Project Valence

Project Experience Report

April 5th, 2024

Nicolas Ansell, Julian Gonzales, Michael Osachoff, Cameron Wilson

Contents

Table of Figures

Project Purpose & Overview

The gaming industry thrives not only on innovation, but also on the connection it fosters among its community of players. Casual strategy games provide a platform for individuals to immerse themselves in quick, yet interesting game sessions. This allows the play to engage in experiences that tap into both pre-existing and new mechanics. Since the members of our team grew up surrounded in a culture of gaming, we were raised in a community of people who loved to spend their time playing games together. So as a team, we hoped to give back to the amazing community that we knew and loved, and bring to fruition some of our own ideas in a casual strategy game that combines some of our favourite game mechanics and ideas throughout the years.

One of the largest inspirations for creating Project Valence was a game called ‘PlateUp!’, where chefs in a kitchen frantically prepare recipes for customers, wash dishes, and bus tables. We wanted to take this idea of a frenzied management game and bring it into the world of chemistry. The possibilities for products that can be synthesized in a laboratory are nearly endless, and the complexity of different machines to interact with allows for an interesting and diverse experience with the opportunity for each play to be different from your last.

Project Valence is a 2D, top-down casual strategy game. Players are spawned in a chemistry laboratory where their goal is to complete a contract provided by various clients. With the option to select a new one each day, these contracts require the player to synthesize and deposit products generated in their laboratory. Failure to complete the day before the time runs out will cause the player to complete their run. After successfully completing each day, the player will receive payment for the contract and move on to the next day. The products are created by combining the correct ingredients in the correct machines, otherwise garbage will be created. The recipes for these products are provided to the user to allow them to create the correct result. The physical structures of these labs are from a few predefined sets, but the positions of the machines and depots of ingredients are randomly generated to create a unique experience for every playthrough.

Planning

Customers

Deciding our North Star customer was an interesting one to say the least. Especially for a video game, the targeted audience can make or break all the hard work put in. One discussion that we often had was between targeting more for the competitive and more hardcore players or towards the more casual and friendly side of the video game community. Both sides of the community have their own kind of advantages and disadvantages.

The competitive community tends to be tighter and as a result gives you a more loyal player base that follows the organization through multiple iterations of the game. Take ‘The Binding of Isaac’, because of its loyal fanbase their development company were able to push the intellectual property (IP) into four and counting releases. However, due to the competitive nature of the community they tend to have a smaller player base at the same time.

When we thought of this game, we only talked about how fun it would be to play with our friends and make some chaotic memories. With this in mind, we opted to lean more towards the casual community of players as it will allow us to reach a broader range of audiences. It will also allow players the freedom to play without the pressure of having to commit an absurd amount of time to be good at the game.

Envisioned Mechanics

Initially, we were all on the same page when it came to the basic mechanics of the game. As shown in figure 1 & 2. The process from main selection screen, the basic game mechanics such as taking/putting ingredients, machine functions and timers, and the success/fail states.

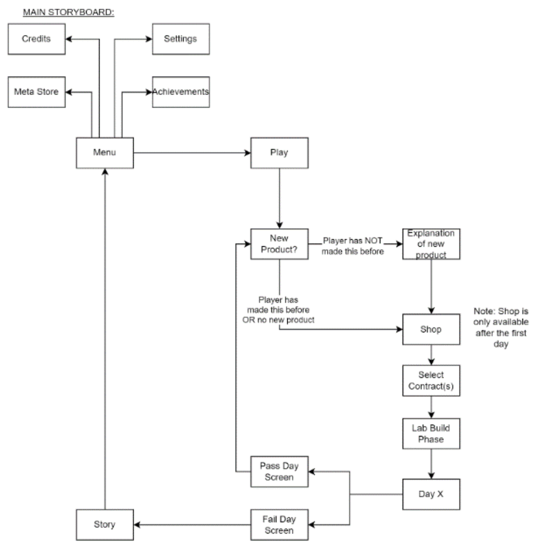

MAIN STORYBOARD:
Credits
Settings
Meta Store
Achievements
Menu
Play
New Product?
Player has NOT made this before
Explanation of new product
Player has made this before OR no new product
Shop
Note: Shop is only available after the first day
Select Contract(s)
Lab Build Phase
Pass Day Screen
Day X
Story
Fail Day Screen

Figure 1 Storyboard.

