# IBM Social Game 2018 - SpaceRaze

Clive Wong Vincent Ooi Zeqian Cao Dharshana Lakshmanan
Pallavi Hrisheekesh Benedikt Kolbeinsson
Adilzhan Nussipzhan

Electrical and Electronic Engineering Department
EE3-DTPRJ Design and Build Project (2017-2018)

June 30, 2018

# DOCUMENTATION

# Contents

# 1 Introduction

## 1.1 Project Aim

Despite the significant technological leaps that the modern world has experienced in recent decades, people are more interested in the development of new board games than video games. This can be seen in Figure 1 below, which shows how much more successfully new board games are funded than video games.

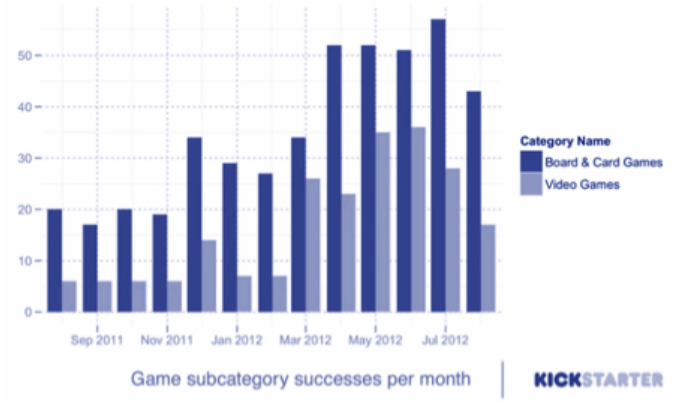


Figure 1: Board games versus video games

It is currently the golden age of board games. One cause of this is the barrier to entry for video games, as it has a longer initial set up and a steep learning curve. Additionally, most video games do not cater to the needs of people with disabilities or those who are unfamiliar with computers or video game consoles.

Nonetheless, board games also have their drawbacks. As most do not utilise modern technology, they may not be as immersive and captivating as video games. Furthermore, board games cannot be played with friends and family who live in different cities or countries. The aim of this project is to create a game that incorporates the immersive aspects of video games with the sociability of board games while also being inclusive of everyone.

## 1.2 The Client

IBM is an American technology company that operates all over the world. They produce computer software and hardware, and are also one of the biggest research companies in the world. They have developed an AI (artificial intelligence) called Watson. It is already being used in healthcare, fashion, and even weather forecasting. IBM proposed this project in order to discover new applications of their AI, as well as to demonstrate its flexibility and ease-of-use.

## 1.3 Customer Requirements

IBM outlined a project to make a technology driven two-player space game with less of a barrier to entry than online or PC games, using IBM Watson services. The game interface involves each player having a model spaceship, of which he or she is the captain. The objective is to find and destroy all other players' ships. The following lists the criteria outlined by IBM for the ship:

- Accept voice commands to move forward, turn right or left, shoot, or shield.
- Physically turn the model ship left or right when it hears the respective voice commands.
- Have a radar system, which detects how close the other ship(s) are.
- Show damage by the colour of the LED corresponding to 'health points' for each respective ship compartment.
- Use Watson-Text-To-Speech to anthropomorphise the game.
- Have audible engine/attack noises for game immersion.

The game is hosted on IBM cloud. The following lists the criteria outlined by IBM for the digital element:

- Place ships on a 2-dimensional grid, where each ship uses up a 2x2 square space.
- Update the positions and health points according to the player's voice commands.
- Return this data to the physical ship so they can implement the radar system.

### 1.4 Prerequisites and Services Used

The project requires the knowledge of implementing a full stack web application. It is also necessary to read through the documentation to understand how to use API provided by IBM Cloud to integrate Watson services, and host the application on IBM Cloud. The IBM Cloud services required for this project are:

1. Text to Speech Service - convert text to audio output (such as .wav format)
2. Speech to Text Service - transcribe the user's speech to text
3. Personality Insights - analyse the user's personality based on tweets
4. Hosting - host the web application
5. Watson Assistant - design dialogue flow to converse with the user

On the other end, one needs to be proficient in Linux and analogue circuits design to integrate Software and Hardware of the Raspberry Pi as it runs on Raspbian Stretch OS. Therefore, most configurations need to be done through the Linux Terminal.

## 2 Management

### 2.1 Operation Management

The nature of the project requires extensive research and learning as the team had limited knowledge of full stack web application implementation. To ensure smooth and effective communication, the team decided to work on the project together daily from the morning until the evening. This allowed efficient communication for troubleshooting. In addition, due to the consistent workload and constant exposure to everyone's work, the project progressed according to schedule. When working from home, the team communicated via Slack and shared all the files via GitHub/Google Drive.

### 2.2 Milestones and Timeline

The following Gantt Chart, shown in Figure 2 represents the timeline and milestones the team set as well as the individual contribution.

