to monitor received packet IDs from each peer, and ignore any packet IDs that are out of order. Having the receivers separated means that the IDs do not need to be synchronized between games, which is extremely beneficial in terms of complexity.

As the network manager has a list of accepted sockets, and IPs, it can step through each peer in order to send data to it. Using a generic send method, the network manager will look at the last time it sent a packet of a certain type, and not send another type of that packet if it was within x milliseconds. This provides a very simple way to avoid overloading the network. Using the packet type, the network manager will also decide whether to send the packet by TCP or UDP. All game data is sent via UDP, with the exception of the important game lobby packets, and wave change packet, being sent via TCP.

### The Game Lobby

The game lobby is created by the host. This creates a Socket for the host and starts listening for TCP connections and UDP data. During the lobby, peers will type in the host’s IP address and establish a connection to the host. This will trigger the network manager to accept the connection and create a connection back to the connecting peer in response. The host will then respond to the peer with a list, detailing all of the peers that are currently connected to the game. The peer will look at this list and establish a connection to any peers that it doesn't yet know about.

Each player has a graphical lobby that will detail the names of each player on each team, and graphically indicate which team is which colour. Each peer then waits for the host to send a packet (TCP) indicating that the game should start.

**The Game**
The game splits into two subsections. These are the creation of packets, and the receiving of packets.

**Creating Packets**
During each iteration of the game loop, each player generates a series of packets. Each packet details different aspects of the game. The player's current position is placed into a CharacterPacket, for example, and the bullets for a player gets placed into a BulletArrayPacket. The host has the extra job of generating packets for all of the enemies. After the "update" loop has finished, all of these packets will be sent to the network manager to be sent out over the network.

**Updating the Game State**
The network manager, when it receives a packet, will parse it and retrieve all of its data. Using the interface to game that the network manager has, it calls methods in game (using the packet data as parameters) that update the relevant parts of the game. For example, a position packet may do something such as "game.updatePosition(id, x, y)"

**The Packet**
Each packet extends the supertype packet, which offers functionality to aid with turning variables into byte arrays. Each packet must implement a method that will pack all of the data from the constructor into a byte array, and a method that will reverse this process. Each packet begins with a global packet header, so that it's possible to read this header to understand what type of packet needs parsing.

## Collision Detection

We have implemented our collision detection using a byte array. This byte array contains data on certain squares on the map, marking if they are solid or not solid. This array is created in our map creator and then written to a text file, and read in as the collision map during the loading of maps in the game. When the game is played, this collision map is updated to hold information about players, enemies and pickups, all represented by a different number in the byte array.

We first tried to implement our collision detection by defining one point on the player to determine whether they were touching another object or a wall. It would contain the value of the players top left position and then use their height and width. For example, if the player was moving right, it would add the player's point plus the the players width. If they were moving down, it would calculate the player's point plus their height. However, if you moved down and then up (very slightly) the point where it calculated you to be, and what square you were actually in, differed completely. Therefore we decided that the best way to implement the player's position was to update the collision map with every square that the player occupied. Our tiles were 10 pixels by 10 pixels, and our player was 50 pixels by 30 pixels, therefore the player took up 15 squares.

This also made handling collisions with bullets and pickups a lot easier. When a player moved, it would check all the squares where the player was moving to. If in any of those squares was a solid, then the player could not move. If in any of those squares there was a pickup, it would find what pickup it was, delete it, and give the player the benefit of that pickup.

Representing our collision map this way also made bullet detection a lot easier. Each player and enemy had an array list of bullets. Each time a bullet moved, it would detect where it was on the map. If it was currently in the square of an enemy of player, it would take health off of them. If it was on a wall, it would delete the bullet from the respective character's array.

We also created a visual editor in our game to make debugging a lot easier. This visual editor displays the collision map as the game is being played. Different objects are be shown in different colours, and are updated dynamically. This helped us to see if objects were being represented on the collision map properly. For example, when a pickup is created, its squares should be added to the collision map. When it is deleted, the squares should be removed from the collision map.

