Feed Jonathan Foundation
Game Project

Preliminary Design Report

Software Engineering Project

COMP 361

McGill University

October 23, 2015

Table of Contents

Project Overview

Team Information

Name	McGill ID	Team Duties
Alex Ilea	260629223	Database and network programming
Jonathan Lucuix-André	260632816	Backend programming
Karl Chiraz	260640768	Network programming
Simon Thompson	260619314	Backend programming
Stella Lee	260508902	Frontend programming and database

Client & Stakeholders Information

This project does not have a particular client. Instead, it targets an audience interested looking to immerse themselves in an expansive world and engage themselves in a simplistic yet intuitive game experience to pass their time and get away from the daily routine. Specifically, the project is targeting the desktop platform. As such, anyone with a personal computer and an internet connection can try the game.

Moreover, the main stakeholders in our product are its users. Our targeted audience is casual gamers who are not interested in spending the time to overcome a steep learning curve. Instead, we are targetting gamers looking for an intuitive experience that will require minimal effort in exchange for maximum pleasure. Thus, we are targeting people between the ages of 18 to 35. We believe that this approach gives us a broad and financially-independent audience, that has and is willing to spend money on entertainment for themselves and their kids.

Project Information

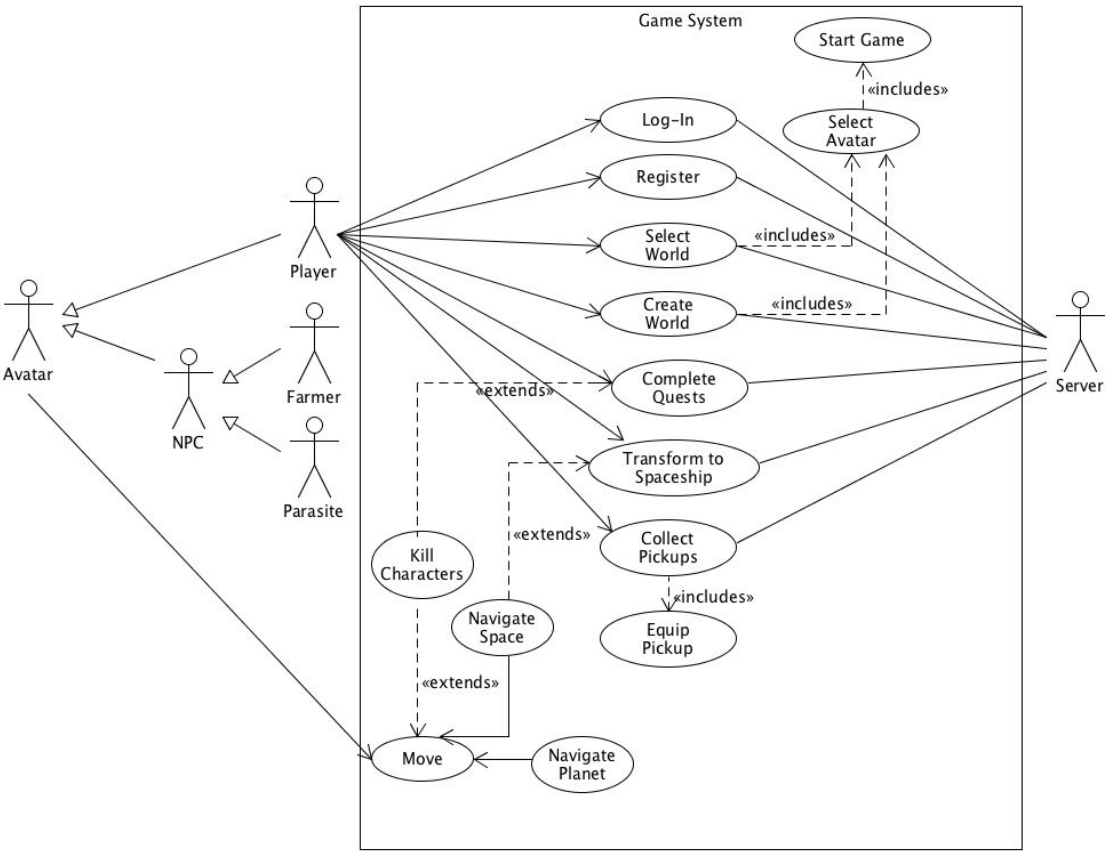
Feed Jonathan Foundation is a team of developers that will improve the world of casual gaming. The majority of popular games are either too simplistic to engage the customer on a long-term basis, or too complex and expensive to convince the masses to try it. We are dedicated to creating a simple, affordable, and immersive casual gaming experience that will engage our users for longer periods of time, ultimately resulting in maximized profits.

