Vi-Char Game
Web Server Development

Scott Dunham, Michael Lim, Eli Mallon, Kimberlee Redman-Garner

I. Introduction

The following document details the requirements of the web server portion of Vi-Char's developing game. The web server team consists of Scott Dunham, Michael Lim, Eli Mallon, Kimberlee Redman-Garner, and Logan Jones. Michael has experience with Java, Python, C++, MySQL, Javascript languages, HTML, CSS, Internet protocol suite (TCP/IP stack), socket programming. Personal skills include long-term and short-term planning, setting and defining achievable, actionable goals. Scott has experience with several programming languages and standards, including Java, C++, and OpenGL and has worked as a freelance web designer and developer, requiring the use of HTML, CSS, JavaScript, jQuery, and PHP, and REST API web services. He also has knowledge of various database management systems, including MySQL and Oracle. Logan has experience with Java, HTML, CSS, and has worked with Microsoft Excel and MATLAB in research settings. Kim has experience with Java, Python, Javascript and web design using HTML and CSS on an Apache web server. Additionally, she has done extensive programming for Microsoft Excel and R (CRAN Project) for analysis of biological data sets. Eli has experience with web programming, mostly with Python or PHP. He has developed a Django application for class registration that was used at the Victory Briefs Institute, the largest debate camp in the country. Additionally, he has worked with MongoDB.

II. Executive Summary

The Vi-Char Game project involves using a video game console, mobile devices, augmented reality, and the web to create a platform that involves many players. The web server development team will work to create communication between the game engine and mobile devices. Data will need to be passed efficiently to create an effective gaming experience.

Additionally, in order to incorporate a larger player base, social media will be integrated to allow interaction from players outside of the general gaming area. Leader-boards, player profiles, and previous game footage will be available online.

III. Application Context

Our product will serve two main purposes. First, we will act as an intermediary between other divisions, providing the necessary communications for interfacing between the game engine and the mobile devices. This function will be used frequently while the game is played to relay all the necessary information. At the most basic level, we will be providing a protocol and accompanying transport service. By defining a standard format for messages, we can effectively and efficiently pass necessary data from various devices without the need for a physical connection. In this capacity, we will essentially be acting like a post office, receiving messages, checking the format is correct, and routing them to the correct device. Depending on the requirements of other divisions, we may be able to inspect the payload of messages in order to error check or even reformat them for the recipient.

Second, we will allow others to interact or view the game through an online presence. This function will allow those without compatible mobile devices, and even those not in close physical proximity, to participate in the game. Our web server will be able to provide a place

where interested people can come to be involved in the project, even if they aren't in the class. By keeping track of high scores, previous game footage, and other data, we will be able to offer a media-rich experience to visitors. We also hope to offer some way for third parties to interact with an ongoing game. While the specifics are still up in the air, one likely option would be utilizing social media (such as Twitter) to add things like power ups to the game. In this way, spectators will be able to directly influence the game, even without specialized hardware or having to be present while the game is played. Another possibility would be to stream a live game to our website, allowing people to spectate in real time.

Our main set of users is directly related to the project, other divisions of Vi-Char. Primarily, we will be dealing with the machine running the game engine, and the mobile devices. Depending on the implementation of other groups, we may be interfacing with the Kinect from the motion capture group, though this seems unlikely. Because these users are directly involved, and will be a part of the design steps, we should be able to tailor our services to them fairly well. Our other users will be spectators, potentially random people who have taken an interest in the project. Because these users will have little to know prior knowledge of our system, their experience will be harder to create. Extensive testing and possible focus groups should help to iron out wrinkles in functionality and user interface.

IV. Functional Requirements

We've established that there are basically two projects under the purview of one project– the "Messenger" and the "Web Interface." Their functional requirements are almost entirely disjoint. In all likelihood, the Web Interface itself will just be another client that connects to the Messenger along with everything else. As such, the specific requirements of each service will be addressed separately.

Messenger – Game initialization
Actors: Android Phones, Game Engine
Scenario:

1. Game Engine connects to the server and requests a new game instance and gives parameters for how many mobile devices it will allow to connect.
2. Server receives request and creates a place for the different users to join the game.
3. Server responds to the Game Engine that the game instance has been made.
4. Mobile devices join the game instance.
5. Server responds to the Mobile devices that they have joined the room.
6. Once the parameters of the game engine have been met, the game begins.

