



# ADVANCED HIGHER COMPUTING PROJECT

## The Pirate Game

### [Abstract](#)

A multiplayer grid-based game, developed in Python using the PyGame and PodSixNet libraries.

Megan Briers  
6K1-1K1

## Table of Contents

<b>Analysis and Planning .....</b>	<b>3</b>
Project Proposal .....	3
Analysis and Research.....	4
Outline Project Plan .....	9
Detailed Project Plan .....	10
Gantt Chart .....	12
<b>Requirements Specification .....</b>	<b>13</b>
Purpose of Solution.....	14
Description.....	13
End Users .....	15
Functional Requirements.....	15
Inputs and Outputs .....	16
Scope/Boundaries .....	17
<b>Test Plan .....</b>	<b>19</b>
Component .....	19
Modular .....	23
System .....	24
Users .....	25
<b>Interface Design.....</b>	<b>26</b>
Main Interface.....	27
Movement.....	30
Validation .....	30
Inputs and Outputs .....	33
Purpose and Functions .....	34
<b>Data and Program Structure .....</b>	<b>36</b>
Pseudocode.....	36
Class Diagrams .....	72

Data Flow .....	73
RAM Based Entities .....	85
Refinements .....	89
Validation .....	90
<b>Implementation.....</b>	<b>91</b>
Development Issues .....	91
Main Interface Elements.....	97
<b>Final Testing.....</b>	<b>99</b>
Interfaces .....	99
Main .....	104
Server .....	114
Modular .....	116
User Testing.....	117
Corrective .....	121
Problems .....	121
<b>Evaluation .....</b>	<b>122</b>
Summary .....	122
Features .....	123
Analysis .....	124
Development Process .....	126
Skills gained .....	127
Own Performance.....	127
<b>Bibliography.....</b>	<b>128</b>
<b>User Guide .....</b>	<b>131</b>

(For additional support, please see pink book or black folder)

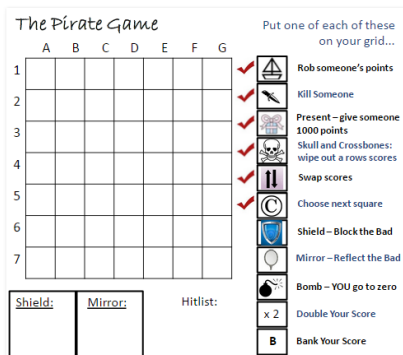
## Document 1 – Project Proposal

### Introduction:

Outline of software product:

The Pirate Game is a quick game that starts by requiring the player to place different tiles in a 7 by 7 grid. The game will randomly generate a grid co-ordinate after each player has filled up their grid and the action of the player each round will be dependant of what is placed in the square. It is a multiplayer game which will have to be played over a network as it requires three players. The interfaces will change throughout to allow users to make choices based on their tiles.

The grid interface will be designed as below:



### Aim of the game:

- To have the most money out of all the players at the end of the game.

### Implementation:

(Advanced Higher Project Criteria)

- Will use 2D arrays to store grid co-ordinates
- Will use a sort algorithm to help generate random co-ordinates throughout the game
- Will use a sort algorithm to assess who has won the game

### Challenges to overcome:

- Work with different players over a network
- Work with 2D arrays in Python
- Interactive User Interface with PyGame

### What I will gain from the project:

1. Skills in working over networks and on multiplayer games
2. Extended skills in Python
3. Working in PyGame

### End User Group:

- High School Children (Old enough to understand the rules of the game), as the game was originally created for secondary school maths classes as an end of year game.

**Commented [MB1]:** Plan was to initially have a minimum of three players, but later decided to set a maximum of three players. This could be adapted in the future relatively easily

**Commented [BR2]:**

**Commented [m3]:** Initially the plan was to use a sort algorithm to generate a high score table as an additional interface, however this plan was later scrapped

## Document 2 - Analysis and Research of Project Proposal

### ***Assessment of User Requirements***

#### User Survey

To assess the user requirements, I created a survey using SurveyMonkey which I shared with my network to gain responses.

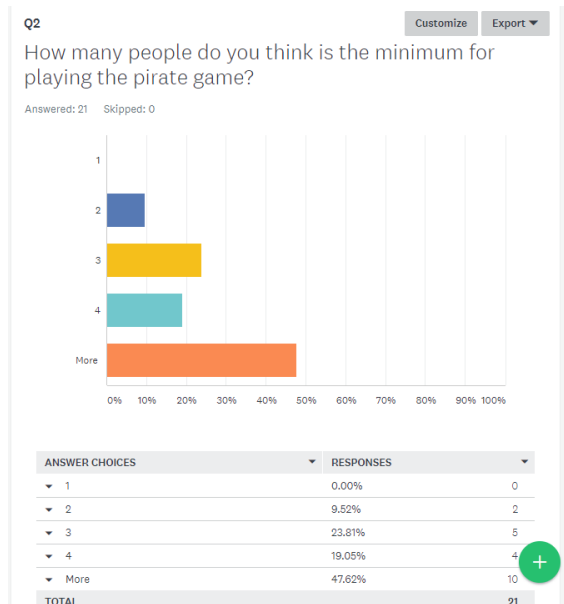
Link to my survey:

<https://www.surveymonkey.co.uk/r/DPSGMMMD>

Paper copy of my survey:

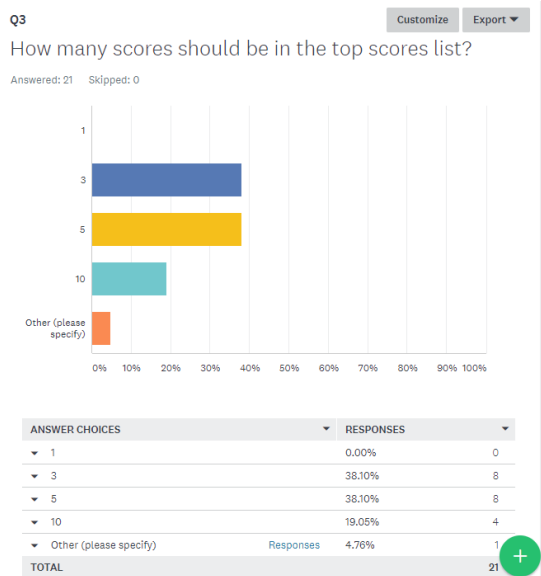
1. Have you ever played the pirate game before?
  - Yes
  - No
  
2. How many people do you think is the minimum for playing the pirate game?
  - 1
  - 2
  - 3
  - 4
  - More
  
3. How many scores should be in the top scores list?
  - 1
  - 3
  - 5
  - 10
  - Other (Please specify)
  
4. Please comment on what features you would expect in a digital version of the pirate game
  
5. Would you require the rules to be explained before beginning the game?
  - Yes
  - No
  
6. Would you like to be able to drag the tiles onto the grid during set up or type in the grid co-ordinates of the place you want your tile in?

## Analysis of Survey Responses



- Just under the majority agreed that **more than 4 people** are needed to play the pirate game, however I need to see how I will implement my server and how difficult that is before I can get a clear understanding of how many people I will be able to get playing the game

**Commented [MB4]:** Initially I didn't have an idea on how many people should be allowed to play at one time, but as implementation went on I made the decision to restrict it to 3 so I wouldn't have to implement a 'choose how many players to play' feature and also I could test it more easier.



- I will choose between whether the high score table has 3 or 5 players
- The other option that was stated was 98.

Q4

Export ▼

Please comment on what features you would expect in a digital version of the pirate game

Answered: 19 Skipped: 2

RESPONSES (19) TEXT ANALYSIS MY CATEGORIES

Categorize as...

Filter by Category ▼

Search responses



Showing 19 responses

Animation of the different characters interacting

9/13/2017 6:25 PM

[View respondent's answers](#)

Gifs, some animation (not too much!), scored

9/12/2017 4:05 PM

[View respondent's answers](#)

Slightly crazy german maths teacher

9/12/2017 9:55 AM

[View respondent's answers](#)

Unsure

9/11/2017 11:34 PM

[View respondent's answers](#)

Computer

9/11/2017 9:04 PM

[View respondent's answers](#)

Grid, options to steal/kill other players etc : )



Rules

9/11/2017 4:06 PM

[View respondent's answers](#)

Sound effects, animations for things exploding, a bank to count your money easier

9/11/2017 2:17 PM

[View respondent's answers](#)

Motion pictures

9/11/2017 12:22 PM

[View respondent's answers](#)

sound, animations

9/11/2017 12:15 PM

[View respondent's answers](#)

Randomize squares option? (e.g. the computer chooses where all of your items are placed)

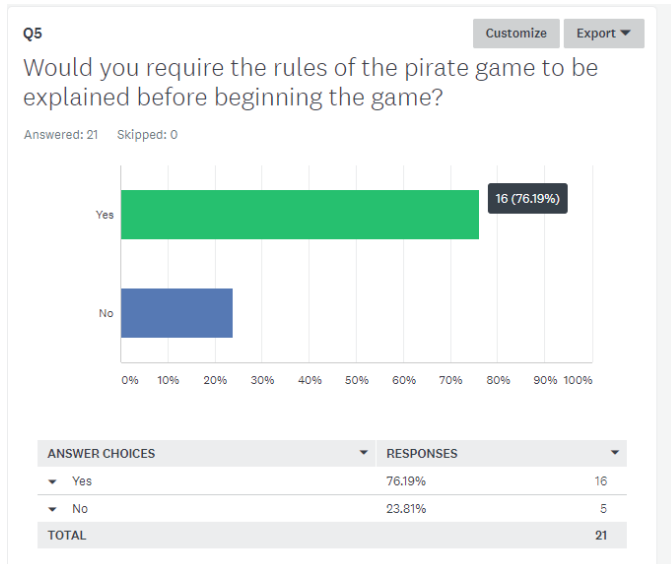
9/11/2017 12:15 PM

[View respondent's answers](#)

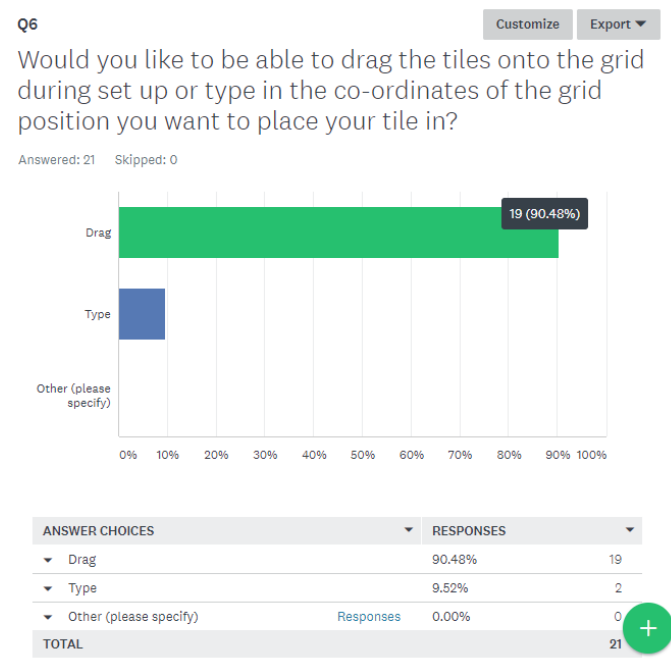


**Commented [MB5]:** The plan initially was to have a high score table to allow the users to see how they compared to other players, however due to time restraints and the lack of need to implement it due to using sorting algorithms at other points in my program, I decided not to implement the high score table.

Here is a sample of some of the responses from the question asking what features potential end users would expect from the pirate game. I will take these comments into consideration.



No comment required, due to an overwhelming majority I will make sure I implement an optional rules interface at the start of the program.





Even though most people suggested the drag option, I had to settle with implementing a keyboard-based input system. I choose to go with the keyboard option due to my initial lack of experience in PyGame, and the fact I wanted to make progress with the game.

Commented [MB6]: 26.10.17

#### Key End User Requirements:

Using the survey results and discussions with potential end users, I would say the key end user requirements are:

- Rules Interface
- 3 players at minimum
- Grid Display
- Close as possible to the original game
- Simple to play
- Images on the grid

#### ***Feasibility Study***

##### Economic

- My project will require no extra costs, as I will be working in Python and PyGame, which are both free to install and already installed on the computer
- I will be working on the current school computers so no additional hardware will be needed to be purchased.
- Since this project will not require any financial input and it will not be sold/marketed then there will be no costs throughout the process, and so it is economically feasible.

##### Legal

- My game is a computerised version of 'Pirate Game'. As far as I am aware I am the first person who has decided to implement the Pirate Game on a computer. I found the game on a teaching resources website and the terms of service state that I can collate, remix, list, organise or otherwise use the party's material if it does not entitle me to any intellectual property rights. Since I am not intending to make profit on this game then I am not breaking any clauses in the terms and conditions.
- The name of my game will be 'The Pirate Game' but it will not be on the game market or sold after completion of the project and so it will not breach any laws. Therefore, my project is legally feasible.

##### Technical

- This study does not rely on technologies that are currently under development, undergoing testing or not developed yet.
- Again, this study does not rely on any new hardware or software to be installed on the machine, as all it will need to be developed is the computers we currently have at school and the software that is currently installed on the computer. The project will be able to run in the environment it is designed for. This means that the project is technically feasible.

### Schedule

- This study will run approximately from end of September to the end of February. In this time, I will have to implement my program and complete all the documentation necessary to hand in the project. As part of my detailed project plan I will include a Gantt Chart to show the breakdown of the schedule. Throughout the development of the project, I will have to consider other schoolwork/prelims and potential breaks over the holidays. However, at this point in the process I believe I will be able to complete all the required tasks before the deadline, and so this project is feasible, in terms of schedule.

### Document 3 – Outline Project Plan

#### ***Record of progress***

- For my record of progress, I decided it in the form of a blog on Glow. The link to this blog is below.
- <https://blogs.glowscotland.org.uk/fi/ahproject/>
- You will need a glow account to access this blog

#### ***Time scale***

- The time scale for this project is roughly *starting at the end of September* and *finishing at the end of the February*.
- In this time planning, analysis, design, implementation and evaluation must be carried out.
- The program has been split into smaller sub units and rough estimates of time scales have been planned out.

First Draft of Timescale:

<b><i>Task</i></b>	<b><i>Time</i></b>
Requirements Specification	2 weeks
Test Plan	1 week
Interface Design	3 weeks
Program/Data Structure Design	2 weeks (though continuous process)
Implementation	12 weeks
Final Testing	1 week
Evaluation	1 week
Final submission of project	1 week

Final Draft of Timescale:

- These tasks are not all dependant on each otherwise

<b><i>Task</i></b>	<b><i>Time</i></b>	<b><i>Target Date</i></b>
Requirements Specification	2 weeks	24/9/17
Test Plan	1 week	1/10/17
Interface Design	4 weeks (split up through development)	14/1/18
Program/Data Structure design	2 weeks (split up through development)	15/12/17
Implementation	17 weeks	4/2/18

Final Testing	2 weeks	11/2/18
Evaluation	1 week	25/2/18
Final Submission of project	1 week	28/2/18

Tasks like Interface Design and Program/Data Structure are continuous throughout the implementation of the code and are always evolving, so that is why their target dates may not be as set as the other dates.

## Document 4 – Detailed Project Plan

**Note 1:** Tasks and Sub Tasks and **Note 2:** Realistic Time Scales

In the **documentation** of the project there are 7 main identifiable tasks:

*Analysis and Planning*

Time Scale:

2 weeks

*Specification*

Time Scale:

2 weeks

*Test Plan*

Time Scale:

1 week

*User-Centred Interface Design*

Time Scale: 6 weeks (alongside *Interface* task in implementation), continual process

*Program and Data Structure*

Time Scale: 2 weeks (continual process)

*Final Testing*

Time Scale: 2 weeks

*Evaluation*

Time Scale: 1 week

In the **implementation** of the project there are five main identifiable sub tasks:

*Networking* – Making my game multiplayer

Time Scale:

16 Weeks (including 2-week holiday in the middle)

*Interface* – Making the interfaces required

Time Scale:

Main: 6 weeks (this is long because I am completing this task alongside other tasks)

Interfaces included in the game: 4 weeks

**Commented [MB7]:** Initially 4 weeks

**Commented [MB8R7]:**  
(Refinement 16/10 : I initially made this timescale a lot shorter, but I have changed it because I have not encountered networking of games before and I am expecting to experience a lot more difficulties than initially expected)

Instructions: 2 weeks (will be included in the game part of the Gantt chart)

*Input* – Allowing the program to accept input  
Time Scale: 3 Weeks

*Game* – Making the fully working version of the game  
Time Scale: 16 Weeks

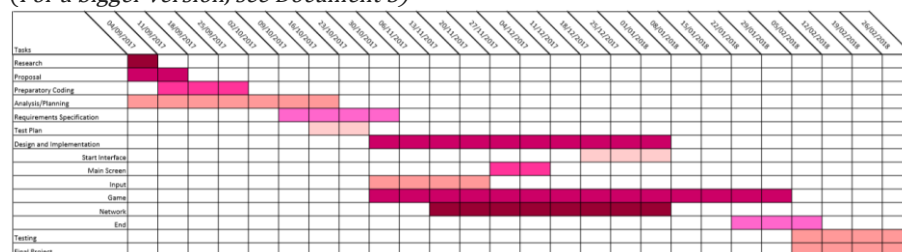
*End of Game (High Scores etc):*  
Time Scale: 6 Weeks (this time scale is long because the task will be complete alongside tasks)

(Please see Page 8,9 and 10 in my pink booklet for the extended breakdown of the sub routines)

Dependant Tasks:

- Must complete requirements specification/analysis and planning before I move onto implementation
- Must complete test plan before moving onto implementation
- Must complete some practice PyGame tasks before moving onto game (shown in Gantt Chart)
- Must complete input before moving onto game
- Must complete game before moving onto evaluation

**Note 3(Gantt Chart) is also included in a separate document (document 5)**  
(This is the finalised Gantt Chart; all the previous adaptations are included in document 5)  
(For a bigger version, see Document 5)



**Note 4:** Resource Management

- Need to have access to Python and PyGame, which are already on the school computers.
- Need to have access to the following libraries in Python: PodSixNet, time, PyGame, PyGame.font, PyGame.event, PyGame.draw, string, random, Buttons, os and sys.
- I don't believe at this stage in the process that I will require any additional hardware or software than what is already installed on the school computers. This is addressed more in the technical section of the feasibility study.

**Commented [MB9]:** 5 Weeks Initially

**Commented [MB10R9]:** This was set as quite long due to my inexperience in PyGame

**Commented [MB11R9]:** (Refinement 08/02 – Took less time than initially stated)

**Commented [MB12]:** 12 Weeks Initially

**Commented [MB13R12]:** (Refinement 08/02 – I initially made this shorter but it actually took me longer to fully develop the main game due to the difficulties integrating other aspects of the game into the program)

- To create the graphics, I am planning on using Paint and Paint 3D, another program already installed on the computers I am using to create this project.
- Need to have access to a computer (desktop or laptop) for implementing the program and need to have access to minimum 3 computers for testing purposes.

No additional hardware or software is required to complete my project.

#### Note 5: Potential Difficulties

To combat some potential difficulties, I have created a list of tasks to be completed before starting the project:

- Must understand basic commands in PyGame (for example: how to display images on the screen)
- Must do some initial research on networking tasks
- Must strengthen my understanding of 2D Arrays
- Must gain some experience in PyGame

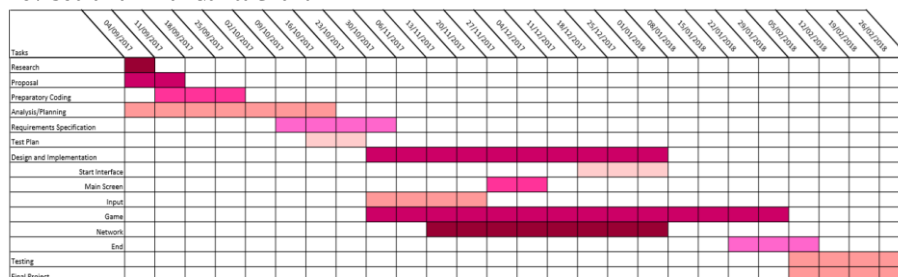
#### Some potential difficulties:

- Due to my lack of experience in PyGame, I am aware I may run into a lot more errors than I would do if I was coding in Python only. I am also aware this may cause slight delays in my schedule and it might take longer to complete certain sections of the project. I have limited understanding of the code that I will need for some PyGame tasks in my code and so more research is needed for these tasks, adding time onto the schedule.
- I have never tried to make a game multiplayer before and so I am prepared to run into difficulty when setting up networking and have factored in extra time in response. I also need extra time to decide the method I will use for connecting the players to a network and making the network run.
- A third potential problem I could run into is the timing of the project. If certain sections of the project over run this can have a knock-on effect which may have great consequences on the overall progress of the project. Therefore, I will need to make sure that the project stays on schedule.

## Document 5 – Gantt Chart

This section has been revised multiple time, please see comment for previous Gantt charts and timescales. For previous versions, please also refer to the folder.

#### Revised and Final Gantt Chart



(Also, a full-page copy can be found in the folder)

**Commented [MB14]:** (Refinement – 23/10, I have tried to complete some PyGame tasks before I went straight into the project, and I have added in the list of tasks to be completed before starting the project so I feel more prepared)

**Commented [MB15]:** Another initial task (no longer relevant)  
-Must do some work with records to display tables

**Commented [MB16]:** Time Scale (taken from Document 3):  
**Task**

#### Tasks:

- Research
- Proposal
- Prep. Coding
- Analysis/Planning
- Requirements Specification
- Test Plan
- Design and Implementation (Start Interface, Main Screen, Input, Game, Network, End)
- Testing
- Final Project

## Document 6 – Requirements Specification

#### Objectives:

- To create a grid-based game that works with 2D arrays to allow input and positioning of pictures in the grid in PyGame
- To make a multiplayer game so players can compete each other on different computers.
- To create a realistic and fun computerised version of the Pirate Game.

#### Simple Description of Game:

- User must fill a seven by seven grid in with various types of tiles that each correspond to different actions.
- Game begins and a 2D array is sorted based on a random set of numbers. Each round a co-ordinate is chosen from the array.
- The tile placed at that co-ordinate is played and the action of the tile is carried out.
- This is then repeated until all the tiles on the grid has been played.
- At the end of the game the winner is the person who has collected the most money both in their bank and their current amount.

#### Purpose of the solution:

- To allow users to play a computer version of the Pirate Game and compete against each other to win the game.
- The purpose of the game is to build the player's strategic thinking and allow them to develop their situational awareness.

#### Description of the Project:

##### Aim of the Game:

- To have the highest score of all the players

My project will allow users to compete in a multiplayer computer version of the Pirate Game. It will allow players to connect from different computers. A server is set up on a computer by entering the computer's IP address. Players will then have to enter the IP address of the computer which the server is set up on when they start the game. The program will load up an interface that has two options, one for a game (main) and one to view the instructions. The instructions page will display a set of instructions for the game.

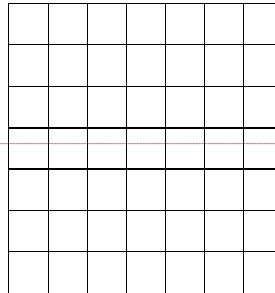
If the main game option is chosen, each player will get an interface showing a seven by seven grid like the one shown to the right.

A tile will be displayed to the user and the user will then be able enter (by the keyboard) the co-ordinates of where they want the current tile to be placed on the grid. A tile is an action icon (a square that will have an action to perform) and there will be 49 tiles for the user to place in the grid. An example of a tile is the Rob tile, which allows the player to steal money from another player.

The co-ordinates will be entered in the form(x,y) where x and y will both be single digit integers. Example: (1,4)

There will have to be input validation placed on the co-ordinates entered, to ensure the co-ordinates are within range and have not been entered before.

This input process will be looped until all the tiles have been placed on the grid. The game will then begin, and the game will alternate between each player, and each player will take their turn. There will be an order of the tiles, as some tiles must be played before others.



**Commented [MB17]:** Original plan was to get input by the mouse

#### Ordering of the tiles:

The tiles relating to money will all be played before the tiles relating to attacking (eg. Rob), using ordered IF statements. The shield and mirror are also played before action tiles.

This is to make sure that if a player has a shield that the computer is aware of this and so can shield any future attacks.

E.g.

Player 1 – Bomb

Player 2 – Rob

Player 3 – Swap

Player 4 – Double your score

Player 4 will have to go first so that if Player 2 decides to attack them, they attack them with the current amount of money.

**Commented [MB18]:** The order that tiles will be played:

- Money, Present, Double
- Bank
- Shield, Mirror
- Rob
- Kill
- Swap Scores
- Bomb

Original Ordering of tiles

**Commented [MB19R18]:** This idea was scrapped after I decided to just play money tiles before the action tiles

Each of the tiles have a different action as shown below:

**Rob** – The player chooses someone to rob and then they will receive all that person's money.

**Kill** – The player chooses someone to kill and that person's score will go back to zero.

**Present** – The player will choose someone to gift 1000 points to (this will not be taken away from the player who has the present tile's score).

**Swap Scores** – The player who has this will choose another player and these two players will swap scores.

**Choose Next Square** – This allows the user to choose one of the other actions. They have a free choice of what action they would like to perform for that round)

**Shield** – This will allow the player to prevent the action of the tile being played. (If they were robbed, they would be able to shield it from happening, so they wouldn't be robbed, and it would have no effect)

**Commented [MB20]:** Initially choose next square meant: This means the player will be able to choose the next co-ordinate that will be played.

**Commented [MB21R20]:** It was changed in order to make implementation a bit easier.

*Mirror* – Similar to shield but the action will be performed on the player that sent the initial action. (If player 1 tries to rob Player 2 but Player 2 plays a mirror, then Player 1's score goes back to zero)

*Bomb* – The player who has this tile score's will go back to zero.

