Give the People What They Want: a VR Game Development Journey Retrospective

Nathan Hahn – GAME798 – George Mason University Computer Game Design

May 17, 2022

Abstract

Virtual Reality (VR) development can be challenging for all developers, but especially for new developers learning how to design and program for the VR. As the team members of Starlight Vintage Studios put together their first VR game, we learned about the potential of VR games to let players experience games in new ways as well as pitfalls to avoid in the development process. If we could go back in time and start from the beginning, we could have made critical design decisions faster and been able to properly scope the game to match our abilities as new VR developers. This retrospective breaks down the development timeline of the game into distinct phases illustrating our VR game development learning journey. Along the way, key programming, design, and general development strategies will be pointed out, giving knowledge to game developers looking to get into VR development.

Give the People What They Want

Over a period of two and a half years, Starlight Vintage Studios created the game *Give the People What They Want*, a casual action VR game where you play as a body part swapping robot and throw things at people's faces. While we created our company in January 2020, we first began our development journey when we met at an AR/VR game development class at George Mason in the fall of 2019. The core concepts of the final game, *Give the People What They Want*, were developed from January 2021 to May 2022, but there was a lot to learn before we were ready to release a finished VR game.

The story of the development of *Give the People What They Want* is full of twists and turns. For many of the members of the team it was their first shipped game, and for all of us it was our first long-term project developed for VR.

The team of Starlight Vintage Studios consists of the following four developers:

- Nathan Hahn: Producer, lead programmer, designer, writer, and audio engineer
- Matt Busch: Programmer
- Matt Hazelworth: 3D Artist and Animator
- Zac Fischer: 3D Artist and Level Designer

Because our project had three distinct phases, it makes sense to tell the story of development in each of those stages with a summary of key successes and lessons learned for each phase. Through each phase, we grew and evolved as a studio and learned more about the VR development process. We hope that the lessons we learned can help other new studios who are looking to make VR games.

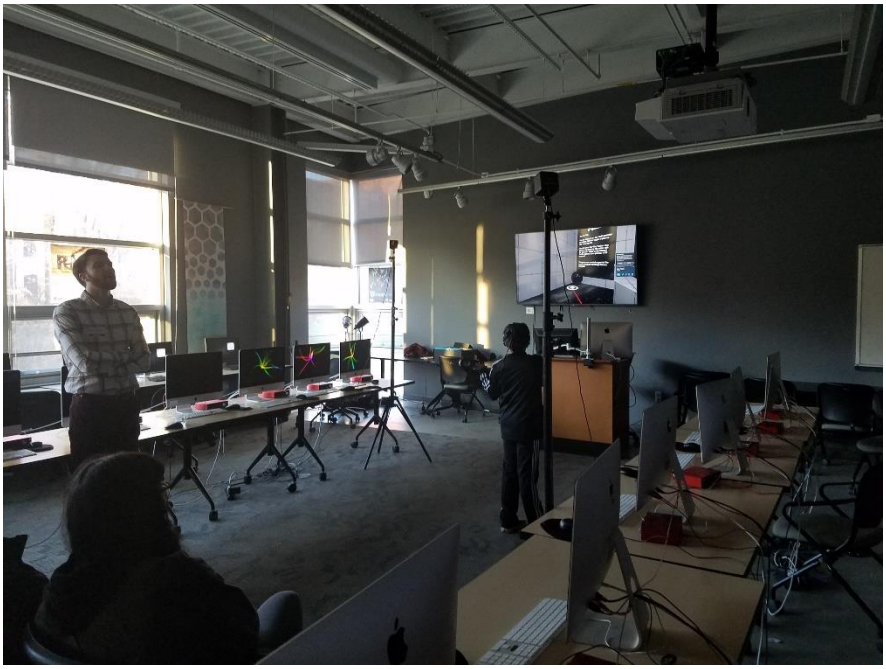
Part 1: V-Rcade (November 2019-December 2020)

While we did not explicitly start with the concept of *Give the People What They Want* until 2021, the members of Starlight Vintage Studios began working in VR in November 2019 as part of a George Mason University class. For the class, we developed a VR game that focused on four different physical arcade machines, a basketball throwing game, a football throwing game, a basket toss game, and a skeeball game. After the class was over, we stayed together to continue working on the game, as we believed that it would be easy to expand the idea a little bit and then release it to a video game distribution platform.

