*Department of Computer Science & Engineering*

*University Of Nevada-Reno*

Project Glimmer

Team 15

**Mike DesRoches**

**Christian Garcia**

**Jake Lahr**

**Alexandra Pasinski**

Instructors

**Dr. David Feil-Seifer**

**Devrin Lee**

External Advisor

**Dr. Engin Arslan**

November 18, 2019

**Table of Contents**

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

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

High Level and Medium Level Design …………………………………………………………...3

Detailed Design …………………………………………………………………………………...6

Initial Hardware Design …...……………………………………………………………………...8

User Interface Design…...………………………………………………………………………...9

Glossary………………………………………………………………………………………….17

Contributions of team members………………………………………………………………….19

**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.

**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.

Project Glimmer has decided to use Spark Streaming for secure processing of the stream chat commands. Using this API, data can be ingested from many sources and can be processed using complex algorithms expressed with high-level functions. Team 15 has also decided to include command statistics to show the streamers how many viewers are interacting, how many commands have been sent, the most used commands, and other statistics that might be useful. The team will also develop a local chat application in order to test the API without the need to connect to a streaming site.

**4. High-Level and Medium-Level Design:**

System-Level Diagram:

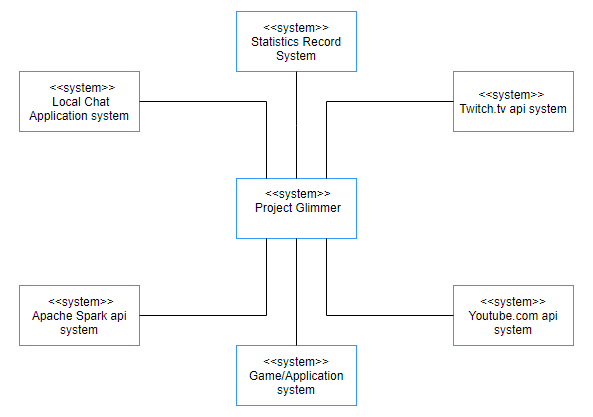


Fig. 1. Context Model

As an overall system diagram, Figure 1 is a context model of the Project Glimmer system that shows the other systems in its environment. Project Glimmer can connect to a local chat system, a Twitch.tv api system,and a Youtube.com api system which are allow Project Glimmer access to read and interpret chat data of that particular system. The Statistics Record system allows Project Glimmer to process data from these chat systems and create statistics that can help the user with meaningful info. The Apache Spark system allows Project Glimmer to process chat data from the systems mentioned earlier and create packets to be processed by the Statistics Record System and the application system which is what the users system will be.

Program Units

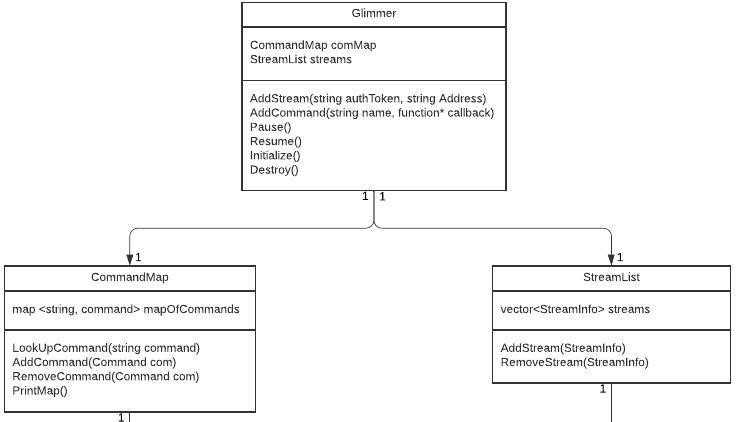


Fig. 2.UML Class Diagram Part 1

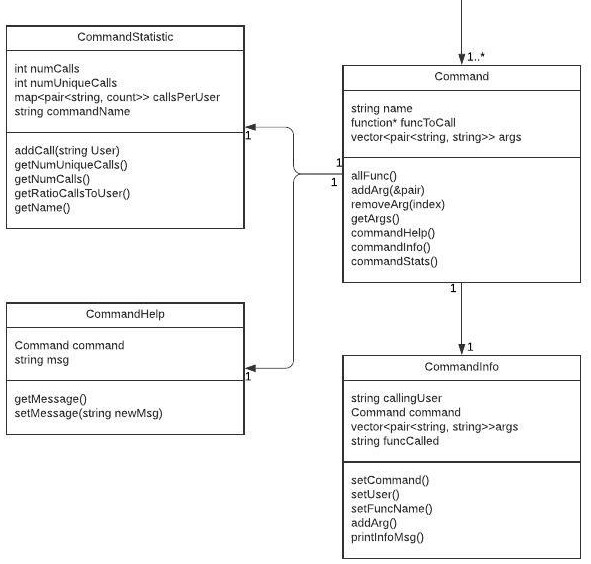


Fig. 3 UML Diagram Part 2 (Connected at the top to CommandMap)

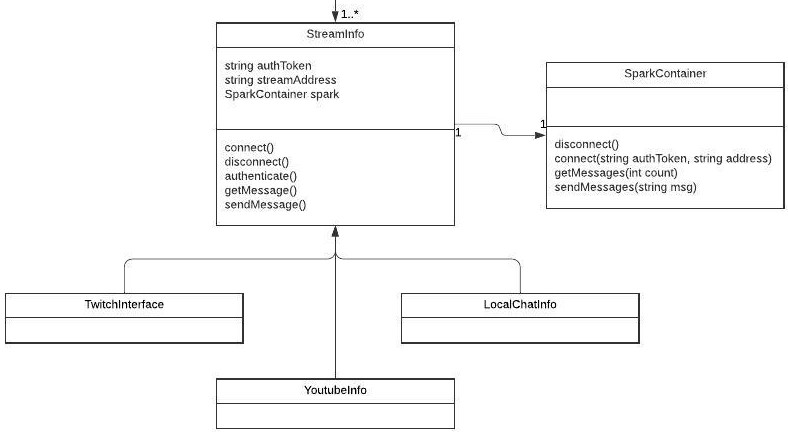


Fig. 4 UML Diagram Part 3 (Connected at the top to StreamList)