*Double Your Score* – As described

*Bank* – This protects the current money by putting it in the bank. Any money in the bank cannot be affected by any tiles and will be added to current at the end of the game.

*Money Squares* – There is many money squares. The money in these squares will be added to a running total of current money which will be displayed on the screen and this is known as the current money. If the player is attacked, then this is the money that is taken.

The game will run through randomly choosing a next square. There is a 2D list of all the possible co-ordinates generated at the start of the code. Each time the game starts random numbers all allocated to each set of co-ordinates using the random library and then the list is sorted using a bubble sort to give a randomly sorted list of co-ordinates to run through.

At the end of the game, the current money and the bank money will be added up for each player. This will be their total. The player with the highest total is the winner.

#### **End Users:**

This game is aimed at high school age children between the ages of 12 and 18. It is a game aimed to be used at the end of school terms as a recreational activity. The end users will most likely have intermediate computer skills and will be able to use the game without any additional learning materials. End users will most likely be students, but the game would also appeal to adults.

#### **End User Requirements:**

- Navigate through the game with ease
- Can access rules for the game at the start
- Can enter the co-ordinates for the tiles on their grid
- Can choose the player they want to attack in relation to the current tile
- Can choose which action they want to perform as part of the wild card tile
- Have access to their score and the winners score at the end of the game
- Can play with other people on different computers
- Want to play a game as close as it can possibly be to the original

#### **Functional Requirements:**

- The program should allow users to go between the instructions interface and the start interface.
- The program should allow users to start the game from the start interface.
- The instruction interface should allow the users to read the rules of the game.
- The program needs to sort a 2D array filled with all the possible co-ordinates to allow the game to generate random co-ordinates throughout the game.
- The program needs to be able to add money to a running total of the money collected by the player.

**Commented [MB22]:** This will be generated by the random function and multiplying a randomly chosen integer by 100 to get the X co-ordinate and then repeating for the Y co-ordinate. (Initial Idea)

**Commented [MB23R22]:** Refinement on the 8<sup>th</sup> of February 2018



- The program needs to be able to bank player's money when they have a bank square.
- The program needs to display an interface allowing players to choose who they want to attack when they have an action tile.
- The program needs to display an interface allowing players to choose which action they want to perform as part of the wild card tile.
- The program needs to display an interface summarising the actions performed against the player each round.
- The program needs to add the current money and the money that is banked to the total money at the end of the game.
- The program needs to determine the winner, by using a finding the maximum algorithm.
- The program needs to display an interface to the runners up, telling them their score and the score of the winner.
- The program needs to display an interface to the winner, informing them they have won the game.
- The program needs to be able to perform validation on users' inputs of the tile position to ensure it is a whole single number. (e.g. 1, 2).
- The program needs to be able to perform validation on users' inputs to ensure the user cannot enter the same co-ordinate twice.
- The program needs to be able to let the players each have their turns in order of the tiles. (See figure 1)

#### **Inputs and Outputs:**

##### *Inputs:*

- User must input (by keyboard) which screen they would like to go onto
- User must input (by keyboard) the position of the tiles on the grid
- User must input (by button) which player they want to perform the action on chosen tiles in the main game.
- User must input (by button) which action they want to perform when they get the choose next square tile.
- Users must input (by keyboard) any key to continue with the game after input and to exit the game at the end.

(The program will receive input from the keyboard and the mouse)

##### *Outputs:*

- The interfaces will each be displayed respectively when the keys for their interfaces are pressed (example: main game, instruction interface).
- The winner (the person with the highest score) and runner ups will be outputted at the end of the game, to show the order of the players.
- The winner needs to see a separate winning screen from the runner ups, which shows only that they won and not the other scores.
- The program needs to output to the players what tile is at the co-ordinate randomly generated each round and the action description of that tile.
- The program needs to output an interface to allow the user to choose between who they want to attack if they get an action tile.

- The program needs to output an interface to allow the user to choose what action they want to perform as part of the wild card.
- The program needs to output an interface to allow the user to see which actions have been performed against them each round.
- The program needs to output a game beginning screen at the start of the game.

(The program will display output in the PyGame window)

For my 'Record of Progress' I created a blog, there is a link to it below:

<https://blogs.glowscotland.org.uk/fi/ahproject/>

#### Timetable:

The time span for the project is the specific length it is because it started once I had developed enough skills in PyGame to progress on my project and the ending date of Mid-March is the date specified by my teacher by the date I must finish my project.

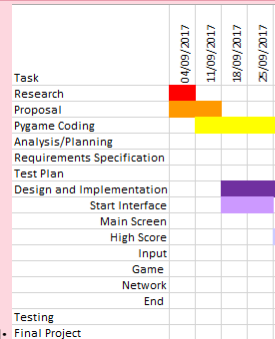
#### Resources:

- Desktop Computer
- Python Version 3.4
- PyGame library
- Networking library

#### Scope and Boundaries of the Project:

Scope:

Functionality	Description
Connecting to a server	Users will enter an IP address and connect to a server, allowing them to start input.
Number of players	The game is played with 3 players. The game cannot be started unless 3 players have connected and started their input.
Input	Players will enter tile co-ordinates through the keyboard.
Grid display (active)	The grid will highlight the tile green when in play.
Grid display (finished)	The grid will show an icon with a red cross across it when the tile has been used.
Choose who to attack	The game will allow users to choose which player to attack by a button interface.
Choose which action to play	As part of the wild card, the game will allow users to choose which action they want to perform.



#### Commented [MB24]:

*Research* – Researching into the fundamentals of the code that I want to do and doing some supplementary coding to prepare for the project  
*Proposal* – Writing up my project proposal  
*PyGame coding* – Practice coding in PyGame  
*Analysis/Planning* – Completing required write ups  
**(Design and Implementation)**  
*Main Interface* – design and implement an interface that will be used throughout the main game and will have the features to expand the interface to accept input  
*Start Interface* – design and implement an interface for the start of the game when the player is positioning their tiles  
*High Score Interface* – design and implement an interface that displays the name and score of the top 5 players of all time in the pirate game  
*Networking* – allow the game to become multiplayer by connecting it up to a network  
*Input* – allow the program to accept input in PyGame and display text on the screen  
*Game* – allow for each tile to perform its action when it's position is chosen in the grid  
*End* – display the order of players at the end of game and quit the game  
*Testing* – As specified, following a written test plan  
*Final Project* – Finish off and complete any necessary write ups and analysis

**Commented [MB25R24]:** This was the initial gantt chart

**Commented [MB26]:** This is the last possible date to hand the project in, the end of the easter holidays (16/4/18)

Prioritising actions	The server will perform money and shield/mirror actions before any of the other action tiles.
Shield/Mirror	The game will allow the user to shield/mirror any actions when they have that tile.
Rob	The game will allow the user to rob another player of their choice when they have the rob tile.
Kill	The game will allow the user to kill another player of their choice when they have the kill tile.
Mag.glass	The game will allow the user to see another player of their choice's score when they have the magnifying glass tile.
Present	The game will allow the user with the present tile to gift another player of their choice 1000 points.
Swap	The game will allow the user to swap scores with a player of their choice when they have that tile.
Choose Next Square	The game will allow the user to choose the action they want to perform for that round.
Double	The game will allow the user with the double tile to double their current total.
Winner Screen	The program will display an interface congratulating the winner at the end of the game.
Runner up screen	The program will display an interface informing the runner ups their score and the winners score.

#### Boundaries:

- This game will be multiplayer but will only let three players play at one time. There must be three players to start the game.
- The game will allow input of grid co-ordinates but only through keyboard, not dragging the tiles onto the grid.
- The game will not contain any animations or music.
- The game will not contain a high score table.
- The game will not allow users to enter their own names for players, they will be allocated a player number based on when they connected to the server in relation to other players.

## Document 7 – Test Plan

To test my program, I will use a mixture of

- Component Testing (testing each component separately)
- Modular Testing (testing modules together)
- Systems Testing (testing the whole system)
- Usability Testing (testing with beta testers)

Due to the nature of my program I will be focusing more on the component and modular side of testing, but I will ensure that the system is fully tested as well.

I will have to test all features of my program and I have plans to create smaller grid prototypes to help me out testing some of the actions. I will use a mix of extreme, accepted and exceptional test data.

I will also carry out Usability testing. I will give my program to a wide range of end users to get their opinion and see how it stands up outside the normal testing environment.

Planned Component Tests:

Interfaces

Purposes:

- Interface testing will allow me to check the interfaces display the right information and also respond correctly to key presses
- Interface testing will also allow me to check they are responding correctly to progression in the game.

Feature to be tested	Expected Result	Stage of development when tested
Home interface	Displays an interface allowing the user to move between rules and game screens, and allowing the user to start the game	Before integration and after integration to full system
Instructions button	Displays an interface explaining the rules of the game	Before integration and after integration to full system
Game button	Takes the user to the start of the game	After integration to full system
Input interface	Allows users to enter co-ordinates for positioning of tiles on the grid	Before integration and after integration to full system
Game beginning screen	Displays a short message stating 'Game beginning' to inform players of start of game	Throughout development of 2 by 2 prototype
Summary Screen	Displays a summary of the actions being performed in the round against them and any use of mirror/shield	Throughout development of 7 by 7 game

Winner Interface	Displays a screen at the end telling the user they have won	Throughout development of 7 by 7 game
Runner up Interface	Displays a screen at the end showing the user their score and the winners score	Throughout development of 7 by 7 game
Grid Interface	Displays a 7 by 7 grid with a) Position of current tile that was entered b) All the tiles placed in the grid c) The current tile d) Used tiles	Before integration into 2 by 2, throughout development of 2 by 2 and 7 by 7
Active Tiles	Shows the green version of the tile when in play.	Throughout development of 7 by 7
Finished Tiles	Shows the cross version of the tile when the tile has been used.	Throughout development of 7 by 7
Normal State Tiles	Shows the normal version of the tile when not been used/ not in play.	Throughout development of 7 by 7

## Main Game

### Purposes:

- Input Validation testing will allow me to check users cannot enter invalid co-ordinates, which would mess up the progress of the game.
- Another purpose of main game testing also allows me to check the tiles are performed at the right time and in the right order and they are having the desired effect on the person that the user chose.
- I will also be able to assess if the bank is added, to ensure that the final scores are based on the user's total money, not just their current amount.

Feature to be tested	Expected Result	Test Data (if required)	Stage of development when tested
Input Validation (between 1 and 7)	Will not allow input of numbers greater than 7 and less than 1. Will accept two-digit numbers, only taking the first digit entered	2,4 (Accepted) 5,8 (Exceptional) 1,7 (Extreme) 7,7 (Extreme) 42, 56 (Accepted)	Start of development of 2 by 2 and end of development of 7 by 7
Input Validation 2 (not entered before)	Will not allow the user to enter the same co-ordinate twice	2,5 (Accepted) 2,5 (Exceptional) 3,8 (Exceptional) 3,5 (Accepted)	During development of 2 by 2 and end of development of 7 by 7

Takes first number of input	Will take the first digit of the input	34, 678 (Accepted) 82, 32 (Exceptional)	Start of development of 2 by 2
Performs the money actions between the rest of the actions	Will carry out the money tiles before any of the other action tiles.		During development of 2 by 2
Rob	Will give the player robbing the current score of the player they attack.		During development of 2 by 2 and end of 7 by 7
Kill	Will send the player who is attacked current score to zero.		During development of 2 by 2 and end of 7 by 7
Swap	Will swap current scores of the attacker and the player they chose.		During development of 2 by 2 and end of 7 by 7
Bomb	Will send the player who has this tile's current score back to zero.		During development of 2 by 2 and end of 7 by 7
Double	Will double the current score of the player who has this tile.		During development of 2 by 2 and end of 7 by 7
Present	Will give 1000 points (not out of the 'attacker's' current score) to the player chosen by the 'attacker'		During development of 2 by 2 and end of 7 by 7
Mag.Glass	Will show the score of the player chosen by the attacker.		During development of 2 by 2 and end of 7 by 7
Mirror	Will allow all the actions to be mirrored for that round.		During development of 2 by 2 and end of 7 by 7
Shield	Will allow all the actions to be blocked for that round.		During development of 2 by 2 and end of 7 by 7
Wild Card	Will allow the user to choose an action to perform for that round.		During development of 2 by 2 and end of 7 by 7

Wild Card (pt 2)	Performs the action chosen by the user.		During development of 2 by 2 and end of 7 by 7
5000	Adds 5000 to the users current score.		During development of 2 by 2 and end of 7 by 7
3000	Add 3000 to the user's current score.		During development of 2 by 2 and end of 7 by 7
1000	Add 1000 to the user's current score.		During development of 2 by 2 and end of 7 by 7
200	Add 200 to the user's current score.		During development of 2 by 2 and end of 7 by 7
Progresses through rounds correctly	Goes to Round 2 once Round 1 has finished etc		During development of 2 by 2 and end of 7 by 7
Waits for each player to get their shot	Allows the users to progress one at a time through their actions.		During development of 2 by 2 and end of 7 by 7
Add banks to the total of the player at the end	Adds the bank score for that player to their current score at the end of the game.		During development of 2 by 2 and end of 7 by 7

#### Server

##### Purpose:

- To ensure the server lets people connect and is informing all the clients of any updates with the rounds/money totals.
- This testing will help me have confidence that the server is correctly communicating between all the clients and that it waits for all the players to finish their input before starting the game.

Feature to be tested	Expected Result	Test Data (If required)	Stage of development when tested
Connects to the server when correct IP address is entered	Adds one to the total players value each time someone joins		During development of 7 by 7

			(different method used in 2 by 2 prototype)
Communicates between the clients	The server should be able to send data to the clients, and to specific clients		During development of 2 by 2 prototype
Updates the score after a round	The server sends the updated scores to the clients when an action is performed, so the screen shows the updated score.		During development of 2 by 2 prototype and 7 by 7 game
Starts the game when 3 players have connected	Waits until 3 players have finished their input before starting the game.		During development of 2 by 2 prototype and 7 by 7 game
Waits for one player to choose someone for an action before asking another player	Only lets one person at a time choose a response to an action		During development of 2 by 2 prototype and throughout development of 7 by 7 game.

### Modular Testing

There will be four main modules that I will test that they work independently

#### Input

- Check that the input interface is displayed at the right time in the game
- Check that the input validation works on the co-ordinates and names
- Check that the input produces the right effect, and places the tile at the co-ordinate entered

Main Game (cannot test this independently of input but will make sure it works on its own once input is fully tested)

- Check the tiles perform the right actions
- Check that the mirror and shield squares trigger an image to be displayed in the mirror and shield boxes
- Check that once a co-ordinate is chosen, then the program proceeds playing the tiles in the specified order



- Check that the bank total cannot be edited by any actions and the current money will be changed instead
- Check that the game will loop through until all tiles have been played
- Check that the choose next square tile will let the player choose the next co-ordinate and the tiles from that co-ordinate are played

## Interfaces

- Check that all the key presses respond correctly and display the right interface
- Check that pressing over keys does not perform any actions
- Check the instruction interface shows a step to step guide of how to play the pirate game
- Check the game interface (once g key is pressed) will start a game

## Network

- Check that the game allows multiple people to play the game
- Check the game allows people to play together if they enter the right server IP address
- Check the game will let players take turns and will give all players connected a turn

## System Testing

This will be carried out before the end user testing. I need to individually test that three users can play fully through the game and check all the modules and the components work.

I will just play through the game fully and check that all features work.

I will need to check the system works by typing in invalid input and 'trying to break' the program.

I will need to make sure that it works for a varied combination of positions of tiles.

## Recording Test Solutions

It will depend on the context of the testing on how I will record test data. For most testing I will record screenshots using snipping tool and add them into the testing document alongside the expected results and what I was testing for. I will also have to check the console output from the server and the clients for proof that the actions have been performed correctly.

## Examples of different Scenarios:

I will have to check the interfaces to see if they display the right content and screenshot this result

**Commented [MB27]:** Check that the high score interface shows the top 5 scores of all time imported from a CSV file

Original part of testing

**Commented [MB28]:** Original planned scenario: I will have to check the CSV file to see if it has been updated and screenshot this result

I will have to check that the tiles produce the right effect. It will be harder to record this, but I will use the usual output from server and clients.

#### Input Validation Testing

Part of the program	Accepted Input	Reason for Input Validation	Test Data
Tile Position Input	Any two whole numbers in the between 1-7 in the form (x,y) Example: (6,5)	Must be co-ordinates in the 7 by 7 table	2,4 (Accepted) 5,8 (Exceptional) 1,7 (Extreme) 7,7 (Extreme) 42, 56 (Accepted)
Tile Position Input 2	Any two whole numbers that have not been entered before	Cannot place two tiles in the same co-ordinate	2,5 (Accepted) 2,5 (Exceptional) 3,8 (Exceptional) 3,5 (Accepted)

#### Test Plan for End User Testing

I will get various potential users to test my game, running through it themselves, placing their tiles in co-ordinates they choose themselves. I will then let them run through the game send to a link to a questionnaire at the end to find out how they found the system and the game.

<https://www.surveymonkey.co.uk/r/Q7T2LM7>

The questionnaire will be as follows:

The screenshot shows a 'Usability Survey' form. At the top, there is a blue header with the text 'Usability Survey' and a 'PAGE TITLE' field. Below the header, there are three questions, each followed by a rating scale represented by five stars. The first question is '1. How easy did you find it to play the pirate game?' with labels 'very hard', 'difficult', 'moderate', 'easy', and 'Very easy' above the stars. The second question is '2. How did you find the input of the co-ordinates?' with labels 'very hard', 'difficult', 'moderate', 'easy', and 'very easy' above the stars. The third question is '3. How did you find the user interfaces overall?' with labels 'horrible', 'not so nice', 'neutral', 'i liked them', and 'aesthetically pleasing' above the stars.

4. Did you enjoy the game?

- ☐ Yes  
☐ No  
☐ Neutral

5. Please comment possible improvements

6. Please state your favourite feature

7. Please state the feature you didn't like the most and why?

8. Did you find it easy to keep up with the progress of the game?

0 ease 100 ☐

9. Did you understand the rules?

- ☐ Yes  
☐ No  
☐ Not applicable, did not read them

10. Have you played the pirate game before?

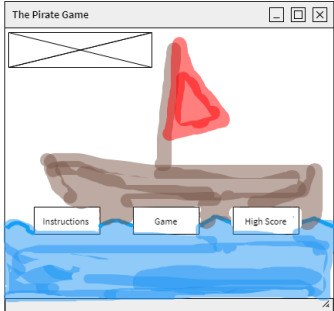


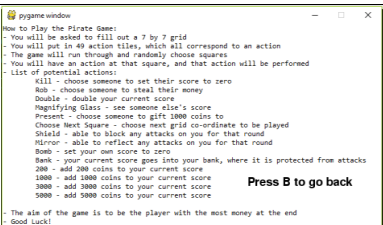
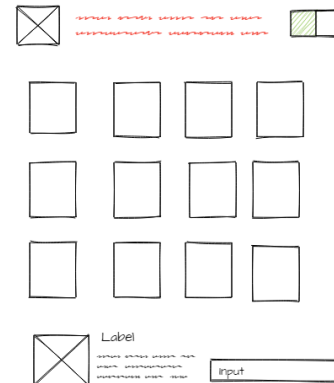
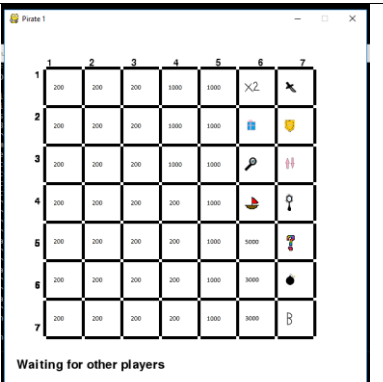
- ☐ Yes  
☐ No

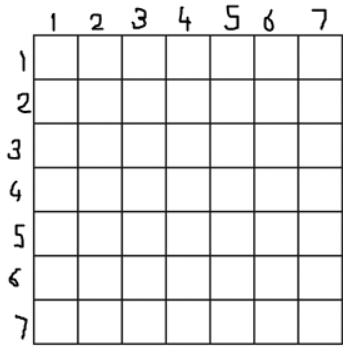
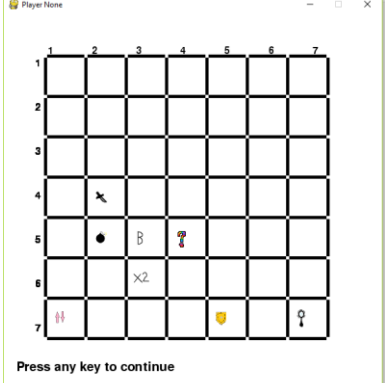
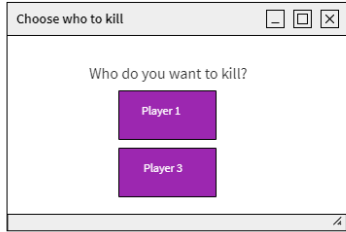
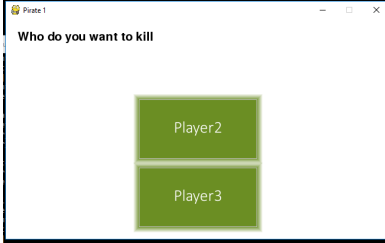
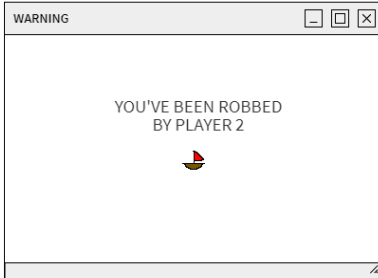
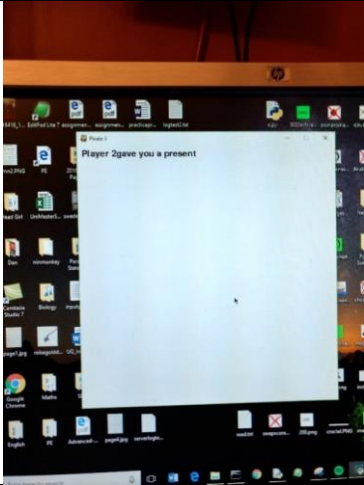
## Document 8 – Interface Design

Throughout the project I changed the design of multiple interfaces. To see more in-depth and iterative designs please see the '*Interfaces*' section in the main folder. This contains annotated drawings/sketches of all the UI components. This document will only contain the first thoughts on the interface and the final product, as well as the purpose of all the functions of the interfaces, the inputs, outputs and validation criteria, the interrelation of the interfaces and the justifications of any adaptations to the interfaces.

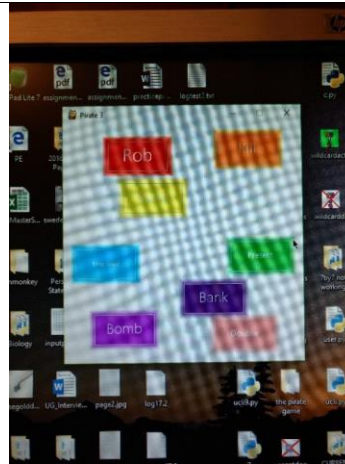
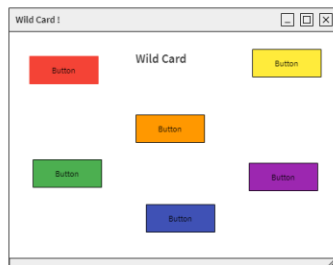
## Initial and Final Interface Designs:

### Main Game

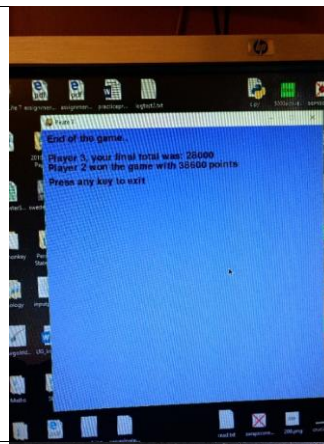
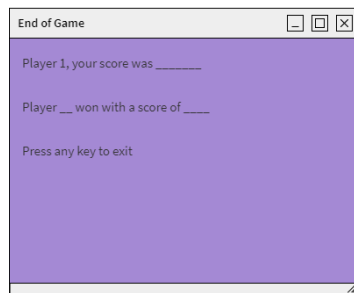
Interface	Original Wireframe	Actual Interface
Intro screen		
Instructions		
Game		

Grid		
Choose Player		
Summary Screen		

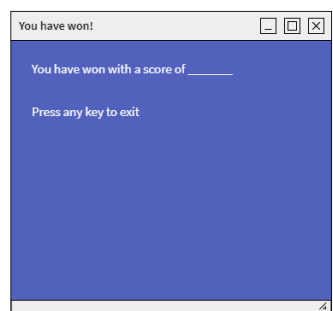
## Wild Card



## End Screen

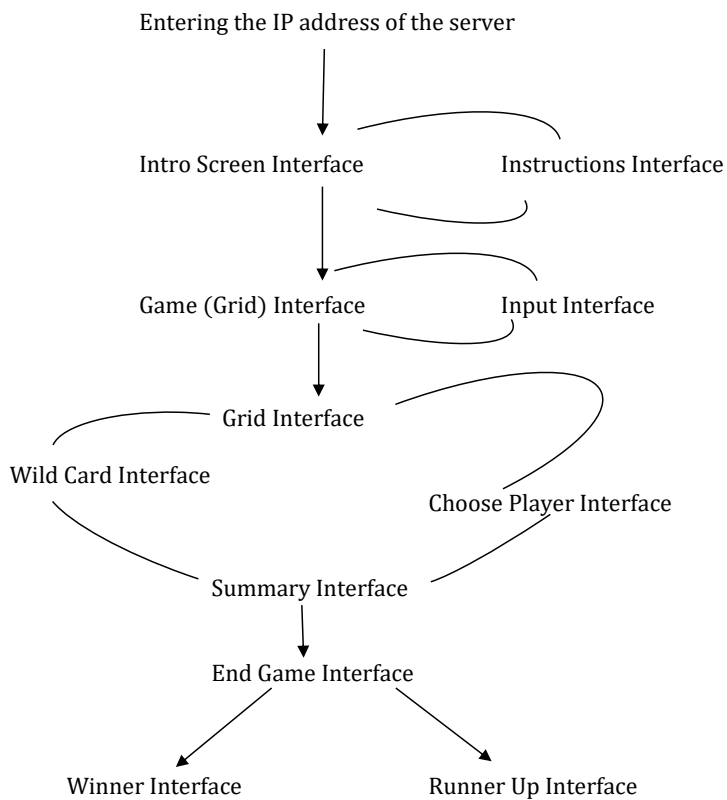


## Winner Screen



Movement through the interfaces

(Please see P36-40 in pink book and folder for more details)



#### Necessary Validation

- Input Validation 1

One of the necessary validations is to make sure that the co-ordinates the user enters are within the grid, and they are a whole number between 1 and 7. This is necessary to ensure that the user cannot place tiles at co-ordinates that are off the grid. This is client side validation and occurs in the inputvalidation subroutine on line 320 on the client code. It will ask the user to re-enter co-ordinates if either the x or y co-ordinate is outwidth the range.

