UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Jens-Stefan Mikson

Virtual Reality Game Design Analysis Based on Tribocalypse VR

Bachelor's Thesis (9 ECTS)

Supervisor: Raimond-Hendrik Tunnel, MSc

Tartu 2017

Virtual Reality Game Design Analysis Based on Tribocalypse VR

Abstract:

This paper provides an analysis for virtual reality game developers, describing problems and solutions regarding virtual reality game design. Game aspects analysed in this thesis are level design, graphical user interface and game mechanics. Analysis is based on a virtual reality game Tribocalypse VR whose development the author of this work was part of. The game was developed during the creation of this thesis. For assessing the quality of features of the game, a measurement of *immersion* and *clarity* is proposed and used.

Keywords:

Virtual reality, computer game design, computer game development, computer game, analysis, immersion, clarity, Unity

CERCS:

P170: Computer science, numerical analysis, systems, control

P175: Informatics, systems theory

Virtuaalreaalsusmängude disaini analüüs Tribocalypse VR näitel

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärgiks oli analüüsida virtuaalreaalsusmängude disaini omapärasid. Uurimise alla võeti järgmised disaini aspektid: mängude tasemete ehitus, graafiline kasutajaliides ning mängumehaanikad. Aspekte analüüsiti mängu Tribocalypse VR põhjal, mis valmis antud lõputöö käigus. Erinevate disainide kvaliteeti hinnati selle põhjal, kui kaasav ja arusaadav see mängija jaoks oli. Tulemustest selgus, et mida arusaadavam ja hõlmavam oli aspekt, seda parem oli ka kasutaja kogemus. Samas leiti ka olukordi, kus mingi aspekti hõlmavamaks tegemine vähendas selle arusaadavust. Sellistes olukordades oli oluline leida optimaalne lahendus, et antud disaini aspekt oleks nii piisavalt hõlmav kui ka arusaadav.

Võtmesõnad:

Virtuaalreaalsus, arvutimängude disain, arvutimängude arendus, arvutimäng, analüüs, hõlmavus, selgus, Unity

CERCS:

P170: Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

P175: Informaatika, süsteemiteooria

Table of Contents

1. Introduction

In the last couple of years the popularity of virtual reality (VR) has increased and the speed of this increase seems to be on the rise [1, 2]. Virtual reality has both entertainment and non-entertainment uses in fields such as gaming, medicine[1,2], sport[3,4], architecture[5]. VR helps reduce the cost of actions that would otherwise be too expensive or dangerous to perform. Examples include trainee fighter pilots and surgeons. Virtual reality allows us to take virtual risks in order to gain virtual experience very similar to real life experience. Even though virtual reality itself is not a new concept [3], it is only now that the technology allows the VR developers to transform their imagination into immersive products that are available to a wide audience [4]. The immersiveness is sometimes also called presence [5], which describes the extent to which the VR media can represent the real world. Since modern immersive VR is only now starting to become more widespread, the design patterns and user experience paradigms of VR are still unexplored. There is still very much to discover in terms of which design choices work in VR and which do not. This means that guides on how to create good VR experiences are of great help to the VR game developers[6].

The aim of this thesis is to provide an analysis of virtual reality game design. Thesis will point out problems regarding game mechanics, level design and graphical user interface. Possible solutions for the described problems are presented. This work uses a virtual reality game Tribocalypse VR (TVR) [6] as the basis of observation and analysis. Ideas and solutions presented in this work became evident during and after the development of TVR in which the author was involved in (project lead, programming) and considers as part of this thesis.

Chapter 2 describes virtual reality in more detail after which chapter 3 will introduce and describe Tribocalypse VR. Chapter 4 introduces a way of measuring the quality of each element in a game using immersion and clarity. This way of measuring will be used to analyse TVR. Chapters 5, 6 and 7 go into detail when describing the problems and solutions encountered when developing and designing VR games. These three chapters will be heavily based upon the TVR game, providing many examples. Chapter 5 will describe how

[1] http://www.techrepublic.com/article/10-ways-virtual-reality-is-revolutionizing-medicine-and-healthcare/
[2] http://medicalfuturist.com/5-ways-medical-vr-is-changing-healthcare/
[3] http://sportsworld.nbcsports.com/virtual-reality-sports-arkansas-kentucky/
[4] http://promoovertime.com/virtual-reality-sports-sports-marketing/
[5] https://medium.com/studiotmd/virtual-reality-uses-in-architecture-and-design-c5d54b7c1e89
[6] https://en.wikipedia.org/wiki/Video_game_developer

to create a believable world for the player. We explore the problems and solutions regarding graphical user interfaces for VR games in Chapter 6. Chapter 7 will describe how the items in TVR were made interactable and will give insight to the problems and solutions of developing each item in great detail. The 8th chapter will provide an overview of the accomplished work and provide a final conclusion. Glossary, with descriptions for concepts used in this thesis, can be found at the end of this work. Appendix has information about HTC Vive (I), TVR's development team (II), levels (III) and enemies of TVR (IV).

2. Virtual Reality

Our senses are what allow us to experience the reality around us. The experience of reality is based upon the sensory information we receive and then process in our brains. “Virtual reality (VR) typically refers to computer technologies that use virtual reality headsets to generate the realistic images, sounds and other sensations that replicate a real environment or create an imaginary setting” Wikipedia [7].

2.1 Motion Sickness

It is important for a VR system to provide a seamless experience and provide appropriate responses in real time as the user explores their surroundings. Problems arise when there is a delay between the person’s actions and the system response which then disrupts the experience. The person becomes aware that they are in an artificial environment and thus the user is not immersed in the virtual world. For example, when the user tilts their head in rea life but camera display in the VR does not change its orientation in response, the user will most likely experience motion sickness. This is due to the conflict between the vestibular system (balance system) in the user’s ears and their vision. The vestibular system sends a message to the user’s brain informing that the head is tilted, however the vision input says otherwise.

2.2 Hardware

There are many different devices, which are used to achieve a VR experience. These include headsets[7], gloves[8], bodysuits[9], remote controllers[10] etc.

The headset allows the user to look around the created virtual environments. These environments are mostly three dimensional and appear life-sized to the user. Some headsets are capable of tracking the user’s movement as well. This means if the user walks around in real life with the headset on, the user should also move in the VR environment accordingly.

The controllers can be used to track the user’s hand position, to allow the user to press buttons on the controllers for input and for sending haptic pulses (vibrations) to the user.

The peripherals that allow input from the user (gloves, remote controllers) can range from application to application. The analysis in this thesis is based on headsets that allow the

[7] https://en.wikipedia.org/wiki/Virtual_reality_headset
[8] https://manus-vr.com/
[9] https://futurism.com/teslasuit-full-body-suit-lets-feel-virtual-reality/
[10] https://virtualrealitytimes.com/2016/07/12/list-vr-controllers/

tracking of the user's movement in 6 degrees of freedom[11]; the input from the player is received through remote controllers.

Common hardware platforms for VR are HTC Vive[12], Oculus Rift[13], PlayStation VR[14], Google Daydream View[15] and Samsung Gear VR[16]. [8] For specifics about each device, see Table 1.

Table 1. Comparison table of different VR hardware platforms .[17]

	HTC Vive	Oculus Rift	PlayStation VR	Gear VR	Cardboard	Daydream
Resolution	1080x1200 per eye	1080x1200 per eye	1920x1080 per eye	1280x1440 per eye	up to 1440x1280 per eye	1440x1280 per eye
Refresh rate	90Hz	90Hz	90-120Hz	60Hz	60Hz	60Hz
Field of view	110°	110°	100°	101°	60°	90°
Head tracking sensors	Yes	Yes	Yes	Smartphone	Smartphone	Smartphone
Positional tracking	Lighthouses	optical	optical	No	No	No
360° tracking	Yes	Yes - requires extra tracker	Yes - requires extra tracker	No	No	No
Roomscale	Yes	Yes	No	No	No	No
Motion controllers	Yes	Yes	Yes	No - optional controller available	No	No - 1 remote
Sound	Yes - not built-in	Yes	Yes	No	No	No
Weight	555g	470g	610g	318g	220g	<100g
Wireless	No - wireless plugin available	No	No	Yes	Yes	Yes
Multiplayer ready	Yes	Yes	No - other players not in VR	No	No	No
Price	$799	$599 - $798 with Oculus Touch	$399	$100	$15	$79
Score	**10/12**	**9.5/12**	**7/12**	5/12	4/12	4.5/12

Green: 1 point
Orange: 0.5 point

[11] https://en.wikipedia.org/wiki/Six_degrees_of_freedom
[12] https://www.vive.com/eu/
[13] https://www.oculus.com/rift/
[14] https://www.playstation.com/en-us/explore/playstation-vr/
[15] https://vr.google.com/daydream/
[16] http://www.samsung.com/global/galaxy/gear-vr/
[17] http://strawberrycode.com/blog/virtual-reality/vr-headsets-virtual-reality/

The Samsung Gear VR, Google Daydream and Google Cardboard require a phone to work. A phone is placed inside the headset, and the phone's screen will act as a display for the user. Their downside is that they do not track the user's translational movement but only the rotation. This means that even if the player moves around in real life, he stays put in the virtual reality world. This often causes motion sickness and thus, the user prefers to be stationary in real world [9].

PlayStation VR (PSVR) has the ability to track the player's translational movement. The PSVR uses a camera[18], which is usually placed in front of the player, near the TV screen. This setup is similar to those used in Nintendo Wii's controller tracking[19]. The downside of this setup is that if the camera cannot "see" the headset or the remotes, the tracking is lost. This can occur when the player turns the headset away from the camera or if the back of the headset is covered with something. Due to this limitation, the experiences for those kinds of platforms should be designed in a way where the front of the headset will always be at least partially visible to the camera. This can be fixed by purchasing an additional camera or placing the camera in a very specific way[20].

2.3 Room-Scale Solutions

Oculus Rift and HTC Vive hardware allow for room-scale tracking, which means that these systems are able to track the player in a wider area and independent of the user's real world rotation. This allows the player to crouch, crawl, jump and spin around without having tracking problems. Thus creating a more immersive experience. Oculus VR requires at least 2 sensors for 360-degree tracking but only 1 is included in the Oculus Rift's set. Also when we started developing TVR we did not have remotes for the Oculus Rift. However, since we had access to the HTC Vive set, located at the Institute of Computer Science, and it allows 360-degree and room scale tracking (without having to buy any additional hardware), we chose it as the VR hardware solution for Tribocalypse VR. More about HTC Vive and its room-scale tracking can be found in Appendix I.

[18] https://en.wikipedia.org/wiki/PlayStation_VR#Hardware
[19] https://en.wikipedia.org/wiki/Wii_Remote#Sensing
[20] https://www.reddit.com/r/PSVR/comments/5a4p3u/never_lose_tracking_hack_360_degree_gameplay/

3. Tribocalypse VR

Illustration 1. Logo of Tribocalypse VR.

Tribocalypse VR is a VR game (see Illustration 1 of the logo above), which was made by a team of students and game developers. Full list of team members can be found in Appendix II. The game was in development for 6 months, from the beginning of August 2016 to the beginning of February 2017. The game is currently fully released on Steam digital distribution platform[21] for 6.99€.

3.1 The Game loop

TVR is a wave-defender game where the player's village (located in each of TVR's levels) is attacked by waves of enemies and the goal of the player is to repel these enemies using various items and game mechanics. In the game the Vive controllers are represented by hand models (see Illustration 2 below).

Illustration 2. Screenshot of player character's hand models in the game.