1-1: V-Rcade: Class Project (November 2019-December 2019)

We had a very short timeframe to develop our initial prototype, but we were able to achieve what we set out for our vision by coordinating in-person working sessions where we developed functionality, integrated assets, and made design decisions on the fly. While we didn't have much of an explicit task list, I tracked the work required both from asset scope and bug fixing to deliver the game on

time and in a functional state. Our lessons learned here were about the basics of VR development in the Unity game engine.



Presentation of the game at an open house for the College of Visual and Performing Arts

Give yourself time for setup and testing

For our game we used Unity and the SteamVR toolkit to develop our game for the HTC Vive. The SteamVR plugin gave us a lot of functionality out-of-the-box, but it had a non-trivial setup process to integrate it into Unity. It took a couple of hours just to get the game running and for the headset to connect to the Unity editor. After initial setup, plugging the system into the computer was not always reliable, and often required rebooting Unity, SteamVR, the entire computer, or performing the connection and disconnection process in specific orders to try to get the connection working again. Every time we wanted to work on the game, there was a certain amount of hoping the headset would work for that development session so we could be productive, which ate into our development time. Integrating the art and code assets also proved challenging, because for every machine we set up, we found development required an immense amount of tuning to ensure the games were playable as a VR

experience. We were not experienced at modularizing our art assets, so any testing which revealed the models had design-related issues meant a lot of rework for the artists.

The necessity of faking fidelity

While there is hope that VR can provide an accurate representation of physical actions inside a virtual space, the movement of the human body cannot be captured accurately in the limited tracking points provided by the current generation of VR. There was a large challenge in creating a playable version of our game machine where the player was expected to throw a football through a ring, because the way VR controller tracking functions does not account for the multiple muscles used to provide spin for the ball. While watching videos about how to throw footballs, I recognized that key to a good football throw was a straight throw without angular velocity. Angular velocity would cause the ball to veer off course from the throw trajectory. To simulate this phenomenon, we turned off the angular velocity on release of the ball to give players a more enjoyable football throwing experience. For VR games, programming needs to happen behind the scenes of VR design mechanics to make them feel right to players, which requires a deep understanding of what developers are asking players to do and the mechanics of the action being simulated.

UI is hard to communicate when players are distracted in VR

The experience of VR can be overwhelming, especially for people who have not used it before. For each of the arcade machines, we wanted there to be a high score element so people could try the machines multiple times and try to beat their scores. While people responded well to the audio cues that we provided, such as playing a music track as the game was running and playing a sound every time they scored a point, players spent very little time looking at their actual score and time limit we placed on a UI in the world. This was because they were heavily engaged with the task at hand and didn't want to concern themselves with reading while there was music pumping them up and getting them excited about throwing things in VR. UI is such a common method of showing players what they need to do in a

game, but it can be hard to direct a player's attention in VR the same way you might in a non-VR game using UI. In VR games, it is best to consider UI elements as a backup information strategy, rather than a primary one.

1-2: V-Rcade: Starlight Vintage Studios (January 2020-December 2020)

After the class had finished, the majority of our team chose to stay together and keep working on our game with the hope of releasing the game to Steam eventually. As we were looking at our code to understand how we wanted to clean it up from our prototype, we decided to see if there was a VR toolkit available that would provide us with a better set of core functionality to develop from. We found that the Virtual Reality Toolkit (VRTK) system had great examples that included much of the functionality we wanted, like climbing, guns, grabbables, snap points, doors, and lots of other interactable elements. It took us about two months to convert our assets and functionality from SteamVR to VRTK, but immediately after we were happy with our choice.

For this version of the game, we wanted to create an interior arcade-like space with a bit of a story element. You were entering a broken-down arcade that was previously owned by three siblings who were feuding with each other over how to run the arcade. Each one had a different specialty, and the player would learn about their backstory through audio logs as they went through the arcade fixing and playing the various arcade machines.

For this stage of the project, we started to learn how to deal with a complicated toolkit, performed proper playtesting to see what works from a design perspective in VR, and dealt with complicated VR hardware scenarios. It was challenging to put together a more detailed experience from our prototype not only from a code perspective, but from a design perspective, since it was hard for us to find the fun in our game after adding lots of different things for the player to do with everything that was provided out-of-the-box with VRTK.

The basketball game. In order to fix the game, the player needed to screw in the four screws to hold the backboard in place. Problem: tiny items are hard to see in VR, so players were constantly losing the screws.