Code:

```

# ----- INPUT VALIDATION 1 -----
#Checks the number is between 1 and 7 to fit on the grid
#Required to ensure that the client doesn't enter grid co-ordinates that aren't available
def inputvalidation(coord):
    #Turning into an integer in order to carry out comparisons
    coord = int(coord)
    while coord == 0 or coord>7:
        #Setting the screen to the correct screen size
        screen = pygame.display.set_mode(size)
        #Setting up the clock in order to put in a delay later
        clock = pygame.time.Clock()
        #Setting up the font
        basicfont = pygame.font.SysFont(None, 25)
        #Setting up the text to be blitted to screen
        #Error message
        text = basicfont.render('Invalid Input, please enter a number between 1 and 7', True, (white), (black))
        textrect = text.get_rect()
        #Making sure the text is displayed in the middle of the screen
        textrect.centerx = screen.get_rect().centerx
        textrect.centery = screen.get_rect().centery
        #filling the screen in black
        screen.fill(black)
        #displaying the input validation message to the screen
        screen.blit(text, textrect)
        #updating the screen
        pygame.display.update()
        #including a delay in the program
        pygame.time.wait(500)
        #fills the screen in black
        screen.fill(black)
        #asks the user again to enter an x co-ordinate again
        coord = (ask(screen, "Co-ord"))
        #turns it into an integer
        coord = int(coord)
    #returns co-ordinate to pass it back to the main subroutine
    return coord

```

#### - Input Validation 2

A second but also equally necessary type of validation is to ensure that the user cannot enter the same co-ordinate twice. You cannot place two tiles on the same spot in the grid so the input validation must ensure they are entering a unique co-ordinate. This is again client side validation and occurs in the inputval2 subroutine on line 358 of the client code. It will ask the user to again enter a co-ordinate if the co-ordinate they entered was not unique. It is also linked to input validation 1 to ensure the unique co-ordinate the user is entering is also valid.

Pseudocode:

*Def inputval2* (Parameters = arrayofcoords, xcoord, ycoord and valid)

Set a flag to false, only setting to true if the co-ordinate has already been entered

Loop round for the number of co-ordinates entered

    If the value for xcoord is equal to the xcoord in the array at that position  
    AND the value for ycoord is equal to the ycoord in the array at that  
    position

        Set alreadyhere to True

        Break out of the loop

    End if

End for

If alreadyhere is False

    Set valid to True

End if

Return valid



Code:

```
# ----- VALIDATION OF THE INPUTS PART 2 -----
#Compare the two values in the X and Y variables to all the other duples in the arrays
#Need to make sure that they are not entering the same numbers
#Cannot have two pictures in the same square
def inputval2(arrayofcoords,xcoord,ycoord,valid):
    #setting the boolean variable to false, only set to true when found in array
    alreadyhere = False
    #looping round for the whole length of the first array in the 2D array
    for j in range(len(arrayofcoords[0])):
        #compares the values for both X and Y
        if xcoord == arrayofcoords[0][j] and ycoord == arrayofcoords[1][j]:
            #set boolean value to true because the co-ordinate is already there
            alreadyhere = True
            #breaks out of the loop
            break
    #If the value hasn't been found in the array then it is a valid input
    if alreadyhere == False:
        valid = True
    #returning whether or not the input
    return valid
```

- Validation in the actual input  
This validation ensures that the user cannot enter an empty co-ordinate/no co-ordinate.  
Code:

```
while 1:
    #get the key pressed
    inkey = get_key()
    if inkey == K_BACKSPACE:
        #If a backspace is entered, removes
        the last string from array
        current_string = current_string[0:-
1]
    elif inkey == K_RETURN:
        if len(current_string) != 0:
            #breaking out of the loop if something has been entered
            break
        else:
            pass
    elif inkey <= 127:
        #Appending the array with the character
        current_string.append(chr(inkey))
```

Doesn't let you enter the co-ordinate if it is empty

This just makes sure the program does not crash, as if there was no input it could not take the integer of nothing, so the game crashes. This was not initially a planned validation but testing it made me aware that it was necessary.

#### Code:

```
#This subroutine actually displays the window asking for input
def ask(screen, question):
    #Initialising the font for the input box
    pygame.font.init()
    #This will be an array to take in the current characters
    current_string = []

    #Joining the current string to a blank space as required by python 3
    #calls the display box subroutine to ask the question
    display_box(screen, question + ": " + "".join(current_string))
    #forever
    while 1:
        #get the key pressed
        inkey = get_key()
        if inkey == K_BACKSPACE:
            #If a backspace is entered, removes the last string from array
            current_string = current_string[0:-1]
        elif inkey == K_RETURN:
            if len(current_string) != 0:
                #breaking out of the loop if something has been entered
                break
            else:
                pass
        elif inkey <= 127:
            #Appending the array with the character
            current_string.append(chr(inkey))
        elif inkey > 127:
            #take one off the array
            current_string = current_string[0:-1]
        #Joining the current string
        testing = question + ": " + "".join(current_string)
        #calls the display box subroutine to ask the question
        display_box(screen, testing)
    #returning the values so that they can be used in the next subroutine, joined to a blank string space
    return "".join(current_string[0:1])
```

#### Inputs and Outputs

##### Inputs:

- User will have to input r if they want to access the rules interface(keyboard).
- User will have to input g if they want to start the game and begin input(keyboard).
- User will have to input all the co-ordinates of where they want to place the tiles on the grid(keyboard).
- User will have to input who they want to perform their current action against(buttons).
- User will have to input which action they want to perform as part of the wildcard tile(button).
- User will have to input any key to exit the final screen(keyboard).

##### Outputs:

- The program will display a start interface with options to go to the instructions interface or start the game.
- The program will display an instructions interface (if the user presses r) describing how to play the pirate game.
- The program will display which tile that the user must enter co-ordinates for.
- The program will display what the user has typed in, whilst entering co-ordinates.
- The program will display the grid with the users tiles on it at multiple points throughout the game.

- The program will display the players current score and what round it is currently on the grid display throughout the main game.
- The program will highlight which action tile is currently being played during each round.
- The program will display an interface asking for which player the user wants to attack based on their current tile.
- The program will display an interface displaying all the options the player can choose when they have a wild card.
- The program will display an interface to the winner, showing a congratulatory message and a message on how to exit the game.
- The program will display an interface to the runner ups, showing their score, the winners score and a message on how to exit the game.

#### Purposes and functions of screen elements

##### Intro Screen

Function	Purpose
Press R for rules	To allow the user to go to the rules screen
Press G for game	To allow the user to start the game

##### Rules Screen

Function	Purpose
Press B to go back	To allow the user to go back to the intro screen after reading the rules

##### Input

Function	Purpose
Image of the tile	To show which tile the user is entering the co-ordinate for
Description of the tile	To describe to the user the function of the tile
Input Box	To allow the user to see what they have entered
Press enter to submit co-ordinate	To allow the user to enter their co-ordinate
Input Validation 1	To ensure the co-ordinate entered is between 1 and 7
Input Validation 2	To ensure the co-ordinate entered is a unique co-ordinate

##### Grid Display

Function	Purpose
Grid Numbers	To allow the user to see where they have placed their co-ordinates
Image of the tile	To show the user which tile is where

Variable message	To allow the user to receive any necessary input about why they are seeing the grid
Active Tiles	To allow the user to see what the current tile is
Finished Tiles	To allow the user to see what tiles have been used
Round/Score (only in game)	To allow the user to see what their current score is and what round they are on

#### Choose Player/Wild Card

Function	Purpose
Variable message	To allow the user to see what they are having a choice about
Buttons	To allow the user to choose which player they want to perform the action against

#### Summary Screen

Function	Purpose
Variable messages	To allow the user to see who has attacked them/ what has happened to them within the round

#### End Screens

Function	Purpose
Your score(both)	To allow the user to see their score
The winners score (runner up)	To allow the user to see how far they were away from the winner
Congratulations message (winner)	To congratulate the winner
Press any key to exit	To allow the user to exit the game after completion

**Fully Annotated Versions and previous versions of the user interfaces can be found in the supporting folder, along with a further breakdown of the connection between interfaces.**

Justification of decisions made in relation to End Users:

The user interfaces were all designed with the end users in mind. I tried to make the interfaces as easy to read and understand as possible. I received feedback from potential users about the interfaces throughout the processes. This allowed me to modify them continually and make the clearest and usable interfaces as possible.

Examples:

- Originally in the 2x2 prototype you just entered your co-ordinates and then at the end of input you saw your grid and where the tiles were placed. When I was scaling the game up I realised there was no chance of the user being able to remember the co-ordinates they have entered without seeing the screen and so edited the code, so it displayed the grid to the user at the end of each of the inputs of the individual co-ordinates. Every time the user entered the co-ordinate the user will get to see the grid.
- When I first gave the program to the end users to test out, I had summary screens built in to summarise what had happened each round. If any of the users had just got money and not been attacked, they would have just seen a screen stating, 'No actions have been performed against you'. Due to the speed of the game when all the users get money this subsequently meant that the users did not see their score for very long and struggled to know what was happen, so I took this feedback on board and made sure that no actions had been performed against the user, they got to see their grid with the updated total, which made the game easier to follow.
- Early in the development process the grid did not have numbers at the side. As soon as it was tested for the first time, potential end users stated that having grid numbers was a necessary and so they were soon implemented.

## Document 9 – Data and Program Structure

*Pseudocode (This is the pseudocode for my whole program)*

Client Pseudocode

Import all the necessary libraries

Import all from PyGame.locals

Import all from PyGame

Import ConnectionListener and connection from PodSixNet.connection

Import time from sleep

Import random from randint

Set up the dir\_path to the directory path of the current file

Set up cwd as the current working directory

Change the current working directory to the directory path of the current file

Initialize PyGame

Initialize PyGame font

Set up white as [255,255,255]

Set up black as [0,0,0]

```

Set up size to be [600,600]
Set up gameDisplay2 to be a screen of size 550,350
Set up an array descrp storing the descriptions of all the tiles
Let background = PyGame.image.load('introscreen.png')
Let instruct = PyGame.image.load('instructions.png')
Let grid = PyGame.image.load('square.png')
Let logo = PyGame.image.load('logo.png')
Let twohundred = PyGame.image.load('200.png')
Let thousand = PyGame.image.load('1000.png')
Let threethousand = PyGame.image.load('3000.png')
Let fivethousand = PyGame.image.load('5000.png')
Let bank = PyGame.image.load('bank.png')
Let bomb = PyGame.image.load('bomb.png')
Let double = PyGame.image.load('double.png')
Let magglass = PyGame.image.load('magnifinghglassreal.png')
Let present = PyGame.image.load('present.png')
Let rob = PyGame.image.load('rob.png')
Let shield = PyGame.image.load('shield.png')
Let swap = PyGame.image.load('swapscores.png')
Let kill = PyGame.image.load('kill.png')
Let mirror = PyGame.image.load('mirror.png')
Let wild = PyGame.image.load('wildcard.png')
Let instruct = PyGame.image.load('instructions.png')
Let logo = PyGame.image.load('logo.png')
Let normallinev=PyGame.image.load("normalline.png")
Let normallineh=PyGame.transform.rotate(PyGame.image.load("normalline.png"), -90)
Let twohundreda = PyGame.image.load('200active.png')
Let thousanda = PyGame.image.load('1000active.png')
Let threethousanda = PyGame.image.load('3000active.png')
Let fivethousanda = PyGame.image.load('5000active.png')
Let banka = PyGame.image.load('bankactive.png')

```

```

Let bomba = PyGame.image.load('bombactive.png')
Let doublea = PyGame.image.load('doubleactive.png')
Let magglassa = PyGame.image.load('magnifinghglassrealactive.png')
Let presenta = PyGame.image.load('presentactive.png')
Let roba = PyGame.image.load('robactive.png')
Let shielda = PyGame.image.load('shieldactive.png')
Let swapa = PyGame.image.load('swapscoresactive.png')
Let killa = PyGame.image.load('killactive.png')
Let mirrora = PyGame.image.load('mirroractive.png')
Let wilda = PyGame.image.load('wildcardactive.png')
Let twohundredd = PyGame.image.load('200done.png')
Let thousandd = PyGame.image.load('1000done.png')
Let threethousandd = PyGame.image.load('3000done.png')
Let fivethousandd = PyGame.image.load('5000done.png')
Let bankd = PyGame.image.load('bankdone.png')
Let bombd = PyGame.image.load('bombdone.png')
Let doubled = PyGame.image.load('doubledone.png')
Let magglassd = PyGame.image.load('magnifinghglassrealdone.png')
Let presentd = PyGame.image.load('presentdone.png')
Let robd = PyGame.image.load('robdone.png')
Let shieldd = PyGame.image.load('shielddone.png')
Let swapd = PyGame.image.load('swapscoresdone.png')
Let killd = PyGame.image.load('killdone.png')
Let mirrord = PyGame.image.load('mirrordone.png')
Let wildd = PyGame.image.load('wildcarddone.png')

Set up a tiles 2D array storing all the normal state images in the first array, all the active
state images in the second array and all the done state images in the third

Set up a tilesdescr array storing the one word description of the tiles

Set up an empty 2D array called money with three arrays

Set introdone to False

Append fivethousand to the money[0] array
Append fivethousanda to the money[1] array

```

Append fivethousandd to the money[2] array

Loop twice

Append threethousand to the money[0] array

Append threethousanda to the money[1] array

Append threethousanddd to the money[2] array

End loop

Loop ten times

Append thousand to the money[0] array

Append thousanda to the money[1] array

Append thousanddd to the money[2] array

End loop

Loop twenty five times

Append twohundred to the money[0] array

Append twohundreda to the money[1] array

Append twohundredd to the money[2] array

End loop

Set up an empty array called moneydescrip

Append '5000' to the moneydescrip array

Loop twice

Append '3000' to the moneydescrip array

End loop

Loop ten times

Append '1000' to the moneydescrip array

End loop

Loop twenty five times

Append '200' to the moneydescrip array

End loop

Set up n as 200

Set up screen\_w, screen\_h to be 640,480

**WINNER SCREEN**



```

Def MainWinner
Initialize PyGame
Set screen to be a screen of width screen_w and height screen_h
Set fontobject to be PyGame.font.Font(None,30)
Set the caption of the window to be 'You are the winner'
Set the background to equal PyGameSurface(screen.get_size())
Set the background to background.convert
Create N randomly placed stars
Set clock to be PyGame.time.Clock()
Whilst 1
Set the clock.tick to 22
For event in PyGame.event.get
If event.type is equal to Quit
Quit
Else if event.type is equal to keydown
Quit
End if
End for
Fill the background in blue
Draw stars to the screen
If star[0] is less than 0
Set star[0] to the width of the screen
Set star[1] to a random integer between 0 and screen height
Blit the background to the screen
Blit a congratulatory message to the screen
Blit 'Press any key to exit' to the screen
Flip the display

Def Game_Start
Set up screen to be a screen of 550 by 450
Fill the screen in white

```

```
Set up fontobject to be PyGame.font.Font(None,32)
(Parameters = arrayofcoords, xcoord, ycoord, valid)
Set alreadyhere to False
For j in range length of arrayofcoords[0]
If xcoord is equal to arrayofcoords[0][j] and ycoord is equal to arrayofcoords[1][j]
Set alreadyhere to True
Break out of the loop
End if
End for
If alreadyhere is False
Set valid to True
Return valid
```

#### **MAG.GLASS SHOW TO THE SCREEN**

```
Def show(Parameters = self,chosen,points)
Set screen to a display of size size
Fill the screen in white
Set basicfont to PyGame.font.SysFont(None,24)
Set text to a message displaying the string of chosen and the string of points
Blit the text to the screen
Flip the display
Set a time wait of 2000 milliseconds
```

#### **INPUT**

```
Def display_box(Parameters = screen, message)
Set fontobject to PyGame.font.Font(None,18)
Draw a rectangle in the middle of the screen in white
Draw a rectangle in the middle of the screen in black
If the length of the message is not 0
Blit the message to the screen in the box
Flip the display
```

```
Def ask(Parameters = screen, question)
```

```

Initialize the font
Set current_string to an empty array
Call display_box passing in screen, question + ":" + "".join(current_string)
While 1
    Set inkey to get_key()
    If the key pressed is backspace
        Take one off the current_string array
    End if
    Else if the key pressed is return
        If the length of the current_string array is not zero
            Break out of the loop
        Else
            End if the inkey is less than or equal to 127
        Append the key pressed to the current_string array
        Else if inkey is greater than 127
            Take on off the current_string array
        End if
        Set testing to question + ": " + "".join(current_string)
        Call display_box, passing in screen and testing
        Return "".join(current_string[0:1])

```

#### **DISPLAYING THE TILE TO BE PLACED ON GRID**

```

Def firstdisplay(Parameters = tiles, descrp, i)
Set screen to a screen of size 300 by 300
Set currenttile to tiles[0][i]
Blit the currenttile to the screen
Flip the display
Initialize the font
Set myfont to PyGame.font.Font(None,25)
Set texttodisplay to descrp[i]
Set textsurface to a message containing the texttodisplay
Blit the message to the screen

```

Flip the display

Put a time delay of 1000 milliseconds on

### **ASKING FOR INPUT OF TILE CO-ORDINATES**

Def main(Parameters = tiles, descrp, size)

For I in range(length of tiles[0])

Set screen to a screen of size size

Fill the screen in black

Call the firstdisplay subroutine, passing in tiles, descrp and I

Fill the screen in black

Update the display

Set xcoord to the result of ask, passing in screen and “ x co-ord”

Set xcoord to the result of inputvalidation, passing in xcoord

Set ycoord to the result of ask, passing in screen and “ y co-ord”

Set ycoord to the result of inputvalidation, passing in ycoord

If I!=0

Set valid to False

While valid is false

Set valid to inputval2, passing in arrayofcoords, xcoord, ycoord, valid

If valid is False

Fill the screen in black

Set basicfont to PyGame.font.SysFont(None, 24)

Set text to a message saying ‘Co-ordinate was already entered’

Blit the text to the screen

Flip the display

Set a 2000 millisecond delay

Fill the screen in white

Call game2 subroutine, passing in arrayofcoords, tiles, money, False, False and True

Flip the display

Call firstdisplay, passing in tiles, descrp and i

Fill the screen in black

Update the display

```

Set xcoord to the result of ask, passing in screen and " x co-ord"
Set xcoord to the result of inputvalidation, passing in xcoord
Set ycoord to the result of ask, passing in screen and " y co-ord"
Set ycoord to the result of inputvalidation, passing in ycoord
End if
End While
End if
Append xcoord to arrayofcoords[0]
Append ycoord to arrayofcoords[1]
Call the game2 subroutine, passing in arrayofcoords, tiles, money, False and False
End for loop
Set blank to an empty array
For I in range (length of money [0])
Append "" to the blank array
End for
For I in range (length of money [0])
Set screen to a screen of size size
Fill the screen in black
Call the firstdisplay subroutine, passing in money, blank and I
Fill the screen in black
Update the display
Set xcoordm to the result of ask, passing in screen and " x co-ord"
Set xcoordm to the result of inputvalidation, passing in xcoordm
Set ycoordm to the result of ask, passing in screen and " y co-ord"
Set ycoordm to the result of inputvalidation, passing in ycoordm
While valid is false
Set valid to inputval2,
If valid is False
Fill the screen in black
Set basicfont to PyGame.font.SysFont(None, 24)
Set text to a message saying 'Co-ordinate was already entered'

```

Blit the text to the screen  
 Flip the display  
 Set a 2000 millisecond delay  
 Fill the screen in white  
 Call game2 subroutine, passing in arrayofcoords, tiles, money, False, False and True  
 Flip the display  
 Call firstdisplay, passing in tiles, money and i  
 Fill the screen in black  
 Update the display  
 Set xcoordm to the result of ask, passing in screen and " x co-ord"  
 Set xcoordm to the result of inputvalidation, passing in xcoord  
 Set ycoordm to the result of ask, passing in screen and "y co-ord"  
 Set ycoordm to the result of inputvalidation, passing in ycoord  
 End if  
 End While  
 End if  
 Append xcoordm to arrayofcoords[0]  
 Append ycoordm to arrayofcoords[1]  
 Call the game2 subroutine, passing in arrayofcoords, tiles, money, False and False  
 Set introdone to True  
 Def game2 (Parameters = arrayofcoords, tiles, money, finished, maingame, reenter = None, rounds = None)  
 Set screen to a display of size size  
 Fill the screen in white  
 Set up basicfont as PyGame.font.SysFont(None,24)  
 For x in range(7)  
 Set test to a message containing string of x+1  
 Blit test to the screen at (x+1)\*70,60  
 For y in range(7) within the for loop above  
 Blit normallisth to the screen at x+1 \* 64+5, y+1 \* 64  
 End both for loops  
 For x in range(8)

```

For y in range(7) within the loop
Set test2 to a message containing the string of y + 1
Blit test2 to the screen
Blit normallinev to the screen at x+1 * 64, y+1 * 64+5
End both fors
For l in range(length of arrayofcoords[0])
If l is less than or equal to ten
Blit tiles[0][l] at arrayofcoords[0][l]*64+12, arrayofcoords[1][l]*64+20
Else
Blit money[0][l-11] at arrayofcoords[l][l]*64+12, arrayofcoords[1][l]*64+20
(11 less places because ten action tiles)
End if
Set fontobject to PyGame.font.Font(None,30)
If finished is false and reenter isn't true
Blit a message saying 'Press any key to continue' to the screen
Flip the display
Set keypressed to False
While keypressed is False
Get the current event
If a key is pressed
Set keypressed to True
End if
Elif finished is False and reenter is True
Blit a message saying 'Please enter a co-ordinate not already chosen' to the screen
Flip the display
Set a 2000 millisecond delay
Elif finished is True and maingame is True
Pump the connection
Pump the client
Blit the current score of the player to the screen
Blit what round the player is on to the screen

```

Flip the display

### **CHOOSE SOMEONE TO USE ACTION AGAINST**

#### **Class Button**

Def \_\_init\_\_ (Parameters = self, player, action, points, shields, mirrors and order)

Set self.player1 to None

Set self.player2 to None

Set chosen to None

Initialize the font

Call self.mainbutton, passing in player, action, points, shields, mirrors and order

Def display (Parameters = self)

Set a screen of size 650 by 370

Def update\_display(Parameters = self,action)

Fill the screen in white

Set up self.fontobject as PyGame.font.Font(None, 30)

If action isn't present or magnifying glass

Blit a generic message to the screen displaying the name of the action

Else if action is present

Blit 'Who do you want to give a present to' to the screen

Else if action is magglass

Blit 'Whose score do you want to see' to the screen

Create Button1

Create Button 2

Flip the display

Def mainbutton(Parameters = self, player, action, points, shields, mirrors and order)

Let self.Button1 be Buttons.Button()

Let self.Button3 be Buttons.Button()

Set goround to True

If the current player is 3

Set self.player1 to 1

Set self.player2 to 2

Else if the current player is 2

Set self.player1 to 1



```

Set self.player2 to 3
Else if the current player is 1
Set self.player1 to 2
Set self.player2 to 3
Update the display
While goround is True
Pump the PyGame.event
Update the display with buttons to choose from
For event in PyGame.event.get
If event.type is Quit
Quit the game
Else if event.type is mousebuttondown
(Inside the elif) If Button1 was pressed
Set chosen to self.player1
(another if inside the if )
If action is magnifying glass
Set pos to the result of linearesearch, passing in order and chosen
(nested if)
If the person chosen does not have a shield or a mirror
Call the definition show, passing in self, chosen and points
Else if the person has a shield
Do nothing
Else if the player chosen has a mirror
Call the show definition, passing in self, player and points (which will show them their
own score)
End if
Send a network action back to the server, sending action, playerchosen and player who
is attacking
Pump the connection
Pump the client
Set goround to False
Else if button 2 is pressed

```

Set chosen to self.player2  
 (another if inside the if)  
 If action is magnifying glass  
 Set pos to the result of linearsearch, passing in order and chosen  
 (nested if)  
 If the person chosen does not have a shield or a mirror  
 Call the definition show, passing in self, chosen and points  
 Else if the person has a shield  
 Do nothing  
 Else if the player chosen has a mirror  
 Call the show definition, passing in self, player and points (which will show them their own score)  
 End if  
 Send a network action back to the server, sending action, playerchosen and player who is attacking  
 Pump the connection  
 Pump the client  
 Set goround to False

