*Department of Computer Science & Engineering*

*University Of Nevada-Reno*

Project Glimmer

Team 15 - MAJC

**Mike DesRoches**

**Christian Garcia**

**Jake Lahr**

**Alexandra Pasinski**

Instructors

**Dr. David Feil-Seifer**

**Devrin Lee**

External Advisor

**Dr. Engin Arslan**

December 12, 2019

**Table of Contents**

Abstract …………………………….………………….……………………………………..….2

Introduction.………………………………….……………………………………………….... 2

Prototype Objectives & Functionality……….……………………………………….………..3

Develop Prototype..……...………………...………………………………………….………..4

Evaluate Prototype....…...…..………………..……………………………………….………..7

Demo Prototype….……....……………………………………………………………………...8

Changes Needed to Software Design…………...……………………………………..…….8

Contributions of team members……………………………………..………………..……….9

**2. Abstract**

Project Glimmer aims to create a software API that game developers can use to process comments entered in a chat log on a given web video streaming service (such as Twitch.tv). The intended use of this API is to allow for game developers to design commands that change the state of their games and which can be called by the viewers in a stream chat, giving games a more interactive experience for the viewers. The purpose of this document is to give an overview of the initial prototype for Project Glimmer that the team developed and the user feedback meetings that the team demoed the prototype in.

**3. Introduction**

Streaming services, such as Twitch, have become an important part of the video game industry. With millions of people watching video game streams every day, game developers are realizing the potential for marketing their games on these streaming platforms. Project Glimmer’s objective is to create an application programming interface (API) that game developers can utilize to create audience interaction between players live streaming their games and viewers watching those lives streams. Streamers would also be able to use this API to modify their favorite games to allow audience interaction in games that do not already have this interactive feature. Live stream viewers would be able to interact with game streamers in a real-time fashion and have an impact on the current game state.

Our objective is that, by integrating our API into a game, the life-span of a video game can be increased by the increase in possible interactions from external sources such as live stream viewers. Our main software design goal is to create an API that game developers can easily integrate with their existing software, whether they are using their own game engines or utilizing an existing engine, such as Unreal or Unity. Our product will have a unique significance in the video game development industry as, while live stream chat integration in video games is not a completely new concept, most solutions need to be developed in house and few resources exist to allow independent developers the ability to easily achieve this type of interaction in their games.

The team has implemented a prototype of the Project Glimmer API and demoed it by integrating it into a basic graphics application. The prototype currently implements basic functionality such as developer specified commands and command statistics. We were unable to resolve networking issues with connecting to live streaming services so we opted to instead simulate a live stream chat window in the console for the prototype. We have decided to remove the Spark Streaming API from our software design as we believe it complicates the process of designing the software while producing very few benefits to our API in terms of connecting to live streaming services.

**4. Prototype Objectives & Functionality**

The team has categorized Project Glimmer into three main parts: command handling, network or streaming service handling, and the main Glimmer instance that allows the game developer to control the API. The goal of the team’s prototype was to show off the underlying structure of the command handling and control sides of the API. The reason why the team focused on command side was because we were encountering issues with connecting to Twitch, or any streaming platform, and receiving chat commands. To get around this limitation for the prototype, the team created a basic simulation of a chat interface using the terminal. So, for the prototype, the team created a simple OpenGL program that had a triangle that could change colors and a background that was constantly changing colors. The point of this program was to demonstrate that our API could be used to run in parallel to a separate game application and then call functions from that game application when the API encounters commands from the chat interface that the developer specified.

The first functional requirement that we implemented for the prototype was FR2. This requirement specified that we were to parse a chat stream on the Twitch.tv platform for commands. Due to the networking issue that we mentioned earlier, we modified this requirement slightly for the prototype by using a simulated chat interface inside the terminal instead of Twitch’s chat stream.

The second functional requirement that we decided to implement for the prototype was FR6. This requirement specified that we provide tools for users of our API to specify what functions in their application code they wanted us to call and what commands would be used to call them.

The third functional requirement we implemented for the prototype was FR8. Here, we had to implement a way to keep track of which users were entering commands. We did this by creating a map in each command’s statistic instance that mapped a user’s username string to an integer that represented how many times they had called that command in the chat.