Pay close attention to hardware and compatibility

After the class was over, I no longer had access to a Vive for VR development. I decided to purchase a VR headset so that I could continue programming the game. I tried to do my research to find a good headset, but I had a difficult time interpreting for myself which headset to purchase. Ultimately, I settled on the Oculus Quest, as I saw the Oculus team was providing good support for the platform and it was a cheaper VR headset compared to others in the market. I failed to realize that in order to connect the Oculus Quest to the computer an Oculus Link cable was required. I purchased a third-party oculus link cable, which did not work with my computer to connect with Unity, then purchased a first-party cable, which didn't work with my computer either. As a last-ditch effort to get the Unity engine to

connect to my VR headset, I tried a third-party wifi solution. While I was able to get the connection working, it was too laggy to perform any reasonable design tests.

In total, I probably spent around 20-30 hours attempting to get the Oculus Quest link to work to my computer before I gave up on the process and purchased an Oculus Rift S. However, I also needed to buy an adaptor to ensure the Rift S headset could connect via DisplayPort to my computer. Once that entire process was resolved, I could actually begin development. Not all VR hardware is the same and trying to use consumer marketing to inform development hardware purchase decisions can lead to disaster, so from my experience I would advise finding a mentor to advise you on your hardware purchase decisions.

The more a toolkit gives you, the more it will restrict you

For months, we tried our best to get the VRTK system to work for our game. When it came to the very basics of placing an item in the world, it could get the job done. However, we found the programming architecture of VRTK made it both brittle to any changes we introduced into the system and nearly impossible to troubleshoot when functionality did not work as intended. Every VRTK interactable object was broken down into over 100 different subcomponents, all completely separate Unity objects, linked through an event-oriented architecture. The event-oriented architecture meant that when bugs occurred, there was no effective stack trace that could be relied on with traditional troubleshooting methods. Instead, troubleshooting required looking through the entire component stack to attempt to identify misconnected event listeners. The massive quantity of Unity GameObjects caused the game to have performance issues as the number of interactable objects in the world increased, ultimately causing our game to require 2-3 minutes to initialize every time we tried to play it from the Unity Editor. Lastly, because the system was so specifically architectured, adding additional functionality that did not precisely follow their architectural pattern caused the code to become clunky and unmaintainable.

In December 2020, after frustration with the limitations of the system reached its peak, we began to swap our game over to the Oculus SDK toolkit. We saw immediate performance improvements in startup of the game, which with the same number of objects improved to around 2-3 seconds. In addition, the far simpler architecture enabled us to easily see where changes could be made while staying within the expectations of the Oculus SDK architecture. I have immense respect for the team that is developing VRTK, and I really appreciate that they make their solution free to use. However, it is not a toolkit I would recommend to a new VR development team due to its complexity.

Creating a variety-focused VR experience was a mistake for us

In the fall of 2020, we finally started testing our game with people outside our team. At this point, we had implemented the basic level layout and had an environment-focused narrative that led players through a story about three siblings inheriting a run-down arcade after their father passed. This experience involved the player walking around the arcade, fixing the machines, playing them, and learning about the history of the place through environmental puzzles.

When we began playtesting, I immediately saw the problems in our design assumptions. We had lots of exciting activities, including playing the arcade games, finding objects in the world, and repairing the machines, but all the interactions required an immense amount of direction for the players. Every time the player moved to a new activity, they needed to relearn what they needed to do to advance the game. In a terrifying vision of the future, I saw myself needing to implement a lot of UI and focused direction in the game to keep players on track, and I saw the challenges in putting all of that together for the game. Worst, I saw myself struggling to explain to every single player what they needed to do, so that when we delivered this game to ordinary players we would receive negative critiques that our game was too confusing.

Getting players into VR was hard enough. Throwing new, complicated mechanics at our players meant that as soon as players felt they were getting good at an activity or game, we were pushing them into a different one. I was worried that would leave players unsatisfied with the experience during a longer playthrough beyond a one-day playtest. My concern over this design problem led me to imagine a game where we went the complete opposite direction.

We had wide open spaces for the player to wander around, but players had trouble understanding what to do with highlights, snap zones, tools, and other objectives. VR itself was hard enough.