### **CHOOSE AN ACTION TO PERFORM WITH WILD CARD**

#### **Class Button\_Wild**

Def \_\_init\_\_(Parameters = self)  
 Initialize the font  
 Call self.mainbutton  
 Def display (Parameters = self)  
 Set up a 400 by 400 screen  
 Def update\_display (Parameters = self)  
 Fill the screen in white  
 Set self.fontobject to PyGame.font.Font(None, 10)  
 Create Button1  
 Create Button2  
 Create Button3  
 Create Button4

```

Create Button5
Create Button6
Create Button7
Create Button8
Flip the display
Def mainButton(Parameters = self)
Set self.Button1 to Buttons.Button()
Set self.Button2 to Buttons.Button()
Set self.Button3 to Buttons.Button()
Set self.Button4 to Buttons.Button()
Set self.Button5 to Buttons.Button()
Set self.Button6 to Buttons.Button()
Set self.Button7 to Buttons.Button()
Set self.Button8 to Buttons.Button()
Set goround to True
Update the display
While goround is True
Pump PyGame.event
Call self.update_display
For event in PyGame.event.get
If event.type is PyGame.Quit
Quit
Else if event.type is mousebuttondown
If Button1 is pressed
Send to the server that rob is the action chosen
Pump the connection
Pump the client
Set goround to False
Elif Button2 is pressed
Send to the server that kill is the action chosen
Pump the connection

```

Pump the client  
Set goround to False  
Elif Button3 is pressed  
Send to the server that swap is the action chosen  
Pump the connection  
Pump the client  
Set goround to False  
Elif Button4 is pressed  
Send to the server that present is the action chosen  
Pump the connection  
Pump the client  
Set goround to False  
Elif Button5 is pressed  
Send to the server that magglass is the action chosen  
Pump the connection  
Pump the client  
Set goround to False  
Elif Button6 is pressed  
Send to the server that bank is the action chosen  
Pump the connection  
Pump the client  
Set goround to False  
Elif Button7 is pressed  
Send to the server that bomb is the action chosen  
Pump the connection  
Pump the client  
Set goround to False  
Elif Button8 is pressed  
Send to the server that double is the action chosen  
Pump the connection  
Pump the client

Set goround to False

## **CLIENT**

**Class MyNetworkListener** (parameters = ConnectionListener)

Def \_\_init\_\_ (Parameters = self, host and port)

Connect to the server

Set up a screen of size 550 by 350

Set self.font to be PyGame.font.SysFont(None,20)

Set self.totalplayers to None

Set self.num to None

Set self.pressedkey to None

Caption the window "Pirate x" where x is the player number

Set self.rplay to zero

Set self.ready to False

Set self.allready to False

Set self.current to [0,0,0]

Set self.currentsquare to [0,0,0]

Set self.action to None

Set self.points to [0,0,0]

Set self.playernot to zero

Set self.actioncomplete to False

Set self.playerc to zero

Set self.playernot1 to None

Set self.shields to an empty array

Set self.mirrors to an empty array

Set self.orders to an empty array

Set self.position to zero

Set self.summary to an empty 2D array with 5 arrays

Set self.place to 10

Set self.summaryflag to False

Set self.rounds to 1

Set self.banks to an empty array

```

Set self.bankflag to False
Set self.introdone to False
Def Network (Parameters = self,data)
Print data
Def Network_players (Parameters = self, data)
|
Set self.totalplayers to data['playerno']
Def Network_number (Parameters = self, data)
Set self.num to data['num']
Set caption of the window to Pirate x where x is the player number
Def Network_playrno (Parameters = self, data)
Set self.rplay to data['playno']

```

#### GETTING THE ACTION BASED ON THE CO-ORDINATE

```

Def Network_changesquare (Parameters = self, data)
Set self.currentsquare[0] to data['x']
Set self.currentsquare[1] to data['y']
Set self.currentsquare[2] tp data['counter']
Set xcoords to self.currentsquare[0]
Set ycoord to self.currentsquare[1]
For I in range (length of arrayofcoords[0])
For j in range(length of arrayofcoords[0])
If xcoords is arrayofcoords[0][j] and ycoords is arrayofcoords[1][j]
Set self.position to j
Break out of the loop
End if
End fors
Pump the connection
Pump the client
If self.position is less than or equal to ten
Set self.action to tilesdescrip[self.position]

```

**Commented [BR29]:**

**Commented [BR30]:** Okay, this is actually Megan but word is set up under my dad's account  
A side note on PodSixNet:

From here on in you start getting Network Events, and these get data in from the server.

When you see for example data['playerno'] this means any of the data passed over in the client action 'playerno' is passed into the variable it is assigned to

**Commented [BR31]:** If it is less than or equal to ten, this means it will be a action tile as there is only 10 of these action tiles

Set tiles[0][self.position] to tiles[1][self.position]

Call game2, passing in arrayofcoords, tiles, money, True and True

Else

Set self.action to moneydescrp[self.position -11]

Set money[0][self.position-11] to money[1][self.position-11]

Trigger game2. Passing in arrayofcoords, tiles, money, True, True, False, self.points[self.num-1] and self.rounds

End if

Send the server the self.action array and the self.num

Pump the connection

Pump the client

#### UPDATING THE CURRENT SCORES

Def Network\_current (Parameters = self, data)

Set self.points[0] to data[0]

Set self.points[1] to data[1]

Set self.points[2] to data[2]

#### UPDATING THE TILES TO FINISHED STATES

Def Network\_complete (Parameters = self, data)

Set self.actioncomplete to data['actc']

If self.position is less than or equal to 10

Set self.action to tilesdescrp[self.position]

Set tiles[0][self.position] to tiles[2][self.position]

Else

Set self.action to moneydescrp[self.position-11]

Set money[0][self.position-11] to money[2][self.position-11]

#### CALLING THE SUBROUTINE TO ALLOW USER TO CHOOSE WHO TO ATTACK

Def Network\_choose (Parameters = self, data)

Set self.playernot to data['not']

Set self.action to data['square']

Set self.shields to data['shields']

**Commented [BR32]:** Sets the image to active state

**Commented [BR33]:** Due to there being 10 action tiles, 11 must be taken off to get back to the equivalent part of the money array

**Commented [BR34]:** Sets the picture to active state

**Commented [BR35]:** Sets the image to finished state

**Commented [BR36]:** Sets images to finished state

Set self.mirrors to data['mirrors']

Set self.orders to data['actorder']

Call Button, passing in self.playernot, self.action, self.points, self.shields, self.mirrors and self.orders

Pump the connection

Pump the client

#### **CALLING THE SUBROUTINE TO ALLOW USER TO CHOOSE AN ACTION**

Def Network\_wild (Parameters = self, data)

Set self.playernot to data['choose']

Set self.action to data['square']

Call the Button\_Wild() subroutine

#### **GENERATING A SUMMARY SCREEN FOR THE ROUND**

Def Network\_summary (Parameters = self, data)

Set self.summary to data['array']

Fill the screen in white

Set found to False

For I in range(length of self.summary[0])

If self.summary[1][i] is equal to self.num (if they have been attacked)

Set found to True

If self.summary[0][i] is present

Blit a message to the screen telling the user which player gave them a present

Else if self.summary[0][i] is magglass

Blit a message telling the user which player saw their score

Else if self.summary[0][i] is swap

Blit a message telling the user which player swapped score with them

Else

Blit a message telling the user which action was performed and by who

End if

Increment self.place by 15

If self.summary[4][i] is True



```

Blit to the screen that the action was mirrored
Else if self.summary[3][i] is True
Blit to the screen that the action was shielded
End if
End for
For I in range(length of self.summary[0])
If self.summary[2][i] is self.num (if they have attacked someone)
If self.summary[3][i] is True
Blit which player used a shield against their action
Increment self.place by 15
Else if self.summary[4][i] is True
Blit which player used a mirror against their action
Increment self.place by 15
End if
End if
End for
If found is False
Show the grid still (call game2, passing in arrayofcoords, tiles, money, True, True, False,
self.points[self.num-1] and self.rounds)
End if
Flip the display
Set a 5000-millisecond delay
Pump PyGame.event
Set self.place to 10
Set self.summaryflag to True
ADDING BANK SCORES TO THE END TOTAL
Def Network_bank (Parameters = self, data)
Set self.banks to data['scores']
For I in range(3)
Set self.points[i] to self.points[i] + self.banks[i]

```

End for

Set self.bankflag to True

### **MAIN LOOP**

Def loop (Parameters = self)

Set self.font to PyGame.font.Font(None,30)

Set test to True

While test

Pump the connection

Pump the client

Trigger game\_start()

If self.ready is False

Send to the server the playernumber

End if

Set self.ready to True

While self.rplay is less than 3

Pump the connection

Pump the client

Pump PyGame.event

Call game2, passing in arrayofcoords, tiles, money, True and False

Pump the connection

Pump the client

Pump PyGame.event

End While

Set self.allready to True

Fill the screen in red

Blit a message saying 'Game Beginning' to the screen

Flip the display

Set a 2000 millisecond delay

Send to the server that the client is ready

For I in range(49)

```
While self.actioncomplete is False
PyGame.event.get
Trigger game2, passing in arrayofcoords, tiles, money, True, True, False,
self.points[self.num-1] and self.rounds
Flip the display
Pump the connection
Pump the client
PyGame.event.get()
End while
Set self.rounds to self.rounds + 1
Fill the screen in white
Flip the display
Pump the connection
Pump the client
Blit to the screen the current points
Flip the display
While self.summaryflag is False
PyGame.event.get()
Pump the connection
Pump the client
Send the network action 'loopy' to the server
Set self.actioncomplete to False
End for
Set keypressed to False
While keypressed is False
Make the screen lilac
Blit 'End of the game' message to the screen
Blit the end score of the player to the screen
Set winner to the index of the max value in self.points
If the player is the winner
```

Trigger mainwinner

Else

Blit the winner and their score to the screen

End if

Blit press any key to exit to the screen

Flip the display

If a key is pressed

Set keypressed to True

Quit

End while

Set test to False

#### **CONNECTING TO A CHOSEN SERVER**

Blit a message asking for the IP address of the server

Blit a message saying leave empty for local host

Set server to be the result of input for server ip

If server is "

Set server to 'localhost'

End if

Set client to MyNetworkListener(server,31500)

Start the client loop

#### ***Server Pseudocode***

Import all necessary libraries

From random import random

From time import sleep

From PodSixNet.Channel import Channel

From PodSixNet.Server import Server

Set dir\_path to the current working directory of the file

Get the current working directory

Change the current working directory to dir\_path

Def linearsearch (Parameters = array, searchkey)

Set pos to zero

Set found to False

While pos is less than the length of the array and found is not true

If array[pos] is equal to searchkey

Set found to True

Set foundat to pos

End if

Pos = pos + 1

Return foundat

#### **RANDOMIZING THE LIST OF CO-ORDINATES**

Def random\_squares

Set setofcoordinates to an empty 2d array with 3 arrays

For x in range(7)

For l in range(7)

Append x+1 to setofcoordinates[0]

Append l + 1 to setofcoordinates[1]

Set number to a random number between 0 and 1

Append number to setofcoordinates[2]

End fors

For x in range(49)

For l in range(49)

If setofcoordinates[2][i] <= setofcoordinates[2][i+1]

For n in range(3)

Swap the values in arrayofcoords [n][i] with the values of arrayofcoords [n][i+1]

End for

End if

End fors

Return setofcoordinates

**Class ClientChannel** (Parameters = Channel)

Def \_\_init\_\_ (Parameters = self, \*args, \*\*kwargs)

Channel.\_\_init\_\_(self, \*args, \*\*kwargs)

Set self.play to zero

Set self.play2 to zero

Set self.action to None

Set self.number to None

Def Network (Parameters = self, data)

Print data

Def Network\_playready (Parameters = self, data)

Set self.play to data['playr']

Append self.play to the server array playersready

Send the length of playersready to the clients

Def Network\_sendcoord (Parameters = self, data)

Set self.play2 to data['playr2']

Append self.play to the server playersready2 array

#### **RECEIVING THE ACTION AND THE PLAYER INTO THE SERVER**

Def Network\_actiontodo(Parameters = self, data)

Set self.action to data['actionsarray']

Set self.number to data['number']

Append self.action to the server array actions

Append self.number to the server array actionsplayer

Def Network\_ready (Parameters = self, data)

Set the server variable allready to True

#### **ASSINGING VARIABLES BASED ON CLIENT INPUT FOR ACTIONS**

Def Network\_chosen (Parameters = self, data)

```

Set the server variable player attack to data['playera']
Set the server variable playerattacking to data['playern']
Set the server variable playeraction to data['tobedone']
If the server variable playeraction is rob
Set the server variable robflag to True
Else if the server variable playeraction is kill
Set the server variable killflag to True
Else if the server variable playeraction is present
Set the server variable presentflag to True
Else if the server variable playeraction is swap
Set the server variable swapflag to True
Else if the server variable playeraction is magglass
Set the server variable magflag to True
End if
Append the value of the server variable playerinaction to the server sum[0] array
Append the value of the server variable playerattack to the server sum[1] array
Append the value of the server variable playerattacking to the server sum[2] array
Append the value of the server array shield[playerattack-1] to the server sum[3] array
Append the value of the server array mirror[playerattack-1] to the server sum[0] array

```

#### **HELPS THE SERVER TO ASSESS WHEN TO MOVE ON TO NEXT ROUND**

```

Def Network_loopy (Parameters = self, data)
Append ready to the server array loopagain

```

#### **ASSINGING VARIBALES BASED ON CLIENTS INPUT FOR WILDCARD**

```

Def Network_wildchosen (Parameters = self, data)
Set the server variable wildaction to data['tobedone']
Set the server variable wildflag to be True
(end of class)

```

### **ROB SUBROUTINE**

Def rob (Parameters = self, i)

Set playerinaction to self.players[self.actionsplayer[i]-1]

Trigger the network event choose in the client

While self.robflag is False

Pump the server

End while

Set pos to the result of linearsearch, passing in self.actionsplayer, self.playerattack

If self.shield[pos] is False and self.mirror[pos] is False

Set self.currentscores[self.playerattacking-1] to self.currentscores[self.playerattacking-1] + self.currentscores[self.playerattack-1]

Set self.currentscores[self.playerattack-1] to 0

Send to all the clients the current scores

Pump the server

Set self.robflag to False

Else if self.shield[pos] is True

Send to all the current scores

Set self.robflag to False

Else if self.mirror[pos] is True

Set self.newtobeattacked to self.playerattacking

Set self.newattacker to self.playerattack

Set self.currentscores[self.newattacker-1] to self.currentscores[self.newattacker-1] + self.currentscores[self.newtobeattacked-1]

Set self.currentscores[self.newtobeattacked-1] to 0

Send to all clients the current scores

Set the self.robflag to False

End if

Set self.robflag to False

### **KILL SUBROUTINE**

Def kill (Parameters = self, i)

Set playerinaction to self.players[self.actionsplayer[i]-1]



```

Trigger the network event choose in the client
While self.killflag is False
Pump the server
End while
Set pos to the result of linearsearch, passing in self.actionsplayer, self.playerattack
If self.shield[pos] is False and self.mirror[pos] is False
Set self.currentscores[self.playerattack-1] to 0
Send to all the clients the current scores
Pump the server
Set self.killflag to False
Else if self.shield[pos] is True
Send to all the current scores
Set self.killflag to False
Else if self.mirror[pos] is True
Set self.newtobeattacked to self.playerattacking
Set self.newattacker to self.playerattack
Set self.currentscores[self.newtobeattacked-1] to 0
Send to all clients the current scores
Set the self.killflag to False
End if
Set self.killflag to False
BANK SUBROUTINE
Def bank (Parameters = self, i)
Set self.bank[self.actionsplayer[i]-1] to self.bank[self.actionsplayer[i]-1] +
self.currentscores[self.actionsplayer[i]-1]
Set self.currentscores[self.actionsplayer[i]-1] to zero
Send to all the current scores
Pump the server
BOMB SUBROUTINE
Def bomb (Parameters = self, i)

```

Set self.currentscores[self.actionsplayer[i]-1] to zero

Send to all the current scores

Pump the server

#### **DOUBLE SUBROUTINE**

Def double (Parameters = self, i)

Double self.currentscores[self.actionsplayer[i]-1]

Send to all the current scores

Pump the server

#### **PRESENT SUBROUTINE**

Def Present (Parameters = self, i)

Set playerinaction to self.players[self.actionsplayer[i]-1]

Trigger the network action choose on the client

While the present flag is False

Pump the server

Add 1000 points to the current score of the player chosen

Send to all the current scores

Pump the server

Set the present flag to False

#### **SWAP SUBROUTINE**

Def swap (Parameters = self, i)

Set playerinaction to be self.players[self.actionsplayer[i]-1]

Trigger the network action choose on the client

While self.swapflag is False

Pump the server

Set pos to be the result of linearsearch, passing in self.actionsplayer and self.playerattack

If the chosen player does not have a mirror or a shield

Set up a temporary variable to store the current score of the player attacking

Let the player attacking's current score be the current score of the player they chose to attack

Let the player who is being attacked's current score be the temporary variable

Send to all the current scores

Pump the server

Set self.swapflag to False

Else if the chosen player has a shield

Send to all the current scores

Pump the server

Set the self.swapflag to False

Else if the chosen player has a mirror

Send to all the current score

Set the self.swapflag to False

#### **MAGNIFYING GLASS SUBROUTINE**

Def magglass (Parameters = self, i)

Set the playerinactionto be self.players[self.actionsplayer[i]-1]

Trigger the network event choose in the client

While self.magflag is False

Pump the server

#### **THE SERVER CLASS**

**Class MyServer**(Parameters = Server)

Set channelClass to ClientChannel

Def \_\_init\_\_ (Parameters = self, \*args, \*\*kwargs)

Server.\_\_init\_\_(self, \*args, \*\*kwargs)

Set self.players to an empty array

Set self.playersready = []

Set self.currentscores = [0,0,0]

Set self.actions = []

Set self.actionsplayer = []

Set self.square = random\_squares()

Set self.allready = False

Set self.robflag = False

Set self.playerattack = None

```
Set self.playerattacking = None
Set self.killflag = False
Set self.playeraction = None
Set self.presentflag = None
Set self.bank = [0,0,0]
Set self.magflag = False
Set self.playerscore = None
Set self.loopagain = []
Set self.swapflag = False
Set self.temp = None
Set self.sum to [[],[],[],[],[]]
Set self.shield to [False, False, False]
Set self.mirror to [False, False, False]
Set self.newtobeattacked to None
Set self.newattacler to None
Set self.wildflag to False
Set self.wildaction to None
```

```
Def Connected (Parameters = self, player, addr)
Append player to self.players
Send the player their playernumber
Send to all the current length of self.players
```

#### **ALLOWS DATA TO BE SENT ACROSS**

```
Def SendToAll (Parameters = self, data)
[p.Send(data) for p in self.players]
```

#### **MAIN LOOP**

```
Def Loop (Parameters = self)
While True
Pump the server
Sleep for 0.0001 seconds
```

If length of self.playersready is 3  
If self.allready is True  
For I in range(49)  
Set self.loopagain to an empty array  
Trigger the network event changesquare in the client  
Pump the server  
Whiel length of self.actions is less than 3  
Pump the server  
For I in range(3)  
If the current action is mirror  
Set self.mirror[i] to True  
If the current action is shield  
Set self.shield[i] to True  
If the current action is 200  
Add 200 to the current players score  
Send to all the current scores  
Elif the current action is 1000  
Add 1000 to the current players score  
Send to all the current scores  
Elif the current action is 3000  
Add 3000 to the current players score  
Send to all the current scores  
Elif the current action is 5000  
Add 5000 to the current players score  
Send to all the current scores  
Elif the current action is double  
Trigger the double subroutine, passing in self and i  
Elif the current action is bank  
Trigger the bank subroutine, passing in self and i  
Elif the current action is bomb

```
Trigger the bomb subroutine, passing in self and I
End if
End for
For I in range(3)
If the current action is wild
Set playerinaction to self.players[self.actionsplayer[i]-1]
Trigger the wild network event in the client
While self.wildflag to False
Pump the server
End while
If the wild action was rob
Trigger the rob subroutine, passing in self and I
Elif the wild action was kill
Trigger the kill subroutine, passing in self and I
Elif the wild action was present
Trigger the present subroutine, passing in self and I
Elif the wild action was bomb
Trigger the bomb subroutine, passing in self and I
Elif the wild action was swap
Trigger the swap subroutine, passing in self and I
Elif the wild action was magglass
Trigger the magglass subroutine, passing in self and I
Elif the wild action was double
Trigger the double subroutine, passing in self and I
Elif the wild action was bank
Trigger the bank subroutine, passing in self and I
End if
End for
For I in range(3)
If the current action is rob
```

Trigger the rob subroutine, passing in self and i  
Elif the current action is kill  
Trigger the kill subroutine, passing in self and i  
Elif the current action is present  
Trigger the present subroutine, passing in self and i  
Elif the current action is swap  
Trigger the swap subroutine, passing in self and i  
Elif the current action is magglass  
Trigger the magglass subroutine, passing in self and i  
End if  
Set self.robflag to False  
Set self.killflag to False  
Set self.magflag to False  
Set self.presentflag to False  
Set self.swapflag to False  
Set self.wildflag to False  
Pump the server  
End for  
Set self.actions to an empty array  
Set self.actionsplayer to an empty array  
Trigger the network action complete in the client  
Send to all the clients the self.sum array in the summary network action  
Set the self.shield array back to [False, False, False]  
Set the self.mirror array back to [False, False, False]  
Set self.sum to [[],[],[],[]]  
While length of self.loopagain is less than 3  
Pump the server  
End while  
Send to All the bank scores

#### **ENTERING SERVER IP**

Print a message asking for the ip address of the server

Print examples of server ips

Print a message telling the user to leave empty for local host

Set adresse to the input for server ip

If adresse is left blank

Set adresse to 'localhost'

End if

Set myserver to MyServer(localaddr = (adresse, 31500))

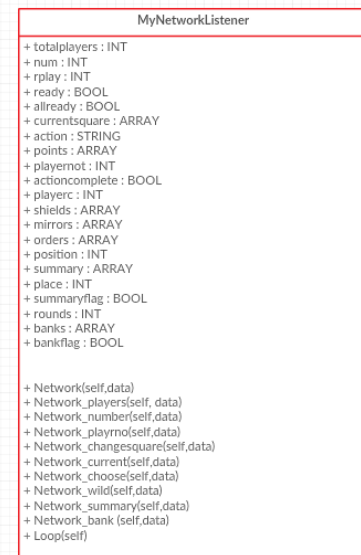
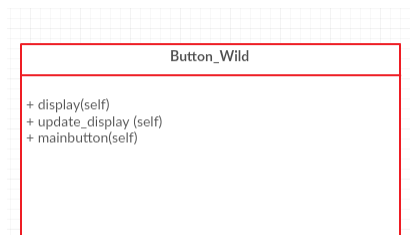
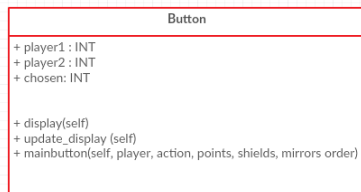
Start the myserver.Loop



## Class Diagrams

My code contains classes, but no sub-classes.