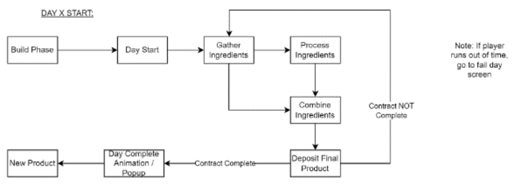


Figure 2 Single Round Storyboard.

Where we got mixed up however was on the details of those mechanics. We would ask ourselves a series of important questions. How many ingredients can a player hold? Where should machines spawn? How fast should we make a full day be? Thanks to our mentor Adam, he was able to point us in the right direction into solving these problems by having us take these mechanics into a board game setting. Where we took the time to sit down on a table and play with minifigures to essentially act out our game in real time and physical form. In doing this exercise, we were able to eliminate almost all our confusion. This enabled us to finally divide work individually and get working on the game at a rapid pace.

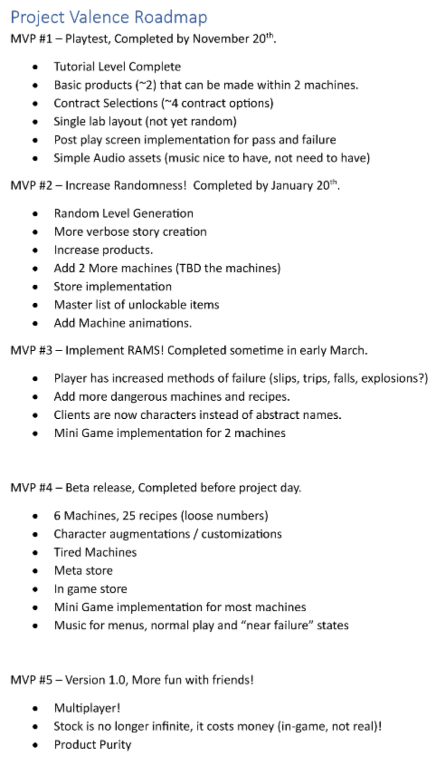
Technology

When developing a game, we obviously needed a game engine. We had a couple of options to choose from, including Unreal, Unity, and Godot game engines. In the end we chose Godot as it fit our criteria more than the others. Firstly, Godot is a lightweight game engine which in turn doesn't have as many features as the other two but is easier to work with, and with our plan of a 2D game, the features that we were missing out on were not necessary. Being a lightweight engine also makes it very friendly to lower end computers and overall, just a smoother experience. Unlike the other two, Godot has its own script editor, which grants us the ability to write code within the game engine itself instead of using a third-party software such as VSCode to write our scripts in, save, then double check if it was saved correctly in the engine. Due to Godot being open source, it has a massive and quickly growing community. It is constantly being updated and fixed, and there is a lot of help available online. It is also free and will always be free. Compared to some other game engines, Godot will always be ours and we won't have to worry about its license and fees.

One of the struggles we had with Godot is understanding pull requests and merging other branches, as reading the scene files is quite difficult with various changes. Fork, a visual git GUI was an immense help for us understanding the differences of each file from merging branches. Another heavily used software that was indispensable for us is Aseprite. Asperite is a software made for pixel-art and pixel-art assets such as animations and sprites or any kind of graphics for that matter. Game assets are a vital part of a video game and Aseprite helped us greatly in portraying the kind of visual feedback we wanted the players to see. Lastly, we also used an audio software called Audacity to trim and modify audio assets for the game sounds.

Envisioned Roadmap

Our initial roadmap had four MVPs planned to be completed by project day, and a final fifth MVP if we would continue to develop our game in the future. Each of these MVPs would contain enough features to justify a new pre-alpha release of our game. Like other games within the industry, these releases would be incremental in their design, building off strong core concepts from the previous release. The initial envisioned roadmap can be seen in figure 3 below.



