Overview:

The client is a group of PhD students that conduct research on the flight behaviour of the urban gulls in Bristol (e.g. how they utilise thermal air currents to conserve energy, how their behaviour differs from gulls living in a rural area, etc.). They use this research, as well as information about the natural sensory organs of other animals, to find ways to maximise the potential of drones to solve problems and contribute to society.

In addition to this research, the PhD students run a SCEEM Outreach workshop that is related to the research they are conducting on bio-inspired flight, to engage them with STEM. Their program consists of a workshop, where attendees are given a problem and asked to solve it using drones. Attendees are given a "budget" that they can use to buy "bio-sensors" (sensors inspired by natural sensory organs of animals) for their drones that may be useful to solving the problem (e.g. echolocation for navigating dark tunnels). They are then asked to manoeuvre a real drone around a physical obstacle course.

The PhD students have requested for a platform similar to the workshop. Though the solution we have presented will also be entertaining to the users, the domain of the project is mainly education; the app will focus on general orienteering of the workshop attendees. The problem that the project will solve is the continuous engagement of attendees after the workshop has ended, to keep them inspired to pursue a career in STEM. As such, the project is not meant to serve as a replacement to the workshop, but rather as something for the attendees to take home and keep them engaged with STEM.

Our proposed solution was to design and create a mobile (Android-based) 3D game, to replicate the challenges that are given to the participants of the workshop. As with the original workshop activity, these challenges will consist of an assault course of problems, as well as a customization system where they will consider the different "bio-sensors" they put on their drones.

User Stories

Workshop Attendees:

These are the people who attend the workshops. They are likely to be students from secondary schools and colleges.

User Stories:

- **The apathetic student.** This student is only attending the workshop out of obligation. They are likely to be part of a school group and would not otherwise have chosen to be there. Primarily, they are in the workshop because they wish to play with and use the drones. They do not necessarily have any interest in STEM and aren't likely to have much knowledge in the field. This is likely to be due to a lack of exposure to the field, and any chances that they may have had before having not engaged them.
- An apathetic student will first and foremost want the game to be enjoyable to play. As they don't have interest in the research side of the app, they are likely to skim over any details we give them. This could lead to us considering hiding some research topics behind a further menu, so as to not overwhelm the users: especially those that would not want to see this information anyway.
- Furthermore, this student will want the game to be engaging. As they have little to no interest in the extra info that we will be providing on the bio-inspired mechanics, we must make sure

that the game on its own is still about to keep their attention. To further make sure of this, the student would want the game to be re-playable, therefore allowing it to hold their attention for longer.

- **The curious student.** This student has more of a general interest in both their education and their schooling. They may possibly already have an interest in STEM fields and could have some basic insight into what the workshop could entail. They have chosen to attend this workshop in order to gain experience and as an opportunity to gain more knowledge in these STEM areas. This student will be interested in more than just having fun flying drones and will hopefully be receptive to the research information that the workshop provides.
- A curious student will want our game to be informative in regard to the bio-inspired aspects that are integrated. They will want to be able to look further into the research within our app. Their main aim in this regard is to become more informed on drones and bio-inspired flight in general.
- This student may also wi.sh for the game to be representative of real-world scenarios. They would appreciate the control scheme matching that which they used in the workshop and would possibly be put off if the physics in our game are too disassociated with real-life drones.

- **The driven student.** This student is likely to be exceedingly interested in STEM fields and already have a fair bit of background knowledge in the subject. They are also likely to already be considering their future prospects and what they would like to do in life. Out of these three defined types of students, the driven students are the ones that would be most enthusiastic in attending the workshop. They wish to prepare themselves and to get an insight into what their higher education could resemble (specifically with regards to a possible future in STEM). Like with the curious student, the driven student will also wish to expand their knowledge in the subjects covered in the workshop.
- A driven student will want the game to be educational and informative. They would appreciate being able to learn more about the topics that they're already interested in, and so will have a heavier focus on the research information that we provide; much more so than the other users.
- This student will also be looking to use this research in order to inform their decisions in pursuing a career in STEM. We will need to make sure that the information we have been given is displayed clearly and in an interesting manner. We do not want these students to lose interest due to a poor layout. This student may take inspiration from our clients' research in order to shape their future prospects.
- As with the previous students, this user would also want the game to be engaging. Even though they may be interested in the subject which is being covered, if the game is not enjoyable to play or not gripping enough, they will soon look elsewhere for their needs. The student will still want to enjoy the game itself.