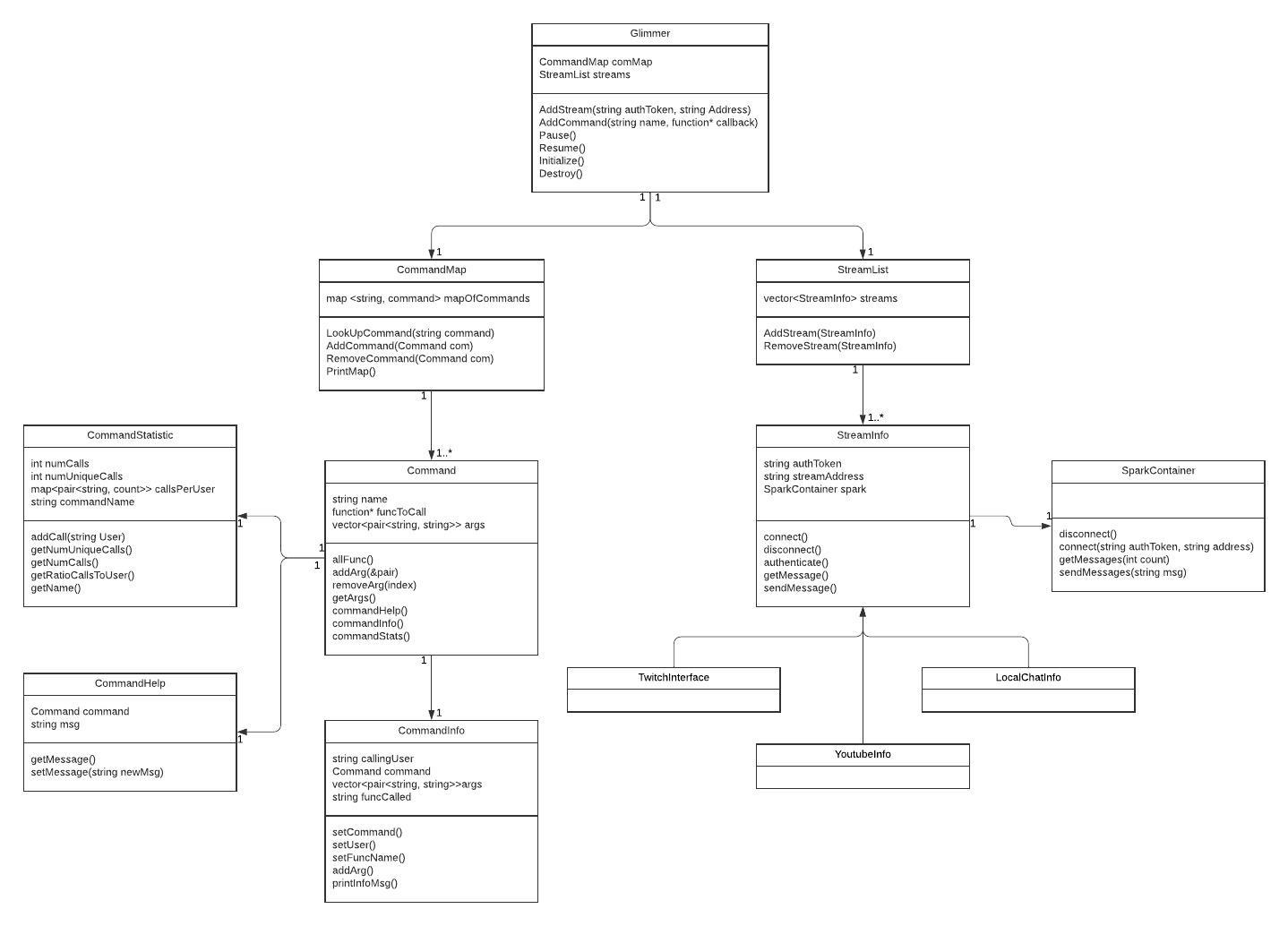


Fig 5. Full UML Class Diagram

Project Glimmer is an object-oriented solution and figure 2-5 presents an overall layout of Project Glimmer in a UML class format.

The solution will have an overall handler class called Glimmer that uses both Streamlist and CommandMap objects (figure 2). CommandMap creates a map of Command functions while StreamList works similar in that it will have a container of multiple instances StreamInfo to call upon.

Command in figure 3 will be the full command class that calls upon CommandStatistic, CommandHelp, and CommandInfo that will complete what a command in the project will be comprised of.

On the right side of the UML in figure 4, StreamInfo is an interface that calls upon SparkContainer which will use Apache spark to process the messages from each of the inherited classes TwitchInterface, YoutubeInfo, and LocalChatInfo (from the StreamInfo interface). This will have all the tools necessary to process live stream chat and channel info from multiple sources.

Main Data Structure:

We will use a message queue for storing the latest commands and messages received from the live stream. The CommandMap class will be used to manage the mapping between message string and Command classes with their callback functions to the game/application.