[21] http://store.steampowered.com/

The player can move around the levels by either moving around in real life or teleporting to fixed positions located in each level. To teleport, the player can cast a teleportation spell using the Vive controller's touchpad. The hands are also used for picking up, holding and throwing items and pulling the bowstring of the bow. Spellcasting will be covered in chapter 6 and items in chapter 7.

When the game is launched the player is placed inside the Home Village level, which is essentially a cave. The player is first taught to cast fire and teleportation spells. This is done by showing a tutorial text on big stone slabs (see Illustration 3 below).

Illustration 3. Tutorial stone slab.

The player can move between different locations in the cave using the teleportation spell. Three of these locations allow the player to select a different level to play in. One is for seeing the developer credits, which list the developers, and the last one is a tutorial zone. See Appendix III for full list of TVR levels alongside with their design documents.

In the tutorial zone the player can try using all the weapons available in the game: shield, bow, spears and bombs. The player can attack non-offensive enemies with the bow, spears and bombs or block incoming arrows whenever the shield is picked up. If player teleports to a level choosing location, the player can select a difficulty setting (easy, medium, hard). After the difficulty is selected, a new level is loaded. In each of the three playable levels there is a menu to start a game. The player is placed on a platform, which sits high on a tree trunk. On the platform there is a bow and some bombs. The player can navigate the menu using the bow (see chapter 5). After the game is started, a wave of enemies (fixed number) attack the village that the player is defending. The enemies have one of the two targets: the player or the totem pillar. The latter is present in each level. Enemies trying to attack the

totem pillar, make their way to it by navigating through the village, usually starting from a green portal outside the village. When they reach the totem pillar, they attack it. After a certain number of attacks (calculated based on the damage that is set for the opponent) is made, a piece will fall off from the totem pillar. Some of the enemies (minimum 1) will then try to deliver the piece to one of the portals. Others will keep attacking the totem pillar. If the piece carrier is killed by the player, then the piece is dropped and at least one enemy will try to find and deliver it. If all totem pieces are loose from the pillar (the pillar is destroyed) then enemies, who do not carry any totem pieces, will escort the carriers to their destination. If all totem pieces are delivered through the portals, the game is over for the player. If all enemies of the wave are killed before the pieces are delivered, the player wins the match and can continue playing the next wave by activating a corresponding button on the menu. The next wave is harder than the previous one up to a cap of 30 levels. The settings of the 30th wave is used for every consecutive wave. For example, there is no difference between waves 30 and 41.

There are also 2 types of enemies who can attack the player by shooting arrows with a bow. If the player gets hit by the arrow then any items that are currently equipped by the player will be dropped, the display will be rendered in grayscale and the player is unable to pick up any new items. This state will last for 5 seconds. More about the enemies of TVR can be read in Appendix IV.

To defeat the enemies, the player can use 3 offensive weapons: the bow, spears and bombs. The bow requires two hands to use and spears and bombs require one. Spears and bombs are throwable weapons. The player can also use a defensive shield to block any incoming arrows fired by the enemies. Weapons will be discussed in detail in chapter **7**.

For each kill the player receives score. This score can be used to purchase additional bomb slots in between the waves. Highest scores of each player will be saved and displayed in Steam's global leader board. These leader board scores are unique in each level and difficulty setting. The player can receive extra score for performing certain actions such as killing multiple opponents in a short amount of time or receiving more score the further the opponents are from the player at the time of killing etc.

3.2 Used Technologies

Tribocalypse VR was developed using the Unity3D game engine[22], modelling was done using Blender3D[23]. 2D art, such as textures and marketing material (posters, banners), was created using Adobe Photoshop[24] and GIMP[25]. These programs were used because the developers had previous experience with them. Alternative programs to use would be for example Unreal Engine[26] as a game engine, Maya[27] as a 3D modelling software and Sketch[28] for 2D material.

[22] https://unity3d.com/
[23] https://www.blender.org/
[24] http://www.adobe.com/ee/products/photoshop.html
[25] https://www.gimp.org/
[26] https://www.unrealengine.com/what-is-unreal-engine-4
[27] https://www.autodesk.com/products/maya/overview
[28] https://www.sketchapp.com/#2

4. Immersion and Clarity

Spatial presence and flow are considered by Weibel and Wissmath in their article "Immersion in Computer Games: The Role of Spatial Presence and Flow" (2011) to be the key concepts for explaining immersive experiences. Spatial presence refers to the sensation of being there in a mediated world, whereas flow rather refers to the sensation of being involved in the gaming action. They also state that the clarity of player's actions has a big role when trying to increase the flow of a game [10]. This is the reason why in this thesis clarity, alongside with spatial presence, is used as a part of a measurement tool for assessing the quality of certain game aspects of TVR. A feature that has a good level of clarity means that it feels natural for the player and causes no confusion. Even though the article by Weibel and Wissmath considers immersion to be a combination of spatial presence and flow, this work hereinafter uses the term *immersion* to refer to spatial presence alone. This is due to the fact that when players usually describe the games they play as immersive, they mean in fact the spatial presence [11].

Thesis proposes a measurement tool of immersion and clarity. That tool is used to analyse each design decision, problem and solution in this thesis. For each game element it is described how immersive and clear an element of a game is. Every statement about immersiveness or clarity is objectively and logically explained. Usually when an element of a game has little clarity, the players will feel confused. Likewise, if the player is not immersed the game the game will mostly likely be boring for the player (see Illustration 4 on the left). The measurement of immersion and clarity can be used for many aspects of a computer game. Levels of a game can be described as immersive if they feel consistent. Level's clarity can be measured based on how well the level guides the player towards his goal. Graphical user interface can be immersive if it fits well into the game world; it should also be easy to use (clarity). Level design, graphical user interface and game mechanics will be analysed in the following three chapters

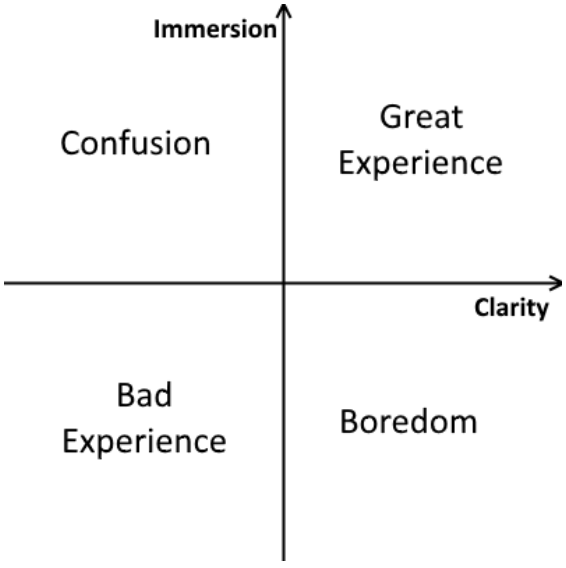

Illustration 4. Graph showing the impact of immersion and clarity on gameplay.

based on their level of immersion and clarity. The analysis will focus on VR, which means that the solutions proposed below might not work when designing non-VR games.

5. Environment and Level Design

The levels in TVR were designed with immersion and clarity in mind. The players must feel as if they are part of the created world. Game developers and artists work hard to provoke this feeling in the players. HTC Vive helps a lot in this area by adding immersiveness. The art style does not need to be realistic for the player to feel immersed in a VR game. This means that levels can look either realistic or cartoony or stylized in another way[29]. The goal should be to create a world that looks consistent.

The levels should guide the players toward their goals. In TVR the goal of the player is to defend the village from raiding enemies. For the level to guide the player it is important that the player knows where the enemies are coming from and where should the player character be located. It was also important to make the levels of TVR interesting and not feel mundane after couple of play sessions. To achieve this, multiple paths for enemies were added. This forced the players to very often shift their focus from one place to another. It also gave more value to the teleportation system, which made the players choose a more strategic location depending on the situation.

In levels of TVR, the enemies spawn from big portal gates, a small distance away from the village. The portals, that are active and spawn enemies, glow green (see Illustration 5 below).

Illustration 5. Portal gate outside the tribal village.

[29] http://store.steampowered.com/app/448280/Job_Simulator/

Most of the players who tried TVR the first time (at public expositions[30] and at private testing) had problems locating the first enemies who attacked them. Often the players noticed the enemies only when they had already arrived at the village. When asked the testers why this was the case, the answers were all very similar: they did not know to look at the place where the enemies initially spawned even though the portals emitted light.

There were 3 main reasons we think the players did not spot the enemies:

1) The gates and the enemies were simply too far away and were visible only when the player knew to look for them and concentrated their vision further away from the village. Instead they often looked at the details in the village, which they were protecting. Also the gates and enemies were quite small on the background.
2) The spawning action of the enemies was not made clear enough to the player. The audio effect was too quiet and the visual effect too small.
3) The portal effect blended in with the rest of the world because every first wave on each level starts during daytime. A lighting change far away often goes unnoticed when that change does not produce enough contrast compared to the rest of the environment.

There are many possibilities to resolve this problem. The key is drawing the player's attention. This can be achieved by sound and visuals. For example, the portals could be placed closer to the village or the mountains around the village could be slightly pointed towards the portals, which helps guide player's vision. The lighting around the portals could be made brighter and the visual effect of the enemy's spawn bigger and more noticeable. Perhaps add a beam of light that falls down from the sky into the portal each time an enemy spawns, coupled with a loud enough sound effect. Whatever technique used, it is essential for the player to understand at any given moment, which part of the level to look at. A good book to read on the subject matter would be "An architectural approach to level design" by Christopher W. Totten [12].

Illustration 6. Tree platform, which the player stands on.

We also had to decide what would be the best place where to position the player. We concluded that we needed the player to

[30] TVR at GameOn 2016 - https://www.youtube.com/watch?v=A_zGhkgavrk

be above the enemies. This is when we came up with the tree platform design (see Illustration 6 to the above). The platform is at the height of 10 in-game units (10 meters) and has a radius of 2 in-game units (2 meters).

The upside of this design is that the players have a good overview of the whole level. However, we encountered a problem that some players were scared of heights. To mitigate this, we made the platform large enough and added spikes around the edges of the platform. The spikes were also slightly tilted outside. This gave players more courage to spend their time near the edges and even look straight below (see Illustration 7 below).

Illustration 7. Player looking down from the tree platform.

Although players have a good vista of the level from all the locations they can teleport to, each one either has a blind spot (an area not visible to the player) or the spot is ineffective against far away opponents. This kind of approach has two benefits. Firstly, the player has an overview of most of the level at each spot, which means the player can come up with a plan in any location. Secondly, the player is encouraged to move between different locations, adding diversity to gameplay, making the game experience more versatile. When designing levels in VR it is important to make them feel consistent. The players should also feel comfortable, no matter where they are located.

The next chapter will describe how to design graphical user interface in VR in such a way that the GUI would be easy to understand and feel immersive.

6. Graphical User Interface

The graphical user interface (GUI) in computer games usually consists of informative or interactable elements such as menu screens, score tables, buttons and levers in the game and many more. In this thesis, visual effects of the camera are also considered as part of the GUI. The GUI in TVR consists of four bigger elements:

1) The menu
2) Hand tattoo
3) Score and leader boards
4) Visual effects of the camera

When designing the GUI of TVR, the goal was to create an immersive yet understandable and intuitive user interface. In order for the players to start and exit the game, implementing a functional menu was the first priority among the GUI features.