Workshop Ambassadors:

The people who run the workshop and guide the students through the activities. This includes our clients, and the other people that they have enlisted to assist in the running of the workshops.

User Stories:

- **PhD students running the workshop (our clients).** Our clients are incredibly passionate about their work and their research. They are proud of what they have researched and wish to share it with other people through the medium of their outreach workshops. They wish to get the attendees excited and engaged in the subject of bio-inspired flight, and through this they look to inspire further engagement into the fields of STEM. They also strive to make sure that their workshop is entertaining and informative (in terms of data from their research) at the same time.
- The PhD students will want our game to be engaging. They wish for it to further expand on the workshops that they have ran and allow the users to gain more insight into their research in bio-inspired flight. As a result, they will want the game to be realistic enough to portray real-world scenarios, whilst still being fun to play. They would want a good balance between the entertaining gameplay aspects and the informative research areas of our app.
- The PhD students also wish for our game to assist in the students remembering their workshops fondly. They want the students to stay engaged in the subject once the workshops are completed. As a result, we will make sure that the features in our game run parallel to the workshop tasks, therefore reinforcing what the students covered during their sessions. To further compound this, they would like to expand their research out to other people that were not able to attend the workshops. By having the users take the game home with them, they may be likely to share it with their friends and demonstrate the experiences that they had to more people.

- **The other ambassadors running the workshop.** These people are helping out with the workshops and making sure that the activities run smoothly for the students to take part in. With more relation to the app which we are creating, these ambassadors may be the ones helping to install it onto the students phones. They may also teach them how to use the app and play the game.
- An ambassador will want the game to be easy to explain to the students; they won't want to be spending ages trying to help the students get to grips with it. As a result, they will be looking for our interfaces to be simple and intuitive, and would want the control scheme to match the controls used in the workshop as this would mean they would not need to teach a second set of controls.
- The ambassadors would also not want the game to overshadow the workshop. This outreach program is meant to be demonstrating the research that has been completed, and so they will want the game to reflect that. It should reinforce the workshop experience, not retract from it.
- Finally, they would want to students to be engaged by our game without too much hassle. They would not want the students to be put off from the app by having a difficult set-up procedure or too steep of a learning curve.

Requirements:

Core user stories and flow steps:

Out of the user stories above, the ones we will choose to focus on are for the **curious student** and the **PhD students running the workshop**.

The curious student:

As a curious student, I want the game to be engaging, informative and representative of real-world scenarios. I would like this so that I can further look into the subjects and be more informed.

Basic flow:

1. Attend the workshop.
2. Participate in workshop activities.
3. The workshop ambassadors tell us about this app.
4. Download and install app from the Google Play Store.
5. Open application on mobile device.
6. Select "Play" on the main menu.
7. Get taken to the drone customization screen.
8. Read flavour text on different bio-inspired drone sensors.

Alternative flow after Step 7:

1. Select drone parts as normal.
2. Begin playing the game.
3. Complete objectives.
4. Earn achievements/collectibles.
5. Go to main menu.
6. Select the option for achievements/collectibles.
7. Read flavour text in achievements.

During this alternative flow, the student may feel particularly rebellious. This could lead to an exceptional flow during gameplay:

9. Instead of completing objectives, explore the level by controlling the drone.
   a. The student may choose to simply roam freely around the map and practice flying the drone.
   b. Alternatively, the student may choose to actively find ways to break the game for their own amusement.

While this would not necessarily help the student achieve their initial goals of looking further into the subject of the workshop, it could still lead to the student gaining an interest in drones and/or programming in general, which would push them towards STEM.

PhD students running the workshop:

As a PhD student who runs the workshop, I would want this game to be engaging, inspiring and somewhat informative. I would like this so that the attendees can remember the workshop fondly and may wish to share their experience with other students (via the game).