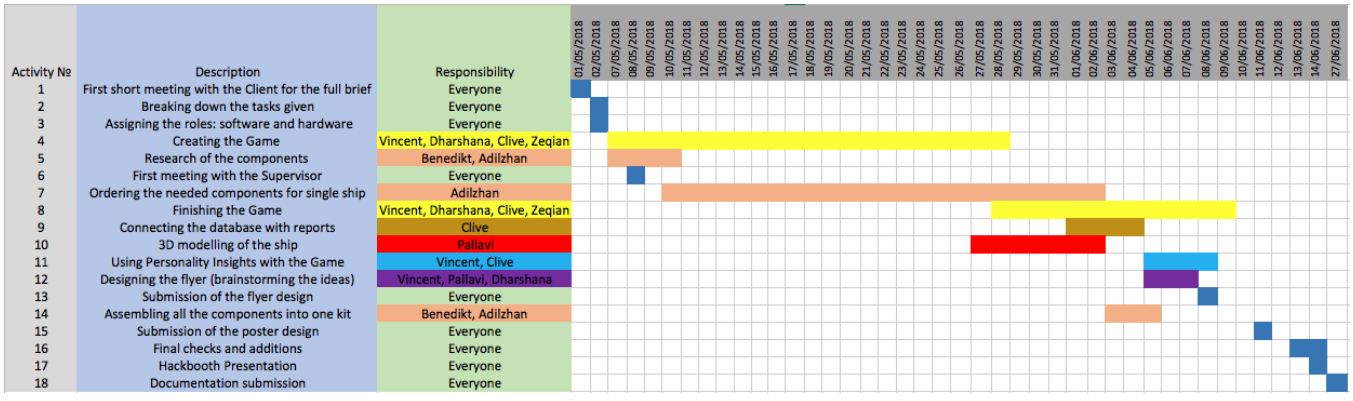


Figure 2: Timeline of the distribution of tasks

Following the Gantt Chart, it can be seen that the team delegated the tasks and had inner deadlines for specific tasks. As aforementioned, the team met every day and worked together to minimise communication issues. The

details of the agenda mentioned in the Gantt chart are documented in the **meeting minutes** attached in the Google Drive, where its link can be found under Section 3. A blog was also made to document the team's weekly progress which can be accessed via this link https://spaceraze.wordpress.com/.

### 2.3 Breakdown of the tasks

The team spent considerable amount of time discussing the game and its implementation as it determined how the tasks could be allocated. After the system design plan is made, the team was split into three sub-teams - software, hardware and 3D modelling teams. As this project was heavily software oriented, the software sub-team consisted of more members. 4 people were allocated to software team, whereas the hardware team had 2 people followed by 1 member responsible for 3D modelling.

Software team had to learn JavaScript and Web App development to build the game from the scratch. Hardware team was responsible for designing circuits that would operate smoothly and as expected through General Purpose Input/Output (GPIO) pins on the Raspberry Pi. To model the 3D version of the ship, Fusion 360 was used as it proved to be more user-friendly and easier to manage.

### 2.4 Expenditure

The total budget given was £500, which could be extended if needed. The total expenditure on the components used is shown in Table 1. It should be noted that the components ordered from RS Components and Farnell have a discount of 19% on them. The total **net spending** was £**254.796**, which is within the budget given.

Table 1: List of components

| Company | Components | Price,£ | Amount | Total price per component,£ | Total price per component with discount,£ |
|---|---|---|---|---|---|
| RS | Raspberry Pi 3B+ with microSD | 38.38 | 2 | 76.76 | 62.1756 |
| RS | Speakers | 1.42 | 12 | 17.04 | 13.8024 |
| RS | LEDs | 1.438 | 10 | 14.38 | 11.6478 |
| RS | USB to microUSB | 3.21 | 1 | 3.21 | 2.6001 |
| RS | AA battery holder | 2.84 | 4 | 11.36 | 9.2016 |
| RS | HDMI to DVI | 8.65 | 1 | 8.65 | 7.0065 |
| RS | Stepper Motor | 23.96 | 2 | 47.92 | 38.8152 |
| RS | HDMI to VGA | 17.29 | 1 | 17.29 | 14.0049 |
| RS | USB Dongle | 7.1 | 1 | 7.1 | 5.751 |
| RS | Power bank | 25.45 | 2 | 50.9 | 41.229 |
| EEEStores | AA batteries | 0.36 | 12 | 4.32 | 4.32 |
| EEEStores | Breadboard | 2.98 | 2 | 5.96 | 5.96 |
| EEEStores | Battery 9V | 1.98 | 2 | 3.96 | 3.96 |
| EEEStores | 9V holder | 0.22 | 2 | 0.44 | 0.44 |
| EEEStores | Superglue | 1.35 | 4 | 5.4 | 5.4 |
| EEEStores | Cable ties | 0.02 | 10 | 0.2 | 0.2 |
| Farnell | Demultiplexer | 0.408 | 4 | 1.632 | 1.32192 |
| Rapid | Stepper motor driver | 7.18 | 1 | 7.18 | 7.18 |
| Amazon | Earpods | 19.78 | 1 | 19.78 | 19.78 |

## 3 Source Code Management

To collaborate the team's work efficiently, the following software services were utilized:

1. **Google Drive** was used to share large documents, mainly:
    - Hardware design files – Circuit diagrams, schematics and pictures of breadboard connections
    - 3D printing design files
    - Audio files – Sound effects used during the report phase of the game

- Meeting Minutes

Other files related to the group project submission were also shared such as the leaflet and poster designs. The drive can be accessed with this link- https://drive.google.com/open?id=1MRqLoFitqGOkA3CwrAaqkDaoJFeN6XFE

2. **GitHub** was used to code collaboratively since the software team consisted of 4 people. Several branches were also created in order to not disrupt the main working code. Each branch's code was then tested separately and then merged with the master branch. The GitHub repostitry can be accessed via the following link- https://github.com/CliveWongTohSoon/IBMSocialGame.

3. **Slack** was used for day to day communications to ask other team members for their progress and updates on their work. It was also used to discuss and troubleshoot any issues encountered. Since the team met frequently, most of the communication was verbal and then subsequently jotted down in the minutes of the meeting.

4. **WebStorm** and **Visual Studio Code** were used as the main IDEs so that the style of the code can be formatted similarly for ease of readability.

# 4 Product Design

## 4.1 Game Design

The project is intended to play out as an online duel-style game, wherein each player acts as the captain of a ship. The game is played on a virtual unseen grid, with the win condition being to seek and destroy one or more enemy ships, and be the last surviving ship. The game will be time-limited and turn based. Each turn allows players to give a maximum of 3 instructions within 60 seconds, at the end of which each instruction will be executed sequentially, in parallel with all other players in the game. After each turn, the game enters a report phase, during which players will be given feedback on the consequences of their actions for that turn. Game inputs come in the form of voice commands through a microphone, and game outputs come through external speakers, as sonar pings or report feedback from the virtual crew, and LED lights on the ship model.

