Offbeat VR

Joshua Pulido
jpulido@oxy.edu
Occidental College

1 Problem Context

With the rapid development of technology in recent years in both the virtual reality and drone industries, what were once niche experiences have now become common in households, begging the question: why has there not been software that brings the two spaces together in a fun and engaging way? While there are certainly methods for using virtual reality (VR) headsets to view drone footage, very little effort has been put into pushing the space further and innovating in the augmented reality (AR) field using drones.

Perhaps the best media for such software would be in the form of an augmented reality game, which is a type of game that focuses on overlaying digital objects on top of the real world, built for VR headsets such as the Quest 2, which the player uses to communicate with the drone and see the output, alongside overlaid game components. Such a game would provide for a far more immersive experience than mobile apps that currently dominate the space, and would be comparable to currently existing augmented reality headsets, such Microsoft's Hololens, without requiring such an expensive piece of equipment.

Unfortunately, it turns out that it is still far too early to make the claim that such a game could be made within only a few months, due to a lack of documentation for both AR and VR games, as well as a general lack of support for flying most drones with custom software. For this reason, a significant decision was made early on in this project to focus primarily on the construction of a VR game, with gameplay intentionally designed with the implications of flying a real drone kept in mind. This way, once the game was done, more effort could be put into accomplishing the rather difficult task of implementing the real drone into an AR version of the game.

With this transition in mind, the focus of project saw a significant shift towards many of the important aspects of a VR game. It was now important that the game feel immersive, and the controls were intuitive for players. Accessibility was also a key factor in the development the game, since VR traditionally provides a barrier to entry due to the relatively high probability that new players experience some form of motion sickness or dizzyness. Of course, it was also important that the game actually be fun for a wide audience of players. Keeping all these factors in mind, Offbeat, a rhythm game where the player avoids the notes instead of hitting them, was developed.

2 Technical Background

One of the key decisions to be made in developing a VR game is which platform to create a game for. Most VR headsets in the market currently require a separate computer with a powerful graphics card, and are often very expensive. However, in recent years, Meta (formerly Oculus) has developed inexpensive headsets that do not require the use of a computer, known as standalone headsets. Their Quest and Quest 2 platforms use mobile hardware and a modified version of Android, and have the ability to run games independently. Due to the current rumors that other companies are intending to transition to a similar model[3], as well as the greater accessibility of these models, it made the most sense to develop for the most popular standalone headset at the time, the Meta Quest 2.

The current game engines that are easy to use and free for creating VR games are Unity and Unreal Engine, both of which have their own advantages. However, for the purposes of this project, Unity was the obvious choice because of its integration with the C# language, as opposed to the flowcharts used by Unreal Engine 4 and 5. The most well known benefit of using Unreal Engine is its superior graphics, which for this project amounted to a negligible benefit. Unity also provides the added benefit of being able to compile the same game for multiple platforms without having to port code, in this case both computer-based (PCVR) and Android-based platforms. Another very useful feature of Unity is the existence of "prefabs" which are essentially saved copies of in game objects, which can be used to duplicate the object in the editor or through code, even if the in game object is later deleted. This is very useful for the creation of custom levels, as objects can be dynamically loaded as needed by the game, and also allows for the developer to easily create variants without having to remake the whole object.

Meta provides a software development kit specifically for creating VR games for their devices in Unity[5]. The library provides full access to the various components of the

Meta Quest 2, such as the ability to enable pass-through in Unity as well as access to more complex forms of input and control, such as raycasting and hand tracking. Most notably, for Offbeat, this library enables the use of motion controls in a Unity game. These work by checking the position of either controller in relation to the headset, whenever an update function is called. In Unity, the update function would be called at least once per frame, if not more often depending on how quickly inputs needed to be registered. The SDK also provides prefabs for some important parts of a VR game, so that anyone can quickly get a VR ready game up and running. Perhaps the most notable is the OVRCameraRig, which replaces a traditional camera in Unity, and allows for the player to move their head around to view the entire world around them. It also provides other necessary classes, like the OVRManager, which handles communication with the device used for input and other processes.