## Software Engineering, Risk Management

Our team took an incremental plan driven approach to creating our project. Initially we created a requirements definition in our specification that we submitted in week 2. We also submitted a GANTT chart of when each of those increments will be completed.

The requirements were gathered using all team members agreeing what we wanted, and what we believed we could achieve. We also spoke with our tutor to make sure that the project was viable, and that it met the specification of our module.

When the requirements had been defined, we then had to estimate how long each one would take, and who would be responsible for each one. Before we started, we each had a meeting to discuss each others strengths, and what they would like to do. Based on that, we allocated the work and made sure it was equal. Then we met and created a GANTT chart on teamwork, we made sure that features that were most vital to the project were assigned first. We then went around the group asking how many work hours we thought each task would take, and converted that into weeks based on us spending on average 3 hours a day on the project.

Each week when our individual pieces of requirements were completed, we would meet to discuss the progress that we had made, and where to continue from there. We decided to create the most vital parts of our game first, and create a working game as quick as possible, and then add additional features from there. We also made an integrated game from the start, and added additional functionality on top of what we had already created, so that we had a working product at the end of each iteration.

There are a couple of examples of when we have checked our progress and changed our original specification. The first is when we spent too long on trying to implement players moving between waves. It wasn’t a vital requirement, however we spent too long on trying to implement this feature. Another example of this is when we took too long implementing networking, we decided to remove the requirement of multiple game modes, as our networking was a more vital requirement. Apart from a few minor changes, we stuck to our specification extremely closely.

Throughout the project we documented a test log, with bugs we had found throughout the project. We would then discuss them during our next meeting, and try and resolve them. Then, in our final week, we had more extreme testing where we tried to break our game.

We used many software engineering principles that are important in incremental development, such as pair programming, and we split the entire project into mini-projects. Pair programming increased the efficiency of our development, as bugs were detected quicker, and helped us to understand each other’s code, as we wrote it together.

In week 7, we had a usability testing session of other people's games, we took feedback that they gave on board, and made the required changes to our GUI. Informal feedback was also obtained by showing our work-in-progress game to our friends. An example of feedback taken on board from usability testing is the addition of health bars to the game.

**Risk Management**
Before the project, we tried to manage risk as much as we could. When we created our GANTT chart, we allowed contingency time at the end of our project, as we allowed more time than we needed for testing our project and writing our report, which was needed as our networking took longer than expected. When we defined our requirements, we evaluated whether any of them were high risk, if it would cause a threat to our game or that we could not implement. If they were vital to our game and the requirements of the module, such as networking and AI, we made sure we could implement these features, and that we had selected the right people to do them.

We evaluated the strengths and weaknesses of each team member before we started, so that we could distribute the work based on these, and see what risks could affect the development of our project

**The Team**

**Strengths**
- Strong team relationship/communication
- Good abilities with Java
- Some software engineering experience
- Knowledge of OOP

**Weaknesses**
- Little to no game development experience
- No previous networking experience
- Missing a team member
- Little to no graphics experience with Java
- Little to no HCI experience
- Most members have no AI experience

**Dave**

**Strengths/Preferences**
- Maths experience
- Would like to learn some simple AI

**Weaknesses/Dislikes**
- No previous AI experience

**Jaeren**

**Strengths/Preferences**
- Good java knowledge
- Good problem solving skills

- Good project management skills

**Weaknesses/Dislikes**

- No experience with AI
- Not enough experience with networking
- No experience with using well known libraries in my program
- Poor graphic making skills

**Jordan**

**Strengths/Preferences**

- A little networking experience with Java
- Some AI experience
- Preference to do work either with networking or the game engine

**Weaknesses/Dislikes**

- Wish to not do graphics/no graphics experience

**Zoe**

**Strengths/Preferences**

- Experience with graphics software
- Maths experience

**Weaknesses/Dislikes**

- No experience with AI or real-time networking

We then created prototypes, to show a non-integrated version of the functionality we needed. Dave created an AI prototype, Jordan a networking prototype, Jaeren a map creator prototype, and Zoe created our first map.