There are a total of 4 different types of actions that a player may choose:

1. Shield + Direction - in which a player may reduce any damage incoming from a chosen direction for the remainder of the turn

2. Shoot - in which a player does damage to any enemy ships directly in front of them

3. Move - in which a player moves their ship forward one unit of measurement

4. Rotate + Direction - in which a player turns their ship 90 degrees clockwise or anticlockwise

The actions above are displayed in order of priority, such that a player who chooses to use Shield for their first action will have that action executed before a player that chooses their first action to be Shoot. If both players choose the same action, then the consequences of their actions are only processed after both actions have been executed.

The ship is separated into 4 departments on the virtual grid, occupying a 2x2 square on the grid. The 4 departments are separated into left or right weapons, and left or right engines. In the occasion that neither left nor right weapons are considered alive, the player may not choose Shoot as any of their actions in a turn. In the occasion that neither left nor right engines are considered alive, the player may not choose Move as any of their actions in a turn. The player is considered to be defeated if all 4 departments are no longer considered alive and is then removed from the game.

Each ship is customised to the player's personality by adjusting several ship parameters. This allows for different styles of play based on individual ship parameters. The following is a list of adjustable ship parameters:

1. Sonar range - the range at which a player can detect the location of an enemy ship

2. Weapons range - the maximum distance which a player can damage an enemy ship if Shoot action is used

3. Attack - the damage done to an enemy ship before reductions if Shoot action is used

4. Defence - the amount of reduction from damage taken from enemy ships and collisions if a player's shields are activated

5. Health - the maximum amount of damage a ship department may take before being considered to be no longer alive

The algorithm that calculates ship parameters from personality scores is intended to encourage and reward a balanced personality, giving a score of 50% as the optimum value, with each score affecting two separate ship parameters.

## 4.2 System Design

The game system will be separated into three main parts, the web application running on IBM Cloud, software running on the Raspberry Pi and the hardware system. The web app consists of a starting website that serves as the pre-game interface, for players to log in to their twitter accounts, join games and indicate when they want to play. It also functions as a database and server, storing data, executing game functions and (indirectly) communicating with all hardware of the player in the game. The hardware system is the input/output interface, which is controlled by the software running on the Raspberry Pi. Figure 3 is a flowchart representation of how data is communicated within the system, as well as how the system enters different phases within the game.

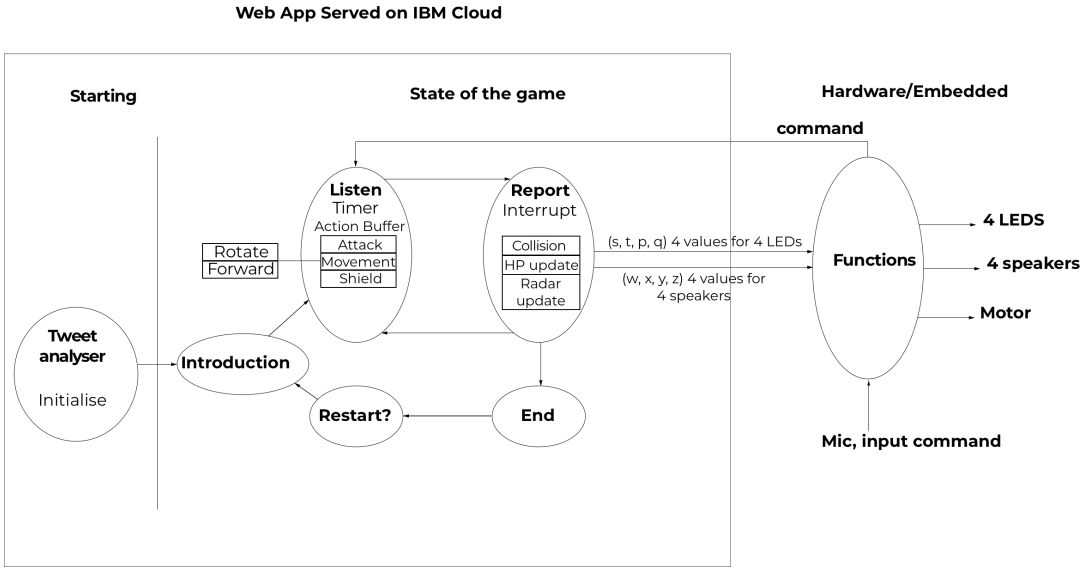


Figure 3: System Design Flowchart

## 4.3 Hardware Implementation

The first and most important hardware decision was which microcontroller to use. The processing needed to be performed on the microcontroller was minimal but it had to be able to connect to Wi-Fi (IEEE 802.11). Other requirements included output pins to control all the LEDs, the motor and the speakers as well as connecting to a microphone. To make this decision, a short list of affordable and compatible microcontrollers was created. A shortened version of this list can be seen in Table 2.

Table 2: Microcontroller Comparison

| **Microcontroller** | **Pros** | **Cons** |
|---|---|---|
| Raspberry Pi Zero | - Great for prototypes<br>- Inexpensive<br>- Meets all requirements | - Out of stock |
| Raspberry Pi 3B+ | - Great for prototypes<br>- Meets all requirements | - Expensive |
| Arduino Mega | - OK for prototypes<br>- Meets most requirements | - Needs separate chip for Wi-Fi and bluetooth<br>- Expensive |
| Arduino Uno Wifi rev2 | - Great for prototypes<br>- Meets all requirements | - Out of stock |
| ATtiny and ATmega | - Similar to production<br>- Cheap | - Not good for prototypes<br>- Needs separate chip for Wi-Fi and bluetooth |

The team decided on the Raspberry Pi 3B+ as it would be great for the prototype and met all the aforementioned requirements.

For the four LEDs the Kingbright 465 / 525 / 630 nm 3 RGB LED consists of three LEDs in one package (red, green, blue). Our prototype only needs the red and blue LEDs but future plans include the usage of all colours.