Since Offbeat is a rhythm game, there are special properties to be aware of when considering how the game performs. Rhythm games require very tight tolerances for when beats are played, as errors would result in incorrect rhythms, thereby ruining the gameplay experience for players. Therefore, a game that is not optimized, or a slow processor, could result in a disaster of a game. Thankfully, the Quest 2 has been able to keep up with the tempos of songs that have been implemented so far, and there is no evidence to suggest that it could struggle in the future. Nevertheless, this was kept in mind during the development process and is why the ease of switching platforms in Unity was a key factor in the decision.

As for the drone, it would not be possible to fly just any drone since the industry standard is to use radio frequencies to communicate with and control the drone. For this reason, communicating with most drones would be impossible with the Quest 2, and a drone that communicated over WiFi would be required. For this project, the Tello, by DJI and Ryze, was the perfect candidate due to a comparatively large developer community compared to other drones, as well as its WiFi connectivity. It is important to note, however, that latency plays a huge issue when working with any device over wireless communication, and in this case it is especially be aware of latency of the video stream. One study on latency[2], performed by researchers at the National Institute of Information and Communications Technology in Yokosuka, Japan, found that the use of the 2.4 GHz radio band caused major latency issues due to the large amounts of interference. Although their research focused more on beyond-line-of-sight flight, the latency still poses an issue at closer distances, and worsens with the amount of wireless devices in the area. The latency of the Tello drone is in theory reasonable within approximately 100 meters from the player. Had the drone been properly implemented into the game, this might have restricted what was possible with gameplay somewhat, but would have allowed for a working end product.

3 Prior Work

The game that I took most inspiration from is Beat Saber, a VR rhythm game that has very similar game design and style to that of Offbeat. In the game, notes spawn a few beats before they are actually played in the music, and move towards the player. The player uses ”lightsabers” to slice through notes, which inspired the original mechanic of colliding with notes in time with the music in Offbeat. Beat Saber is lenient with the amount of notes that can be missed, which also inspired the health system, although it was required to be adapted to fit with the other mechanics of Offbeat. Perhaps one of the most notable inspirations from Beat Saber was the level design, which offers different experiences for every song that is included in the game. The goal with Offbeat was to take this a step further, implementing a large variety of different types of notes. Unlike Beat Saber, where different notes require different mechanics, different notes in Offbeat can be introduced without the complication of overwhelming players with different mechanics. Instead, the many different notes can be used to design gameplay for each song with minimal development effort required. Offbeat also takes inspiration from the visual identity that each level has, and currently provides options for level designers to use colors, rotations, and corridor types to give each song its own personality, with further options planned for the future.

For this rendition of Offbeat, the core gameplay mechanics were prioritized over visuals, so significant inspiration was taken from older video games built on less powerful engines. In particular, the game ”Run” inspired the corridor visual style; in the game, the player moves along a corridor that contains many holes they must avoid. Although the corridor is not part of the core gameplay, the default look of Offbeat emulates that of Run’s corridor, and it surrounds the player with holes randomly placed inside of it, allowing the player to look through into the background. Furthermore, the rotation of the corridor in Run inspired the rotations in Offbeat, although they can be a bit overwhelming for players in VR and have the option of being disabled. These visual elements are likely to stay; however, in the future the abstract visual identity that is attained using primitive elements, such as cubes, with no texture is likely to change. Taking inspiration again from Beat Saber, the plan is currently to use a darker default theme for the notes, corridor, and background, with colored lighting giving each level its own personality.