The final functional requirement that we implement for the prototype was FR10. The requirement specified that we keep track of what chat commands are entered so that the API could provide feedback to the game developer. We did this by implementing a still somewhat simple statistics framework that logged calls for each command entered and for which users entered them. Then, we provided an interface to get these statistic calculations at runtime.

Another aspect of the prototype, that was not explicitly detailed in our functional requirements, was concurrency. We wanted our API to work properly without needing to block execution of the main application whenever we wanted to do something. This is especially important for video games which run in real-time. Therefore, our API was designed to run on its own thread and call developer specified functions from that thread whenever a command was detected. This is demonstrated in the prototype with the demo game application, the colored triangle, having a background that constantly changed colors. This occurred even while the API was waiting for commands.

**5. Develop Prototype**

UI Snapshots

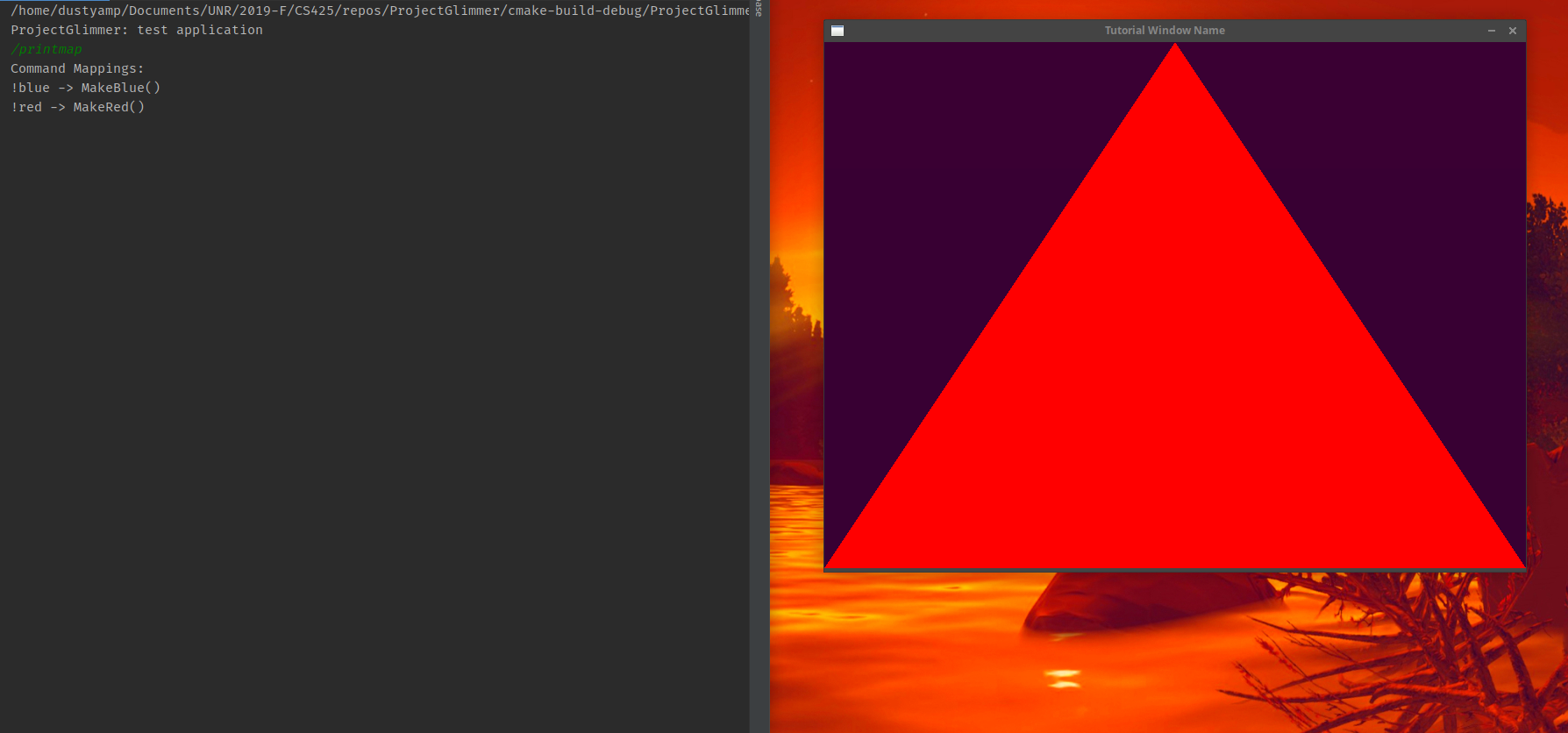
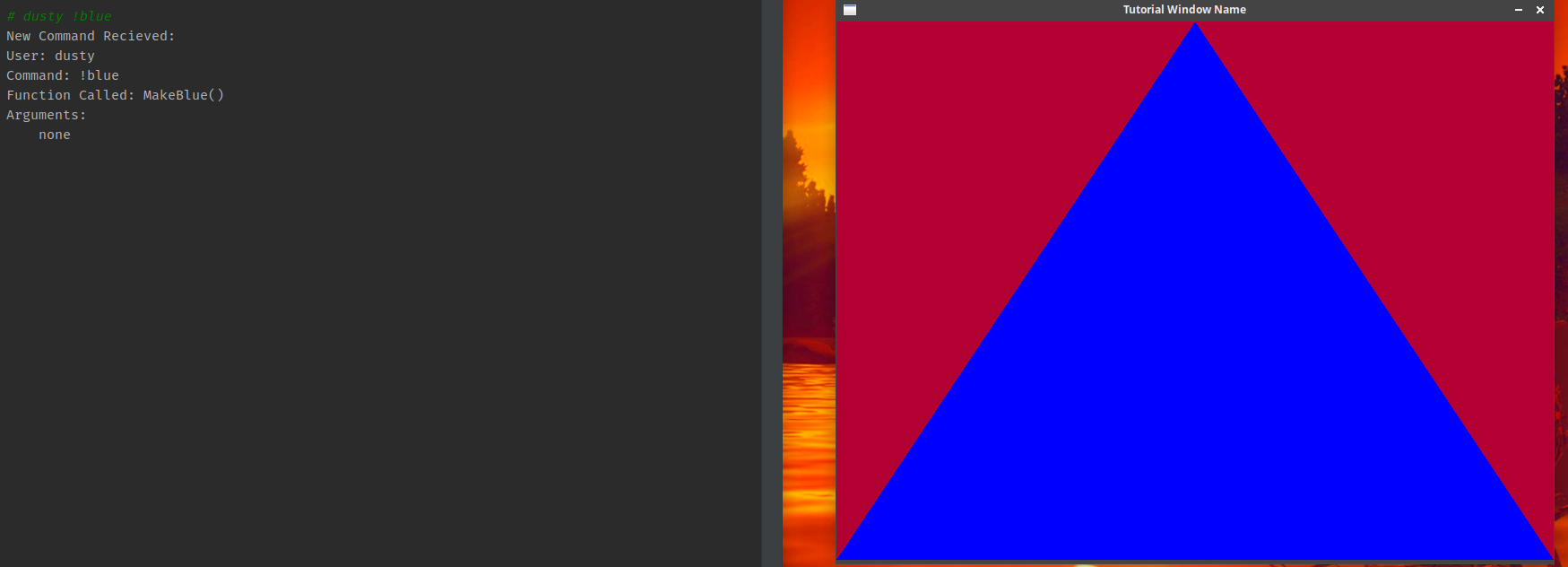
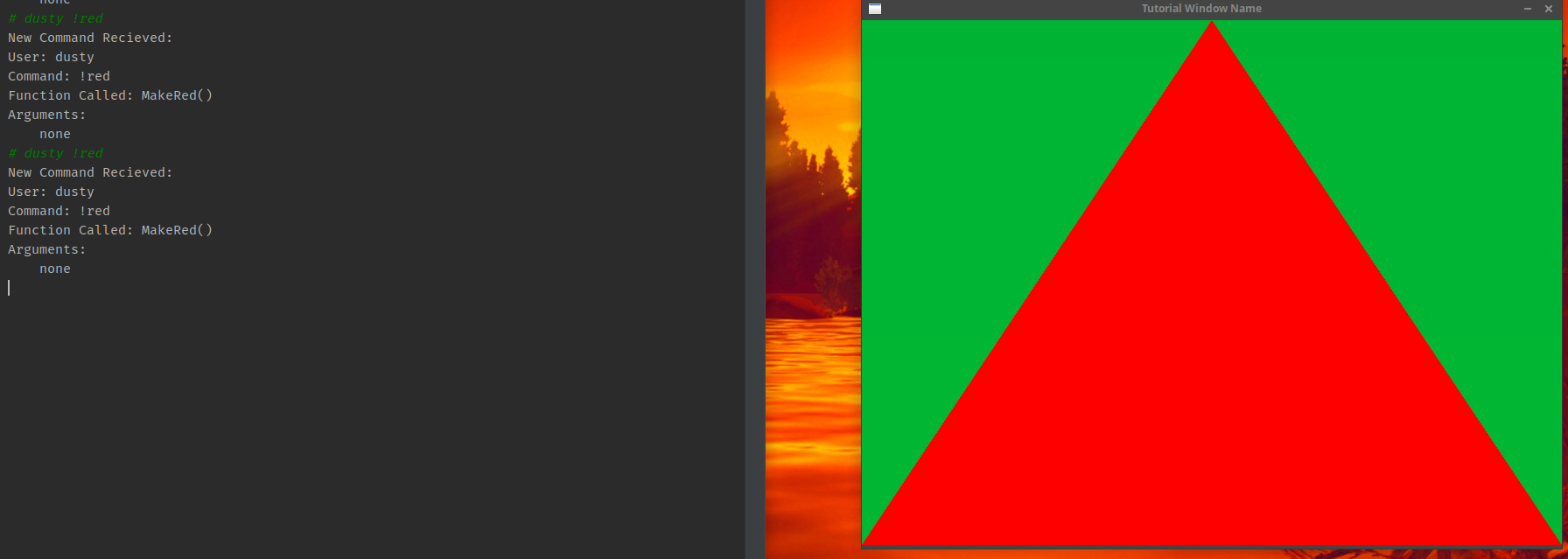