Project Valence Roadmap

MVP #1 – Playtest, Completed by November 20th.

- Tutorial Level Complete
- Basic products (~2) that can be made within 2 machines.
- Contract Selections (~4 contract options)
- Single lab layout (not yet random)
- Post play screen implementation for pass and failure
- Simple Audio assets (music nice to have, not need to have)

MVP #2 – Increase Randomness! Completed by January 20th.

- Random Level Generation
- More verbose story creation
- Increase products.
- Add 2 More machines (TBD the machines)
- Store implementation
- Master list of unlockable items
- Add Machine animations.

MVP #3 – Implement RAMS! Completed sometime in early March.

- Player has increased methods of failure (slips, trips, falls, explosions?)
- Add more dangerous machines and recipes.
- Clients are now characters instead of abstract names.
- Mini Game implementation for 2 machines

MVP #4 – Beta release, Completed before project day.

- 6 Machines, 25 recipes (loose numbers)
- Character augmentations / customizations
- Tired Machines
- Meta store
- In game store
- Mini Game implementation for most machines
- Music for menus, normal play and "near failure" states

MVP #5 – Version 1.0, More fun with friends!

- Multiplayer!
- Stock is no longer infinite, it costs money (in-game, not real)!
- Product Purity

Figure 3 Initial Roadmap

The first MVP would be our "Playtest", and it would have all our basic game mechanics accompanied with a tutorial level for players to grasp the core concepts. We wanted to create a demo/tutorial level that allowed the player to walk around and interact with items and machines within their laboratory. Players would learn concepts such as picking up and putting down items and loading them into machines to create a new product. We also hoped to have basic "contract" tracking where players can see their progress to their final goal, a pass state upon reaching that final goal, and a failure state when the player runs out of time.

In our second MVP, we envisioned expanding upon our playtest by adding more machines and products, as well as implement forms of random level generation and store implementations. While we did not define our new machines and products, we wanted to create unique recipes that won't take identical orders of products and machines. For instance, a recipe may take item A and item B make item C in a mixer, but item C and item D could make item E (the contracted product) in a separator. Then a new recipe may use similar items, but would require a different order, such as item A and item D in separator would make item F. We hoped that accompanying these items with unique visual sprites/art would allow the player to create their contracted products without needing to read the item's name. The store implementations would allow the player to spend their money earned in previous days to have more machines or upgrade their speed. Random level generation would allow lab layouts to be unique and offer a fresh sense for each of the player's days.

For the third and fourth MVPs that we envisioned creating by project day, we hoped to flesh out our game to include various user experience improvements. This included an out-of-game store, mini-game implementations, additional methods of failures such as fire or flooding, and of course additional products and machines to add more unique options for the player. The out-of-game store would include permanent upgrades for each player that can be purchased from points earned at the end of in-game runs, and these would reward the player for progressing through the game. Mini-game implementations would allow players to speed up an in-progress machine by performing specific interactions on it. For instance, spinning a valve/cog on the mixer would make the beater move faster and the mixer finish quicker. Additional methods of failure such as fire or flooding would allow players to fail in other ways than running out of time, allowing for more risks in the player's actions.

Then for our fifth MVP to be targeted at after project day, we would look to include aspects such as Multiplayer, limited stock for ingredients, and even a product purity based on how well players are able to synthesize their products. We envisioned multiplayer having a maximum of four players connected from different devices, with the possibility of having a "couch co-op" where instead players will play on the same device with multiple controllers. Limited stock for items would be a possible way for players to run out items as an additional form of failure, and we hoped this could force the player to be more conscious of what products they may attempt to make. Finally, product purity would be a concept that we encourage the players to make products "perfectly" by either running the machines for the correct time or ensuring the right amount of an ingredient is added to the machine before starting it.