While this game is a rhythm game at its core, the ”flying a drone” component of the original idea manifested itself in the immersion and controls, which were heavily inspired by

space flight games such as No Man's Sky and the Star Wars series of games. In all of these games, the flight of aircraft is simplified in such a way that the players feel as though they are still in complete control of the vehicle, despite not having access to the fine controls available when flying a real aircraft, even a drone. Even though Offbeat is a VR game and uses motion controls instead of traditional game controllers, the core principle of making the controls easy and intuitive remains. One difference, however, is the manner in which that the y-axis responds to player input. No Man's Sky and the Star Wars games have the controls inverted, where moving the joystick down results in the spacecraft up, and vice versa. Due to both the immersion that VR provides as well as physical feedback that motion controls provide to players, the inversion felt really bad during user testing, so the more intuitive approach of "moving up goes up" was implemented instead.

Although the drone AR version of the game did not function as intended, it was a key part in the inspiration and development of the game design, and there is an interesting game that existed at some point which accomplished a similar goal. Drone Prix AR[6], a game by developer Edgybee, is the earliest attempt at creating an augmented reality game using a drone when searched on Google, and was developed for the Android and iOS platforms. Despite lacking the immersiveness that a VR headset would bring, the game accomplished a similar feel to the AR concept that was planned for Offbeat. In many ways, it inspired the simple approach to game design that were taken, and also highlighted the importance of a simple visual design when working in the AR space. There is also further evidence that such a game could have been viable, with many projects available online that run parallel to this idea. For example, one group of researchers[7] sought to create an augmented reality spectating application for drone FPV racing, and the paper was written several years ago when the technology was not as capable as it is today. Unfortunately, these projects did not provide actual documentation to support the development process and their impact can not necessarily be seen directly in the final product; they instead served more as an inspiration for the direction of the project as a whole.

4 Ethical Considerations

With the development of any game, there are some ethical considerations that should be taken into account, especially with relation to certain aspects of the game design. This section identifies just a few of them, including accessibility, the use of music, and data tracking. Accessibility was a key concern throughout the development process, and the current solutions are explained below, but the use of music and data tracking are more relevant in the future if the game is released, and potential solutions are offered.

4.1 Accessibility

Given that this is a game that targets a wide audience of different types of players, it is almost a certainty that the default version of the game may not be accessible to all players. They may require some sort of accommodation to ensure that they are able to play the game and have a good time without having a negative experience, whether that is frustration or an actual physical side effect. The immersion into another world that VR provides often leads to motion sickness for some players, and this must be taken into account when adding any sort of activity in the game. For this reason, it was important to add the accessibility mode to the game and attempt a reduction in the impact that Offbeat has on motion sickness for its players. Interestingly, another important factor with regards to motion sickness is the runtime efficiency of the game, which can be measured in frames per second (FPS). The Quest 2 has a high refresh rate of 120Hz, which helps trick the brain into thinking the VR world is realistic[1]. Any movement which appears unrealistically slow or jerky counteracts this effect; therefore, accessibility, among other reasons, make it important that the game performs well, ideally with a framerate higher than 120 FPS.

Another important consideration when developing the game was the use of visual and audio features as part of the core game design. There is a high likelihood that some players could have hearing impairments or colorblindness, which impact their ability to perceive the game as most players do. With regards to the audio portion of this, the music is an important part of the game, but serves mostly as a means to enhance the gameplay experience. Therefore, players who are unable to hear the music will hopefully still have a fun time playing the game. Players who are colorblind may have their experience more impacted, since the different types of notes are currently identifiable only by their color. The game partially remediates this by warning the player of what type of note they will hit through the collision indicator, which contains a symbol in the upper right hand corner that identifies the note. A plus symbol indicates that a health note will be hit, an "x" symbol indicates that a death note will be hit, and a triangle with a square in the middle indicates that a regular note will be hit. However, this is not enough to distinguish the different types of notes, since the indicator is only active when the player is directly in front of the note, meaning they can not identify the other notes around them that are coming up. The solution to this is obvious and involves changing the models to be noticeably different for the health and death notes, but this is not as straightforward as it seems. The models must be implemented in a way that does not impact the intuitiveness of the game. For example, a "plus sign" model for the health system would not make sense, since it would have a hitbox

that breaks away from the standard set by all other notes. This will be done in the near future, but it needs to follow the same pattern of design and user testing that the rest of the game has gone through.

