EXTENSION REPORT

ARCHITECTURE AND DESIGN

ADDITIONS/CHANGES TO REQUIREMENTS

The complete extension of the software we have chosen, 'When Planes Collide[1]', demands that the additional requirements received from the client be satisfied in the final product. The additional requirements are:

AER 1. Each airspace in your game must contain at least 2 landing and take-off options.
AER 2. Add a multiplayer mode to the game, so that (at least) two controllers may compete.
AER 3. Add a handover facility to the game, so that one player can pass responsibility for a flight to the other player.

These requirements are additions to previous requirements [2] given by the client. The non-functional requirements[2] which were used in previous assessments have not changed and are still being used for testing both the initial game and resulting game. The aim of this extension project is to create a solution that effectively builds on the original solution we have received, while ensuring that the previous requirements are still satisfied where practical and additional requirements are satisfied.

Though we intended to keep many previous requirements[2] satisfied in the extended game, satisfying some of these previous requirements would have hindered the success of the extended game. Also, some of these previous requirements were not used because the element which they referred to has now been updated to improve the user's experience with the software. These requirements and the rationale behind them being exempted from influencing the extension of the software are documented in the test report.

The process of eliminating these requirements was part of our effort to modify the previous requirements to be in accord with the additional requirements we received; we did this so they could be effectively used for regression requirement testing. The process of eliminating requirements that would not be used is also a part of our basic software extension engineering approach[3] specified in assessment three.

SR10 which required the game to automatically save scores has been removed as a requirement, this is because we do not intend to have a high scores list in the menu and SR10 would have been part of implementing this feature. SR11 has also been removed for this same reason.

SR15 which required the ability to alter a flight plan has been removed, this is because the game's controls have been switched from mouse to keyboard controls, this makes implementing this feature unfeasible and also may not have been desired in multiplayer gameplay. SR24 which required the game to simulate bad weather to provide a greater challenge to the players has been relaxed because it is difficult to implement and test, and is an extra feature which is not essential for basic gameplay. SR25 which required the simulation of equipment failures has also been relaxed for the same reason.

ER4 which required the implementation of restrictions on how many flights could land and take off at the same time has been removed, this is because the separation rules already enforce this restriction. ER7 and ER8 which are related to the implementation of a high score list have also been removed and are not implemented because the high score list is not required to satisfy the customer's requirement. This was also an extra feature that would have been implemented if there was more time.

[1] Game by Team PSA
[2] Previous requirements listed in appendix.

CHALLENGES

1. **Preliminary testing:** As with assessment 3 and in accordance with our basic software extension engineering approach[3] , the first thing we did to approach extending the software we received was to conduct preliminary testing -initial regression testing[4]. During this process, we ran the built-in automated tests that came with the code base. We received a total of 107 tests from team PSA and on the initial run of these tests; we had an overall 12.15% success rate. The problem with this result was that it meant we had inherited error filled built-in tests that could not be used for regression testing.

   Our first approach to dealing with this problem was to disable 51 tests[5]. Some of the tests that were disabled included those that were involved in testing the addition of flights to an airspace, the error in these tests was found to be due to an infinite loop in the function that builds the flight plans, this function was rewritten as a for loop instead of the initial while loop and the tests that were linked to this function were re-enabled. However, this solution only increased the overall success rate to 23.21%.

   To achieve a higher success rate, the tests that tested graphics content were removed, because there was no potential for them to ever work in the JUnit environment, and more automated tests[5] were built. After evaluating and modifying the automated tests we received, we were left with 65 tests, all of which are effective and give a 93.85% success rate.

   Further tests were removed, reducing the number of initial regression tests to 63. However these tests were not removed as part of solving this challenge/problem, they were removed because the functionality they were linked to was removed from the software as it was not needed to satisfy the customer's requirements; the functionality was an extra feature.

2. **Deciding game plot/mechanics:** Deciding the ideal game plot/ mechanics that would satisfy the client's requirements was challenging. Deciding the game plot involved defining the scoring/penalisation procedure and rules because these determine how the players perceive the game play or their aim/goal in the game. This was difficult because of the client's supplementary requirement for the multiplayer game to be "competitive and collaborative". This gave rise to a lot of brain-storming[6] within the team and also frequent meetings with the client. Striking a balance between these two opposite ends of the spectrum initially seemed impossible, but we were able to draw up a game plot[7] (solution) within the team and negotiate this with the client who accepted our solution.

3. **Time management:** Time management during the completion of this 4th assessment was a challenge. This was majorly due to receiving error filled regression tests, the team not being able to arrive at a suitable game play decision on time, and also due to a stall in work progress during the Easter holiday. This created a backlog of work that needed to be completed before the deadline date, increased the work pressure and also meant we didn't have much time to arrange meetings with our client as we would have preferred to. We responded to this challenge by implementing an agile methodology which meant that we organised out time and tasks by implementing sprints[8], each member had to inform other team members what they intended to complete by the next meeting; each team member's goal was then documented for the team to refer to. In general, we resolved this issue by abandoning our time plan from the 1st assessment because it couldn't deal with time frame and workload we were now dealing with, and resolved to contacting our client via email rather than arranging meetings (with an exception to when we conducted the acceptance testing).

---

[3] The software engineering approach diagram used in assessment 3 is available in the appendix.
[4] Initial regression testing is documented in the test report
[5] The tests are documented in the test report
[6] Possible ideas that rose from brain-storming sessions are in the appendix
[7] Final game plot is described in detail in the 'Gameplay' section of this report
[8] Agile sprints and initial time plan are in the appendix

4. **Deciding implementation:** Deciding how to implement the multiplayer functionality was also challenging, we had a few substantial decisions to make and various options. The cores of the client's requirement for a multiplayer game could majorly be satisfied by a real-time based game, so it was easy to decide between implementing a real-time versus turn based game. However, the main challenge in this area was deciding if the multiplayer game would be implemented to work using a single computer or two computers.

   In the case of using a single computer, we had to decide if we were going to implement the need for multiple player controls as keyboard vs mouse or, if we were going to split a keyboard into two different controls; dividing the keys amongst the two players. The latter was discussed to be a much better approach because it meant no player could be at a disadvantage due to the type of control device they were using.

   In the case of designing a game that required the players to be using different computers, we would need to implement a network (client-server architecture) design. This idea was a potential idea because it required less editing of the GUI.

   To solve this challenge, we decided to look at all these options and decided to implement a game design that would work using a single computer, where the players would be sharing a single keyboard. This decision was made because of the potential networking problem we could have faced with matchmaking, especially since we doubted the ability to complete such a design (given all the possible problems) within the time frame we had.

GAMEPLAY

- **The Initial Gameplay**[9]

  Control: The original software we received was a single player game, it had three difficulty levels; easy, medium and hard. The player experienced the world as an air traffic management controller through a bird's eye view of the airspace. The player could navigate the planes freely in the airspace, where the planes were subject to physical laws and effect that dominate the airspace e.g. crashing.

  The game was organized using a flight plan of waypoints that was randomly generated, and the aim of the player was to successfully navigate each flight along the waypoints specified on their flight plan, and out of the airspace through the exit point specified. The more way points they flew through, the more points they accumulated. The player had the control of landing and changing the altitude, bearing, speed and flight plan of each flight. If two planes were too close, a red line was drawn between them, warning them of a potential crash. If the player is unable to avoid the crash, the planes explode and the game will then come to a halt.

  Scoring and Achievement: In the initial game, 'When Planes Collide' by team PSA, the player got 60 points for every minute played. For every waypoint they passed through within 30x30 pixels, they got 20, 50 or 100 points, depending on how close they were to the waypoint. Negative points were given when the player changed the flight plan (-10), and when the player lost the plane or it did not go through the exit point (-50).

  Most importantly, the initial game we received had a gameplay that satisfied all the client's requirements till the date it was received.

- **Description and justification of extended gameplay:**

  Control: The final product of this extension project is a multiplayer game, with a maximum of two players at a time. Each player controls a set of colour coded flights using their designated block of keys on the keyboard, and the objective for each player is to gain points in a similar manner to the single player experience explained above.

[9] More information about the gameplay can be found in the user manual attached to this document.

Changes in the gameplay from the single player game to the multiplayer game include the addition of a handover command by which one player can pass control of their selected flight to the other player; this is a purely unilateral decision. In order to discourage repeated handovers and prevent one player from swamping the other with their flights, there is a score penalty applied for every time a player initiates a handover. There is also a five second cool down period randomly placed on each player every time they complete a handover; during this period, the player cannot use the handover command.

As opposed to the initial single player game, when two planes crash in this multiplayer version, the game doesn't end but there is a significant score penalty applied to the owner of each plane involved in the collision. The game ends when one player reaches 3000 points; that player is then deemed the winner. There are also two different airports in this extended version of the initial game. Each player can take-off and land their planes in any available airport of their choice. However, in order to land a plane, it must be at its minimum speed and altitude; 1000ft and 200 miles per hour, respectively.

Scoring and Achievement: Both players individually gain from events involving their flights in the same manner as the initial single player game barring the two exceptions detailed above; handovers and crashes. The game ends when one player reaches 3000 points, at which they are declared the winner. When a crash occurs 500 points are deducted for each plane from the player or players involved in the crash. The handover penalty is 50 points for the player who initiates the handover, the same penalty as when a flight leaves the airspace without having completed its flight plan.

Justification: The extended gameplay satisfies the client's requirements. The multiplayer feature in the game satisfies the client's requirement, AER2; "add a multiplayer mode to the game, so that (at least) two controllers may compete". This extended feature is supported by having each player control a set of colour coded flights (red or blue), the set of controls for each player are implemented by two separate blocks of keys on the keyboard.

The addition of an extra airport is another feature that has been implemented in the extended game in order to satisfy the client requirement, AER1; "each airspace in your game must contain at least 2 landing and take-off options". AER1 is satisfied by having two different airports; BHD and DHB airport, to support the landing and take-off of each player's set of planes.