Client:



Server:



Data flow:

Client Side

Subroutines:

**Mainwinner()**

Parameters set up inside the subroutine:

- Screen
- Fontobject
- Background
- Stars
- Clock

Global variables used:

- N
- Screen\_w, screen\_h

Parameters passed out:

None

### **Game\_start()**

Parameters set up inside the subroutine:

- Screen
- Fontobject
- Gpressed
- Event

Global variables used:

- White
- Logo
- Background
- Passes into main:
  - Tiles, descrp, size

Parameters passed out:

None

Subroutine called:

- Main
- 

### **Get\_key()**

Parameters set up inside the subroutine:

- Event

Global variables used:

None

Parameters passed out:

None

Subroutine called:

None

### **Linearssearch()**

Parameters passed in:

- Array
- Searchkey

Parameters set up within the subroutine:

- Pos
- Found
- Foundat

Global variables used:

None

Parameters passed out:

Foundat

Parameters set up within the subroutine:

- Clock
- Screen
- Text
- Textrect
- Basicfont

Global variables used:

- Black
- Size

Parameters passed out:

Coord

### **Inputvalidation()**

Parameters passed in:

- Coord

Parameters set up within the subroutine:

- Screen
- Coord
- Basicfont
- Text
- Textrect

Global Variables used:

- Black
- Size

Parameters passed out:

- Coord

### **Inputval2()**

Parameters passed in:

- Arrayofcoords
- Xcoord
- Ycoord
- Valid

Parameters set up within the subroutine:

- Alreadyhere

Global Variables used:

None

Parameters passed out:

- Valid

### **Show()**

Parameters passed in:

- Self
- Chosen
- Points

Parameters set up within the subroutine:

- Screen
- Basicfont
- Text

Global variables used:

- White
- Size

Parameters passed out:

None

### **Display\_Box()**

Parameters passed in:

- Screen
- Message

Parameters set up within the subroutine:

- Fontobject

Global variables used:

None

Parameters passed out:

None

### **AskQ**

Parameters passed in:

- Screen
- Question

Parameters set up within the subroutine:

- Current\_string
- Inkey
- Testing

Global Variables used:

None

Subroutines called:

- Display\_box

Parameters passed out:

- `"".join(current_string[0:1])`

### **FirstdisplayQ**

Parameters passed in:

- Tiles
- Descrp
- I

Parameters set up within the subroutine:

- Screen
- Currenttile
- Myfont
- Texttodisplay
- Textsurface

Global Variables used:

- White

Parameters passed out:

None

## **Main()**

Parameters passed in:

- tiles
- descrp
- size

Parameters set up within the subroutine:

- screen
- xcoord
- ycoord
- valid
- basicfont
- text
- blank
- introdone

Global variables/arrays used:

- size
- black
- arrayofcoords
- money
- tiles

Subroutines called from within main:

- firstdisplay
- inputvalidation
- inputval2
- game2

Parameters passed out:

None

## **Game2()**

Parameters passed in:

- arrayofcoords
- tiles
- money
- finished
- maingame
- reenter (= None)
- score (=None)
- rounds (=None)

Parameters set up within the subroutine:

- screen
- basicfont
- test
- test2
- fontobject
- keypressed
- event

Global variables used:

- white
- size
- normallinev

Subroutines called from within main:

- None

Parameters passed out:

None

## **Class – Button**

*Init()*

Parameters passed in: self, player, action, points, shields, mirrors, order

Parameters set up:

- Self.player1
- Self.player2
- Chosen

Subroutines called:

Self.mainbutton (Passing in player, action, points, shields, mirrors, order)

*Display()*

Parameters passed in: self

Parameters set up:

- Self.screen

*Update\_display()*

Parameters passed in: self, action

Parameters set up:



- Self.fontobject

Sub routines called:

Self.Button1.create\_button (Passed in: self.screen, (107,142,35), 225, 135, 100, 0, "Player" + str(self.player1), (255,255,255))

Self.Button2.create\_button (Passed in: self.screen, (107,142,35), 225, 250, 200, 100, 0, "Player" + str(self.player2), (255,255,255))

### *Mainbutton()*

Parameters passed in: self, player, action, points, shields, mirrors, order

Parameters set up:

- Self.Button1
- Self.Button2
- Goround
- Self.player1
- Self.player2
- Chosen
- Pos

Subroutines called:

- Buttons.Button()
- Self.update\_display()
- Self.display()
- Show

Network events called:

- Connection.Send('chosen')

### **Class – Button\_Wild**

#### *Init()*

Parameters passed in:

- Self

Subroutines called:

- Self.mainbutton()

#### *Display()*

Parameters passed in:

- Self

Parameters set up:

- Self.screen

*Update\_display()*

Parameters passed in:

- Self

Parameters set up:

Self.fontobject

Subroutines called:

- self.Button1.create\_button (Parameters passed in: self.screen, (255,0,0), 75, 30, 100, 50, 0, "Rob", (255,255,255))
- self.Button2.create\_button (Parameters passed in: self.screen, (255,165,0), 260, 20, 100, 50, 0, "Kill", (255,255,255))
- self.Button3.create\_button (Parameters passed in: self.screen, (255,255,0), 100, 100, 100, 50, 0, "Swap", (255,255,255))
- self.Button4.create\_button (Parameters passed in: self.screen, (50,205,50), 280, 200, 100, 50, 0, "Present", (255,255,255))
- self.Button5.create\_button (Parameters passed in : self.screen, (0,191,255), 20, 210, 100, 50, 0, "Mag.Glass", (255,255,255))
- self.Button6.create\_button (Parameters passed in: self.screen, (75,0,130), 200, 270, 100, 50, 0, "Bank", (255,255,255))
- self.Button7.create\_button (Parameters passed in: self.screen, (148,0,211), 50, 320, 100, 50, 0, "Bomb", (255,255,255))
- self.Button8.create\_button (Parameters passed in : self.screen, (240,128,128), 250, 330, 100, 50, 0, "Double", (255,255,255))

*Mainbutton()*

Parameters passed in:

- Self

Class – MyNetworkListener

*\_\_Init\_\_()*

Network()

Network\_players(self,data)

Network action data sent from	players
Use of data	Updating the clients variable storing the number of players connected to the server

Network\_number(self,data)

Network action data sent from	number
Use of data	Updating the clients variable storing the player number of the client

Network\_playrno(self,data)

Network action data is sent from	playrno
Use of data	Updating the clients variable storing the number of players that have finished the input

Network\_changesquare(self,data)

Network action data is sent from	changesquare
Use of data	Updating client variables to allow the client to find out which tile they have at the current co-ordinate

Network\_current(self,data)

Network action data is sent from	current
Use of data	Updating client variables to store the current scores of all the players

Network\_complete(self,data)

Network action data is sent from	complete
Use of data	Updating the grid display, since the client knows the round has been complete so can display finished state of the tile

Network\_choose(self,data)

Network action data is sent from	choose
Use of data	Triggering a button interface which allows the player to choose who they want to attack

Network\_wild(self,data)

Network action data is sent from	wild
Use of data	Triggering a button interface which allows the player to choose which

	action they want to perform as part of the wild card
--	------------------------------------------------------

Network\_summary()

Network action data is sent from	summary
Use of data	To display an interface to the user informing them if they have been attacked/ any actions performed against them

Network\_bank()

Network action data is sent from	bank
Use of data	To update client variables to add the bank scores to the final score at the end of the game

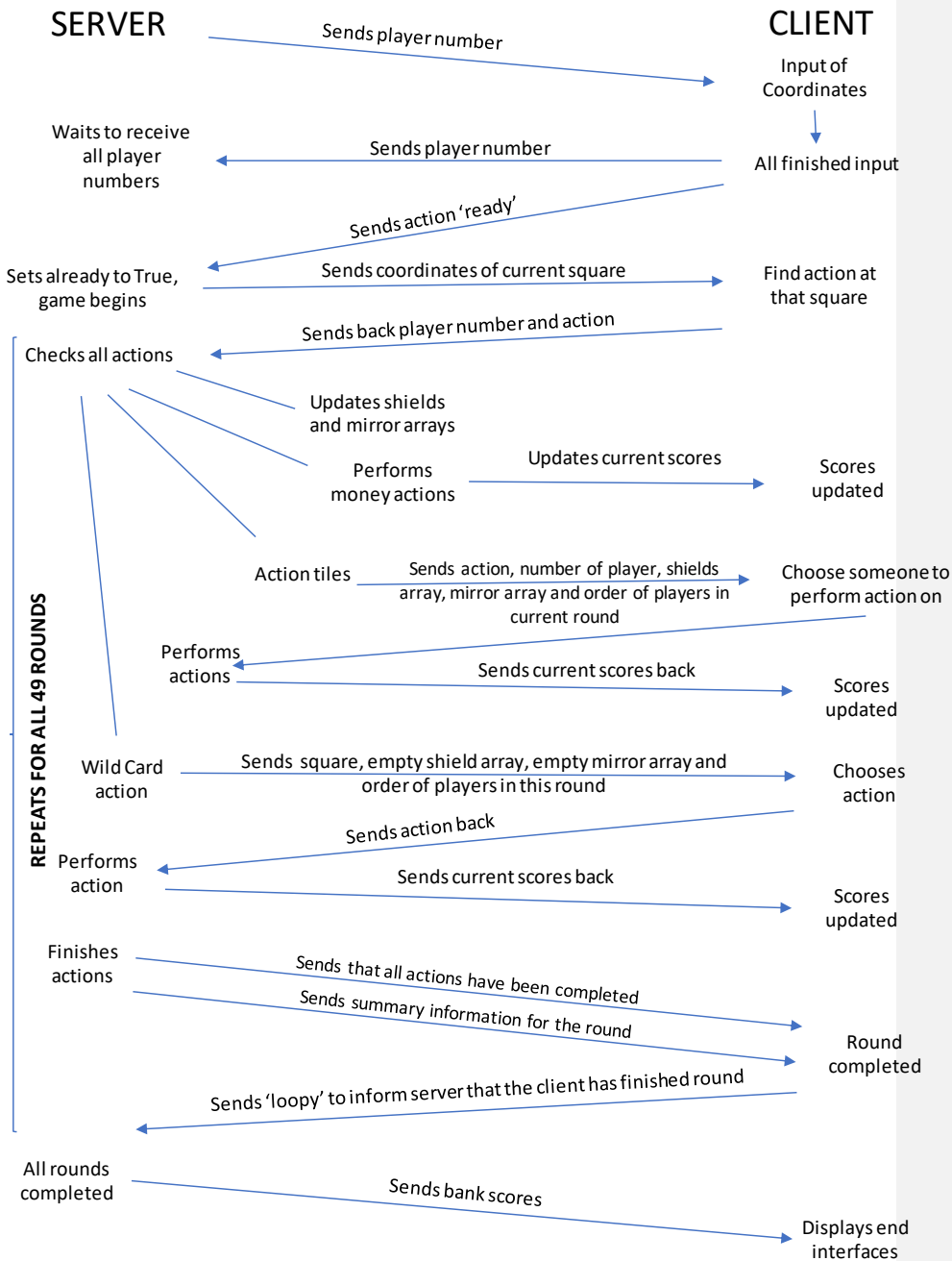
Loop()

#### REFINEMENT/ SUB-REFINEMENT OF COMPONENTS

For proof of refinement/sub refinement please see the following pages in the folder:

- **Input Section**
- *High Score plan, High score table code, Button, how to choose random squares, Action Plans part 1 and 2, shield and mirror initial, Choose Next Square, Shield/Mirror expanded, Table display, Random Squares in **Game Section***
- *Server Practice Code(page3-7), Communication Plan, Action tiles, Initial server thoughts in **Network Section***
- *Input Validation 2(Code), Order of Actions(Planning), Kill(Code), Present(Code), Shield Plan(Code), Summary(Code) in **Additional Pink Book***

## Data Flow between sub-programs



## RAM Based entities

I used the following data structures within my client program:

- *Variables:*
  - Dir\_path – stores the directory path of current file
  - Cwd – stores the current working directory
  - gameDisplay2 – used to store a display of size 550 by 350
  - background – used to store the intro screen image
  - instruct – used to store the image of the instructions
  - grid – used to store the empty squares for the grid
  - logo – used to store the pirate game logo
  - twohundred – used to store the 200 icon
  - thousand – used to store the 1000 icon
  - threethousand – used to store the 3000 icon
  - fivethousand – used to store the 500 icon
  - bank – used to store the bank icon
  - bomb – used to store the bomb icon
  - double – used to store the double icon
  - magglass – used to store the magnifying glass icon
  - present – used to store the present icon
  - rob – used to store the rob icon
  - shield – used to store the shield icon
  - swap – used to store the swap icon
  - kill – used to store the kill icon
  - mirror – used to store the mirror icon
  - normallinev – used to store the vertical line for the grid
  - normallineh – used to store the horizontal line for the grid
  - twohundreda – used to store the 200 active icon
  - thousanda – used to store the 1000 active icon
  - threethousanda – used to store the 3000 active icon
  - fivethousanda – used to store the 500 active icon
  - banka – used to store the bank active icon
  - bomba – used to store the bomb active icon
  - doublea – used to store the double active icon
  - magglassa – used to store the magnifying glass active icon
  - presenta – used to store the present active icon
  - roba – used to store the rob active icon
  - shielda – used to store the shield active icon
  - swapa – used to store the swap active icon
  - killa – used to store the kill active icon
  - mirrora – used to store the mirror active icon
  - twohundredd – used to store the 200 finished icon
  - thousandd – used to store the 1000 finished icon
  - threethousandd – used to store the 3000 finished icon
  - fivethousandd – used to store the 500 finished icon

bankd – used to store the bank finished icon  
 bombd – used to store the bomb finished icon  
 doubled – used to store the double finished icon  
 magglassd – used to store the magnifying glass finished icon  
 presentd – used to store the present finished icon  
 robd – used to store the rob finished icon  
 shieldd – used to store the shield finished icon  
 swapd – used to store the swap finished icon  
 killd – used to store the kill finished icon  
 mirrord – used to store the finished mirror icon  
 n – stores how many stars on the screen  
 screen\_w – stores the width of the winner's screen  
 screen\_h – stores the height of the winner's screen  
 screen – stores a screen of size screen\_w, screen\_h  
 fontobject – stores font of size 30  
 background – stores surface of size of the screen  
 clock – stores the PyGame.time clock  
 Gpressed – stores a Boolean value relating to whether a key has been pressed  
 Event – stores the PyGame event  
 Pos – stores the position in the array in the linear search  
 Found – stores a Boolean value on whether the value has been found in the linear search  
 Foundat – stores the position of where the searchkey was found in the linear search  
 Coord – stores the co-ordinate entered  
 Basicfont – stores font of size 25  
 Alreadyhere – Boolean value storing whether a co-ordinate has already been entered  
 Text – stores a message to be blitted to the screen  
 Inkey – stores the key pressed  
 Testing – stores the value of the question variable joined to the current answer  
 Currenttile – stores the value of tiles[0][i]  
 Myfont – stores font of size 25  
 Texttodisplay – stores descrp[i]  
 Textsurface – stores the font to blit to the screen  
 Xcoord – stores the x co-ordinate entered  
 Ycoord – stores the y co-ordinate entered  
 Valid – stores a Boolean value on whether the co-ordinate has already been entered  
 Xcoordm – stores the value for money x co-ord entered  
 Ycoordm – stores the value for money y co-ord entered  
 Introducedone – stores a Boolean value on whether input is finished  
 Test – stores text to be blitted to the screen  
 Test2 – stores text to be blitted to the screen  
 Keypressed – stores a Boolean value on whether a key has been pressed  
 Self.player1 – stores the number of one player to be chosen

Self.player2 – stores the number of another player to be chosen  
 Chosen – stores the number of the player chosen  
 Self.screen – stores a display of size 650,370  
 Self.fontobject – stores font of size 30  
 Self.button1 – creates an instance of a button  
 Self.button2 – creates an instance of a button  
 Goround – stores a Boolean flag  
 Self.button1 – stores a rob button  
 Self.button2 – stores a kill button  
 Self.button3 – stores a swap button  
 Self.button4 – stores a present button  
 Self.button5 – stores a magnifying glass button  
 Self.button6 – stores a bank button  
 Self.button7 – stores a bomb button  
 Self.button8 – stores a double button  
 Self.font – stores font of size 20  
 Self.totalplayers – stores the total number of players  
 Self.num – stores player number  
 Self.rplay – stores the number of ready players  
 Self.ready – stores a Boolean value related to if the player is ready  
 Self.allready – stores a Boolean value related to if all the players are ready  
 Self.playernot – stores the player who is choosing who to attack  
 Self.actioncomplete – stores a Boolean value indicating if the actions have been completed  
 Self.playerc – stores the player chosen  
 Self.action – stores the players current action  
 Self.position – stores the position of the current square in arrayofcoords  
 Self.place – stores the x position of text on the screen  
 Self.summaryflag – stores a Boolean flag to indicate whether summary is completed  
 Self.rounds – stores the current rounds  
 Self.bankflag – stores a Boolean flag to indicate whether bank scores have been added  
 Winner – stores the index of the position in the self.points array with the highest values  
 Server – stores the IP address entered by the user  
 Client – stores MyNetworkListener(server,31500)

- *1D arrays:*

White – stores the rgb for the colour white  
 Black – stores the rgb for the colour black  
 Size – stores the standard screen size  
 Descrp – used to store descriptions of all the action tiles  
 Tiles descrp – stores short descriptions for identification of actions  
 Moneydescrp – stores text descriptions of all the money tiles



Stars – stores the n number of stars on the screen  
Current\_string – stores the current characters entered during input  
Blank – stores "" for every value in money[0] (descriptions of money)  
Self.currentsquare – stores the current co-ordinate  
Self.shields – stores Boolean values for whether clients have shield  
Self.mirrors – stores Boolean values for whether clients have mirrors  
Self.orders – stores the order of the three clients  
Self.banks – stores the bank values  
Self.points – stores the users points

- *2D arrays:*

Grid positions – stores 1-7 in 2 arrays  
Tiles – stores all the tile images in normal, active and finished states  
Money – stores all the money images in normal, active and finished states  
Arrayofcoords – stores the co-ordinates of the clients tiles  
Self.summary – stores information for the summary screen

- *Classes:*

Button  
Button\_Wild  
MyNetworkListener

I used the following data structures within my server program:

- *Variables*

Dir\_path – stores current working directory of the file  
Cwd – stores the current working directory  
Pos – stores the index in the array in the linear search  
Found – stores a Boolean value on whether the searchkey had been found  
Foundat – stores the position that the searchkey is found at  
Self.play – stores the player number of the player  
Self.action – stores the action of the player  
Self.number – stores the number of the player  
Playerinaction – stores the player that has the current action  
ChannelClass – stores ClientChannel  
Self.allready – A Boolean flag storing whether players are all ready  
Self.robflag – stores a flag indicating whether have received data from client about who they want to rob  
Self.playerattack – stores the number of the player being attacked  
Self.playerattacking – stores the number of the player attacking  
Self.killflag – stores a flag indicating whether have received data from client about who they want to kill  
Self.playeraction – stores the players action

Self.presentflag - stores a flag indicating whether have received data from client about who they want to give a present to

Self.magflag - stores a flag indicating whether have received data from client about whose score they want to see

Self.playerscore - stores the score of the individual

Self.swapflag - stores a flag indicating whether have received data from client about who they want to swap scores with

Self.temp - stores a player's score in the swap subroutine

Self.newtobeattacked - stores player number during a shielded/mirrored action

Self.newattacker - stores player number during shielded/mirrored action

Self.wildflag - stores a flag indicating whether have received data from client about which action they want to perform

Self.wildaction - stores the action chosen as part of the wildcard

Addresse - stores the IP address on which the server is set up

Myserver - stores MyServer(localaddr=(addresse, 31500))

- *1D arrays*

Self.players - stores the players

Self.playersready - stores the numbers of the players ready

Self.currentscores - stores the current scores of the players

Self.actions - stores the actions of the players

Self.actionsplayer - a parallel array to self.actions storing player numbers

Self.bank - stores the bank scores of the players

Self.loopagain - stores the numbers of the clients ready for next round

Self.shield - stores an array of Booleans on whether clients have shields

Self.mirror - stores an array of Booleans on whether clients have mirrors

- *2D arrays*

Setofcoordinates - stores all the co-ordinates in the grid and their random number

Self.sum - stores info to send back to the client for the summary screen

- *Classes*

ClientChannel

MyServer

#### Refinements:

- Originally the plan was for magnifying glass was to send data to the client to let them decide whose score they wanted to see and then send data back to the server before sending the player chosen's score back to the client. However, this included un-necessary steps and sending of data, so it was decided that magnifying glass would be performed on the client right after they have made their decision and when the data was sent back to the server it would just update

a flag and move on with the game. This just saved un-necessary data flow between the server and the clients.

- The plan for the original choose next square tile was to allow the user to choose the next co-ordinate to play in the game. This would have involved the client having to choose the next co-ordinate, have that co-ordinate validated and then sending the validated co-ordinate back to the server to have it checked to see if it hadn't been used before. If it had it would have had to be sent back and the process may repeat itself for a long time. However with the implementation of the new choose next square tile it made the data flow less heavy between the client and the server and made it easier to make as an action.

### Input Validation:

#### Code:

```
# ----- INPUT VALIDATION 1 -----
#Checks the number is between 1 and 7 to fit on the grid
#Required to ensure that the client doesn't enter grid co-ordinates that aren't available
def inputvalidation(coord):
    #Turning into an integer in order to carry out comparisons
    coord = int(coord)
    #While the co-ordinates are outside the range
    while coord == 0 or coord > 7 or coord < 0:
        #Setting the screen to the correct screen size
        screen = pygame.display.set_mode(size)
        #Setting up the clock in order to put in a delay later
        clock = pygame.time.Clock()
        #Setting up the font
        basicfont = pygame.font.SysFont(None, 25)
        #Setting up the text to be blitted to screen
        #Error message
        text = basicfont.render('Invalid Input, please enter a number between 1 and 7', True, (white), (black))
        textrect = text.get_rect()
        #Making sure the text is displayed in the middle of the screen
        textrect.centerx = screen.get_rect().centerx
        textrect.centery = screen.get_rect().centery
        #filling the screen in black
        screen.fill(black)
        #displaying the input validation message to the screen
        screen.blit(text, textrect)
        #updating the screen
        pygame.display.update()
        #including a delay in the program
        pygame.time.wait(500)
        #fills the screen in black
        screen.fill(black)
        #asks the user again to enter an x co-ordinate again
        coord = (ask(screen, "Co-ord"))
        #turns it into an integer
        coord = int(coord)
    #returns co-ordinate to pass it back to the main subroutine
    return coord
```

## Input Validation 2

Code:

```
# ----- VALIDATION OF THE INPUTS PART 2 -----
# Compare the two values in the X and Y variables to all the other duples in the arrays
# Need to make sure that they are not entering the same numbers
# Cannot have two tiles in the same square
def inputval2(arrayofcoords,xcoord,ycoord,valid):
    #setting the boolean variable to false, only set to true when found in array
    alreadyhere = False
    #looping round for the whole length of the first array in the 2D array
    for j in range(len(arrayofcoords[0])):
        #compares the values for both X and Y
        if xcoord == arrayofcoords[0][j] and ycoord == arrayofcoords[1][j]:
            #set boolean value to true because the co-ordinate is already there
            alreadyhere = True
            #breaks out of the loop
            break
    #If the value hasn't been found in the array then it is a valid input
    if alreadyhere == False:
        valid = True
    #returning whether or not the input
    return valid
```

## Call Statements:

```
#Asks you to enter the X co-ordinate for placement of current tile
xcoord = (ask(screen, " X co-ord"))
#calls the input validation for the x coord
xcoord = inputvalidation(xcoord)
#Asks you to enter the Y co-ordinate for placement of current tile
ycoord = (ask(screen, " Y co-ord"))
#calls the input validation for the y coord
ycoord = inputvalidation(ycoord)
#have this in because if it is the first time round, there wouldn't be any need to compare and validate against other coords
if i != 0:
    #assuming the co-ordinate is not valid
    valid = False
    while valid == False:
        #calling the valid subroutine
        valid = inputval2(arrayofcoords,xcoord,ycoord,valid)
```

## Refinements:

- Originally input validation occurred in the main input subroutine, but to make it more efficient and easier to ensure that the inputs went through validation every time, both input validations were moved into separate subroutines that are now called from the main input subroutine.

## Document 10 – Implementation

### Development Issues Encountered:

Throughout development of the program I came across three main issues:

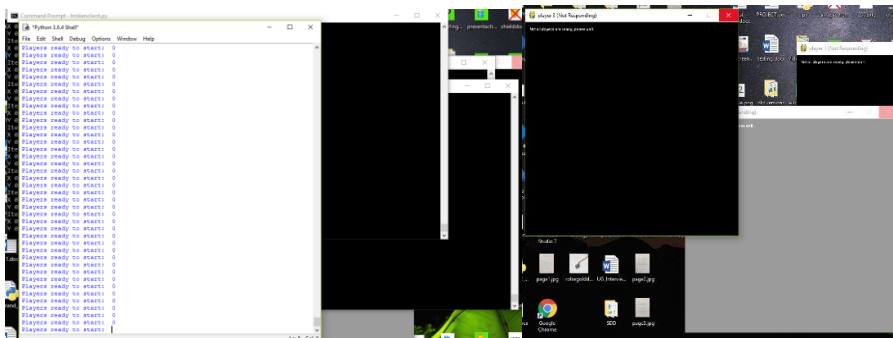
- Server
- Magnifying glass tile
- Button (start) Interface

### Server:

Starting the project, I had very little experience or knowledge in multiplayer games in Python so I knew that this area would cause me difficulties. I began trying to start very simple networks where clients could connect to the server and communicate between each other.

5/1/2018 – Server was not recognizing the fact that 3 players had finished input in the 2 by 2 prototype.

This copy of the code can be found in the folder 'Implementation'.



Server was outputting

'Players ready to start: 0' repeatedly and clients were outputting Iteration: 0, X : 0, Y : 0 in the command prompt and not doing anything on the PyGame window.

