Preliminary Design Document

Benson Perry Matt Dannenberg

Brian Shaginaw

COMP 361 Software Engineering Project
School of Computer Science
McGill University
Montreal, Canada

Friday October 7th, 2011

Table of Contents:

Project Overview

1.1 Team Members

Name	Mcgill ID	Team Duties
Benson Perry	260361134	Team Leader
Brian Shaginaw	260368016	Graphics
Matt Dannenberg	260357809	Networking

1.2 Client Information

In-house Game

Contact Person: Joseph Vybihal

Address: McConnell 3rd Floor

Email: jvybihal@cs.mcgill.ca

Target Market: We are trying to make a game that will be accessible to casual gamers, but with complexity that the hardcore gaming scene can appreciate. We hope to appeal to fans of tower defense games such as Plants vs. Zombies.

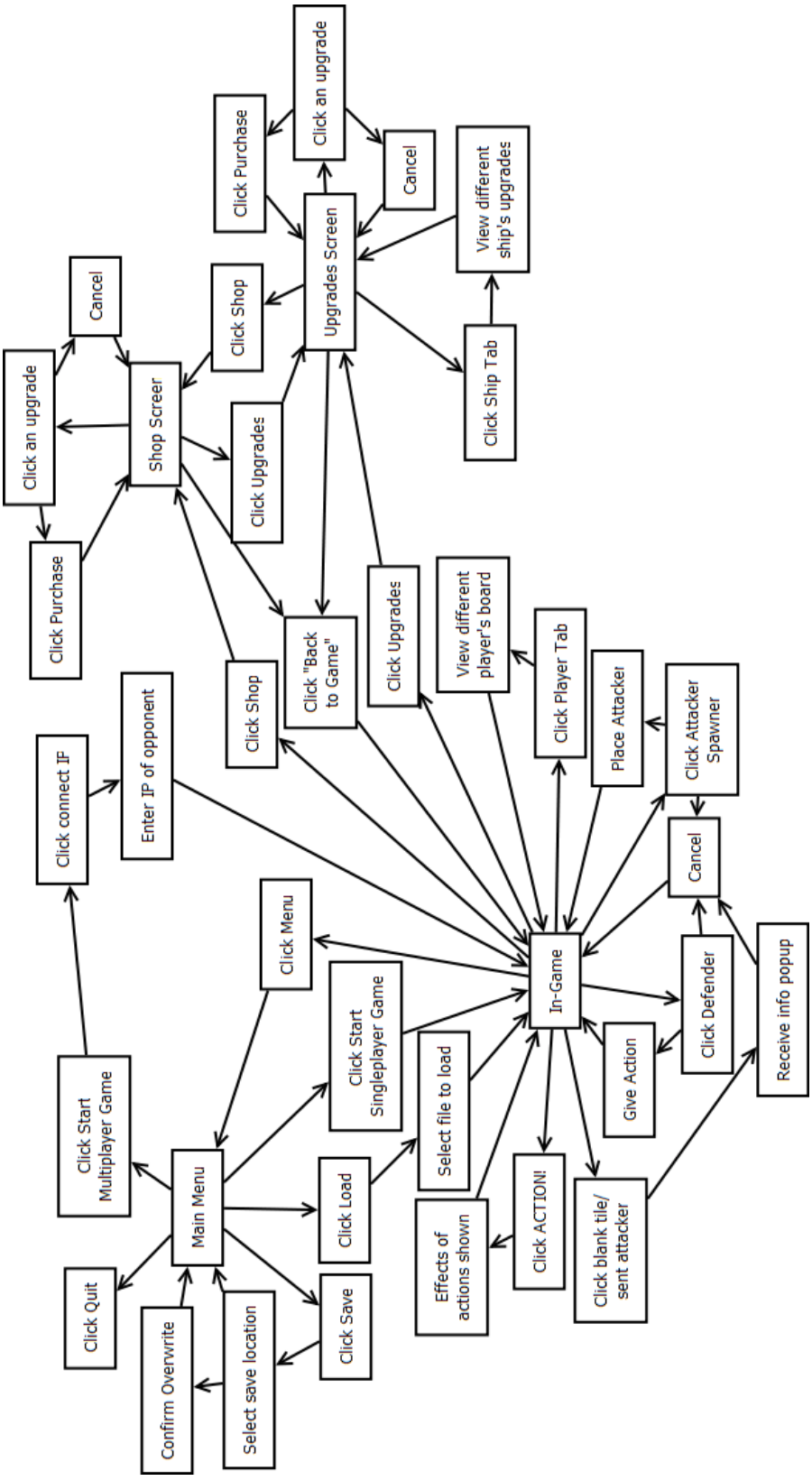
1.3 Project Overview

Our in-house game will be a blend of turn-based strategy game and side-scrolling shooter. It will have numerous features, including networked multiplayer, AI opponents, and save/load functionality. Each player will have a group of defensive watercraft (submarines and boats) that is constantly moving forward through an ocean. The ocean will have randomly generated terrain containing hazards which the player must avoid and gold for the player to collect. This terrain will be unique for each player. Each player will have a limited number of moves to make each turn - the player will dole out these limited moves to his different units, which will require planning ahead to avoid obstacles. Players gain money and experience by destroying obstacles and collecting items (treasure chests, floating coins, etc). This money is then used to buy units to place in the enemy's ocean. Any player can purchase squids, floating mines, angry fish, and other obstacles to send at the other players in an attempt to harm their opponents' units. A player wins when all opposing players' units have been destroyed.

There are two resources the player must manage: money and experience. Money is gained by destroying obstacles and collected from various sources on the map. Money is spent on offensive units in an attempt to destroy the opposing player's ships. Experience is gained by destroying offensive units and passively by simply staying alive. If Player 1 destroys a torpedo Player 2 has launched at his ships, Player 1 gains experience and money. Experience is used on upgrades for each individual ship. This allows players to upgrade one ship heavily at the expense of having their two other ships be weaker, or to spread the upgrades evenly, allowing for a more balanced fleet.

This game will feature networked multiplayer, saving and loading of game states, and a single-player mode for playing against a computer-controlled player. It will be written in the Python programming language using the PyGame library, a set of Python modules designed for game development (http://www.pygame.org).

