UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Hans Matthias Andreas

Twitch Integration for Blastronaut Game

Bachelor's Thesis (9 ECTS)

Supervisor: Jaanus Jaggo, MSc

2022

Twitch Integration for Blastronaut Game

Abstract:

Twitch live streaming is a popular form of entertainment for many people with interest toward video games. Since acquiring players for their games is a challenge for most developers, live streams provide an additional way to get noticed. Many games provide creative ways to make the live stream more engaging by integrating different Twitch services. This thesis describes the design and implementation of a Twitch integration for the game Blastronaut. The goal is to develop an in-game robot that is controlled by the Twitch viewers and can be played before the final release of the game. The solution will then be user tested and the feedback will be analysed to measure the suitability of such a solution.

Keywords:

Live streaming, Twitch, integration, Godot, video games, user testing

CERCS:

P170: Computer science, numerical analysis, systems, control

Twitch integratsioon Blastronaut mängule

Lühikokkuvõte:

Otseülekanded Twitch'is on videomängudest huvitatud inimestele populaarne meelelahutus. Kuna mänguarendajatele on uute mängijate leidmine keeruline, siis sellised otseülekanded pakuvad neile head võimalust oma mängu levitada. Paljud mängud on leidnud loomingulisi viise, kuidas oma otseülekannetesse inimesi kaasata. Selleks liidestavad nad oma mängu Twitch teenustega, mis muudavad otseülekanded kaasahaaravamaks. Käesoleva Töö eesmärgiks on arendada Blastronaut mängule Twitch'i vaatajate poolt kontrollitav robot, mida saab proovida juba enne põhimängu avaldamist. Töös viiakse ka läbi kasutajatestimine, mille tagasiside põhjal hinnatakse loodud lahenduse sobivust.

Võtmesõnad:

Otseülekanded, Twitch, integratsioon, Godot, videomängud, kasutaja testimine

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction

Live streaming has become a very popular type of entertainment on the internet. One of the most popular types of live streams are people broadcasting themselves playing video games. As such, video game developers and publishers have slowly begun catering games to live streams. One of the ways of doing this is by creating integrations for games where viewers can in some way interact with the game that is being streamed. This can be a good way to increase the visibility of those games and in turn, gain more players for them [1].

Blastronaut[1] (Figure 1) is a game about mining and exploring in a procedurally generated world, developed by Jaanus Jaggo with the assistance of multiple students. The aim of this thesis is to add a Twitch integration for the Blastronaut game that makes the game playable through a stream. For this, an in-game robot character is created which can be controlled through a chat room on Twitch.

The goals of the present thesis are:

- To describe the design of various live streaming integrations for video games.
- Implement a robot that is controlled by Twitch for the game Blastronaut.
- Test the developed solution to validate its suitability as a way of acquiring new players.

Figure 1: Screenshot from Blastronaut

[1] https://store.steampowered.com/app/1392650/BLASTRONAUT/

Firstly, a brief overview of live streaming and Twitch will be given in chapter 2. This will include a more detailed description of what live streaming is and the different types of streams. A better understanding of why live streaming and video games have an overlapping audience will also be given to the reader. In addition, the reason for choosing Twitch as the platform for the integration instead of various other live streaming platforms will also be explained.

Secondly, there are a variety of different ways one can implement a live stream integration for a game. Thus, different types of integrations will be explained in chapter 3 and examples of how existing games have implemented them will be given. The integration developed in this thesis, falls under one of those types and the reason for the choice can also be found in the chapter.

Finally, the design and implementation of the integration will be explored in chapter 4. This will include how the game connects with the platform Twitch. Due to the type of integration chosen for this project, the base game of Blastronaut and the integration version have considerable differences. Thus, many design choices needed to be made and implemented to accommodate them. The reasoning behind those choices, as well as the implementation of them will also be explained in this chapter. The final product will also be tested and the results analysed, which will be done in chapter 5.

2. Live Streaming

Live streaming is the act of broadcasting media in real-time over the internet. This is typically done on platforms designed for live streaming and is a participatory experience where one person shares the media and other people can choose to view it. Live streams generally also have a chat room attached to them, where viewers can interact with the broadcaster and with each other [2]. By far the most popular approach is having the host(s) stream themselves doing something while they interact with the chat. This can include people doing some physical activity like exercising, playing an instrument or eating, or it can also include people sharing their computer screens while they do something on it [3].

One of the most widespread uses of live streaming is people sharing their screen while they play a video game and then commentating their thoughts and reactions while interacting with the chat room of their channel. A big reason why these types of live streams are popular is because many people who play video games on PC spend a large amount of time behind their computer. Thus, many of them also seek other forms of entertainment on the device. This inevitably leads some of them to live streaming platforms, where it is very likely that they can find a broadcaster which interests them. These streams also have a way of retaining users, as they can act as communities for people with shared interest. [4]

2.1 Twitch

Video game live streaming is where Twitch comes into importance. Twitch is a live streaming platform made primarily for video games, and while in recent years it has gained popularity among other types of categories, it is still the number one live stream platform for video games [5]. There are many reasons for this, one being that it is one of the oldest live streaming platforms, having first been founded under the name Justin.tv in 2007, later in 2014 fully rebranding itself to Twitch [6]. Another important factor is that it has been able to add many widely used features over the years which its competitors lack. They are also very supportive of independent developers as they have a comprehensive API[2] that is publicly available.

[2] https://dev.twitch.tv/docs/

In addition, the people behind Twitch also understand that they have a symbiotic relationship with video games as they both gain from each other's successes. As such, they have many systems in place in order to aid video games gain new users, retain users , create engaging experiences and to help the game's influencers monetize off of their game. One of these systems is called the bounty board, where developers or publishers can make postings offering sponsorships to somehow promote their game. This can just be the broadcaster playing it or doing something else which relates to the game like viewing the game's trailer or cosplaying a character from the game. There is also a system called Drops which enables viewers to gain in-game items by viewing promoted broadcasters. Lastly, from Twitch's own side, there are game extensions, which are Twitch overlays showing game information fetched from the game being broadcasted on top of the streamed video. These all create an ecosystem for gaining viewers and users for both the game and the people streaming it (Figure 2). [7]

