Ghost Grab



Dylan Dover, Selina Chen, Jason Mayer, Lawrence Waller

ENGM 3700.02

04/21/2016

# Executive Summary

Our company will design an innovative Android game called GhostGrab that will utilize GPS coordinates to create an immersive environment. We will utilize a number of tools and frameworks to accomplish this goal. Among them are Unity on the client-side and node.js on the server-side.

GhostGrab will be a location-based game that will allow users to use their phones to view and capture “ghosts” across a chosen location radius and release them or set them on others for points. Users will have to play mini-games or solve riddles to capture ghosts and there may be random team competitions, allowing users to collect bonus points. There will be a leaderboard, which will update in real-time to show who is leading in points and allow users to track their rank and the ranks of their friends. Created in Unity, it will be easily portable to a variety of platforms, though for a minimum viable product, we will be releasing an Android-only version.

To keep the game interesting over time, updates will be available in the form of new ghosts and new mini-games. This model makes it possible to easily update the game with minimal time investment, meaning that our company will have time to pursue other projects simultaneously.

Contents

Executive Summary 2

The Project 4

Background and Rationale 4

Budget 4

App Functionality 4

The Code 5

Scheduling 6

Responsibility Matrix 9

Risk Management 10

Problems & Solutions 11

Quality Assurance 12

Future Plans 13

Ghost Grab Updates 13

Similar Applications 13

# The Project

## Background and Rationale

GPS has become widely embedded in mobile devices, and games that interact with users in the real world have become more and more popular. With the recent announcement of the VR (Virtual Reality) game, Pokémon GO™, and already existing augmented reality mobile apps such as Ingress, there is a lot of widespread interest in games that users can bring into the real world.

The idea for a game based on catching and releasing ghosts came from the movie franchise, Ghostbusters, which is getting a reboot this year. The ghost theme is generally well-received by consumers. As such, there is reason to believe that an application combining a virtual reality component with a ghost-based game will be successful in the marketplace.

## Budget

GhostGrab did not require any financial investment to create. If we were to launch it to the Android app store we would incur some costs but we have not reached that point. We would also have significant cost involved in handling a large volume of requests to our server. Our only investment was labor hours. We had four people working on the project and spent about 50 hours combined developing the product.

## App Functionality

This app is available for Android phones. A basic walkthrough of how the app is used is demonstrated through the pictures below:

First login by creating a username and password? [INSERT PIC]

Next

Happy ghost hunting!

Some other features include a leaderboard, where you can see who is leading in points.

[INSERT PIC]

## The Code

The code for this app as well as other documentation is publicly available and can be found at <https://github.com/wallerl2/ghostgrab>. A basic layout of the way the code is set up is demonstrated in the UML below:



Figure 1: UML Diagram for App

The client, which is the code written in Unity, holds the graphics for the game as well as the most of the game logistics. The server, which is being hosted on an Amazon Web Services instance holds information such as ghost types, location of all the ghosts in the game, and the scores of the players. The client requests information from the server whenever it needs to for tasks like determining ghost locations. It also posts information to server as well. For example, the updateLeaderboard() function is called when a user’s score changes, and the updated score is posted to the server.

# Scheduling

This project took place over a period of about 3 weeks, starting on Tuesday, March 29th, 2016 and ending on April 21, 2016. An overview of the tasks performed for the project are listed below in Figure 2.(MIGHT NEED TO CHANGE #)



Figure 2: Gantt Chart Documenting the Project Schedule

The project was split into four phases: Initial Design, Bare Bones, Art and Advertising, and Final Product. Though called ‘phases’, these periods do not necessarily go in order or depend on each other’s completion. For example, the tasks in Phase III: Art and Advertising necessarily succeed the Phase II subtask ‘Set up running Project in Unity’, but do not need to succeed the other subtasks in Phase II. The Phase III subtasks of designing ghost types and making the app pretty can easily be performed concurrently with the Phase II subtasks of setting up the server and client.

Our project has a single milestone on 4/20/16, which is the printing and binding of the final report, which is done on 4/21/16, the day before the report will be handed in.

For a better look at the entire project schedule and which tasks must precede others, one can view the Network Diagram form of the schedule below.