The client's third requirement, AER 3; "Add a handover facility to the game, so that one player can pass responsibility for a flight to the other player" is satisfied by the handover feature implemented in this extended version of the initial game, this handover command allows one player pass the responsibility of a flight to the other player. To avoid one opponent swamping the other and ensure that the game was still fair whilst being competitive, we have also implemented a cool down feature to support the handover command; The scoring and penalisation mechanism has been implemented to encourage players to aim at gaining points in the game and following the game plot, hence a competitive and immersive experience. The satisfaction of this requirement means that the customer's previous requirement to end the game once a collision occurs no longer holds in the game play.

Previous requirements from assessment two and three are also satisfied in the game play if they are practical and do not counter the multiplayer feature. The list and details of these previous requirements and how they are satisfied in the game play are documented in the test report.

DESIGN OF THE EXTENDED GAME'S INTERACTION

- Menu: There are four buttons presented to the user in the menu interface; 'Help', 'Versus mode'- multiplayer, 'Challenge mode'- single player and 'Quit'.
- Help: When the player clicks on the help button, the game displays a screen which describes the mechanics and controls of the game to the user-the user manual. This event is handled by the Menu state class in java which handles the game state. When the help button is clicked, the conditional statement which handles all four buttons in the menu handles this request accordingly. This is opened in the web browser.

- Quit: The quit button basically quits the game and closes the window using the command given in the conditional statement when the quit button is clicked.
- Versus mode (Multiplayer mode): When the 'Versus Mode' button is clicked, the game goes into the multiplayer state and the Multiplayer state initialises the airspace which contains the airport, entry and exit points.
- Challenge mode (Single mode): When the 'Challenge Mode' button is clicked, the game goes into the single player state and this state initialises the airspace, the airspace contains the airport, entry and exit points.
- The game is entirely keyboard driven, when any flight is created by the Update method, it will be automatically designated to a "red" or "blue" controller. If there is a flight designated to a controller in the airspace, the flight will be automatically selected by the controller. i.e. provided a controller has been designated a flight(s) that is in the airspace, they must select one.
- Controls:
    - The buttons are arranged into blocks on the keyboard to allow for multiplayer usage; it is possible for each player to control their designated planes effectively with only one hand.
    - All the controls are stored in a key binding's class in Java which allows for easy modification of the controls (maintainability). Controls are handled in the controls.java class.
    - The update method in controls.java listens to the keyboard and the event of a key press, causes an event which calls the method needed by the action bound to this key.
    - The interactivity of the game over screen is dependent on the fact that a player could either play another game or just quit.
        - Red player
            - Climbing = Key W, S= descending, A= Left, D=Right, E=Accelerate, Q= Decelerate, X and C= Change flights in the controllers list of designated flights, V= Context sensitive for the airport (Landing and takeoff), B= Handover.
        - Blue Player:
            - Climbing = up arrow, descending= down arrow, Left= left arrow, right= right arrow, [: decelerate, ']': accelerate, '.' and, ',' change flights in the controllers list of designated flights, L= Context sensitive for the airport (landing and take-off), ' ;'= handover.
- When a player goes through a waypoint successfully they gain points. In the multiplayer mode, once a player has reached 3000 points they are declared the winner and the 'win' interface is displayed. The 'win' interface gives the players the options of playing the game again, quitting or returning to the main menu.
- When there is a crash in the 'Versus mode', points are deducted for each plane from the player or players involved in the collision and the game continues. In the challenge mode, the game ends. And the game over interface is displayed. The game over interface gives the player the options of playing the game again, quitting or returning to the main menu.
- In the event of a handover being initiated by a player, their selected flight is handed over to the other player on the basis of a unilateral decision. The game doesn't pause during this process, and there is a cool down period for each player where they can't handover any of their flights. The cool down period is displayed as a countdown at the bottom of the screen.
- Each player is able to identify their planes through the colour coding used in the game. The planes in the airspace are either red or blue.

Justification of interaction design: The design of the games interaction helps with making the goal of the game obvious; the theme of the GUI as well as the penalisations and scoring mechanism have been implemented to help a player understand the game plot. The inclusion of a score for each player, and scoring and penalisation mechanisms that apply to each players score ensures that the game is competitive but also discourages players by penalising actions that could potentially monopolise the game, ensuring that the customer's requirement for the game to be "competitive and collaborative" is satisfied. Another requirement that is satisfied by the availability of a score for each player is AER2- "Add a multiplayer mode to the game, so that (at least) two controllers may compete". This requirement is also satisfied by the option of a "Versus mode" in the main menu.

The two separate sets of keys with identical functions that have been programmed to work in this game; one set to each player, also support the implementation of a multiplayer mode in the game. This is because they ensure that the two players can both control their designated flights to the same degree hence facilitating the multiplayer mode and ensuring AER2 is satisfied. The colour coding of each flight to indicate which plane/planes are controlled by which player, and the compulsory assignment of each plane to a player also facilitate the satisfaction of the requirement, AER2.

The interaction design was sketched in a scrum meeting before being implemented to ensure coherence[10].

MODIFICATION OF THE SOFTWARE

Code libraries/ Framework: Our game was built on team PSA's assessment three game, which was itself built on team WAW's assessment two game. All of these games used the Slick2D library to handle graphics and user input. As with every other team's product, our game is built using Java.

- **Achievements.java:** This class has been removed. This is because we wanted to keep as many common features as possible between the single player and multiplayer games and there was no obvious place for an achievements system in the multiplayer experience; especially as the amount of work required to convert this code for multiple instances (one for each player) was not worth attempting given that there was no requirements given by the client for an achievement system and we were working to a tight deadline.
- **Airport.java:** The Airport class has been modified to be a subclass of Point which makes it simpler for other functions to access the location of an airport. The class's constructor has also been modified to allow for multiple airports in the airspace. There is also one new function, getInverseRunwayHeading, which is used to simplify the code that allows the flights to land from either end of the runway.
- **Airspace.java:** The Airspace class has been modified to include an extra field (isMultiplayer) which is either true or false and is set when the airspace is constructed depending on whether it is constructed from the PlayState or the MultiplayerState. Taking into account the fact that flights now have an owner; either "red" or "blue," or "single" for flights in the single player mode, new functions have been added; isFlightWithOwner which returns true if a flight exists with a given owner, and getListOfFlightsWithOwner which returns only the flights with a given owner. These are used heavily by the Controls class as each Controls instance is only concerned with one person's flights. The newFlight function has been modified slightly so as to randomly assign each new flight to either the red or the blue player in multiplayer mode. There are also changes to allow multiple score tracking instances, one for each player in the multiplayer mode; consequently the getScore function was modified slightly to take a parameter to determine which player's score to return. There are also a few changes to allow multiple airports in the airspace including in the render function and in theNewFlightFunction when generating the entry point for a new flight. The rate at which flights spawn was increased slightly so as to counteract the perception of flights not being created quickly enough in the multiplayer mode. This class also controls the handover cool down period , this feature is controlled by these functions in this class: resetRedHandoverCountdown, resetBlueHandoverCountdown which are called by the flight class when a handover is initiated, and isRedAbleToHandover, isBlueAbleToHandover which are queried by the flight class to determine if a handover is possible.
- **Controls.java:** The Controls class was completely rewritten to support keyboard controls as opposed to the purely mouse driven controls implemented by the previous team (team PSA) which would have not been practical for a single computer multiplayer game of the type we wanted to build. There are two controls instance in the multiplayer mode; each controls instance deals with flights belonging to a single player and is passed a key bindings object on construction which determines which keyboard keys the controls instance would monitor. The controls instance also draws the circle around each selected flight and indirectly causes the line showing the flight plan of the selected flight to be drawn.
- **EntryPoint.java and ExitPoint.java** : Although we have added an extra exit point instance and an extra entry point instance for implementing the second airport, the actual classes themselves have not changed.
- **Flight.java:** The main change to the Flight class has been the addition of the owner field which stores the flight's current controller - this can be "red" or "blue" or "single". The render function has also been changed to draw flights

[10] The sketch of the interaction design is included in the appendix.

in different colours depending on their owner. The existing system for colour coding flights by their speed has therefore been removed and the speed is instead drawn below the flight's altitude next to the flight's icon on the screen. The speed colour coding system was removed because a flight icon can practically not be coded by two different things; can either be red or blue but not orange to prevent confusion. Changes have also been made to the land function so it's supports multiple airports.

- **FlightMenu.java:** This class was for the radial menu implemented in team PSA's 'When Planes Collide' and has been deleted because we moved the controls to be implemented using a keyboard rather than a mouse. The radial menu is not functional in the multiplayer game.
- **FlightPlan.java:** This class has not been changed because it already supported multiple airports by virtue of their attached entry and exit points.
- **Point.java:** This hasn't changed and its function still remains to store the x and y coordinates of classes that extend it.
- **ScoreTracking.java:** This function has been changed to allow multiple instances, and a new function, applyCrashPenalty to the controllers of a crashing flight in multiplayer. To accommodate the penalty for handovers, the existing penalty for flight loss was used.
- **SeparationRules.java:** This class has not been modified; the change to crash behaviour for multiplayer was handled purely by handling the output of the existing getGameOverViolation function differently in the multiplayer state i.e. it takes points off instead of showing the game over screen.
- **Waypoint.java:** This class has not been modified.

QUALITY ATTRIBUTES OF THE SOFTWARE ARCHITECTURE

- **Performance:** The software architecture of the extended game has a high responsiveness; it responds to events instantaneously. For example, when a player presses a key from their set of keys, the function linked to the key occurs instantaneously. This can also be seen in the scoring and penalisation which happens instantaneously as well. This attribute of the software is obvious to the human eye.
- **Reliability:** The architecture of this software is reliable. Its performance does not depreciate over time as it is being operated. The architecture of the system never fails therefore, it is always **available**.
- **Security:** The system is local and doesn't have any links to the web. This makes the system immune to online threats.
- **Modifiability/ Modifiability:** The software's architecture supports quick and cost effective changes being made to the system. For example, the software comprises of logical classes which consist of code that have substantial comments. The game has also been built to simply work locally on computers so changes can be made to the code locally without having to access servers.
- **Portability:** the software architecture is portable to a practical extent; it can be run on laptops and desktops but not on mobile phones or tablets.
- **Functionality:** The system does all the work it was intended to do, this can be seen in the test report where units have been tested and give a positive result, as well as in the requirements validation where all the clients requirements are broken down into systems requirements and give a positive result.
- **Reusability:** The architecture of the system has been built in a way that makes expansion and modification feasible. The various components of the system can be removed/ replaced to create a different version of the game. The game's architecture can serve as a foundation for related products, as in a product line.
- **Subsetability:** This is a significant attribute of the system's architecture, especially since an agile methodology was implemented. During the system's building process, a minimal system which had completely satisfied all the client's requirements was made to run early on at each meeting and functions were added to it over time until the whole system was ready and satisfied all the client's requirements.
- **Conceptual integrity:** The design of the system's architecture is consistent and coherent overall. The logical classes, the interface and other components of the system all follow a unifying theme- Air Traffic control theme; classes are named according their purpose relating to Air traffic and the interface has been built to mirror this. The conceptual integrity of the system also extends to the naming of variables in the source code as well as the object orientated coding style that is used throughout.