We then integrated our prototypes in week 3, and started using the SVN straight away. This reduced risk as all of our source code was continuously backed up, and if any of our computers crashed, then the source code was backed up. It also meant that functionality we added that could cause a problem to our game could easily be reverted back to when the game worked, by reverting to an earlier revision. This early integration also meant that we always had a working product, so that we didn't have to integrate all of our different prototypes together at a later stage, and not having a working game at the deadline.

We also used pair programming continuously throughout our project.  Jordan and Dave together helped implement our AI engine, Jaeren and Jordan implemented networking, Zoe and Jaeren implemented collision detection, and Zoe and Dave implemented new maps together. This helped reduce risk as if one of our team members was ill, or couldn't work on a particular aspect for some reason because they had another deadline or commitment, then the other member would understand the code, and could carry on with development.

## Evaluation

Our finished product is extremely close to our specification given in the second week of development, and meets nearly all of our requirements. We came up with the idea early at the start of the second week and incorporated all of our members ideas. The designs that we created, including our class and networking diagrams, were followed closely for our finished product.

The game we have created covers the specification given to us at the start of this module. We have created a game which has considered HCI issues well as discussed above, with a well designed GUI. It also has competitive and cooperative aspects on our online game, where different players can use different tactics. For example one team member concentrates on killing enemies and the other on collecting randomly generated pickups that are spawned across the map.

There is also fantastic AI, where enemies will follow any numbers of players on any map, around obstacles of the map. It will also stop and shoot a player when it has a clear line of sight, and target the closest player to the shooting enemy. Our networking is easy to use, as it uses peer to peer networking, with no need for a server. Gameplay online is smooth, as we created our own packets, so that minimal data is sent across the network. Our game features high quality graphics, that are all original.

Our game is also easily maintainable and can easily be extended. All of our AI and networking will work on any map, and any change made to single player will automatically change online gameplay. Our map creator, used for creating our current maps, can also be used easily to create new maps for future versions of our games.

Because we started early, it allowed us contingency at the end of our project when we overran on aspects of our game.  We also integrated our work from the very start of programming it meant that we didn't waste time later on integrating our different programs. Our use of pair programming helped to increase efficiency in our programming, and also reduced the risk in our project. We also tested from the very start, in the first few weeks we had informal testing where we would each post on the Facebook group bugs we had found, and then we would discuss them when we met. After our meeting with Ela, who suggested we documented these, we kept an ongoing test log. We also created JUnit tests which were used throughout implementing new features, to test that our main functionality, such as collision detection and sending of network packets, still worked. We believe the testing we carried out has made our game more robust.

We worked well as a team, had more than one meeting per week, and all got along extremely well. We were honest with each other with the amount of work we could do each week, and skills that we had, so that we could accommodate to different peoples deadlines. We each worked to our strengths and divided work up equally at the start so that responsibilities were known early on in the project and our topics could be researched.

Although our game met most of the requirements that we set out in our specification, there were certain requirements that we did not meet. Our main criteria that we missed was that we were going to implement multiple game modes, which we did not have time to implement.  We also managed to not completely meet one our non-functional requirements.  We hoped that our game would work perfectly on at least 3 platforms.  We have tested our game on all 3 platforms, and occasionally the game can crash on mac when music is playing.  The reason we did not meet some of these requirements is because networking took longer than we expected. We started networking quite late and this delayed the development of features such as the competitive aspect of our game, and therefore we didn’t have enough time to implement multiple game modes. The reason we started networking late was that we wanted our single player game to be complete before we started the networking, so that we knew what information needed to be sent.

We followed our class design extremely closely, however our networking diagram did have to be changed. Throughout the game we had hoped to send packets using both TCP and UDP, however we found out about TCP induced UDP packet loss, and our design had to be changed slightly. Our original idea for collision detection also had to be changed. Originally we would only have one point which would represent where the player was stood, and we would calculate if there was a collision using that one point. However we had to change our design, as collision between bullets and pickups could only collide with a player from that exact point.