6.1 The Menu

The menu in TVR is a system that consists of functional buttons that allow the player to either select the map, start the match, purchase additional bomb slots or exit and save the game. Initially we came up with 3 different designs.

The Crystal Design

Firstly, we tried using floating crystals as menu buttons. These would be placed in air, in a 360-degree circle around the player, around 10 game units away from the player (see Illustration 8 below). They were around 2.5 in-game units high in relation to the main platform the player stood on. In order to interact with the menu, the player had to fire an arrow at the crystal. Upon collision, the crystal shattered and the "button" was pressed.

Illustration 8. A view of the 360-degree crystal menu.

However, this idea had problems with clarity. Since they were floating almost at the height of most players' heads and around the player, they were quite difficult to spot. One of the reasons being that anything outside the central field of vision gets ignored easily [13]. Another reason for the player not seeing the menu was that the crystals were too spread out. Even if the player did see one of the crystals, the rest of them were out of the player' field of view, giving the player an impression that there was only one menu button present (see Illustration 9 below).

Illustration 9. Player viewing crystal button.

The clarity problem could have been probably fixed by placing the crystals a little bit lower and spreading them out less. A solution would be to add them into a half-circle of only 180 degrees (see Illustration 10 below). So that if the player looks at one button, he can see at least one other button next to the one he is currently looking at.

Illustration 10. A view of the 180-degree crystal menu.

The problem with this solution is that it is not very scalable. If all the crystals were to be at the same height, then the amount of the crystals inside the half-circle would be limited to 4

to 5 crystals maximum. This is due to the fact that the space on the half-circle would run out quite quickly. A solution would be to add multiple rows of crystals. Continuing with the thought of multiple crystal rows, we came up with the tree design idea where the buttons are placed at different heights. The tree solution turned out to be more immersive than the crystal design.

The Tree Design

The menu tree consists of a big tree trunk, which has hanging signs attached to the branches with rope (see Illustration 11 below). The player has to shoot arrows in order to interact with the signs, similarly to the crystal design. The tree menu design is more immersive compared to the crystal design. The model for the tree trunk is the same as for every other tree that we used in the game. And since those trees matched with the environment, the menu tree matched as well (even though it was much larger compared to the other trees). Secondly, the signs were also made of wood, which matched our art style. Thirdly, whenever the player shot at the sign, the sign would start to move and wobble due to the forces applied to it by the arrow. This was a small detail, which added a fair amount of immersion to the game thanks to its responsiveness to the player input.

Illustration 11. A view of the tree menu.

Clarity of the menu also improved a little bit. This was due to the fact that all the menu "buttons" were concentrated in one location. This means that when the players discovered the tree menu, they already knew to look for other buttons, simply because they were very near to the player's central field of vision. The signs were also placed in a way that the chances of the player hitting the wrong sign would be minimized. Functional signs at the same horizontal or vertical axis were placed far away from each other. Also, the branches

of the trees easily led the player's vision to other signs. The branches of the trees could be considered as navigation paths that guide the player's vision.

However, the problem with this design was that, similarly to the crystal design, players had hard time finding the menu. This was caused by three reasons. One reason was that the tree was placed quite far away from the player. The players usually expect to see important elements of the game, such as the menu to be close to them.

Second reason was that though we managed to create an immersive menu, we made it too immersive. The tree blended into the environment too well. One might think that the bright white text on the menu signs would be visible to the player and he could easily notice the menu. However, this was not the case and is the third reason why the player had hard time finding the menu. The size of the text was not a problem in terms of reading it. All the players were able to read the signs clearly when they discovered them. Before discovering the menu, the players probably did see the text on the menu buttons but did not think them to be important. According to the testers, this was due to the fact that the tree blended in too well with the environment and so did the text. A way to fix this might be to make the text stand out visually amongst other elements of the environment [14]. If the environment is mostly static the text could be animated. Attention could also be drawn to the text by pointing light or scene's geometry at it (see Illustration 12 below). Sound assisted attention guidance could also work. For example, an announcer or storyteller could point the player in the right direction.

Illustration 12. Using geometry and light to guide player's vision.

In order to interact with the menu (with the exception of the tutorial level), the player has to pick up the bow and shoot at one of the signs. Its advantage is that it adds immersion to the game. Disadvantage is that it lowers clarity.

Many players that tested TVR (especially those with first time experience with the Vive) tried to place their in-game hand on the button to select it, even though the buttons were around 20 in-game units away from the player (see Illustration 13 below).

Illustration 13. Player trying to press the menu buttons on the tree.

This was not foreseen because the chances of something similar happening in real life were very small. It is not common for a person to try to hit the light switch that is more than 20 meters away from the person. The reason for this was that the hand models in TVR are ~3 times bigger compared to the player (see Illustration 14 below).

Illustration 14. In-game hand models compared to real hands.[31]

[31] https://www.youtube.com/watch?v=d6Tp1BJ6kYM

Since the hands were so big the first impression of the players was that the hands were not actually big but instead very close to them. This, for some reason, made them believe that the menu signs were also close to them. The problem seemed to occur less after the players looked around and got more used to being in VR. The problem also did not occur a single time amongst players with previous experience in a VR world.

Another problem encountered while testing was that the players who were not used to aiming with the bow, had problems hitting the wooden signs on the tree menu. This is good in a sense that players can do target practise on the menu signs. If they cannot hit the signs, they most likely cannot hit the moving enemies either. On the other hand, this also caused some frustration amongst some of the players. This was especially the case when the players were at public events and they had audience in the form of other players waiting to try out the game. The frustration was most likely caused by the fact that the players did not want to look bad in front of other players by not being able to hit a menu button. Tutorial level, which has a practise range, is one of the ways to solve this problem. The player can practise hitting static targets, which are not part of the menu. Unlike the menu signs, the targets at the practise range are meant for practising. This was not tested but it can be hypothesised that at public events, the frustration could be reduced by having the player hit targets at a practise range, rather than let them practise on the menu buttons.

The third menu design takes into consideration the fact that players had difficulties with spotting the menu and hitting the signs. Thus, the idea behind the third design was to bring the menu closer to the player.

The Stone Slab Design

The stone slab design is used only in the tutorial level. It was not tested in other levels due to time restrictions caused by the launch of the game. This design uses small stone slabs as buttons that the players can press down with their in-game hands (see Illustration 15).[32]

[32] Pressing stone slabs - https://youtu.be/m_RgxynuvDY

Illustration 15. Slideshow of player pushing the stone slab. Top left – hand is not touching the stone slab. Top right – hand is touching the stone slab. Bottom left – button is pressed midway. Bottom right – button is completely pressed down and thus activated.

The benefit of using this design is that it is immersive and natural to use. The act of pressing a button is intuitive for the players and is more clear than having to shoot arrows at menu signs to activate those buttons. The disadvantage of this design is that it requires space in an area where the player can move. This means the stone slabs would have to be placed sparingly if used in other levels other than the Home Village. Perhaps they could be fit onto the platform that the player stood on. This would however decrease the play area of the player, which is already quite small. Another option would have been to create a separate teleport location for pressing the buttons. This has the potential to reduce clarity because it requires the player to teleport to the menu area each time the player wants to press the button. This would be worsened if the menu area provided no strategical location nor weapons for the player to fight against the enemy. With the tree design, it is theoretically possible to activate the buttons from any location and at the same time battle with opponents. Another option would be to create a separate location in each level for the stone slabs. This would be the menu area, which can be accessed by the player only in between the waves. If the

wave ends then the player is automatically teleported to this area. If the wave starts, the player is automatically teleported away from the area.

The next menu design tries to solve the problem of travelling to certain teleport locations to interact with the menu while also bringing the menu even closer to the player.

The Doll Design

The fourth menu design was planned to be in the shape of a voodoo doll (see Illustration 16 below). This was inspired by the voodoo doll weapon from the previous unfinished Tribocalypse game[33] where the player could attack another player from a range. The player could choose, which part of the doll to hit with a needle: the head, chest or shoulder. Each body part had its own special effect when hit, that was applied to the target player.

Illustration 16. The doll model.

Using the doll as a menu would bring the menu closer to the player, making it more accessible and thus increase overall clarity. The menu interaction would not be skill based. As was observed when letting players test the game, they tried to place their hand on the menu signs. Using the doll design, this would have worked naturally and intuitively. Also, the doll would not have broken immersion, since it fit with the art style of TVR. However, the doll was not developed and thus, was not tested. In hindsight, this might have been the better solution compared to the previous ones.

When trying to bring this design to life, certain design questions need to be answered. Does the doll act as any other pickable item in the game? Or is it in player's hand by default when a level is loaded? How does the player interact with the doll once it's picked up? Does the

[33] http://blog.tribocalypse.com/index.php/about-tribocalypse-pc/

player use Vive controller's touchpad to select the body parts to activate or does the player have a needle in the other hand, which has to be pointed at a certain body part? How to make the doll scalable? These questions and probably many others need to be answered when developing this kind of menu system.

Even though we did not develop the doll design, we did implement a similar feature, for casting spells, which will be the next topic.

6.2 The Hand Tattoo

The hand tattoo is a GUI element, which is placed on top of the player's in-game back of the hand (see Illustration 17 below). Its function is to allow the player to cast magic spells. Currently, two spells are included: fire and teleportation. The player can use the fire spell to light either the tip of an arrow or the tip of the spear. This helps the player light up dark environments. The teleport spell is used to move between fixed locations around the map.

Illustration 17. The hand tattoo.

The spells can be cast at any time during play when the casting hand is not holding an object. The fire spell can be cast on both hands simultaneously, the teleport spell can be cast with only one hand at a time. To cast a spell, the player must press, hold and release (this will be explained below in detail) the correct area on the touchpad. The touchpad's top half area is used for teleportation, the bottom half area for the fire spell. The touchpad can be seen as a 2D grid with the x and y axis, both ranging from values -1 to 1. The centre of the touchpad is also the centre (zero point) of the grid. In code, if the touchpad detects a press, it is possible to get the exact location of the player's finger on the touchpad. In TVR's case, the grid is separated into five different areas: (see Illustration 18 below).

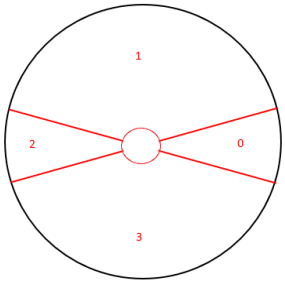

Illustration 18. 4 sectors mapped on touchpad.

The left, right, top, bottom and middle. The reason for creating an empty area in the middle was to encourage players to move their finger in the extremities of other areas. If the player put their finger in the middle of the touchpad, up to 4 spell areas could easily overlap (before creating the empty middle

section). This was confusing and thus reduced clarity of the spell casting system.

Initially we planned to add 4 spells to the game. This meant mapping 4 inputs on the touchpad, essentially slicing the touchpad into 8 imaginary sectors. In our first iteration the spells could be cast when the correct area of the touchpad was pressed and held down without a pre-cast effect that would be caused by simply touching the touchpad. The problem with this was that the players did not know for certain, which spell they were going to activate before they pressed the trigger. It is quite difficult, if not impossible, for the players to see where the boundaries of the spell casting areas on the touchpad are. This resulted in players frequently casting wrong spells. Furthermore, new players not familiar with Vive thought that the touchpad was one large button. They thought it did not matter on which part of the touchpad they placed their finger on.

The touchpad's touch functionality (see Appendix I for a reminder of HTC Vive's controllers) was used to fix the problem of casting undesired spells. If the player touches over the touchpad without pressing down, it is seen which spell will be cast when pressing the touchpad down. See Illustration 19 below.