4.2 Use of Music

As a rhythm game, a key feature of Offbeat is the songs which make up the different levels of the game. Since the focus of the project is on the development of the game itself, it currently uses popular songs as the built in levels. This brings up some potential problems with regards to the copyright system, since the rights to the songs are owned by other companies. This is not currently a huge issue, since there are no public builds available, and no revenue is being made with the game at the moment. However, this might not be the case in the future, and the songs will have to be removed if the game is ever publicly released, and new songs which are not copyrighted will be added (likely ones composed specifically for the game). This partially solves the problem, but in the future there could be further complications if players are allowed to upload their own songs to a database that interacts directly with the game. It might suffice to provide a way for copyright owners to take down versions of their music from the database at will, but this would have to be explored further.

Even if the legal issues were non-existent, which can be accomplished through the use of songs that are not copyrighted, there is another factor to consider: whether or not the artist would want their music portrayed in a certain way. This argument has recently been used by many illustrators in the online community due to the explosion in popularity of AI generated art forms, which train on datasets of images that are available online. These generators use artist's works as a part of their training process without the artist's explicit consent, causing much controversy over the ethical use of such generators. This allows anyone to create new images in the style of particular artists, based on what the model learned from the work, allowing for the creation of art that does not align with the artist's views or original intent[8]. Although Offbeat is a video game and thus not involved with the AI art space, there are some important parallels to take into account. The game is currently designed in a way that anyone can create a level with visual components that they define which are intended to capture the personality of the music. However, these can theoretically be used to alter that message instead. As an example, someone can take a song with long runs that is meant to be upbeat and happy, and add darker colors and much rotation, thereby altering the song to seem more chaotic. Furthermore, the use of algorithms to define visuals based on musical properties found in the song, which was originally planned for the game and may make a return in the future, can also result in a wrong visual output and thus does not solve the problem. This is probably not a huge problem that should hinder further development of the game, but is definitely something to keep in mind when allowing the use of other people's music in the game. It would make sense to give the artists the power to remove their music from the game at will, just like how copyright owners should be able to.

4.3 Data Collection

The Meta Quest 2 has many sensors and cameras on device, which allow it to have the same features as other headsets, without the use of external sensors. While they provide for a great experience for customers, there are some ethical concerns to be considered by developers who have access to the data produced by the device. Such data could be used maliciously to learn more about players without their consent, especially any data collected by the camera which can provide insight into the private homes of players.

Of course, such use of data by any developer without notifying the players is against Meta's developer data use policy[4]. Even if it was not against the policy, there is no reason to be collecting player data for the purpose of the game; therefore, motion tracking data will only be used at runtime to play the game, and will not be collected or stored in any way. It is still important to notify players that their data is not being collected for their own peace of mind. This could be done through a data policy that is present on a website or the Quest app store, but also should somehow be communicated to players in game, perhaps through the main menu, to ensure they are able to find it.

5 Methods and Evaluation

In order to create a VR game within a few months, it was important to follow a strict schedule so that the game design and systems were developed in time for a prototype to be done by the end of the semester. There are a few distinct pieces that come together to form the game, which include an idea for the game design, an intuitive control scheme, and a modular game implementation which allows for customizable levels in the game. Given that this is a project to create a video game, the main criteria for the evaluation is how "fun" it is. However, "fun" can mean a variety of different things, which is why it is important to perform playtesting throughout the project to evaluate whether or not the game was a success.

5.1 Game Design

The first step in the development of the game was coming up with an actual idea that could be implemented within

VR. This was slightly challenging due to the requirement that the game needed to be able to accommodate an AR experience with a real drone. There were a few possible routes, but the one that stuck the most was the aforementioned rhythm game concept, which required players to fly into the path of notes, so that it was hitting them in time with the music. Although inspired by other rhythm games, this idea had not yet been done before in VR, and provided for a unique experience that could be replicated using a drone.