Basic flow:

1. Set up the workshop.
2. Brief attendees.
3. Run them through the workshop like normal.
4. After workshop activities, inform attendees about this app.
5. Encourage attendees to download the app.
6. Attendees download the app, gaining access to educational resources (and a fun game!).

An alternative flow would be to inform the attendees during a break in their workshop.

As for exceptional flows, the app will be Android-based, so attendees will be unable to install it on iOS, for example. However, this is not an issue which we can resolve, due to being locked to creating our app for Android by the project requirements.

Our main goal is to fulfil the client's request of increasing the attendees' engagement with STEM. As such, we will base our requirements mainly around the user story of the **curious student**, as they are the attendee that is most likely to benefit from such a platform.

Functional Requirements:

GAME-1 The game must be optimized to run at a constant framerate of at least 30 FPS to ensure a smooth gameplay experience, even with 3D assets and the (relatively) limited hardware of a smartphone.

GAME-2 The core game features will function normally offline. This is to ensure that the user is able to use the app at any time.

GAME-3 The user interface must have a control scheme that is considered easy to learn by at least 7 out of 10 play-testers.

> **GAME-3.1** The initial design of the user interface will use the same control scheme as the drones in the workshop, to allow participants to learn the controls quickly.
>
> **GAME-3.2** The requirement **GAME-3** takes priority over **GAME-3.1**.

GAME-4 Collision detection will happen before applying player control during each iteration of the game loop, to ensure that the player does not move when it isn't supposed to (i.e. if the player is trying to phase through a building).

INFO-1 The application will include a page with scientifically accurate resources related to the client's research of bio-inspired flight and sensors, to further the client's goals of encouraging the user to pursue an education in STEM.

> **INFO-1.1** The main resources related to this research must be accessible offline to allow the user to read through them at any time, at their own pace.

INFO-2 If the game has a loading screen, the application will also include facts related to the client's research on the loading screen, to allow the user to learn during gameplay without being overly intrusive.

Non-functional Requirements:

APP-0 The size of the app's APK file **must NOT exceed 50MB**, to comply with the Google Play maximum APK file size limit on all devices. (Some versions of Android allow for APK files of up to 100MB, but we want the app to be accessible to as many people as possible)

APP-1 The game must be suitable for the PEGI 3 rating, to allow attendees of all ages to benefit from the app.

GAME-5 The different problems presented in the game must mirror the problems in the workshop activities, to supplement the client's existing workshop.

> **GAME-5.1** The development team will ask the client for advice if any additional scenarios are to be added to the game, to ensure that a sense of realism is preserved.

SEC-1 If a login system is implemented, user accounts must be secured with a password of min. length 8, containing at least one uppercase letter, one lowercase letter, and a number. This is to ensure a reasonable level of security on the user's end.

> **SEC-1.1** Any passwords stored this way must be stored in its database in salted hash form, to ensure a reasonable level of server-side security.

SEC-2 The app must NOT allow the user to store sensitive information (i.e. address, health conditions, etc.), as the app is intended to be used by people of all ages (see **AGE-1**), including children as young as 6 years old, who likely do not understand the implications of sharing such data.

Architecture

With regards to the client's specific requirements, the main ones were that they wanted a mobile aide to their workshop to help further engage the attendees. This came with it, a whole load of technical requirements for our product. As well as deciding to create a game for the client, we decided the game needed to be an Android project, to allow us to fulfil the mobile requirement, whilst keeping the project Java based.

After agreeing to a game aide, the clients were also not too particular on whether they wanted a 3D game or a 2D game. After taking part in their workshop and seeing what their outreach programs were like, we opted for a 3D game. During the activities, the clients specified that they wanted almost a simulation of their workshop to give to the attendees and with the challenges they go through in the workshop; it would be very hard to replicate the challenges realistically in only 2 dimensions.

However, when it came down to deciding how we wanted to implement the game, the decision of needing it to be on Android was quite the hinderance. We went about looking for a 3D games engine which we could use for our game, but the best candidates for them were just out of the requirement spectrum. Unity had amazing functionalities and very easy portability into an Android project but sadly did not fulfil the Java requirement (due to it being C# / C++ based). After extended searching for a games engine to use, we believe we found a Java based games engine which boasted Android portability, JMonkeyEngine. After a week's worth of tinkering with the software, we came across multiple issues with its Android portability and could not find the solution. We eventually rested on the idea of creating our engine for which our game will have to run on, including its own physics engine, controller classes, etc. As for the 3D rendering engine, we were recommended a 3D graphical engine called Processing to use as it was a Java based, simple graphical engine that we could easily integrate into an Android project. This saved us from also having to program it ourselves on top of everything else.