Results & Outcomes

Reworked Roadmap

Shortly after creating our initial envisioned roadmap, we asked for feedback from our mentor, Adam Tilson, on whether it would be accurate and viable for the scope and timeframe of our Capstone project. He recommended several changes based on his experience as a game designer for the past ten years in his undergraduate and graduate studies, and hosting Global Game Jams.

His first recommendation was to develop and design the tutorial as the last feature of the game since it would contain quick introductions to all the other features, and it would be best to design the tutorial around the features, instead of vice-versa. Secondly, he said that random level generation and multiplayer concepts would have a larger complexity and scope to implement than we would likely imagine, and to start early when developing these features to properly flesh them out and polish by project day. The final large recommendation he provided for the roadmap and our game design was to first ensure that the core concepts and features within the game of interacting with items and machines to create products was solid. These would need to be completed and easily extendable before attempting to add too many features that were focused solely on user experience.

After working through the recommendations that Adam provided and our motivations for creating the game, we decided to rework our roadmap early on to better represent our priorities. As per the recommendation and common practices in the video game industry, we opted to move our tutorial to the last MVP before project day. This would allow us to prioritize on the core concepts of our game's design without worrying on how they may be implemented in the tutorial, or having to go back to the tutorial to make sure it is implemented and shown properly as well.

We noticed that in our initial ideas and goals for the game we desired the multiplayer and random level generation aspects. Many of the games that we had inspiration from such as 'PlateUp!' were implemented with multiplayer and had unique levels that created an enjoyable experience for players to enjoy with their friends. For this reason, we decided to bring forward our multiplayer and random level generation to earlier MVPs with implementation steps for each.

Our multiplayer would include a headed server and client implementation (one player hosts, and friends can join the match) with Local Area Network (LAN) joining as a 'step 1' for project day. Then our 'step 2' would be to allow for headless servers and an easier hosting/joining process than a LAN setup with port forwarding.

The random level generation was created with four steps total. First, we would create static laboratories that have the full capabilities to complete contracts within them. Second, we would randomize the placements of machines, item depots, counter tops, and any other interactable objects within the laboratories while ensuring all of them are accessible. Thirdly, we wanted to build the lab around the zones of available machines, instead of placing the machines within static lab layouts. This would require "tiled" areas or "rooms" that the laboratory would essentially procedurally generate and create the architecture of the building around. Our fourth and final step would include a "build phase" that the player could pick up and place their machines and/or item depots within the laboratory to an optimal place. For project day we aimed to complete up to the second step of randomized interactable objects within a static layout.

We also pushed back several user experience features within the game that would be out of scope after including the core concepts, multiplayer, and random level generation. This includes aspects such as custom audio assets and music, in-game and out-of-game stores, character customization, and machine mini-games. These were all things that we wanted to include within our game but would be out of scope for the eight-month project of Capstone, so we kept them in our roadmap but pushed to an MVP after project day.

Overall, we were satisfied with how our reworked roadmap and change of scope applied to our game, Project Valence. The new aspect of random level generation received positive feedback from our user tests, and many of our end-users expressed interest in multiplayer capabilities. Focusing on our core game concepts being solid before adding too many user experiences allowed us to reduce bugs and create DRY (Do not Repeat Yourself) code before building upon it. This also allows our envisioned future MVPs to be implemented more effectively and user-tested in defined releases.

End Product

The end product we created was a 2D, top-down casual strategy game called Project Valence. Each day, a player can select a contract to synthesize products with the given machines in their laboratory. If they combine the correct ingredients in the correct order, they can make their contract's item(s), else they will make unusable garbage. After successfully completing each day, the player will receive payment for the contract, and they proceed to the next day to again select a new contract. Failing to complete the contract within the day cycle will end the player's day and moreover end the player's "run".

The player's laboratory has a physical structure determined by a few predefined sets, but the positions of the machines and item depots are randomly generated to create a unique experience. There are also hazards that can occur within the lab such as fire and explosions when the oven is left to cook something for too long, or flooding when the sink is left to run for too long after filling a container.