**5. Detailed Design**

State Diagrams:

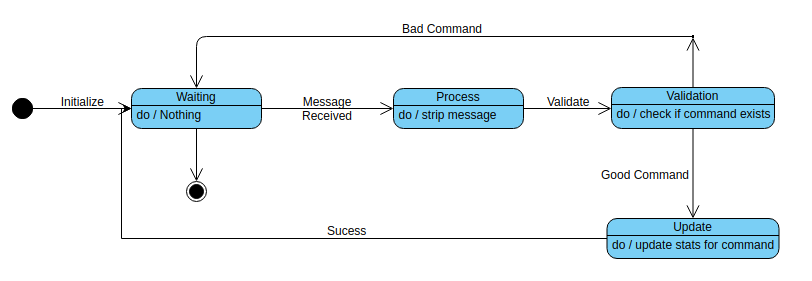
****

Fig. 6. State Diagram of Update Stats

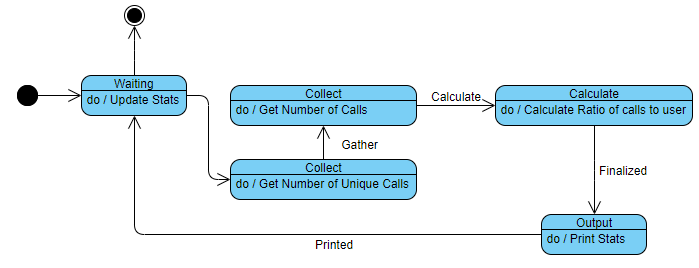


Fig. 7. State Diagram of Print Stats

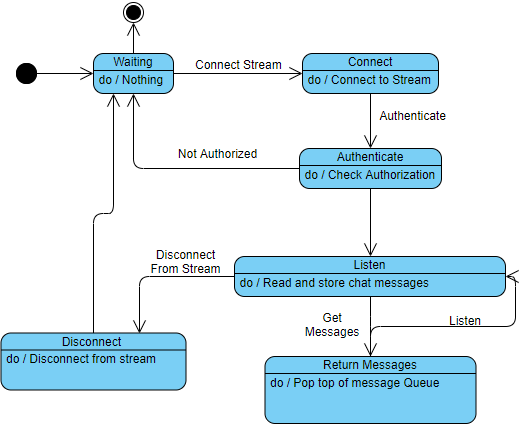


Fig. 8. State Diagram of Stream Layer

Activity Diagram:

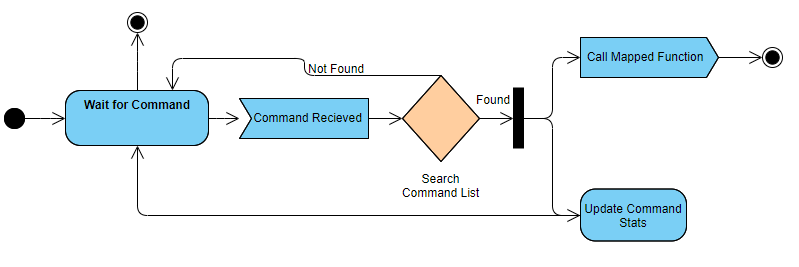


Fig. 9. Activity Diagram for calling a mapped function with a command.

Project Glimmer has four main functionalities. One functionality is figure 6 update stats. Here process will be waiting until a command is called in chat once a command is received it must validate whether it is a functional command that was defined by the user. If it is not a valid command it will trash that command and wait until a new command is called, if it is a valid command Project Glimmer will update the relevant stats for the valid command.

The second functionality demonstrated in figure 7 is the print statistics functionality where like the update stats the process will be waiting for a function call to print the statistics gathered. Once the print call has been placed the function will gather the number of unique calls and number of total calls then calculate the ratio between the total number of calls and total number of users. Then the process will display that data.

The third functionality which is demonstrated in figure 8 is the stream layer functionality. Here Project Glimmer will wait until the process has connected to a valid stream ID. If the stream key is valid the process will start to read messages from the chat stream and store them in a queue. If the getMessage function is called the message on top of the stack will be popped off. This will continue to listen until the stream is disconnected.

The final functionality demonstrated in figure 9 is an activity diagram of the call mapped function. Which will wait for a command and search through the command list defined by the user. If the command is found the process will fork into two directions and both call the mapped function, as well as update the command stats. This is so the system can both track command stats for the session and do the process that is mapped to the called command.

**6. Initial Hardware Design**

We do not have any hardware components for our project.

**7. User Interface Design**

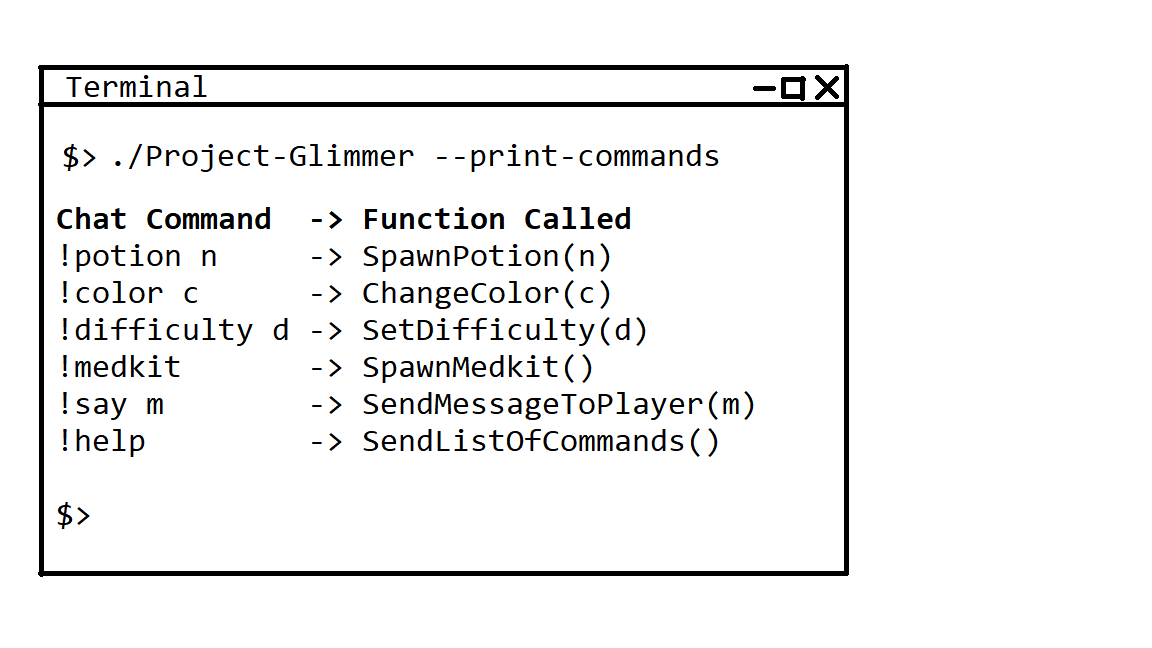


Figure 10. The game developer uses the API to print out the mapping of commands to game functions.