Also, as product of making our own game classes, we decided that making controls touchscreen as opposed to the mobile platform's capabilities of gyroscope controls would make the project more feasible. Having to program controls using gyroscope sensors would take a lot of time and complexity. Also, we also felt that gyroscope controls were lacking in fidelity, and very different to their workshop, which revolved around R/C drones; we felt that the on-screen touch controls were simpler to implement and represented R/C controls more.

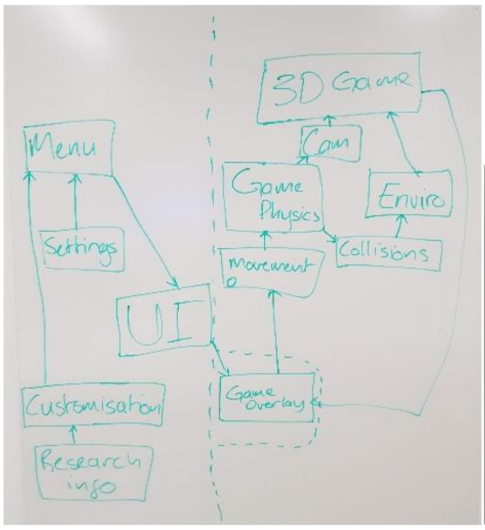
One requirement we derived from our client meetings was that the workshop employees were not interested in maintaining a server. Hence making the game completely offline (not requiring servers to maintain) would be the best approach for this project. Progress, high scores and login details will be saved locally on the device (almost fully securing it) the game is installed on. Having all these files be stored locally will mean the game will be fully capable of being played in an area with little to no connectivity. This does completely remove the multiplayer aspect of most mobile games, but with the content we plan to introduce to the game, it'll be more focused on completing achievements and high scores than playing against/ with friends.

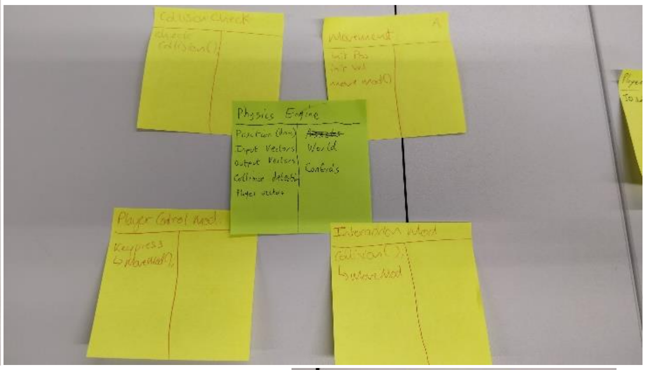
The user interface was the simplest thing to decide on as the Android Development Studio is a very simple and easy way of programming a front-end UIs which we plan on using to create our menus and possibly a UI overlay for the 3D game (to handle the controls). This will contain our main game menu, settings, achievements board, customisation options, and more if we come up with more features to add.

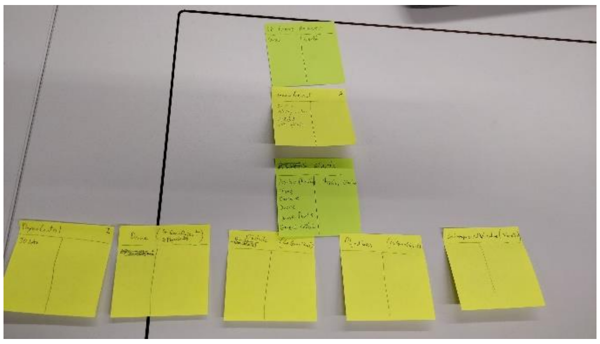
As for the high level diagram for the project, we decided it best to operate through the front-end UI and gameplay loop separately. This seemed like a good idea as yes, it will increase loading times between levels (due to needing to load level specific assets between bouncing around the menu and game), but it will also have an enclosed gameplay loop that is only interrupted by pause conditions and completion which is a lot simpler than other UI controls intervening; also we are hoping that computations in the gameplay loop would have more resources to work with as the UI has no need to be maintained during gameplay, making the game run smoother.