The online game *agar.io* is set as our baseline. The competitor offers a very simplistic 2D user interface, void of a storyline. The baseline game is extremely repetitive as it only has one primary objective. This inevitably leads to rapid loss of interest , and consequently a potential loss of profit. Our goal is to improve upon this experience and provide our users with more engaging storyline that will ultimately lead to a lower churn rate and increased profit through client retention. By using a minimalistic, yet captivating and immersive user interface, we could attract a multitude of users to try our product. Moreover, the integration of the engaging storyline will get our customers to choose our game over the competitor's and remain loyal. Finally, the online multiplayer capabilities would incentivize people to share the game with their friends and family and create an even greater overall engagement for our product.

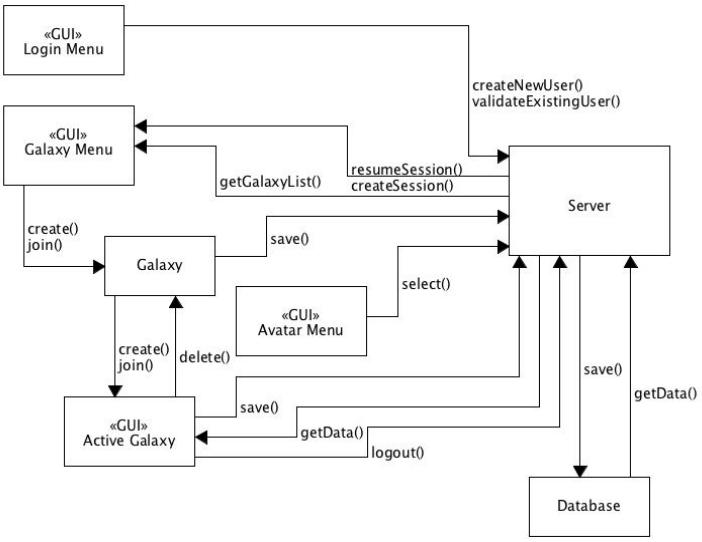
The game storyline revolves around an avatar floating around the game galaxy. The movement mechanics of the game are centered on momentum. When the player left clicks in a specific position on the map, his avatar gets propelled in the direction opposite to the click. When the avatar comes into contact with his enemies (farmers or parasites), he absorbs the mass of said enemies, thus becoming bigger. Whenever the user clicks the screen to thrust the avatar, the player loses a small amount of mass. Hence, the player's goal is to balance his use of propulsion with his current mass, making for an engaging and addictive experience. The multitude of different planet maps available to the player will compose the journey of his avatar, where he will encounter different quests and challenges throughout the game. The player will be able to save his progress at any time, and play with his friends over the Internet. The game itself will be developed in C++, using the Unreal Engine framework. On top of the option to downloading from our website, the game will also be offered through the STEAM network. This will make it accessible on all popular desktop platforms (Windows, Linux, OS X) for more than 10 million of users.

System Description

Use Case Diagram



Block Diagram



The block diagram presents the view of the game interface from the user perspective. To interact with other users, each instance will communicate via the server. However, the database is not visible to users, and the user will be able to communicate with the server by interacting with the menu options. The GUI will be composed of the initial login menu, where the user can choose to register or login with a previously created username and password. Once the user has successfully logged in, the user will be presented with a list of existing, active galaxies via the “Galaxy Menu”. The user has the option to create a new galaxy or join a pre-existing, active galaxy. The user will then be prompted to choose one of three avatar characters, and that avatar will be placed on a planet in the galaxy of his or her choosing. The main gameplay will occur in the “Active Galaxy” GUI. The user will have the option to save the game at any time and can return to the saved game upon his or her next login. Pressing the logout button or closing the game window will log the user out of the game session and remove the user’s avatar from the respective galaxy. The creator of a galaxy may delete said galaxy at any time.

