BushBurg: Survival in Africa

Project Plan

for

Amapalo, Inc.

Matthew Callen, Khangal Erdenetsogt

**Section 1: Motivation and Objectives**

**Motivation for Pursuit**

It is easier than ever to feel connected to the world around us—at least, parts it. It is common for us to keep up with news in Europe, Asia, and the Middle East. Many of us have regular interaction with those an ocean apart through online communities. However, increased connectivity between inhabitants of the first world has further disconnected us from those of the third world. It is still difficult to make people care about those with which they have shared no contact. Additionally, the image of charity has taken something of a beating in recent years. Between fraudulent enterprises, counterproductive results, and the opacity of most charity expenditures, it has become difficult for charities to justify their existence to new generations. The pursuit of funding and volunteers for humanitarian efforts must evolve with the 21st century.

BushBurg is an attempt at exactly that type of evolution. Gaming is regarded as one of the most subversive and effective means to understand how it feels to be in another person’s shoes. The medium has also been shown to convince people to invest money willingly on a regular and long-term basis. These two facets of the industry lend themselves very well to raising money for charity. The trick, of course, is to make a compelling game experience which facilitates donation.

Our job is to build the foundations of that game experience, for a casual mobile market. The client wishes to simulate, in spirit, a plausible development of a community in rural Zambia. The cultivation of land, the perils of the environment, and the hope of investment paying off should all be represented within the game. There is no expectation of the struggles of the player to be true to life, but the idea is to develop a kinship with the simulated inhabitants. Done right, this process will motivate players to donate through in-game purchases; perhaps this will be done for self-interested rewards in game, or simply for the sake of helping. Those in need will benefit either way.

Programming Objectives

It is important to distinguish what it means to build the foundations of a game, especially in the context of one meant to be a social experience. A team of two people with full schedules cannot hope to build a well-rounded game experience in 13 weeks. In addition, many aspects of the process require skill sets that are not developed in Computer Science curriculum. Most of the problematic requirements are involved with the development of art assets. Things like sprites, models, particle effects, sound effects, and music are far outside the expertise of the team. Fortunately, these aspects can be developed at a different time, with a different team. Integration of graphical and sound assets is actually made quite easy by current-generation software. Amapalo, the client involved with this project, has expressed that these aspects are planned for at a later period, and will not require our involvement.

Social angles to gameplay also cannot be reasonably implemented. This would require use of both a server infrastructure and a test community- neither of which is currently available. Instead our goal is to produce a single player prototype that examines the most fundamental part of gameplay. We will develop a set of rules which govern how the user manages his or her citizens in a strategic manner. This is a vertical prototype, meaning it fully explores a single set of ideas as strongly as possible. From there, we can fully understand the implications of that set of mechanical rules and how they might be altered to accommodate more complicated rules in future iterations.

Section 2: Project Organization

**Team Makeup**

* Matthew Callen
  + Leader, scribe, documentation, design, development
  + Game programming degree
  + Experience with Unity engine and language to be used, C#
  + Two self-started independent game design projects
  + Career goals in independent development
* Khangal Erdnetsogt
  + Version control, tracking, development
  + Computer science degree
  + Experience with Unity engine and language to be used, C#
  + Experience with tools such as Photoshop, Illustrator, 3DSMax
  + Past work as webmaster and software development

Roles in Implementation

The Unity engine is an object-oriented environment with many possible ways to develop and integrate elements of design. Two separate systems generally would require a four step process.

1. Discuss the purpose and role of each system abstractly, making estimations of how they might interact and how to make integration as straightforward as possible.
2. Develop the systems independently and verify their expected functionality.
3. Merge the systems into the same environment. Because of the modularity of Unity, this rarely presents any problems.
4. One developer is then tasked with the interaction between the two new systems. The other developer may begin work on a new system independently.

From initial discussions between the team members, we have decided to split the implementation into two major categories: gameplay and interface. The sequence of four steps described above is somewhat altered in this case, because instead of using two independent systems, one system is reliant on the other. In this case, one member creates a new module. While a second module is being developed, the first is being integrated and interpreted by the UI to relay relevant information back to the user. This creates more of a pipeline effect and requires fewer integration steps. The roles for implementation are as follows:

Matthew: Due to experience with game design, Matthew will be responsible for the generation of rules for play and the implementation of new gameplay features. Player input, game object organization, and data sourcing will all be done by Matthew during development.