Figure 1: A chat user has entered the Glimmer command ‘/printmap’ which has printed a mapping of commands to game application functions to the chat window. These mappings were defined by the game developer at runtime using the Glimmer API.



(a)

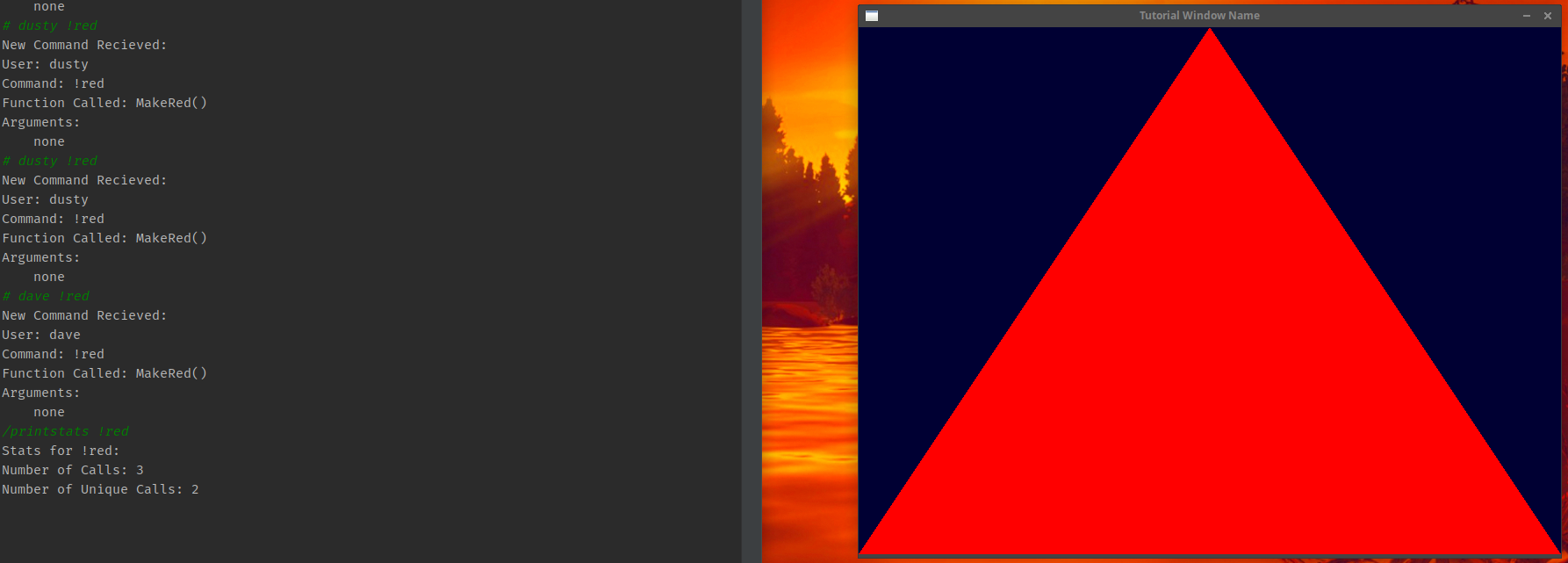


(b)