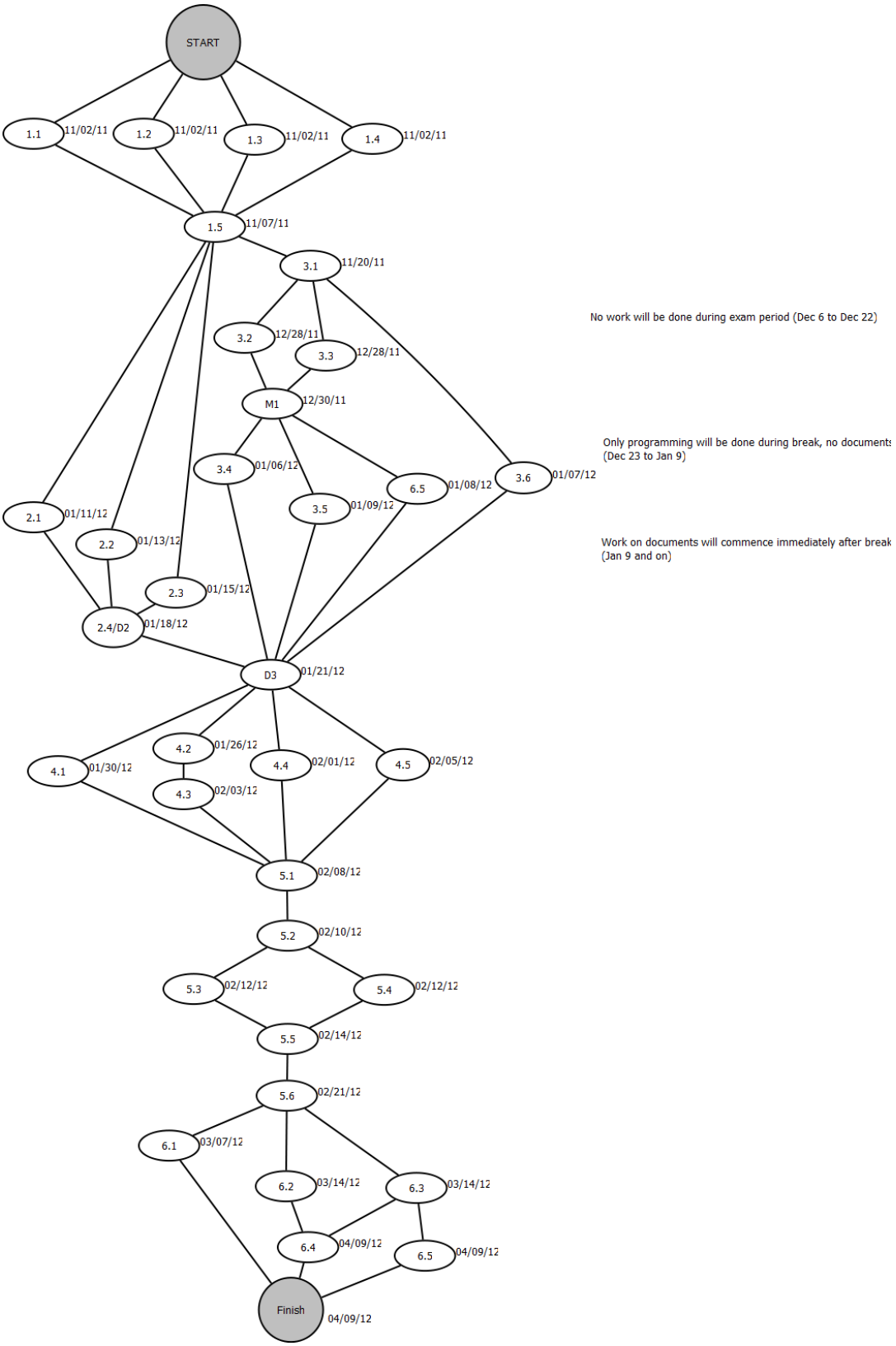
2 System Diagram



3 Formal Phase Breakdown

Phase Breakdown	
Step	**Time Estimation (days)**
Phase 1: Requirement Document	
Step 1.1: Draft intro	1
Step 1.2: Draft description	1
Step 1.3: Draft requirements	2
Step 1.4: Draft diagrams	3
Step 1.5: Revise and repair doc	3
Phase 2: Design Document	
Step 2.1: Draft intro	1
Step 2.2: Draft architecture overview	3
Step 2.3: Draft design (diagram heavy)	5
Step 2.4: Revise heavily	3
Phase 3: Basic Game Functionality	
Step 3.1: Game grid/board	1
Step 3.2: Moveable defensive units	1
Step 3.3: Sending offensive units	1
Step 3.4: Turn system for moving and sending	3
Step 3.5: Unit collisions/interactions	2
Step 3.6 Gaining/spending resources	1
Phase 4: Multiplayer Functionality	
Step 4.1: Single machine version of Multiplayer	2
Step 4.2: Win/loss conditions	1
Step 4.3: Game termination	1
Step 4.4: Basic menu	2
Step 4.5: Basic AI	6
Phase 5: Networking	
Step 5.1: Set up host game	2
Step 5.2: Have players connect to host	1
Step 5.3: Sending moves to host	2
Step 5.4: Host sending moves to other players	1
Step 5.5: Displaying results of move in GUI	3
Step 5.6: Game termination/cleanup	1
Phase 6: Upgrades, environment, and all the things to make the game fun	
Step 6.1: Saving/Loading (add to main menu)	1
Step 6.2: Map generation	5
Step 6.3: Upgrades - unit diversity	8
Step 6.4: Balance with new units/upgrades	2
Step 6.5: Improved graphics	10

4 Dated Activity Graph



5 Milestones and Deliverables

5.1 Milestones

Event ID	Title	Result	Date	Dependencies