RS Pro 32Ω 1W Miniature Speaker was selected as it is small and quite loud. For the demultiplexer, the CD4051BE chip was selected as it is cheap and reliable. The motor selected was the RS Pro Hybrid while the motor driver was the A4988 Pololu. This motor fit the size requirements of the prototype (42.3 x 42.3mm) and does not require a gear system. For the actual product a cheaper 5 V stepper motor could be used but would require a gear system.

To summarise, the hardware used for each ship are:

- Raspberry Pi 3B+ × 1
- RS Pro 32Ω 1W Miniature Speaker × 4
- CD4051BE demultiplexer × 2
- RS Pro Hybrid motor × 1
- A4988 Pololu motor driver × 1
- Kingbright 465 / 525 / 630 nm 3 RGB LED × 4

#### 4.3.1 Circuitry

It is important to note that circuitry plays a very crucial role in the actual design and working conditions of the project. First of all, rough schematics were created in the first week. As the team did not have enough time to create a PCB, it was decided to use a breadboard for the prototype circuitry.

The initial schematic can be seen in Figure 4. The resistor values for the red and blue LEDs are slightly different to match the brightness levels.

To minimise the requirements of the microcontroller (benefits would be seen in production as a simpler and cheaper microcontroller could be used), only two digital output pins are required to supply the signal to the four speakers instead of four analogue output pins. This was accomplished using two demultiplexers and a few resistors. The demultiplexers are connected in such a way that the first four channels go to one speaker while the last four channels go to a different speaker. These four different paths have different resistances which in turn reduces the volume of the speaker. This results in the microcontroller to be able to supply and change the volume between four distinct sound levels. The downside of this method is that only one of the two speakers can be used at a time, but by grouping the "front speaker" with the "back speaker" and the "left speaker" with the "right speaker" the problem is avoided. The game does not require the previously mentioned speaker pairs to be active at the same time.

The motor is powered by a separate battery supply as the microcontroller cannot provide the required voltage. A capacitor was added to reduce current spikes over the battery. The motor is controlled by a motor driver as seen in the schematic.

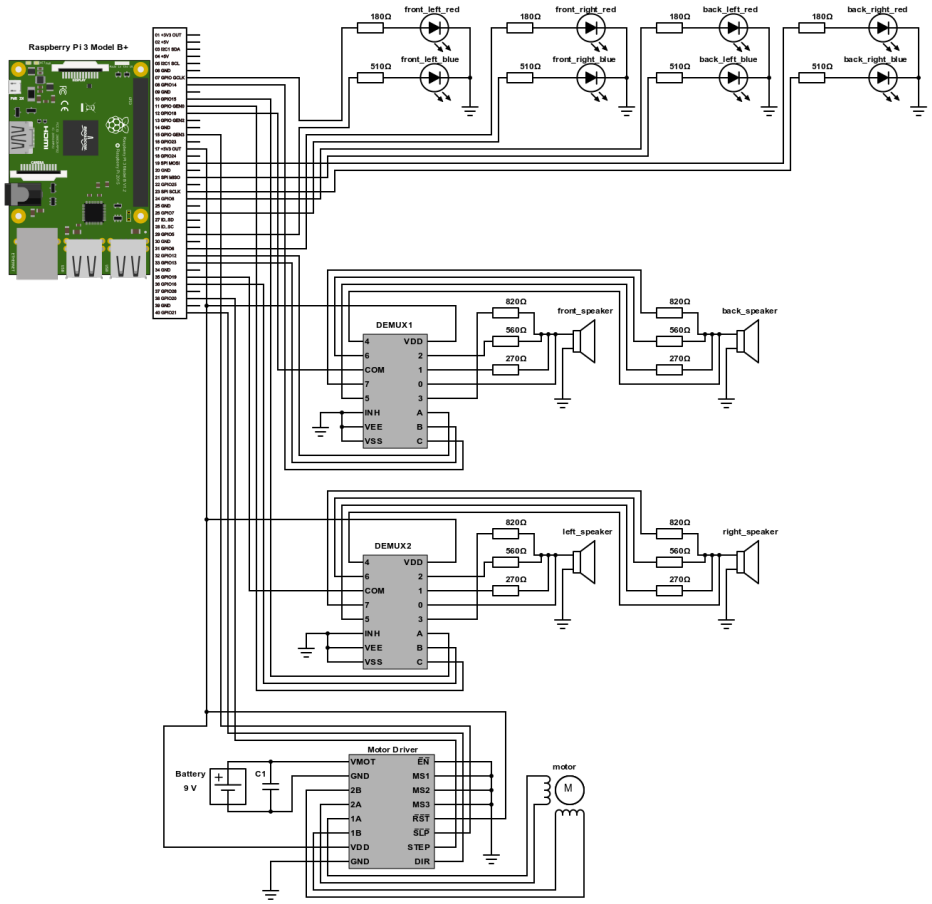


Figure 4: Initial Schematic

One of the major problems the hardware team encountered was the loudness of the speakers in the configuration seen in Figure 4. It was apparent that the speakers were very soft and were barely heard. Hence, the hardware team decided to use NPN Bipolar Junction Transistors to act as a switch so that the current drawn would come from the rails rather than the signal. The final schematic can be viewed in figure 5. This proved to be very effective as the volume increased significantly.