Illustration 19. Slideshow of player casting the fire spell. Top left – touchpad not touched. Top right – touchpad touched. Bottom left and right – touchpad pressed.

The combination of touch and press mechanics can both reduce and increase clarity. It increases clarity by helping players cast the correct spells. However, often when players were hovering over the touchpad and the pre-cast effect started playing (the small fire for example), players often thought that this effect was the spell at its maximum effect. They then tried to light arrows and spears with the small flame effect without pressing the touchpad down. This caused confusion because they already saw the fire effect and thought that this was enough to use the spell.

Casting the teleport spell proved to be problematic as well. If the player wants to teleport somewhere they have to:

1) Press touchpad's top sector
2) Point towards teleport area
3) Release the sector while pointing towards the teleport area

If the player touches their finger over the teleport segment of the touchpad then a grayscale effect will be applied to the camera, making player camera's colouring scheme use a grayscale effect (see Illustration 20 and Illustration 21 below). The only exception being the teleport spots, which will be lit with a blue light and decorated with particles flying towards the sky. This helps the player find the teleport locations. It should be noted, however, that using a blue colour on a black and white (grey) background might not be the best option of colour because it blends in with the background pretty well.

Illustration 20. Before casting the teleport spell.

Illustration 21. During the cast of teleport spell.

If the player holds down the touchpad, then at each frame, a capsule cast is done, which can hit a collider centered around the teleport spot. If collision occurs then a channelling effect is displayed and after every 0.1 seconds a weak haptic pulse in the controller is activated (see Illustration 22 below). Small particles will fly from the pointed fingers of the player's hand towards the teleport location. At any time during channeling, the player can teleport to the pointed location by releasing the touchpad.[34]

Illustration 22. Casting effect of the teleport spell. Leftmost – pointing. Middle – touching the touchpad. Rightmost – pressing and holding the touchpad.

The teleport mechanic's clarity also suffered from the touch and press mechanic. Players often assumed that when they touched the touchpad then they would be able to teleport and the channelling would work. This is caused by the small effect, which is shown similarly to the fire effect upon touching (see Illustration 23 below).

[34] Teleporting in TVR - https://youtu.be/M_sW86Seu64

Illustration 23. Teleport touch effect.

Another problem was pointing at the target location. We tried to make the capsule cast as wide as possible to make pointing as simple as possible. However, since the players usually do not look down their in-game hand when pointing, they often pointed their hands too high or too low. This means no collision happened and since they saw the preview effect at the tip of the fingers, they often became confused.

However, the biggest problem was that the players did not understand they needed to release the touchpad in order to complete teleportation. Those players who got to the phase of channelling (seeing flying particles and receiving haptic feedback) often thought that the teleportation would automatically happen after a small period of time. This was due to the fact that players are used to the channelling effects in other games[35], where in order to complete the channelling, the player does not need to add any additional input. In the case of TVR, however, in order to successfully complete teleportation, the player has to release the touchpad. This was also counterintuitive due to the fact that players' first thoughts were that if they were to release the touchpad while channelling, the channel effect would simply get cancelled.

In order to increase clarity for the teleportation spell, the channelling should automatically end in a success after a short amount of time has passed, rather than wait for player's input. This of course does not have to be the case for other types of spells. For example, if the player is able to cast a fireball that could be launched, then the hold-press-release mechanic could be quite useful. For example, a fireball could be charged up by holding down the touchpad. This would increase its damage and size. After reaching its maximum size, a sound effect might start to loop to indicate that the channelling has reached its final state. After reaching its final state the ball is not launched automatically. Instead, the player can aim the ball, and when ready, launch the ball by releasing the touchpad. The input pattern, which the player uses, should be consistent in order to not confuse the player.

In addition to the spell casting issues, visibility of the tattoos is another problem for newcomers. Since the tattoos are on the back of the hands, the players rarely notice them.

[35] http://dota2.gamepedia.com/Channeling

This is due to the fact that attention is withdrawn from the area of the visual field where the player's own hand is currently at [15].

While the used solution is immersive, it has little clarity despite the fact that it is taught in the Home Village.

6.3 Score and Leader Boards

In addition to the tree menu, there are two more GUI elements located in each map of TVR (except the Home Village). One of them is the wave end result board (score board) and the other one is a global leader board (see Illustration 24 below). Both blend in with the environment in order to be immersive and both are also clear about the data they represent. They were placed and sized accordingly so that they would be noticeable and the players would see what is written on them. It was important to give as much information to the players without overwhelming them. In our design we decided to use as many icons as possible since these are easier and faster to make sense of than plain text. Reading text in VR requires fonts to be large enough. This however can create problems with available space.

Illustration 24. Score board (left) and leader board (right) side by side.

The wave end result board contains statistical information about the previous wave. This information is displayed upon wave completion and it contains statistics about player's accuracy, long distance shots, combos and overall score. Different designs were created for the result board (see Illustration 25 below). First of these designs consisted of having

multiple boards, which would be much wider than the final result and would also display more data.

Illustration 25. Different result board designs.

Second idea was to create a board with a timer. Similarly to the previous design, this one would also display 3 different boards. However, with this design, only one of the boards would be visible at any time. For example, the data on the board would change every 5 seconds and this would loop until the level is changed or until the next wave starts. Thus overcoming the available space problem. However, if the player is at the first panel and wants to view information on the third one, he has to wait for 10 seconds.

The benefit of these designs is the ability to display a lot of information. Such as the exact amount of different enemies killed, accuracy with different weapons and so on. The downside of the first solution is that it would take up too much space. To make the statistics on the boards visible to the player, the boards would have to be scaled up quite a bit. Having three massive boards in the levels might draw too much attention, reducing immersion. The available space in the level might also become a problem. There were already problems placing all the GUI elements in the Stonehenge level due to that level's small size. Having 3 big boards would amplify this problem. Another solution instead of the timer would be to use a wooden sign to change the boards. This sign would work like the ones on menu trees, which can be activated when shot at.

The timer design could have been used, however in the end it was decided to simply use one panel, which would display only the most important statistics to the player. This was simple to read and also had no extra functionality, which would have increased the complexity of the GUI and potentially reduce the clarity of the system.

7. Items

Items that the player can pick up in TVR are spears, bombs, the bow and the shield. First three are offensive items, while the shield is the only defensive item in the game. To pick up any of the items, the player has to place their character's hand (using the Vive controller) on the item and press the trigger button on the Vive controller (see Illustration 26 below).[36]

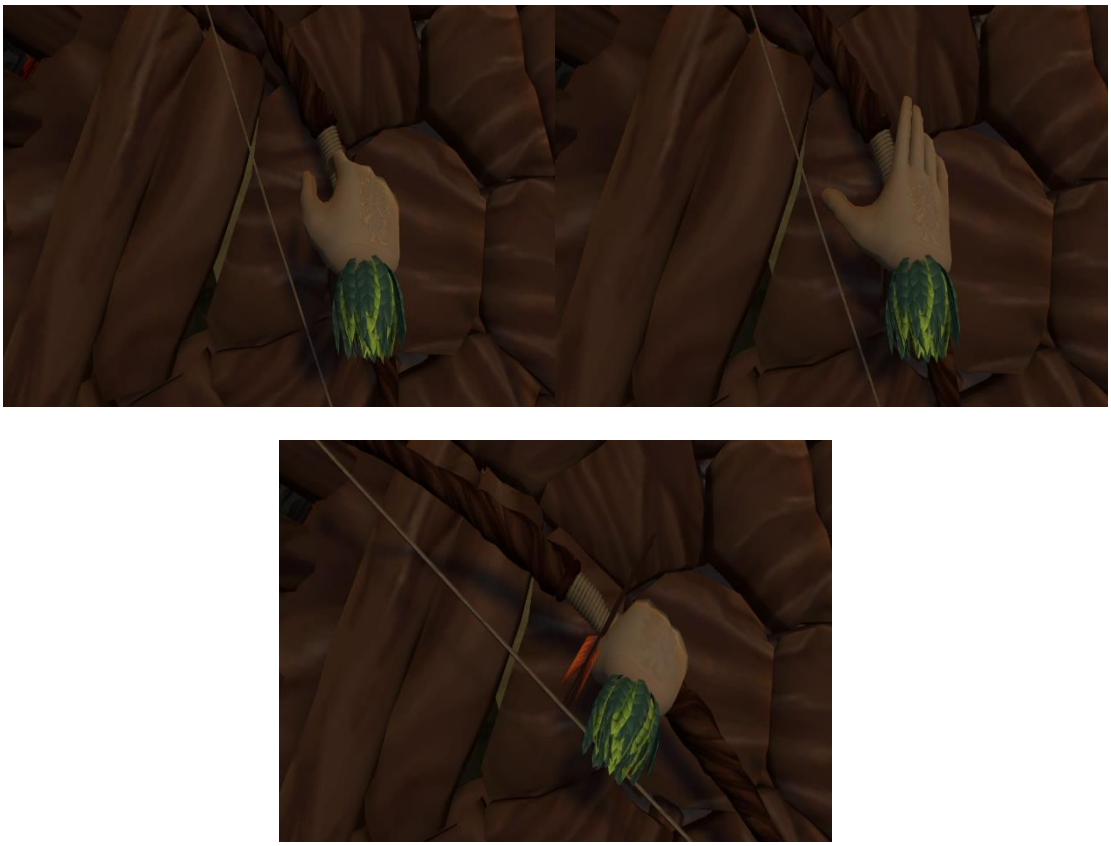

Illustration 26. Player picking up the bow.

Item handling system was designed to be ambidextrous. This means that players can pick up items with either hand. Another way to solve this, as some other games[37] do, is to let the players choose their dominant hand. They can do it when the game starts and when needed, change it later in the settings menu. This often results in two handed weapons (such as the bow) being placed in the player's dominant hand, regardless which controller (left or right) the player uses to pick up the item. This kind of solution however reduces immersion. It feels more natural to be able to pick up the weapon with the hand the player has currently placed on the weapon. If the player feels like the weapon is in the wrong hand, the player can simply drop the weapon and pick it up with the other hand. A better solution for

[36] Picking up items - https://youtu.be/P_vRz-YM-n0
[37] http://store.steampowered.com/app/400940/Budget_Cuts/

switching the weapon, would be to make the player able to simply take the weapon from the other hand, without having to drop it first. This solution should increase both the immersiveness and clarity. It is more natural, in real world, to pick the item from the other hand, without having to drop the item first.

All of the items have an attached rigidbody component. A rigidbody component makes any object in the game to be able to be subject to the physical forces acting in the environment such as gravity and collisions with other items. If an object in TVR is picked up, its own collider is disabled and each frame, the object sets its position to be equal to the weapon specific animation bone, located in the hand's hierarchy. Also, an invisible "follower" object is created. This invisible object has a collider and a rigidbody and also smoothly lerps to the picked up item's position. The purpose of the follower object is to make the held item interact physically with other smaller items when they collide. For example, this gives the player an ability to swipe items that are laying on the ground, such as bows and shields.[38] This small detail adds immersion the game.