Part 2: Give the People What They Want (January 2021 – May 2022)

As we were performing the switch from VRTK to the Oculus SDK to create a more stable system to build on, I placed a bunch of objects in the scene and began throwing them around without any specific direction. Divorced from placing the objects in a hoop, through a ring, or into a basket, I found the basic throwing mechanics to be very enjoyable and simple. It made me completely reconsider the continual design complexity I had been piling into the game to try to make it interesting and made me

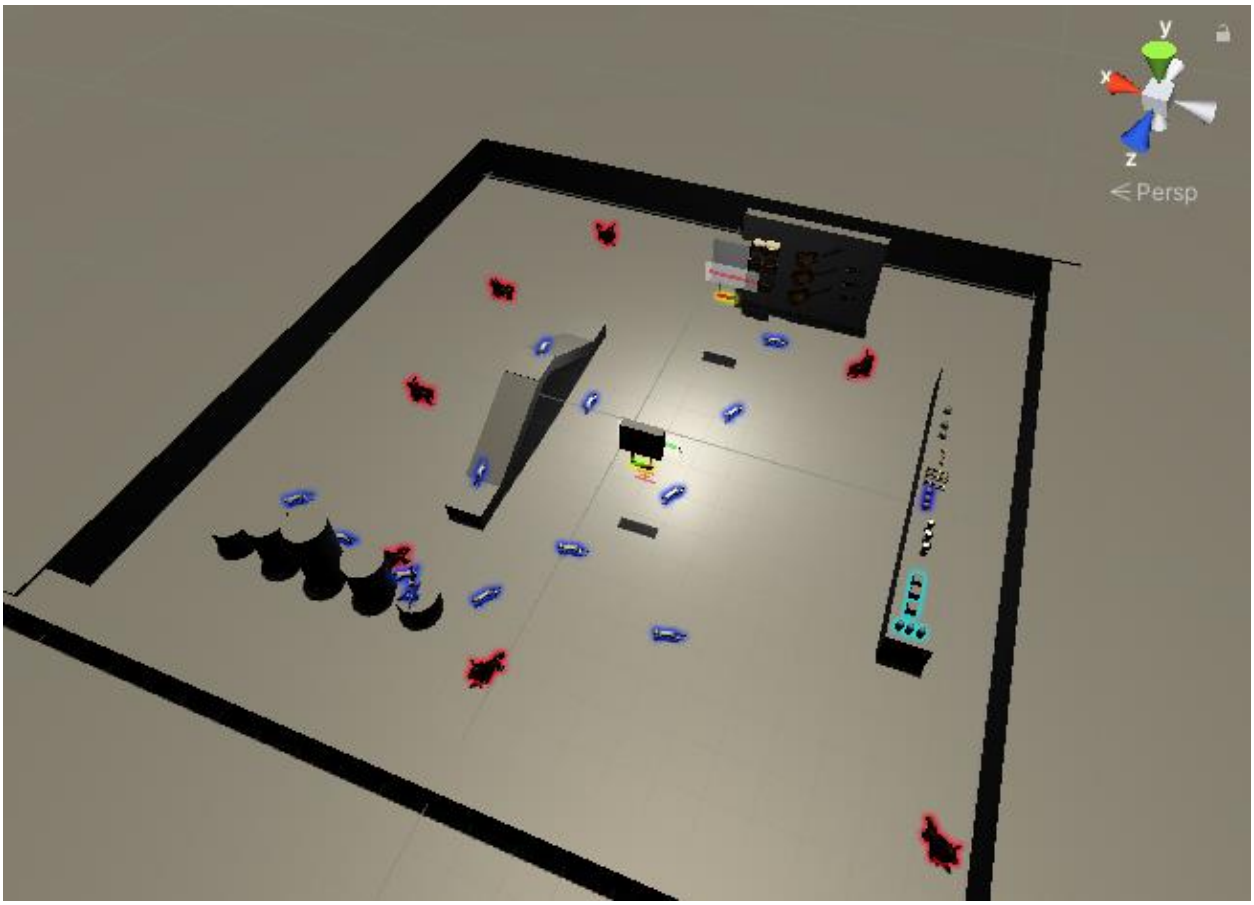
wonder if basing the game around a simpler mechanic would help players feel a better sense of progression over the course of the whole game experience.

I asked the team to give me one month to put together a prototype for a completely different style of game focused on throwing and catching objects. The core principle was that there would be people that wanted items and you threw things at them, and then they gave you some form of money that you had to catch.

The basic system was trivial to implement in about 2 weeks, and once I had the main mechanics working, I realized I had immense satisfaction with the simple game mechanics. Therefore, we completely shifted the game we were working on from a variety VR experience that required immense amounts of learning and tutorializing for the player to a simpler VR experience where players could engage in risk and reward decision making based on growth of personal skill in simple throwing and catching.