Finally, our game has multiplayer functionality to allow players to play with their friends over a LAN connection, with a four-player maximum. This is done through a headed server (the server is also a player, simply hosting it for the clients) and clients can connect to it on the host's IP address. Like many other LAN-based games within the industry where players can join on different devices, the host can set up port-forwarding within their router to allow players to connect across the Internet. While this may be a valuable experience for players, port-forwarding and opening your device to external sources could pose as a security threat to your network, so users should not let untrusted sources connect whenever possible.

Lessons Learned

One thing we learned throughout this project is how important game art and assets are throughout the development process. Most of our game assets were used from the ModernInteriors pack (LimeZu), which was selected in the beginning of our development. Although it's an amazing general pack made for the masses, it unfortunately doesn't quite fit the laboratory setting that we were envisioning. We had to play around quite a bit with different assets to be adequate for our theme. Various items and specific machines were also not included in the game assets that we have bought; therefore, it was necessary to have a team member dedicated to making these game assets the whole project. There is also a slight issue with how the bottom walls are portrayed (Figure 4: Wall issue). It looked a little weird just sticking out, and there was no solution to this from the game asset pack. Fixing the visual issue ourselves would have taken too long, so it was left as is.

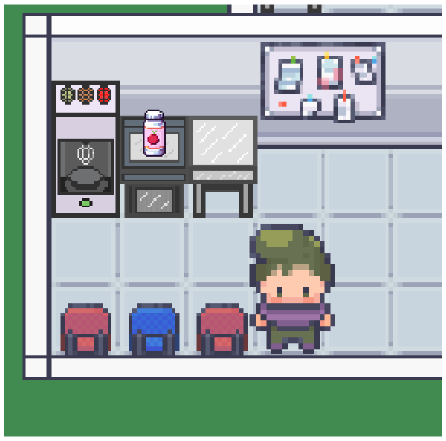
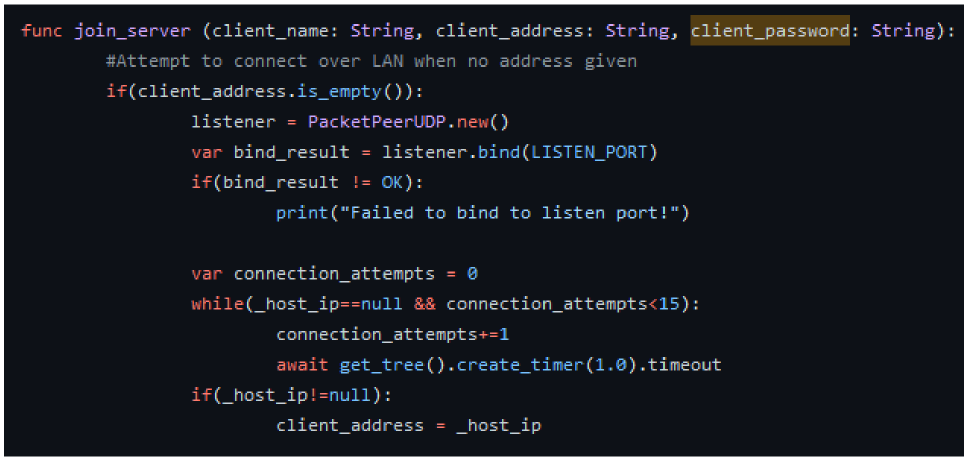


Figure 4 Wall Issue.

Multiplayer

One of the biggest struggles for developing our game was in bringing our game from only singleplayer functionality to full multiplayer functionality. We had multiple aspects for multiplayer that needed to be created to tie the entire experience together. For instance, we need to have players connect to a server and shown through a lobby, spawned in the same game and placed within the same laboratory, interactions and game states duplicated, and errors shown for when each of these problems may inevitably fail.