With the key elements of the game's design having been decided, a couple more design processes needed to take place in parallel. Much like with other parts of this game's development process, playtesting was necessary to iterate on the game's design to ultimately have a fun game that was enjoyable by a diverse group of players. Immediately, it was discovered that flying into notes that were coming towards the player felt wrong and had a poor user experience, which led to an adjustment in the core design: the players would now avoid notes, instead of trying to hit them. Among other benefits that came from this change, this allowed for the game to take on more of a "bullet hell" design, which most players found to be fun, although sometimes frustrating. The next couple of playtesting sessions led to a back and forth of adding features to decrease the frustration of the game, getting feedback that it was now too easy, increasing difficulty, and again being told that it was frustrating. Through this process, the following key gameplay features were added: collision detection indicator, the health system, and variable difficulty through health notes, which give health back the the player, and death notes, which cause the player to immediately lose the game regardless of their current health. The game is still quite difficult for most players; however, recent feedback indicates that it is a lot less frustrating, and losses are perceived to be caused by slow reaction time, rather than being the fault of poorly designed gameplay.

Another feature that was designed immediately after coming up with the rhythm game concept was the level design, which took a pseudo-random approach to gameplay. The idea was that every time the game was replayed, it would be a little different, but still maintain some of its identity through each play session, allowing players to learn the outline of the level through repetition. So, the models of the notes would remain the same, and the general location where they spawned would remain the same, but sometimes the exact location would change. For example, a "left note" would always spawn on the left side of the corridor, but could spawn at any point on the y axis, whereas a ring that the player flies through would always spawn at the same time in the same location, with no variance.

5.2 Control Scheme

One of the most important design elements for this game is the control scheme, which needs to be intuitive enough so that anyone can pick up the game and learn how to play it with minimal guidance. As a result, the project started out with the development of the controls, using the Oculus Quest 2 controllers. Initially, this was done using the the joysticks to move the player around, but felt underwhelming in the VR space.

This resulted in a transition to using motion controls to move the player around. Although they deviated significantly from what it would be like to fly a real drone, the motion controls felt more immersive and intuitive, resulting in a better user experience during playtesting, so they were kept. Unlike regular joysticks, however, the Oculus Integration SDK did not provide an easy way to interact with the controllers, and instead provided access to the position of each controller with relation to the headset. Simulating motion controls was done by storing the last location of each controller, and checking if the delta between their new and old position was above a defined threshold. In order to move around, the players must move both controllers in the same direction. This way, assuming that the players started out with both controllers directly in front of them, then wherever they moved their controllers in relation to the headset would directly correlate with where the player was in the VR world, leading to intuitive movement.

The requirement that both controllers be moved in the same direction for movement also opened up possibilities for using different movement combinations for other inputs, which would take the form of tricks that the player could perform. As an example, the player could move their right controller left and their left controller right, in effect crossing their arms, to do a barrel roll. In theory, the duration of the trick would be sufficient time to reset the position of the controllers so that standard movement could continue. Due to the fast paced nature of the game, however, this form of input had the opposite effect and was unintuitive, making tricks hard to use. Instead, the traditional button inputs were used for tricks, which is currently only the barrel roll, since pressing a button was much quicker than using motion for input. Since the buttons were already being used for tricks, additional inputs were added to be able to pause the game, as well as quickly restart the game in the event of a bad play or loss, both of which were requests that came from playtesting sessions and provided for a better user experience.

5.3 Game Implementation

At the start of the development process, the decision was made to make the codebase of the game as modular as possi-