2-1: Linear Stage Experience (January 2021 – May 2021)

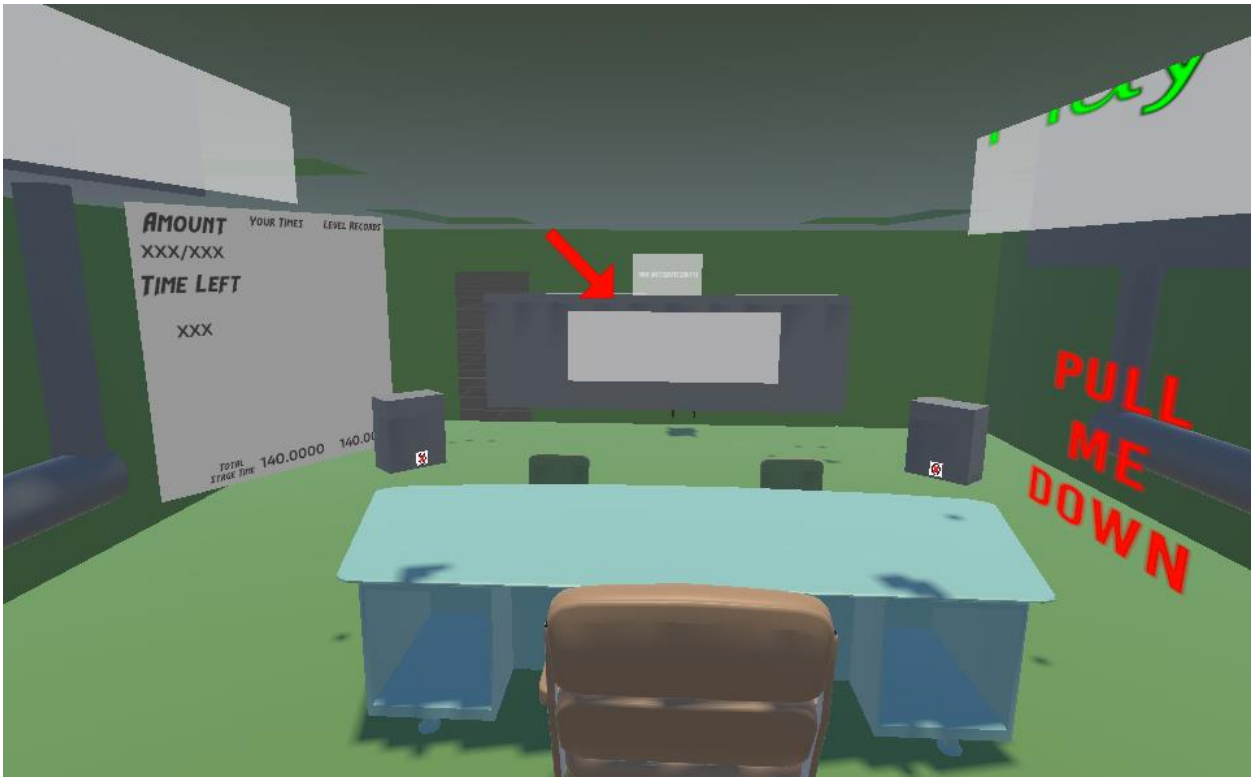
Initially, the game was developed with a linear structure in mind. There were 5 stages, and each stage had 10 difficulty levels. The player needed to complete a difficulty level to move onto the next one, and once they completed the levels for a stage they could move on to the next stage. The lessons we learned here were important for us to get to a place where players could feel like they could proceed without making the game too easy for them to beat.



The final room of the linear setup required players to use all their learned experience

Complicated ask mechanics were too much

There was one stage in particular that was my personal favorite but proved too challenging for anyone who played it. It was a stage where the player was asked to turn their mechanics learnings on their head, and based on specific instruction from what people asked, either give them an item or pull a lever which would drop the people through the floor. This mechanics setup was too much to comprehend, and the way it was implemented broke the free movement we allowed in the other stages. The amount of time we spent on this stage could have been avoided if we had recognized and stuck to the core promise of what the player had learned up to reaching that stage, that they would move around and throw items at people.



Playtesting quickly revealed the setup of this stage was too much for players to deal with, even with arrows and text attempting to tell them what they needed to do. It was a harsh lesson to relearn that in VR, you cannot rely on UI to tell your players what to do.

Having art does wonders

In our early playtests, players were largely relying on our graybox environments, which made it hard for them to appreciate what they were doing. We purchased an asset pack with human characters as stand-ins for the people, and once we implemented the pet models for a pet shop stage, we found the playtesters were really having fun. Getting art in the game that is a representation of your setting so players can understand where they are is important, especially in VR, where people need identifiable landmarks to orient themselves and their head direction.