Preconditions: The server is up and running, able to send and receive communication from the various devices. The users have a method of how they are able to connect to the server.
Post-conditions: A game has been created and started for all connected parties.

Messenger – Player Data Updates
Actors: Android Phones, Game Engine, Web Interface
Scenario:

1. A message containing information which needs to be conveyed from one actor to one or more of the others is sent to the web server.
2. The web server receives the message and updates a central record of the current "game state," which can be retrieved by players' devices at any time via the "Game State Polling" protocol (see below).

Preconditions: There must be a clearly defined format messages will take from device to device. This is necessarily vague because the exact nature of the game is not yet fully fleshed out, but minimally a few fields will be required with each message.

- "Source" – Where does this message originate? There will be server-side checking to make sure that sources aren't spoofed.
- "Destination" – Where is this message headed? Most frequently these will be either to the Game Server or to an Android device. "ALL" will be an acceptable destination for messages that need to go everywhere–"ROUND OVER" comes to mind. A message is sent to a particular destination, which could include server, android device, or all devices.
- "Action" – To facilitate communication between the different actors, actors will define a set of actions. They send us data and we will know where to send it from there.

Post-conditions: The game state data is updated with the relevant new information.
Alternatives:

- There should be enough information provided to the client that if messages are delayed for a small amount of time it can interpolate what is likely going on in the game and provide that information to the player.
- If an actor submits malformed data, an appropriate error message will be sent back or caught.

Messenger – Game State Polling
Actors: Android Phones, Game Engine, Web Interface
Scenario:

1. The actors will periodically poll the web server for a record of the current game state (i.e. position of characters, projectiles, etc…).
2. This data is used to update the players' view of the world.

Preconditions: A connection must have been established between us and the client. The client should be querying frequently about the state of the game on a regular or synchronized basis.
Post-conditions: Any changes for the client or other information about the state of the game will be conveyed.

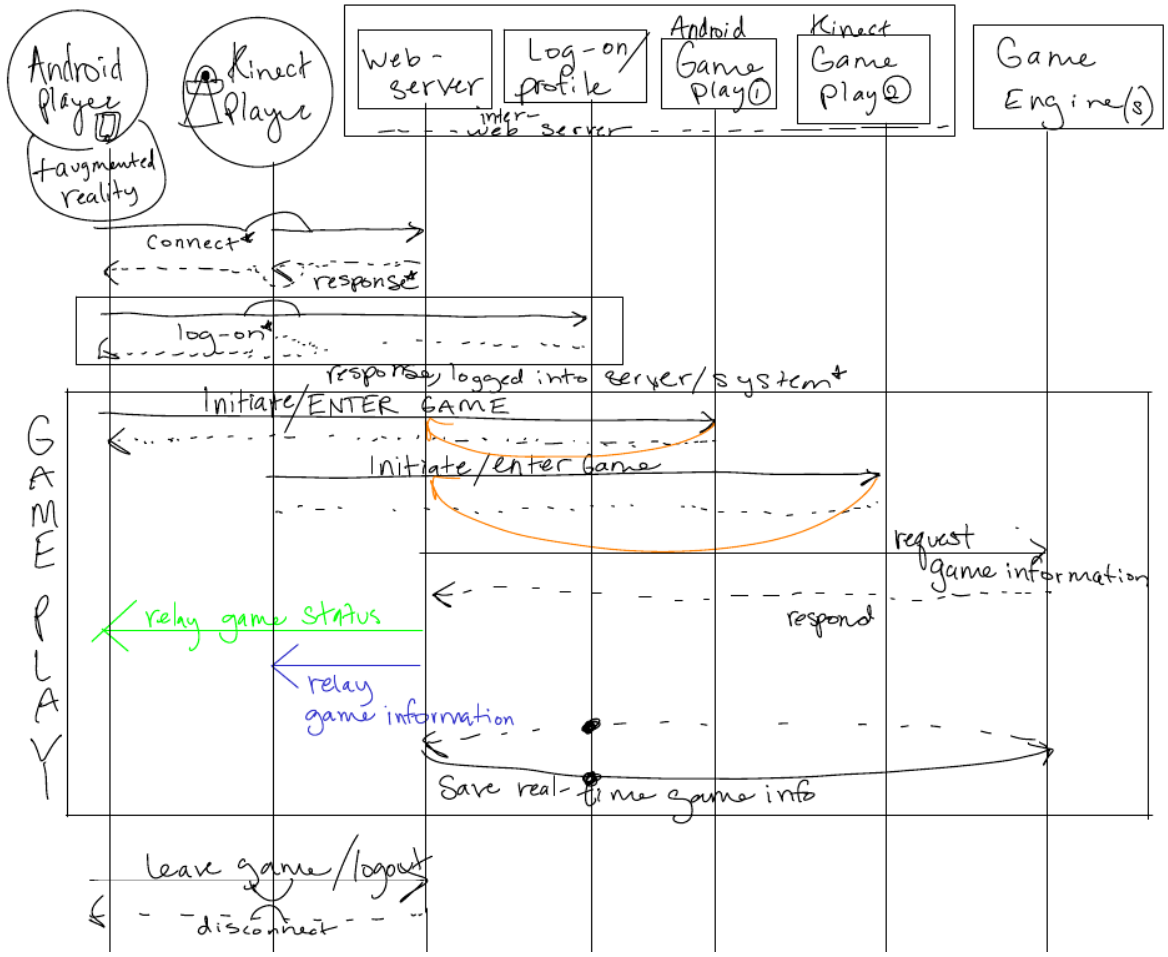
Messenger – User system
This can be as straightforward as having each player enter their name when they connect to the system, but it's important to keep track of which users are playing for leaderboard purposes.
Actors: Android Phone(s), Motion Capture team via Game Engine
Scenario:

1. Before a game starts, the players are instructed to enter a name with which they can be identified.

2. Resulting scores are saved and associated with this name for use in the leaderboard.

Preconditions: The connection between the devices is established and a game instance has been made.

Post-conditions: The webserver keeps track of the users' scores to maintain a user leaderboard.



1. System Sequence Diagram of the basic structure of the message-relay system which is controlled by the Server. There are 4-sub server functions to take note of: the server itself, game initiation for all devices and at least two instances to allow game play to work from the structure of the game. Many more players may be added on from here as far as the server is concerned but this is the bare minimum to reach the game concept's parameters for a full, playable game.

Web Interface – Current Game Status

Actors: Users who want to monitor the status of the current game.

Scenario:

1. When a game begins, the web interface will be populated with the names of each of the players and their current score.
2. As the game is played and the players' scores change, this information is updated (asynchronously) on the web interface.

Preconditions: A game needs to be in progress, and users need a web browser to access the interface.

Post-conditions: The current score for each player in the game is made visible.

Web Interface – Social Media Integration

Actors: Anyone that wants to interact with the game via social media, including non-players.
Scenario:

1. Users monitoring the score of the current game via the web interface are preiodically presented with the option to tweet one of several hashtags, each of which correspond to a different consequence in the game (i.e. assist the player, or hinder them)
2. After a predetermined amount of time has passed the resulting tweets are counted, and the hashtag with the most tweets wins.
3. The associated action is sent to the game engine to be carried out.

Preconditions: Users should have a web browser, and a Twitter account which can post public messages. The game will be constantly querying social networks to see if any relevant messages have been passed.
Post-conditions: The server will send a message to the game engine to carry out the appropriate action.

Web Interface – Leaderboard

Actors: Users who want to know the users with the highest scores in previously-played games
Scenario:

1. Users navigate to the web interface, and are presented with a list of the player names and scores for the players who earned the highest scores.

Preconditions: At least one game has been played so that the list can be populated, and users have a web browser to access the web interface.
Post-conditions: Users know who scored the highest in our game.

V. Nonfunctional Requirements

1. Security requirements

The components created by the Web Server Development team must be secured from unauthorized use. Since this part of the system will be responsible for routing data between the other project components, it is imperative that only authorized users and devices be allowed to interact with the web server. Failure to secure communications between project components could result in disruption to gameplay, or potential system failure.

2. Performance requirements

The web server must be able to quickly communicate data to and from a multitude of concurrent user interface devices. At any given time, a game can have users participating via the connect, mobile devices, and the web interface. Being able to quickly route relevant game data between these users will be essential ensuring smooth gameplay.

3. Supportability requirements

The message protocol developed by the Web Server Development team must be flexible enough to adapt to the changing needs of the other development groups. As game development progresses, the format, type, and frequency of data which needs to be

communicated between the various components is likely to change. Since we cannot wait to begin development until after the other groups' communication requirements are better defined, the work we begin now will need to adapt along with the rest of the components.