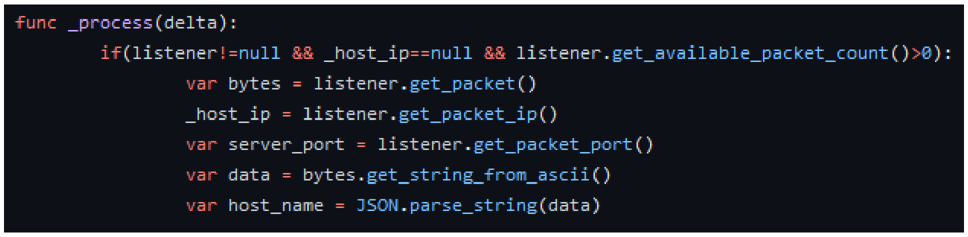
For players connecting and shown in the same lobby, we required a server to be started on the host, and clients to connect to it through the required IP address. When initially testing on localhost, this functionality was simple to implement as it required essentially no networking functionality. After attempting to create this lobby with two different devices, clients were unable to find the host's IP address. This was resolved by broadcasting the game lobby to the host's last-octet broadcast address and listening port (ex: 172.16.1.255:25585) for clients to listen on. The process for listening to the broadcast for the client can be seen in Figure 5 below, where the client attempts to retrieve an address from the listening port. If the player does not input a connection address in the GUI, then the client creates a listener object that binds to the listen port. From here, the client waits fifteen seconds before timing out, and if it receives a packet from a host, it sets the clients address and attempts to connect. Within the process function (which can run every frame), if a listener exists, the client attempts to listen to the broadcast address for any incoming packets to set the host ip, see in Figure 6 below.



```
func join_server (client_name: String, client_address: String, client_password: String):
	#Attempt to connect over LAN when no address given
	if(client_address.is_empty()):
		listener = PacketPeerUDP.new()
		var bind_result = listener.bind(LISTEN_PORT)
		if(bind_result != OK):
			print("Failed to bind to listen port!")

		var connection_attempts = 0
		while(_host_ip==null && connection_attempts<15):
			connection_attempts+=1
			await get_tree().create_timer(1.0).timeout
		if(_host_ip!=null):
			client_address = _host_ip
```

Figure 5 Multiplayer Joining Server Code.



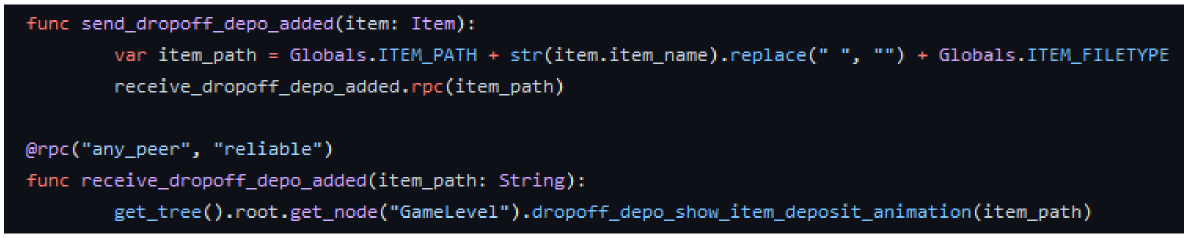
```
func _process(delta):
	if(listener!=null && _host_ip==null && listener.get_available_packet_count()>0):
		var bytes = listener.get_packet()
		_host_ip = listener.get_packet_ip()
		var server_port = listener.get_packet_port()
		var data = bytes.get_string_from_ascii()
		var host_name = JSON.parse_string(data)
```

Figure 6 Multiplayer Packet Listener.

With the listener functionality implemented, clients were then able to listen to all incoming broadcasts and connect over a LAN network. The lesson learned for multiplayer networking was that localhost functionality often requires additional components to work over a LAN or Internet connection.

For a significant portion of the rest of multiplayer, Remote Procedural Calls (RPCs) are used to synchronize states between the server and the clients. Originally, we attempted to use Godot's built-in 'MultiplayerSynchronizer' class to synchronize the states. This proved to be difficult for any user-defined properties, and we ended up only using this built-in class for synchronizing the players' positions within the lab.