Khangal: While we do not intend to develop any art assets, we do require a means of distinguishing objects for our own testing purposes. Khangal has experience with graphical interfaces in Unity, and will be handling the interface requirements set up by gameplay changes. After implementation of a new module, Khangal will integrate this module so that it can be represented in the interface in a useful way.

Section 3: Risk Analysis

Development of a video game is inherently more risky than traditional software. This is because there is no guarantee that fulfilment of software requirements and objectives will achieve a desirable outcome. As an example, consider word processing software. It is obvious what the design goals of this software will be, and why specific elements, such as spell-checking, should exist within it. As long as these elements are developed to specification, the goal of the software has been met. With games, however, the goal is much more abstract. Design a compelling gameplay experience. The rules of play themselves, that is, the list of requirements of implementation, are somewhat experimental. Even if they are implemented perfectly, those rules may have been incompatible with a ‘fun’ gameplay experience to begin with. There is no way of knowing this for sure until the ideas have been tested within the game. This means that game design is highly iterative. The requirements will surely change over time, and there is no way to predict the exact risks that await us. There are some *categories* of risk, though, that can be described.

1. **Gameplay elements and UI elements are unable to be easily integrated due to coding practices:** User interface elements draw on data from the game during runtime. This means accessing variables or methods with return functions in order to maintain updated information. To avoid problems with the integration of these two things, meetings will need to take place early on to develop a standard mode of interaction.
2. **The UI system is unable to display required information in an accurate way:** There are several implementations of a user interface available to us. It is hard to know, ahead of time, which system is most appropriate. Canvas-based UI elements, for example, scale automatically with screen size and have high versatility in terms of text. Object-based UI elements require more work and don’t scale as easily with screen size, but have the advantage of being very precise. We will need to choose one early, and monitor its effectiveness. It is important to figure out whether our choice is appropriate as early as possible in order to minimize lost work.
3. **It is not possible to reconcile the mechanics with the constraints of the platform.**  We will be developing and testing the prototype through a Windows PC developer environment. While conversion to a new platform is relatively simple on the software front with Unity, conceptual mistakes could lead to points of no return in terms of small touch-screen interactivity. We must evaluate the game as it stands at each milestone to ensure that conversion to the platform constraints would not render the game unplayable.
4. **The core mechanical system implemented is inherently uninteresting or unbalanced.** The first milestone will be an experiment of sorts. There is no way of knowing whether our system of rules will provide the experience we want. During the implementation process, we must make note of alternative designs that would be easy to iterate quickly, in case the first system is dysfunctional.
5. **Scripts become too difficult to manage and navigate over time.** Because we don’t have a rigid system skeleton to rely on, it is very likely that code structure will need to be reconsidered at various points during progress. This is a risk that will almost surely occur at some point. As a result, there is a planned period of refactoring that will occur after the first milestone. This will occur after the milestone has been tested, and gameplay concerns are better understood by the team. At that point, we will have a better idea of ways to manage inheritance, polymorphism, and the relationship between objects. Attempting to do this in advance would most likely be counterproductive, as we cannot reasonably model the relationships between mechanics we have not yet decided to use.
6. **Our implementation does not meet the expectations of the client.** Early discussion about gameplay ideas is abstract, and there are some assumptions made by the team that will be tested after considerable development. It is possible that the direction of the game will be misunderstood on our part, leading to a set of mechanics that, while feasible, are not the desired outcome in terms of the client. Regular correspondence, as well as a mid-milestone discussion, will be required to mitigate this possibility.