The reason to not simply use the picked up item's collider is because collisions between rigidbodies in Unity works best if these rigidbodies are moved through physical functions. If the transform of a rigidbody is directly modified, then the collision calculations do not give expected results. This is because when manipulating the rigidbody's position only directly through the transform component, the rigidbody's internal velocity variable is a zero vector. If the collision happens however and the velocity is a zero vector, then the collision would be handled as if the collision happened with a static object. The problem however is that the player swings the object. To the player the object looks as if it is in motion. To the physics engine, since only the transform is modified, the object remains static. Through direct modification of the transform component the physics engine sees as if the object is simply placed or teleported to a different location. Now, when the player swings at object P, two outcomes are possible:

1) The player swings too fast and at frame *n*, the swingable object has not yet collided with P. If the player is very fast and the object that the player tries to hit is small, then at frame *n*+1 it is possible that the position of the swingable object has already gone past object P. No collision happens and it looks as if the swingable object simply went through object P.

[38] Swiping items on the floor - https://youtu.be/dMnfEKGQSbU

2) At frame *n*, object P-s collision box and the swingable object's collisions boxes overlap. Collision happens. However, to the physics engine in Unity, it seems as if the swingable object's collider simply appeared in the object P's collider. To solve this collision, the physics engine tries to push both items away from each other, often resulting in undesired force vectors. For example, the object P could fly in the opposite direction of the swing motion, due to physics engine detecting collision too late.

To fix this problem, a solution using the follower object was used. The follower object has a collider identical to that of the picked up item's collider. The following function uses physics to smoothly follow the object. This means that collisions work correctly because the follower object's rigidbody has a velocity greater than zero when moving, resulting in correctly handled collisions. The only downside to this is that the collision is slightly delayed compared to the visuals of the picked up object. Thus, for a very short amount of time, the swinged object could be seen overlapping with an object it should collide with. However, this amount of time is negligible to have any impact on immersion and clarity of the game.

Another option to solve the collision problem would be not to use the follower object and use the picked up object's own collider. And instead of directly modifying its transform the object could be made to lerp to the player's hand using the same function as the follower object does for lerping [16]. This solution proved to break immersion as the following of the object was too slow and it felt more like that object "swam" to the player's hand, rather than being a static object, held by the player. The faster the players moved their hands, the more evident the problem became.

If this solution was to be used, then the following improvement could be made to the technique: Add the weight of the carried item to the lerping formula. For example, items with heavier weight, follow the hands slower than items with smaller weight.

Another solution for adding artificial weight to the objects is to make the player lift up heavier items slower, and lighter items faster. This means if the player tries to pick up an item, a link is created between the player's hand and the pickable item. This link makes the item move towards the hand. The heavier the item, the slower it moves. If the distance between the item and hand becomes too large, the link is broken and the item falls to the ground [17].

7.1 Throwing

All items that can be picked up by the player, can also be thrown. This can be achieved by first picking up the item with the trigger button and then releasing the trigger button to throw the item. The faster the player moves an arm in real life (the faster the Vive controller moves), the further the item in the game is thrown. It was necessary to get the throwing mechanic feel right. It had to feel as close to throwing items in real world as possible. Throwing an item in real world consists of these elements:

1) feeling the item in one's hand
2) feeling the weight of the item
3) motion/trajectory and the speed of the hand
4) the act of throwing the item (releasing).

The feel of having something in one's hand is already accomplished by having the player hold the remote. Luckily, the shape of the Vive controllers is good enough to give an expression to the player that whatever they are holding in TVR, is also held in real life. It also does not seem to matter whether it is an item with a handle, such as the spear or shield or a round object such as the bomb.

Unfortunately, the weight that the player feels, is limited to the weight of the controller plus anything else attached to the controller. It is possible to tell the player to attach something to the controller in order to make the weight of a certain item in game feel more realistic. However, if this has to be done very often, adding and removing weight from the controllers breaks the immersion heavily, as it brings the player out of the game too often. This kind of approach might work if an extra weight has to be added only when starting the game. Some people has even suggested to add weights in the form of bracelets and hand gloves[39]. By not being able to modify the weight of the controller automatically through the game somehow, it was necessary to use other techniques to make the throwing feel as natural as possible.

When throwing an item in TVR, the velocity of the controller is multiplied by a force multiplier (hard coded float value). This velocity is then added to the throwable item's rigidbody. Modifying the force multiplier gave us the ability to fine-tune the throwing mechanic to feel as real as possible. All of the items in TVR used the same formula. This means, their in-game weight was not included in the calculation:

[39] https://www.reddit.com/r/Vive/comments/55soc4/best_way_to_add_weight_to_controllers_for_more/

$$itemVelocity = forceMultiplier \cdot controllerVelocity,$$

$$itemAngularVelocity = forceMultiplier \cdot controllerAngularVelocity.$$

However, the used solution has a drawback. Regardless how big the item in the game is or how heavy it looks, the throwing force is only affected by the motion and speed of the Vive controller. For example, throwing the small bomb with force *N* and throwing the big shield with force *N*, without any obstacles in the way, both of these objects would land at the same location. This is clearly immersion breaking, since the lightweight bomb should fly much further than the heavier and more air resistant shield. In order to overcome this problem, the weight of the throwable items could be added to the formula. This means that items with different weights when thrown, in a same way (using the same motion and speed of the hand), would land at different locations. An improved formula would be:

$$itemVelocity = forceMultiplier \cdot \boldsymbol{itemMass} \cdot controllerVelocity.$$

7.2 Weapon Design

In TVR the main goal was to design the weapons in a way that each weapon would be more efficient in certain situations than other weapons. This means that the players have to use weapons strategically. That would add more variety to player's actions and thus more content to the game.

An improvement of this would be to add weapons, which are effective in the same situation and only that situation. For an example: Weapons A and B are effective in situation 1, weapons C and D are effective in situation 2. Whenever the player encounters situation 2, he will most likely choose between weapons C and D. This gives the player 2 options for each of the situations. However, if the player only had weapon A for situation 1 and C for situation 2, then the variety of weapons would only come into play when the situations change. However, having 2 weapons for each situation, the variety of weapons could be maintained even if the situations do not change.

In VR this design thought could be improved even further. While in PC games only a mouse button is used for firing a weapon, in VR the whole hand movement could be considered. This is especially useful for weapons, which do not have a trigger in real life, e.g. melee weapons, throwable weapons, bows etc. Those can be immersively emulated in a VR environment requiring the user to mimic the real life actions for use of the weapons.

This also gives the VR developer a chance to produce a small amount of situations but many different weapons, which are effective for those situations and still make the player feel as if there is a lot of variety in weapon strategies. It should be noted that if many weapons are used for only a single situation, then these weapons should have no clear advantage over one another. The bow and spear in TVR are a good example of this design choice. They both do the same amount of damage, have a very similar attack rate and when used against enemies, are both very accurate. However, the hand movements for using them are completely different. While the spear requires the player to make a throwing motion, the bow requires the player to make a pulling motion with their hands. The variety has to come from using these weapons differently, rather than just having different weapon visuals.

7.3 Single Target versus Area of Effect Damage

Even after making the action of throwing feel as natural as possible, there was still the problem of throwing items accurately. This is not a problem when throwing items, which have no effect on the enemies, such as the bow and the shield. These kind of items are not meant to be thrown at specific targets. It becomes a problem when designing a weapon that is meant to kill the enemy by throwing the weapon at him. The first throwable weapon designed was an axe. The idea was to create a weapon that could be thrown very fast in a rapid succession with each hand. It was very difficult to throw the axe accurately even after we made the "weight" of the axe feel right. Even after a lot of practise, the thrown axes would fly a little bit to the left, right or over the opponent's head. This problem was also amplified by the fact that the axe was very small and the enemies were always quite a distance away from the player. However, since we still wanted to implement a throwable weapon, we came up with the idea of creating an area of effect damage weapon.

7.4 The Bomb

Illustration 27. The bomb.

The bomb (see Illustration 27 above) is an explosive item that could be thrown. On impact the bomb detonates, dealing damage to every enemy within its explosion radius of 7 in-game units (7 meters).[40] This way, the bomb does not have to directly hit the enemy, giving the player room for mistake. This reduces irritation caused by inaccurate throws not hitting a single enemy.

The idea was to make the bomb extremely efficient against groups of enemies while being useless against flying enemies. Due to the long usage cooldown of 30 seconds, the bombs have to be used sparingly. It is wise, in most cases, to bombard groups of enemies. Similarly it would be unwise to use a single bomb or even more on a few number of enemies. We tried reducing their cool down time to nearly 0 seconds to make them usable the whole time, however they simply became too powerful and reduced the challenge of the game.

Later in the design process, we came back to the idea of having a single target throwable weapon. Thanks to our earlier experience we were already aware of the inaccuracy problems, present in the virtual reality environment when throwing items.

[40] Bomb Explosion video - https://youtu.be/D9c4huDa10M

7.5 The Spear

When designing the spear (see Illustration 28 on the right), we had two goals in mind. Firstly, we wanted to create a single target throwable weapon, similar to the axe described earlier. Secondly, we wanted to give the player an alternative main weapon (like the bow). This means that the player could either use the hold and pull mechanic to kill most of the enemies (the bow) or use throwing as the main mechanic. Since bombs have a usage cool down and are not usable through the whole match, they cannot be the main weapon. The main problem with designing the spear was making it accurate.

Illustration 28. The Spear model.

Aiming mechanics

First, we tried the same method as with throwing the axe: adding no aim assist and hoping that since the spear was bigger than the axe, the aiming would become a simpler task. To try this design, we rotated the spear in different ways on the hand, trying to get the best orientation. Orienting the spear in a way that it was perpendicular to the controller, resulted in most of the spears being thrown to the ground. Since the hand motion of the player starts from the back of the head and ends near the waist, the spears were often released too late in the motion of throwing. This resulted in the spears being driven straight to the ground. After that we tried to rotate the spear parallel to the controller. This meant that the player was forced to throw the spear as they would in real life. Bringing their hand to the back of their head and quickly leaning forward when throwing the spear. While this solution was better than the previous one, it still lacked accuracy on the horizontal axis and was also too demanding on endurance and stamina. Since we wanted to appeal not only to sporty players, this was not a suitable solution.

Third option was to make the player aim with their head.[41] This means that the spear always flew where the player was looking at. This way, the spear inherits only its speed from the controller but not the direction. While this solution was accurate when correctly used, it did not feel natural. Players would often tilt their head towards the ground when throwing the spear, making the spear fly straight to the ground, similarly to the first solution. Often they would also look in a completely different direction compared to the direction of their hand movement. Since this solution had very little clarity it was also discarded.

[41] Aiming the spears with the head - https://youtu.be/bSoGtsQBNFo

Rotating The Spear in Player's Hand

We decided to implement automatic rotation of the spear in the player's hand. The spear's orientation was perpendicular to the controller, pointing away from the player. When the spear is below the player's head, the tip of the spear is pointed in the same direction as the Vive controller. When the spear is above the player's head, it is turned around 180 degrees.[42]

The purpose is to rotate the tip of the spear in the direction where it will be thrown. Usually in real life, this kind of 180-degree rotation is done manually by the person holding the spear. In VR the designer should not expect the player to rotate their controller in real life, to get the correct orientation for the spear. Given solution reduces immersion because an item, which is in the player's hand is moved independent of the movement of player's hand. However, it adds clarity, by making the act of holding and rotating the spear as effortless as possible.

After adding automated rotation to the spear when held, the act of holding and aiming the spear was easier for the player. This means that throwing the spear in the direction of the enemies was not a problem anymore. It was still difficult to hit the enemies. The players quickly discarded the spear as a weapon because similarly to the axe, it was still too inefficient due to the lack of accuracy.

Aim Assist

To make the spear accurate, we decided to implement aim assist.[43] It was designed to work in a way that allows for greater aiming error. In that sense, the spear would work similar to the bomb, giving room for mistakes. The aim assist system consists of two parts: detecting the enemy and flying towards the enemy.