- **Testability:** It is easy to create broad test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. This attribute can be seen in the test report which tests the system and its components thoroughly, and show that faults in the system can be isolated in a timely effective manner.

Justification: These quality attributes being satisfied in the system's architecture ensure that the non-functional requirements documented for this system are satisfied. More evidence is given in the test report which documents these non-functional requirements in detail, and their tests and results accordingly.

GUI

Changes that have been made to the GUI are as follows:
Firstly, the main menu has been modified to add a versus mode option, this provides the player with a quick, simple and succinct method for selecting their preferred game mode, whether it be single or multiplayer.
The main menu's graphics have all been updated, which gives the game a cleaner and more professional look relating to the game theme. This was done to give the player a more aesthetically appealing experience when playing the game and to differentiate our branding from team PSA's and other teams that adopted their product.

Figure 1 Main menu interface of extended game

The sidebar/panel which was seen on the left of the initial game's interface has been removed. This is because the sidebar leads to a "crammed" airspace once you add in a second airport. Removing the sidebar was necessary when complying with the new requirements of making the game multiplayer, especially requirement AER1[11]. All aircraft commands are now implemented with keyboard input and therefore the mouse controls have been removed. The modifications to the controls also resulted in the elimination of the radial menu around a selected plane, due to it not being functional and it being redundant.

Figure 2 interface of previous game

To further satisfy the new requirements there has been an addition of another airport and a relocation of the old one away from the centre to accommodate it. They have been placed far enough away that the player would be able to avoid confusion when using an airport.

In addition, the planes have been modified so that if a plane is selected, its waypoints are now indicated by lines that start from the plane icon, go through all its waypoints and end at its destination. The waypoints for a selected plane were originally displayed in the sidebar, but with its dismissal from the game, the waypoint lines seemed the simplest and most logical choice for the player. This allows the player to remain concentrated on their plane, changing its direction instantly, without the need to search for specific waypoints across the game screen.

To further assist the player to provide them with information quickly and concisely, the speed of a plane has now been added to show above a plane when it is selected. This was necessary due to speed no longer being displayed in a sidebar. Further

[11] Each airspace in your game must contain at least 2 landing and take-off options.

information such as the timer and score that would have been removed along with the sidebar, have now been placed on the top left of the screen. This maximises gameplay space and the score for the red and blue player has been colour coded accordingly, satisfying AER2[11].

When versus mode is selected in the main menu, the planes have been modified so that they are either red or blue, depending on which player they belong to. This simplifies the game further for the player(s), it allows them to quickly know which plane is theirs and make the appropriate quick decisions that are necessary when playing the game. This feature also supports the objective to satisfy AER2[12].

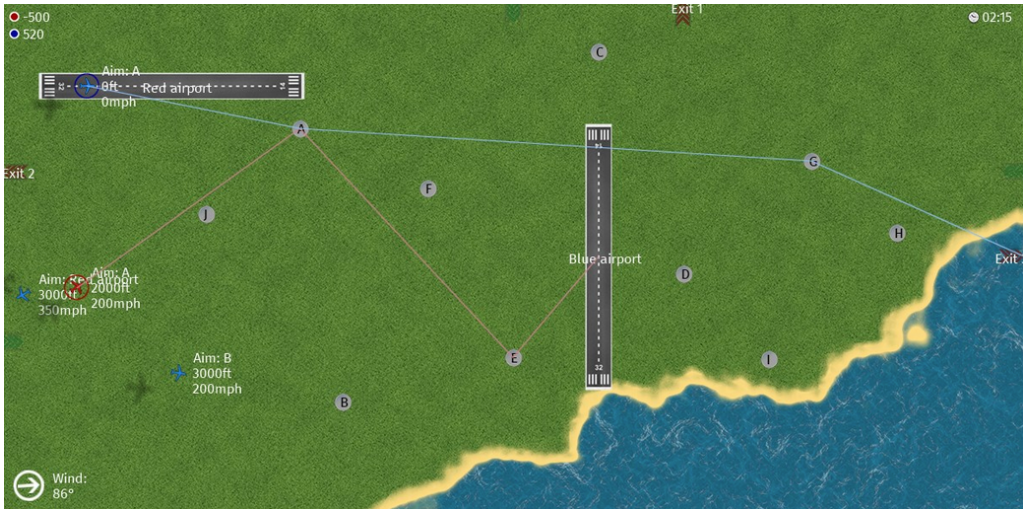


Figure 3 Screen shot of extended game GUI

The GUI has been designed to support interactions with users, it supports the tasks users aim to carry out. Each user's manoeuvring of any flight is reflected in the GUI design. For example, when a user presses a key that has been programmed to turn the corresponding selected flight left, the GUI is updated to show the flight turning left. This example also cuts across the various control commands that have been made available to the user in the game. A closely linked example is how the scores are updated in the GUI accordingly with events that lead to penalisation or increase in a player's score.

The overall engineering process used to design the GUI was user-centred software engineering. The decisions made on how the GUI should reflect a player's interaction with the system were based on what, how and why it would be best for the user. For example, succinct information is relayed to the user on the GUI, because users generally don't want to have to divert their attention to reading a block of text, as it will distract them from the objectives in the game and may reduce their score.

ADOPTED SOFTWARE ENGINEERING SOLUTIONS AND APPROACHES

This project is based on the extension of existing software, a traffic controller management game. We are the third team to work on this software; the initial software was built by team WAW as 'Don't Crash' to satisfy the client's initial requirements[13]. The software was then extended using additional requirements[12] given by the client, team PSA handled this extension and labelled the extended software as 'When planes collide'. The aim of our team, team BHD, is to successfully extend the software further to satisfy the extended additional requirements given by the client for this third phase of the software's development.

As opposed to the initial time plan[14] which suggests that the waterfall model will be used in this stage of the assessment, we implemented an agile approach. We made this change because compared to the other stages in the SEPR assessment; this stage appeared to have a higher degree of complexity, and building a multiplayer game was new to us as a team. The imposed deadline also meant that we needed to work in short iterations to keep the intensity and focus going on in the project, as well as discover errors as soon as possible by testing and producing functional modules as part of each sprint. The major factor for this decision was realising we had three weeks left until our deadline and we still hadn't made any decisions about the design and documented any changes to the software we received. This factor meant that we were totally behind schedule with regards to our initial time plan, and as planned in our risk assessment[15], we mitigated this by rescheduling; while doing this we took into consideration that we were now using an Agile methodology as opposed to the waterfall model that was enforced in the time plan, this meant we had shorter deadline dates for smaller chunks of the project.

[12] Add a multiplayer mode to the game, so that (at least) two controllers may compete
[13] The client's requirements for the previous two development phases can be found in the appendix
[14] Initial time plan for this phase is included in the appendix
[15] The risk assessment document can be found in the appendix

Implementing an agile methodology gave room for experimentation, improvement and reprioritisation; all of which were needed to tackle a task with a high complexity and one which was new to us as a team. We also involved the client (user) throughout the process of designing and implementing the changes that needed to be made, this was done to ensure that we were on the right path to producing an extended software that satisfies all the client's needs given for this phase of the software's development. The overall design of the product was client-driven.

To use this methodology, we made effort to document what each member of the team should have completed by the next meeting[16]. In these meetings, we spoke about the problems we were encountering with completing our designated tasks, how to deal with them and came to a conclusion on how to approach the tasks that should be completed for the next iteration. These meetings were held twice a week; on Tuesday and Friday.

The period of completing designated tasks before each meeting was like a sprint in agile technology and the meetings were to implement scrums meetings. The main features of our implementation of this software engineering approach in this project were; continuous improvement through time-boxed iterative deliveries and reviews, implementation of most important functionalities first and constant collaborative communication. Our informal conversations were held via Facebook[17], while the formal ones were done using E-mail. Our scrum meetings were held in a computer laboratory and most of our online discussions were informal.

We used ArgoUML to create the UML model diagram[18] for the product. Dropbox was used in the early stages of the project to convey and debate possible GUI designs for the product[19]. Bitbucket was used as our code repository to ensure that every team member could access it; that changes to the code could be recorded and seen by each member, and ensure that the code was consistent and coherent throughout. Google Drive was used as a repository for our reports and plans to allow every member access to reports and suggest or implement changes without the team losing track of previous versions.

In general, our initial approach was to use the software engineering approach specified in assessment three[20]. Although we did all the procedures the approach specifies, after carrying out the preliminary testing and modification of requirements accordingly we decided that we had to implement an agile methodology. This meant that instead of going through the different procedures accordingly, we did a number of cycles through the other processes in the specified approach, in no particular order; this gave us flexibility which was much needed. For example, we would come to a conclusion about the design of a particular functionality at a scrum meeting, review its impact analysis on testing and, the existing design and system architecture, and then it would be implemented during a sprint and then reviewed (tested) in the next meeting. This is a cycle that happened a number of times with respect to different attributes of the assessment, including the compilation of reports.

In terms of team leadership, we divided the whole assessment into four parts and assigned leaders accordingly; extension report- Aishat, GUI documentation-Chris, Implementation- Tim and Radostin, and test report- Katie. We did this to ensure that each part of the assessment was led effectively and so that everyone felt involved and took part in the project.