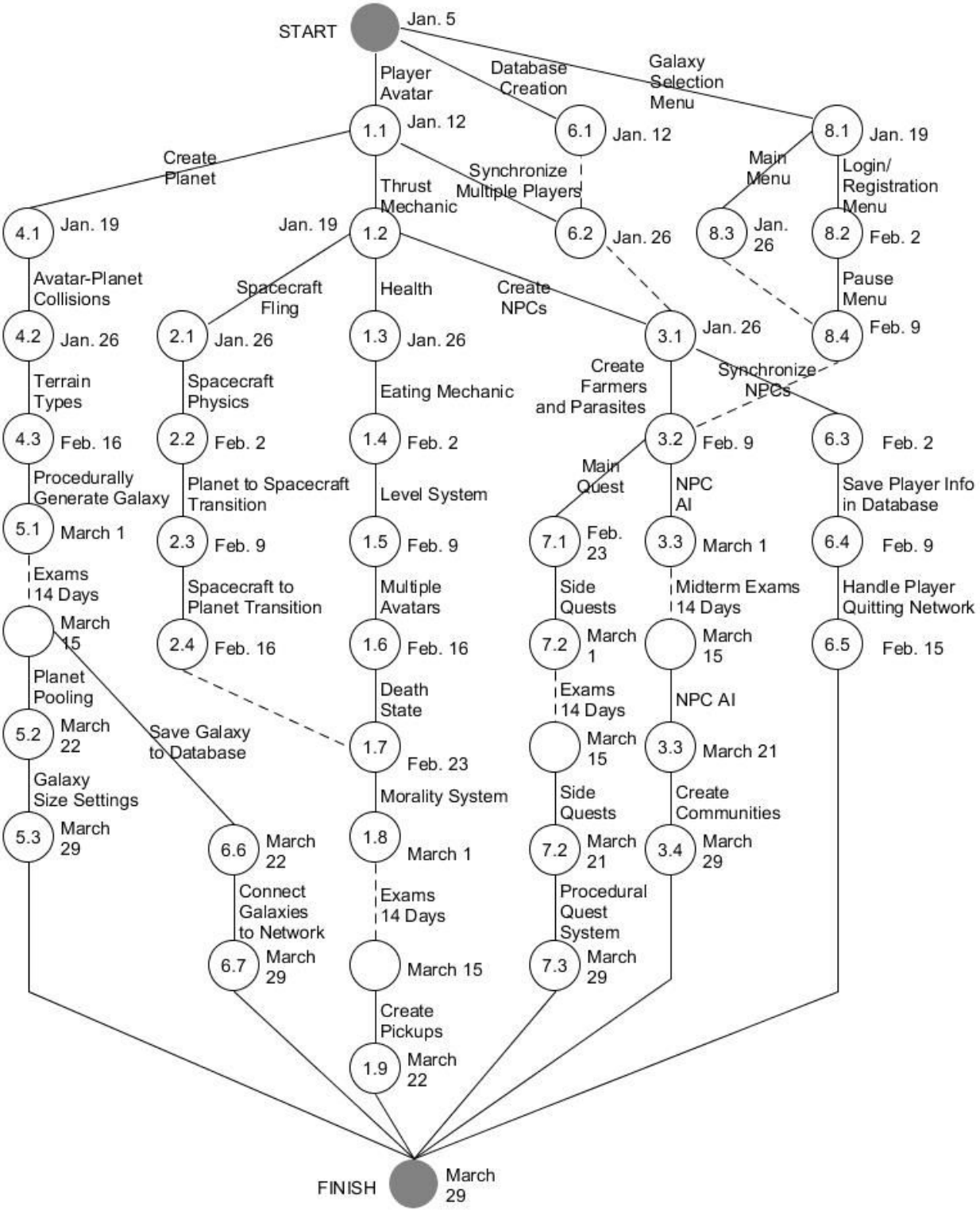
Technologies Used

Framework	Unreal Engine 4 (Game Engine)
Hardware	Platform: PC. User Input: Keyboard, Mouse
Languages	C++, SQL
RDBMS	MySQL
Libraries	iTween for Unreal Engine
Server	Apache HTTP Server