M1	Board with automatic (offensive) and manual (defensive) movement	Grid system implemented with path and user directed movements	12/30/11	
M2	Resource management	Experience and gold gaining and management added	01/07/12	M1
M3	Collision detection	Turns system implemented and units collide and die as expected when turns play out	01/09/12	M2
M4	Single machine multiplayer	Two players on one machine	01/30/12	M1, M2, M3
M5	AI	Bot controlled opponent finished	02/05/12	M1, M2, M3
M6	Networking	Connect multiple machines and send info back and forth	02/14/12	
M7	Networked multiplayer	Multiplayer game complete	02/21/12	M1, M2, M3, M5, M6
M8	Saving/Loading	Game can save and load states	03/07/12	M1, M2, M3, M7, M8
M9	Map generation	Make the map dynamic and varied	03/14/12	M1, M2
M10	Upgrades	Upgrade trees and upgrade abilities implemented	03/14/12	M1, M2, M3
M11	Graphics	All map effects, offensive and defensive units, and attacks will be animated	04/09/12	Can be done to each piece as completed

5.2 Deliverables

Event ID	Title	Result	Date	Dependencies
D1	Requirement document	Client shown requirement document	11/07/11	
D2	Design document	Client shown design document	01/14/12	
D3	Board	Client is show board layout with moveable offense and defense and collisions	01/21/12	M1, M2, M3
D4	Multiplayer	Client is show full functioning multiplayer game	02/21/12	M1, M2, M3, M4, M5, M6
D5	Game complete without graphics	Client is shown full game with only animations missing	03/14/12	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10
D6	Game done	Everything is finished completely	04/09/12	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11