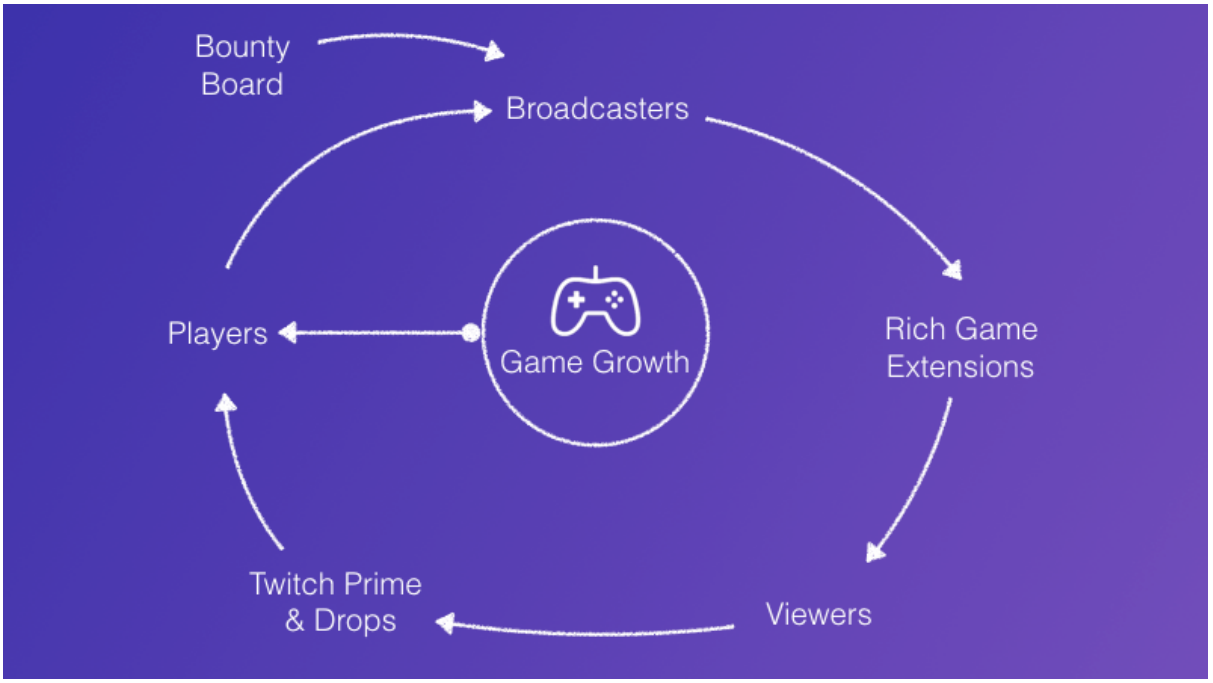

Figure 2: Twitch ecosystem for game growth[3].

Finally, there are software solutions from game developers themselves, which are known as twitch integrations. In short, twitch integrations are features in games which connect with twitch and the chosen channel from which they read the chat room messages and cause interactivity between the game and messages which contain keywords the game recognizes.

[3] https://blog.twitch.tv/en/2019/03/29/twitch-for-game-developers-b664163bed65/

These integrations can create either fun dynamics between the broadcaster and the viewers or they can also be unique experiences where the viewers are the main focus. Twitch integrations differ from each game to the next, but how the messages are received from twitch is generally the same.

2.2 Other platforms

Aside from Twitch, there are many other platforms which also offer people the ability to broadcast themselves to viewers. Some of these are platforms which are made for live streaming purposes, but focus on having them in private or controlled environments like Zoom[4]. It is a great application for having private meetings or conferences, but it isn't a platform for actual broadcasters. This is because the platform has no features to support growth for individuals or for discovery.

There are also platforms like Twitch, in the way that the entire platform is made solely for live streaming to a public audience, like DLive[5]. Others however, are more broad and only have a section for live streaming like YouTube[6] or Facebook[7]. Many of these are also very popular, but are nowhere near as popular in the video game market. They also either lack a publicly available API or just have an inferior one, which does not support many things, like options to interact directly with chat. As such, for the needs and purposes of this thesis, Twitch is the best choice.

[4] https://zoom.us/
[5] https://dlive.tv/
[6] https://www.youtube.com/
[7] https://www.facebook.com/watch

3. Types of live stream integrations

Twitch integrations have been done for a multitude of different games over the past few years and have gained moderate popularity during them. As such, there is also quite a bit of variety in regards to the type of integrations which exist [8]. These could be categorised in many different ways, but in this thesis they will be categorised in four different ways:

1. Viewers play along with the broadcaster - the viewers are playing along with the streamer, generally in a passive way not directly affecting the broadcaster's experience.
2. Viewers affect the broadcaster - the broadcaster is the main character of the game and viewers through the integration interact with the broadcaster's character in some way.
3. Viewers as the characters - the viewers are the characters of the game and the broadcaster is just an observer or host of the session.
4. Autonomous stream - a situation where there is no actual broadcaster as in human influencer behind the broadcast, but the stream is autonomous and the game is fully played by the viewers.

While there are games which have integrations that could fit into multiple of these categories, generally that is not the case.

3.1 Viewers play along with the broadcaster

In an integration where the viewers are playing together with the streamer, it is often a passive experience which is used often in trivia games for example, a type of game where players are given a question and generally get to choose from multiple answers as to which one is correct. The integration for this type of game would typically be one where viewers also vote which choice they think is correct and the viewer votes get pooled together, having the choice with most viewer votes be the final answer for the viewers. As such, the broadcaster and viewers can compare themselves with each other and it can create fun dynamics and discussions between the two parties and among the viewers themselves.

A good example of this type of integration is Trivia Murder Party from the Jackbox Party Pack game series[8]. It is a party trivia game where viewers can vote along with the actual players. Every player has a score above their name, which is represented in dollars. This also applies to the viewers, who are all treated as one entity that is separated into the bottom left corner (Figure 3).

Figure 3: Screenshot from Trivia Murder Party.

3.2 Viewers affect the broadcaster

While in trivia game twitch integrations, as mentioned in the last chapter, the viewers don't have a meaningful impact on the broadcaster's gameplay, with action games, it is often the opposite. The most popular type of twitch integration for action games is one where the viewers can have a significant impact on the broadcaster's experience. This is most typically done by having periodic voting, where viewers decide what should happen to the broadcaster's character or in the game world. These votes generally only contain a small amount of the total possible options and these options could be beneficial or harmful to the broadcaster. They could also just be chaotic neutral which means that they have both negatives and positives [8].

[8] https://store.steampowered.com/app/434170/The_Jackbox_Party_Pack_3/

One game which applies this type of integration is called Immortal Redneck[9] (Figure 4). It is an action shooting game, where the player has to clear through various floors filled with enemies in a temple, with the goal to reach the exit. Some of the possible scenarios which the users could vote on are the following: enemies turn into chickens, the floor becomes lava, player movement gets randomised.

Figure 4: Screenshot from Immortal redneck. The twitch voting option is seen as a bar at the top of the screen.

3.3 Viewers as the characters

Some integrations are designed with the intention for the streamer to just relax and chat with their viewers. In such games, the viewers are made into characters in the game. In these types of integrations, the viewer characters are usually made to participate in some sort of event with each other while the broadcaster commentates on the happenings [8]. Broadcasters often come up with some rewards to give to their viewers who win when they decide to play these types of games as well.

The most popular game with this type of an integration is Marbles on Stream[10], a game fully designed to be played while live streaming. When the broadcaster has set up the session for

[9] https://store.steampowered.com/app/595140/Immortal_Redneck/
[10] https://store.steampowered.com/app/1170970/Marbles_on_Stream/

this game, viewers need to type a specific command into the chat to have a character of themselves put into the game. This is a marble with the participant's name attached to it. Once the viewers have joined and the broadcaster decides to start the game, all the marbles are thrown into an obstacle course. trying to roll their way to the ending of the course (Figure 5).

Figure 5: Depicts gameplay of Marbles on Stream.