Figure 2 (a, b): The commands ‘!blue’ and ‘!red’ are executed in response to calls by users in the chat window. A description of each command with accomponing caller and parameter information is displayed after each command is entered. The triangle’s color is updated in real-time when the command is entered. The background of the game continues to change color while the Glimmer API is waiting for commands from the chat window.



(a)

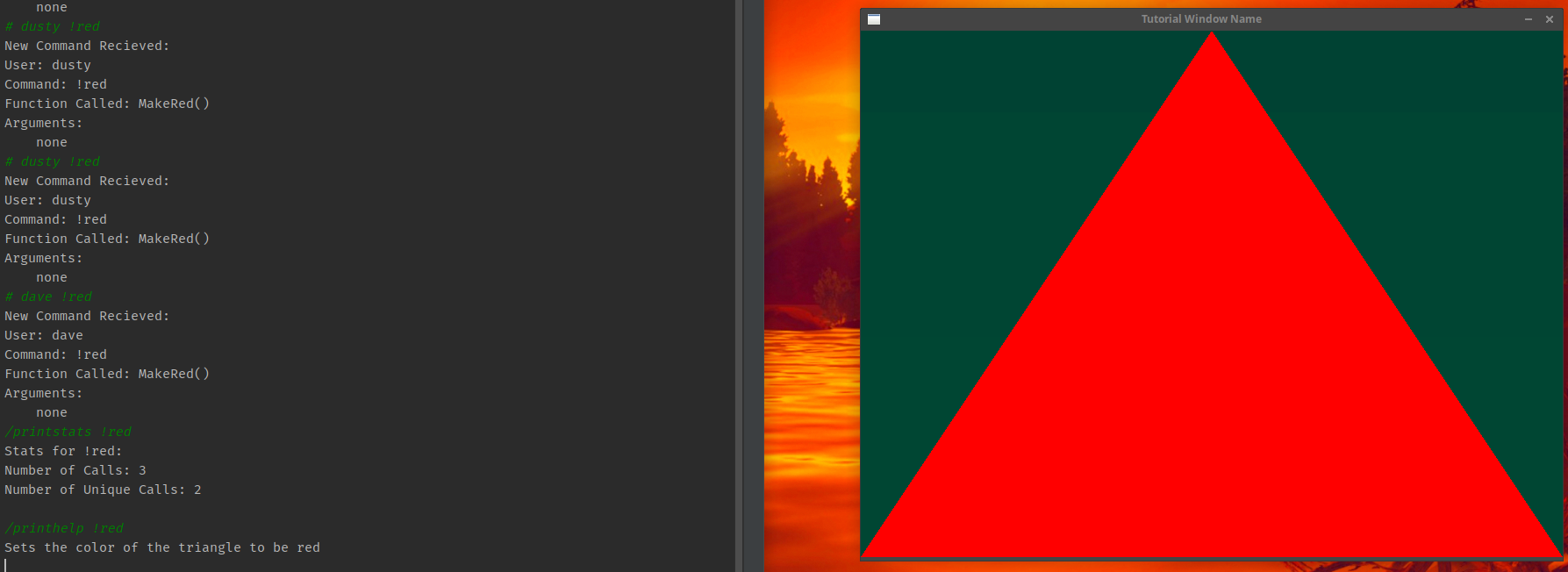


(b)

Figure 3 (a, b): The Glimmer commands ‘/printstats !blue’ and ‘/printstats !red’ are called in the chat window to print the running statistics for each command. Several instances of each command had already been called prior to calling /printstats. The result is a simplified output of the number of calls for a command and the number of unique calls for each command. Each call of a command is tied to a specific chat user in the command’s statistics. This way, we can keep track of per user statistics as well.