Taking a closer look at the beginning of the schedule we see that the Initial Design Phase had four subtasks: Prepare Tools, Writing Documention, Take Pictures, and Research. The 2nd Phase, which involved getting the bare bones of the app set up followed after the Research activity.



Phase II: Bare Bones had Phase IV: Final Product as a successor. However, Phase III has only ‘Set up running project in Unity’ as a predecessor and no successors. The disconnect from the other Phases is due to the fact that the subtasks in Phase III involve activities that are really only about adding extra features to the app, which can be added to or stopped at any time (there is no cap to how beautiful we can make the app or how many ghost types we can add).



The server must be set up before we can connect the server to the app and obviously, the final product must be finished before we can present and hand in the project.

# Responsibility Matrix

The responsibility matrix, assigning tasks to each person is shown below. As Project Manager, Dylan Dover had an ‘A’ for Approval on all tasks with the exception of the tasks which he was Primary or Secondary on.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Step Name** | **Dylan Dover** | **Jason Mayer** | **Lawrence Waller** | **Selina Chen** | **Complete?** | |
| Prepare Executive Summary | A |  | P |  | Y |
| Make Gantt Charts | A |  |  | P | Y |
| Unity Web Requests--Client | S |  | I | P | Y |
| Unity Web Requests--Server | P |  | I | S | Y |
| Mobile Interactions | A | P | I | S | Y |
| Art Assets | S | P | S | S | Y |
| QA Testing | S | P | S | S | Y |
| Web Server | P |  | S |  | Y |
| App Integration with Web Server | A | S |  | P | Y |
| Location on App | A |  |  | P | Y |
| Map on App | A | P |  | S | Y |
| Collision and Encounters | A | S | P |  | Y |
| Leaderboard Updating | S |  | P |  | Y |
| Setting Traps | A |  | S | P | N |
| Trailer/Intro Video | A |  | P |  | N |
| Proximity Detection | A | P |  |  | Y |
| Location Interface | S | P |  |  | Y |
| Web Site/Forums for Fans | A |  | P |  | N |
| Prepare Final Bound Report | S |  |  | P | Y |
| Prepare Poster | A |  | P |  | Y |

# Risk Management

Some concerns that posed risks to our project are shown in the table below. We wanted to keep our Risk Rating below 30%.

|  |  |  |  |
| --- | --- | --- | --- |
| **Production Delay** | **Likelihood** | **Impact** | **Expected Delay** |
| Incompatibility between server and client-side code. | .65 | 2 days | 1.3 workdays |
| Creative Differences and conflict resolution among the team might slow production. | .20 | 2 days | 0.4 workdays |
| The Google Play store might reject our app for copyright infringement. | .05 | 4 days to redo art assets. | 0.2 workdays |
| Server Downtime could delay testing of functionality. | .01 | 1 day | 0.01 workdays |
| Chromecast might not work with campus internet. | .50 | ½ a day | 0.25 workdays |
| **OVERALL DELAY:** | | | 2.16 workdays |
| **DELAY, AS A PERCENTAGE OF TOTAL WORKDAYS (21)** | | | **10.29% Risk** |

Our project faced two different kinds of risk—monetary risks and time risks. The time risks are shown above.

Because our project did not require any funding, there was no risk of squandering start-up capital. Indeed, the only assets wasted during the production of GhostGrab were the man-hours of our team members.

One financial risk that potentially could have threatened our project involved liability. Ghosts can spawn anywhere in the United States near a registered user, and since over 1% of the country is paved over with roads this represents a small chance that ghosts spawn in dangerous locations. We added a disclaimer to our app warning users of this fact so that users cannot sue us if they use the app while driving or in dangerous situations.

The most critical risk is one of Copyright Infringement, as some of our ghosts (such as ‘Shrok’) are based on already existing franchises. Our game idea in general was actually based on the Ghostbusters™ film franchise. In order to mitigate this risk we have worked to keep our content and art within the Constitutionally-protected realm of parody and satire.