3.4 Autonomous stream

This final category is a type of integration where there is no actual broadcaster entertaining the viewers. Instead, a game is being broadcast to a twitch channel and the viewers are the only players playing. This type of a solution has typically been done by third-party members not affiliated with the developers of those games and so the games themselves have not been modified. As such, viewers are just playing an existing game together, one they could easily go and play alone themselves [8]. In these types of solutions, the game actions can occur on a timer, with the most frequently typed viewer command being chosen at the end of each timer interval. They can also occur in a way where all commands are executed in a sequence.

An example of this is the twitch channel Twitch Plays Pokemon[11], where old pokemon games are being streamed and the viewers try to finish the games as a community. This used to implement both command behaviour systems, where viewers could vote for which one to use. [9]. Currently though, commands are only executed sequentially. In these streams, one portion of the screen is also dedicated to showing what commands have been recently typed by the viewers.(Figure 6)

Figure 6: Screenshot of the Twitch Plays Pokemon stream.

3.5 Chosen type for Blastronaut

When choosing what type of integration to create for Blastronaut, an important thought was “What would be cool and interesting?”. Many of these integration types have been thoroughly explored by vast amounts of different games, but there was one exception. While autonomous game streams have been done for existing games, it is very rare for a game to have been specifically designed for it. As such, this thesis intends to explore what the results would be when creating a mode of gameplay for Blastronaut that is only intended to be used by an autonomous stream. This type of stream can also be broadcast before the release of a game, which could be a way to acquire new players.

[11] https://www.twitch.tv/twitchplayspokemon

4. Design and Implementation

As part of the thesis, a working Twitch integration for Blastronaut was implemented which included four main parts:

- A connection with the Twitch platform API for user interaction in the Twitch chat rooms.
- Chat room commands for the users to be able to interact with the game.
- An overarching gameplay loop specific to the integration. This could be treated as a separate mode of gameplay to the base Blastronaut game.
- Various functions for increased interactivity.

Since the game *Blastronaut* is created with a game engine called Godot[12], it was also used in this project. The primary programming language in Godot engine is GDScript. When comparing it to other commonly used programming languages, it is most similar to Python.

4.1 Connection with Twitch

Firstly, to establish a connection between the game and Twitch, the Twitch API was used. It is a RESTful API, which allows developers to get information about various subjects, like users or channels, on the Twitch platform. What is mainly sought after using the API, is retrieving user input data from twitch users that type in the appropriate twitch channel's chat or privately to the designated Twitch account. One of the most well maintained Godot addon for Twitch integration is called *GIFT - Godot IRC For Twitch*[13]. This has all the needed functionality. Thus, there was no need to recreate something already existing and only slight modifications were done to the addon for the specific needs of this project.

To first get user and message info from the chat's, a connection needs to be established with the specified channel. When Blastronaut is loaded up, the game sends a login request to the Twitch API. After that, the Twitch API connects to a bot account, which acts as an

[12] https://docs.godotengine.org/en/stable/about/index.html
[13] https://godotengine.org/asset-library/asset/432

intermediary between the chat and the game. Finally, the bot account joins the desired channel and is able to see its features, like the chat. This process is shown on (Figure 7).

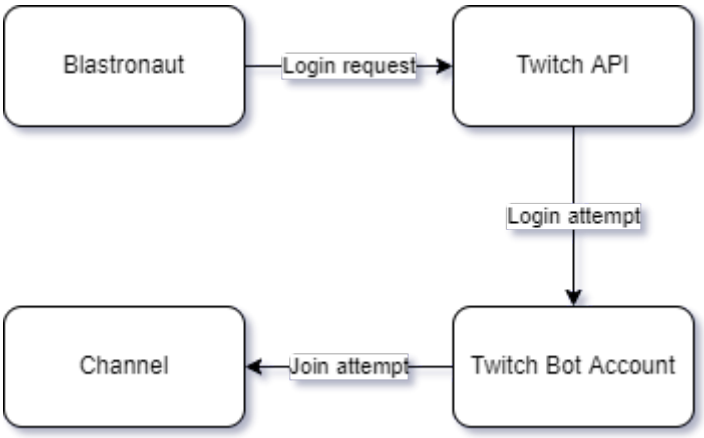

Figure 7: The process of establishing an initial connection with a twitch channel

Through the API, the intermediary bot account starts reading the twitch chat and relaying messages back to Blastronaut. Once Blastronaut recognizes a valid command, the action described for it will be acted out. (Figure 8)

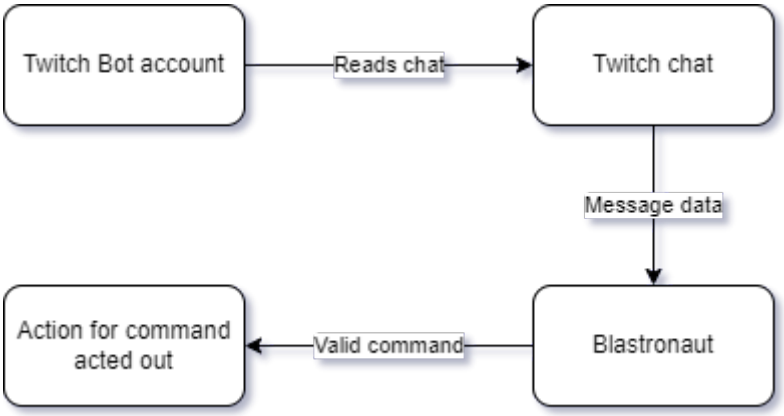

Figure 8: How messages are retrieved from twitch chat

With this, viewers will be able to type commands into chat to play the integration version of Blastronaut.

4.2 Commands

For the chat room users to interact with the game, they need to type specific commands which correspond to the actions they want to do. For example, if the user wants to move the character in a certain direction, they need to type the command for it (Table 1: 3-6) into the chat room. All the various commands can be seen in Table 1, and the solutions in which they are used are described in the later subchapters.

Table 1: Chat commands

Nr	Command	Parameters	Type	Description
1	!commands		Chat interactivity	Posts a message to the chat room which contains and explains all the various commands.
2	!me		Chat interactivity	Posts a message to the chat which informs the user how many valid commands they have typed and how many points they have.
3	!right	(optional) 0-100%	Movement	Moves the character right the max possible distance. If a parameter is given, distance varies depending on the given percentage.
4	!left	(optional) 0-100%	Movement	Moves the character left the max possible distance. If a parameter is given, distance varies depending on the given percentage.
5	!up	(optional) 0-100%	Movement	Moves the character up the max possible distance. If a parameter is given, distance varies depending on the given percentage.

6	!down	(optional) 0-100%	Movement	Moves the character down the max possible distance. If a parameter is given, distance varies depending on the given percentage.
7	!shoot	(optional) cardinal or ordinal direction (N,S,E,W,NE, NW,SE,SW)	Shooting	Shoots in the direction the character is facing. If a parameter is given, shoots in the corresponding direction.
8	!dock		Station	Used to dock the character to a nearby station of one is present.
9	!option	Whole number between 1 and 6	Station	Used to purchase from the station shop. Once the command is active, the station menu is closed and notification of purchase is shown.
10	!say	Message (any combinations of characters after the initial command)	Chat interactivity	Broadcasts the message to the in-game screen. Costs points gained from typing valid commands. Username of the person who wrote the command is also shown after the message.