Formal Phase Project Breakdown

ID	Task	Duration (Sprints)
Phase 1: Create the player (10 sprints)		
1.1	Create a player avatar with a placeholder visual component	1
1.2	Develop the propulsion mechanics for regular planet traversal	1
1.3	Implement character health (loss or gain of mass)	1
1.4	Implement mass-eating mechanic for characters	1
1.5	Implement leveling system	2
1.6	Create multiple player graphics for avatar selection and level-ups	1
1.7	Implement player death state	1
1.8	Create morality system	1
1.9	Implement Pickups	1
Phase 2: Integrate the spaceship-controlling mechanic (4 sprints)		
2.1	Prototype spacecraft-flinging controls	1
2.2	Implement spacecraft physics and collisions	1
2.3	Allow the player to transform to a spacecraft at the edge of a planet	1
2.4	Transition from spacecraft to planet	1
Phase 3: Implement NPCs (8 sprints)		
3.1	Create generic NPCs	1
3.2	Create farmers and parasites	2
3.3	Implement NPC AI	4
3.4	Create communities	1
Phase 4: Build the planets (5 sprints)		
4.1	Create a planet which the restricts the player's movement	1
4.2	Add collisions (bouncing) when NPCs hit the outer edge of a planet	1
4.3	Create terrain types	3
Phase 5: Generate the galaxies (4 sprints)		
5.1	Procedurally generate a galaxy	2
5.2	Implement optimization procedures (planet pooling) for galaxy generation	1
5.3	Allow generation of small, medium or large galaxies	1
Phase 6: Setup network functionality (9 sprints)		
6.1	Create database	1
6.2	Set up a server to connect multiple players on a single planet	2
6.3	Synchronize NPCs on a server	1
6.4	Store player information in database	1
6.5	Handle the case of players quitting a game	1
6.6	Save galaxy state to a database	1
6.7	Allow players to connect to all galaxies on the network	2
Phase 7: Generate the quests (5 sprints)		
7.1	Implement main quest	2
7.2	Create side-quests	2
7.3	Implement procedural quest system	1
Phase 8: Build the graphical user-interface (GUI) (6 sprints)		
8.1	Create galaxy-selection menu	2
8.2	Implement login/registration screen	2
8.3	Create main menu	1
8.4	Create pause menu	1

Dated Activity Graph



COCOMO

We want to estimate the effort and the number of months required to complete the project using the COCOMO 1 method. We consider our project to be a semi-detached project because we are creating a traditional product with a tight, 2-and-a-half-month time constraint, and a game engine and programming language unfamiliar to most team members.

Hence we choose values of $a = 3.0$, $b = 1.12$, $c = 2.5$, $d = 0.35$

To estimate our effort we use the following formula:

$$Effort = a \times DLOC^{b} \ (DLOC = delivered\ lines\ of\ code)$$

We estimate our project to have approximately 20000 lines of code based on an educated guess from our prior experience creating games of similar size. Thus, we have that

$$Effort = 3.0 \times 20^{1.12} = 85.96$$

To calculate amount of months needed to complete the project, we use the following formula:

$$Months = c \times Effort^{d}$$

Hence, we obtain: $Months = 2.5 \times 85.96^{0.35} = 11.88\ months \approx 12\ months$

It will take 12 months for a single person working 152 hours/month to complete this project. Since our team has 5 team members, each member will require to work

$$11.88 / 5 = 2.38 \approx 2.5\ months.$$

Finally, the support effort for software maintenance is:

$$EffortMaint = ACT \times Effort = 1000 \times 85.96 = 85,960$$

ACT = 1000 lines of code changed per year

Estimated Effort

Phase	Effort	Months	Schedule	People
Requirements	30%	3.6	0.75	4.8
Design / Code	40%	4.8	1	4.8
Integrate / Test	30%	3.6	0.75	4.8
Total	100%	12	2.5	-

**People = Month / (Effort * Total time to complete the project)*

Lorenz / Kidd OO Effort Estimation

We use Lorenz/Kidd method to estimate our effort based on the following variables:

$a = estimated\ number\ of\ classes = 60$
$b = Graphical\ GUI = 2.5$
$c = number\ of\ person - days$

In order to compute the effort we used the following formula: $Effort = (a + (a * b)) * c$
Hence, we obtain: $Effort = (60 + (60 * 2.5)) * 7 =$ 1,470

Statement of Project's Estimated Length

From the dated activity diagram, we know that the longest path from the root to the end lasts a total of 85 days. Therefore, we estimate that the project will require 85 days of development time, including holidays and exam times. The critical paths are each of the four branches which end on March 29. In fact, a delay in any of these branches would delay the delivery time of the project. Further, it is important to note that buffers and padding were implicitly included in the graph. In fact, certain branches merge to the "FINISH" node earlier than others. Further, near the end of the project, there are fewer than 5 nodes working in parallel. Thus, at these points, we acknowledge extra time which may be needed for unexpected delays or emergencies.