Throughout development, we occasionally spent too much time on features that weren’t vital to our game. One example of this is that we spent multiple days on trying to get walking to an exit between waves to make our game look better, however in the end we discarded the functionality for online play, as we realised we needed to concentrate on other things. In our plan we didn’t allow enough time for testing, and only had one week to test all functionality in our game that hadn’t been spotted before that. Although our game was created extremely early, and we had a finished game over a week before the game was due, we had more bugs than we expected near the end, and were working late to fix bugs close to the deadline.

If we were to to carry out this project again, then the main thing that we would change would be to start our networking earlier. We wanted to finish single player before we started networking, however we should have just created the base of our game, and defined our requirements more specifically so that we knew exactly what information needed to be sent across the network without creating the entire single player game before hand. Our networking was started in week 6, however collision detection was implemented in week 4.  We could have started writing packets, and sending small amounts of data in week 4 or 5 at the latest.

We would have also researched the use of TCP and UDP at the same time in more detail, to find the problems that could occur. When we were testing our game, unexpected errors occurred that were due to the use of TCP and UDP and delayed development. As a team we also should have met more to try out 4 player online functionality, as when implementing networking we only tested it using 2 players. However many different problems occurred when playing our game with 4 players as more information needs to be sent and more information needs to be processed, for example with our AI. If we did this project again, after every major networking feature was added we would have met as a 4 to play our game.

We should have also dedicated more of our time to our testing. As we kept a log of all our bugs that we found, we believed that our game wouldn't have a large amount of bugs. However when we tested our game trying to intentionally break it in the last week of development, we found more bugs than we expected. Due to the amount of time spent in the last week, we still have a robust game, however that was more time than we would have expected during the last week to be spent on testing.

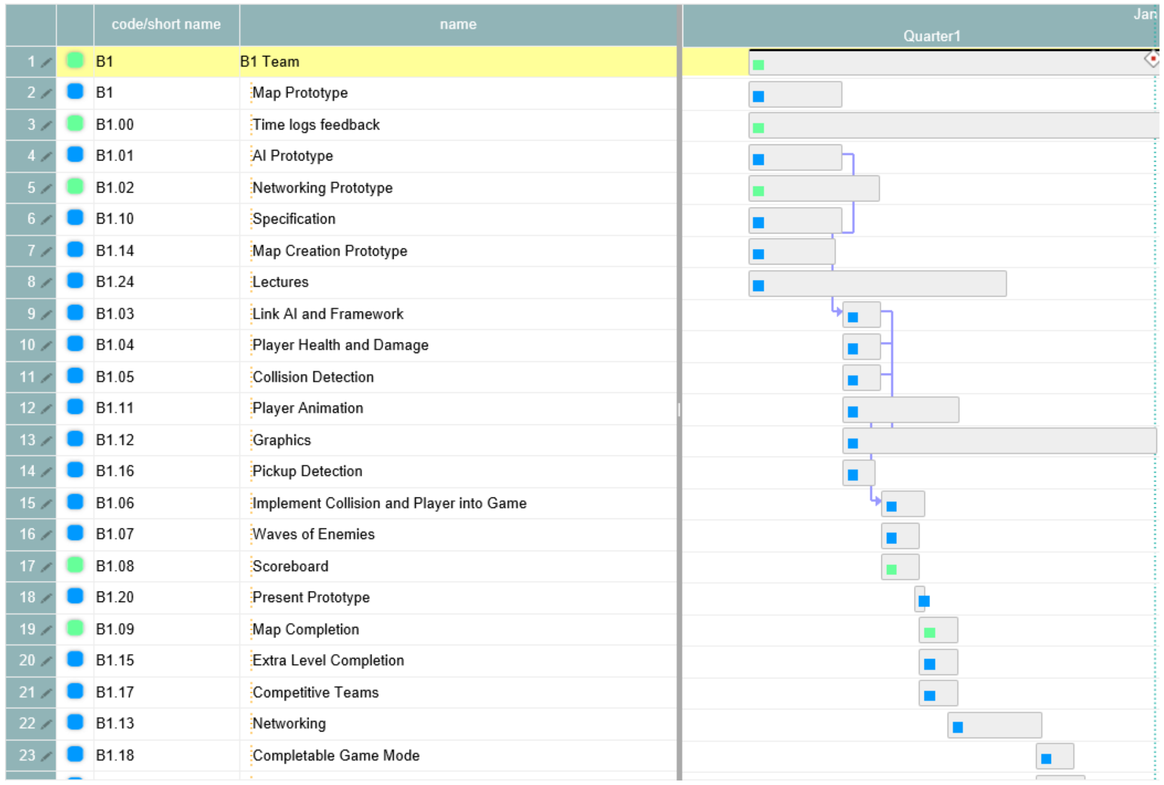
Overall our project was carried out extremely successfully, and the final product we created we are extremely proud of. Nearly all of the functionality we wanted to implement, and that needed to be to meet the specification of the module was implemented. We also followed a detailed design and specification for our product, made in the second week of the product extremely closely. Our team worked extremely well and formed a close bond. We discussed all problems we encountered face to face, or over Facebook, and there were no conflicts whatsoever. Work was distributed evenly, and each team member made a massive contribution to our project.