4.3 Gameplay mechanics

Blastronaut is a game where the general goal of the player is to explore the world and mine various ores in order to upgrade the character or build new structures. The Twitch integration game also has a similar game loop. The viewers control a mining drone that can explore, mine and sell minerals to upgrade its gear. However, they can not build new structures as it is reserved for the PC game. However, many changes are done to the specifics of how that is accomplished.

The biggest difference between the base Blastronaut game and the integration version is how the main character is controlled. In the base game, one person controls the main character directly through peripheral input devices, which is typically both a mouse and a keyboard. This allows for the user to make precise real-time inputs. While this type of character control is currently the industry standard and used by the vast majority of video games, it is not a viable option for the integration. This is because the character in the integration version of the game is intended to be controlled through a chat and by multiple people at the same time with no upper limit to the number of people who can participate. As such, a system for user inputs is created which considers the inputs of every user. This however, comes at the cost of fluidity and precision and heavily impacts the flow of gameplay. In turn, the gameplay also needs to be adjusted to the input system.

To accommodate the changes to movement, the model of the playable character was also changed to a robot. The new model can be seen in Figure 9.

Figure 9: New character model.

4.3.1 Timer

To combat the large number of possible inputs, the characters' every action will be done based on a timer. At the end of each timer interval, all of the valid command inputs will be pooled together and the most frequent command will be chosen. The action which represents the command will be done during the next timer interval. In the game, the timer can be seen as a progress bar above the playable character (Figure 10). The percentage of the bar which is green, represents the passed time of the current timer interval.

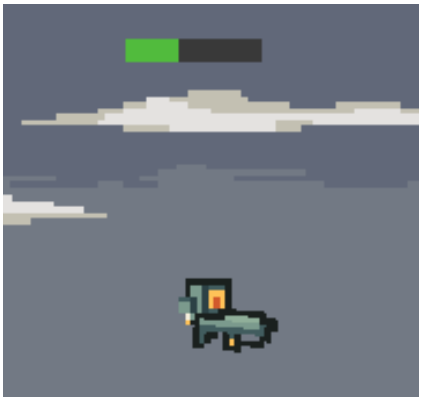

Figure 10: Character with progress bar above it.

4.3.2 Movement

Due to the fact that movement can not be as precise with the integration as in the base game, some changes had to be made. Firstly, players can only move in four directions: left, right, up and down. This was done because through this every location can still be reached and it is very simple to understand. Thus it limits confusion and having limited options makes it easier for multiple people to decide on the same action. All of these directions have their corresponding movement command (Table 1: 3-6).

Secondly, in the base game the character is limited to move on top the breakable block tiles. For the integration, the decision was made to remove this limitation and allow the character to move through blocks (Figure 11:A). The reason for this is that with imprecise movement, exploring the world will inevitably become slower. As such, the change was made with the attempt to keep moving around and exploration stay interesting and fun.

Since the character is now allowed to move through blocks, there could be situations where the character stays inside the blocks after the movement commands. This is not ideal as the

character can damage themselves with their weapons. To prevent this, if the character is inside blocks (Figure 11: B), it will keep moving until it is in an area where it is safe to shoot again (Figure 11: C).

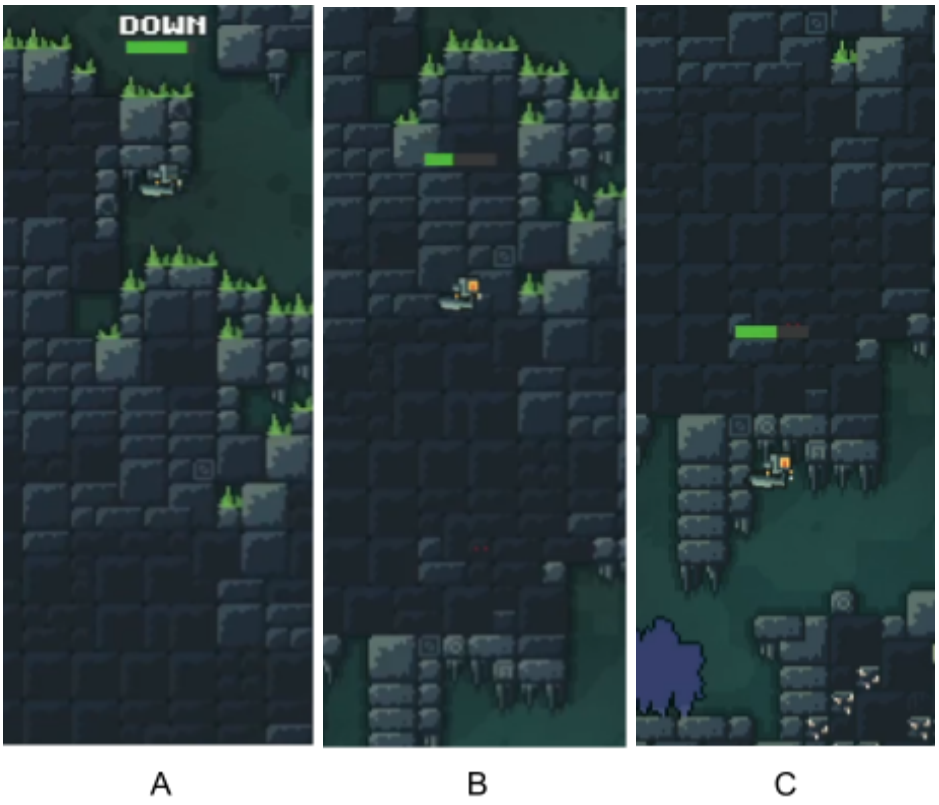

Figure 11: Movement. A – Character moving down through blocks, movement command active. B – Character moving down through blocks, no commands active. C – Character is outside of blocks, stopped.

4.3.3 Shooting

While nothing was really changed to the core of using weapons to shoot, it was necessary to limit the direction one can do it. This is because the input of every user has to be considered. Even though it would still be possible to allow users to choose an angle around the player with degrees or radians. This would create the need for players to calculate them on the fly for where they want to go, which would just create confusion and be very tedious. As such, the directions will be limited to cardinal and ordinal directions. Shooting is done through the use of the ‘shoot’ command (Table 1: 7) and an example of the character shooting east can be seen in Figure 12.

Figure 12: Depicts the character shooting

4.3.4 Ores

In the base game, once the character breaks ore blocks, they fall to the ground. To collect them, the player needs to move very close to the dropped ore blocks for them to get registered to the character. With the movement system in the integration, this would make it very tedious to collect ores. Thus, whenever ores are broken (Figure 13:A), they start drifting toward the player immediately once they drop, until eventually reaching the player where they get registered to them (Figure 13:B).

Figure 13: Ore collection. A – Ores dropping upon getting broken. B – Dropped ores drifting towards the player.

4.3.5 Tasks

To give clear goals for the users to strive towards, a multitude of tasks were added. The main set of tasks guides the players through all the different biomes asking for them to collect resources in them (Figure 14). Between moving zones a task to purchase a new option from the shop is also given. Once all the main tasks are done, there is a new set of tasks from which a random task is chosen each time a task is done. This allows for there to always be a task on the screen for the users to complete.