Deciding to focus on PC VR

In January 2021, our team looked at the potential for delivering the game for the Oculus Quest, but at the time the only way to get the application onto the store was to work in partnership with Oculus, and since our team did not already have an established development track record, we didn't think it would be feasible. In May, as part of a different game prototype project, we investigated the Quest deployment pipeline, and decided that because we were already developing on the Oculus Rift S, we didn't want to split the build to develop across two different platforms.

Since then, Oculus has announced an easier publishing path through their App Lab process, which reduces requirements necessary to make games publicly accessible through the quest headset. However, performance requirements still apply for App Lab uploads, and our team decided that with our limited development time our initial release should focus on delivering the functionality of our game. We struggled to hit our frame rate targets on higher powered PC builds, so we decided aiming for the requirement of 72 FPS for Oculus Quest builds was outside of our reach until future development.

2-2: Randomized Experience (June 2021 – May 2022)

Because we wanted players to get the opportunity to see all of our content and not be gated by their individual ability, we decided to change the stages from being linear, puzzle-oriented stages to being slightly randomized with bonus challenges. That way, players could see the different stages in a playthrough even if they didn't have a lot of skill in throwing and catching in VR yet.

Our work on this project culminated in our demo at Magfest 2022, where over 50 people had the opportunity to play our game. Our fears about playability were relieved through the extensive playtesting that happened at the event. Nearly all the players who had prior experience with VR required no instruction to play the game, while only a few users who had not used VR needed any

instruction at all on how to play. About 10% of our players said they experienced motion sickness, which was surprising consider the level of freedom of motion we allowed in our game.

Booth at MagFest 2022

Player testing paid off

From the outset with this new project, we had a goal of having a playtestable build every month and performing playtests with testers outside the team. This playtesting was immensely valuable on two fronts. First, it ensured that the game would reach a playable state at the end of every month, so that we weren't dragging along features that weren't greatly contributing to the player experience for extended periods of time. Second, playtest feedback from both players who had no experience in VR and extensive experience in VR allowed us to see which mechanics needed tuning to make the game more fun, or where we needed to implement code changes to alter the way the world worked to give the player a better feeling of control and handle edge cases.

Our solid, simple, self-directed tutorial worked wonders

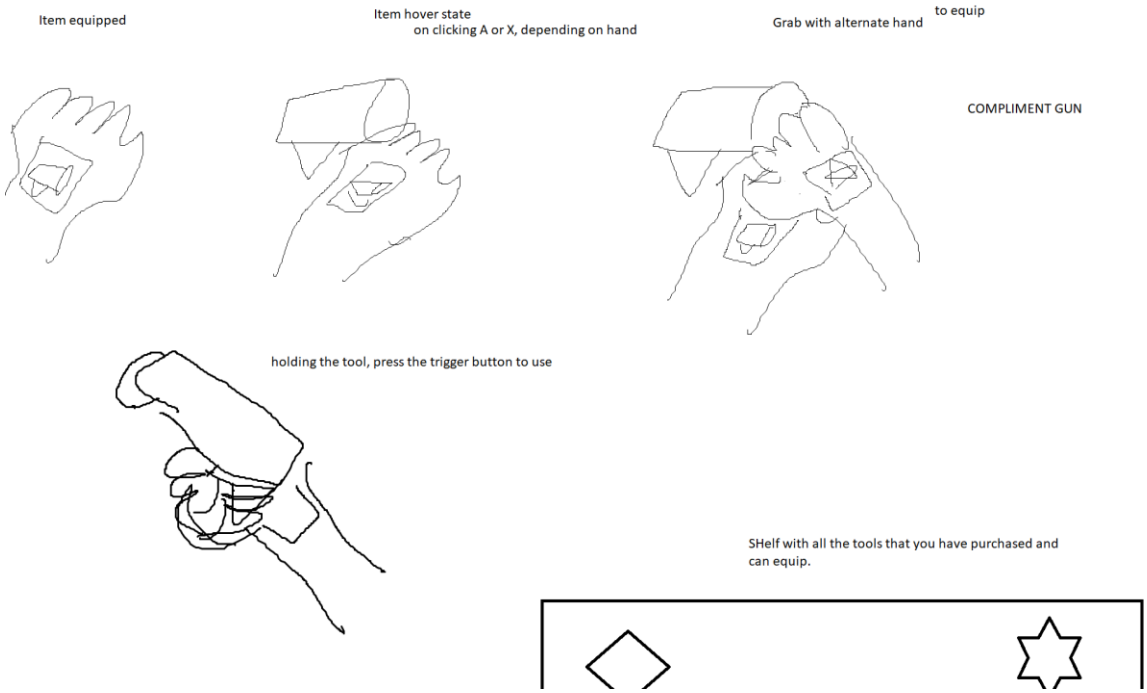
We built a simple tutorial for new players to follow that was remarkably effective. We had a series of 4 rooms, each showing the player more about the system and having them learn at their own pace. The first two rooms showed the player the items they need to pick up, the third room showed them how to throw objects, and the fourth room showed them the customer loop. The key element to making sure the tutorial was solid enough to be followed independently of instruction was including text instructions near the doorway. That way, if the player was lost about what to do and tried to leave the room without completing the tutorial objective, they were reminded what to do. This is the one effective use of UI: non time-sensitive directional information given to the player.