We carried out various tests to ensure that the system and its various components were tested thoroughly. We carried out pre-extension and post-extension testing. Testing consisted of regression testing, requirements validation, unit testing, and acceptance testing accordingly. All these tests are documented fully in the test report which should be referred to for more details.

[16] A screenshot of the documentation of tasks for each meeting is included in the appendix
[17] Evidence of Facebook chats included in the appendix
[18] The UML model diagram for the software is included in the appendix.
[19] Initial proposed GUI design is in the appendix
[20] This approach is included in the appendix

BIBILOGRAPHY

[1] “Introduction to Multiplayer Game Programming”, Book of Hook. [Online]. Available: http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/IntroductionToMultiplayerGameProgramming . Accessed: 20/04/2014.

[2] “Evaluating a Software Architecture”, PTG Media. [Online]. Available: http://ptgmedia.pearsoncmg.com/images/020170482X/samplechapter/clementsch02.pdf . Accessed: 01/04/2014.

[3] Team BHD Assessment 4 Test Report.

[4] Bob Hughes, Mike Cotterell , Software Project Management. Berkshire, England: McGraw-Hill,2002.

APPENDICES

Appendix 1:

Previous Requirements

Extension requirements (Assessment 3):

1. An aircraft can land at an airport - **ER1.**
2. An aircraft can take off from an airport - **ER2.**
3. Airports must exist and be visible to the user - **ER3.**
4. There must be constraints on how many aircraft can land or take off at any one time - **ER4.**
5. Flight plans must be able to enter and land at an airport or take off from an airport and exit - **ER5.**
6. The score must be displayed for a particular game while it is being played - **ER6.**
7. When a game has finished, the score should be added to a list of high scores - **ER7.**
8. The list of high scores should be available for the user to see - **ER8.**
9. At least ten flights should be allowed in the airspace at any one time - **ER9.**

Initial requirements (Assessment 2):

10. Have access to an in-game "help" screen - it is important for new users to be able to understand the mechanics of the game - **UR1**.
11. See airspace - the game is simply meant to emulate the daily work of an air traffic controller - **UR2.**
12. Start the game after examining the airspace - since the airspace in the game might be new and complicated for the player, they should be able to examine it before getting into the game - **UR3**.
13. Alter the course of flights that enter the airspace - without this ability, the game will lack any gameplay whatsoever - UR4
14. Direct flights towards waypoints - the simplest way to direct flights - **UR5.**
15. Have flights land and take off - a "landed" flight is a possible way to score points - **UR6**
16. Know how much time they have spent in the game - a way to help the player score and weigh down their own performance without necessarily ending his game **- UR7.**
17. See a score, which will be a way for the player to assess their own performance **– UR8.**
18. Have a feeling for "challenge" - provide an online leader board to create an impetus for users to play the game - **UR9.**
19. Be able to "lose" - without a challenge and a risk of losing, most games are perceived poor by critics and players alike - **UR10.**
20. Be able to quit and see a high score without "losing" - a simple preventive option, meant to be used if the player would want to quit playing before actually losing the game - **UR11.**
21. Be able to pause and restart the game "at will" without exiting it - **UR12.**
22. Have a main menu - the easiest and most common way for interfacing between the player and the game prior to the launch of a game session - **SR1.**
23. Have a help/instructions screen - in direct relation to UR1 - **SR2.**
24. Have a "start" button for the user to start the game - in direct relation to UR2 - **SR3.**
25. Generate an airspace - without having the system create a random or predefined airspace, no gameplay can be conducted - **SR4.**
26. Simulate airspace graphically - the best and most engaging way to present the game to the players - **SR5.**
27. Populate the airspace with flights - without flights, the game will be exempt of gameplay -**SR6.**
28. Have a GUI with a score and a timer - a GUI allows for easy and engaging control of the game by the user - **SR7.**
29. Have varied characteristics for flights - variety is a great way to improve the player experience - **SR8.**
30. Display score - in direct relation to UR8 - **SR9.**
31. Save score - in direct relation to UR9 - **SR10.**
32. Upload score to a master repository - in direct relation to UR9 - **SR11.**
33. Monitor separation rules - separation rules are an authentic way to provide a challenge for the players and challenge is very important, as previously stated - **SR12.**
34. Generate and remove flights from the airspace via entry and exit points - the game must persistently give the player flights to manipulate - **SR13.**
35. Pre-set a flight plan - when flights come into the airspace they should have a sensible flight plan - **SR14.**
36. Alter a flight plan - players should be able to manipulate the flights in the airspace - **SR15.**
37. Change course - in direct relation to UR4 - **SR16.**
38. Change altitude - in direct relation to UR4 - **SR17.**
39. Land at airport - in direct relation to UR6 - **SR18.**
40. Take off from airport - in direct relation to UR6 - **SR19.**
41. Turn left or right by particular degree - in direct relation to UR4 - **SR20.**

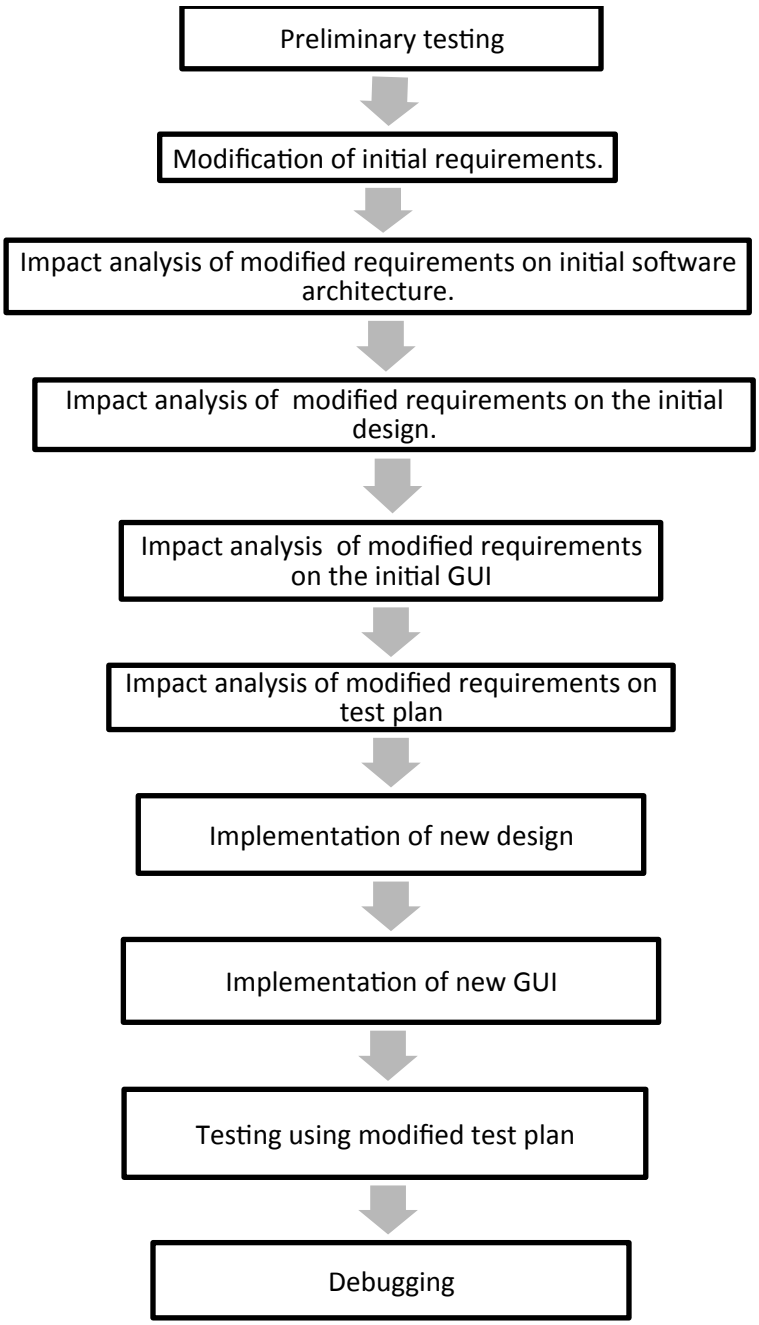
42. Display updates regularly - if updates are not regular enough, the game will feel unresponsive and “stutter”-y, which will decrease player immersion and enjoyment - **SR21.**
43. Have waypoints - in direct relation to UR5 - **SR22.**
44. Occasionally set flights on crash course/near miss - without this feature, the game will mostly lack challenge - **SR23.**
45. Simulate bad weather - a feature that will provide greater challenge to players - **SR24.**
46. Simulate equipment failures - a feature providing greater challenge to players - **SR25.**
47. Have a quit button - the player must be able to exit the game seamlessly - **SR26.**
48. Game becomes harder as the timer goes up - to provide an increasing challenge arc and keep the player engaged - **SR27**.
49. Have a pause /resume functionality - SR28.

Non-Functional Requirements

These aspects of the game will increase player enjoyment and immersion, but all these requirements need not be satisfied in order to have a working product:

- **Efficiency** (reflect changes within 30ms) - Faster updates will provide for better and more agile decision making on the users’ part.
- **Documentation** - Made even more important due to code switching.
- **Testability –** i.e. can we test that we have fulfilled our requirements?
- **Stability** - a game filled with bugs is not an enjoyable one.
- **Reliability** - The game must behave consistently, to stop players from feeling that the game has “artificial difficulty”.
- **Maintainability** - code must be maintained for all 3 assessments.
- **Extensibility** - code will be built up and improved upon in 3 distinct steps, therefore must be easily extensible.
- **Accessibility –** features such as colour blind mode or large font to allow for a wider audience to enjoy our games.
- **Open source** - the code will be shared between groups, thus being open source, though not necessarily free software, is absolutely necessary.
- **Time-to-market** - our team works on a tight schedule and must adhere to it.
- **Immersion** - features such as advanced graphics and/or sounds can increase player enjoyment.
- Update website

Appendix 2:
Software Approach Diagram



Appendix 3
Notes from meetings

Agile sprints for summer term

Tue/ Wk 1 to be done by Fri/wk1

- Tim: start implementing score tracking and multiplayer function.
- Radostin: Implement airport
- Katie: Requirements validation and regression test.
- Aisha : problems we have faced so far and game mechanics.
- Notes:
    - Control radials were not multiplayer friendly.
    - Multiplayers sharing keyboard

Fri/Wk1 to be done by Tue/Wk 2

- Katie: Initial regression testing and validation written up
- Writing manual requirements test in a ready to run format.
- Tim and Radostin: Complete a working multiplayer game.
- Chris: completed GUI document
- Aisha: complete game plot/ mechanics

Tue/ Wk2 to be done by Fri/wk2

- Katie and Chris : Acceptance testing
- Aisha: Complete design interaction part of report
- Tim: User Manual
- Radostin: fix bugs in game

Fri/wk2 to be done by Tue/wk3

- Katie: finish all of test report apart from unit and regression testing sections.
- Tim: User Manual
- Radostin: Unit tests
- Aisha: Finish remaining sections of the extension report

Figure 4 Evidence of rescheduling using Agile sprints implemented in summer term.

