**Engineering Methods**

**Software Process Model:** Waterfall - we have already started on this method. We initially did our 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.

**Programming Development Method Selection:** Incremental - we will first develop a single-player prototype where there is a board and movable pieces. Then we will develop the second board, with possibility of sending units to 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.

**Team Organization:** Our team will use pair programming - this helps avoid coding mistakes and will help when talking through logical errors. 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 2 votes, and that is the direction we'll take. As for specific roles, Brian will focus on graphics, Matt will focus on networking, and Benson will contribute to all other areas as will Matt and Brian. This means all members will contribute to most programming tasks.

**Process Assessment Rules:** Unit tests built in PyGame library and risk management as outlined below. We will also use the unit test library built into Python.

**Revision Control Methods:** For this project, our team is using GitHub as an online code repository. Git handles version control and is easy to use, especially through GitHub. User requirement changes shouldn't be an issue, as this is an in-house game so all requirements have been outlined already.

**Risk Identification**

Risks for our project are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Title** | **Description** | **Probability** | **Exposure** |
| 1 | PyGame library lacks functionality | PyGame library could 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 |

**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.