For detection, a capsule cast is used: a collider with the shape of a capsule is placed at the tip of the spear. This capsule has a length of 300 and a radius of 2 in-game units. If any enemies collide with the capsule collider, then the spear starts moving towards the closest enemy. The spear even detects enemies behind obstacles, such as walls or the terrain. In order to avoid this and detect enemies only visible to the spear, an additional ray cast could be made from the tip of the spear to the enemy. If nothing obstructs the ray, then an enemy is visible to the spear.

[42] Spear's automatic rotation video - https://youtu.be/p0Y0IKwS-p8
[43] Aim assist for the spears in action - https://youtu.be/bLTVTQM7eJA

If the enemy is detected, then he will be set as the target of the spear. From this point onward the spear's velocity remains a constant (equal to velocity at the moment of detection). This will remain true as long as the spear either hits something or the target enemy is killed. In each following frame the spear is given a new rotation so that it would reach its target. If new rotation cannot be calculated, then the spear retains its rotation.

In this context `rb` references the rigidbody component of the spear. First, a direction vector from the spear to the target is found.

```
Vector3 dir = target.transform.position - rb.transform.position;
```

If the magnitude of vector `dir` is greater than 2 and if the absolute angle between the `dir` and `rb.velocity` vectors is less than 90 degrees (the dot product between these vectors is greater than 0) then the calculation continues. Otherwise, if the spear is closer than 2 units, no further calculation is required because the spear will most likely reach the target. If the dot product is equal to or less than 0 then the target is considered to be located behind the spear and thus ignored.

In order to rotate the spear, the necessary Euler angles of the new orientation must be calculated. The found Euler angles will be `x`, `y` and `z` representing the rotation in the corresponding axes. Since Unity uses a left-handed coordinate system, the `x` represents the pitch axis (tilting forward and backward), the `y` represents the yaw axis (turning left and right), and the `z` represents the roll axis (tilting side to side). The `y` and `z` angles can be calculated by converting the `dir` vector to a quaternion and then extracting the Euler angles from that quaternion.

```
Vector3 euler = Quaternion.LookRotation(dir).eulerAngles;
float y = euler.y; float z = euler.z;
```

This is sufficient to rotate the spear in the target's direction on a *yz*-plane (see Illustration 29 below).

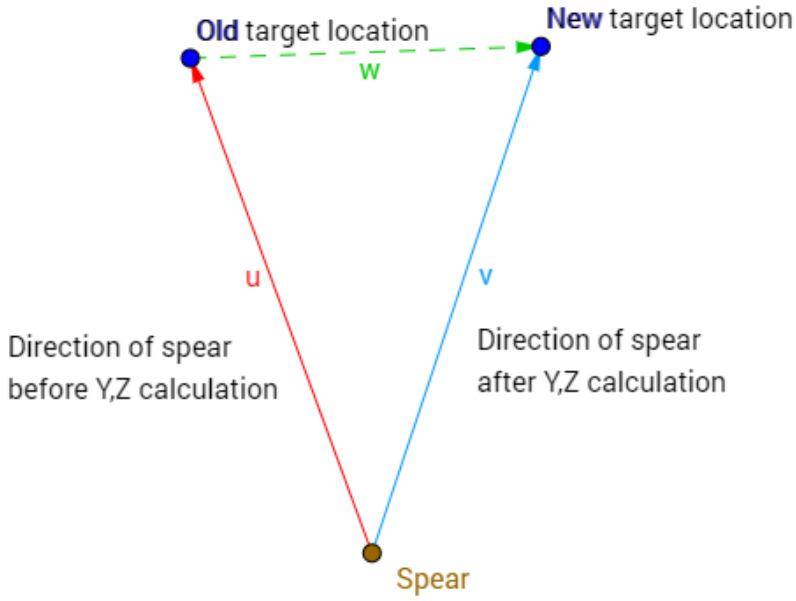

Illustration 29. Spear's new top-down orientation after new y and z angles are applied.

Next step is to calculate the x angle that is required to launch the spear at from its current position. The following formula will result in the necessary angle:

$$\theta = \tan^{-1}\left(\frac{v^2 \pm \sqrt{v^4 - g(gx^2 + 2yv^2)}}{gx}\right)$$

Where v is initial launch speed, g is the gravitational constant, x is target's distance (height is not taken into consideration) and y is height difference between two points. In TVR's case v is spear's rigidbody's velocity (constant), g is -9.8 units per second squared (default in Unity), x is the distance between the target and the spear (without calculating the height) and y is the height difference between the target and the spear.[44]

```
float distance = (new Vector3(targetPos.x, 0, targetPos.z) -
                  new Vector3(rbPos.x, 0, rbPos.z)).magnitude;
float height = targetPos.y - rbPos.Y;
```

In the calculation of θ only the root prefixed by the minus sign is used. If the root is imaginary, then no such angle can be found that the spear would reach its destination. Increasing the velocity of the spear enough would fix this problem. However, in TVR a constant speed was preferred. If the angle θ exists, then all necessary Euler angles for

[44] https://en.wikipedia.org/wiki/Trajectory_of_a_projectile#Angle_.7F.27.22.60UNIQ--postMath-00000010-QINU.60.22.27.7F_required_to_hit_coordinate_.28x.2Cy.29

rotating the spear are can be found and a new rotation quaternion can be created. Angle `x` will be equal to θ.

```
Quaternion newRot = Quaternion.Euler(x, y, z);
```

However, the spear's rotation is not immediately set to be equal to this quaternion. Instead, spherical linear interpolation (slerp) is used (see Glossary for information about slerp) to smoothly rotate the spear towards the new rotation. A coefficient (float from 0 to 1), which is dependent on the `dir` vector's magnitude, is calculated. This is used to set the speed of Slerp. The closer the spear is to its target, the faster the orientation is changed.

```
float coef = Mathf.Clamp(Time.deltaTime *
             (4.0f - 4.0f / dir.magnitude), 0.0f, 1.0f);
```

Finally, the spear's rotation is set based on previous calculations,

```
rb.transform.rotation = Quaternion.Slerp(rb.transform.rotation,
                        Quaternion.Euler(-x, y, euler.z), coef);
```

and the spear rigidbody velocity's direction is changed to make the spear fly forward in the new orientation. See Illustration 30 below.

```
rb.velocity = rb.velocity.magnitude * rb.transform.forward;
```

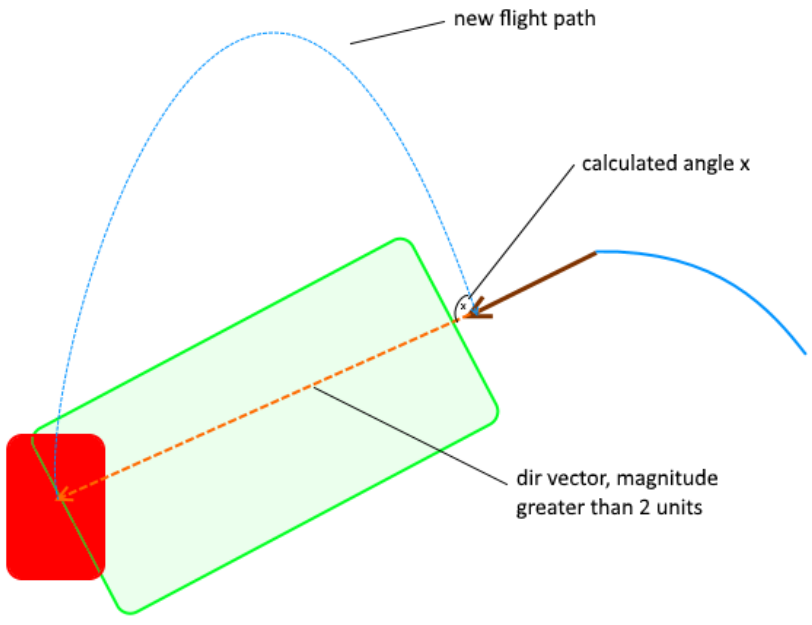

Illustration 30. Flight path of the spear. Includes the calculated angle x and the new exaggerated flight path.

The aim assist, described above, greatly increased the accuracy of throwing the spears. Thanks to this, the spear can now be considered to be on par with the bow. It has same damage, similar fire rate and most importantly, very similar accuracy statistics. This means that the spear and the bow could be used in similar situations, while neither necessarily overpowers the other by being more effective. The property that differentiates these weapons is the way they are used. Thus adding more variety to the game. The player has an option to either use the hold and pull mechanic or the throwing mechanic, as their primary mechanic to deal with the enemies.

7.6 The Bow

Illustration 31. The bow.

The bow (see Illustration 31 above) is the only weapon in TVR that requires 2 hands to use. It is a very accurate weapon in the game that launches arrows as projectiles. The player has to hold the wooden part of the bow with one hand and pull the bowstring with the other one.[45]

Beside the spear, the bow is the second main weapon in the game. It is accurate, has quite a high fire rate and moderate damage. Initially it was considered to be the main weapon of the game. However, after implementing spear's aim assist, the bow is now on par with the spear. Though testing shows that players still prefer using the bow instead of the spear. This might be due to the reason that it is much easier to hit the menu buttons on the tree, as the bow is much more accurate when trying to hit a certain point. The spear only becomes as accurate as the bow when the targets are actual enemies.

The bow is a feature that was the longest in development during the whole development cycle of TVR. The development of the bow started at the beginning of the project and was

[45] Player pulling the bowstring - https://youtu.be/rASrQmV_ohU

not totally finished even at the end of the project. The bow became functional in a month, however continuous testing, modification and bug fixing was required. The main problem was the inaccuracy of the bow. Many testers pointed out the same problem: "The bow doesn't feel very accurate. The arrow does not fly to where I point the tip of the arrow at". There were exceptions, of course, however the majority had problems with aiming.

When the bow is held but the bowstring is not pulled then the bow works as if any other picked up object: its position and rotation are equal to that of the holding hand and it has an invisible follower item. However, if the bowstring is pulled with the other hand, the bow's orientation will be dependent on both hands. The bow's forward vector is set to be equal to the following subtraction: the position of the hand holding the bow minus the pulling hand's position.

```
Vector3 start = pullingHandPosition;
Vector3 end = bow.hand.position;
Vector3 direction = end - start;
Vector3 axisToRotateAround = bow.hand.forward;
bow.transform.rotation = Quaternion.LookRotation(dir,
                         axisToRotateAround);
```

The position of the bowstring's middle part will thus be at the fingertips of the visual hand seen in the game. This kind of rotation means that the player could easily rotate the bow in any direction with only the pulling hand. For example, if from the player's view, the pulling hand is placed in front of the bow holding hand, the player would be aiming at himself.

This solution causes problems with accurate aiming. If in real life a person were to hold the bow in front of him and tried to rotate the bow with only the pulling hand (using the string) then the bow would not rotate freely in the holding hand. Rotating by the string in real life would also cause the holding hand to rotate. In TVR the holding hand rotates, however the real hand remains stationary (see Illustration 32 below).

Illustration 32. Bow pulling illustrated. Left column – in-game, right column – in reality. On the bottom row, the in-game hand is rotated, however the bow holding real hand retains its orientation.

This can be problematic when the player in TVR tries to aim with the bow. If in real life a person correctly brings the middle part of the bowstring near the chin, the bow itself is rotated in the same direction as the person's bow holding hand (see Illustration 33 below).