6 Resource Requirements and Cost Estimate Tables

6.1 Resource Requirements Table

Item ID	Description	Unit Cost	Quantity	Subtotal
1	Sprite	$10.00	100	$1,000.00
2	PyGame	Free	1	$0.00
3	Laptops	$1,000	3	$3,000.00

6.2 Cost Estimate Table

Programmer	Hourly Rate	Total Hours	Subtotal
Matt Dannenberg	$30.00	316	$9,480.00
Brian Shaginaw	$30.00	316	$9,480.00
Benson Perry	$30.00	316	$9,480.00

6.3 Cost Estimate Explanation

Laptops must be upgraded every two years, but must be purchased immediately at $1,000 per laptop for all three team members. Sprites must also be purchased from a graphic artist for all units and items in the game. To use but not retain rights to a small, custom sprite would cost approximately $10.00 per sprite. The hourly rate of $30.00 for each programmer is a market average for medium-level programmers and lead programmers on an independent game. We figured that the 79 work days we counted will not be full 8 hours day and probably be closer to 5 hours. The total cost for the project will be: $32,440.

7 Engineering Methods

7.1 Software Process Model

Waterfall - we have already started using this model. We did our initial system planning by laying out what we wanted to create - a turn based strategy game with upgrades and resource management. We then (in this document and the next) will write up our requirement definitions. Following this, we will begin design according to our requirements, then development. Our waterfall method will be "Modified" in that our "Integration & Test" phase will allow us to backtrack one or two steps to design and development as we continue to add more features due to our incremental programming development method.

7.2 Programming Development Method Selection

Incremental - we will first develop a single-player prototype where there is a simple board and movable pieces. Then we will expand to a second board, with possibility of sending units at the other board. The third module will be multiplayer support on one computer. Then, a single player versus computer controlled player. The next will be networked support for multiplayer on different machines. The last modules will be additional features we add to the already working game, such as the dynamically generated maps and expansive tech trees.

7.3 Team Organization

Our team will pair program - this helps avoid basic coding mistakes and will help when working through logical errors and algorithm design. We will also use a democratic method for settling issues within the team - since it's a team of three, one of two sides will receive two votes, and that is the direction we'll take. While each member will contribute to every element of the game, we decide that we should each also have an area of focus for the concepts we are less familiar with. Brian will focus on graphics, Matt will focus on networking, and Benson will keep the documentation tidy.

7.4 Process Assessment Rules

Unit tests will be built in PyGame library and we will handle risk management as outlined below. We will also use the unittest library built into Python, so that we can quickly discover when new code breaks older code.

7.5 Revision Control Methods

We will use git because it handles version control well and is easy to use. For this project, our team will use github to store our online code repository. User requirement changes shouldn't be an issue, as this is an in-house game so all requirements have been outlined already.

8 Risk Identification

8.1 Table of Potential Risks

ID	Title	Description	Probability	Exposure
1	PyGame library lacks functionality	PyGame library may not have methods or modules that are necessary for our game	High	High - the cost of impact would be a reduction of features in the game or possibly the inability to complete the game
2	AI Issues	The AI player for single player could be too strong/too weak	Medium	Medium - imbalanced AI would not be game-breaking but would make it much less fun for single player.
3	Randomly generated terrain is unfair	Each player has randomly generated terrain - if it isn't fair, one player could be at a significant disadvantage	Medium	Medium - this would make the game much less fun

8.2 Risk Management

1. PyGame library lacks functionality: This risk could potentially be very harmful (high exposure). Luckily it is very easy to prepare for and solve. PyGame is a set of Python modules (a library) that gives easy access to many tools necessary for game development. However, it could be that all the functionality we need to design our game is not provided within this framework (for example, no code for saving and loading a multiplayer game). The way to alleviate this risk would be to write our own modules or add to the framework the necessary functionality.

2. AI Issues: From working on single player versus AI games before, our team realizes that having a strong yet beatable computer-controlled opponent is a very difficult task. Therefore it is a risk that must be dealt with ahead of time. Although it could be that designing an appropriate AI is easy, there is a medium probability of running into large issues, so this again must be factored into our risk management. To plan for this risk, balance issues and simple ways of controlling player actions must be kept in mind at each step. If writing a computer-controlled player becomes impossible, single-player functionality would be lost. To quantify this, a player who has never played the game before but has experience in video games should be able to beat the AI within their first three games.

3. Randomly generated terrain is unfair: Each player has a board that will have randomly generated terrain and bonuses. If this board isn't fair, then it could make the game very imbalanced. We will have to take steps to make sure that boards are random but quantifiably equal. Even if there are specific limits to board features, improper timing of obstacles appearing or lots of bonuses at the beginning of a game could cause unfairness. This will have to be kept in mind during programming of board generation.