Players cannot skip the room until they hit the target and can review the text if they don't know what to do. The core focus, though, was a limited design with what looked like an obvious goal

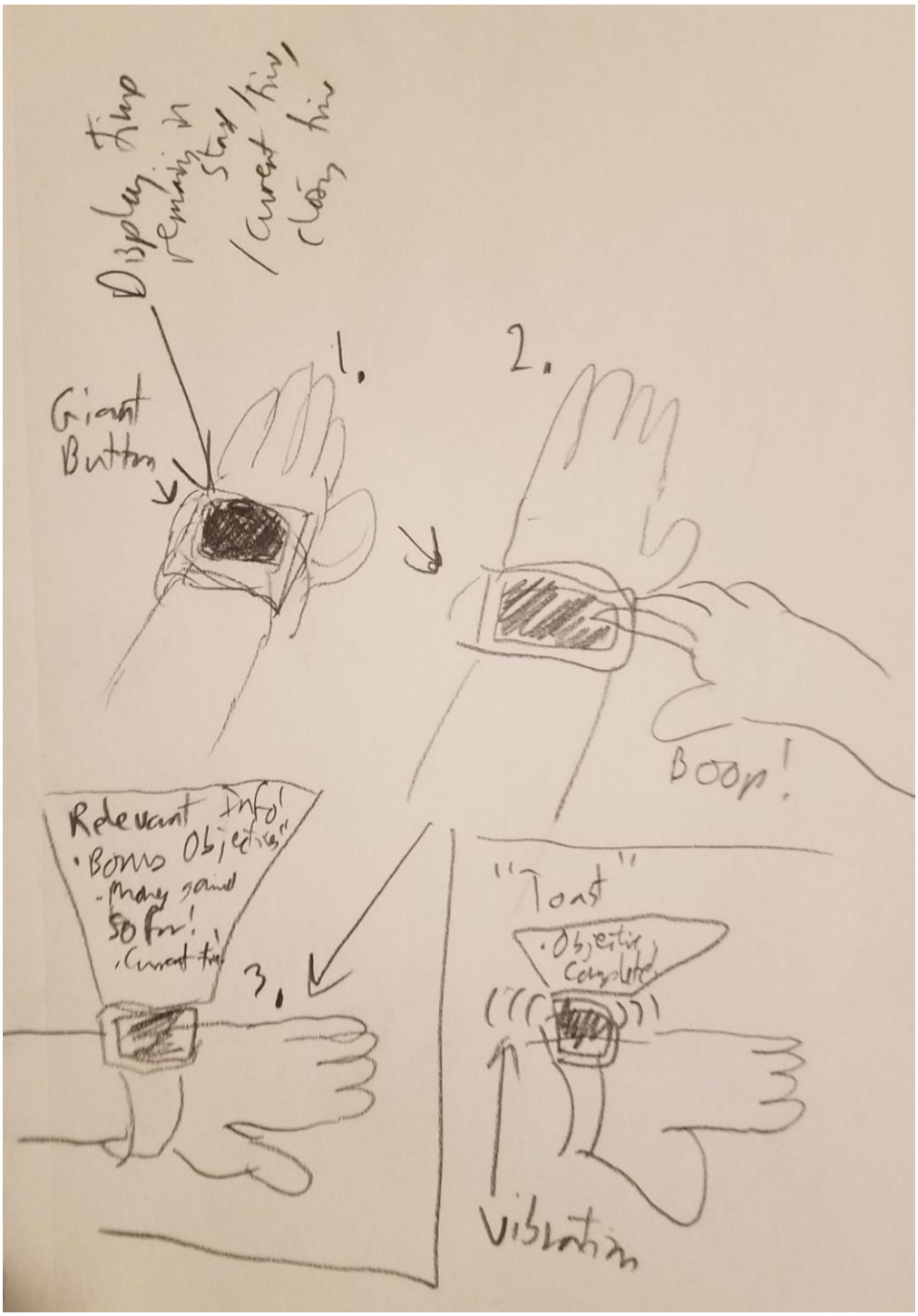
Lack of design doc caused major scope troubles