Illustration 33. Bow's orientation is dependent on the hand.[46]

This results in accurate aiming in real life, but is not represented in that way in TVR. In addition, since in VR the tip of the remote (representing the middle part of the bowstring's location) cannot be brought next to the chin without struggle (due to obstruction of the

[46] http://www.webyshops.com/tech-talk/Conversations/TruGlo-and-the-Archery-Connection.html

headset), another problem is encountered. The small distance between the tip of the controller and the person's chin adds a small angle on the horizontal axis of the bow, causing arrows to fly in a wrong direction (see Illustration 34 below). The arrows fly correctly where they are aimed at in relation to the bow. However, the players do not seem to notice the origin of that error.

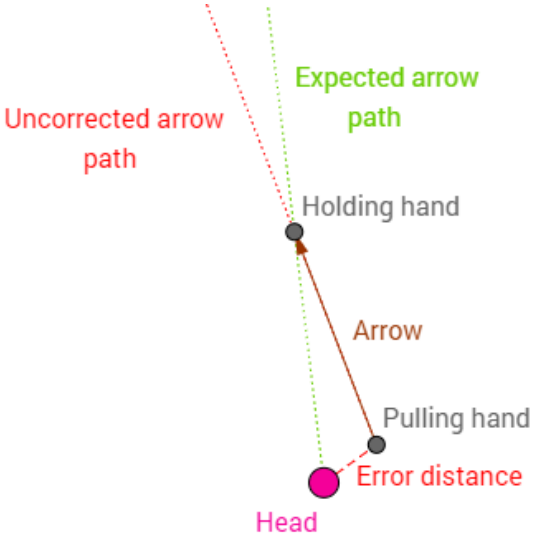

Illustration 34. Uncorrected arrow path caused by the obstruction of Vive's headset.

We tried to solve this by offsetting the pulling hand's position in the code in a way that if the player pulls the controller near the headset (usually to the side of it), then in the game the pulling hand would be moved a bit closer to player's head position (near the middle of the headset). See Illustration 35 below.

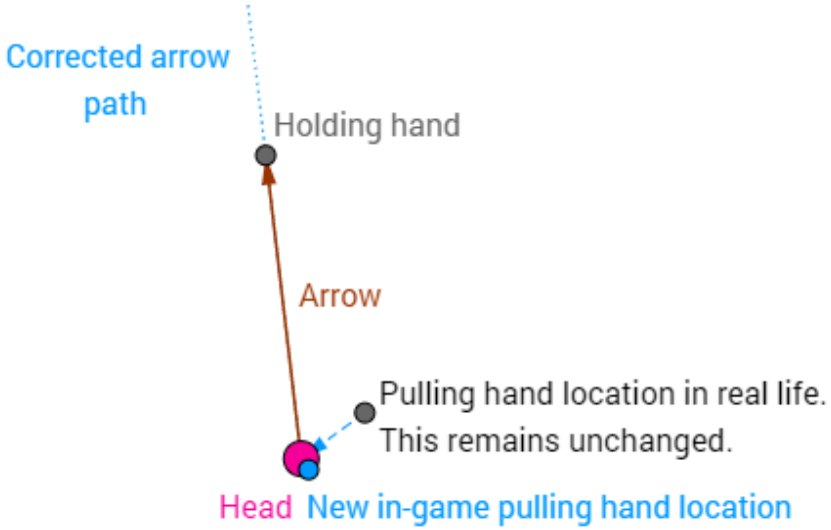

Illustration 35. Corrected arrow path. The in-game pulling hand is moved closer to the head while the real life pulling hand remains stationary.

This means that the controller's position and rotation no longer match the exact position and rotation of the in-game hand (see Illustration 36 below). This was not a problem since the majority of the time when the string was pulled, players did not look at their hand. The mismatch was not visible for them, thus not reducing immersion of the game. If the players did look at the hand out of curiosity, then they would see their hand rotated and positioned incorrectly.

Illustration 36. The hand is tilted downwards in VR.

This kind of solution worked only for a small number of players. The majority still had problems with aiming the arrow. This was due to the fact that each person does draw the string a little bit differently than others do. Some bring the controller to the side of the mask, some bring it in front of the mask. Some a little bit below and so on. This kind of action was caused due to the fact that people simply cannot aim with the bow due to lack of experience in real life. They concentrate their view on the tip of the arrow and predict that the arrow will fly in the pointed direction. However, since time became a problem for us, this problem was not solved completely.

There are alternative ways to let the player aim the bow slightly. One of them, used in the VR bow game QuiVr[47], is to let the player adjust the hand's offset in the game's menu settings. The benefit of this is that each player can set up the bow's rotation in a way that is suitable for them and thus allowing them to be accurate with the bow. The downside to this is that the player has to do manual adjustment, which can be time consuming due to trial and error method of testing the offset.

[47] http://store.steampowered.com/app/489380/QuiVr/

Another solution would be to launch the arrow from the tip of the drawn arrow with the rotation equal to player's camera's forward direction (see Illustration 37 below). This should ensure that the arrow always flies to where the player is aiming the arrow, at least on the horizontal axis. The downside to this is that it requires extra checks for situations where the player is not aiming down the sights. It should be decided then if the arrow should fly where the drawn arrow is aiming at or should its rotation still be equal to the forward vector of player camera's rotation. This solution was not tested, but it would seem that the solution would sacrifice immersion for clarity.

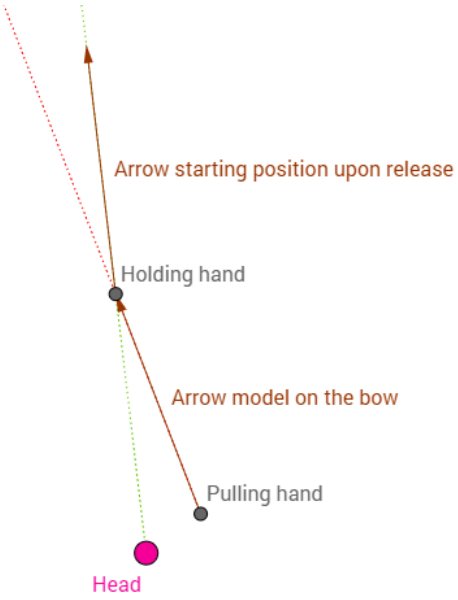

Illustration 37. Arrow model on the bow would be in the correct position when aiming. When the bow string is released, the arrow that will go into flight will start from the tip of the bow's arrow model.

7.7 Player Fatigue Reduction

Over long periods of time, both the bow and spear/bombs seem to be equally tiring to the player in terms of stamina. However, because the hand motions required for using these weapons are different[48,49], players can easily switch from the spear/bombs to the bow if the act of throwing starts to become too cumbersome. The muscles that were used to perform the act of throwing are not used as heavily when holding the bow and pulling the string. This means that these muscles which are used for holding and pulling are not as tired as the ones used for throwing. This opportunity of having the player to switch between weapons

[48] http://www.livestrong.com/article/364336-the-role-of-bones-joints-muscles-for-the-javelin-throw/
[49] http://www.livestrong.com/article/263814-what-muscles-to-build-up-for-archery-shooting/

and relax certain muscles, increases the overall play duration, which would be otherwise shortened by fatigue.

Another technique used in TVR that reduces fatigue of the player is that the players can rest as long as they like between each match. The player could even save their progress, return to the game after a week and continue where they left. The downside to this is that some challenge is taken away from the experience. The player is given time to rest, thus making the next wave less challenging compared to having no rest or very little rest between the waves.

8. Conclusion

This thesis presented a game design analysis for virtual reality game developers by investigating certain game aspects of a virtual reality game Tribocalypse VR. The given analysis was made possible due to the prior development of Tribocalypse VR by the author and the development team for 6 months. The aspects analysed were level design, graphical user interface and game mechanics. A measurement of immersion and clarity was proposed and used to assess the quality of different aspects. Thesis described how each feature should increase the immersion or the clarity of a VR game.

In order to achieve good environment and level design it is important to keep the visual style consistent. Each level should be able to guide the player to their objective and should not confuse them. It is very important to remember that even in VR the players have a difficult time spotting things that are far away. The developer should also take into consideration player placement in level design. Each location must feel comfortable to the player and not cause discomfort e.g. from motion sickness or fear of heights.

When designing a GUI, it is good to remember that icons work better than plain text in VR (unless the text is animated). Also, creating GUI elements that blend in too well with the environment (increasing immersion) could reduce clarity, because players might have hard time finding these elements. Each interactable element of the GUI should be very easy to use. Closer distances between the player and the GUI elements increase their usability.

The thesis also covered common game mechanics such as interacting with pickable items and casting magic spells. Thesis described solutions for making the interaction between the player and items feels as natural as possible. Thesis also proposed different solutions for casting magic spells in a way that would not be confusing to the player. Two solutions were proposed in the final chapter for problems that involve player fatigue and tiredness caused by prolonged play of VR games. It is important to use every possible medium (haptic, sound, input, visuals) to make the experience as believable and immersive as possible.

The author believes that this work will help VR developers to avoid the design mistakes encountered during the development of TVR and will save them development time. In addition to avoiding mistakes this work should give insight on how certain game mechanics can be developed. The author of this thesis might continue development on TVR in order to add a social aspect to the game. This could be in the form of online cooperative gameplay.

The aim of this would be to increase the quality of TVR and to research social aspects in virtual reality games.

The author of this thesis would like to thank the whole TVR-s development team for making the creation of Tribocalypse VR and thus this thesis possible. Also, University of Tartu's Institute of Computer Science and their Computer Graphics and Virtual Reality lab for providing the means for developing TVR. Special thanks to supervisor Raimond-Hendrik Tunnel for assisting with the completion of this work. And finally, friends and family of the author who provided support during the development of TVR and writing of this thesis.

9. References

[1] Statista, “statista,” Statista, [Online]. Available: https://www.statista.com/topics/2532/virtual-reality-vr/. [Accessed 9 5 2017].

[2] T. Bidaux, “The Rise & Popularity Of Virtual Reality In Games Media,” *VR focus*, p. 1, 2016.

[3] V. R. Society, “Virtual Reality Society,” Virtual Reality, [Online]. Available: https://www.vrs.org.uk/virtual-reality/history.html. [Accessed 9 5 2017].

[4] J. Mirt, “Virtual Reality HMDs 2016 Sales Numbers,” *viarbox*, p. 1, 2017.

[5] “en.wikipedia.org,” [Online]. Available: https://en.wikipedia.org/wiki/Presence_(telepresence). [Accessed 9 5 2017].

[6] “http://store.steampowered.com,” Valve Corporation, 3 2 2017. [Online]. Available: http://store.steampowered.com/app/520510/Tribocalypse_VR/. [Accessed 9 5 2017].

[7] “en.wikipedia.org,” 9 5 2017. [Online]. Available: https://en.wikipedia.org/wiki/Virtual_reality. [Accessed 9 5 2017].

[8] D. Ergürel, “The latest virtual reality headset sales numbers we know so far,” *Haptic.al*, p. 1, 2017.

[9] “en.wikipedia.org,” 26 4 2017. [Online]. Available: https://en.wikipedia.org/wiki/Virtual_reality_sickness#Newest_technology. [Accessed 9 5 2017].

[10] D. Weibel and B. Wissmath, “Immersion in Computer Games: The Role of Spatial Presence and Flow,” *International Journal of Computer Games Technology*, no. 282345, p. 14, 2011.

[11] J. Madigan, “The Psychology of Immersion in Video Games,” p. 1, 2010.