Part of the code where error was occurring:

```
def Loop(self):
    while True:
        myserver.Pump()
        sleep(0.0001)
        if len(self.playersready) == 3:
            myserver.Pump()
            while len(self.playersready2) < 3:
                #BREAKING HERE, SELF.PLAYERSREADY2 IS STILL SET TO ZERO
                print("Players ready to start: ", len(self.playersready2))
                myserver.Pump()
                # wait
                #print("Less than three ready to start")
            if len(self.playersready2) == 3:
                print("Got to the next step")
                # start a loop
                # SendToAll the co-ordinates of the chosen square
                for i in range(4):
                    #X co-ordinate
                    self.SendToAll({'action':'changesquare', 'x':self.square[i][0], 'y'
```

Reason why code wasn't working:

Initial value:

```
def __init__(self, *args, **kwargs):
    Server.__init__(self, *args, **kwargs)
    # initialise an empty array of players
    self.players = []
    print("Server initialised")

    # empty array of players ready
    self.playersready = []
    #####
    self.playersready2 = []
    #####
    self.square = random_squares()
    #print(self.square)
```

Self.rplay was not being updated, so it was not getting into the self.allready == True

```
#####
def Network_sendoord(self, data):
    self.play2 = data['play2']
    self._server.playersready.append(self.play)
    print("Total number of players ready to start - " + len(self._server.playersready))
    self._server.SendToAll({'action': 'play2', 'play2': len(self._server.playersready2)})
#####

this is the server definition, which handles and propagates Network events
class MyServer(Server):

    channelClass = ClientChannel

    def __init__(self, *args, **kwargs):
        Server.__init__(self, *args, **kwargs)
        # initialise an empty array of players
        self.players = []

        if self.ready == False:
            connection.Send({"action": "playready", "player": self.num})
            self.ready = True
            # while there are less than 3 players, loop
            # changed to 2 for now
            while self.play < 2:
                self.screen.fill((0,0,0))
                self.screen.blit(self.font.render("Not all players are ready, please w
                pygame.display.flip()
                # pump to check for changes
                connection.Pump()
                client.Pump()
            # once all players are ready, break out of while and set to True
            self.allready = True

            while self.allready == True:
                connection.Send({"action": "sendoord", "play2": self.num})
                # pump to check for changes
                connection.Pump()
                client.Pump()
            # server ready again
```

```
#####
self.play2 = 0
# when client does connection.Send(mydata), Network method is called
def Network(self, data):
    # prints all passed network data to console
    print(data)

# handles information about which player pressed a key and sends
# it back to all players
def Network_pressedkey(self, data):
    self.pressednum = data['pressedkey']
    print("Server info: player " + str(self.pressednum) + " pressed a key")
    # send the information back to all players
    self._server.SendToAll({'action': 'keyplay', 'pressednum': self.pressednum})

# when client says its ready, sends which player is ready to here
def Network_playready(self, data):
    self.play = data['play']
    # appending the id of the player that's ready
    self._server.playersready.append(self.play)
    print("Server info: player " + str(self.play) + " is ready")
    # print number of players ready
    print("Total number of players ready - " + str(len(self._server.playersready)))
    # send the information back to all players
    self._server.SendToAll({'action': 'playno', 'playno': len(self._server.players
#####
def Network_sendoord(self, data):
    self.play2 = data['play2']
    self._server.playersready2.append(self.play)
    print("Total number of players ready to start - " + len(self._server.playersr
    self._server.SendToAll({'action': 'play2', 'play2': len(self._server.playersr
#####

#####
def Network_playno(self, data):
    print("updating total players to " + str(self.totalplayers))
    # not needed
    def Network_players2(self, data):
        self.totalplayers2 = data['']

    def Network_number(self, data):
        self.num = data['num']
        # print out player number
        print("I am player number " + str(self.num))
        # set window title with player number
        pygame.display.set_caption("Player " + str(self.num))

    # this collects info on who pressed the key and updates the display
    def Network_keyplay(self, data):
        self.pressedkey = data['pressednum']
        print("Server says player " + str(self.pressedkey) + " pressed a key")
        self.screen.fill((0,0,0))
        self.screen.blit(self.font.render("Total players: " + str(self.totalplayers),
        self.screen.blit(self.font.render("Player " + str(self.pressedkey) + " pressed
        pygame.display.flip()

    # receives information on how many playhile self.ready == True: are ready from
    def Network_playno(self, data):
        # updated to number of the array in the server
        self.rplay = data['playno']
        print("Number of players ready is " + str(self.rplay))

    def Network_changesquare(self, data):
        self.currentsquare[0] = data['x']
        self.currentsquare[1] = data['y']
        ##### added to capture counter
        self.currentsquare[2] = data['counter']
        print("Received the updated square")
        print(self.currentsquare)
```

Something was going wrong with the sending back of the data to the clients meaning self.rplay was not being updated so it could not get into the starting loop.

I had quite a couple of issues with the server and it took a few iterations to get a fully working server that communicated successfully between clients.

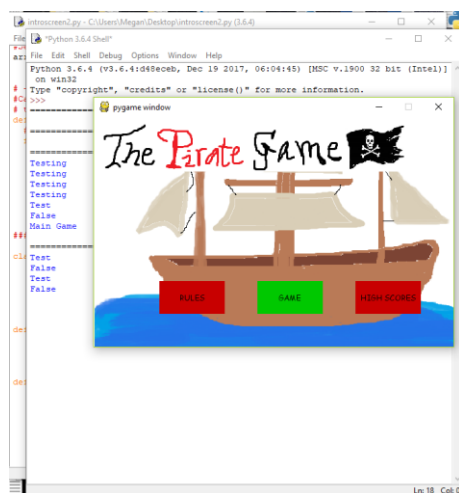
### Magnifying Glass.

Magnifying glass is a new added tile to this version of the pirate game, it was not a part of the original game played at school. It required more thinking and planning than other

actions and I encountered a few more problems compared to the other tiles. It was the tile that took the longest time to implement. I do not have any annotated code for this development problem.

#### *Button (start) Interface:*

The original plan was to have buttons to allow the user to navigate between rules, the start interface and game. However, I ran into a lot of difficulties around calling the subroutines from within the button.

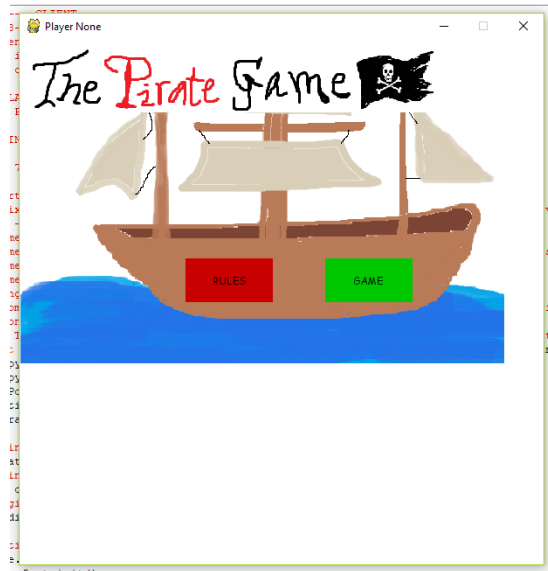


This was the original interface planned:

However, even though all the buttons performed what they should have in the practice 'introscreen2' code, when integrated to the main code, it would not call the game subroutine successfully.

Output:

- Showed screen above
- Allowed user to enter input for whole grid
- Went back to this grid after input



Code:

```
# ----- INTRO SCREEN CODE -----
class Background(pygame.sprite.Sprite):
    def __init__(self, image_file, location):
        #Call Sprite initialize
        pygame.sprite.Sprite.__init__(self)
        #loading the image into self.image
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location

def text_objects(text, font):
    #setting up the text surface
    textSurface = font.render(text, True, black)
    #returning values
    return textSurface, textSurface.get_rect()

#Button definition
def button(msg, x, y, w, h, ic, ac, flag, intro, action = None):
    #getting the position of mouse
    mouse = pygame.mouse.get_pos()
    #returns a sequence of booleans representing state of all the mouse buttons
    click = pygame.mouse.get_pressed()
    #if the mouse is over the button
    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(gameDisplay2, ac, (x, y, w, h))
        #and there is a click and there is an action stated
        if click[0] == 1 and action != None:
            #set the flag to be False
            flag = False
            #call the subroutine
            action(flag, intro)
        #else if there is no action stated
        elif click[0] == 1 and action == None:
            #set flag to False
            flag = False
    else:
        #draw a rectangle ??
        pygame.draw.rect(gameDisplay2, ic, (x, y, w, h))
    #set up the text
    smallText = pygame.font.SysFont("comicsansms", 12)
    #not sure what this does
    textSurf, textRect = text_objects(msg, smallText)
    textRect.center = ((x+(w/2)), (y+(h/2)))
    gameDisplay2.blit(textSurf, textRect)
```

Should have called game, passing in flag and intro

Also in the class: (called when the game button is pressed)



```

#doesn't wait and doesn't clear screen

#calling the main game subroutine
def game(flag, intro):
    #setting flag to False
    flag = False
    intro = False
    print("Flag is now", flag)
    print("Intro is now", intro)
    #calling the main game subroutine, starting the input
    main(tiles,descrip,gridpositions,positionx,positiony,size)
    #self.introdone = True

#triggering the interface back to the start screen
def trigger(flag, intro):
    #setting up gameDisplay as it was

```

As soon as the main subroutine finished it came back to the button class because the subroutine input (main subroutine) was called in was not complete, so I had to scrap this code and I opted for a key press interface because I was running out of time and it was not an essential requirement.

Also, when moving over to the key press start screen, I had a slight recursion problem, in the fact that because I called the subroutine from within the subroutine, when the player finished input it took them back to the intro screen because it hadn't finished the first calling of the subroutine.

I tried to build in a flag system but that failed to work.

Code with errors:

```

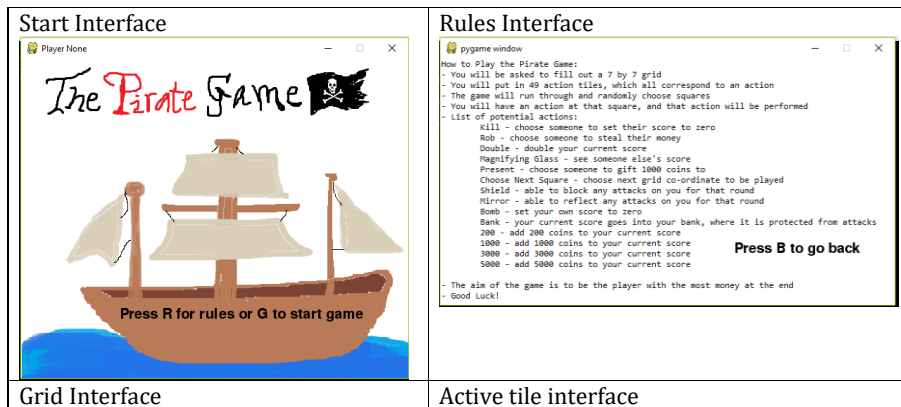
#####
def game_start():
    #updated size of the display
    #initialising some colours that can be used later
    #again initializing the screen
    #screen = pygame.display.set_mode(size)
    screen = pygame.display.set_mode((550, 450))
    #filling it in white
    screen.fill(white)
    #updating the display
    fontobject = pygame.font.Font(None,30)
    screen.blit(logo, (20,0))
    screen.blit(background, (0,110))
    screen.blit(fontobject.render("Press R for rules or G to start game" ,True,(
    pygame.display.flip()
    Gpressed = False
    while Gpressed == False:
        #pygame.event.get()
        event = pygame.event.poll()
        if event.type == KEYDOWN:
            if event.key == K_g:
                Gpressed = True
            elif event.key == K_r:
                screen = pygame.display.set_mode((650, 350))
                screen.fill(white)
                screen.blit(instruct, (0,0))
                # need to make a button to send back to home interface
                screen.blit(fontobject.render("Press B to go back" ,True,(0,0,0)), (42
                pygame.display.flip()
                #pygame.event.pump()
                event = pygame.event.poll()
                elif event.key == K_b:
                    game_start()
    # move on with the game
    main(tiles,descrip,gridpositions,positionx,positiony,size)
#####

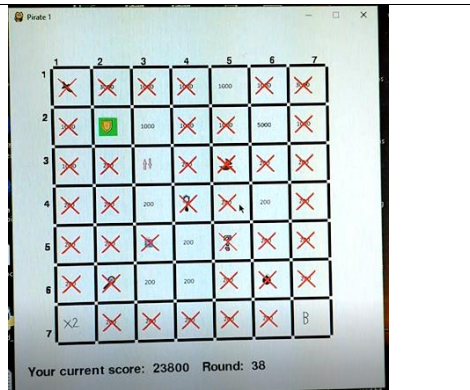
```

## Solution:

```
#####
def game_start():
    #updated size of the display
    #initializing some colours that can be used later
    #again initializing the screen
    #screen = pygame.display.set_mode(size)
    screen = pygame.display.set_mode((550, 450))
    #filling it in white
    screen.fill(white)
    #updating the display
    fontobject = pygame.font.Font(None,30)
    screen.blit(logo, (20,0))
    screen.blit(background, (0,110))
    screen.blit(fontobject.render("Press R for rules or G to start game" ,True, (0,0,0)), (140,350))
    pygame.display.flip()
    Gpressed = False
    while Gpressed == False:
        #pygame.event.get()
        event = pygame.event.poll()
        if event.type == KEYDOWN:
            if event.key == K_g:
                Gpressed = True
            elif event.key == K_r:
                screen = pygame.display.set_mode((650, 350))
                screen.fill(white)
                screen.blit(instruct,(0,0))
                # need to make a button to send back to home interface
                screen.blit(fontobject.render("Press B to go back" ,True,(0,0,0)), (420,260))
                pygame.display.flip()
                #pygame.event.pump()
                event = pygame.event.poll()
            elif event.key == K_b:
                screen = pygame.display.set_mode((550, 450))
                #filling it in white
                screen.fill(white)
                #updating the display
                fontobject = pygame.font.Font(None,30)
                screen.blit(logo, (20,0))
                screen.blit(background, (0,110))
                screen.blit(fontobject.render("Press R for rules or G to start game" ,True,(0,0,0)), (140,350))
                pygame.display.flip()
                Gpressed = False
            # move on with the game
    main(tiles,desorp,gridpositions,positionx,positiony,size)
#####
```

## Main Interface Elements:

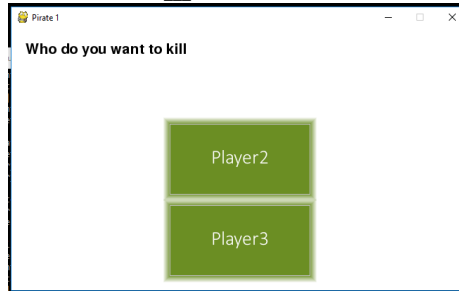




Finished tile interface



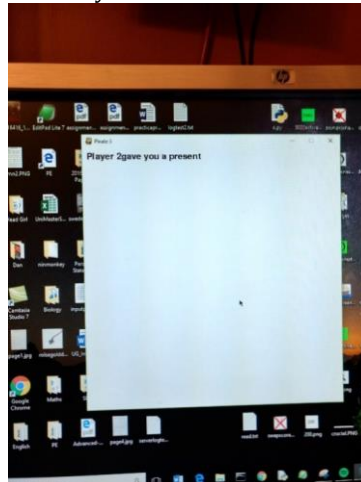
Choose who to \_\_ interface

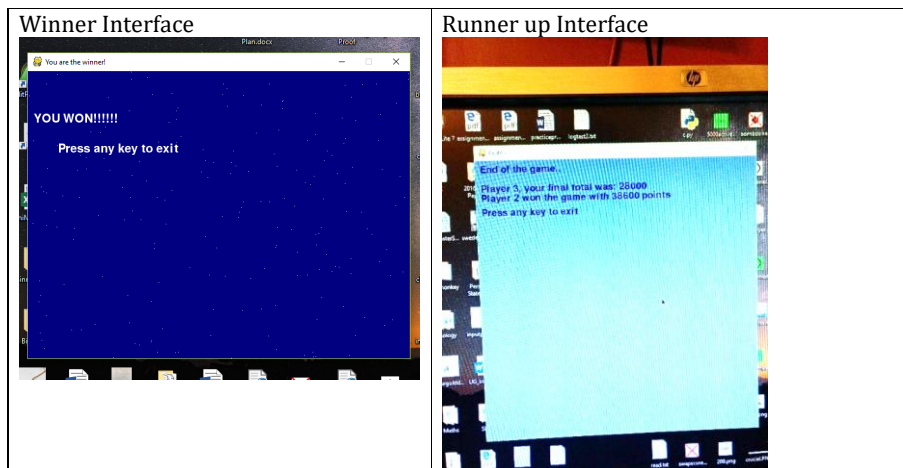


Choose an action to play



Summary Screen





## Document 11 – Final Testing

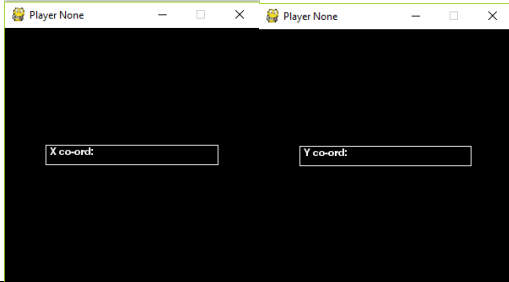
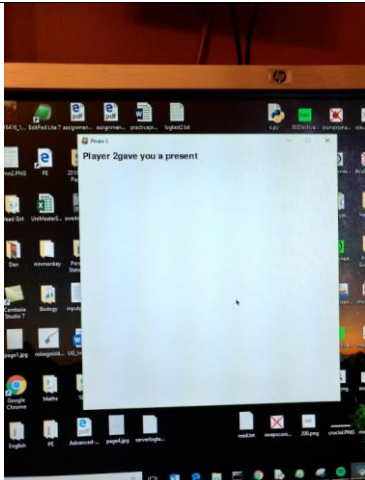
Throughout the progress of the project I completed various forms of testing. This document contains the proof that all the functions of my game are working and match up to the requirements specification.


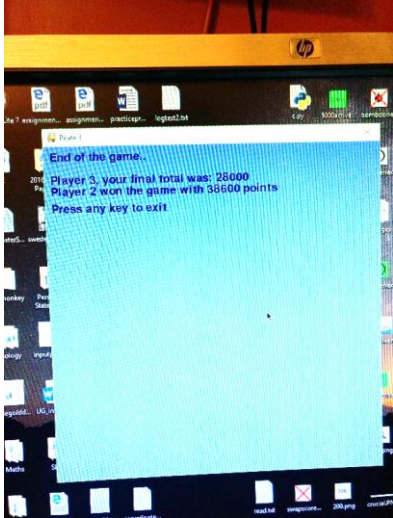
Any videos referred to in this document will be stored in the folder '*Video testing proof*'


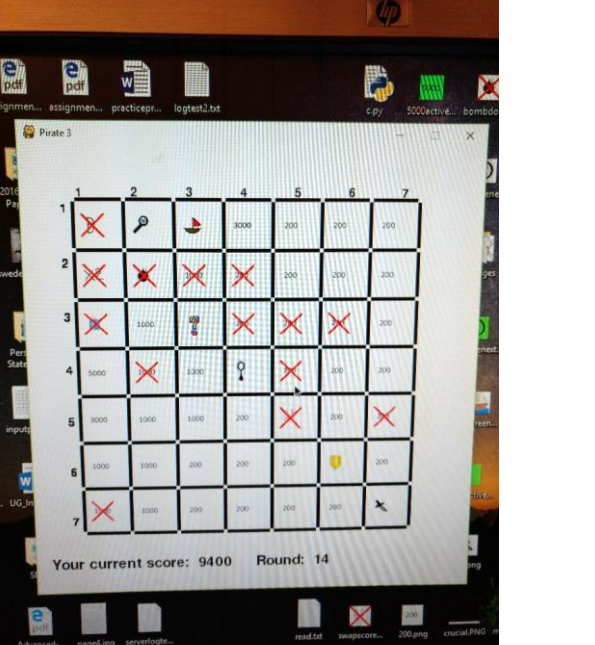
This section of the document focuses in on the various sections of the main program.

### Interfaces

Feature to be tested	Expected Result	Actual Result
Home interface	An interface which allows the user to go to either the instructions page or the game page	

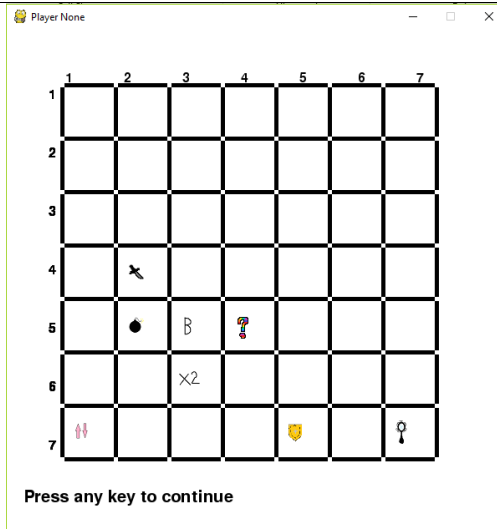
Rules key	When r is pressed, rules screen is displayed	See video 1
	When b is pressed, program goes back to start interface	See video 1
Game button	When g is pressed, game screen is displayed	See video 2
Input interface		
Game beginning screen	When the game is starting, an over screen is displayed to inform the users	See video 3
Summary Screen	Summarises the actions performed against the player	

Winner Interface	Shows the winner that they have won	
Runner up Interface	Informs the user of their end score and the winners end score	
Quit the game (Winner)	Allows the user to quit the game with a key press	See Video 4
Quit the game (Runner up)	Allows the user to quit the game with a key press	See Video 5
Grid Interface		See video 8

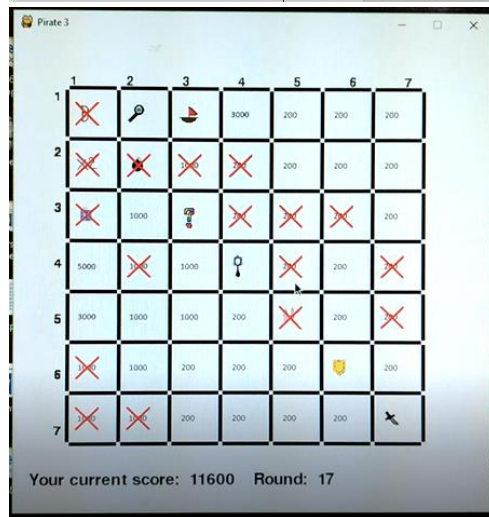
Active Tiles	Shows the user which tile is currently in play by highlighting it green.	
Finished Tiles	Shows the user which tiles are finished by crossing a X through them.	

Normal  
State  
Tiles

Displays  
normal icon  
of tiles  
during input  
and when  
not being  
used




Input



Not in use

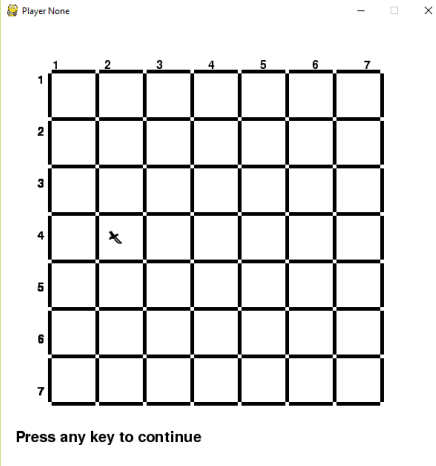


Wild Card Interface	Shows the user which choices they have as part of their wild card.	
---------------------	--------------------------------------------------------------------	-----------------------------------------------------------------------------------

### Main Game

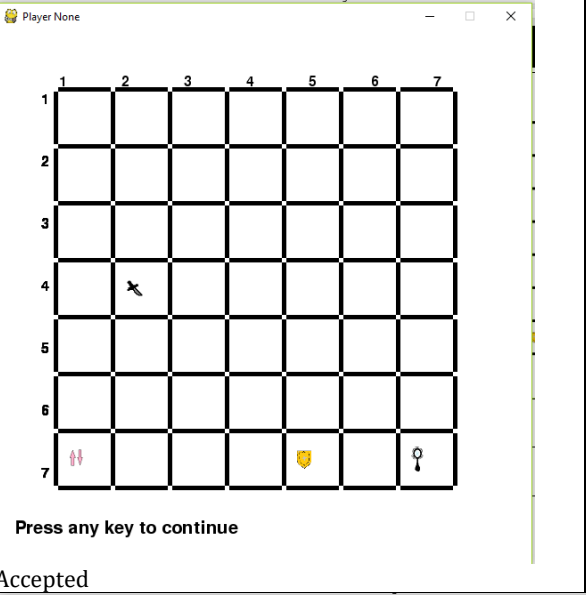
For most of the testing for the main game it is hard to get screenshots of the game in play, so I am using proof found in the server/client logs and the text that is outputted from the program.

For some of the screenshots, you may see player None. You can progress through input without having to have a server up and running, meaning the player number defaults to None.