Figure 14: Task which requires collection of ores.

4.3.6 Station

For the purpose of upgrading the character, machines referred to as stations are scattered around the world. These stations act as shops, where the health and fuel of the character can be refilled and various upgrades can be bought. To remove the need to accurately navigate the character to the shop as it can be tedious, it is possible to directly move to one using the ‘dock’ command (Table 1:8) if a station is visible on screen (Figure 15).

Figure 15: Depicts docking.

When one uses the shop, a menu showing all the various options opens on the screen (Figure 16: A). The possible upgrades are ones to health and fuel, upgrade to the character's speed, and an option to buy a different weapon. All of the choices in the shop cost money. A currency that one can get by collecting ores and then interacting with the shop, where the ores will automatically be converted to money. Once something from the shop is purchased, the shop will automatically close and what was bought will be shown above the head of the character for a few seconds (Figure 16: B). Buying from the shop can be done with the 'option' command (Table 1: 9)

Once the shop is opened, the timer for user actions increases and stays that way until something is purchased or an action is made which closes the shop. This is done to give the users some additional time to think, since purchases will remove currency from the character.

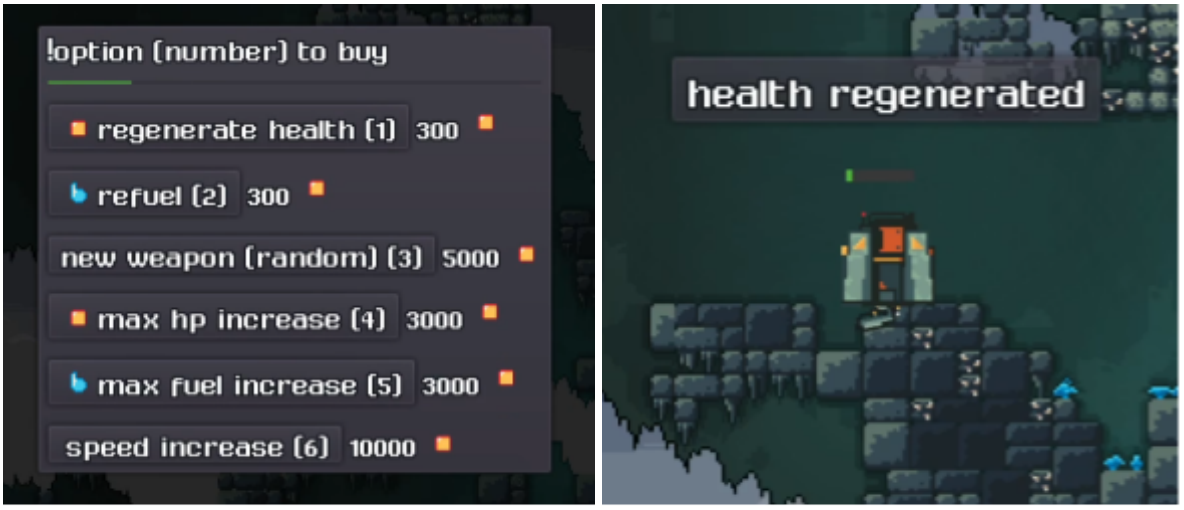

Figure 16: Station shop. A – Shop menu. B – Notification of purchase.

4.4 Chat interactivity

Since the game is played through a chat room. There is also a need to acknowledge it and give assistance inside of it. For assistance and information purposes, there are two messages which the bot account connected to the integration can post inside the chat. One is a list of all the possible commands, the valid inputs which determine the actions the robot can do (Figure 17). It can be prompted with the 'commands' command (Table 1: 1)

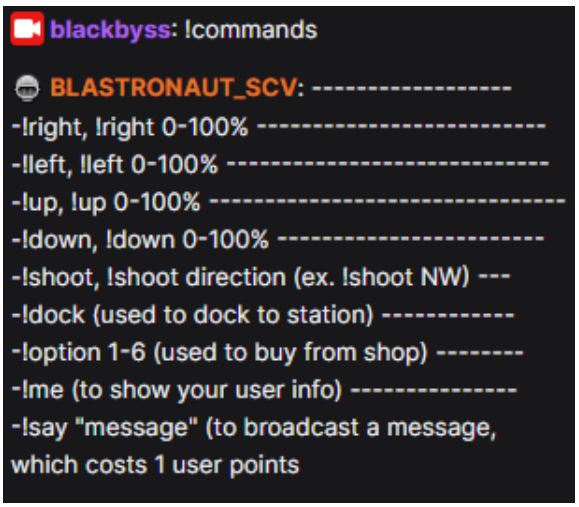

Figure 17: Depicts information about commands in the Twitch chat

Another is a message which displays the amount of actions a user has contributed in doing and the amount of points the user has (Figure 18). The information displayed is regarding the user who asked for it. The command in question is called ‘me’ (Table 1: 2).

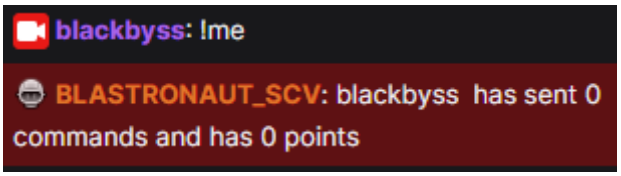

Figure 18: Depicts the !me command

The points which were mentioned in the previous paragraph are gained by typing valid commands into the chat. They are used for message broadcasting, a way to have the user's message shown inside the actual game through the ‘say’ command (Table 1: 10). In the message shown in the game, the username of who this message belongs to is also visible. (Figure 19)

Figure 19: Depicts message broadcasting.

5. User testing

To test the game, a live stream was set up on a specific Twitch channel, where the integration version of Blastronaut was being streamed. Afterwards, users were given a form (Appendix: 3) which had the basic instructions of the game and directed them to the live stream. Going through the form, they were introduced to all the commands listed in Table 1 and all the major mechanics described in chapter 4.3. After the introduction to each mechanic, they were expected to try it out themselves and then give their feedback about it. The test group consisted of 8 people who were all individuals that play video games and watch live streams centred around them on a daily basis. They had also never heard about Blastronaut before participating in this. A screenshot from the test session can be seen in Figure 20.

Figure 20: Screenshot from the test session

The initial two questions were about movement. The first one asked participants to give their rating on how difficult it was to move to a location they wished to go. It was a linear scale question with choices ranging from 1 to 4, with 1 representing very easy and 4 representing very hard. The second question asked users to write their thoughts on movement, be it any issues they had with it or some general feedback.

The general consensus among testers was that it was not difficult to move to their desired location as seen in Figure 21. Only one person had difficulties with the movement which was

due to a feature in Twitch that didn't allow users to send identical messages twice in a 30 second period. It is possible to turn that feature off in future testing sessions. Aside from that, an issue which many users had was having to type % when specifying movement length. This was something they just found annoying to constantly do as the symbol requires a combination of keys to type on many keyboard layouts. The requirement to have the symbol was added for the sake of clarity, but users seemed to enjoy specifying length much more frequently than was expected. Clarity should not come from the cost of inconveniencing users. In the future, this could be solved by removing the requirement for the symbol, or by changing the way one can specify movement length. One way of doing this could be by specifying the actual amount of blocks moved, instead of a percentage of the maximum movable length.