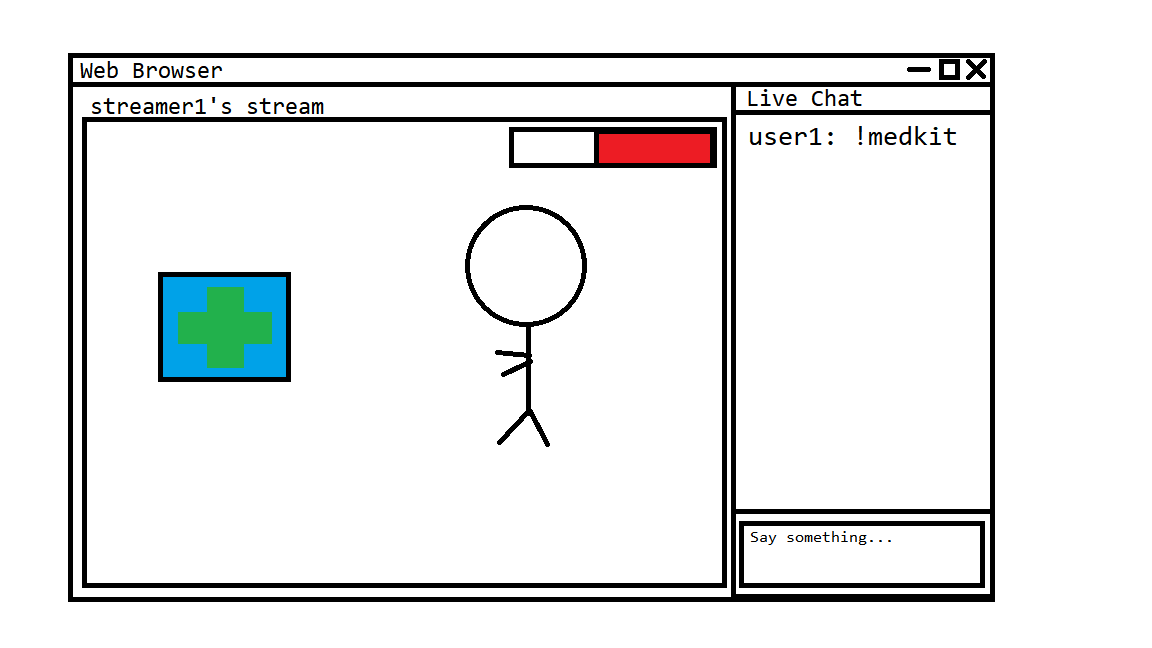


Figure 11. Viewer, user1, enters zero argument command, !medkit, on a live stream service and a medkit is created in the game.

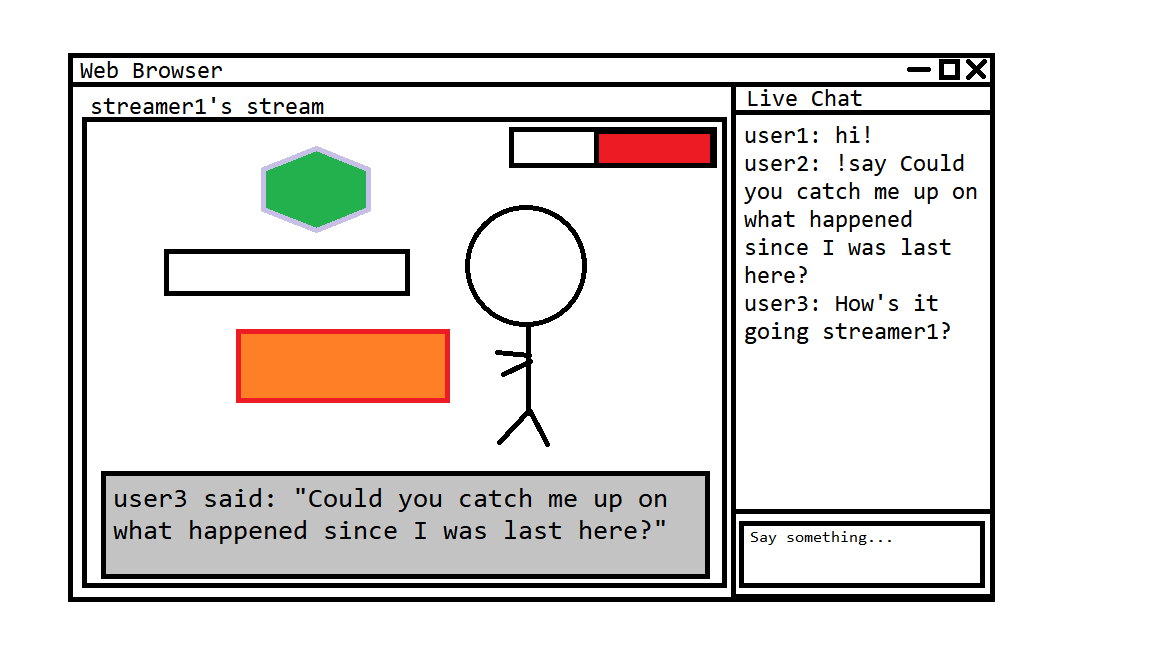


Figure 12. Viewer, user2, enters single argument command, !say m, on a live stream service and the message ‘m’ is displayed in the game.