The biggest problem once we shifted to a randomized-style experience was how to give the player a sense of progression. There were many features that we implemented but then scrapped when we saw they weren't working or testing well. These included features like a tool system, wearable hats, a stage unlock system, and a wrist-watch UI. Before implementing any of these features, we should have

spent more time in the design phase of the project to determine the real player goal of implementing the feature and how we could reinforce those features through the main game loop. Instead of creating features that integrated well with the core experience, the features we implemented felt tacked-on and had too much maintenance and tutorial overhead to be polished features that would work well with our players. Rather than trying to implement different features, our time might have been better spent implementing additional stage variety to make the environments the player travels to more engaging with a more basic mechanic structure.



Concept for the Tool Gloves feature that was implemented, then removed. While functional, tools would have made the game more difficult to balance and required additional tutorials.



We couldn't figure out how to effectively train the player to pay attention to the watch feature at the right times

Progression and decision interactions: simpler is better

We experimented with many different types of interactions the player could use to interact with the world around them. Some of these were button-push style mechanics, a pull-down for starting a stage, and a rotating knob the player could use for selection. Ultimately, the best interaction types were the simplest. The “hand-scanner” UI that we implemented was very easy for players to understand, as it provided visual, tactile, and auditory feedback to indicate the player they could interact with it to proceed.

Even though the player character doesn’t have human hands, every player still intuitively understands what to do here, especially when the vibration feedback starts

The robot character was the perfect amount of world flavor we needed

Once advanced feature we really wanted to implement was a full-body player character. We implemented a full-body IK rig, and Matt Hazelworth initially created a human male body for the player.

While the human rig was functional, the body felt off because of the way the rig sometimes needed to contort to match the way the player moved in VR. After talking with a fellow student in the George Mason Computer Game Design Program, our team decided to change the player character from a human to a robot. This gave us the opportunity to implement a system of customization on the character and for the character to feel more natural in terms of direct control through the IK rig.

Even though we removed any game system impact from part unlocks, players still have fun with it. The strangeness of the robot body disguises any IK visual issues

Final Thoughts

Our company released *Give the People What They Want* to Steam on May 6, 2022. When we began the project, we wanted to make a game that we could expand on after release if we chose to do so. We believe we succeeded on this front, as our game is expandable both through the addition of new stages into the structure of the game as well as adding new customization options for the player.

However, we may not be able to keep up and maintain the game for a different reason: the rapidly changing landscape of VR hardware and software. As we were working on the game, the Oculus

Rift S was discontinued by Oculus in favor of the Oculus Quest. While we originally intended to release for the Quest as well as the Rift S, we ultimately decided that developing for the Quest was out of scope for us. In addition, late in the game development cycle we needed to update the Oculus SDK because the Oculus native functionality was going to be retired in favor of OpenXR. It can be challenging to keep up with the changing VR technical landscape, and it may take some time before VR can truly be a stable development and deployment platform. As many of our team members do not have as much time as we had while in school, we cut back on our update plans for the game to focus only on necessary maintenance.

VR game development is not for the faint of heart. From the challenges of ensuring you can begin development with the right hardware to adapting toolkits to your game to working with the design constraints of VR, there can be lots of unexpected roadblocks you will need time, patience, and expertise to overcome. There were some days where we failed completely to make progress towards completing a feature and needed to roll back changes because the complexity of VR made those changes unmaintainable from a programming perspective or unplayable from a design perspective. For any teams or individuals who are looking into VR development, we hope that this retrospective has shown you the importance of starting small, focusing on design, and playtesting well to ensure you can create an enjoyable VR game.

If you'd like to see the results of our development process, you can find our game now available for purchase on Steam.
(https://store.steampowered.com/app/1594440/Give_the_People_What_They_Want/)

The team of Starlight Vintage Studios taking a day off. Don't forget to take breaks and do fun things with your team, too!