(a)



(b)

Figure 4 (a, b): The ‘/printhelp !blue’ and ‘/printhelp !red’ Glimmer commands are called in the chat window. These Glimmer commands will print the help message of whatever command is appended to the end of them. These help messages are defined by the game application at runtime and can be modified later.

**6. Evaluate Prototype**

Aside from the prototype demo meeting with Professor David Feil-Seifer. The team conducted 2 user feedback sessions. One of these sessions was with our advisor, Dr. Engin Arslan, and another was with a potential stakeholder, Dr. Alireza Tavakkoli. In both meetings, the team presented the same prototype demo that was shown to Professor Feil-Seifer. In all three meetings the team received insightful feedback which varied depending on the stakeholder’s field of expertise. The one point of feedback that was the same across all the meetings was to focus on getting network communications going because that is a vital step towards the progression of the project.

In our meeting with Dr. Engin Arslan, he suggested to look further into some of the data science and statistics research that can come out of this project. He gave us some considerations on how to handle the amount of data and commands that our API may encounter while trying to parse the chat streams. He suggested looking into implementing support for a cap on commands and priority queue for certain commands. He desire was for us to focus on the research side of the project. Once the groundwork from the engineering side was done he suggested to focus on the “smart features”, to analyze user data in many ways, and think of unique ways commands can be processed.

In our meeting with Dr. Tavakkoli, he was focused more on the actual display of the project itself and how to really show off what this project can do. Tavakkoli suggested we create a demo that can handle commands with more that zero arguments and show how the API can be used to affect a game that is more interactive than just changing the colors of a triangle. He suggested we design a strategy/RTS game with many units and interactive parts to show off what the limits of our API are and what can it can be programmed to do.

**7. Demo Prototype**

The team gave a demo of the prototype to Professor David Feil-Seifer on Thursday December 5th at 10:30 am. Professor Feil-Seifer told the team that the most important thing for us to work on over the next few weeks was connecting our API to Twitch, as that is the most important groundwork for our project. He said that while the functions that are implemented right now are well done, the API cannot show them off without a practical example using Twitch. He suggested that we use JavaScript to connect to Twitch via HTTP requests because most of the Twitch API guides use JavaScript. Professor Feil-Seifer said that we should also focus on researching avenues for integrating our API as a plug in for the major game engines like Unity and Unreal to make the project available to more developers. One major point that he made was that the team should work on the project before the start of the next semester to make sure that we have all of the level one functional requirements implemented.

**8. Changes Needed to Software Design**

Each stakeholder that the team interview provided their own feedback specific to their field of research. Dr. Tavakkoli recommended that we focus on designing a nicer game application to use as a demo for our API. Dr. Arslan recommended that we look into the potential research aspects of our project in the realm of statistics and feedback for the game developer. Dr. Feil-Seifer wanted us to mainly focus on getting Twitch support working.

The first update that we will make to our design is to drop the Spark Streaming API from it as it did not provide us with much benefit in connecting to Twitch and seemed to create quite a lot of unnecessary overhead for the application.

Secondly, we plan to focus more on getting Twitch integration to work in our API so that we can start moving forward on new features that our stakeholders were interested in. We are also going to look at possible statistics tools that we can utilize to bring richer statistics to our users.

Finally, we are going to try to implement better multi-threading capabilities into our API to expand the types of commands that can be specified and increase the number of game applications that our API can support.

**9. Contribution of Team Members**

Alexandra Pasinski

Hours Worked: 8

Parts Worked On: Developed the Glimmer API and integrated it into a basic graphics application for demoing. Lead the demoing and discussion of the Glimmer API prototype in our user feedback meetings. Took each screenshot for part 5 and wrote the captions for each of them. Contributed to the sections 2, 3, 4, 6, 7, and 8. Proofread and edited the document.

Jake Lahr

Hours Worked: 8

Parts Worked On: Researched Twitch IRC integration, Helped program the triangle graphics application, organized user feedback meetings. Wrote prototype objectives & functionality. Compiled feedback and evaluations. Thought about what software changes we needed to work on for the future.

Mike DesRoches

Hours Worked: 4

Parts Worked On: Contributed to intro. Partook in stakeholder interviews and provided answers when stakeholders asked questions.

Christian Garcia

Hours Worked: 5

Parts Worked On: table of contents, demo prototype section, programmed the triangle graphics application for the demo, proofread and edited.