# Team Work

As a team of four, we knew right from the beginning that teamwork would be very important to us. We knew that we would have to effectively operate as a single efficient unit in order to create a final product that was of the highest standard and was something at we were all proud of. We also knew that each individual member would have to work harder and that having a strong team would be essential in order to support each other. Fortunately, our team bonded very quickly and we were performing as a single unit almost immediately.

During the design process, we all met very regularly (three times a week) in order to decide on the initial idea. Everyone suggested several ideas for games and all the other members of the team discussed each idea and contributed their own thoughts. This was very useful, as it highlighted the strengths and weaknesses of each idea and also allowed the person suggesting the idea to elaborate more and think a little about how certain features would be implemented. For the final game idea, Dave suggested that we should create a wave-based game featuring zombies, but Zoe felt that Zombies were too overused, and she wanted a more original idea. This is a great example of how the great communication within our team, and our team member’s ability to listen to each other’s ideas, made a good idea into a great one.

It was clear from the outset that every team member was very motivated. In the first week, we divided up the main areas of the game between our team members, based on each member’s strengths and interests (somthing discussed immediately). This was designed to ensure an equal division of work. We then used this to create this Gantt Chart:



| | | code/short name | name |
|---|---|---|---|
| 1 | 🟩 | B1 | B1 Team |
| 2 | 🟦 | B1 | Map Prototype |
| 3 | 🟩 | B1.00 | Time logs feedback |
| 4 | 🟦 | B1.01 | AI Prototype |
| 5 | 🟩 | B1.02 | Networking Prototype |
| 6 | 🟦 | B1.10 | Specification |
| 7 | 🟦 | B1.14 | Map Creation Prototype |
| 8 | 🟦 | B1.24 | Lectures |
| 9 | 🟦 | B1.03 | Link AI and Framework |
| 10 | 🟦 | B1.04 | Player Health and Damage |
| 11 | 🟦 | B1.05 | Collision Detection |
| 12 | 🟦 | B1.11 | Player Animation |
| 13 | 🟦 | B1.12 | Graphics |
| 14 | 🟦 | B1.16 | Pickup Detection |
| 15 | 🟦 | B1.06 | Implement Collision and Player into Game |
| 16 | 🟦 | B1.07 | Waves of Enemies |
| 17 | 🟩 | B1.08 | Scoreboard |
| 18 | 🟦 | B1.20 | Present Prototype |
| 19 | 🟩 | B1.09 | Map Completion |
| 20 | 🟦 | B1.15 | Extra Level Completion |
| 21 | 🟦 | B1.17 | Competitive Teams |
| 22 | 🟦 | B1.13 | Networking |
| 23 | 🟦 | B1.18 | Completable Game Mode |

Without prompt, every team member then went away and wrote one or several prototypes based on their assigned area to show the rest of the team at the next meeting. Since no-one was asked to do this, everyone showed everyone else in the team that they were committed and hard-working, and this helped to establish trust between everyone very early on.

