**Game Proposal and Functional Requirements Document**

**Authors:**

*Jack Kelly  
Conor Golden  
Sanders Lauture  
Arjun Rao  
Chris Kuffert*

**Game Concept & Breakdown:**

**Important**: Unfortunately, due to complications with cocos and the difficulty we had in implementing each piece of our game, not every aspect below was able to be completed. Under the final milestone, we explain what we completed, and what we unfortunately could not.

The concept for our game is a 2D, squad-based/turn-based, tactical shooter in which the player manages both macromanages squad finances and micromanages in-game combat. The combat will be a turn based top down military simulation similar to *Advance Wars* on the SNES, drawing a principle influence in style from Counter Strike. The money managing system will be aimed to reflect a blend of those systems of *X-com: Enemy Unknown* and *Sim City*.

The player will start with an initial budget with which they can hire soldiers and purchase weapons/gear. Once they have finished spending their initial budget they can advance to the next day where they will get their first mission. If the player accepts and completes the mission successfully they will be rewarded with more money to spend on paying bills, paying salaries to soldiers, hiring more soldiers, or making more purchases. If the player declines the mission or fails to complete the reward in turn will be less. If the player has negative funds for too long they lose the game; there is no end-game winning condition because the player can continue to play in order to expand and upgrade their squad indefinitely. When in combat, the player will use their squad members to navigate, organize, and fight on a grid based map against enemy AI. Once all enemy AI is defeated the player will return to the dashboard, where they can continue to manage their finances and move on to the next day. When a soldier from your squad is killed in action, they are permanently dead (yeah, perma-death!!) and neither their life nor their gear can be recovered once they die. If a soldier is injured in combat but not killed, they will be out of action. Their time on the disabled list and the cost of their medical bills will be indicative of the severity of the injury. The player will have to hire more squad members than needed to play a mission so that the roster does not run out of useable soldiers. The total number of units that can be in a battle at the same time is twelve, with six units per squad.

**Programming Distribution:**

* **User Interface.**
* **Gameplay & Menu**
* **Grid-Based Game Maps.**
* **Unit (Sprite) Class.**
* **Enemy Class.**
* **Enemy AI.**
* **Bosses/Tiers**
* **Player Class.**
* **Team**
* **Finances**
* **Game World Class.**
* **Item Class.**
* **Weapons.**
* **Gear & Equipment.**
* **Graphics/Output.**
* **Game Runner.**

**Milestones:**

Our first milestone for November 20th will be to have all of the classes and methods that are deemed necessary to complete the project outlined but not fully implemented, similar to extent of development behind Assignment 3. All of our classes will be designed and well-commented. Some of methods will be written, but most others will only be outlined (with pseudo-code). As for testing, the main method will create the instances of all the classes, and make sure everything is created correctly without errors. This milestone will NOT have any cocos files or projects. We had significant problems with installing this software, and currently are working on fixing this problem. So for this milestone, basic structure of our code will be completed, and the main will test for their correct functionality.

Our second milestone for December 7th: We have added a few new classes to represent the map information; tiles, locations, etc. We have added the required methods necessary for these new classes. At the time of this milestone, we have a runnable game that displays the first map level. Due to finals, projects and the break, we were not able to accomplish what we initially thought we could, however we made quite a bit of progress to our end project.

**Final Milestone, completed project:** Due to complications with cocos, and the time it took to work out the aggrivating and tedious problems with this software, we were not able to completely implement all of the code we wrote into out game. However, all of the code that we set out to write from the start of this project is there. But unfortunately we were not able to fully implement every aspect of our game.

When the game is run, the first map will load, with the player's units are in the bottom left corner of the map. They can move the cursor (which starts in the bottom left) with the WASD keys, and select units with the space bar. Once a Unit is selected, he can be moved by pressing spacebar on another empty square. To attack another enemy, select an enemy with the space bar, and select an enemy by pressing space bar again. This will have your unit attack the selected enemy. This will kill the enemy, leaving them stationary, while all the other enemies move around. Once all the enemies are killed, the game is over and the player wins.

**Verification:**

*In order to address the verification of our game, in the sense to prove that this program will be the result of our collective work, we’d like to state that this game is completely original work of our own.*

**Additional Software:**

The two works of software that we are using in order to build this program are ***Cocos2D*** and ***Tiled***, both of which are designed to work with one another effectively and rather effortlessly. As for the former of the two, Cocos2D is a C++ open source game library that can export to iOS, Android, and Windows Phone. We chose to use this platform because it is:

* *An easy to learn 2D engine.*
* *Effectively integrated with the Tiled map editor.*
* *A great opportunity to learn a graphics library through our application & understanding of C++.*

Tiled is a map editing program that allows the user to create a palette of tiles that can be easily placed/copied over a grid of any determined size. The Cocos2D engine can interpret the *Tiled* .tmx files and thus greatly simplify the map making process for the game. Otherwise depending on the game engine to assist in map development would prove to be difficult given the huge influence that unit location on the game board has in terms of gameplay.

**The Team:**

Our team will consist of five members: Jack Kelly, Conor Golden, Sanders Lauture, Chris Kuffert, and Arjun Rao. Each of us is taking on an equal portion of the coding requirements and we will agree on the roles before taking on the project. However we plan on collaborating on the project from start to finish, helping one another through developing the program; there will be specific classes and segments of the code that must be collectively written by all of us.

While we do not all have the same number of classes to take on, some require more code than others and we plan to balance out the programing work as much as possible so that we are all doing approximately 20% of the code itself.

As previously diagramed above, there is a basic structure of the code and a description of how we plan to divide up the programming work. Here we will go into further depth on how each division of coding work is distributed amongst the group members.

Arjun will take on the user interface and develop the grid based maps, coordinating how the player will interact with the game both in & out of game. Jack will program the unit class – the base class serving all units/sprites that exist in the game world – as well as taking on the (derived) enemy class. Conor will code the player class, where he will focus on all aspects of gameplay that are out of real-time, such as the player finances and team management. He will also develop the world class. Chris will program all of the guns/gear/items of the game, organizing and dividing the weapons by class and by weapon. Sanders will coordinate the graphics by working primarily on the integration of *Cocos2D* and *Tiled* together. He will also work on the game runner class. Ideally, this will cover 20% for each of us and while more classes/methods may be deemed more demanding during development (in terms of timing), we will work to keep each member doing roughly the same amount of code as any other. We are all very comfortable with bringing up the topic of work distribution amongst the group, noting that we recently worked together on a different project in the first Northeastern GameJam of this academic year.

**Conclusion:**

All the parts should come together once we have spent a significant amount of time planning a detailed class structure. If we spend enough time with a very specific class structure, dividing up the work, coding and then piecing it together will not be difficult. The planning phases will all be done together so that we all have a perfect sense of the contents and inner workings of the game, while the actual coding of methods can be done individually. We will meet weekly to check up on progress, bug fix as a group, and continue to iterate on our plan.