Feature to be tested	Test Data/What should happen	Result
Input Validation (between 1 and 7)	2,4 (Accepted) Place at 2,4	

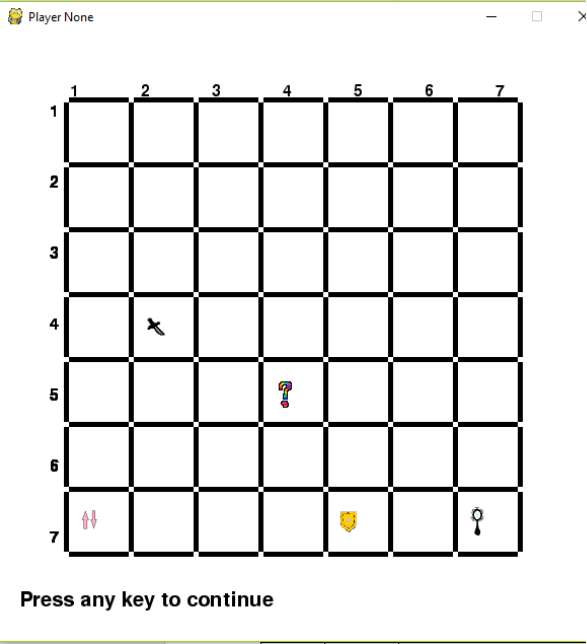
	5,8 (Exceptional) Not accepted	<div><div>Player None</div><div><div>Co-ord:</div></div></div> <div>Asks for co-ord again</div>
	1,7 (Extreme) Placed at 1,7	<div><div>Player None</div><div><div><div><div>1234567</div><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div></div><div><div><div><div>↖</div><div>↗</div><div>↘</div><div>↙</div></div></div></div><div>Press any key to continue</div></div></div><div>Accepted</div></div>

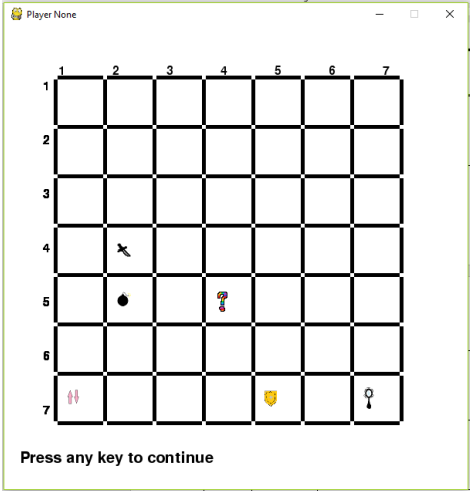
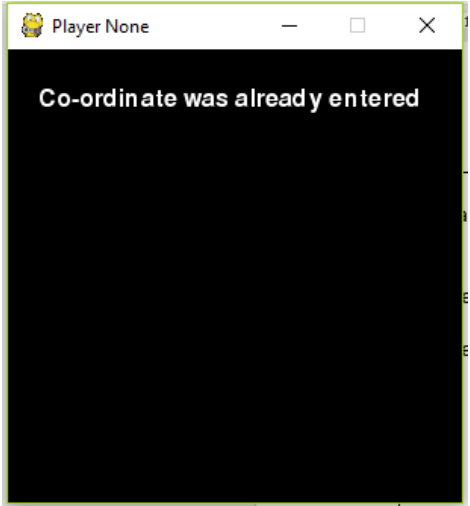
7,7 (Extreme)  
Placed at 7,7

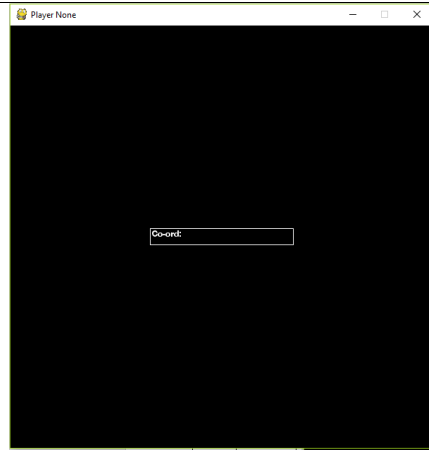


Accepted

42, 56  
(Accepted)  
Placed at 4,5

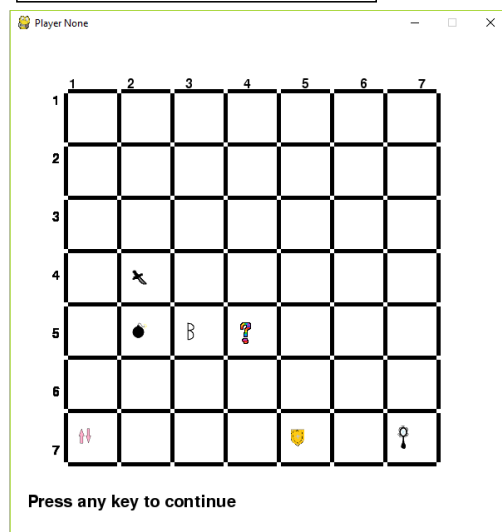


<p>Input Validation 2 (not entered before)</p>	<p>2,5(Accepted) 2,5(Exceptional) 3,8 (Exceptional) 3,5 (Accepted)</p>	<div data-bbox="571 448 933 510">Step 1 – 2,5</div> <div data-bbox="461 519 933 1008"></div> <div data-bbox="571 1037 933 1099">Step 2 – 2,5</div> <div data-bbox="461 1108 933 1612"></div> <div data-bbox="571 1641 933 1704">Step 3 – 3,8</div>
------------------------------------------------	----------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

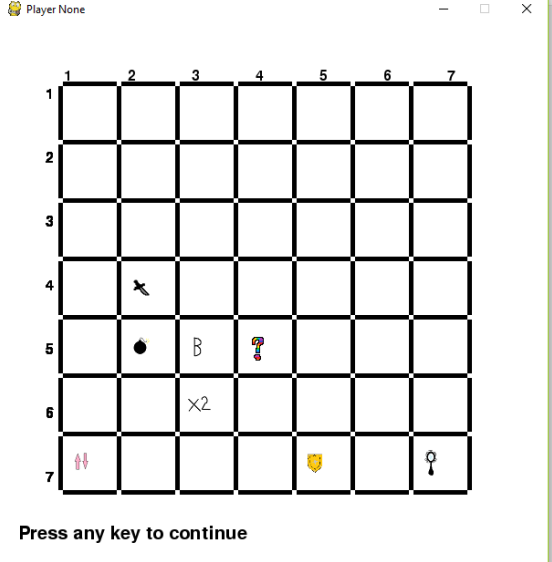
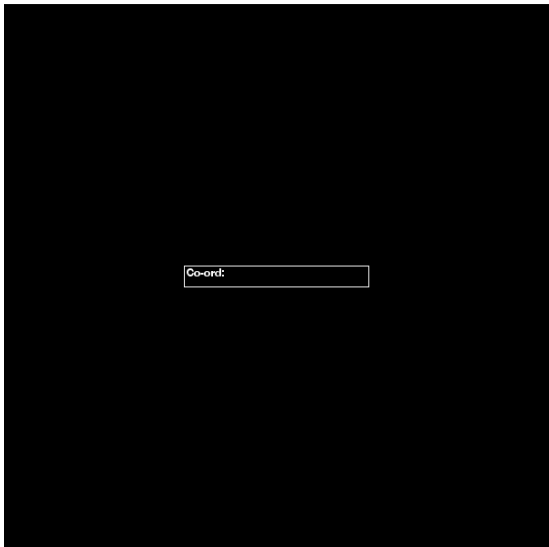


Asks for  
another co-ordinate

Step 4 – 3,5



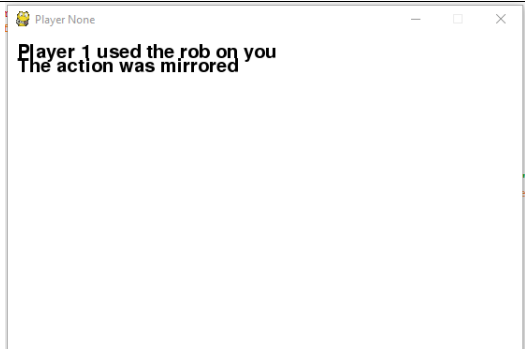
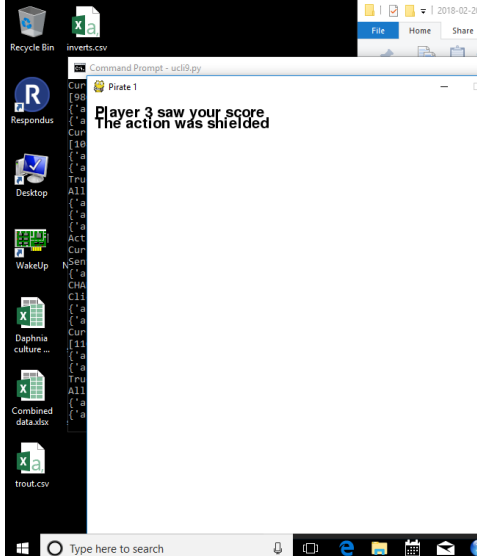
Accepted

Takes first number of input	34,678 (Accepted) Placed at 3,6	
	82,32 (Exceptional) Will be rejected	 <p>Asks for X co-ordinate again</p>

Performs the money actions between the rest of the actions	Will carry out the money tiles before any of the other action tiles.	<p>Even though rob is second in array, performed last</p> <pre>{'action': 'actiontodo', 'actionsarray': '1000', 'number': 1} {'action': 'actiontodo', 'actionsarray': 'rob', 'number': 3} {'action': 'actiontodo', 'actionsarray': '1000', 'number': 2}</pre> <p>These are the actions ['1000', 'rob', '1000']</p> <p>iteration 0  adding to total ←  iteration 1  adding to total ←  iteration 2</p> <p>Doing Rob ←  current scores before rob  [6200, 1400, 600]</p>
Rob	Will give the player robbing the current score of the player they attack.	<p>Current totals after action [13000, 13000, 12200]</p> <p>Sent to the network I am ready</p> <pre>{'action': 'changesquare', 'x': 2, 'y': 4, 'counter': 28}</pre> <p>CHANGING SQUARE</p> <p>Client action: <u>rob</u></p> <pre>{'action': 'changesquare', 'x': 2, 'y': 4, 'counter': 28} {'action': 'choose', 'not': 1, 'square': 'rob', 'shields': [], 'mirrors': []}</pre> <p>DATA HAS BEEN RECEIVED INTO THE CLIENT FOR ACTION</p> <p>Some button was pressed!</p> <p>The value of chosen is <u>3</u></p> <pre>{'action': 'choose', 'not': 1, 'square': 'rob', 'shields': [], 'mirrors': []} {'action': 'current', 0: 25200, 1: 13000, 2: 0}</pre> <p>Current points totals  [25200, 13000, 0]</p>
Kill	Will send the player who is attacked current score to zero.	<p>CHANGING SQUARE</p> <p>Client action: <u>kill</u></p> <pre>{'action': 'changesquare', 'x': 1, 'y': 1, 'counter': 20} {'action': 'current', 0: 1600, 1: 4400, 2: 6400}</pre> <p>Current points totals  [1600, 4400, 6400]</p> <pre>{'action': 'current', 0: 1600, 1: 4400, 2: 6400} {'action': 'current', 0: 1600, 1: 4400, 2: 6600}</pre> <p>Current points totals  [1600, 4400, 6600]</p> <pre>{'action': 'current', 0: 1600, 1: 4400, 2: 6600} {'action': 'choose', 'not': 1, 'square': 'kill', 'shields': [], 'mirrors': []}</pre> <p>DATA HAS BEEN RECEIVED INTO THE CLIENT FOR ACTION</p> <p>Action being performed kill</p> <p>Action being performed by: 1</p> <p>Some button was pressed!</p> <p>The value of chosen is <u>3</u> ←</p> <pre>{'action': 'choose', 'not': 1, 'square': 'kill', 'shields': [], 'mirrors': []} {'action': 'current', 0: 1600, 1: 4400, 2: 0}</pre> <p>Current points totals  [1600, 4400, 0]</p>

Swap	Will swap current scores of the attacker and the player they chose.	<p>Client action: <u>swap</u></p> <pre>{'action': 'changesquare', 'x': 1, 'y': 3, 'counter': 20} {'action': 'choose', 'not': 1, 'square': 'swap', 'shields': [], 'mirrors': [], 'actor': 1} DATA HAS BEEN RECEIVED INTO THE CLIENT FOR ACTION Action being performed swap Action being performed by: 1 Some button was pressed! The value of chosen is 3 {'action': 'choose', 'not': 1, 'square': 'swap', 'shields': [], 'mirrors': [], 'actor': 1} {'action': 'current', 0: 7200, 1: 8000, 2: 6400} Current points totals [7200, 8000, 6400] {'action': 'current', 0: 7200, 1: 8000, 2: 6400} {'action': 'current', 0: 7200, 1: 6400, 2: 8000} Current points totals [7200, 6400, 8000]</pre>
Bomb	Will send the player who has this tile's current score back to zero.	<p>Current totals after action [7200, 6400, 7200]</p> <p>Sent to the network I am ready</p> <pre>{'action': 'changesquare', 'x': 1, 'y': 6, 'counter': 10} CHANGING SQUARE Client action: bomb {'action': 'changesquare', 'x': 1, 'y': 6, 'counter': 10} {'action': 'current', 0: 0, 1: 6400, 2: 7200} Current points totals [0, 6400, 7200]</pre>
Double	Will double the current score of the player who has this tile.	<p>Current totals after action [3800, 7800, 2200]</p> <p>Sent to the network I am ready</p> <pre>{'action': 'changesquare', 'x': 2, 'y': 1, 'counter': 24} CHANGING SQUARE Client action: double {'action': 'changesquare', 'x': 2, 'y': 1, 'counter': 24} {'action': 'current', 0: 7600, 1: 7800, 2: 2200} Current points totals [7600, 7800, 2200]</pre>
Present	Will give 1000 points (not out of the 'attacker's' current score) to the player chosen by the 'attacker'	<p>CHANGING SQUARE</p> <p>Client action: <u>present</u></p> <pre>{'action': 'changesquare', 'x': 2, 'y': 3, 'counter': 36} {'action': 'current', 0: 2200, 1: 3600, 2: 2000} Current points totals [2200, 3600, 2000] {'action': 'current', 0: 2200, 1: 3600, 2: 2000} {'action': 'current', 0: 2200, 1: 3600, 2: 3000} Current points totals [2200, 3600, 3000]</pre> <p>The player awarding the present was:</p> <pre>{'action': 'number', 'num': 1} I am player number 1 {'action': 'number', 'num': 1}</pre>



Mag.Glass	Will show the score of the player chosen by the attacker.	See video 7
Mirror	Will allow all the actions to be mirrored for that round.	
Shield	Will allow all the actions to be blocked for that round.	

Wild Card	Will allow the user to choose an action to perform for that round.	<p>CHANGING SQUARE</p> <p>Client action: wild</p> <pre>{'action': 'changesquare', 'x': 1, 'y': 5, 'counter': 15}</pre> <pre>{'action': 'wild', 'choose': 1, 'square': 'wild', 'shields': [], 'mirrors': []}</pre> <p>Some button was pressed!</p> <p>rob</p> <p>.. . . . .</p>
Wild Card (pt 2)	Performs the action chosen by the user.	<p>Client action: wild</p> <pre>{'action': 'changesquare', 'x': 1, 'y': 5, 'counter': 15}</pre> <pre>{'action': 'wild', 'choose': 1, 'square': 'wild', 'shields': [], 'mirrors': []}</pre> <p>Some button was pressed!</p> <p>rob</p> <pre>{'action': 'wild', 'choose': 1, 'square': 'wild', 'shields': [], 'mirrors': []}</pre> <pre>{'action': 'choose', 'not': 1, 'square': 'rob', 'shields': [], 'mirrors': []}</pre> <p>DATA HAS BEEN RECEIVED INTO THE CLIENT FOR ACTION</p> <p>Action being performed rob</p> <p>Action being performed by: 1</p> <p>Some button was pressed!</p> <p>The value of chosen is 2</p> <pre>{'action': 'choose', 'not': 1, 'square': 'rob', 'shields': [], 'mirrors': []}</pre> <pre>{'action': 'current', 0: 1600, 1: 0, 2: 800}</pre> <p>Current points totals</p> <pre>[1600, 0, 800]</pre> <pre>{'action': 'current', 0: 1600, 1: 0, 2: 800}</pre> <pre>{'action': 'current', 0: 0, 1: 1600, 2: 800}</pre> <p>Current points totals</p> <pre>[0, 1600, 800]</pre>
5000	Adds 5000 to the users current score.	<p>Current totals after action [1000, 2600, 1800]</p> <p>Sent to the network I am ready</p> <pre>{'action': 'changesquare', 'x': 2, 'y': 5, 'counter': 17}</pre> <p>CHANGING SQUARE</p> <p>Client action: 5000</p> <pre>{'action': 'changesquare', 'x': 2, 'y': 5, 'counter': 17}</pre> <pre>{'action': 'current', 0: 6000, 1: 2600, 2: 1800}</pre> <p>Current points totals</p> <pre>[6000, 2600, 1800]</pre>
3000	Add 3000 to the user's current score.	<p>Client action: 3000</p> <pre>{'action': 'changesquare', 'x': 2, 'y': 6, 'counter': 0}</pre> <pre>{'action': 'current', 0: 3000, 1: 0, 2: 0}</pre> <p>Current points totals</p> <pre>[3000, 0, 0]</pre>
1000	Add 1000 to the user's current score.	<p>Current totals after action [3000, 3000, 3000]</p> <p>Sent to the network I am ready</p> <pre>{'action': 'changesquare', 'x': 3, 'y': 1, 'counter': 1}</pre> <p>CHANGING SQUARE</p> <p>Client action: 1000</p> <pre>{'action': 'changesquare', 'x': 3, 'y': 1, 'counter': 1}</pre> <pre>{'action': 'current', 0: 4000, 1: 3000, 2: 3000}</pre> <p>Current points totals</p> <pre>[4000, 3000, 3000]</pre>

200	Add 200 to the user's current score.	<pre> Action complete Current totals after action [4000, 4000, 4000] Sent to the network I am ready {'action': 'changesquare', 'x': 5, 'y': 6, 'counter': 2} CHANGING SQUARE Client action: 200 {'action': 'changesquare', 'x': 5, 'y': 6, 'counter': 2} {'action': 'current', 0: 4200, 1: 4000, 2: 4000} Current points totals [4200, 4000, 4000] </pre>
Progresses through rounds correctly	Goes to Round 2 once Round 1 has finished etc	See video 6
Waits for each player to get their shot	Allows the users to progress one at a time through their actions.	This cannot really be demonstrated, but this does occur, and the computer makes everyone wait until all the actions have occurred before showing the summary screen.
Add banks to the total of the player at the end	Adds the bank score for that player to their current score at the end of the game.	<pre> Action complete Current totals after action [7600, 3800, 1400] Sent to the network I am ready {'action': 'bank', 'scores': [36000, 0, 32200]} {'action': 'bank', 'scores': [36000, 0, 32200]} end score 43600 </pre>

### Server

Feature to be tested	Expected Result	Test Data (If required)
----------------------	-----------------	-------------------------

Connects to the server when correct IP address is entered	Adds one to the total players value each time someone joins	<p>Enter the ip address of the server. example: localhost or 192.168.0.2 Empty for localhost Server ip: 192.168.1.11 Server initialised</p> <pre> connection new connection: &lt;__main__.ClientChannel connected 192.168.1.11: address: ('192.168.1.11', 54409) Sent player their number total players 1 {'action': 'playready', 'playr': 1} connection new connection: &lt;__main__.ClientChannel connected 192.168.1.13: address: ('192.168.1.13', 50809) Sent player their number total players 2 </pre>
Communicates between the clients	Able to communicate between clients	See proof in video 7, example of magnifying glass over two computers
Updates the score after a round	Sends the updated scores back	Proof in the fact the scores are updated (seen in video 6)
Starts the game when 3 players have connected	Waits till all three clients send playready and then proceeds	<pre> new connection: &lt;__main__.ClientChannel connected 192.168.1.11:54409 at 0x5a address: ('192.168.1.11', 54409) Sent player their number total players 1 {'action': 'playready', 'playr': 1} connection new connection: &lt;__main__.ClientChannel connected 192.168.1.13:50809 at 0x5a address: ('192.168.1.13', 50809) Sent player their number total players 2 connection new connection: &lt;__main__.ClientChannel connected 192.168.1.9:51687 at 0x5a5 address: ('192.168.1.9', 51687) Sent player their number total players 3 {'action': 'playready', 'playr': 2} {'action': 'playready', 'playr': 3} {'action': 'ready'} </pre>

Waits for one player to choose someone for an action before asking another player	Only lets one person at a time choose a response to an action	<pre> { 'action': 'actioncode', 'actionsarray': 'present', 'number': 2 } These are the actions ['swap', '200', 'present'] iteration 0  iteration 1  iteration 2 ***** Swap scores Network action has returned to the server Summary screen info so far [['swap'], [3], [1], [False], [False]] Network Chosen event is complete {'action': 'chosen', 'playera': 3, 'playern': 1, 'tobedone': 'swap'} 1 Current score of the player attacking 23800 Current score of the player being attacked 16800 [False, False, False] In linear search Player is at position 1 After action score of the player attacking 16800 After action score of the player being attacked 23800 Swap score has been performed without shield or mirror ***** Present Network action has returned to the server Summary screen info so far [['swap', 'present'], [3, 1], [1, 2], [False, False], [False, False]] Network Chosen event is complete {'action': 'chosen', 'playera': 1, 'playern': 2, 'tobedone': 'present'} 2 Player sending gift 2 Player receiving the gift 1 Present has been performed End of Actions </pre>
-----------------------------------------------------------------------------------	---------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Modular Testing:

**Input:** Before integrating the input screens into the game, I tested the input code as a separate file, and checked all the validation worked.

**Main Game:** Throughout the progress of the game I have been testing the program. I started by making 2 by 2 prototypes. In each version of these I targeted two new action tiles and got them to work in that prototype before moving on to the next action tiles. These prototypes can be viewed in the folder, prototype proofs.

This allowed me to assess all the action tiles on a small scale and really focus in to make sure they all worked. I could also ensure that the mirror and shields worked against different actions and that game was proceeding as usual. Doing this made it much easier when I scaled up the game and made it 7 by 7. It also allowed me to start getting potential users to play the scaled down game and this allowed me to change and adapt my program to make it more usable at an early stage.

**Interfaces:** Before integrating any of the interfaces, I built them in separate python files which allowed me to assess how they looked before putting them into the code and let me make any necessary adaptations. This allowed me to check the key presses worked by letting them trigger dummy subroutines close the subroutines they would be calling in the game. I did have a slight problem with the original button interface at the start of the game. It was working before integration but then putting the file into the game resulted in the button interface not working, so I had to take it out and try to develop an alternative that worked in the game environment, and so I built in in the main program and did not develop it separately. Eventually I ended up with a keypress interface which took the users to the correct interfaces.

*Network:* Before starting trying to develop the game, I developed short pieces of the network code and tried to get a separate piece of code that would update a total on both players screens when either of the players pressed a certain key. This allowed me to test that this potential prototype successfully allowed players to connect and communicated between them. At this point, I integrated the network code into the game and started to build the rest of the game around this small bit of network code eventually expanding it to meet the needs to the program. By doing this it allowed me to feel confident in the base of the server and the communication between the clients. When I built the game up I checked that it was allowing all the players to take their turn, but initial tests showed that two players could be making a choice at the same time, so I built in code to make the server wait until one person had completed their turn before moving onto the next player. At all times in development the server let all the players have their turn.

Commented [MB37]: Code?

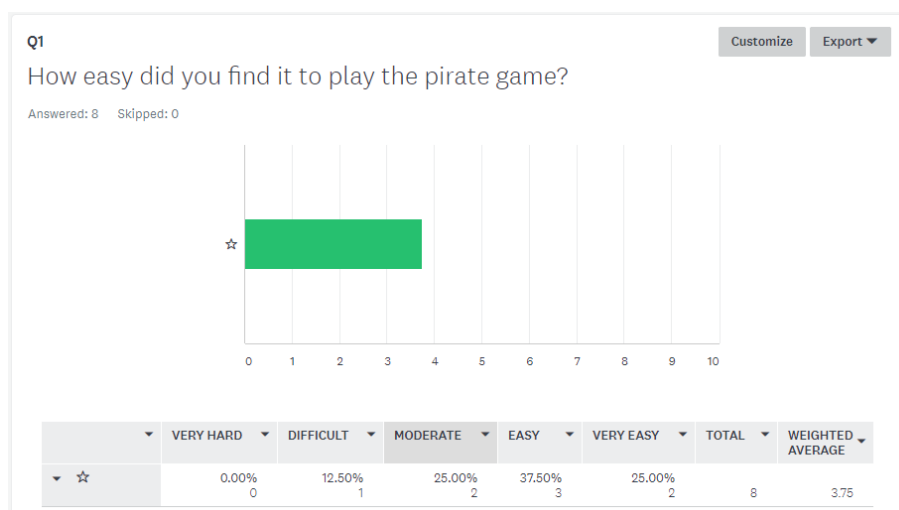
### User Testing Results

To assess how the usability of the project I created a survey for the end users to complete and so I could get feedback on how usable the project is.

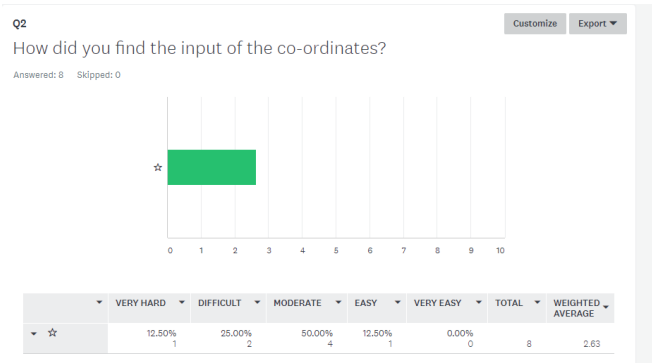
Survey:

<https://www.surveymonkey.co.uk/r/Q7T2LM7>

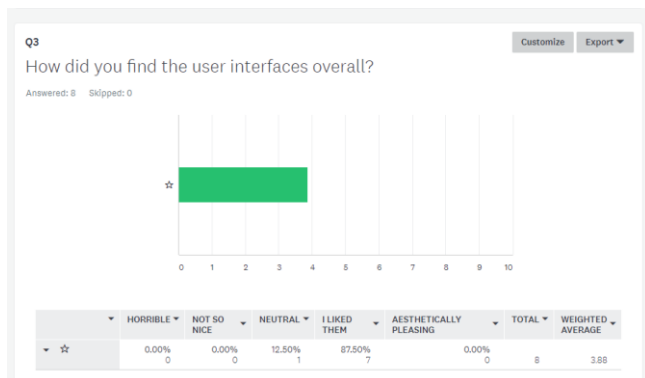
### Results from the survey



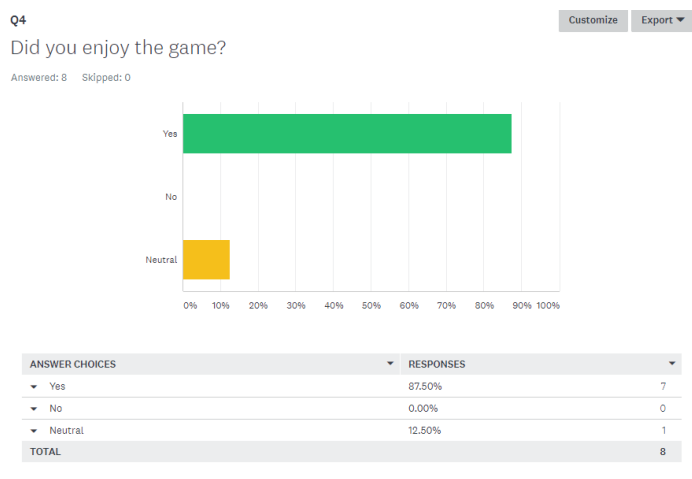
Overall most people found playing the pirate game easy or very easy.



Overall most people found the input moderate. I believe this is to do with the fact you have to enter all 49 of the tiles.



I was pleased to find that the clear majority liked the user interfaces.



Again, the clear majority enjoyed the pirate game.

### Possible Improvements:

Short cut to automatically input co-ordinates for 200s, as they have to fill remaining squares, and if you win the game I would like to know by how much!