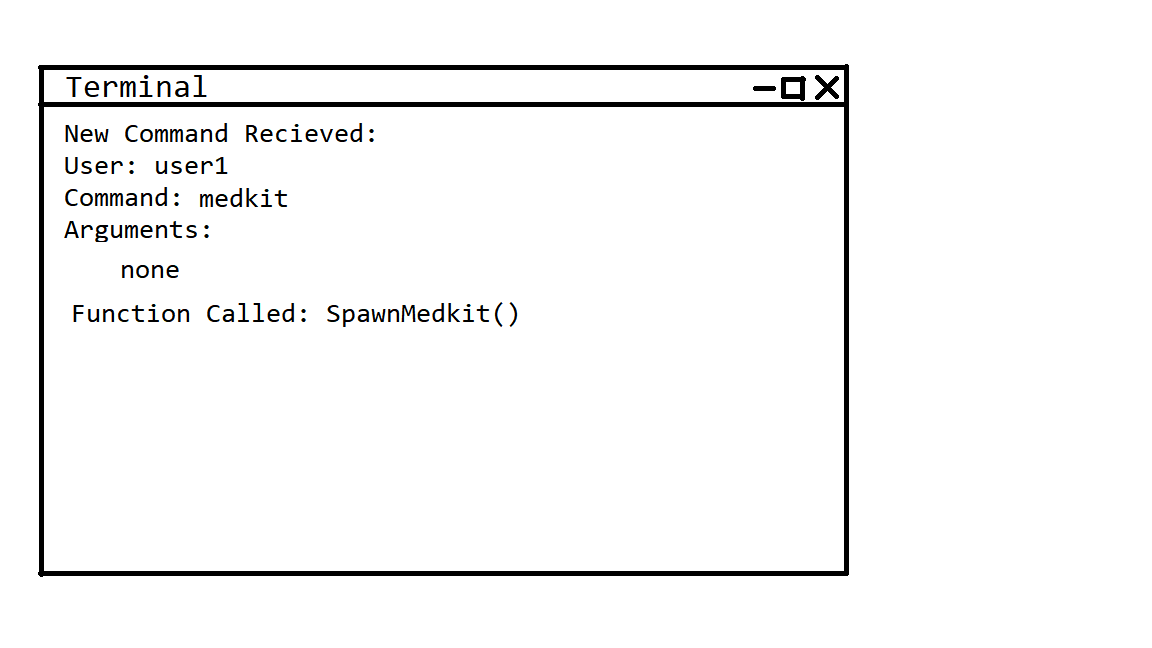


Figure 13. The API prints out the command information, for a zero argument command that was received, to the terminal.

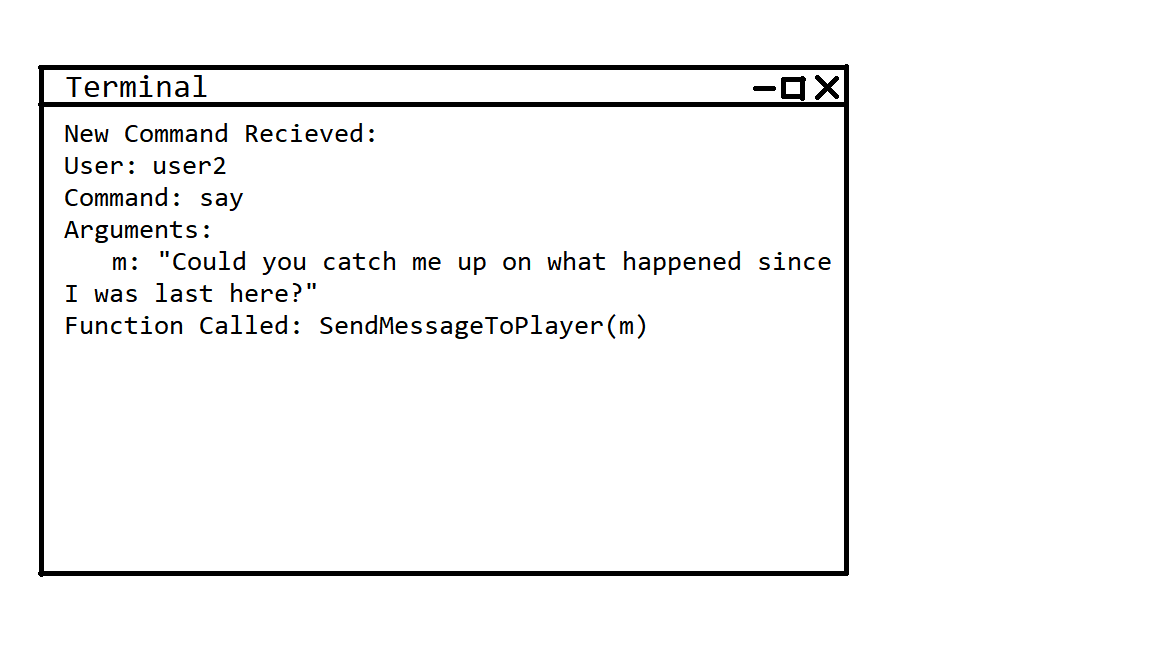


Figure 14. The API prints out the command information, for a single argument command that was received, to the terminal.