One of the main strengths of our team was that we all used SVN almost immediately. After we all made our prototypes, we all spent time combining them into the final game via the SVN repository. This was very beneficial because it meant that every member was working on the same piece of code from the get go, rather than everyone working separately and then struggling to combine their work at the end. As a result, the final game came together very quickly and we had a working single player game with all of the features we desired by the prototype presentation in week 5. We also often committed changes to SVN so that everyone was working on the most up-to-date version of the game. We also made an effort to make detailed comments and Javadoc all of the code that we wrote so that anyone in the team could look at it and understand it and add to it confidently. This made writing the code much more simple and time efficient, as anyone could easily work with anyone else's code, without needing to ask them to explain it first. By the end of the project, we made over 650 commits to our repository. That's an average of 16 commits per person per week for the 10 weeks that we were making the game.

Another strength of our team was our regular use of pair programming. This was mainly used when writing a particularly difficult and complex piece of code. This greatly helped to reduce errors and so cut down the debug time. Another benefit of this was that two people had understanding of the complicated code, rather than just one. This meant that later, there was always someone to ask about the code, if necessary. Over the course to the project, every team member pair programmed with every other member at least once. Examples include: Zoe and Jaeren working on the collision maps, Dave and Jordan working on implementing the AI into the game, Jaeren and Jordan working on the networking, and Dave and Zoe working together to extend the A-star search to work for players as well.

General communication amongst our team was excellent. The ethos of meeting face to face that was established in the first week was kept up throughout the whole project, with face to face meetings happening at least once a week and often more if an important concept needed to be discussed. For for minor issues, we employed the use of Facebook to keep in contact with the whole group wherever we were. As a result of all this, everyone in the team was constantly kept in the loop and allowed to share their opinion on every single decision made.

We also used these meetings as a means of testing our game and giving feedback to each other on individual aspects of the game. Facebook, in particular, proved very useful for this, as someone could highlight a bug and then the person responsible could work to rectify the bug immediately, without the need to waste time getting the whole group together. It was useful to highlight and resolve bugs immediately so that the code was always stable when new code was added. All of the team's observations about the game eventually became the test log and there are hundreds of examples of where a team member has noticed a bug or suggested an improvement that resulted in a much better game.

Our team was also very flexible. Our team members often had coursework deadlines and placement interviews, and all the other members appreciated the importance of these and so allowed that member to work on the game a little less for that week. No-one ever suspected that anyone was being lazy and this is the result of the trust built up between us all early on - it was obvious that everyone was very committed to the project.

In conclusion, our team worked incredibly well together and this affected all areas of our project. We quickly bonded and made a design that everyone was happy with and so willing to work hard. The result of our quick use of SVN and constant communication with each other was that we had a working game by the end of week 5 and ensured that everyone had a clear understanding of the game as a whole. Our constant communication also allowed us quickly spot bugs and alert the rest of the team, resulting in a much more robust game. Over the course of the project, we have gotten to know each other really well  and all of this has led to firm friendships amongst us all that will definitely continue after that project has ended.

# Summary

Below is a summary of each section of our report -

**Introduction -** An introduction to the game, explaining what our game is and the scope of the project

**Software Design** - Shows the design of our game.  Featuring class diagrams, and discussing the main classes in our game.

**Game Design, GUI & HCI** - Discusses the main design principles for why we made the game look like we did, and how it makes the game more pleasurable.

**GUI, AI, Networking, Collision Detection** - How the most important aspects of the game work, and how they improve gameplay

**Software Engineering, Risk Management -** How the project was managed and what risk management techniques we took, to make sure our game was created on time.

**Evaluation** - An evaluation of the project, and how well we worked as a team.  Explains what we would have done differently, if we had the chance to do it again.

**Team Work** - How well we worked as a team, and what techniques we took to make sure everyone was involved, and everyone's input was taken into account

**Summary** - Summarises all sections.

**Individual Reflections** - Reflections from us all on the whole project.  Discusses what we learned, what we did well, and what we would have done differently.

## Individual Reflection - Zoe

As a team of four, our focus was to get programming as soon as possible as we knew we'd have to do more work individually to make up for our disadvantage. We all agreed to set firm deadlines for tasks, and our first deadline came in the middle of the second week, where we said we would finalise our game idea.