ble, separating the different features into their own systems that ran independently of each other, with the exception of some communication between them when necessary. This was important for organization purposes, but also provided for large amounts of customization with regards to levels and game options, since systems could be easily be toggled on or off. With the controls having been properly implemented into their own system, the next core component of the game was the rhythm system, which handled the spawning of notes as well as visual events that occurred in time with the music. It functions by taking in the length of the song, as well as its tempo in beats per minute, and converting the time elapsed since the song started playing into a current beat number. This current beat is then compared to two queues, one which contains the next note that should spawn, and another which contains the next visual event that should take place; if they match, the note or event is handled. To accomplish this, the rhythm system also takes in two json files for every level. The first is referred to as a level json, and contains the song information (title, bpm, length), as well as the two queues of events and notes. Each event/note has a type, which identifies it, as well as the beat where it should be handled. For the spawning of notes, there is a second json file, known as the definition json, which tells the rhythm system what notes this level has access to, as well as what coordinates to spawn at and what animation should be applied. The models of the notes are defined using Unity prefabs, which are defined in a list contained within the rhythm system. A single prefab can be used to define multiple notes in the level, which is why the definitions json is important. For example, the simple cube model is currently used to 7 different types of notes, including the ones that spawn on the left and right side of the corridors, and so on. This separation of notes into different components is the largest contributing factor to the customizability of levels. When a note is spawned into the game, it is given a script which will move the note at the appropriate speed so that it reaches the player at the exact moment the note is played in the music. This same script is also responsible for any animation of the note. The rhythm system is also responsible for keeping track of every note it spawns, so that it can remove them in the event of a restart, or cause the notes to ”explode” if the player loses. Unlike notes, visual events usually require some amount of code to be written for the definition of a new event, so the rhythm system currently is hardcoded to handle the 3 different visual event types that can currently take place. If significantly more are added, it will likely be rewritten to follow a similar structure to that of the notes.

Although the rhythm system is responsible for recognizing that visual events must happen at a moment, it does not actually change anything itself. Instead, it communicates with the corridor handler, which is responsible for the corridor that surrounds the player during gameplay. The corridor handler contains access to prefabs built in the Unity editor for different corridor types, each of which has its own name that identifies it. The prefabs are also designed to work with the corridor wall utility, a simple script that allows the different components of the game object to be modified visually, such as through the change of color or randomly removing panels. The corridor handler is responsible for creating a seamless corridor, which begins at location where the notes spawn, and moves towards the player. To do this, it keeps track of the last corridor it spawned, and spawns a new one once the previous z coordinate has moved past a z coordinate value. It is responsible of keeping track of every corridor instance it has spawned, moving and rotating them as necessary, and removing them once the game is lost or they are no longer visible. The rhythm system can turn this corridor handler on or off at will, and can also change the type of corridor that is being spawned based on a visual event. Although it would have been possible to create a large corridor prefab which encapsulates the player in a similar way, it would not have provided the same type of immersive experience as the current corridor, which changes in time with the music. It does technically cause the processor to do significantly more work, but it does not actually have a noticeable impact on the performance of the game on the Quest 2.

The rhythm system and corridor handler provide the visual identity of the game, and create a really cool experience that almost acts like a music visualizer. However, Offbeat is not an actual game without the health and collision systems that provide a loss condition. The collision system uses Unity's built in collision detection, and identifies when a player collides with a note. It then works with the health system to decide the appropriate action to take. If the player has hit a regular note, they will take damage. If their health is still above 0, they will receive an audible cue that they have hit something, and a warning indicator will show up on screen to show that they have taken damage. If the player has 0 health, or has hit a death note, they will lose the game, and will be notified both through a sound cue as well as a visual cue, where the world explodes (or falls in accessibility mode). Finally, if a player has hit a health note, they will receive one health point, up to their max health of 3, and will be notified that this has happened through a different sound. The visual indicator will also update to reflect the current health of the player. If the player is currently performing a barrel roll, they are given a brief invulnerability. However, the collision system is not turned off; instead, the health system is temporarily ignoring any calls from the collision system that tell it to reduce health.