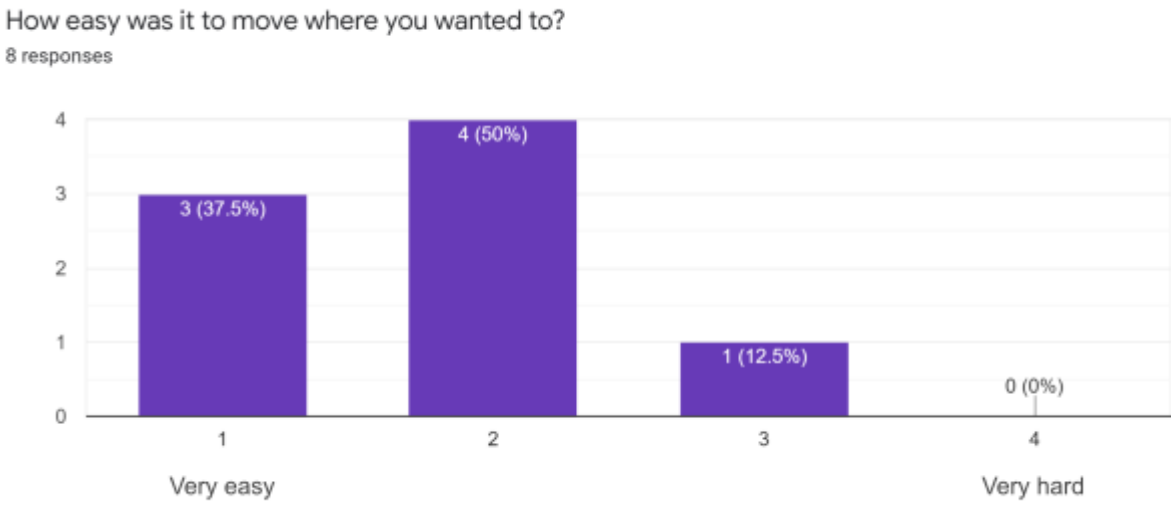

Figure 21: User ratings regarding the difficulty of movement

The next topic which got asked about was shooting. This included 4 questions. The first was a question where users were asked to rate how difficult it was to break the blocks they wished to break. This was a linear scale question with choices ranging from 1 to 4, 1 representing very easy and 4 representing very hard. The second was a yes or no question asking whether the users felt it necessary to have more shooting directions. The next question was also a simple yes or no question. It asked whether the inability to change the duration the character shoots was an issue. Finally, the last of the four was a general freeform question where users could describe any issues they had or write any other feedback about shooting.

Contrary to initial expectation, no one seemed to find shooting the desired blocks difficult (Figure 22). Majority of users sided toward shooting to be just easy rather than very easy though. From the following yes or no questions and the written feedback one, the general consensus is that there isn't a need for more shooting directions or to be able to specify the duration of shooting. Although it should still be said that a couple of people would have liked those options. It is debatable whether they should be added or not though, since adding either could cause more confusion.

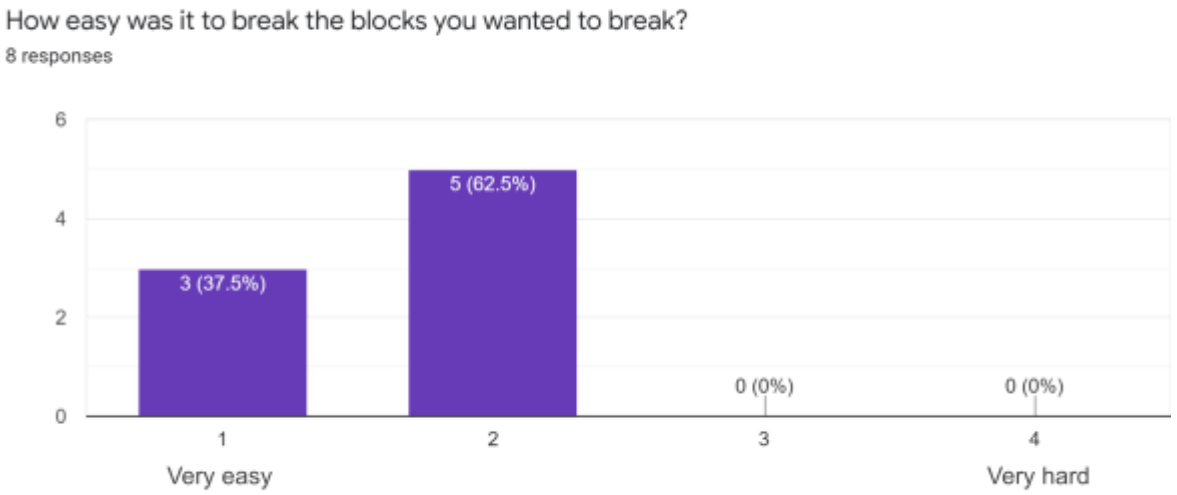

Figure 22: User ratings regarding the difficulty of shooting

Going forward, users were asked their thoughts about the station and its shop. This included three main questions and two explanatory ones. The initial question asked whether interacting with the station and its shop was intuitive. This had a yes or no answer. If the users answered no, they were asked to describe what they found troublesome. Next up, participants were asked if they thought that the shop could have more options in it. This was another yes or no question. If their answer was yes, they were asked to describe what could be added to the shop. The following question asked their thoughts about the prices which the shop items had. This was a linear scale question with answers ranging from 1 to 5, 1 representing cheap and 5 representing expensive.

From user answers about the intuitivity of interacting with the station and its shop, only one person had a negative answer to the question (Figure 23). The reason for that negative answer was that it wasn't possible to buy more than one item from the shop before it closes. This was

done that way because it is generally expected for only one purchase to be made during a visit to the station. Thus, having it close upon a purchase, reduces the need for users to type a command to close the shop.

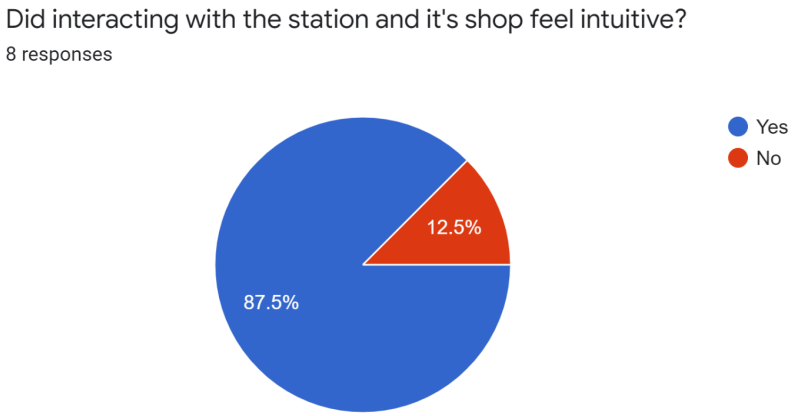

Figure 23: User answers to whether interactions with the station and its shop felt intuitive or not.

