1 Element Descriptions

1.0.1 Architecture Overview

It was apparent from the briefing and requirements that the most suitable architecture for the system would be a client-server model. There will be a central server that stores persistent system related data, and performs the majority of the application logic. This server services requests from many client applications that handle user interaction and present information through a graphical user interface. A requirement of the application was also to provide a website as a second source of user interaction, which will be implemented using the central server.

The server and client may be conceptually divided into several modules. For the server, there is the *Database Module* that manages reading from and modifying the persistent application data, the *Logic Module* that handles game and system calculations, and the *Web Interface* group which contains the *API Module* for interacting with client applications, and the *Website Module* for servicing the website HTTP requests. The client will be constructed to include a *Request Module* for contacting the server and interpreting any responses, a *Logic Module* that is aware of relevant aspects of the game state and performs client-relevant calculations, a *Geolocation Module* for interaction with the Google APIs and the GPS sensor, and a *Window Module* for the graphical user interface.

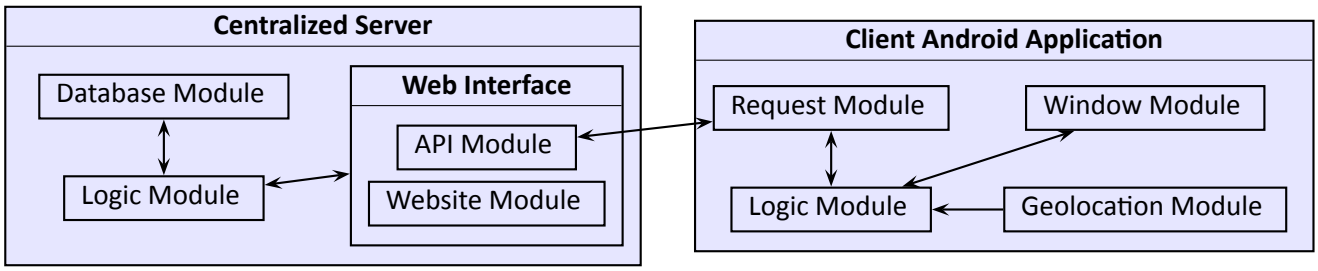

Figure 1: Data flow of the top level of the system, showing the application and server as clearly separated entities with internal modules that encapsulate distinct functionalities of the system.

As *Figure 1* shows, the server and client have a similar overall structure. Both have a main contained logic processing module, a modifiable data source (the *Database Module* for the server and the *Request Module* for the client), and an interface (the *API Module* and *Website Module* for the server, and *Window Module* for the client). This compartmentalisation of processes is intended to make both the server and client subsystems more expandable and maintainable while features are being implemented and testing performed. The interaction between modules is restricted to a small and manageable set of interfaces to help reduce the internal complexity of the system.

1.0.2 Server Module Descriptions

Database Module This module will abstract away interaction with the DMS (Database Management System), making it possible to easily replace the DMS used to support different platforms. For the Windows operating system the server will be running on top of the .NET CLR (Common Language Runtime), and will therefore have access to Microsoft's SQL Compact Edition Server. On Unix systems, the server will be using the Mono CLR implementation, and will therefore have to use an alternative DMS such as SQLite. As well as supporting different DMS connections and SQL dialects depending on the host system, the Database Module will also provide a simple interface to common SQL operations through the use of code generation. This will reduce the amount of errors produced through the use of poorly constructed SQL statements

by delegating the validation task to the debugging facilities provided by the IDE (Integrated Development Environment) used while developing the server.

Logic Module The Logic Module will contain all critical algorithms related to game and overall system state. This will include the operations of account authentication, client location validation, cache attacking, and unit transactions. The module contains a set of core classes that abstract components of the game such as caches and players. Keeping the more intensive calculations in the system separated from the database and interface means they are easier to locate and profile, and unit tests can be produced more simply without having to unnecessarily incorporate unrelated components.

Web Interface This module group is the interface in which incoming HTTP requests are processed and responded to. The requests come in two main categories; website resource requests and API requests. The website resource requests are from users browsing the website and requesting pages or static content such as images. These requests are directed towards the *Website Module*, which processes them and responds with the requested content. API requests are sent by the client application, and will be routed to the *API Module*. These requests are in the form of a command with a series of named parameters, and the response will be sent as a JSON (JavaScript Object Notation) object.

Website Module This submodule of the *Web Interface* handles the delivery of web content such as HTML pages, style sheets, and images. The content is served from a resource directory local to the server program, which is updated during the program's runtime. It also allows dynamic web page generation through the use of inline C♯ scripts in the source HTML files. These scripts are used as a preprocessing language while constructing the page, so the resulting file sent to a user's web browser has content reflecting the current state of the game and the actions of the user. The inline preprocessor scripting system will be implemented specifically for this project, using code generation and .NET's ``Compiler as a service'' facility. The choice to do this instead of simply using PHP or any other pre-existing CGI is so the scripts would have direct access to the server program and its components (this would be true because the scripts are actually contained within the server program itself). This would provide increased performance and would eliminate the need to produce an interface between the CGI and the server.

API Module The second submodule of the *Web Interface* group handles the interpretation and responses to requests from the client application. These requests arrive in the form of a specific command, and the named parameters required by that command. The module contains a class for each command, all extending a general *Request* superclass that provides some facilities common to most requests such as authenticating the requesting user. After the request is processed, an object extending the *Response* superclass is returned by the processing request class. This object is then serialized automatically into a JSON object, which is sent as a reply to the requesting client. The separation of request and response types into classes has the standard benefits of reduced code dependencies and therefore complexity, and also allows requests (or responses) with similar functionalities to extend general abstract classes that implement them. JSON is used as a response format due to its small bandwidth footprint and how trivial it is to parse. It also opens up the possibility of using it with AJAX for the website implementation.

1.0.3 Application Module Descriptions

Request Module This module handles the construction and sending of requests to the central server, and then parses and processes the responses given. Each command has a distinct class encapsulating the

parameters required and the action to perform with the response.

Logic Module Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Geolocation Module Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Window Module Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.