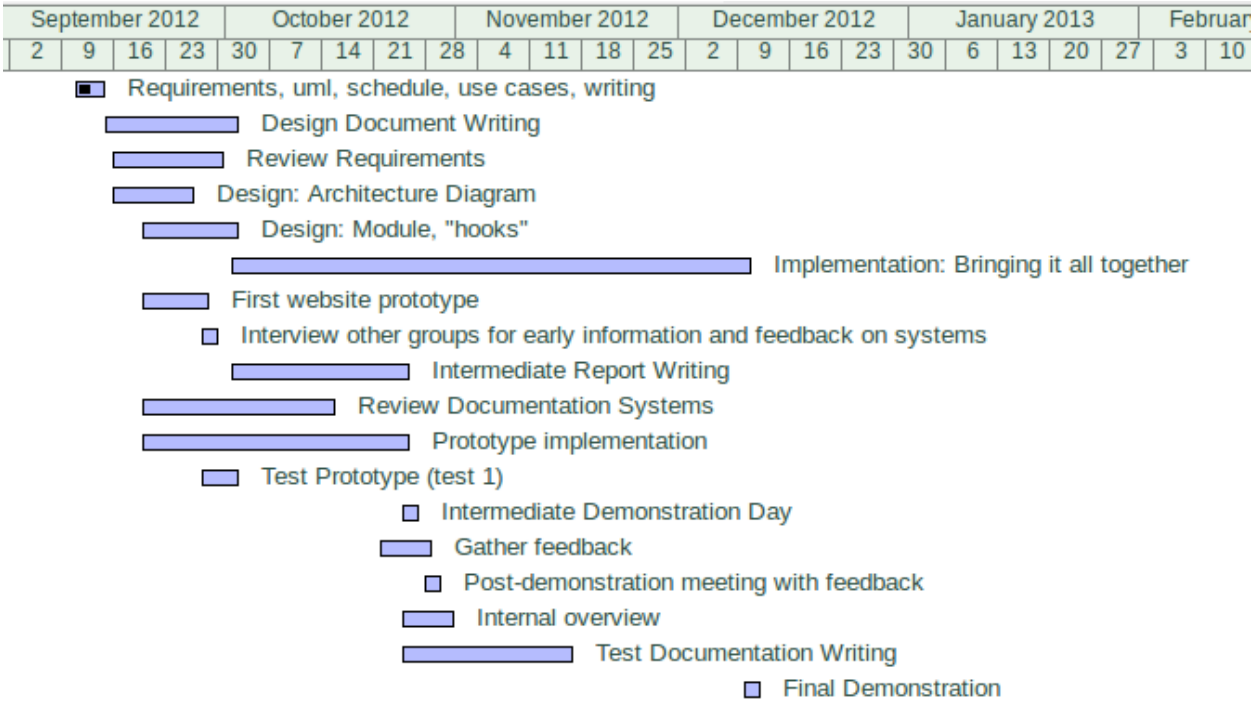
4. Documentation requirements

The messaging API developed by the Web Server Development team must be well documented and easily accessible. Since this API will be heavily-used by the other development groups, they will need to have a clear understanding of how to properly utilize it. Having easily accessible documentation will allow the other groups to incorporate our API into their components of the project.

5. Miscellaneous requirements

The web server must run on the provided Ubuntu server, and at least be accessible from campus. It would be ideal if the server was accessible from off campus as well, however the university's firewall may make this challenging.

VI. Timeline



VII. Potential Challenges

Game messages need to be passed fast enough to facilitate responsive gameplay. HTTP is an asynchronous protocol-- requires a little thinking to get it to pass realtime game messages. These are both issues we will work through as we code. Most people on the team don't speak the same programming languages, however, the team will learn from each other. Our timeline will change throughout the semester, however, as long as we continually update our requirements and update team deadlines this will not be an issue. Communication between

other groups may lead to confusion so it will be important to keep up on documentation and API. Class meetings will help keep everyone on the same page.

VIII. Glossary & References

Representational State Transfer (REST) API - A type of web service in which clients initiate requests to remote servers via the HTTP protocol, and the server returns the suitable response.

UML Sequence Diagrams. http://www.agilemodeling.com/artifacts/sequenceDiagram.htm.