Toward the question regarding the amount of items in the shop. The majority said that there wasn't a need for any more options (Figure 24). Three people still wished for more options though. Through the follow-up question, the things they wished to be added were: additional damage for weapons, more weapons or melee weapons and something for increased progression.

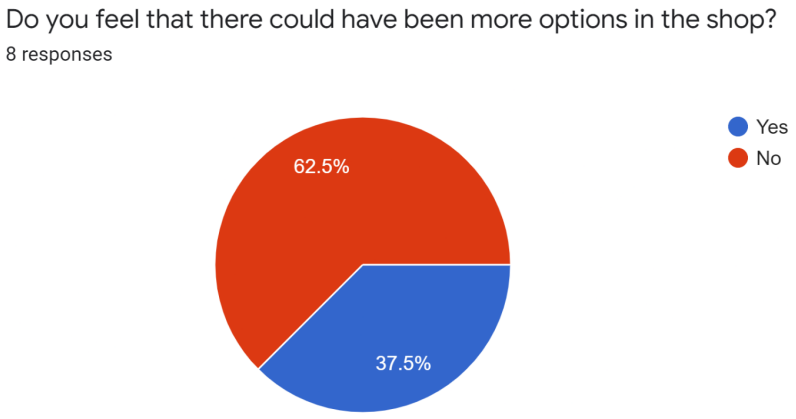

Figure 24: User answers to whether there could be more options in the shop.

The vast majority also felt that the prices in the shop were fairly priced (Figure 25). Only one person felt they were a bit expensive and another person felt that they were slightly too cheap.

Figure 25: User ratings regarding their thoughts about shop prices.

The next question was regarding the message broadcasting command. This was a simple writing question where they were asked their thoughts about it. From user feedback, everyone seemed to love this feature. One suggested idea that could be explored was to have an automated voice bot read out the text which was written as well.

The timer was also something which was asked about. Users needed to give a rating on a linear scale about how fast they thought the timer intervals were. The answers ranged from 1 to 5 with 1 representing too fast and 5 representing too slow.

This had mixed results (Figure 26). Two people felt the timer was a bit too fast, three people felt it was perfect and another three thought that it was slightly too slow. Due to all the answers still being centred around the timer duration being perfect, it should be fair to not make any big changes to it. One idea which could still be done would be to make a command to change the duration of the timer in a reasonable range of values. This command would require some limitations attached to it though so one or two people can not ruin the flow of

gameplay for the many. One limitation could be that over half of the users in the chat have to vote for this for it to be able to go through.

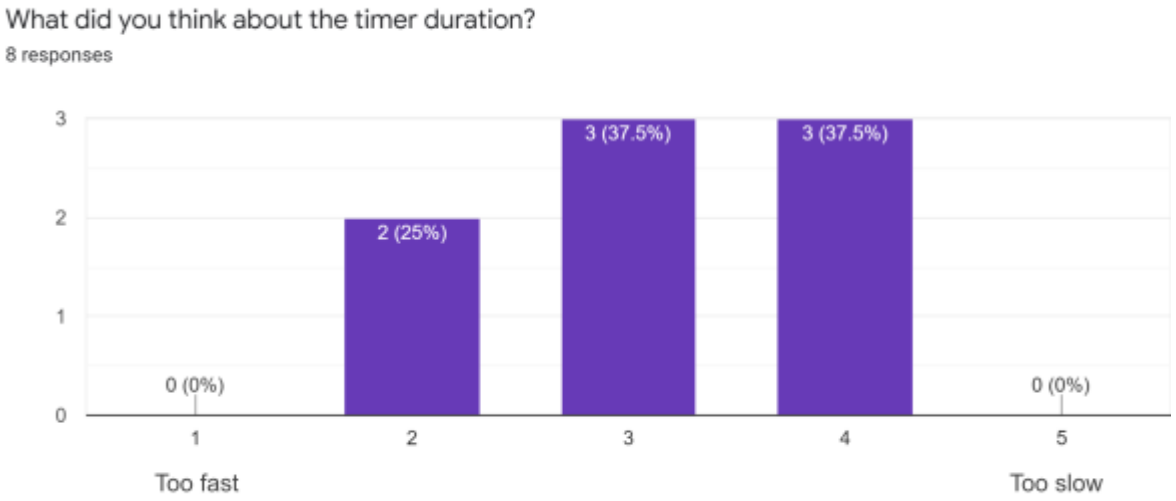

Figure 26: User ratings regarding the duration of the timer.

Next was a question about tasks where the users were asked their opinion about them. Participants were able to select multiple answers from a list of choices. The choices were as follows: “They prompted me to explore more”, “It felt good to complete them”, “They were tedious”, “Barely noticed them”, “Other”. If users selected “Other”, they could write a custom answer with it.

Most users felt the tasks were good to complete as that option had 5 answers. Two people also answered that they barely noticed the tasks. One user was also prompted to explore more. These results are not bad, but they could certainly be better. One way to improve them could be by adding some type of rewards for completing them.

The following was a linear scale question asking users how fun their overall experience was. The answers ranged from 1 to 4 with 1 representing very boring and 4 representing very fun. All the users responded positively to this question, with every one saying that they had fun to some degree (Figure 27). Three people even answered that they found the experience very fun.

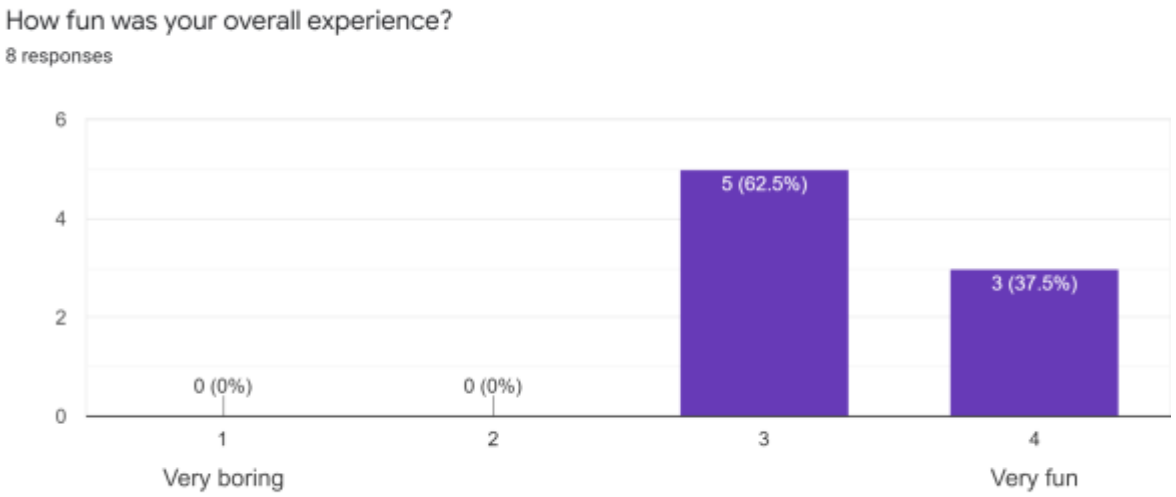

Figure 27: User ratings regarding their enjoyment of the game.