# Problems & Solutions

Many of the initial problems that arose came from client-server interaction. There was some confusion at first about how to connect the server, which is written using jQuery and JSON, with the client code in Unity, which was written in C#. This issue was solved using a public library called UnityHTTP provided by <https://github.com/andyburke/UnityHTTP>, which is itself based on Simon Wittber’s UnityWeb code, and therefore is licensed under GPL. GPL stands for General Public License, which allows end users, whether they be companies or individuals, to run, study, modify, and share the software. This eliminated much of the work and allowed us to write web requests without worrying about how exactly Unity interacts with JSON.

Another issue was finding a way to host the server. Amazon Web Services was the obvious choice, but there are different types of instances available and there were minor issues with many of them. The preferred instance was a Ubuntu machine. However, this machine was incompatible with our server because Node, which was used to run our server, is called NodeJS on Ubuntu and is slightly different. In particular, it is incompatible with the map data structure used for storing user information on the server. Eventually, it was discovered that our server functions perfectly on a Linux RedHat instance.

Yet another key issue was making sure that ghosts spawn close enough to users, but not so close that there is no effort involved in capturing them. This has yet to be fully solved, as we are using static ghost spawning in our minimum viable product, plans to solve this issue in the future have been considered. GPS coordinates can be used to determine an inner radius and outer radius from a user, and ghosts can be spawned in between these.

A final issue was making sure that ghosts do not spawn in locations that would be dangerous to visit. In particular, since our ghost spawning mechanism just places a ghost in a random location between two radii, it is entirely possible that they could spawn on a highway or other dangerous place. Unfortunately, being able to prevent this dynamically proved nearly impossible. To circumvent this issue, we have decided that our final product, once we have solved the previous issue and use non-static ghost spawning, will include a clear disclaimer mentioning this issue and disavowing liability.

# Work Breakdown Structure

GhostGrab

Initiate

Plan

Execute

Control

Close

Brainstorm

Create Art

Assets

In-Class Demo

Write Server

Create API

Connect Server

to UNITY

Turn In

Report

Assign Resp.

Navigate

AWS

Consider Product

Longevity

UML

Diagram

Risk Manage.

Prepare Poster

Post to Google Play

Store

Gantt Chart

Executive Summary

Create

Video

Art

Assets

Proximity

Detection

Set

Traps

Leaderboard

Updating

Manage

Server

Quality Assurance

Figure X: Ranges for display messages

The key to maintaining quality in our application is to make sure that GPS is utilized effectively. For instance, users need to be able to capture a ghost only when they are sufficiently close to it. Still, the reality is that GPS is not perfect and is somewhat variable on different devices. As a result, we have implemented a three-tier notification system. If the user is within .0001° (Pythagorean distance) of the ghost, he or she will be given the opportunity to capture that ghost. Within .0005°, the user will be alerted that a ghost is very near. Within .0007°, the user will be alerted that a ghost is near.

These ranges are not useful only because they provide information to the user, but they also make the system more redundant. If the GPS inaccuracy makes it seem as though a user cannot capture a ghost when they should be able to, they will at least be alerted they are close and can move closer and capture the ghost. This is the best possible solution given that we cannot control the quality of GPS on user devices.

# Future Plans

## Ghost Grab Updates

The current model of our product allows users to capture ghosts for points, which while acceptable for a minimum viable product, is somewhat boring. As mentioned in the executive summary, we intend to release updates, which will contain more features to make the game more interesting. One such update we plan to release is one that will allow users to set their captured ghosts on other users, which will drain their points. Currently, ghosts are differentiated by their appearance and the points they carry. In the future, we would differentiate more between the ghosts by adding mini-games and movements specific to each ghost type. This gives us endless possibilities for updating our game and maintaining our user-base over time.

## Similar Applications

While there is significant room for improvement with our minimum viable product, there are also a number of spin-off applications that could find success in the marketplace and will be investigated in the future. Consider the following:

* **ForgetMeNot**
  + Provide users with the ability to mark on map where they left items
  + Allow users to verify that they visited a given location
    - Package delivery
    - Work or school-sponsored trips
* **CacheTracker**
  + Geo-Caching is becoming extremely popular, particularly in urban areas
    - Users must place physical items and provide clues presently
      * This app could spawn virtual items instantly with no work
      * Makes it impossible to compete with users over long distances
  + Users could manually add items, even if they are not located in the area
  + Difficulty is creating algorithm to create riddles based on item location