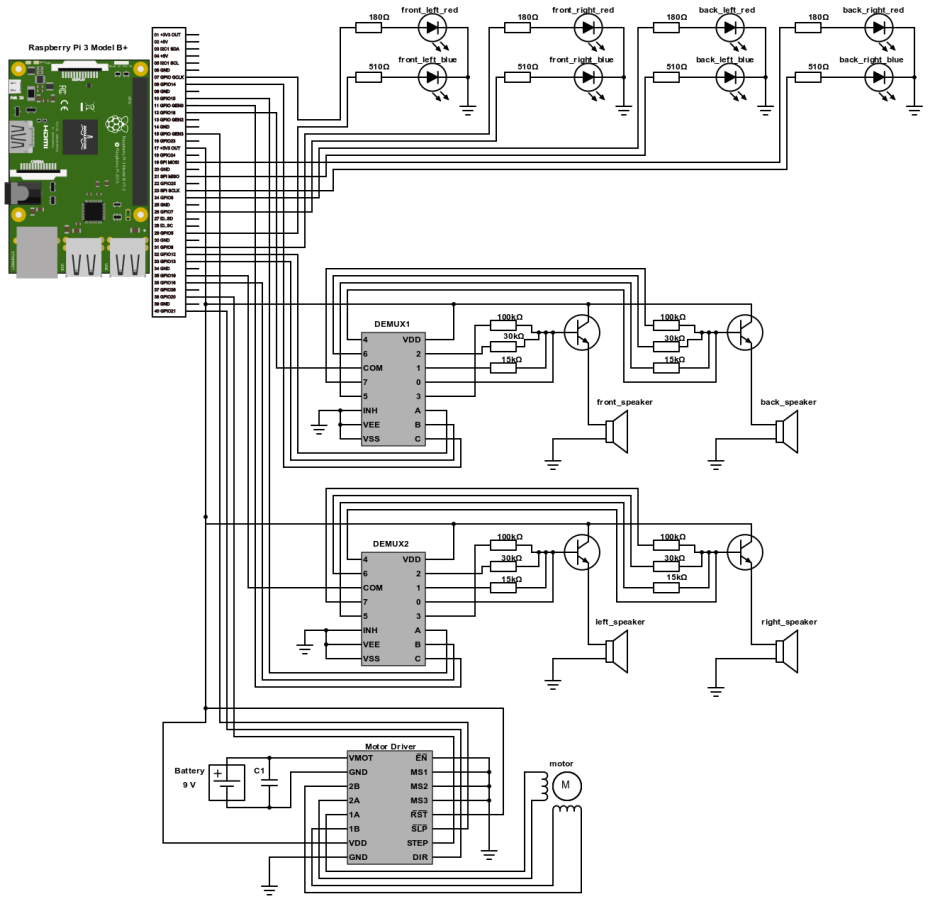


Figure 5: Final Schematic

The full circuitry used is shown as follows:

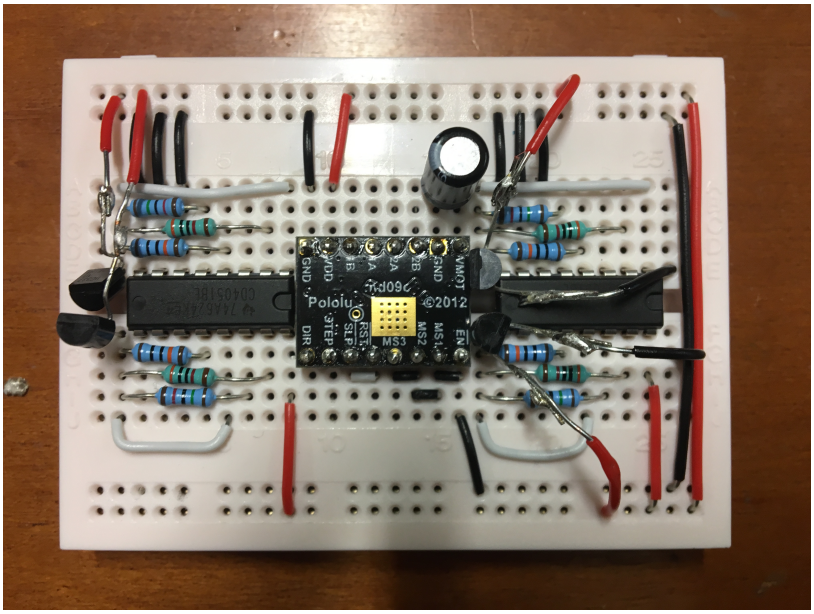


Figure 6: Final Circuitry

In order to change the loudness of beeping from the speakers, it was decided to utilise the ability of the DEMUX

to select different pins for the signal to flow. This was used for the purpose of changing the volume of the beeping in the particular speaker. There were 4 different loudness levels:

- 100% volume had 0 Ω resistance
- 75% volume had 15k Ω resistance
- 50% volume had 30k Ω resistance
- 25% volume had 100k Ω resistance

Since the team used DEMUX as a selective pathway for signal to flow, there is a special combination for each of the speakers' loudness. As can be seen, each breadboard has two DEMUXes, and one DEMUX has 3 select pins, which account for 8 possible states. RIGHT speaker is coupled with LEFT speaker and FRONT speaker is coupled with BACK speaker on the same DEMUX. The following table represents the combination of pins for different levels of loudness for RIGHT/FRONT speakers:

Table 3: Select Pins for working of right/front speakers

| **C** | **B** | **A** | |
|---|---|---|---|
| 0 | 0 | 0 | which is 0 (no resistance, the loudest) |
| 0 | 0 | 1 | which is 1 ($15k\Omega$ resistance, 75% loudness) |
| 0 | 1 | 0 | which is 2 ($30k\Omega$ resistance, 50% loudness) |
| 0 | 1 | 1 | which is 3 ($100k\Omega$ resistance, 25% loudness) |

Respectively, there is a combination of select pins for BACK/LEFT speakers and is as follows:

Table 4: Select Pins for working of back/left speakers

| **C** | **B** | **A** | |
|---|---|---|---|
| 1 | 0 | 0 | which is 4 (no resistance, the loudest) |
| 1 | 0 | 1 | which is 5 ($15k\Omega$ resistance, 75% loudness) |
| 1 | 1 | 0 | which is 6 ($30k\Omega$ resistance, 50% loudness) |
| 1 | 1 | 1 | which is 7 ($100k\Omega$ resistance, 25% loudness) |

Additionally, it is also worth to note that **at most** two speakers work together and that case is true only for RIGHT speaker working with FRONT Speaker or BACK Speaker **or** LEFT speaker working with BACK or FRONT speaker (depending on the direction the other ship approaches from).