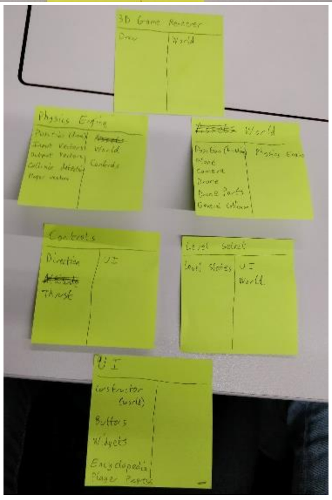
By taking a deeper look into the specific components in our high level architecture, we began to think of the OOP structure we want to have in our code. In the name of expandability, we wanted to really focus on the structure of the OOP concepts in order to allow new assets/ features to be added further along the production line with seamless integration. For example: our physics engine would consist of an abstract class of movement modifiers which consist of only the most key methods (like positionTrack(), input/outputVectors(), etc). Having only these key methods, we open up the opportunity for us to introduce new mechanics like environmental effects which would extend the movementModifier() abstract class and @Override the methods it would affect (i.e. if a thermal vent is used (traversed over) have the class modify (increase) the velocity vector to shoot straight up into the air, simulating how they affect the flights of seagulls).

The project being specifically for an outreach program gave us an unforeseen requirement on how we were going to deploy it. Whether we wanted it to be exclusive to workshop attendees (to encourage the participation in them) or whether we wanted a wider influence on the attendees and the people around them by allowing it to be open for anyone to use. After speaking to the clients, we bounced the idea of publishing it through the Google Play Store in order to make it simple and easy to share and install (as it is just like any other game/ application that the attendees would be using). This strategy further promotes the workshop as it would allow attendees to share the application with people who may have not had the opportunity to do it, at the end of the day, carrying through the end goal of the outreach workshop. Although this method of filling the requirement also came with size constraints of the distributed .APK file which comes with the Google Play Store. This is a worry because of the suspected large size of the 3D assets of the game and other large files like the game physics file. We decided a server containing the assets which are too large to fit into the .APK file would be the best workaround and have routines which occur whenever the application is launched to check for assets needed to be downloaded and to install them in the modules of the application in order to allow the game to function properly.









Development Testing

(Disclaimer: I haven't been able to get the physics engine implemented to a point where I can test it, so I'm just going to talk about my strategy to test it.)

One of the largest and most core components in our project will be our very own physics engine to simulate the physics of flying a drone and enable easy ways of implementing game mechanics as modifiers of the physics being applied to the players. The physics works in a way that uses real world physics calculations being run in each of the moving assets of the gamestate, where calculations are being calculated every frame. To ensure a smooth running of the game, the physics engine has got to be able to output semi-realistic results to update the gamestate with.

Our testing strategy to ensure the calculations were outputting the correct results is to have the results manually calculated (by hand) and create assertion tests in a JUnit testing framework. Testing value types and call-backs are key to ensuring that calculated values are being computed and computed at the correct rate will also be key (as we are planning to lock the framerate at a stable 30fps to keep it simple). Testing for extreme cases are very important as that is the main cause in in-game bugs, and so error catching will be very important in locating interactions that deplete the playability of the game.