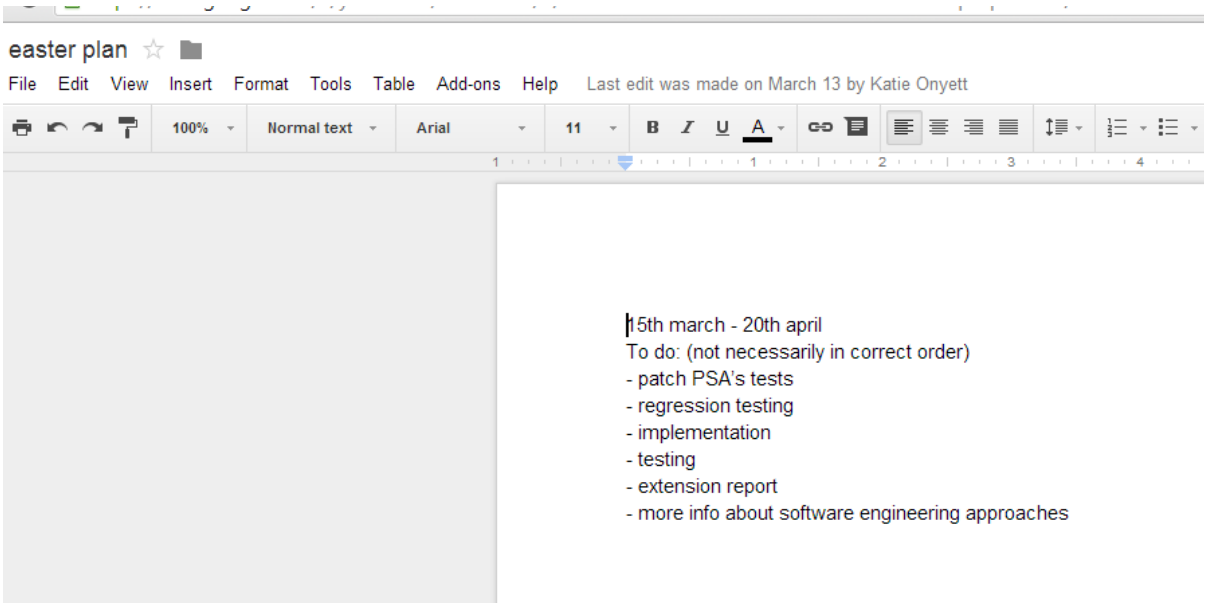


Figure 5Notes from meeting on the 13th of March to plan tasks to be completed during Easter.

ols Table Add-ons Help Last edit was made on March 13 by Katie Onyett

Further Requirements

FR1 - there must be at least two landing and takeoff options (such as airports) for the airspace
FR2 - there must be a game mode where at least two people can play at the same time
    2.1 - there must be a score for each player
    2.2 - there must be a full set* of separate keyboard controls for each player
    2.3 - there must be indication of which plane/s are controlled by which player
    2.4 - each plane must be assigned to a player
FR3 - there must be a facility in the game when at least two people are playing such that control of a flight can be handed over from one player to another

*A full set of controls allows the player to select a plane, land, take off and adjust the altitude, speed and heading of a flight.

ER1-5 all need to be fulfilled for FR1 to be fulfilled
ER6-7 are needed for FR2.1
unsure about ER8 and ER9
SR1-8 already fulfilled for basic gameplay
SR9 needed for FR1 (see ER6-7)
unsure about SR10-11
SR12-22 already fulfilled
unsure about SR23-25 and SR27
SR26 and SR28 already fulfilled

no specific requirement for losing the game if a crash or similar?
- allow crashing but cause large points dock as a result
- TCAS? (though large points dock anyway)

~~- one airport belonging to each player~~
~~or land @ one, take off from the other?~~

~~fuel as incentive to land?~~

- handover initiated by player
- not allowed to handover for some time afterwards
- unable to handover while separation rules are violated
- each player has separate handover timeout

Figure 6 Notes from meeting on the 13th of March – breaking down and sketching of official requirements.

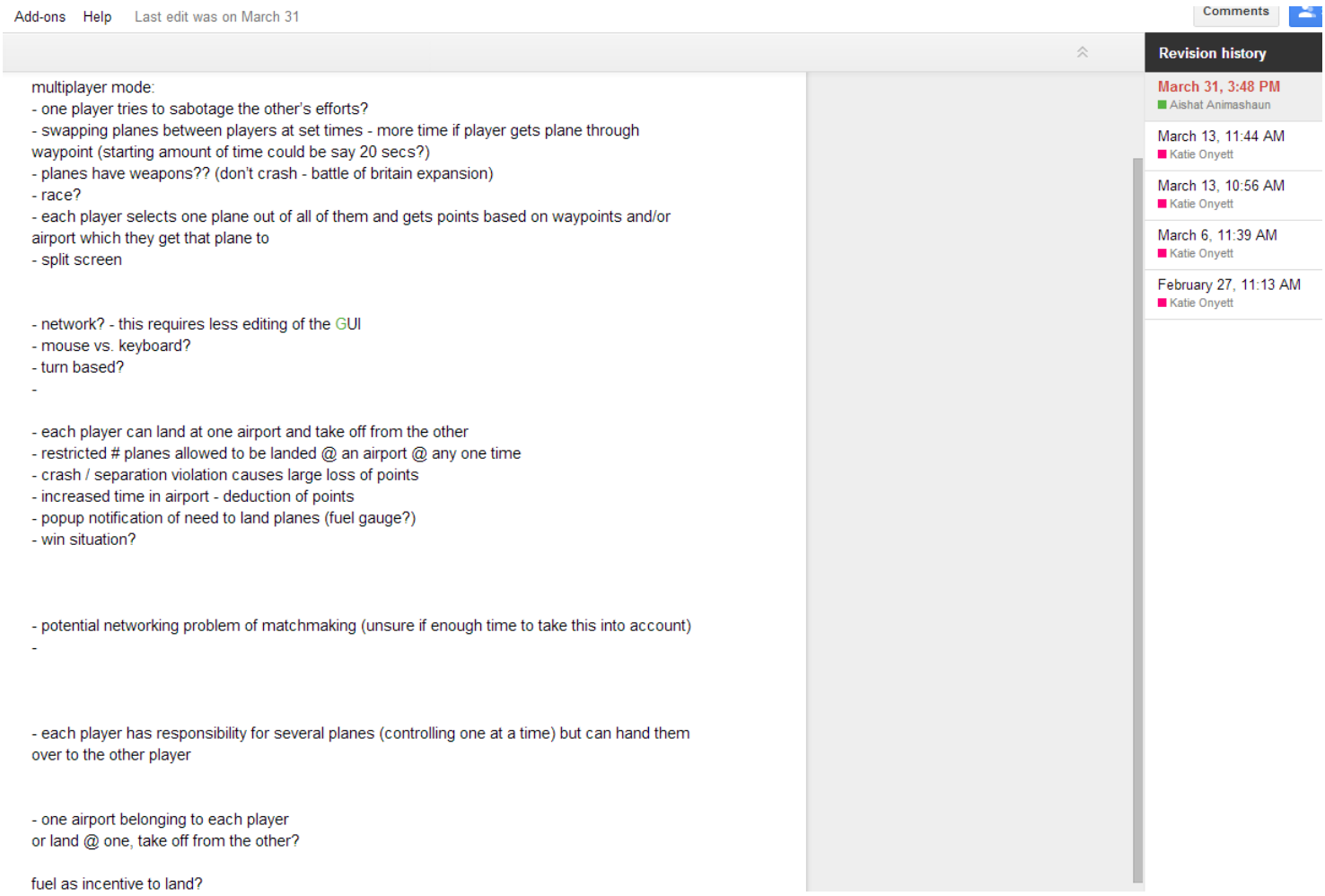


Figure 7 Evidence of brainstorming for implementation ideas, our ideas changed over time as seen in the revision history.