Users were also asked whether there were any features which they wanted to be added into this version of the game. This could be answered with freeform text. Regarding the answers, two people generally just wished for more of everything, another wanted to have the option to change the appearance of the bot and one person wished for building to be an option. These are all things which could be explored. Changing the appearance of the bot was something that was even an option in the shop at one point. The idea was to change your robot's model to that of a robotic spider, which would also change the way the character moved. The remnants of this idea can still be found in the source code, but the idea was eventually scrapped due to the inability to get the movement with the new character model to work properly.

One of the answers to this question also mentioned the desire for multiplayer. A variation of multiplayer was discussed during development and there are plans to implement this in the future. It would include having the robot act as a companion to someone who is playing and streaming the base game of Blastronaut. The idea is to have the companion be able to influence the gameplay of the streamer and vice versa. This could then take the form of them assisting each other, or having the two be in conflict.

The next question asked how interested they were in the base game of Blastronaut now that they had played the integration version. This was a multiple choice question with 5 different options to choose the answer from: “will go wishlist the game on Steam”, “Will go look at the Steam page”, “Might go visit the Steam page”, “Slightly interested”, “Not interested at all”.

The answers to this can be seen on Figure 28. Every person left from the stream with some interest toward the base game. Three people answered they will go wishlist the game on Steam. Two people said they will go look at the Steam page and another two answered that they might go look at the Steam page. The final person just answered that they were slightly interested in it. These results indicate that there is much potential for the integration to be used as a way to gain new players.

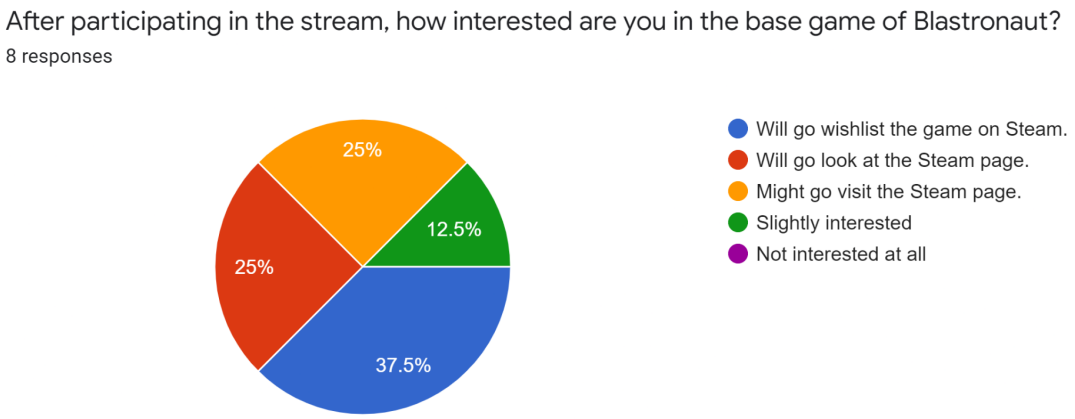

Figure 28: User answers regarding their interest in the base game of Blastronaut.

For future testing, the plan is to stream the game for a prolonged period of time and to possibly have multiple different streams active at the same time as well to see the results. The companion robot will also be tested in the future if development for it turns out to be a success.

6. Conclusion

The purpose of this thesis was to develop a live stream integration for the game Blastronaut as a way to gain new players for the game. Firstly, an overview of live streaming and Twitch was provided. Various other live streaming platforms were also introduced and the reasoning behind choosing Twitch was given. Different types of live streaming integrations were then explained with examples of how they have been used in games.

Creating the twitch integration for Blastronaut entailed the development of an in-game robot which can be controlled through a Twitch chat. This first required the creation of a connection between twitch and the game. Due to the differing way of inputting commands to the character, many mechanics had to be redesigned and some new ones also needed to be added. The main mechanics in question were the timer, movement, shooting, ores, tasks and the station. A way to broadcast chat messages inside the game was also added to encourage interactivity between viewers.

Testing results were largely positive, as the majority of testers had no major issues with any of the mechanics. Neither the movement nor shooting felt difficult to anyone. The station and its shop also felt intuitive to interact with. Message broadcasting was also something everyone enjoyed. Regarding the overall experience, users answered that they found the streamed game to be fun and that they now had an interest toward the base game of Blastronaut. While there are still some changes which should be made based on the feedback, the core concepts and ideas are present and were viewed positively by the testers.

Before the final release of the base Blastronaut game, the idea is to stream the integration version for prolonged periods as a way to attract new players. This could also act as a gathering point or a discussion topic among existing Blastronaut players. In future development, a solution to have the robot be a companion to the streamers could also be added.

References

[1] Mark R.-J., Jamie W. The impacts of live streaming and Twitch.tv on the video game industry: *Media Culture & Society*. 2018.
http://doi.org/10.1177/0163443718818363

[2] Seth G., Nathan M., Joseph S., Rachel M., Jessica H. Design Challenges for Livestreamed Audience Participation Games. 2018.
https://dl.acm.org/doi/abs/10.1145/3242671.3242708

[3] Christopher B. The Ultimate List of Live Streaming Statistics for 2022. 2022.
https://findstack.com/live-streaming-statistics/ (9.05.2022)

[4] Jan de W., Alicia van der K., Joep T. Live Streams on Twitch Help Viewers Cope With Difficult Periods in Life. *Frontiers in Psychology*. 2020.
https://doi.org/10.3389/fpsyg.2020.586975

[5] Number of hour watched on leading gaming live stream platforms worldwide in 3rd quarter 2021, by platform. 2021.
https://www.statista.com/statistics/1030795/hours-watched-streamlabs-platform/ (9.05.2022)

[6] Mike R. Twitch parent company rebrands as Twitch Interactive. 2014.
https://www.gamedeveloper.com/business/twitch-parent-company-rebrands-as-twitch-interactive (9.05.2022)

[7] Timothy A. Twitch for Game Developers. 2019.
https://blog.twitch.tv/en/2019/03/29/twitch-for-game-developers-b664163bed65/ (9.05.2022)

[8] David L. AUDIENCE PARTICIPATION GAMES: Streamer and viewers' engagement in the audience participation features of Choice Chamber and Dead Cells. 2019. http://dx.doi.org/10.13140/RG.2.2.19788.77441

[9] Dennis R.,Jenny S., Jeremy D. Twitch Plays Pokemon: A Case Study in Big G Games. 2014.
https://www.academia.edu/15039351/Twitch_Plays_Pokemon_A_case_study_in_big_G_games

Appendix

I. Source Code

The source code and its corresponding files can be found in the private repository: https://gitlab.com/perfoon/blastronaut/-/tree/HansDevelopment. For familiarising yourself with the source code, you can get in contact via email: hans.matthias.a@gmail.com

II. Survey

The feedback survey can be found inside the attached zip file under “survey.pdf”.

III. Licence

I, Hans Matthias Andreas,

1. grant the University of Tartu a free permit (non-exclusive licence) to: reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

   Twitch Integration for Blastronaut game,
   supervised by Jaanus Jaggo.

2. I grant the University of Tartu the permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work from ***10/05/2022*** until the expiry of the term of copyright,
3. I am aware that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Hans Matthias Andreas
10/05/2021