Test	**Pass?**	**Time Taken:**
Test valid input type to moveMod(x vector3) method: (assertTrue(x == int));	Pass	
Test correct output based on physics: (assertTrue(moveMod(x) == (y));	Pass	
Test over (0.5s) correct input and outputs to movemod(x): (assert(moveMod(x).givenMoveMod(x').givenMoveMod(x'')….);	Pass	

Release Testing

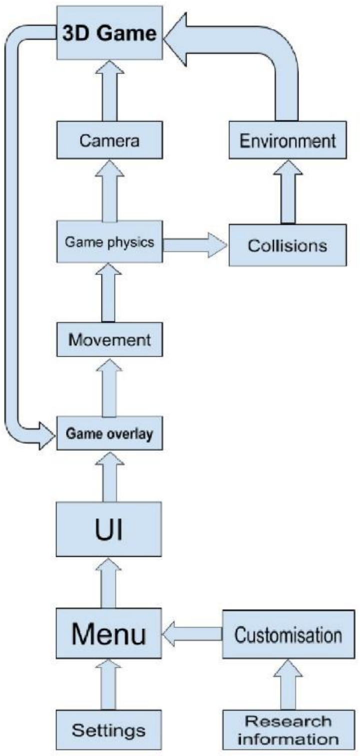
Phase	**Heuristic**	**Test**	**Met?**
Initial Release			
	Working Physics Engine	Passing all physEngineTest Correct update of gamestate	
	Basic Models	Low-poly environment, player model and other assets	
	Functional Controls	Clients can intuitively learn to control game Controls implement the correct interaction with game systems (physics engine and gamestate)	
	Basic Front-End UI	Clear buttons/ widgets to navigate through game Working game calls	
	Level	At the very least, 1 level design complete and work successfully	
Beta Release			
	Higher-poly models	Higher quality environmental design and texturing	
	Optimised physics engine	Faster processing physics engine (measured by computation times)	
	…	…	

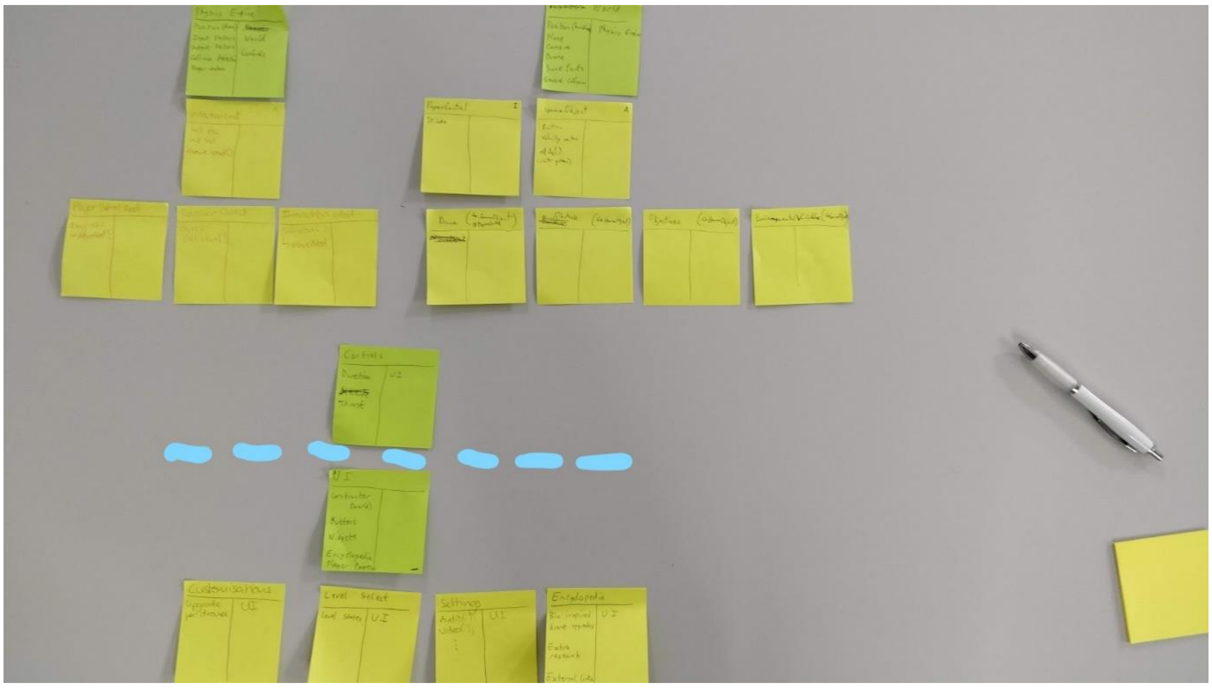
The project being a game, the implementation becomes a much more 'back heavy' project, where the internal architectures have to have been tested thoroughly and integrated well to the entire system architecture. This would give us a large workload focus on the initial release as certain aspects like the physics engine would need to be running almost perfectly with the other systems in order for any deliverables. Subsequent releases will be more a tool to streamline the game and implement additional features like 'customisations', server hosted account system, higher quality graphics, etc.