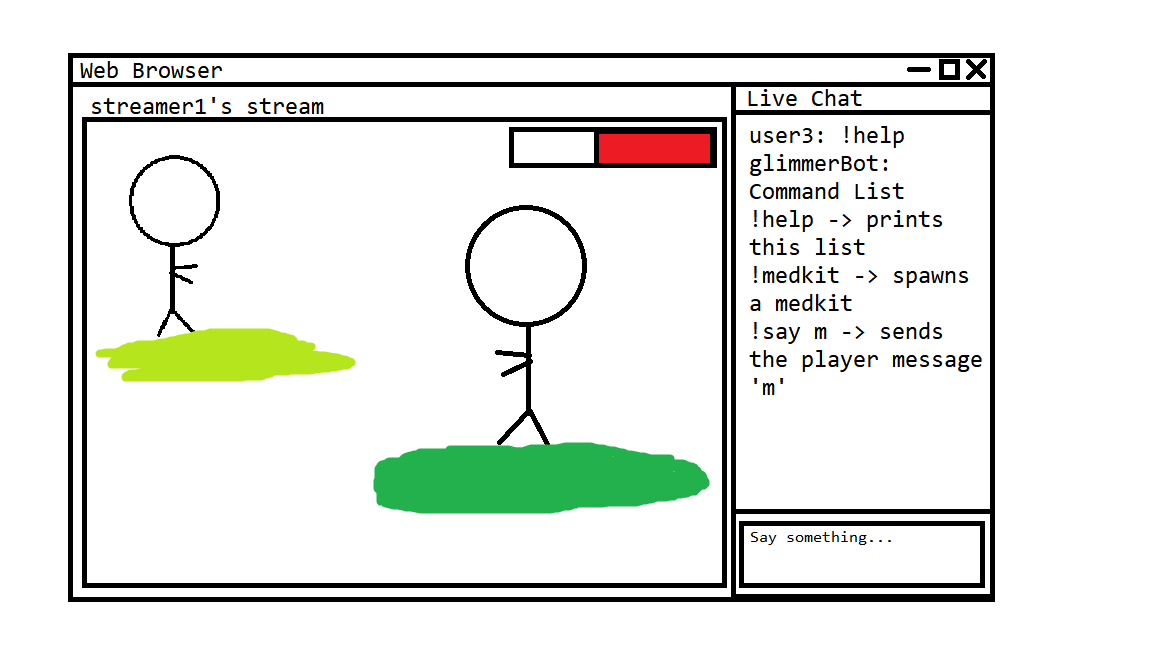


Figure 15. The api responds to the !help command by printing a list of available commands in the live stream chat log.

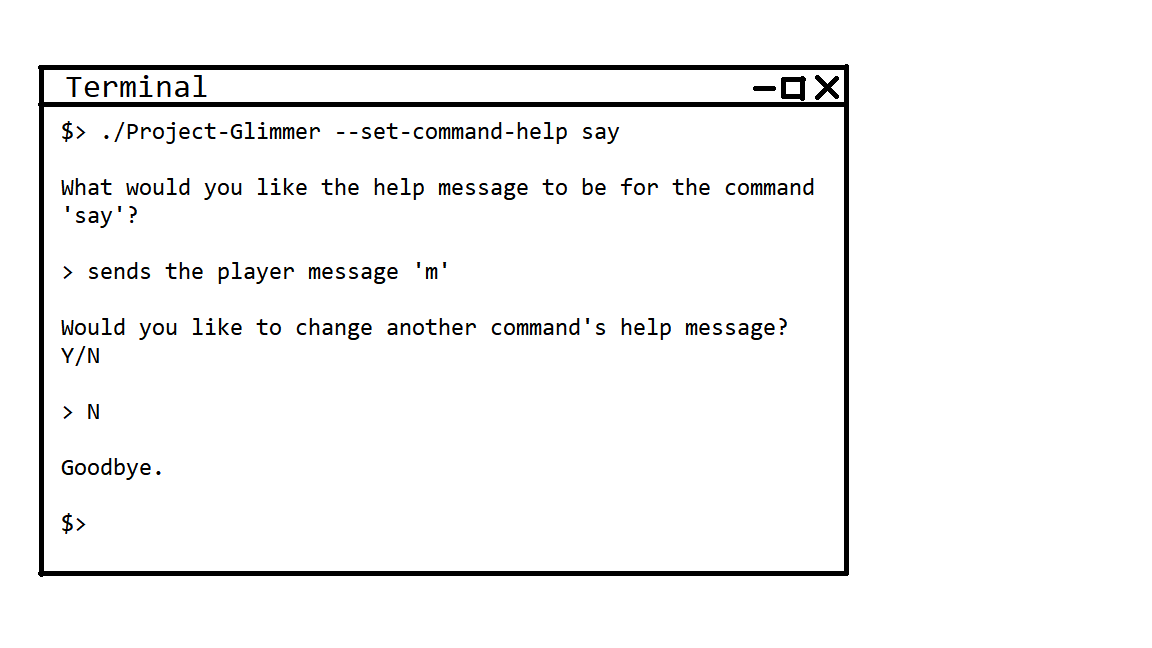


Figure 16. A developer uses the api to set the message that is displayed, for a specific command, when a viewer sends the !help command.

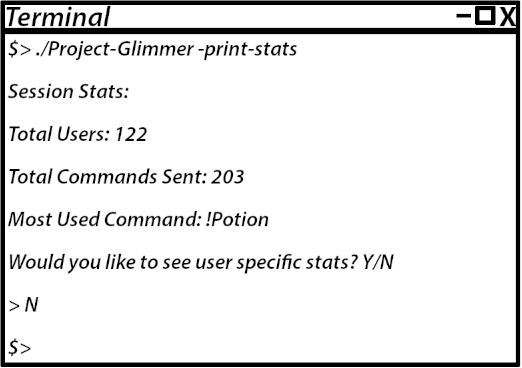


Figure 17. The terminal can print out command statistics as required

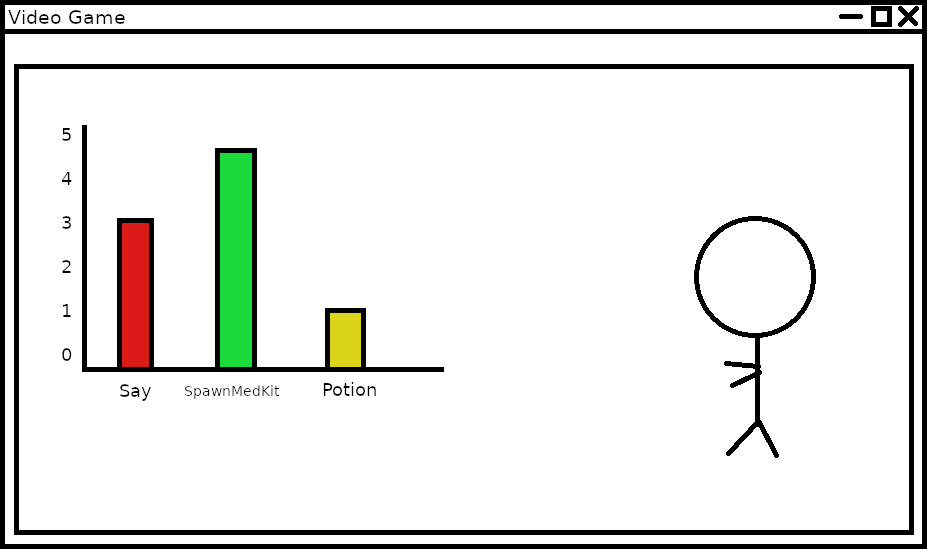


Figure 18. A developer can configure in-game commands to include statistics to print to the game’s screen