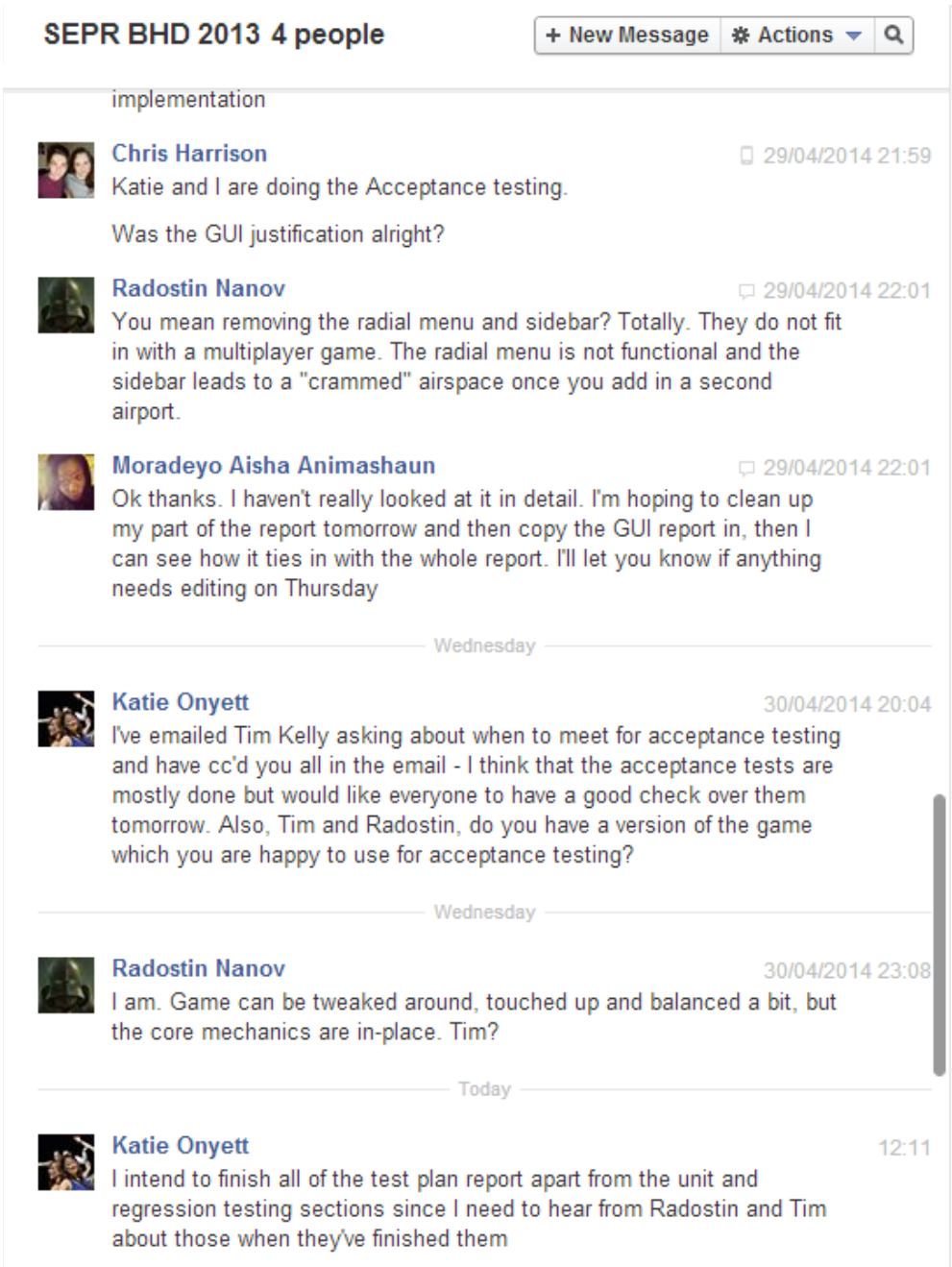
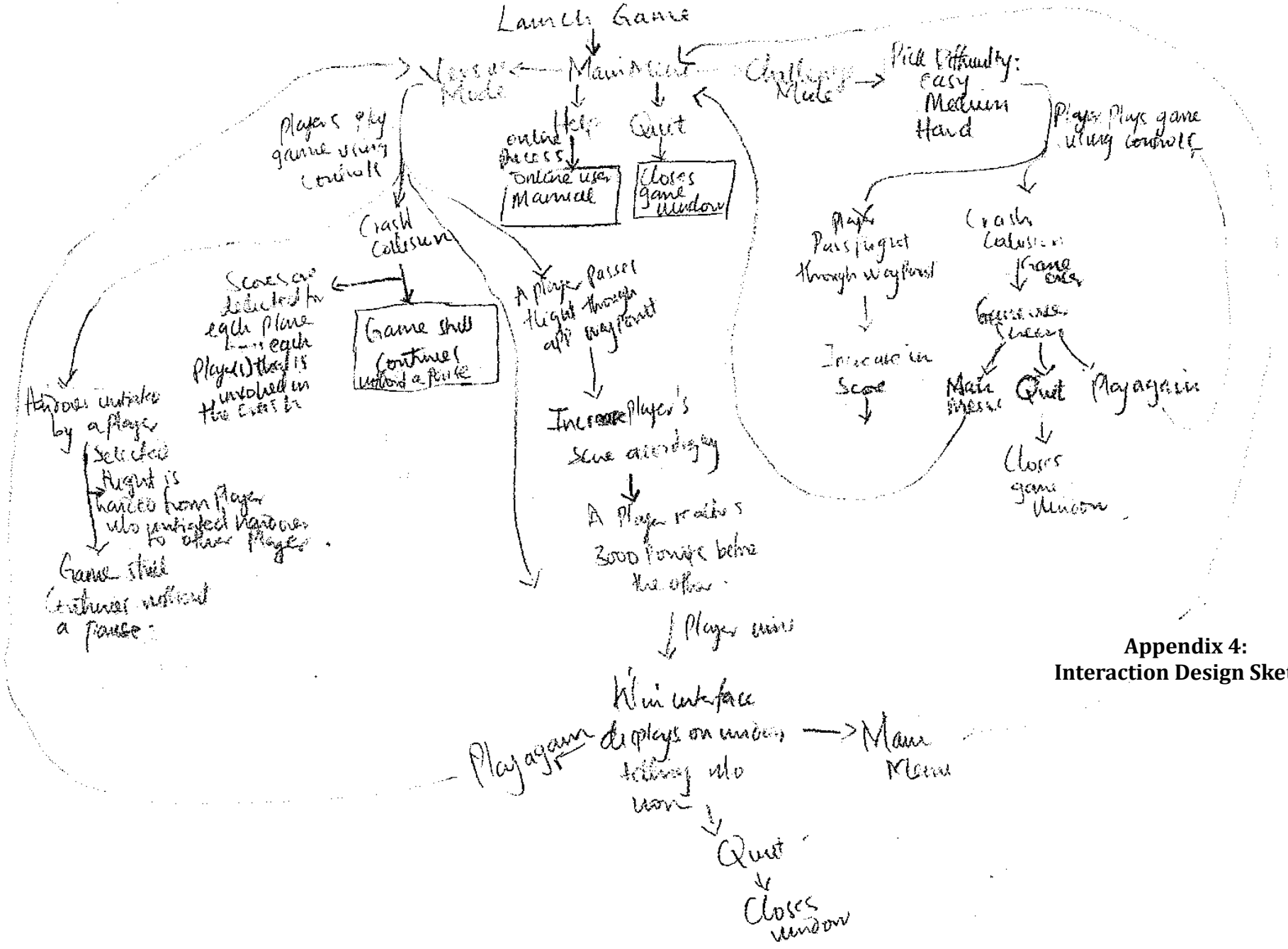
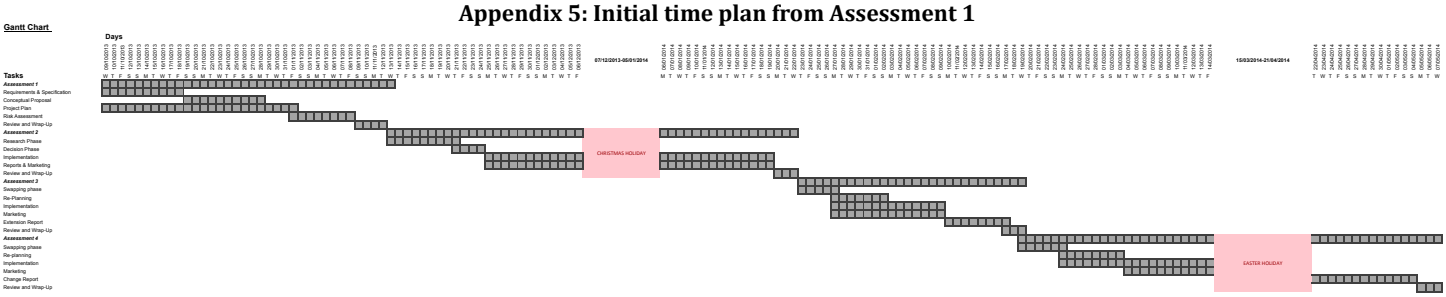