## 4.4 3D Printing

The spaceship was designed using Fusion 360 as the CAD (computer-aided design) software. The dimensions are 43.1 x 10.7 x 25.4cm. This large size was required to fit the Raspberry Pi, power bank, bread board, motor, batteries, and all the connecting wires. In future prototypes, this can be reduced when the breadboard is replaced with a PCB design. The material used was polylactic acid (PLA) which was chosen due to its affordability, high tensile strength and high surface energy. It is also a very common material used for 3D printing. Since the design was larger than any printer our team had access to, the spaceship was spilt into parts. The first copy was printed in orange PLA and took one week to print. Therefore, the second copy was optimised by reducing the thickness from 4mm to 2mm. The second copy was then printed in translucent black PLA in 4 days. Finally, the parts were super-glued together and screw holes compatible with the M2 (2mm) screws were drilled for the lid.

The design can be accessed via the link: https://tinyurl.com/ibm3dmodel. To edit the 3D spaceship design, the file can be downloaded and exported to one of the following editing platforms: Fusion 360, Autodesk Inventor 2016 or Sketch Up.

## 4.5 Software Implementation

There are two main parts of the software that need to be deployed - software running on Raspberry Pi and on IBM Cloud. There are several options for such implementation. The following sections will discuss the choice of language or technology used for each part:

- **Raspberry Pi**

  The aim is to connect the hardware (LED, speakers, etc) to the Web Application served on IBM Cloud. Both Python and JavaScript (using Node) could achieve this goal. For the ease of collaboration, it is decided to use JavaScript, as it is the same language that is used to develop the Web Application. In depth details of implementation is documented in GitHub: https://github.com/CliveWongTohSoon/IBMRaspberry. To summarize the implementation, Node.js is used to connect to the General Purpose Input/Output (GPIO) on the Raspberry Pi, to the database and to IBM Watson Services. These implementation went relatively smooth. However, there are still several obstacles encountered, with the most prominent one being Raspbian OS does not natively support Bluetooth input and output. Additional and lengthy configuration needs to be done so that Bluetooth Headsets can provide clear (less noisy) input and output. It was also worth noting that manual configuration needs to be done to connect to the Enterprise Wi-Fi (such as Imperial Wi-Fi Network).

- **IBM Cloud**

  Two options were considered to host the Web App on IBM Cloud - Node-RED (an online visual web app IDE by IBM) or any other Web App framework which can be written locally. Writing the Web App locally is preferred as it can be edited, tested and served locally, which makes it more robust and less bug prone with the help of an IDE such as WebStorm or Visual Studio Code. It also allows collaboration via GitHub.

  MEAN Stack (MongoDB, Express.js, Angular front-end framework, Node.js) framework was chosen as it allowed a single page application and provided useful packages. It was also a highly modularised framework, which made the delegation of the tasks more modularised (such as design of the web page, functions of each web page and backend API). This allowed software team to write the code more efficiently.

  MEAN Stack Web App is hosted on IBM Cloud, which is connected to MongoDB for database management. SocketIO is also set up in the server to allow the Raspberry Pi to connect to it for real-time data exchange. The detailed implementation and documentation of the Web App is in the Github repository: https://github.com/CliveWongTohSoon/IBMSocialGame.

Object class (or data models) are created in TypeScript, which is the language used by Angular front-end framework to store the state of the game. Such models are updated via the "Observable", a return type object supported by RxJs to allow real time function call. The "Observable" is listening to the sockets on the server, which is called by the Raspberry Pi whenever there is action input. As such, the socket acts as the "middleman" to receive and emit data between Web App on IBM Cloud and Raspberry Pi. Afterwards, the data will be observed by the Observable and the data models will be updated accordingly. In the following sections, detailed data models of the game are explained.

### 4.5.1 Ship Model

Ship model is written in ship-model.ts. The ShipModel class is used to monitor the status change of the battle ship in battle field. The ShipModel class has several member fields to represent different data of the ship.

- ShipAction

  ShipAction is an enumeration class that represents different actions the ship could perform: shield front, shield right, shield back, shield left, shoot, move, turn right or turn left. A 'DoNothing' action was also added to this class as a default value in case the player gives less than 3 commands at the end of the turn.

- Report

  Report is an array that stores the possible reports after an action has been performed, for example: hit target, fire miss, radar detects enemy, etc.

- ShipDirection

  ShipDirection is an enumeration class which contains all the possible direction a ship could be facing: up, down, left or right, relative to the battle field.

- ShipDepartment

ShipDepartment contains an array of Department class, wherein a Department class contains information of a department: department's x and y coordinate, health and alive status. The departmentArray's index is arranged in a way that 0 represents right back engine, 1 represents left back engine, 2 represent left front weapon and 3 represents right front weapon. Hence, the ship is represented as a 2x2 grid where each unit depicts one department. The static function getDepartmentHealth is used to create new Department object by passing in some fields of ship model.

- ShipPosition

  ShipPosition contains the x and y coordinates of the centre of the 2x2 grid.

- ShipStats

  ShipStats class stores the numerical stats of a ship, including Hp (Health Points), attack, attack range, defence, radar range, whether the shield is active and the direction of the shield relative to the ship.

- ShipPhase

  ShipPhase is an enumerical class that indicates the different phases a ship model could be in during a game. Currently, there are four possible states, namely start, action, report and end. A start state indicates the ship is entering the game, an action phase indicate the ship is performing actions. A ship in the report phase will be outputting the report, and finally the end phase indicates that the ship is leaving the game. Notice at the beginning of the game, a ship could be selected by a player as long as the ship is not in the end phase.

- colorFront and colorBack

  The colorFront and colorBack of a ship stores the colour a particular department rendered on the battle field.

- rp

  ShipModel member rp is an array of the RelativePosition class. A RelativePosition object stores the relative polar coordinates of another instance on the battle field which can be detected by the ship's radar.