[12] C. T. Totten, An Architectural Approach to Level Design, A K Peters/CRC Press , 2014.

[13] W. Goldstein, “A Guide to Understanding Your Peripheral Vision,” *EyeHealthWeb*, p. 1.

[14] J. Singal, “How to Get People to Stop Ignoring Traffic Signs,” *Nymag*, p. 1, 2015.

[15] K.-J. Laak, M. Vasser, O. J. Uibopuu and J. Aru, “Attention is withdrawn from the area of the visual field where the own hand is currently moving,” *Neuroscience of Consciousness*, 2017.

[16] N. Abel, “NewtonVR: Physics-based interaction on the Vive (Part 1),” *Vrinflux*, p. 1, 2015.

[17] B-Reel, “Exploring Methods for Conveying Object Weight in Virtual Reality,” *Road to VR*, p. 3, 2016.

[18] “en.wikipedia.org,” 30 1 2017. [Online]. Available: https://en.wikipedia.org/wiki/Room_scale. [Accessed 9 5 2017].

[19] “en.wikipedia.org,” 25 4 2017. [Online]. Available: https://en.wikipedia.org/wiki/HTC_Vive#Technical_specifications. [Accessed 9 5 2017].

Glossary

Unity Related

Scene - A virtual 3D space, which contains GameObjects. The 3D space uses a left-handed coordinate system. This means that y axis is pointed up. x and z axis combined are considered as the ground plane.[50]

GameObject - Object in the game's scene, which consists of components and makes use of the component programming pattern[51]. Transform component is a mandatory part of each GameObject.[52]

Component - The component part in the component patterns, which can be attached to any GameObject to add additional functionality to that GameObject. Must be of class MonoBehaviour.[53]

Transform - A transform is a component consisting of two Vector3 objects and a quaternion: position, scale vectors and a rotation quaternion. Position defines an object's position in the scene. Rotation defines an object's orientation in the scene. Scale defines an object's size in the scene.[54]

Rigidbody - A component, which allows the developer to add physical properties to a GameObject. If not limited by rigidbody's settings then each GameObject with a rigidbody is subject to the gravity constant of -9.8. The movement of rigidbody's can and should be executed using the physics functions, defined by Unity's Physics engine. This ensures correct collisions between objects.[55]

Collision - The state at which two collision boxes overlap. At least one of the colliders must have an attached rigidbody component.

Collider - A 3D volume used for collision detection. Standard shapes in Unity are the cube, sphere and the capsule.[56]

[50] https://docs.unity3d.com/Manual/CreatingScenes.html
[51] http://gameprogrammingpatterns.com/component.html
[52] https://docs.unity3d.com/Manual/GameObjects.html
[53] https://docs.unity3d.com/Manual/Components.html
[54] https://docs.unity3d.com/Manual/class-Transform.html
[55] https://docs.unity3d.com/Manual/RigidbodiesOverview.html
[56] https://docs.unity3d.com/Manual/CollidersOverview.html

Capsule Cast - A temporary capsule collider with a lifespan of 1 frame. Used for detecting collisions.[57]

Game Units – 1 game unit in Unity's game engine is equal to 1 meter in real life.

Mathematics

Lerp - Linear interpolation. This function is used to linearly interpolate between two Vector3s. This is most commonly used to find a point some fraction of the way along a line between two endpoints. For example, moving an object smoothly from point A to point B, over time.[58]

Slerp - Spherical linear interpolation. This is used similarly to the Lerp function. It is used to linearly interpolate between two quaternions. For example, allows for smooth rotation of gameobjects.[59]

Vector3 - A vector3 is a struct, which contains a vector consisting of 3 float components. Example: Vector3(0.3, -25, 7.1).[60]

Zero vector - A Vector3 struct, whose vector consists of 0 floats. Vector3(0, 0, 0).[61]

Quaternion - A struct containing quaternions, mainly used for storing the rotation of an object and also to combine different rotations.[62]

Game Development

Fire Rate - The speed of firing a weapon in a game. Measured in rounds per minute (rpm) For example, fire rate of 10rpms means 10 rounds per second, i.e. the weapon is able to fire a maximum of 10 rounds per second.[63]

[57] https://docs.unity3d.com/ScriptReference/Physics.CapsuleCast.html
[58] https://en.wikipedia.org/wiki/Linear_interpolation
[59] https://en.wikipedia.org/wiki/Slerp
[60] https://docs.unity3d.com/ScriptReference/Vector3.html
[61] https://docs.unity3d.com/ScriptReference/Vector3-zero.html
[62] https://docs.unity3d.com/ScriptReference/Quaternion.html
[63] https://en.wikipedia.org/wiki/Rate_of_fire

Appendix

I. HTC Vive

HTC Vive is capable of producing room-scale experience for the user. “Room-scale (sometimes written without the dash) is a design paradigm for virtual reality (VR) experiences which allows users to freely walk around a play area, with their real-life motion reflected in the VR environment. Using 360-degree tracking equipment such as infrared sensors, the VR system monitors the user's movement in all directions, and translates this into the virtual world in real-time. This allows the player to perform tasks, such as walking across a room and picking up a key from a table, using natural movements. In contrast, a stationary VR experience might have the player navigate across the room using a joystick or other input device.” Wikipedia [18] HTC Vive achieves this by using 2 laser emitting “Lighthouse” base stations that sweep structured light lasers within a space. The headset and the Vive remote controllers have photo sensors, which detect these lasers. With the combination of multiple sensors on an object, the object’s position and rotation can be found [19].

HTC Vive system uses 2 handheld trackable controllers. The controllers have buttons for user input (see Illustration 38 below). These include standard pressable buttons and also a pressable touchpad. The touchpad is also able to track the position of the user’s finger on itself similarly to touch sensitive mobile phones.

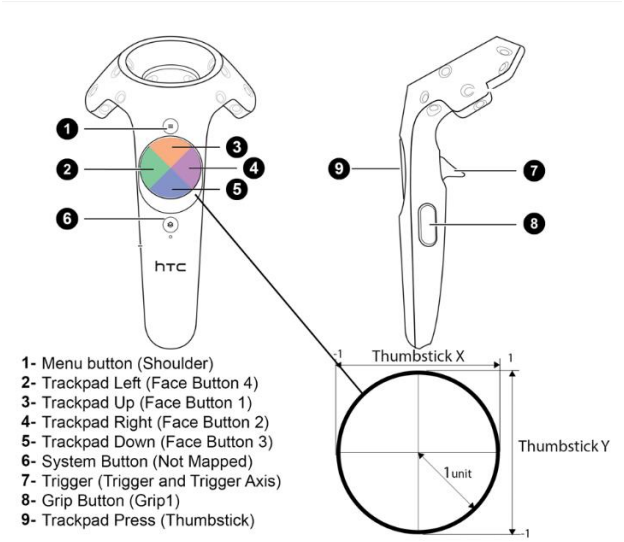

Illustration 38. Vive controller’s button map.[64]

[64] https://forums.unrealengine.com/showthread.php?116576-Htc-vive-non-of-the-buttons-give-any-input

II. Tribocalypse VR team

The development team of Tribocalypse VR consisted of 14 members:

- Jens-Stefan Mikson – Project Lead, Lead Programmer (University of Tartu)
- Raimond-Hendrik Tunnel – Programmer (University of Tartu)
- Marko Korbun – Marketing, Unity Developer
- Jaanus Jaggo – Visual Effects (University of Tartu)
- Erich Brutus – Lead Artist, 3D artist (Tartu Art School)
- Reino Meensalu – 3D artist (Tartu Art School)
- Egmond Merivee – 3D artist (Tartu Art School)
- Leene Künnap – 2D artist (Tartu Art College)
- Valeria Skabardis – 2D artist (Tartu Art College)
- Sylvia Renate Prass – 2D artist (Tartu Art College)
- Nathan Vaino – 2D artist (Tartu Art School)
- Raido Kikas – Concept Artist (Tartu Art School)
- Ruudi Vinter – Quality Assurance (Estonian Information Technology College)
- Taavi Luik – Music

III. Levels of Tribocalypse VR

4 png files of each of the levels of TVR have been added to a folder “Levels”. Stonehenge and Necropolis level designs are represented by level design documents. The Valley level designs is represented by concept art. Home Village’s level design is represented by a top-down view screenshot.

- Levels/1-HomeVillageDesign.PNG
- Levels/2-ValleyDesign.PNG
- Levels/3-StonehengeDesign.PNG
- Levels/4-NecropolisDesign.PNG

IV. Enemies of Tribocalypse VR

There are 3 types of enemies in Tribocalypse VR:

1) The tribols
2) Flying birds
3) The Necros Boss

Tribol

Illustration 39. Tribols walking in the world of TVR.

Tribols are the main characters of the world of Tribocalypse[65]. The player character in TVR also represents a Tribol. They are bipedal characters, around 1.8 meters tall and all of them wear masks in TVR with the exception of Necros's minions (see Illustration 39 above).

Their speed can vary from very slow to very fast. They can also wield weapons. Tribols equipped with the axe usually destroy the totem pillar faster than the ones without the axe. Tribols can also wear shields, which provide them cover from player's bow and spear attacks. Some tribols are equipped with bows. When they spawn they move to a fixed location on a map and starting attacking the player character.

[65] http://blog.tribocalypse.com/index.php/category/the-story-of-fire-and-skirmish/

Flying Bird

Illustration 40. Birds flying in the world of TVR.

Flying birds in TVR carry 1 tribol on their back (see Illustration 40 above). The tribol is always equipped with a bow. The purpose of the bird is to add another combat dimension by forcing the player to direct his view in the skies. Birds spawn further away from the village than the normal tribols. When they reach the village, they start orbiting around the player while the tribol on the back of the bird tries to make short work of the player.

Necros

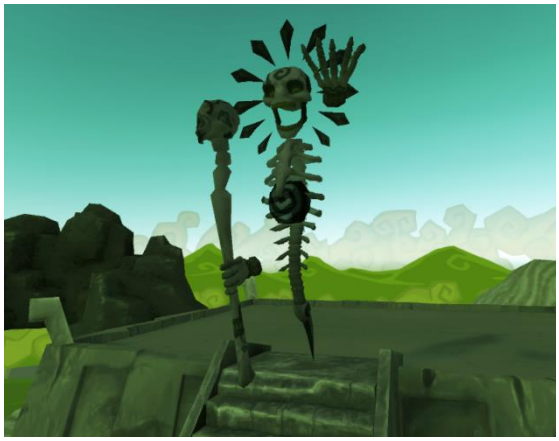

Illustration 41. Necros on top of a pyramid.

Necros is the only boss character in the game and appears only in the Necropolis level (see Illustration 41 above). Necros cannot carry any totem pieces nor destroy the totem pillar, however he does have two abilities. One of the abilities is used against the player. Necros projects a sphere of explosive energy towards the player (see Illustration 42 below). Upon contact with any object, the sphere explodes into thousands of pieces, dealing damage to any living being in a wide area.

Illustration 42. Necros firing an explosive sphere.

Necros can also spawn skeletal tribols, called Necros Minions (see Illustration 43 below). These minions act as normal unequipped Tribols, however look different from the casual ones. Necros summons the minions as long as he is killed. The destruction of the totem pillar is left in the hands of the skeletal minions.

Illustration 43. Necros's skeletal minions about to cross the bridge.

V. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Jens-Stefan Mikson**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Virtual Reality Game Design Analysis Based on Tribocalypse VR,

(title of thesis)

supervised by Raimond-Hendrik Tunnel,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **17.05.2017**