In the first week when we got together and shared game ideas, we were all keen on a zombie wave-based game suggested by Dave. However, I pushed for a change of concept, since Zombies had been done so many times. This is when we looked up classic rival pairings, and agreed upon cops and robbers.

All my team-mates made it evident that they had little to no experience with graphics whereas I had some experience, so it was decided I would make the graphics for the game. It was clear that I had to keep in the forefront of my mind Human Computer Interaction, and designed the maps in a way my team and I felt was intuitive for the user to get around. All sprites, maps, objects, enemies and end screens were made by me. I also implemented the animation in the game in regards to the players and enemy movement.

One of the main things I programed was the multiplayer capabilities. Once Jordan and Jaeren had completed the networking which allowed all users' players to be sent through the network, I implemented our team based idea, which involved typically two teams of two (on blue or red teams). I made sure all team functionality was implemented, and also that the game was completable with relevant end screens for both completion of the game and failure to complete the game. Additionally, I extended the game by implementing music and sound effects, implementing spawn points for enemies as before I did this enemies appeared randomly on the map and we felt it would be better to have spawn points instead, and also worked with Dave to have players use AI to walk to the next map. As well as these functionalities, I have assisted in implementing and debugging many other aspects as have all my other team members.

My team felt it was very important to meet face-to-face frequently, and met up at least once a week as a result of this. We also informed each other of any major changes we have made on a private facebook group. This allowed us to always be up to date with what was going on. On many occasions we did pair programming, which has allowed us to get code programmed faster and also has resulted in us each being more aware of each others code, and our final game having less bugs to deal with.

I am very pleased with my team; all ideas have been run by each other member of the team before implementing, no disputes have occurred, and we just generally get on together which has always been very beneficial in allowing us to work quickly and efficiently. There has never been a time I have been unsure who's working on what or what I should be doing. All this has meant we have worked hard to ensure the fact we only had four team members did not affect the quality or how extensive our game was. I couldn't have asked for a better team!

**Individual Reflection - Jordan**
I was anxious at first about working on a team project, mainly due to the little teamwork experience that I had previously attained. My previous teamwork mainly involved teams of people that I was previously friends with. After learning who my team was, we met up minutes after, and planned a date for our first team meeting.

We quickly, as a team, decided on a target area that we wanted to design our game around, and quickly had an idea about exactly what we wanted to create. As a team, we got on excellently. Everybody was flexible around other commitments, and willing to help out with any tasks that required doing, even if that team member wasn't tasked to work on that area of the project.

Decisions were made unanimously, via the usage of a Facebook group, or via our team meetings. The communication, friendliness, and motivation for the team led us to creating a brilliant game, within the deadline, despite only having 4 team members. Progress throughout the development of our game was both regular and fast.

My main area of development for the game was providing the skeleton code for the game, and developing the networking. The networking was a large part of the project, and fairly complicated. I had an idea of how to implement this, and ran by idea past the rest of the team, and other friends, for feedback. I designed and developed the network manager, and packets, and the required code for sending, receiving, and verifying the packets between peers.

Once I had finished writing the sending/receiving/verifying code, Jaeren and I began working together to create and integrate all of the networking. This included design decisions on how to package different types of data, and handling how the data should be used when received. This led to me and Jaeren having many extra meetings together, outside of the regular group meetings, in order to work on the networking in a pair programming approach.

We used pair programming a lot throughout the project, and this worked really well in my opinion. It meant that multiple people could work on code, without the added time required for familiarisation. It also meant that if a problem was encountered, there was somebody experienced in the code to bounce ideas off of. Jaeren and I certainly benefited from the debugging aspects of pair programming in regards to implementing the networking.

The teamwork skills that I've learnt from this module will certainly benefit me in the future, as employers will value the experience that I have gained. Aside from that, I have benefited through the meeting of my team members - all of whom are great people. I genuinely believe that I could not have asked for a better team for this project; the communication, development and collaboration shown throughout the entire project was simply excellent.

**Individual Reflection – Jaeren**

My individual reflection of this project is extremely positive. We met the first week of team project and decided on what our game would be. From then on, our team met every Wednesday to make progress. In our second week we had created our class diagram, and had an exact idea of what we wanted to create from our specification. We immediately used the SVN, and Jordan created a framework based on our class diagram.