2/24/2018 5:54 PM

[View respondent's answers](#)

1500 MFS!!!

nicknames

2/23/2018 1:43 PM

[View respondent's answers](#)

Automatic placement of 200s Input validation for coordinates Able to see grid during placement

2/23/2018 10:21 AM

[View respondent's answers](#)

### Favourite feature:

Seeing the grid when the game is happening

2/23/2018 10:21 AM

[View respondent's answers](#)

How the same game was being played by different people

2/21/2018 12:50 PM

[View respondent's answers](#)

I like the winning screen! as i like to win, and the wild card options.

2/24/2018 5:54 PM

[View respondent's answers](#)

### Least favourite feature:

If everyone gets points in the round the score is displayed very quickly and the co-ordinates can take a long time to put in. The action and symbol is not always clear, B for example.

2/24/2018 5:54 PM

[View respondent's answers](#)

Putting in coordinates is time consuming

2/23/2018 1:44 PM

[View respondent's answers](#)

Input of coordinates can be confusing to begin with

2/23/2018 1:43 PM

[View respondent's answers](#)

Entering the coordinates as it takes a while

2/23/2018 10:21 AM

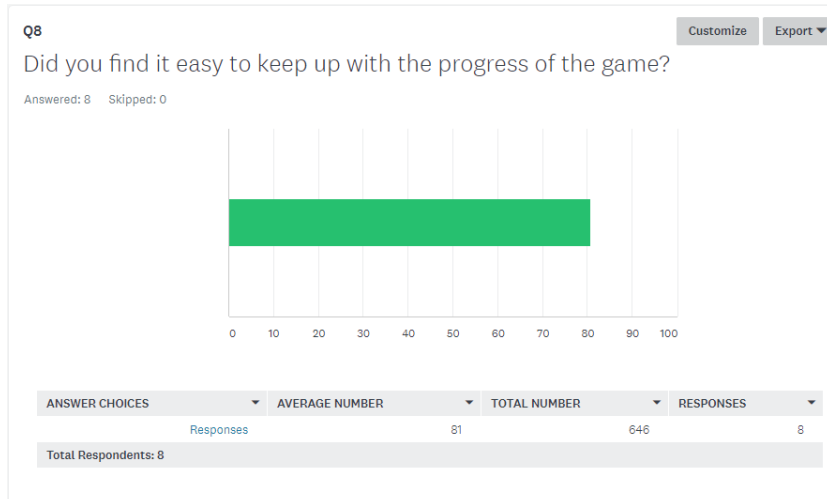
[View respondent's answers](#)

not knowing what player was who

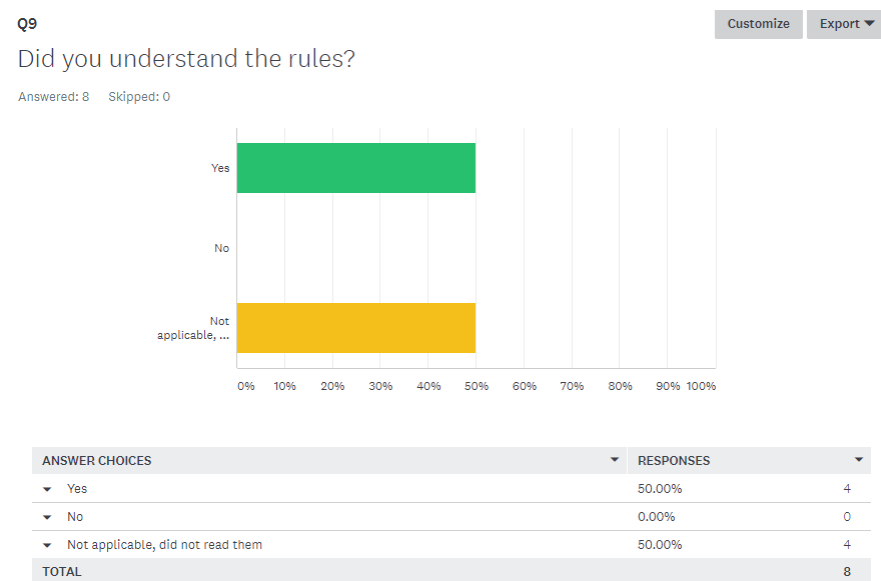
2/21/2018 12:50 PM

[View respondent's answers](#)

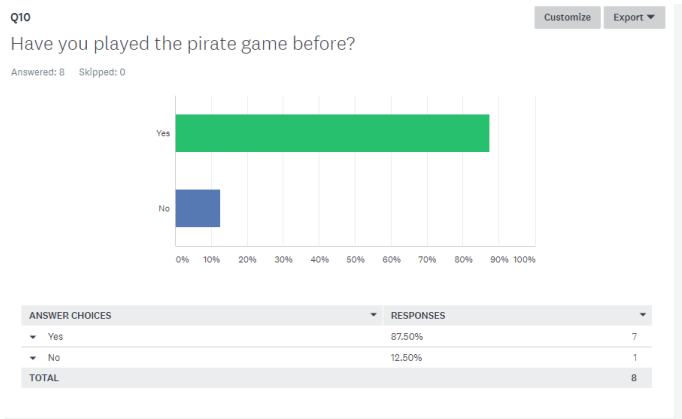




I am happy that users found It easy to keep up with the progress of the game.



Everybody that used the rules understood them, but half of the respondents did not need/use the rules.



Most users of my game had played the pirate game before.

### Final Corrective Maintenance

#### - Changing font size

During initial testing, I noticed discrepancies in the font sizes and that some of the sizes were too small and unreadable. To improve the overall look, I went through and modified the font sizes, so they were more constant and readable.

#### - Updating the grammar in the messages.

Again, throughout testing I noticed the grammar of some of the messages the program was asking was not correct and so I went through and added extra code to modify the messages produced.

#### - Winner Screen

One of the goals was always to have a separate 'special' screen for the winner of the game and so when I came across the star effect code online I decided to implement it at last minute as part of my perfective maintenance.

### Problems I won't have time to fix

#### - Auto- input of tiles

The input of all 49 tiles has been a feature that has been pointed out to me multiple times as a bit of an annoyance when playing the game. Initially the idea was to have an autocomplete function, as suggested in initial user survey but due to time restraints this was not possible. If at any point I came back to this project I would like to add this function of auto filling the grid in, as it would save time and improve usability. I have thought it through and have ideas for

implementation but due to time limits, this will not be added as a feature before the project is handed in.

- A problem with input (**Failed test case**)

During one of my test runs, a tester managed to break the input validation, and the game crashed when they entered a certain co-ordinate because they had a space before the number. Due to the game taking the first character of the input, the game took the value for the space and could not take the integer of it so crashed meaning we had to restart the server and all the games. We had to restart because the server did not recognize that the player had crashed out and still thought they were in input so even if everyone had finished, the game would have crashed out after input. The tester did not know the exact reason why the game had crashed but they reckoned they had entered a space before the number. Due to me not having input validation to deal with letters and space bars/enter etc the game will crash out if a user enters a letter. Due to the timing of the project I do not believe I will have time to fix this bug before handing in the project.

## Document 12 – Evaluation

Summarise your own testing

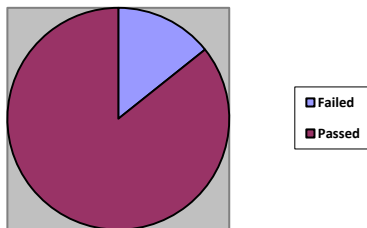
Overall the results of my own testing were positive. My own testing allowed me to spot potentially complicated interfaces that would be hard to understand and it allowed me to see areas that were developed to an acceptable standard and areas where I needed more work. This allowed me to go back to my code and fix any bugs I had spotted during development. Some unexpected results of my testing included finding an underlying problem in my code on the 21<sup>st</sup> of January, where I discovered the clients were not actually performing the correct actions for their grid. It was running through the actions in the wrong order, perhaps because of a deleted line from the day before or perhaps because I just hadn't written the correct code in earlier (though I believe I may have spotted this error earlier if this was the case). This was a massive bug and explained why parts of the code had been running funnily earlier in the days testing. Fixing this let me progress and be more confident that the game was allowing the users to perform the correct actions. Another unexpected result was related to the questions for each of the actions. I had built in code to allow the program to display different worded questions dependant on the action, but during testing it was not displaying the right questions for the actions, for example magnifying glass and present. This meant I had to go back and consolidate that the actions were coming in correctly. However it was, and the error was caused by a mistake in my coding regarding taking the strings of variables. Most of the positive test runs were when I was able to run full versions of each of the prototypes, including the 2 by 2 and the full 7 by 7. When I tested the full versions of each of those grids I became self-assured in the fact that all the underlying parts of the server and the client code were working. This allowed me to build on top of the code confident in the base.

To see an overview of the testing I have completed, please see  
<https://blogs.glowscotland.org.uk/fi/ahproject/category/testing/>

Test Cases (Post development – after main implementation, not before corrective maintenance)

Number of passed test cases – 14

Number of failed test cases – 2



Summarise key points from beta testers

Round 1 of Beta Testing:

- The first time I got a group of potential users to test my program it was a version of the code where every round if no action tiles were performed (only money tiles) it showed a screen saying '*no actions have been performed*' between each round. However due to the fact over half the board contains money squares this meant the game screens went through very fast and people struggled to know what their score was or what had happened in the game unless someone had an action tile. The main feedback at this state was to build in an interface to allow users to see their scores and slow the game down. I took this into account and built code that allowed the user to see their grid, score and round number if no action had been performed against them.

Round 2 of Beta Testing:

- The main comment in the second round of beta testing was about the input. Due to the nature of the normal pirate game, input involves users having to type in (at minimum, if they make no mistakes) 49 tiles, and this often is time consuming and annoys the users. If I had more time for development of the code I would build in an automatic placement of the tiles features. I had planned on doing this but due to time constraints I did not implement this in the end.

**Features of the game (meeting requirements specification):**

- Users connect to a server by entering an IP address into the command prompt
- Users are taken to a start interface after connecting to the server, which gives them the option to go to either the instructions interface or the game
- Users can view an instruction interface to read the rules of the game
- Users can enter the co-ordinates to place the action tiles/money tiles on their grid
- The program performs input validation on the tiles, making sure they are between the 1-7 range and have not been entered before
- Users can choose who to attack in relation to their current tile

- Money is added to the current total when a money tile is performed
- The current total is moved into the bank and the current is set back to zero on a bank tile
- Users can choose which action to perform when they have a wild card
- The program displays different interfaces to the winners and runner ups based on their current and bank total combines
- If the user wins, they will see an interface with a congratulatory message
- If the user does not win, they will see an interface displaying their score and the winners score

Overall the game meets the functional part of the requirement specification and contains all the features that I stated I was going to develop. However, my original requirements specification stated I was going to get input of the tiles by mouse control and that I was going to have a high score table. Neither of these functions are featured in my program currently.

#### End User Features

- Can access both the rules and the start of the game from the beginning interface
- Able to enter co-ordinates via keyboard of the tiles to be placed on their grid
- Able to choose which player to attack in relation to their action tiles
- Able to choose which action to play as part of the choose next square
- Able to see interfaces at the end displaying their score and the winner's score (if runner ups) or a congratulatory message (if winner)
- Can play from separate computers

The game matches all the end user parts of the requirement specification. Based on the results from the usability survey I also received feedback that the game was easy to play, and people could navigate through the game with relative ease, so it met that end user requirement too. The game is as close to the original game as I could make it within the time frame, the only adaptations I had to put in place were:

- Shield/Mirror only lasting one round
- Choose Next Square having slightly different function
- Number of players

These adaptations do not have a big effect on the playing of the game and I do not think any of these changes were major changes.

#### Analysis:

- User Interface  
One improvement I would make to my user interfaces would make it easier to read the instructions. The instructions currently is just an image of a text file with the simple steps to play the game, however it has been commented that the screen could be more simply structured to make it easier to take in and understand. Another improvement I would make to my user's interfaces would be the end screen for the runner ups. I would change the end screen slightly to display the scores of all three players instead of just the current player and the winner's score. This does not bring any benefit to the game, it is just an extra feature that could be added.
- Robustness

The main improvement I would make to the robustness of my program definitely comes down to input validation and the way it deals with characters.

Unfortunately, my input validation does not cover what happens when the user enters letters and symbols such as space, enter or /. When a user does enter one of those characters the game crashes out, giving the input error of *'cannot take the integer of the co-ordinate'*. This was highlighted to me at multiple points during my own testing and also in a failed test case on the 23<sup>rd</sup> of February. I did slightly improve the robustness of the program by building in code that would not let the user enter an empty co-ordinate/input. The reason why the game crashes out is because during the validation of co-ordinates I have to take the integer of the entered co-ordinate and so this causes the game to crash out and the PyGame display to disappear. However, even though the display is gone and the user cannot play the game anymore, the server does not recognize that the user's game has gone down and so the game has to be started from the beginning again regardless of how far along the other users are with input, due to it not removing the player from the server. Fixing this and adding in code to deal with letter input would make the code more robust.

- Reliability

The game overall is fairly reliable. It produces expected results and performs all the actions correctly each time it is played. Money squares work correctly, and the bank score is added to the score at the end of the game. All the functions of the game work correctly.

In one test case the game crashed out after reaching magnifying glass, but this could not be replicated, and it was before I had sorted out the introduction screen, so this could have been a factor in the reason why it crashed. After multiple attempts at re-running the game and looking through the code, no reason could be found for why it crashed.

As far as I am aware there is no improvements I could make on the reliability of my code.

- Portability

To increase the portability of code, I could have made code to mathematically figure out the size of the screens in relation to the computer it was played on. At the moment the screen is a fixed size which is smaller than most of the screens of computers, but I could change code to allow the screen size to adapt to the computer.

Another way I could increase the portability of code is to decrease the amount of files needed to play the game. At present, you need a large number of image files and then two additional folders to play the game. If one of these things is missing the game will not work.

One way I increased the portability of my code, is these three lines

```
#setting dir_path to the directory path of the current file
dir_path = os.path.dirname(os.path.realpath(__file__))
#getting the current working directory
cwd = os.getcwd()
#changing the working directory of the file
os.chdir(dir_path)
```

These lines mean I can have the python files in a separate location from the python command shell and still run the game, meaning it is easier to move around computers without having to place it in the command shell folder.

- Efficiency

The size of the client and the server python files combined are 100 KB. To run the game however you must also have PodSixNet installed (as well as python and PyGame) and have all the 52 images required to run the game.

I used a bubble sort to sort the array of all the co-ordinates into an order based on the random number assigned to them. Bubble sort has an average and worst case running time of  $O(n^2)$ . It is not the most efficient sort algorithm I could have used and so by choosing another sorting algorithm such as quick sort I could have increased the efficiency.

- Maintainability

I would say that my code is maintainable. I have included comments throughout my code that explain what is happening and have summarised the functions of all the subroutines using comments. I have named my variables/arrays/classes descriptive names that relate to the data they are storing, which improves maintainability. The code overall is written in a modular fashion and most of the code is within subroutines, with the exclusion of setting up global variables and asking for the IP address, meaning my code was easier to understand and the code can be taken and used in other projects.

However due to the complicated structure of the server and the clients, I have lost some degree of maintainability because changing one thing may have a major effect on the data sent between clients and the server. I have found this to be a problem throughout implementation as for example trying to build in end interfaces took me a lot longer than expected due to the server code not passing through data successfully.

I would like to hope that my code is readable and easy to understand.

- Input Squares

One piece of perfective maintenance I would like to do possibly in the future is the auto-completion of input. Many end users have commented that the input takes a lot of time, and it does as it requires the user to input 49 tiles. With the addition of a few more lines of code, I could make the program automatically place the 200s (which seem to be the part of input people get most frustrated with). This would increase robustness (as less reliance on input validation) and make the input process easier for the players.

### **Evaluation of development process**

Developing this project was one of the longest tasks I have overtaken so far in school. It spanned from the end of September to the end of February and required a lot of dedication. Time wise, I managed to stay pretty well on schedule, though throughout the process there was points where I had to focus more attention on other things that were happening in school and outside, such as university applications/prelims etc. This obviously meant that the progress of the project at these times was slowed. Going back, if I could change one thing it would be the amount of progress I made in the earlier

months of the project. Although I had to focus most of my time on other areas I would have benefitted from doing some more research/initial coding at that stage, so I didn't have to push so much on with the project in early 2018.

The development methodology I used to complete my project was a combination of both the waterfall method and agile, taking the best aspects from both. I had to do a lot of writing up and planning before starting the coding process but once in the coding process, I was working to produce prototypes of different aspects of the game. This methodology was useful for this project because I needed quick solutions to allow me to think how all the different components of the game were all going to fit together and how the different interfaces were interacting. I think this was the best choice considering my inexperience with PyGame and PodSixNet, as it allowed me to have more chance to explore different possibilities than the waterfall model would have.

I think one thing that went well in the development process was the development of the game on top of the server. Once I had built the server, it took me a lot shorter time than expected to build the game. A lot of the actions followed the same pattern and required the same communication between the server and the client so once I had one action completed, I could follow similar methods to implement the other actions. This meant I could build in a lot of the actions in the space of two weeks meaning I could make progress on other areas of the code sooner than expected.

One thing in the development process that could have gone better was my backup of files. Early in February, I lost a USB stick and then my next stick became corrupted. Although I lost most of the documentation I had done (over 50 pages!!), it taught me the importance of backing important files up in separate locations. Throughout the process I had been taking regular versions/backups and I learnt from the experience of losing the USB stick that it is important to backup documents as well as the program.

#### **Knowledge and Skills that I have gained**

- Experience in PyGame  
Before coming in to the project I had never coded in PyGame before. This meant I had to go on a steep learning curve, but I have picked up techniques and code that I can use in the future for other programs. This additional knowledge will help me developing games in PyGame in the future as I am coming at the games with a lot more practice and understanding of how the library works.
- Experience and skills in multiplayer games  
Like PyGame, I had developed no multiplayer games in Python (or any programming language) at the start of the project. I knew this was going to be hard part of my code to implement and I reviewed multiple ways to make the game multiplayer before settling on PodSixNet. In the future, if I was going to develop another multiplayer game then I would choose to use the PodSixNet library as I feel I am confident in using it to set up a server and a client. By developing a multiplayer game, it has exposed me to more complex code and I have gained a lot of skills and understanding of harder concepts.
- This is the first large scale project I have worked on and it has allowed me to increase my skills in scheduling, documentation of code and writing larger sections of code. This experience will help me as I progress to further education.

#### **Evaluation of own performance**

One thing I have learnt about myself from this project is that sometimes it is good if I push myself out of my comfort zone in projects like these. I have never been someone



who has played a lot of games and I didn't really like the idea of developing a game for my project but through making the pirate game I have learnt a lot of information about programming in general and what goes on behind the scenes of multiplayer games. Developing a multiplayer game is not necessary a project I would have envisaged doing but it has brought on my programming skills a lot and has made me more confident in Python and PyGame, hopefully two languages that I will use again in my future. A second thing I have learnt about myself from this project is that sometimes it is best that I take breaks. Throughout the project I often got to parts in the code where I had reached a problem, and I didn't know what to do. Although I wanted to continue to push on, I learnt that it is best that sometimes you step away from the problem and take a break before coming back and trying again. This will allow me to tackle problems with a fresh look on the code which will possibly save me development time and also ensure that I do not get frustrated with the code and start going round in circles. Apart from the USB stick incident, I feel over all my organisational skills have been good throughout the project. I kept multiple copies and versions of the code and I organised my folders and paperwork, so it was easy to navigate between, allowing me to find essential documents when needed. This organisation sped up development, but I feel next time I take on a project of similar size to this I need to remember to take more backups and organize my computer/file storage a lot better, because printed and physical documents were organised but sometimes I struggled to find documents on the computer. My time management skills throughout the project were good, as I managed to stay on schedule for the end date of February. This allowed me plenty of time to write up the essential documents and collate all the coursework before handing it in. Even with losing my USB stick I was not pressured for time and had enough time to write up the documents I had lost. If I did it again however, as I stated above I would spread the work out as I felt I did the bulk of the implementation work early January. I was able to take feedback and advice on-board throughout my project which allowed me to make necessary adaptations to my project, so it would be more usable overall, and people would enjoy playing the game more. The feedback really helped me throughout the project as it picked up on things I necessary wouldn't and gave me an outside view of the game from people who hadn't developed it. This gave me extra information and more things I could modify, and I thought I took the feedback on well and made sure I made the necessary adaptations.

## Document 13 – Bibliography

- Chandrasekar, A. (2015). *Understanding syntax in Python*. [online] Stackoverflow.com. Available at: <https://stackoverflow.com/questions/31721850/understanding-syntax-in-python> [Accessed 16 Feb. 2018].
- Briers, M. (2017). *Bubble sort not producing desired output*. [online] Stackoverflow.com. Available at: <https://stackoverflow.com/questions/46787230/bubble-sort-not-producing-desired-output> [Accessed 17 Oct. 2017].

***Below is a link to the source of code I took and modified to make my Input Box for inputs of the co-ordinates in the game:***

Downs, T. (2002). *Input Box Module*. [online] PyGame.org. Available at: <http://www.PyGame.org/pcr/inputbox/> [Accessed 9 Oct. 2017].

McCormick, C. (2009). *chr15m/PodSixNet*. [online] Available at: <https://github.com/chr15m/PodSixNet> [Accessed 1 Jan. 2018].

***Below is a link to the source of code I took and modified to make the buttons for the action tiles:***

Larsen, S. (2013). *Button Drawer: Python 2.6 – Simon Hjortshøj Larsen*. [online] Simon Hjortshøj Larsen. Available at: <http://simonhl.dk/button-drawer-python-2-6/> [Accessed 3 Jan. 2018].

McGrath, M. (2013). *PYTHON IN EASY STEPS*. Leamington Spa: IN EASY STEPS LIMITED.

Meyer, J. (2013). *Multiplayer Game Programming for Teens with Python: Part 1*. [online] Ray Wenderlich. Available at: <https://www.raywenderlich.com/38732/multiplayer-game-programming-for-teens-with-python> [Accessed 8 Oct. 2017].

***Below is a link to the source of code I took and modified to make the effect for the winner screen:***

Neto, S. (2009). *PyGame, Simple Space Effect*. [online] silveira neto. Available at: <http://silveiraneto.net/2009/08/12/PyGame-simple-space-effect/> [Accessed 4 Feb. 2018].

PyGame.org. (2018). *PyGame Front Page — PyGame v1.9.4.dev0 documentation*. [online] Available at: <https://www.PyGame.org/docs/> [Accessed 19 Dec. 2017].

Pythonprogramming.net. (n.d.). *Python Programming Tutorials*. [online] Available at: <https://pythonprogramming.net/PyGame-buttons-part-1-button-rectangle/> [Accessed 18 Jan. 2018].

Stackoverflow.com. (2015). *Creating a Multiplayer game in python*. [online] Available at: <https://stackoverflow.com/questions/30227722/creating-a-multiplayer-game-in-python> [Accessed 7 Jan. 2018].

Kinsley, H. (2014). *Game Development in Python 3 With PyGame - 11 - Buttons p. 1*. [online] Available at: [https://www.youtube.com/watch?v=jh\\_m-Eytq0Q](https://www.youtube.com/watch?v=jh_m-Eytq0Q) [Accessed 17 Jan. 2018].

***I give credit to the following people for the code I modified to make features of my game:***

- ***S Neto***
- ***Simon Larsen***
- ***T Downs***

I made sure that I was able to use the code for my game before using it.

## Document 13 – User Guide

The Pirate Game is a multiplayer grid-based game, that requires three players to play. The aim of the game is to collect the most money out of all the 'pirates'. Each player has a grid of actions and each round one of the actions is performed, and this can result in the player losing or gaining money.



This game requires Python 3 at minimum.

### Launching the game:

Before starting the game, please ensure all the following files are present within one folder/in the same location –

*Rob.png*

*Robactive.png*

*Robdone.png*

You should have the three types of files for the following images:

Rob, Kill, Swap, Shield, Mirror, Bomb, Present, Bank, Double, Magnifying Glass, Wildcard, 5000, 3000, 1000, 200

You should also have the following images:

*Introscreen.png, instructions.png, square.png, logo.png, normalline.png*

And the following folders/files:

*Buttons.py*

*PodSixNet*

*PyGame*

*Space.py*

*Client.py*

*Server.py*

Now you are all set to play the game! The game is best run over 3 or 4 computers and by 3 people. It is not recommended that you try and run more than one client on the same computer.

### Steps on how to start up the game

1. Run the *server.py* file. It should allow input of a IP address to set up the server on. Please enter the IP address of the current computer you are running the file on. Once it says '*Server initialized*' the clients are now ready to be run.
2. You are now able to start up the clients. Now run the *client.py* file. (Side Note: Running it in Python means an indentation error pops up, try running it from the

python command prompt/command prompt) (Side Note 2: Do not attempt to run the client/server on the same IDLE shell, it will mean the server will be shut down)

3. A blank/black PyGame window will pop up, go to the command prompt/the input for whatever application you opened the file in and type in the same IP address that the server is set up on.
4. This will mean an intro screen will now pop up on the computer, and you are ready to start the pirate game. Press play when ready to start input, it is not relying on other players being connected. Do not be alarmed if the caption of the PyGame window is Pirate None. This may just be it hasn't updated the player numbers yet, and you can check you have connected to the server by looking at the output from the server in the command shell/whatever you opened it with.
5. Once three players have connected and all finished their input, the game will begin!

If a bug occurs and one of the client's crashes, **PLEASE RESTART THE WHOLE GAME AND THE SERVER**. You must do this because the server may still think that all clients are functioning properly and so will not start until the player who has crashed has finished input, which they cannot.

Enjoy the Pirate Game!