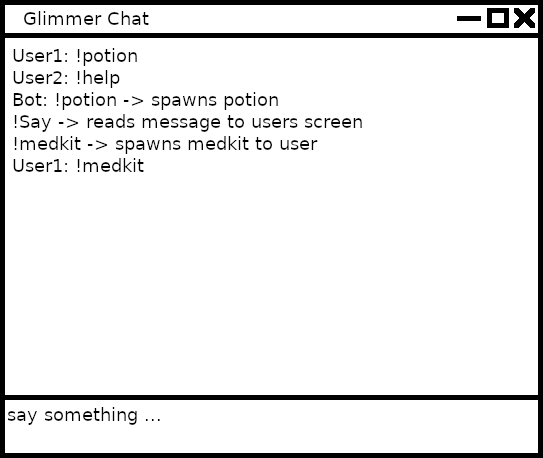


Figure 19. GUI of local chat screen. This chat screen will be used for prototype development so that we do not have to rely on a specific streaming service to implement our api at the start.

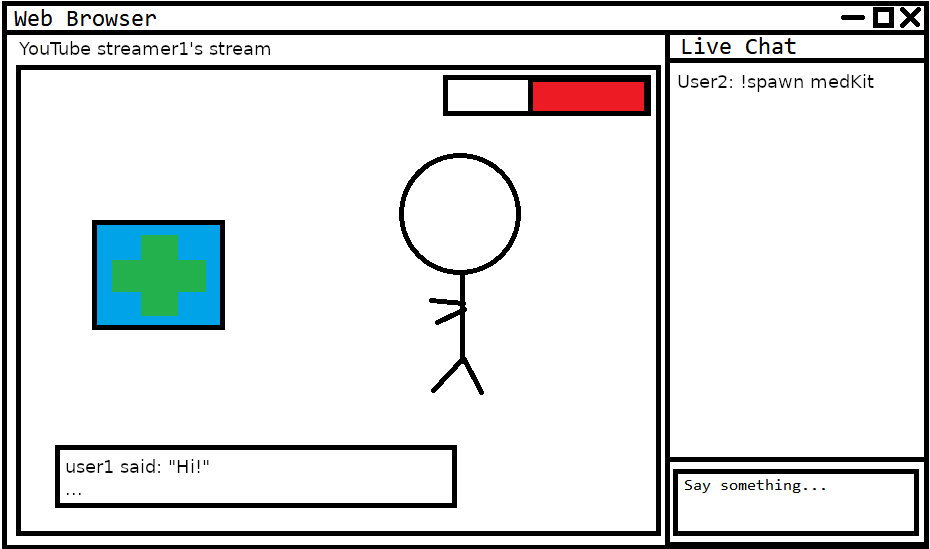
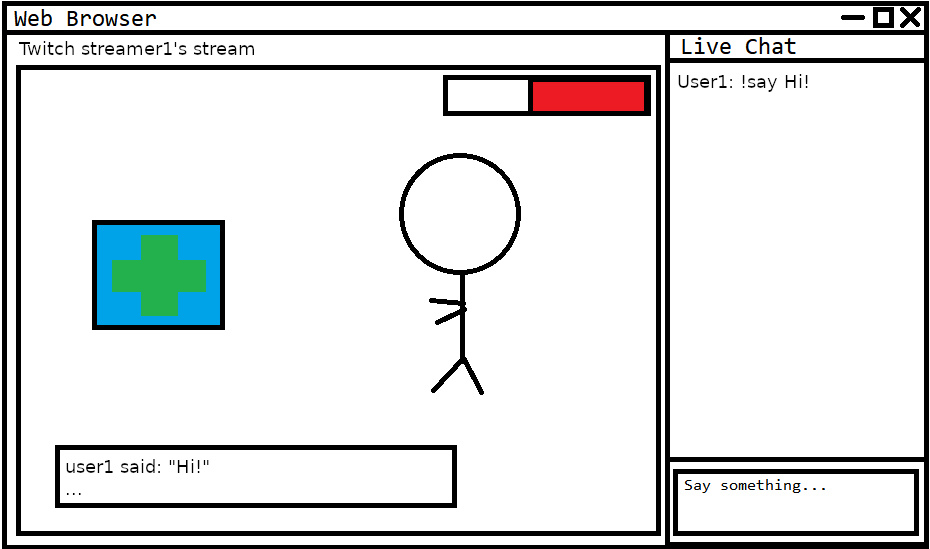
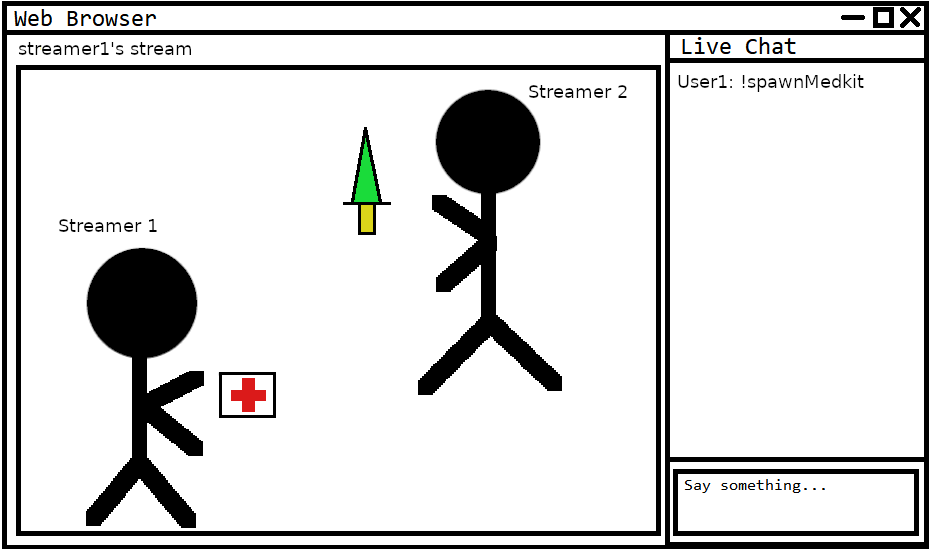