From then on we all started making our own individual contributions, and adding them to the SVN, so that our entire project was integrated since the start. My responsibility in the first half of development was collision detection. Zoe's was to create the graphics and HCI, Jordan created a scrolling map and Dave to create AI and enemies.

I created a map creator that could take a black and white image, or a user could select which items are solid and those that are not, and it would serialise a byte array into a text file, which could be loaded as the collision map in the game. From there on, I added players, enemies and pickups that were dynamically represented on the collision map. Each object was represented on the collision map so we could determine what was to happen when an object came into contact with them. I then added a visual representation of the collision map that displayed as the game was playing.

In the second half of the project I was working on networking with Jordan. We used pair programming, meeting up 3 or more times a week. Jordan was in charge of the actual networking, and I was in charge of sending and dealing with the packets when they were received. Jordan was extremely experienced in networking, and I learned a great deal about low-level networking and how it works.

I have learned a great deal in this module, which will be extremely useful in my future education and career. The main benefit I have learned is new programming habits that other team members have. Such as using enumerations, using interfaces to communicate between two classes, and many eclipse shortcuts. I have also learned how to quickly read and understand other people's code. Another benefit of this module was that I got experience with using version control, and how to resolve conflicts. We often spoke on Facebook, or in person to solve conflicts, and understand the changes other people have made.

Overall our group worked extremely well from the start, and we got on fantastically. Work was equally divided throughout the group and we all worked to our strengths. Throughout this project I have learned team-work, new programming skills, and version control amongst others, all skills that I have been asked at every job I have applied for, so I have no doubt that this module will benefit me massively in the future.

## Individual Reflection - Dave

Going into this project knowing that I was going to be part of a team of four instead of a team of five, I was nervous. I knew that every one of us would have to work harder in order to create a final product of the same quality of the other team. Luckily for me, our team instantly gelled together and we were working as a single unit before the end of the first week.

Our close relationship and strong focus on teamwork allowed us to have a nearly completed single player game by the end of week five. We achieved this by frequently meeting face to face and communicating via Facebook whenever a face to face meeting was inconvenient. We were flexible and accommodating to everyone's individual needs; such as coursework deadlines and interviews and this helped create and strengthen the friendly relationship between us all.We all contributed to the design of our game, ensuring that everyone was happy with the final idea. We also often practised pair-programming when writing particularly difficult or complicated code and this helped us to reduce errors and provide another option on how to do something. I found everyone in my team to be very motivated and this inspired me to work as hard as possible, in order to make our game as good as it could possible be.

My role in the team was primarily AI and enemies. I had no experience of AI before this project and so had to go away in the first week and do a little research. I made a few prototypes and found that an A star search would be ideal to implement for the enemies in our game. I wrote a basic A star prototype to show the rest of the group and they agreed that it would be perfect. With the help of Jordan, I then implemented the A star search into our final game, using the collision map that Jaeren had already created for the map that Zoe had already drawn.

I was also tasked with creating the wave object that holds all the enemies for the wave. I also created methods that increased the wave number and, simultaneously, increased the number of enemies in the wave. After we had added more enemy types, I also wrote a function that increases the probability of more challenging enemies appearing on higher waves, so that the difficulty scales up as the player plays for longer.

As we added more maps, we decided that we would like players to walk to the exit of a map automatically and we concluded that we could reuse the A star algorithm for this. This was simply achieved by allowing all objects that extend Character use the A star, rather than just enemies. Then, all players were assigned an end position based on their ID and a path was created from the player to their end point. Players then followed this path until they reached their end position. When all players reached their end position, the next map would load.I also worked a little with the networking as I had to implement player death and respawning over the network. I also contributed a little with GUI: I drew arrows that shows the direction of all players that are offscreen and added health bars to the enemies and player names to all the players.

In conclusion, I believe that my team and I have worked incredibly hard to produce a game that I, personally, am very proud of. I think that the time and effort that everyone contributed has really paid off. I have honestly never been in a better team and think that we will all remain friends long after the project has finished..