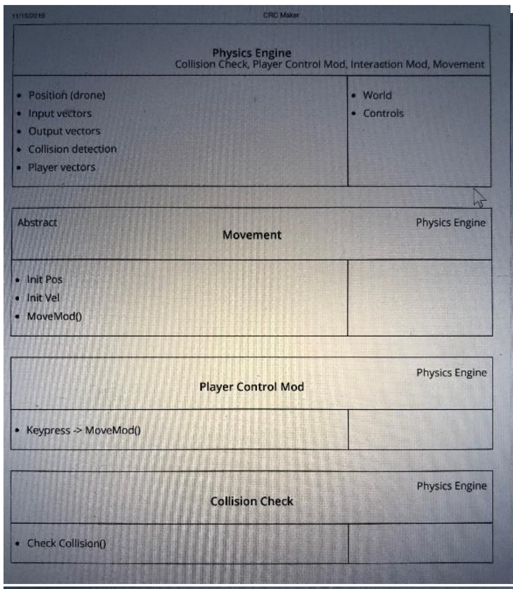
An example of a test being implemented is the test for functional controls. For the initial release, we had implemented the core touchscreen control scheme for the program and given a device to the clients to test for us and give us feedback. We wanted to see if the in-game directions and general layout of the controls made them intuitive enough for first time users to complete the first level with no external explanations. If our clients pass the first level without any help from any of the team, then it proves the functionality of the control scheme. Following their feedback (e.g. whether the believed the controls were restrictive) we would then make revisions to the control scheme adjusting certain aspects and include it in our beta release where we would then get a larger group of play testers.

Another high-level system that needs special testing would be the server-side asset download. Pre-release, we would need to gather a range of devices – probably our own, and the devices of willing stakeholders – and use them to locate issues with installing to weed out OS specific bugs. This will fulfil the requirement for the workshop ambassador – which is to make sure installation is easily explainable. The automation of this process would make it so that the users shouldn't even have to interact with the app (except for opening the application and granting permissions). For a barometer, if we can seamlessly (without needing aide from a team member) install the application with all its assets on at least 8 different Android devices, we would be able to classify the application as “easily installed”.

OO Design & UML







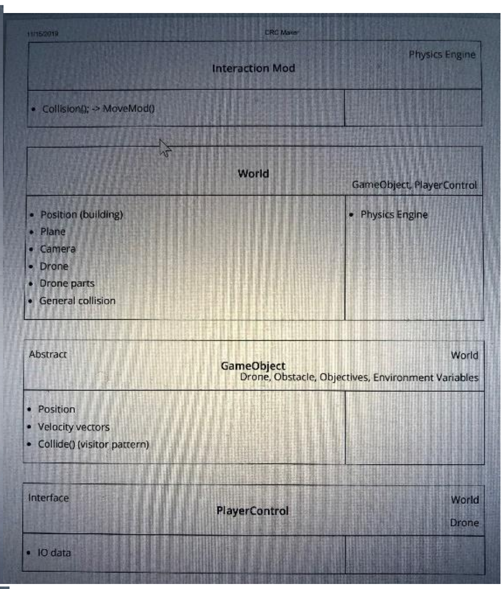
11/15/2019 CRC Maker

Physics Engine Collision Check, Player Control Mod, Interaction Mod, Movement	
• Position (drone) • Input vectors • Output vectors • Collision detection • Player vectors	• World • Controls

Abstract **Movement**	Physics Engine
• Init Pos • Init Vel • MoveMod()	

Player Control Mod	Physics Engine
• Keypress -> MoveMod()	

Collision Check	Physics Engine
• Check Collision()	