Figure 20. Two streaming services being used at once. The streamer has connected their game instance to two separate live streaming services. Commands from both live chats will influence the same game world.



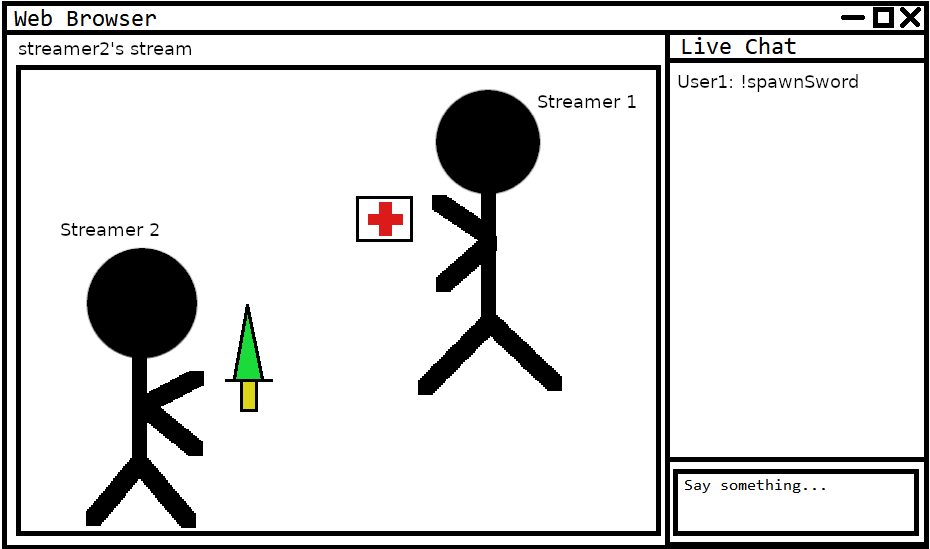


Figure 21. Multiplayer match with two streamers. Each streamer has connected their game instance to their respective streams and their viewers send commands that only affect the streamer they are watching.

**8. Glossary**

**Application programming interface (API):** A system of tools and resources enabling developers to create software applications.

**Chatbots:** Special programs that can moderate a Twitch.tv channel's chat log, greet new viewers, post scheduled messages, and add extra functionality to a livestream.

**Commands:** Chat comments that can be read and interpreted by the API.

**Features:** Characteristics or abilities, things that can be done with the use of the API.

**Game engine:** The basic software, or foundation, of a video game. Contains features and software that can be shared across multiple different games.

**Game state:** Description of the parameters of a game at a given point in time.

**Interaction:** The act of a viewer changing the game state while the streamer is playing the game.

**IRC:** Internet Chat Relay is a service that allows people to chat with each other online under a client/server model where individuals use a client program to connect to an IRC server.

**Live streams:** A live video transmission of a game over the internet.

**Local Chat:** A local client chat window for testing the application so we do not have to rely on an external live streaming site during development.

**Modding:** Alteration by players or fans of a video game that changes one or more aspects of the game, such as how it looks or behaves.

**Multiplayer:** Denoting or relating to a video game designed for or involving several players**.**

**Networking system:** Two or more computers that are linked in order to share resources, exchange files, or allow electronic communications.

**One-to-Many Player Game:** A single player streams live footage of their game to an online video streaming service. The many viewers of that stream then interact by sending commands through the stream chat that modify the game state in some way.

**Packets:** A formatted unit of data carried by a packet-switched network. A packet consists of control information and user data

**Plug-in:** A software component that adds a specific feature to an existing computer program.

**Prototype:** A first, typical or preliminary model of the software from which other forms are developed or copied.

**Streamer:** Someone who streams video games live for an audience.

**Streaming services:** A web site or application that allows for the live transmission video.

**TCP:** The Transmission Control Protocol which is a protocol for transmitting data over the internet.

**TCP sockets:** An internal endpoint for sending or receiving data within a node on a computer network using TCP.

**Twitch.tv:** An online live streaming video platform with a focus on gaming.

**Video game:** A game played by electronically manipulating images produced by a computer program on a television screen or other display screen.

**Viewers:** The people who watch live streamers play games on a live streaming service.

**YouTube.com:** A video sharing site that has also developed a live streaming web application which is similar to the one at Twitch.tv

**9. Contribution of Team Members**

Alexandra Pasinski

Hours Worked: 6

Parts Worked On: Met with advisor to discuss software design ideas. Helped design the context model, program units UML diagram, activity and state charts, main data structures, user interface designs, and the glossary.

Jake Lahr

Hours Worked: 6

Parts Worked On: Met with advisor to discuss software design ideas. Helped design and draw context model, program units, activity and state charts, and user interface designs. Helped write glossary terms.

Mike DesRoches

Hours Worked: 6

Parts Worked On: Created snapshots in user interface design and compiled state diagrams in detailed design.

Christian Garcia

Hours Worked: 5

Parts Worked On: table of contents, abstract, introduction, glossary, and state diagrams.