The last core system of the game is the collision predictor, which tells the player if there is an note directly in front of them, and that they should move out of its way if they

do not wish to take damage. Originally, it was planned to use a raycast emanating out of the player, which would turn on an indicator if it ever hit a note. However, this was a little too complicated for what was intended to be a simple feature, and a more elegant solution was used instead. This was accomplished using a separate game object, which has the same x and y dimensions as the player, but is significantly longer in the z dimension. It is a child of the player, so that it moves whenever the player does, and it is located directly in front at all times. If a note is going to collide with the player, it will first collide with the hitbox of this predictor object, and an indicator will show up on the heads up display to notify the player that they will collide with something. Once there are no notes within the players path, the indicator will disappear until a new one is encountered. Although using Unity's collision system is not necessarily more efficient than using the raycast, there were worries about how often raycasting would need to be performed, and using Unity's built in features instead was the better choice in this case. The predictor also has the ability to distinguish between different combinations of notes in the players path, to provide appropriate feedback. If the player is ever going to hit a death note, the indicator turns red to warn the player, but if regular notes are in the path, it will be yellow to signify that the implied danger is less. If there is only a health note in the path of the player, the indicator will turn green, but it will turn yellow or red if there is another type of note also in the path, to prevent the player from receiving bad feedback.

Every system outlined above has their own version of a pause and reset function, which can be called upon by user input. For organization purposes, there are pause and reset systems, which are simple classes that call upon their corresponding functions in all of the other systems. Pausing will also bring up the main menu, so that levels and game options can be changed.

5.4 User Testing

Offbeat, as a video game should be evaluated primarily on how it is perceived by a diverse group of players. In order to accomplish this, it was important to conduct playtesting sessions, not only at the end of the project but also throughout the entire development process. This way, feedback taken from the players could have a direct impact on the development process and, ideally, result in a much more successful round of final testing. A different approach from the way playtesting usually takes place was taken for this project, where the environment was a lot more casual and relaxed. Rather than telling the players what to look for, they were given the opportunity to jump right in after a brief demo of how to play the game from an experienced player. This way they could form their own opinions on the game without any bias caused by pressure from being in a playtesting environment. There were no imposed time limits on how long they were allowed to play, and they made the decision on when to stop. Afterwards, feedback was gathered in a conversational manner, rather than the use of a form. While this had the downside of not having raw data that could be used to make decisions based on statistics, many interesting discussions were had that gave ideas for new features that could be added. Players were first given the opportunity to give any general feedback that they had, which allowed for improvements to systems that were not directly being looked at during the tests. They were then asked a series of questions about how intuitive they thought the gameplay and controls were, whether or not they had any frustrations with the game, whether or not they felt dizzy, and whether or not they had fun. Their answers to these questions directly affected which new features were prioritized over others, and helped the game progress towards one that was fun and intuitive for all players.

The final round of playtesting was not substantially different from earlier playtesting, but the specific questions were dropped except for whether or not they had fun playing the game. Feedback was still important at this stage, but further adjustments were not realistic at that point before the final presentation. Since the game was in a fully playable state at this point, at least based on prior user testing, it was possible to have a much larger group of players try it out and give their opinion on the game.

5.5 Playability

Alongside the subjective evaluation performed through playtesting, there are also some objective measurements that can be taken to determine the quality of the game. Given that the game is running on the mobile architecture of the Quest 2, the performance of the game could take a hit from poorly implemented game features. It is therefore important the the game be optimized, to hit a target average FPS of 120. Hitting this target ensures that the game is responsive to player input and running smoothly, without any stuttering do to optimized. Hitting the target frame rate also decreases the likelihood that players will experience nausea or feel dizzy afterwards. For all these reasons, the game should be hitting an average framerate around the target; not exactly hitting it would not be the worst case scenario, but straying too far would be.

Aside from the framerate, which is a numerical measurement that can be taken, there some observable characteristics that would make the final version of the game unplayable. This includes game breaking bugs, which should not be present at all. As an example, there was a bug in the game that meant the player sometimes spawned outside of the corridor, rendering the game unplayable. Clearly, such

obvious bugs that happen often must have been addressed. Lag spikes are another occurrence that should not be present in the game at all. They could affect the average framerate depending on how common they are in the game, but even an infrequent drop in performance should not be present, since it breaks the immersive of the VR experience.

