COVER PAGE

Confirm rubric.

Server like we specified in requirement document: controls weather, terrain

Unity object oriented.

TABLE OF CONTENTS

**1. Introduction**

1.1 Introduction

Purpose of the document / audience of the document:

Terms definitions (if any):

1.2 Policies and tactics

Goals of the design: what we want to maximize and what we want to minimize

* maximize usability (user-friendly), simplicity (how do we measure it?)
* multiplatform
* minimize system reqs,

Guidelines: what rules we should follow for the design

* we are constrained to a particular coding style described later
* time constraint
* communication - everyone should be communicating and knowing how they are progressing (scrum meetings?)

Dev methods: how we design, what techniques (scrum?), how we build the program

* slack meeting everyday at a time
* we meet at the start/end of every sprint on Monday at 3:00 - 4:00 loc:TBD (redpath)
* we create a backlog

**2. System Overview**

2.1 Architecture considerations

tech selection: what are the tech choices and why

* unity: accessibility and time constraint
* C#: cleaner language than javascript, similar to java
* git: we’re used to using it and it works (ideological: public or private?)
* MySQL, Azure : Azure provides free space for a MySQL database

idea choices: what and why, we probably won’t do anything for this but we’ll ask the prof

2.2 Design considerations

Assumptions:

* users are stupid so needs to be simple/familiar
* some computers aren’t too well performing so it should work on lower performing ones (no specifics yet)

dependencies:

* Backend
  + Azure working as advertised
* User’s Computer
  + Unity (user will have to install unity player)

constraints:

* Graphical quality
  + Accessibility constraint

2.3 High-level design

System overview:

Design patterns: system wide

MVC

Singleton

Final domain model: top level

identify design patterns in the boxes

class names and relationships

Final deployment diagram:

discuss each box, tech used, purpose

**3. Design Details**

3.1 Software Organization

block diagram of system and subsystem

3.2 Module/Object decomposition

class diagram of subsystems

calling sequence diagrams

state diagram & interaction diagram (as needed)

strategy pattern for animal ai and plant growth

composite pattern for terrain

observer pattern for notifying actions

3.3 Critical Section Specification

Identify the critical sections and provide the algorithm for it

* snippets
* name the alg.

Auction house

using MySQL service from Azure, transferring part of the risk

SQL queries, look up documentation

3.4 Entity-Relationship & Data Dictionary

* items in the auction house, and such

3.5 Low-level architecture (if needed)

**4. Programming Management**

4.1 Directory Structure

In the root directory, we will have 5 subdirectories: *src, bin, test, assets, documentation*. The *src* directory, as its name implies, contains the source files, including Unity source files as well as game scripts, classes into sub-directories. The *bin* folder will one subdirectory for each version of the program and each one will binary executables of that version for different platforms. The *test* folder is subdivided into the different modules that we have. Each module of the project will have a subdirectory that contains the test scripts specific for that module. The *test* folder will also have a general test script that executes the test scripts for each module one after the other. The *assets* folder contains the different assets that the program may use. The assets can be texture images, 3-D models, audio files, configuration files, etc. Finally, there is the *documentation* directory. It is subdivided into *user* and *dev*. *user* subdirectory is used to store user manuals and any documentation files intended to instruct the user. *dev* subdirectory contains different documentation files on our development style to facilitate project transfer in case it happens. This also makes it easier for the programmers to review the coding and repository rules in case they forget. The root directory contains a readme file that describes the project in general and can be used as the presentation page of the project on the remote repository.

4.2 Programming Tools

During the development of our project, we will use many programming tools. For revision control, we will use Git. For text editing and code writing, we opt to use Sublime as it is the one the team members are the most familiar with. We also use the Unity Engine as the framework of the project.

4.3 Software Building Method

Scrum

* 7 days sprint
* Daily standup meeting (on Slack)
* Sprint meeting every monday at 3:00 PM, sprint review and assign tasks

Peer review rules

* Peer review session during weekly sprint review
* Entire group review the code

4.4 Coding Agreement

Coding style

* this{
* }
* primitive types have identifiers: eg. nVar, (look at “Hungarian notation” for specific identifiers and maybe add to this doc?)
* javadoc comment style
  + //function name
  + //description
  + //Params
  + //Return type (and what)
  + //remark: (exceptions ect.)
* always need a debug version/var: bool \_DEBUG
* don’t use stdin stdout
* any additional comments
* return bool instead of void for success check

Repo rules

* never push on master branch or dev branch
* create a new branch when working on something
* push a test script when pushing (not always)

Team communication

* use slack and in person communication

4.6 Installation Procedures

To facilitate the installation, we will release a setup file that allows the user to install the whole program by simply executing it.

4.7 Training Guidelines

User manual and in game tutorial

To train the users, we will write a user manual that details most of the actions that user can make as well as their consequences. For those not patient enough to comb through the documentation, we will also have an in-game tutorial that briefly explains the different components of the game.

**5. Test Cases and Testing Procedures**

5.1 Test-driven Environment

How is the code tested internally

Automated testing methods

In code checks (precondition, postcondition)

Error logging

Test schedule and procedure

5.2 Acceptance Test

Functional test cases

Using the software in real life scenarios

**6. Appendix**

End goal:

Classes:

Player

Terrain

Sand

Mountain

FlatRock

<<interface>> Lifeform

<<abstract>> Animal

A1

A2

<<abstract>> Plant

P1

P2

<<interface>> Item

<<interface>> Consumable

<<interface>> Resource

Food

Metal

AuctionHouse

Ship

Animal Spawner

<<interface>> Tools