#### 4.5.2 Asteroid Model

Asteroid Model is written in asteroid-model.ts. The AsteroidModel class is used to monitor the status change of the asteroids in the battle field. AsteroidModel class has several member fields to represent different data of the asteroid.

- AsteroidPosition

  AsteroidPosition contains the x and y coordinates for the 1x1 asteroid.

- AsteroidMotion

  AsteroidMotion indicates the x and y vector coordinates for the direction the asteroid will travel in at the end of each action executed.

- Damage

  Damage stores the value of damage to be dealt to a ship in the occurrence of a collision with the asteroid.

- Collided

  Collided is a boolean that stores whether an asteroid has collided to allow for the consequences of all collisions in a turn to be processed in parallel.

#### 4.5.3 Game Service

The file game.service.ts contains various implementations of the game. Part of it contains functions that update the state of the game. There are several functions here:

- init ( )

  Creates a square grid of the passed in input length. It also calls the createAstArray which in turn calls genAstPosition, genAstMotion and genAstDamage to initialize the asteroids' position, motion and the amount of damage it could inflict respectively. The genAstPosition function is used to generate a random position similar to the way the ships are initialized where in each asteroid is made sure not to spawn in the same zone initially.

- createShipFromSocket( )

  In order to create a ship according to data in the database, it first listens to the socket from the server, and uses data from the database to update the state of the ship using function updateDepartmentHealth. The getPhase function is used to set the phase. This function also sets the initial position of the ships which is calculated in such a way that the ships don't spawn too close to each other or to the asteroids. For example, in a game with two ships and two asteroids, the grid will be split into four zones where the ships and asteroids will be spawned on alternate zones respectively.

- updateGridWithAllShip( )

  Checks the battle field and ship position, then updates the grid on the web page.

- listenToInstruction( )

  This function listens to the socket on the server which will be emitted whenever the full instruction sets (3 instructions) have been completed and sent by the Raspberry Pi. This allows function execution (such as updating the state of the game) via event listener socket, which will be activated by the Raspberry Pi connecting to the same socket.

- getInstruction( )

  Checks the instruction value and returns the instruction in the instruction class format.

- excecuteInstruction( )

  Calls the appropriate function for the called instruction.

- updateAstGrid() and updateGrid()

  Renders the asteroids and the ships on the grid respectively.

- updateShip( )

  This function is used to update the position, direction and the health of each department of the ship as these values dynamically change during the game.

- worldRound( ) and circleAround( )

  In order to resemble infinite space, the ships move in a circular grid, wherein once the ships reach the extremities of the grid, they should appear back on the opposite side. The worldRound function takes care of this. Similarly the circleAround function tends to the asteroids.

- move( )

  This function checks the current direction a ship is facing and moves the ship one grid towards that direction. As a result, if the ship crosses the border of the battle field, the worldRound function will bring it back to the other side of battle field.

- asteroidMove( )

  This function uses asteroid motion to move the asteroid one unit at a time. Similar to the move function, this function calls the circleAround function to mimic an infinite grid.

- rotate()

  This function updates the shipDirection to perform a left or right turn. It adds or subtracts 1 to the enumeration class shipDirection to execute the required action.

- relativePosition( )

  This function is passed ShipModel as a parameter and calculates all the other objects' polar coordinates relative to it and stores them into the rp member field of that ship.

- checkCollision( )

  The aim of this function is to find out if a collision has happened, assign a resultant direction in which the ships should bounce back to and update the health of the collided departments. This is done in the following way:

  The function loops through all the ships to check if the collision conditions come true. If it does, the checkCollisionHit function then checks which department/s of the collided ships have collided. These departments' health is then updated using updateCollisionHealth.

  The updateCollisionHealth function sets a standard value for collision damage as 300 and is then adjusted depending on two different situations, i.e. if the rammer ship's department has 0 health or if the victim's shields are up. If the shields are up, collisionShieldCheck checks if shield is in the required direction and assigns a reducedDamage value taking into account the defence capabilities of the victim ship. Finally, the calculated damage is subtracted from the health of the department.

  The resultant is calculated similar to vector addition and then added to the resultantMove value to account for multiple ship collisions. Since the game is turn based-real time, there is a possibility of the ships overlapping with each other. Hence if the x and y coordinates of the ships are equal, the assignResultant function designates the resultant value depending on two conditions, i.e. if the collision is the result of bounce back of an already occurred collision or not. The resultant is assigned such that the ship will move in the opposite direction to the direction the ship was previously moving in.

  The function also assigns the number of units the ships must bounce back by.

- checkAsteroidCollision( )

  This function finds out if a collision between a ship and asteroid has occured, assigns a resultant to both and updates the health of the collided departments.

  The resultant is calculated similar to the resultant calculation in checkCollision. The updateAsteroidHealth function is called to assign the damage caused by the asteroid. The asteroid itself doesn't incur any damages. Unlike checkCollision, the number of units the ships must bounce back by is fixed at 3 units.

- performCollision( )

  The aim of this function is to move the ships back by the assigned moveCount value after collision has occured while also accounting for chain collisions with asteroids and/or other ships.

  If any of the ships have bounce back moves left over, those moves are executed one at a time. After each move, the checkCollision and checkAsteroidCollision functions are called again to check if any further collisions have occurred. Finally the resultantMove is reset to 0 for later collisions.

- shoot( )

In shoot function, it passes a ShipModel as parameter, and it enters several for loops which in turn checks all of its weapon departments, attacking range, potential target, and checks if any group of them satisfies the shoot condition. If the shoot conditions are met then it will update the health of the relevant enemy department using the updateShootHealth function condition and if not, it will push an information report of a miss.

In the updateShootHealth function it assumes the hitting condition is satisfied, and it change the victim ship health and pushes the report according to its health and shield condition.

- fullTurns( )

  This function is responsible for the turn implementation of this turn based game. In each turn, it starts with clearing the report from the previous turn, checks all three instructions of all ships based on the priority order, checks collisons of ships and asteroids, stores relative positions, and finally stores all ship data into the data base using socket emit. The alive boolean of the ship is also checked when relevant, mainly after shoot and collision check functions.