Milestones and Deliverables

Milestones:

Event ID	Title	Result	Date	Dependencies
M1	Player design	Design and implementation of basic player avatar	12/01/16	
M2	Database design	Design and implementation of database	12/01/16	
M3	Galaxy selection menu design	Design and implementation of Galaxy selection menu GUI	12/01/16	
M4	Design of thrust movement mechanics	Implementation of basic movement mechanics for characters (thrusting)	19/01/16	M1
M5	Network design	Design and implementation of real time multi-player gameplay	26/01/16	M1
M6.1	NPC design	Creation and design of NPCs	26/01/16	M1, M4
M6.2	Design of farmers and parasites	Design and implementation of farmers and parasites	09/02/16	M1, M4, M6.1
M7	GUI design	Creation of Main, Login, Registration, and Pause Menu GUI	09/02/16	M3
M8	Player feature and spacecraft design	Design and implementation of multiple player avatars, level-up system and spaceship traversal	23/02/16	M1, M4
M9	Planet and galaxy design	Design and implementation of planets and galaxies	15/03/16	M1
M10	Pickup design	Pickups designed and implemented	22/03/16	M1, M4
M11	Quest design	Design and implementation of main quest and side quests	29/03/16	M1, M4, M6.1, M6.2
M12	Program Setup	Final product installed, tested, demonstrated	29/03/16	M1, M2, M3, M4, M5, M6, M6.1, M7, M8, M9, M10, M11

* Integration and Testing will occur at each step due to our choice of the scrum software process model

** Installation of final product does not require extra development time after the final steps, since integration and testing is implicitly integrated in each step of the formal phased breakdown

Deliverables:

Event ID	Title	Result	Date	Dependencies
D1	GUI design	Completion of GUI design and implementation	09/02/16	M3
D2	Player design	Player movement, eating, level system and spaceship control is shown to audience	23/02/16	
D3	Network and database design	Design and implementation of real time network connection, communication and persistence through a database	29/03/16	M1, M4
D4	Final Product	Final product installed and demonstrated	29/04/16	M1, M2, M3, M4, M5, M6.1. M6.2, M7, M8, M9, M10, M11

Resource Requirements Table

Item ID	Description	Unit Cost	Quantity (Units)	Subtotal	Event ID	Dependencies (IDs)
R-1	Unreal Engine 4, the game engine used for development	$0	5	0	M3	1.0 - 8.0
R-2	Servers to retain game state, player statistics and created galaxies	$3600	1	$3600 per year	M8	6.0
R-3	Subcontracting graphic designer	$1000	1	$1000	M3	4.0, 5.0, 7.0, 8.0
R-4	Subcontracting audio designer	$1000	1	$1000	M3	4.0, 5.0, 7.0, 8.0
R-5	Subcontracting plot writer	$500	1	$500	M9	7.0
R-6	App publishing (various distribution platforms)	$250	1	$250	M12	1.0 - 8.0
R-7	Rent expenses	$2500	1	$2500	M1	None
Total				$8850		

Cost Estimate

Programmer	Hourly Rate	Total Hours	Subtotal	Dependencies (ID)
Alex Ilea	$25	500	$12,500	1.0 - 8.0
Jonathan Lucuix-André	$25	500	$12,500	1.0 - 8.0
Karl Chiraz	$25	500	$12,500	1.0 - 8.0
Simon Thompson	$25	500	$12,500	1.0 - 8.0
Stella Lee	$25	500	$12,500	1.0 - 8.0
Total			**$62,500**	

*Our team uses iterative and incremental development process, hence everyone is working on all phases of the project.

Grand Total

Total Resources	$8,850
Total Cost	$62,500
Grand Total	$71,350

Engineering Methods & Team Work Environment

The software process models we have selected are the iterative and incremental models (business modeling, requirements, analysis & design, implementation, test, and deployment). For the first semester, the main method will be the incremental method, as we will be tackling each part of the business model, requirement, and design stages incrementally, with each team member being delegated a task to work on until completion. Once the implementation starts at the beginning of the second semester, we'll add an iterative approach to our overall management model. As each part of the implementation is completed incrementally, the team as a whole will be iteratively going over the requirements and design, changing them as necessary, as well as testing the implementation as it is completed. This method gives us a great deal of flexibility to change the original specifications of the requirements and design stages as we gain insight during development, while placing a large focus on learning, which will allow us to improve our organization and our code with each iteration.