For cases where other aspects of the game need to be synchronized, RPCs were used. Within Godot, an RPC simply allows a client to call a function within the same class on another client. An example of this can be seen in Figure 7 below showing how our 'dropoff depo' (which receives final products) replicates the deposit animation for other clients. Since RPC functions can only accept parameters of primitive types (int, string, dictionary, etc.) the Item to be shown depositing in the depo needs to be converted to a string path to be instantiated on other clients (which can include the server). The function is then called through an RPC which sends a TCP packet to all other clients to execute the function. This allows the other clients to show the item deposit animation for the dropoff depo on their end. RPCs are denoted through an annotation, and in figure 7 we can see that the function can be called by any client by “any peer” (instead of server-only by “authority”) and it is a reliable/TCP packet to be used by “reliable” (instead of UDP by “unreliable”).



```
func send_dropoff_depo_added(item: Item):
        var item_path = Globals.ITEM_PATH + str(item.item_name).replace(" ", "") + Globals.ITEM_FILETYPE
        receive_dropoff_depo_added.rpc(item_path)

@rpc("any_peer", "reliable")
func receive_dropoff_depo_added(item_path: String):
        get_tree().root.get_node("GameLevel").dropoff_depo_show_item_deposit_animation(item_path)
```

Figure 7 Example RPC call.

These Remote Procedural Calls within Godot allowed us to synchronize our game states to ensure the game plays and feels the same for all players. So, the lesson learned for multiplayer here was that the first option that may appear to work for your solutionm may not have the proper capabilities to perform the functionality that you desire. Furthermore, synchronizing data manually may fit better, but it may depend on your situation.

Working together

Game development presents unique challenges in terms of spreading out work, and much of our time at the beginning of the project was spent learning how game development works. A lot of this work had to be done as a group as to keep everyone on the same page. One thing we should have done was paper prototype our essential gameplay mechanics earlier. In an early meeting with our mentor, we had already built basics and had not taken the time to do this. Paper prototyping a game is much different than a user interface, but you should be able to play and enjoy the basic mechanics using paper, dice, cards, and other physical mediums. Once this was completed, we could have split off better when learning features. We feel having a lack of clear role definitions at the start and lack of game development knowledge contributed to this mistake. One major success was using sprints and task boards on a simplified basis, so we knew what had to be done and who was doing it, but we opted to skip time estimations as we determined they were unhelpful in such a small group.

Advice for Future Groups

When it comes to game development, we highly recommend thinking about your game assets at the start of the project due to the implications they bring. For a capstone project, buying premade game assets is probably the way to go as we learned from our experience. It requires a significant amount of time and resources to make game assets from scratch. However, if you are going to buy game assets, make sure they are correct and have most of everything you need. Realizing that the assets you have don't correctly have the visual items you require is a bit of a pain for developers in the long run.

Another recommendation we have when it comes to game development is to research the development standard for your game engine thoroughly. The development process for a game is quite different from other software development processes as every game engine has its own features and shortcuts for development. A variety of coding principles or coding patterns might not be available on certain engines or is streamlined and made easier through various of steps. Godot, for example, has its nodes, scripts, and scenes system to help developers understand how Godot builds its code.

One of the things that worked out favourably for us and we would highly recommend is the dedicated meeting and work time we had as a team every week outside of scrum time. Having this dedicated time together every week allowed us to always be in the same page weekly, and make sure that no one is working on features that is not necessary and contributes to the project overall. It also pressures us to always be working on the project throughout the week so that we always had something to show during the meetings. Overall, it is a great way of making sure the group is progressing through our deliverables in time.

Acknowledgements

We would like to acknowledge and thank Dr. Tim Maciag and Adam Tilson for their feedback, support, and guidance throughout our project. Throughout our weekly/bi-weekly scrums we were able to receive valuable notes and comments about our current progression from Dr. Tim Maciag. With this, he helped us to determine what features would likely be desired by players and when to begin our user testing processes. Adam Tilson provided us valuable guidance and suggestions based on his video game design experience to ensure we would be on the correct track in our designs and goals with this project. Finally, we would like to give a thanks to all our testers who gave us feedback about what they liked and wanted to see changed in Project Valence.

References

LimeZu. (n.d.). *LimeZu*. itch.io. https://limezu.itch.io/