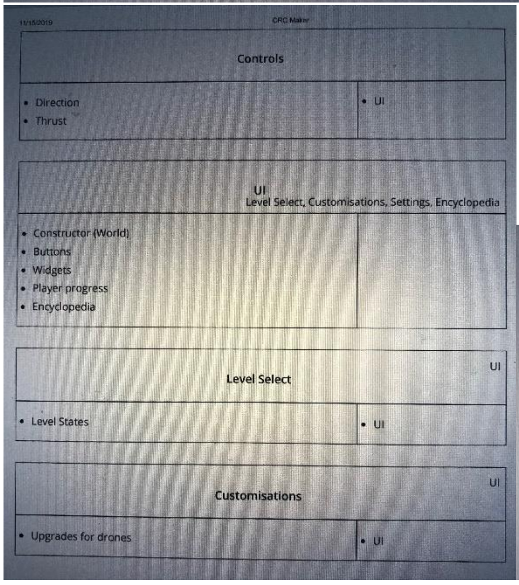
11/15/2019 CRC Maker

Interaction Mod	Physics Engine
• Collision(); -> MoveMod()	

World GameObject, PlayerControl	
• Position (building) • Plane • Camera • Drone • Drone parts • General collision	• Physics Engine

Abstract **GameObject** Drone, Obstacle, Objectives, Environment Variables	World
• Position • Velocity vectors • Collide() (visitor pattern)	

Interface **PlayerControl**	World Drone
• IO data	



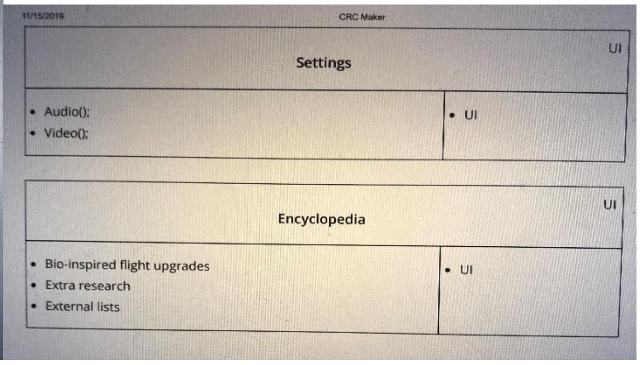
11/15/2019 CRC Maker

Controls	
• Direction • Thrust	• UI

UI Level Select, Customisations, Settings, Encyclopedia	
• Constructor (World) • Buttons • Widgets • Player progress • Encyclopedia	

Level Select	UI
• Level States	• UI

Customisations	UI
• Upgrades for drones	• UI



11/15/2019 CRC Maker

Settings	UI
• Audio(); • Video();	• UI

Encyclopedia	UI
• Bio-inspired flight upgrades • Extra research • External lists	• UI

Initially, to briefly explain the mechanisms of our project, we decided to make a simple flowchart to explain how each component interacts which each other. The flowchart is separated into two distinct parts: the UI component and the game logic. When the game is first loaded, the UI component of the program will initialise in order to show off the main menu. Before the game starts, the game overlay will be put into action. The game overlay serves as the intermediary to both the UI and game logic components as it allows a smooth transition for when the program switches from the UI component to the game loop. Hence it is the only field linked to both components. After the game finally starts, the game loop will run. For each frame, the program will update the movement, the game physics, the in-game collisions, the environment as well as the camera before going back to the game overlay.

Furthermore, we have a global idea of which object-oriented design patterns will be used in this project. Firstly, there is an abstract superclass called GameObject which deals with the different in-game objects' position, velocity and collision. The visitor pattern will deal with the different collisions between the different game objects. For the overall system, the Model-View-Controller pattern will be used. The abstract class MoveModifier will be handed the task of modifying movement. The observer pattern will be used between the World and Renderer components. Lastly, for the different tasks, a composite pattern will be used.

These design choices have potential advantages and disadvantages. Separating the UI and the game loop components will likely hinder loading times, but in return the game is likely to run smoother. With the game overlay being the only link between the two components, the program will rely heavily on it, which means getting the game overlay right is very important. We went with this design because we think game performance is more important than loading times or potential issues with the game overlay component. However, there are still uncertainties with this design layout. For example, we are still unsure of how we can make the research readable to others, and there is still a lot of uncertainties about concurrency such as how we are going to implement it and if we’re going to be able to implement it since it is still in acquisition.