**Section 4: Resource Requirements and Work Breakdown & Schedule**

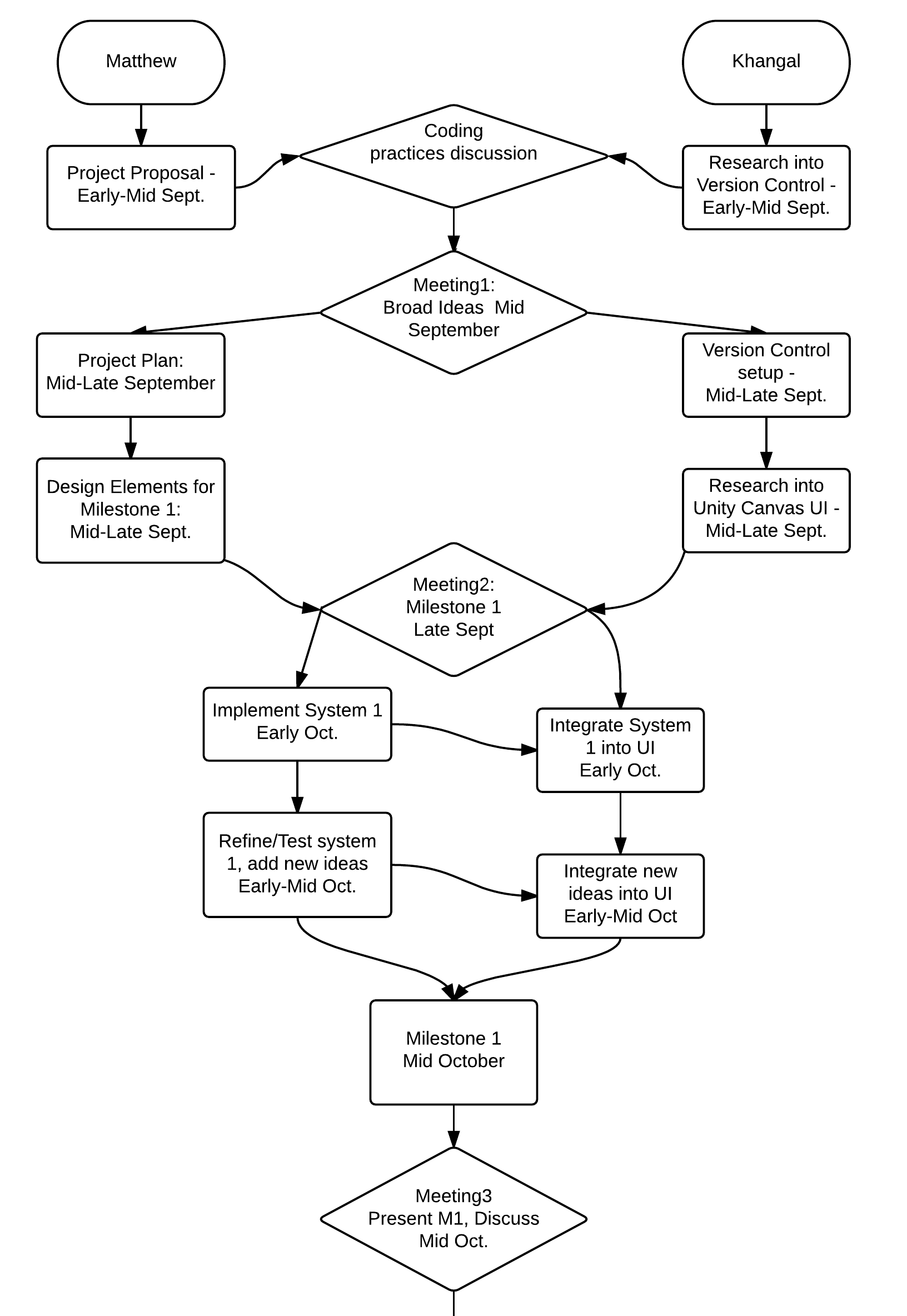
Resource Requirements

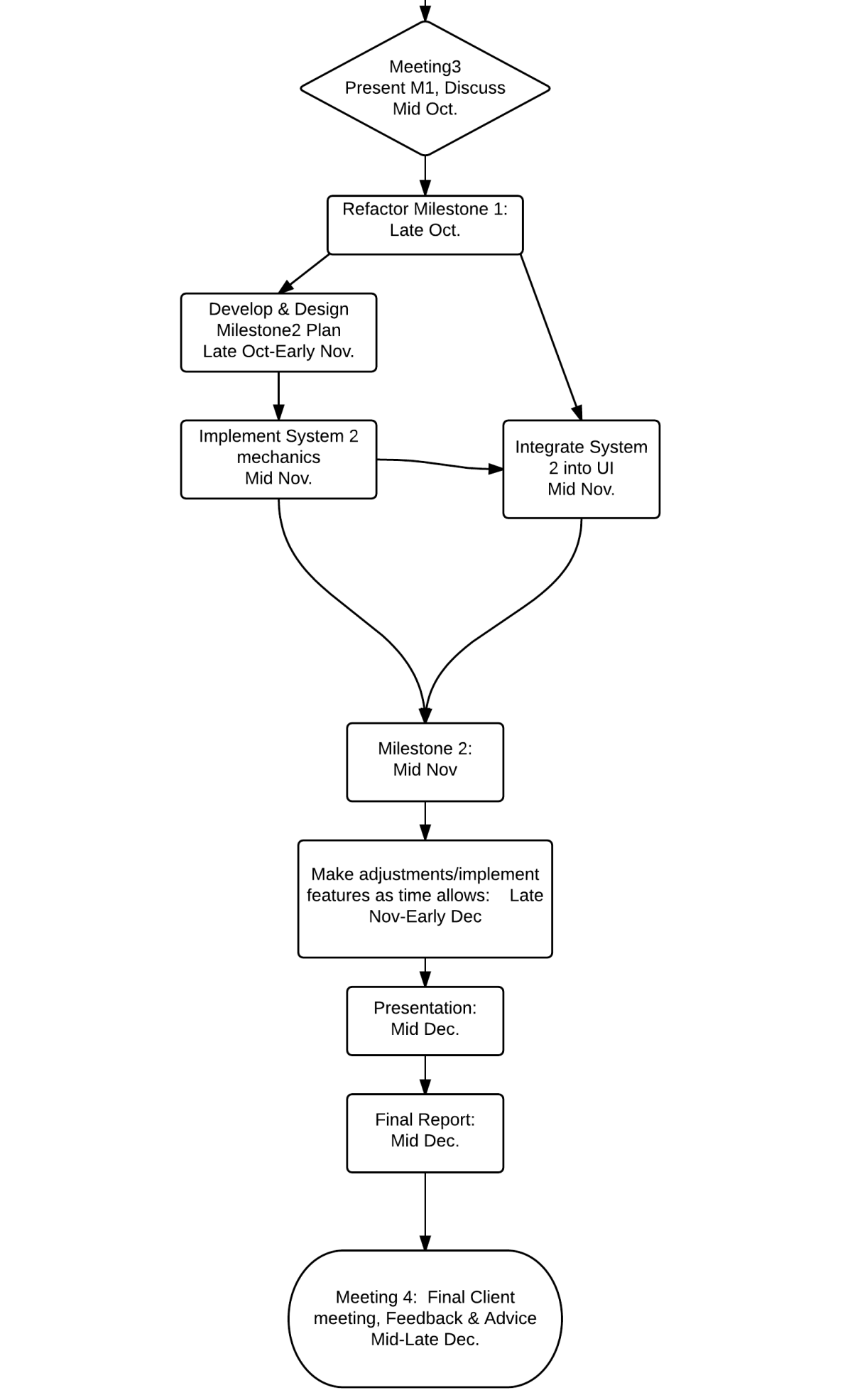
As mentioned in programming objectives of section 1, we are unable to obtain some of the resources that would go into a full-size development. These include things like server infrastructure and a community of testers. Development is scaled back as a result, allowing for minimal outside resources. In the prototype we have chosen to develop, few outside resources are actually needed.

* Unity engine development environment: This is the core foundation of the prototype, and is offered for free non-commercial use from their website, [www.Unity3d.com](http://www.Unity3d.com). Licensing is required for a full commercial release, but we do not need it to begin work. The editor contains its own IDE in MonoDevelop, but we will use integration with Windows Visual Studio in our work.
* Art assets for recognition and distinction of game objects: The prototype in development will not require use of personally developed art assets, but will instead pull images from any domain necessary. Primarily, this means use of photos of various crops, insects, or images which may be used to represent hazards. Because this version is specifically for our own use, and thus fair-use, and will not be featured in any final product, copyright is of no concern.

Work Breakdown Structure/Schedule

Game development is highly iterative, and is generally difficult to predict in regard to module structure. However, given the previous experience in Unity and game development by the team, we are able to present some reasonable expectation of how development will take shape. Below is a preliminary summary of the development process for BushBurg.





Section 5: Monitoring & Reporting

Two weekly meetings will be held before the scheduled class meetings for the course. In order to maintain a working list of documents and code, GitHub will be used. GitHub offers version control and merging utilities which will facilitate easy sharing and coordination of code. The Unity Engine itself will also aid us in that regard, because of its modular design and standard components. Communication out of meetings will be maintained through Slack, an instant messaging tool which is focused on quick response and the sharing of code snippets. Additionally, a Google Drive folder will be set up to share documentation and builds with the client.