Figure 8 Evidence of communicating goals for each sprint via social media



Appendix 4:
Interaction Design Sketch

Appendix 5: Initial time plan from Assessment 1

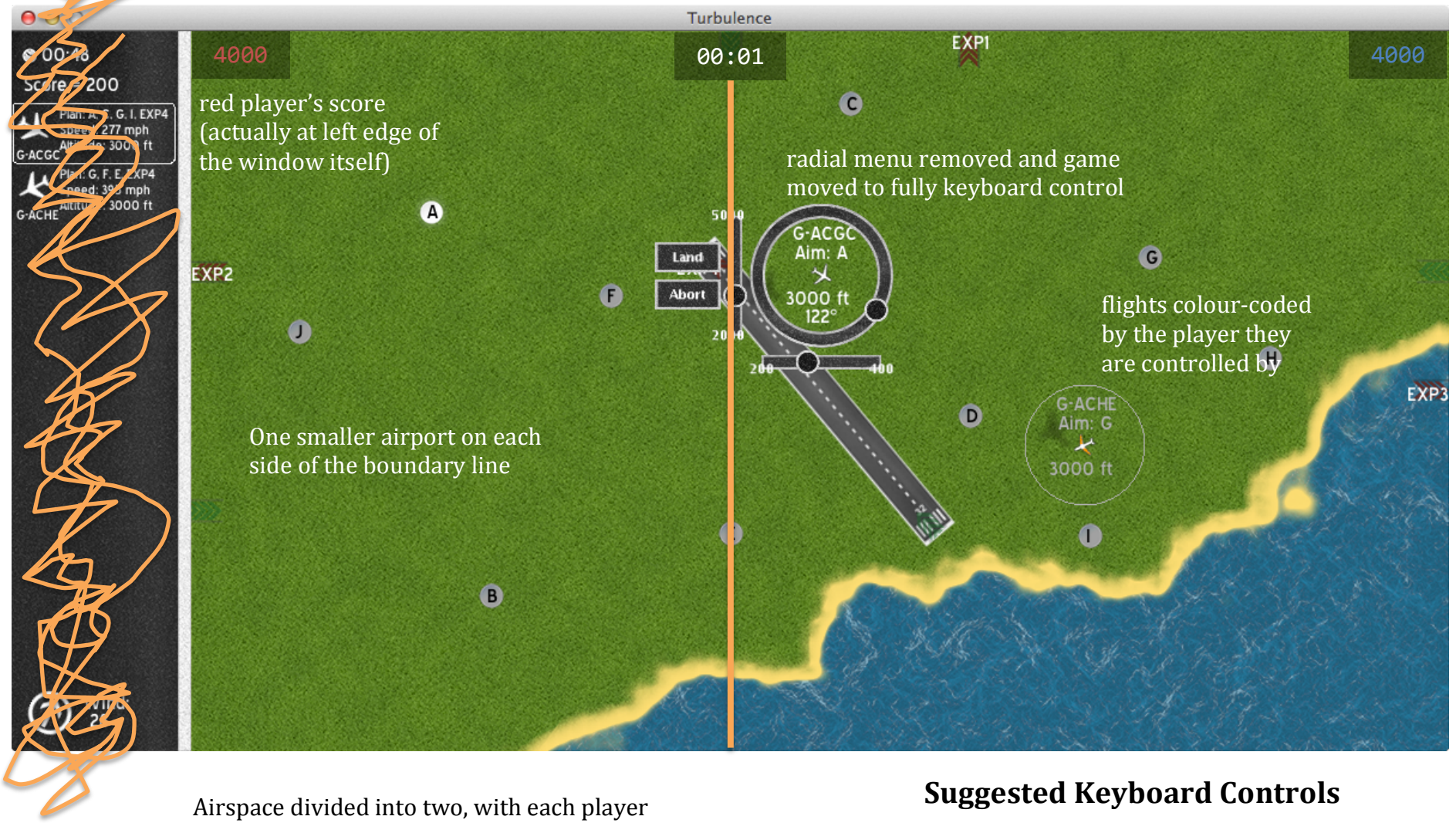


Single Computer Multi-Player Proposal

sidebar removed and airspace expanded to fill whole window

timer moved to the centre

blue player's score



Airspace divided into two, with each player responsible for half. Planes transfer between the two players automatically on crossing the boundary.

Possibly a slight red/blue colour filter on each side of the airspace to make the distinction even clearer.

Suggested Keyboard Controls

Turning and altitude	WASD	Cursor keys
Increase / decrease speed	Q & E	J & K
Cycle between flights	C & V	, & .
Land / Take-off	B	L

Appendix 7: Plan for Assessment 4 produced as part of Assessment 1

Assessment 4

This assessment is due at Noon on the 7th May. The swapping phase will take place over 2 weekdays and one weekend. The requirements change document will then be issued on the following Monday, giving 5 term-time weeks to complete the second phase of the assessment.

Swapping Phase

We can plan to follow almost the exact same procedure as in the Swapping Phase of the previous assessment, although we will be swapping the roles of the two sub-teams around.

Re-planning (*assigned to Aishat, Chris, Katie*)

- Receive a list of required changes from the customer
- Review these, clarifying any necessary points with the customer, and make any necessary alterations to our project plan / requirements / specification etc.

Implementation of the changes (*assigned to Radostin, Valentin, Tim*)

- While the other half of the team is in the re-planning phase, use that time to get familiar with the inherited codebase, any technologies used therein and so on.
- Once the plan has been drawn up, move on to implementing the tests required in order to verify the new / changed requirements.
- Then it should be safe to start actually implementing the relevant changes, following a somewhat similar chunking procedure as that defined above

Marketing (*assigned to Aishat, Chris, Katie*)

- Update the game manual in the light of both the moving to another team's product, and in the light of the change requirements for this assessment.
- Similar updates to the team website and the online documentation thereon.
- Start to update the test plan (not actually writing tests though)

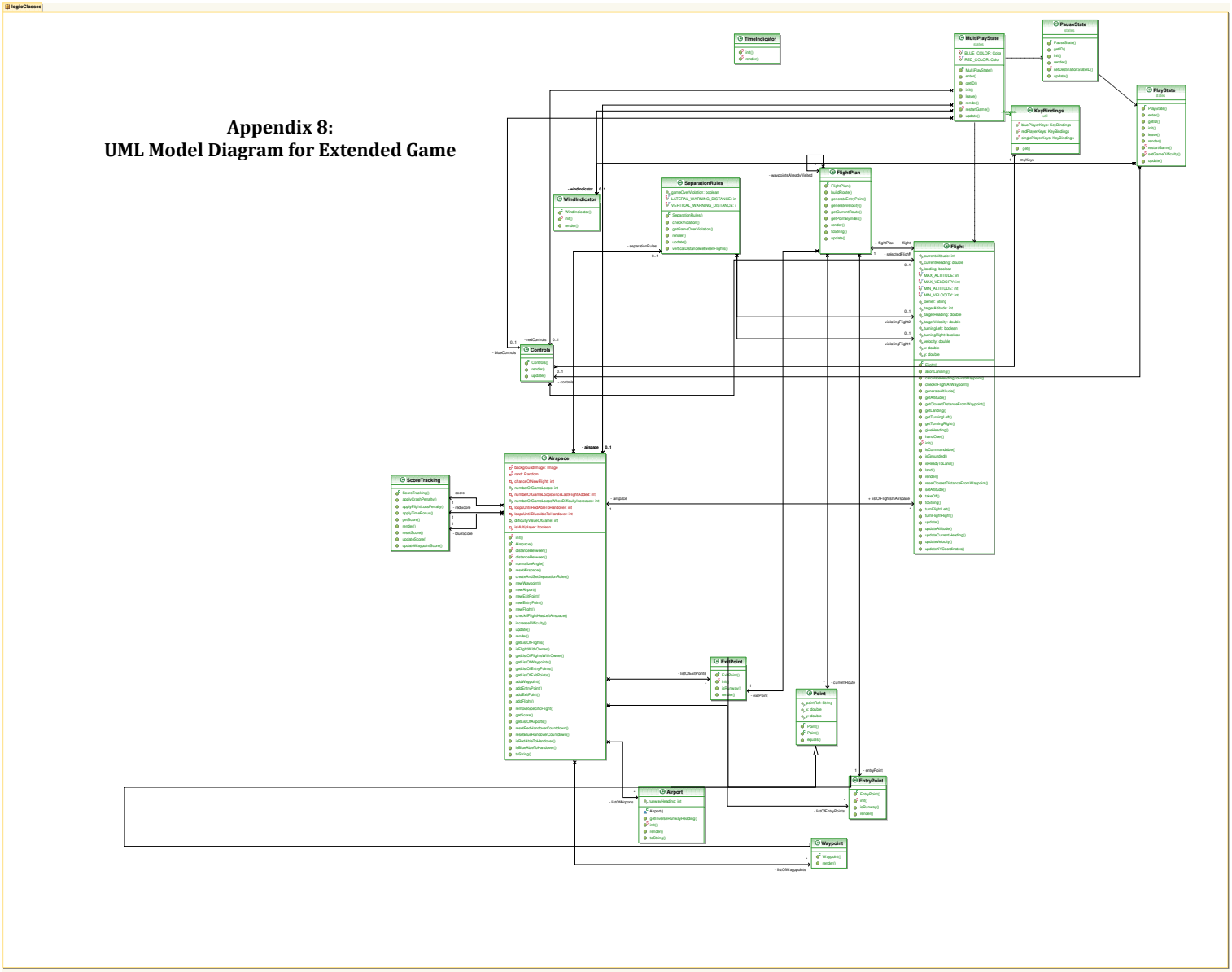
Change Report (*task leader to be decided)*

- Meeting for each of the members (of the implementation team in particular) to feed back information about the challenges encountered in implementing their part of the requirements changes, details of the affected elements and any architectural changes required by the implementation but not actually specified in the requirements.
- Decide the division of labour in actually writing the report.
- Task leader to compile the document together, ensuring consistent formatting and "voice".

Wrap-Up (*to be done as a whole group)*

- Follow the same testing, merging and website updating procedure as given in the first few bullet points under the Wrap-Up section of Assessment 2.
- Package all game code, and any required dependencies, into a single directory (Game4)
- Convert the final versions of the user manual, the test plan and the change report to PDF.
- Create a text file with the URL of the team website (url4.txt)
- Prepare self-assessment form, if necessary (SelfAss4.pdf)
- Zip into a single file (BHD.zip) and then submit.

Appendix 8:
UML Model Diagram for Extended Game



Appendix 9:

Assessment 4 Implementation Plan

Draft 2 – 30/Mar/14

Menu Screens

- Replace the single "Play" button with two new ones, for playing in single- and multi-player modes
- Remove the credits screen and replace with a link to the team website (as we did in the last assessment)
- Update the information on the controls screen, with details of the new keyboard control scheme
- Also consider whether to add a direct link to the user manual

States

- Fill out the `MultiPlayState` class
- Edit (and potentially rename) the `PlayState` class to be consistent with the new multi-player mode

Airspace

- Remove left-hand sidebar, and move the clock to the top-centre of the screen
- Change the background image to match our own design aesthetic
- Remove instantiation of `Controls` from the `Airspace` class and into the game states
- Edit both the `SeparationRules` and `MultiPlayState` classes to make collisions non-fatal in multiplayer games

Controls

- Remove the radial menu, which is the `FlightMenu` class
- Also remove all the mouse control functions in the `Controls` class
- New `util.KeyBindings` class, that will contain a `HashMap` linking game functions (e.g. "turn-left") to keyboard buttons (e.g. `Input.KEY_A`)
- Change `Controls` class to allow multiple instances, one for each player in multiplayer
- Each instance will take a `KeyBindings` object on creation to define the buttons it will handle
- Implement commands for cycling between planes using the keyboard
- Investigate how we can meet/relax the requirement for being able to turn planes to an arbitrary heading

Scores

- Add capability for multiple instances to the `ScoreTracking` class
- Potentially have the `ScoreTracking` class render itself
- Add point additions and deductions for multi-player events:
  - Deduct a large number of points from the controllers of both planes involved in a collision
  - Deduct a smaller amount of points for each separation violation
  - Add points for a successful, planned hand-over
  - Deduct points for an unplanned handover

Airports

- Ensure the `Airport` class can cope with multiple instances
- Add functionality for storing planes, and for automated handover of landing planes
- Airport will also need an owner field
- Actually add a second airport (to both gameplay modes)

Flights

- Add a variable and getter/setter for the flight's owner
- Change `drawFlight` function to colour-code by owner (in multiplayer) instead of by speed
- Potentially add fuel, or similar, functionality to force players to land planes

Appendix 10: Risk Assessment from Assessment 1

Risk assessment and mitigation

The risk assessments for this project were inspired by a series of lists of common software engineering risks found online (see the references list at the end of this document), which we compared to the project to see if they apply in our context. We also held meetings to go through each phase of the project in order to identify possible risks. In identifying the mitigations for each risk we have taken into consideration the fact that this is a university project. Where a higher authority is needed to resolve an issue we have decided to refer to the module leaders for mediation.