6 Results and Discussion

The original goal for the project was to create a virtual reality game that was both fun and immersive for players, and in this regard, it was a success based on the evaluation criteria outlined earlier in this paper. Throughout the entire playtesting process, every player mentioned that they had a good time playing the game. Even when there were frustrations during early testing, the feedback about the core game design was generally positive. More importantly, as points of frustration were addressed, players were more happy with the game until the final round of testing, during which no significant frustrations were expressed and everyone enjoyed their time playing. However, not everything went perfectly, as there was some confusion about the controls. Without the ability to see someone else playing the game, which was the case in most playtesting until the end, some players had difficulty figuring out how to move properly. After some explanation, they were able to figure it out and play the game with no issues. A tutorial or other form of explaining controls and gameplay in game would have helped significantly, and this is something I will be working on in the future. As for the playability section of the evaluation, there are no major bugs or lag spikes present within the game when run on the Meta Quest 2. Furthermore, performance measurements indicate that the device is not being overloaded, so the game is running fine. For some reason, the performance tool failed to provide an FPS output; however, it did provide average CPU and GPU utilization of 76

While the game itself is working as intended, unfortunately the original vision for the project of an augmented reality with the drone was not realized. Although there were ways to get the drone to respond to user inputs on the controllers, none of the existing libraries' camera outputs appeared on the Quest 2. Perhaps it would have been possible to decode the video signal from the drone manually, but at this point it was already late in the semester. A decision had to be made between prioritizing an AR prototype that still might not work, or focusing on polishing the VR version of the game that was not quite good enough to be considered a success at that point. Therefore, no further attempts were made at trying to get the drone to work for this semester. Thankfully, this was something I thought might happen at the start of the semester, and so the pivot towards focusing on the VR game meant that this project could still be considered a success, even without this portion of it functioning properly.

Were I to recreate the game again, I would have done a better job collecting data from user testing. While I think it was important to keep the environment relaxed and casual, I should have at least written down the feedback I got afterwards. This way, I could tangibly show how the game evolved over time based on feedback, instead of having to rely on memory. Additionally, had I known about how locked down the Meta Quest 2 is for developers, I would have chosen to develop for PCVR platforms instead. Not only would this have lightened performance worries and made it so that the drone would have worked better, but it also would have allowed me to avoid scenarios that were out of my control. Specifically, a couple of days before the comps showcase, Meta released a new update which disabled developer mode on all devices, with no way to reenable it at that time. I was thankfully able to use the PCVR version of the game at the showcase, but that was when I realized it was buggy, and thus it would have been better to develop for that platform from the start. Nevertheless, I am satisfied with how far the game has progressed to this point, and am excited to continue working on it into next semester.

References

[1] Butler, S. *How Important Are Refresh Rates in VR?* 2021. URL: https://www.howtogeek.com/758894/how-important-are-refresh-rates-in-vr/.

[2] Kagawa, T. et al. *A study on latency-guaranteed multi-hop wireless communication system for control of robots and drones*. Tech. rep. 2017. URL: https://ieeexplore.ieee.org/abstract/document/8301849.

[3] Lang, B. *Hidden SteamVR Feature May be Another Clue to Valve's Rumored Standalone VR Headset*. 2022. URL: https://www.roadtovr.com/steamvr-passthrough-playspace-setup-valve-index-standalone/.

[4] Meta. *Developer Data Use Policy*. 2022. URL: https://developer.oculus.com/policy/data-use/.

[5] Meta. *Oculus Integration*. 2022. URL: https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022.

[6] Murison, M. *Edgybees Launches Drone Prix Augmented Reality Game for DJI Pilots*. 2017. URL: https://dronelife.com/2017/05/02/edgybees-augmented-dji-drone-prix/.

[7] Sueda, K. et al. *Research and development of augmented FPV drone racing system*. Tech. rep. 2018. URL: https://dl.acm.org/doi/10.1145/3283289.3283322.

[8] Tallon, T., Jumalon, G., and Dinkins, S. *How Will AI Image Generators Affect Artists?* 2022. URL: https://www.sciencefriday.com/segments/ai-art/.