The programming development method will be iterative and incremental. The iterative development, comprised of the phases detailed in the formal phase breakdown, is sliced into incremental portions, with each team member tackling a specific portion. Basically, each person is assigned a task, which they will work on until completion. Looking upon the team as a whole, multiple portions of the project advance simultaneously in an iterative fashion. We chose this method because the incremental method allows each team member to become masters of their specific portion of the project, while the overall iterative approach allows each member to familiarize themselves with the rest of the codebase and allows the team to find flaws with the design early on and go back to fix them.

The team organization system we are using is scrum. The major reasons for selecting scrum are the team's adaptability to changes in the requirements and design, the high emphasis on communication, which allows problems to be detected early on, and increased productivity. These benefits are the ones we have identified as most critical for a relatively inexperienced team of student programmers. A notable difference is that for phase 6 of the formal phase breakdown, setup network functionality, a form of extreme programming will be used, with two of our team members working as a pair to complete the phase. Extreme programming will be useful in this phase as it will allow the two team members to produce

quality code in an efficient manner, which is crucial for implementing the multiplayer aspect of the game.

The quality control method we will be using is the peer review rules. By taking the roles of producer, moderator, reader, scribe and inspector(s), we will ensure each teammate's code is of sufficient quality, while simultaneously updating ourselves on their progress and difficulties. For testing, unit tests will be performed at the end of each sprint and continuous integration tests will be performed on a test branch of our repository in order to test our code as part of the greater project. Automated unit tests (by using unit test frameworks like phpunit) will be used to shore up any weaknesses in the manual tests, as well as help take some strain off the programmers by reducing the bloating of manual test classes and making the unit tests more efficient.

The revision control methods we are implementing are a code repository, namely GitHub, and the scrum methods of handling user requirement changes. GitHub will be used in the scrum style, with every team member being able to push to the repository and group meetings being held to discuss push and merge decisions. As for user requirement changes, a scrum team expects the requirements to change, which changes the product backlog accordingly, forcing the team to adapt to new situations and demands as they arise.

Risk Identification and Impact Estimation

ID	Title	Description	Probability	Exposure
Risk 1	Merging all units	When merging all the different parts of each team member's code, we might run into certain problems of compatibility or unforeseen issues that may not be resolvable.	40%	Cost is 100h 100h*40% 40h
Risk 2	Networking issues	Most of our team members have not had the opportunity to work on a network system for games. Implementing the game program on a server may lead to complications.	40%	Cost is 50h 50h*40% 20h
Risk 3	Using Unreal Engine	We will need to learn all of the basics of the game engine as no team member has prior experience using it. This may delay the project, especially in early stages. It may make the creation of our project more difficult and lengthen the time needed for each deliverable.	80%	Cost is 200h 200h*90% 180h
Risk 4	Incompatible software for client	The client could try running our game on an out of date system and the game may fail to run. This includes the hardware or the software that is not up to date.	40%	Risk Tolerated
Risk 5	Out-of-date game	In the case that the client tries running the game without all the patches that have been added, the client may experience bugs.	60%	Negligible risk

Risk Management

- Risk 1 involves the merging the different parts of our project and we can avoid any problems by implementing automated unit tests to perform rigorous test cases on each individual part, we can avoid having too many issues within the same blocks of code. Integration tests will ensure that any addition to the code will not create side effects in the rest of the program. Also, having a set of programming rules will allow for a faster and more efficient way of either solving an issue or avoiding it.
- Risk 2 involves anything involving the networking part of our project and we can avoid complications by studying the concept of sending information to a servers and

reading information from them, we can make the initial programming of the multiplayer system faster and more efficient. We should also do test with a server installed on the same computer as the game in order to test any features associated with the multiplayer before putting it on our real server. We will also have to test the possibility of the server crashing and make our game signal to the user of this issue.

- Risk 3 can be avoided by using an engine that the team is familiar with, but we will assume the risk. Most team members have already created games using other game engines, and therefore this risk may not affect them as much. To prevent this risk from having a big effect on our project, we can practice with this engine by creating several little programs to familiarize ourselves with the functionality and workflow of the engine before the initial project starts.
- Risk 4 can be avoided by having a system that tests the computer before starting the game and prevents the game from running if the requirements of the system are not met.
- Risk 5 can be avoided by having the game check for updates before it starts up.

Fault Tree for Risk 3 (Using Unreal Engine)