In the table below column **I** represents the impact of a particular risk and column **L** represents our assessment of its likelihood.

Risk	**I**	**L**	**Mitigation Plan**
1. Unreasonable delivery deadlines fixed in project plan:	H	H	We will use the final assessment submission dates and effort required to complete each tasks to recalibrate the delivery date for tasks in the project schedule whilst conferring with team member(s) availability. At present no SEPR work has been scheduled over the Christmas and Easter breaks, as a last resort these weeks could be used in order to catch-up with Assessments 2 and 4 respectively.
2. Inconsistency of the customer's needs relative to the product:	H	L	We will organize a formal requirement gathering session where the client and all team members must be involved, and ensure that all requirements recorded/rectified during the meeting are specific for traceability.
3. The customer shows a low understanding of what they require from the product:	H	M	We will organize formal gathering meetings with the client where a written understanding of his needs is presented; go through them explaining how it affects the final product whilst letting him make changes as well as keeping him updated and manage/inform his expectation throughout the project.
4. Unavailability of the customer for meetings:	M	M	We will organize to contact him on another platform such as Skype or send him emails concerning the planned questions or topic for any meeting he is unavailable to attend.
5. If there is no establishment of rapid communication links with customer and our team:	H	L	We will go to meet him at his office to fill the communication gap, talk to him after lectures or contact other module lecturers about his unavailability. If a team member is unavailable, we will try to get in contact with them by phone (we have exchanged mobile numbers already) and email.
6. Lack of knowledge amongst the team members of tools to be used during the product development:	H	M	Use resources (such as the programming language and graphical toolkit) that are already reasonably well known to the team as a whole. If needed, make use of resources such as online help, documentation, and the module demonstrators. Where possible, stick to the initial chosen resources to avoid the delays associated with re-tooling midway through the project.
7. Failure to perform and document procedural reviews regularly:	H	H	Team member(s) will be assigned to the role of quality assurance keeping/ testing which will involve analysing to validate and verify each task to ensure that it is of high quality satisfies the requirements and ensure all deliverables consistent and flow/match each other appropriately.
8. Unavailability of means to ensure that project conforms with software	H	M	Research software engineering standards and apply them in practice.

engineering standards:			
9. Unavailability of mechanism for adapting to requirement changes:	H	M	Use Agile requirements change management: Do just enough initial requirements envisioning to identify the project scope and develop a high-level schedule and project plan. During development we will storm in a just-in-time manner to explore each requirement in the necessary detail.
10. Unavailability of documentation from chosen team during integration and extension phase.	H	L	Conduct formal meeting with chosen team to provide us with documents and if no documents are available and we're unable to change the team we've picked, conduct a formal meeting to gather vital information needed for adequate extension and understanding of the chosen code.
11. Unavailability of the right combination of skills within the team:	H	M	Engage the affected team members in training/ research using available resources to educate or provide knowledge in lacking areas. Consider re-allocating tasks if that isn't feasible in the timeframe.
12. Unavailability of enough committed team members:	H	M	Encourage each team member to engage in decision-making tasks. Ensure that there are backup plans that take into consideration the possibility of delay in the delivery of a particular task due to unavailability of the relevant team member. Mission-critical tasks should not ideally be assigned to only one team member.
13. No specific methods used for analysis:	H	M	Specific analysis methods for software, architecture and requirements development will be researched and applied.
14. No tools used to support planning and tracking activities:	M	H	Frequent peer reviews. The code base is to be stored into a shared repository via Bitbucket, whose tracking tools we will use. Documentation will be in a cloud folder accessible to all team members.
15. No prototypes:	M	H	Early versions of the product, such as the assessment 2 deliverables, will serve as prototypes for later versions.
16. Unfamiliarity with the product to be built	H	L	Research subject area of product as well as important terminologies needed to ensure that the game qualifies as an air management traffic game.
17. Creation of an optimistic schedule that is "best case" rather than realistic	H	M	Prevent creating the project plan assuming that all parts will take the most optimistic path possible and instead leave space/time for unpredictable errors/mistakes that may delay the delivery of each path. Time given to each task should be realistic and the longest possible without the project suffering.
18. Exclusion of tasks from schedule:	H	L	Monitor the existing project closely and realistically. Breakdown requirements specification into smaller tasks and reschedule action plan when necessary.
19. Underestimation of product size in terms of lines of code and function points:	H	M	Carefully execute/ or briefly run through the design-process as a team again to prevent waste of programming hours. Increase number of team members coding and create stricter deadlines to ensure project is completed by final deadline.
20. Reduction of productivity due to	H	M	Analyse schedule and reschedule task delivery dates taking into consideration the time required by

schedule pressure:			the team member supervising the task to complete the task.
21. Cascading delays in interdependent tasks:	H	M	Reschedule immediately and stick to it to prevent loss of time to complete the project. Involve more people in current tasks causing delays.
22. Team members insist on decisions that lengthen the schedule	M	L	Discourage adding of unnecessary features that deviate from the initial project plan to prevent waste of programming hours.
23. Reduction of productivity due to poor team structure	H	M	Divide each phase of the project into various tasks where each team member is assigned responsibility for supervising the completion of a task(s).
24. Excessive time spent at meetings	M	L	Conduct regular meetings where the duration, the purpose of the meeting (agenda) and the expected outcome are stated and minutes are shared.
25. Inaccurate status reporting			Have a communication plan to ensure every team member is regularly informed. Have a general action plan available so each team member can tick off completed tasks so that everyone is informed of the status of the project.
26. Planning too poor to support the desired development speed:	H	L	Ensure specification is clear, complete and detailed enough to prevent conflict between requirements. Then break down tasks into smaller blocks for more accurate and detailed planning taking into consideration final deadlines.
27. Inadequate facilities:	H	L	Try to make do with the facilities available, or speak to the Department's technical support or the module leaders should any problems arise. We have evaluated alternatives to the 3rd party cloud systems that we are planning on using (see details in the relevant section of the Project Plan).
28. Development tools do not work as expected:	H	M	Ensure that the team as a whole is familiar with development tools chosen initially, use available resources for help or contact technical team or module lecturers for help.
29. Customer insists on new requirements:	M	H	Manage new requirements using the agile method and reschedule the Project Plan accordingly. As new requirements are expected in Assessment 4 we have built some leeway into that plan already.
30. Team does not solicit customer input	H	L	Ensure client is involved in the planning of the project and gathering of requirements. Keep client updated and in order to manage his expectations.
31. Customer will not participate in process reviews	H	L	Refer to initial documentation of requirements by client and try to interpret it to the best of the team's ability. If changes need to be made, use the agile method and reschedule project plan as appropriate.
32. Customer will not accept the software even though it meets all specifications	H	L	Check that all deliverables conform to the file format specified by the module leaders in the assessment document. Make sure that we aim to submit all the assessments in plenty of time, so as to allow for any last-minute failure of the Department's electronic submission systems.
33. Other teams are late to deliver; or	H	M	Ensure that all available code for the integration and extension phase is understood before selecting.

deliver components of low quality			We have built time for familiarisation with another team's codebase into the project timetable
34. Some sections of the project require more work than expected:	H	M	Re-evaluate project plan, analysing and breaking down each assessment phase or task into smaller task blocks and update the project plan immediately. There should be some slack in the plan already.
35. Development of extra software functions that are not required	M	L	Discourage adding of unnecessary features that deviate from the initial project plan to prevent waste of programming hours. Hold regular meetings to ensure everyone stays strictly in line with initial decisions.
36. Conflicts within the team	H	M	Some "storming" within the team is expected, and perhaps even necessary. We should that we listen to each others' perspectives, and should not be afraid to ask the module leaders to intercede.
37. Low motivation and morale reduce productivity:	H	M	Set a realistic schedule immediately and try to stick to it to prevent loss of significant time to complete the project. Hold meetings where everybody's progress is assessed and team member(s) having problems can be helped.
38. Loss of work:	H	M	Ensure that work is uploaded to group repositories to prevent loss. Should work go missing, team members will try to recover from personal directories (git makes this easy as each clone contains all repository history). We should reschedule as needed and consider informing the module leaders.
39. Inappropriate design:	H	M	Hold a meeting to re-evaluate design in line with requirements and make changes with the customer involved. Update project plan as appropriate.
40. Incorrect use of unfamiliar methodology:	H	M	Hold meeting to pick out reasons why the methodology was used in a wrong way, evaluate how it was used wrongly and use available resources to pick out mistakes and inform the team of the correct way to use the methodology.
41. Components developed separately cannot be integrated easily:	H	M	Ensure that meetings are held to conclude on how components should be developed and encourage members to adhere to decisions. Components should be committed to the repository regularly so that other members can review them, and should stick to pre-agreed interfaces where practical.
42. Half-hearted risk management fails to detect major risks	H	M	Ensure that possible risks are discussed regularly as well as their mitigation process in meetings. Identify, assess and prioritize each risk and ensure that they are managed closely.
43. Failure to consider product target market	H	M	Ask the customer about the target market of the product, ensure that research is done into the target market and ensure they are taken into account when gathering requirements. Review regularly to ensure that the product appeals to the market. The project plan should be updated as needed.
44. Insufficient documentation delivered to the customer:	H	M	Ensure that all deliverables listed by the customer are taken into consideration when planning the project and allocating tasks and that all tasks are accompanied by appropriate documentation. Attempt to play-test the game using a volunteer unfamiliar with our code-base.