- worldRound( ) and circleAround( )

  In order to resemble infinite space, the ships move in a circular grid, wherein once the ships reach the rightmost side of the grid, they should appear back on the leftmost side. The worldRound function takes care of this. Similarly the circleAround function tends to the asteroids.

- shield( )

  The shield( ) function is called when the player asks for the shield to be put up on a particular side of the ship. The function then sets the shield as being active and assigns the direction the shield should be up in.

## 4.6 Testing

There are three main stages for testing the product.

### 4.6.1 Stage I

In Stage 1, only the front end web game was completed, so button elements were created in web page to control the behaviour of ship in the web game. By linking the buttons on the html directly to the respective functions, the functions can be tested modularly to make sure it updates the game state correctly (such as the position of the ship, ship stats, etc.)

### 4.6.2 Stage II

In Stage 2, the database was set up using MongoDB. Sample data was manually inserted into the database to test the event listener of the server via socket. At this stage, the web app listened to the instructions, ship status and report status from the database and updated the game state accordingly. Nonetheless, a "done" button was created on the front end to act as an event emitter, which should be executed by the Raspberry Pi. The button mimicked the completion of a player giving commands, which would emit the instruction data to the server and update the instruction sets saved on database. This in turn emits another event to the front end via the function listenToInstruction(). At this stage the game behaved similar to Stage 1, but instructions and game state are now retrieved from the database instead of from the front end.

### 4.6.3 Stage III

Finally in stage 3 when the spaceship was fully assembled and the software running on it was stable, input via voice commands are tested. Voice commands such as "Move Front", "Shoot" was used to test whether Watson transcribed the voice correctly, and understood the context. Afterwards, it was also verified that it connected to the server and updated the database accordingly. The performance was very bad in a noisy environment. However, after reducing the sampling frequency to 8kHz, the performance in terms of accuracy became acceptable. However, Watson still had the issues with differentiating user's speech when there were multiple people speaking (noise). To improve upon it, it was suggested to learn the user's voice model at the start so that Watson could isolate the user's speech from the noise. However, due to time constraints this measure was not implemented.

# 5 Conclusion

## 5.1 Client Satisfaction

The prototype was demonstrated at the hack booth, where it was visited by our client (John Mcnamara) and two of his business partners. He was thoroughly impressed with our prototype and it met all the criteria that he had wanted in the brief. It also went beyond the brief as the game design allowed several players to be connected if more ships were made, and asteroid attacks were added to make the game more exciting. He has asked if he can have our ships on display at IBM to show business partners the versatility of Watson.

## 5.2 Sustainability

The ships were made of polylactic acid (PLA), which is a biodegradable material that can be recycled back into filament for 3D printing at the end of its product life cycle. The hardware inside, the Raspberry Pi, power bank, motor, and batteries, are mostly not recyclable. For further work, to improve the sustainability, rechargable and recyclable batteries should be used. The power bank for the prototype allows it to work a maximum of 10 hours before recharging is required. The power bank can be replaced with a direct connection to an outlet. The Raspberry Pi 3B+ can be replaced with Raspberry Pi Zero, which conserve power and are also half the size. These changes would also result in requiring less space for hardware, so the ships can be made smaller which conserves energy used by 3D printers and uses less material overall.

## 5.3 Total Costs and Product Life Cycle

The expenditure detailed in section 2.4 is for the prototype, but this would be significantly more than the final product that would be marketed. The Raspberry Pi Zero would not only reduce the size and power usage of the product, but it is also much cheaper (£9). Many components, such as the LEDs, speakers, resistors and capacitors, are much cheaper when bought in bulk. Taking these costs into account, bill of materials required per unit would be £20. The building costs per unit can be estimated by looking at how much it costs for a similar existing product, such as the video game console Xbox One, which is £11. After analysing the sales of several video games, it could be seen that approximately 40 percent of the sold price goes towards distribution and the retailers, and this can be used as an estimate for our product. This increases total cost to £52. Therefore, if we are aiming for a 30 percent profit margin, the retail price of a ship should be £75. The product lifecycle is decided by the shortest product lifecycle of all of our components. This will most likely the the capacitors, which is 5 years. Therefore, our product lifespan is also 5 years.

## 5.4 Socio-economic Impact and Benefits

The prototype can be used by just using voice command on the ship, so this achieved the aim of creating a game for people who dislike the barrier to entry for video games or looking at screens, or those who have disabilities that cause difficulties with manipulating the controls of a video game. It can also be played with more than two people provided that each player owns a ship. This makes the game more inclusive of everyone and can be played in large groups, making it a social activity, which was another aim of this project. Alternatively, it can be used to connect people far away from each other as the players do not need to be in the same location. Finally, the cost of this game is significantly less than any video game console, so it could be considered a cheaper alternative.

## 5.5 Ethical Consequences

Using AI in games will anthropomorphise the game and is hence expected to make a huge difference to future of the gaming industry. As demonstrated here, since the game is operated through voice commands rather than using traditional gaming devices such as a controller, keyboard or mouse, the game is much more immersive. In traditional video games, players may stop to take a break due to eye strain from looking at a screen for too long. Unfortunately, in the AI Social Game, there is not a similar situation which would case the player to stop playing. Hence, in the future, when more AI games are in the market and further research has been made, developers of such games might consider anti-addiction systems, like limiting the playing time or verifying the player's age in order to reduce the negative impact to society due to AI games.

Furthermore, one of the future goals of this project was to give the crew members of the spaceship an AI 'personality' and soft problems. The players would also be required to give longer sentences as commands which can be analysed by IBM Watson's Tone Analyser. This allows for a more immersive game wherein the player's

attitude towards the crew would affect the gameplay dynamically. This could have possible ethical consequences where players could associate real life situations with the AI crew's reaction to their commands, i.e. if the player is rude to the crew and yet wins the game, younger or impressionable players may assume the same would be true in real life situations.