1 Playtesting

The first game concept I came up with was a turn-based three-dimensional boardgame. Convinced by the quality of this idea, I eagerly begun to build a physical prototype out of paper, chess pieces and other things I found. To simulate a three dimensional space, I simply put three grids next to another, which everyone who played the game understood surprisingly well. Initialy, I was moving the pieces around and tried to come up with rules, which didn't quite work out, so I asked my brother to play with me (that was on Feburary 26). With some initial difficulties (he didn't quite know what I was expecting him to do), we finally came up with a set of rules which I won't mention now. The game was fairly fun, however there were loats of balancing issues, so two days later I asked another family member for help so that I can try out different variations of the game. Playing the game with a different person was a completely different experience, and a lot of flaws were discovered in the rules. The biggest issue was that my playtester didn't quite the game's goal. Due to this experience, and my insistance on using GoDot as a game engine, which, as it turned out, might not be the best choice for 3D games, I chose to go with I different idea.

This time I wanted to create a seagull simulator, which was about stealing food from other people and completing quests. I did draw a map on paper and made some seagulls and humans from cardboard, tested it out myself, and two days after abandoning the previous idea, I was ready for a fresh playtest with my brother. Stealing food became boring very soon, however he was very keen on completing the quests. Overall, playing this game worked pretty well, and after doing another playtest with a friend I descided to implement a first physical prototype. I descided that I want to go for pixel art and asked a friend who's doing product design for some advice on this particular artstyle. When I included the assets in my prototype, however, I quickly realized that using pixel art won't work at all: assets had different sizes, the pixels also had different sizes, and it took ages to find the right assets. I initially though that I could simply draw my own sprites, but I quickly realized that this was not an option either. As a result, I descided to change the game concept again, and to come up with an idea based on the assets that I found, an approach that seemed to be appropriate at this point, especially because I was running out of time.

I finally found a complete asset pack that I liked at kenneys.nl, and I descided to create a multiplayer platformer. For the time being I didn't do any playtesting and only focussed on learning godot's multiplayer API. Creating a project plan for assignment 1 helped me a lot to get more clarity over how the project should look like in the end. I started to implement the authentication system first, and the invitation functionallity and lobby afterwards. Once this was done, I started to implement the actual game.

On April 18, I was ready to do the first digital playtest. Authentication, inviting players, starting the game from a lobby and playing two levels, as well as switching to the next level, did already work at this stage. It should be noted that the bridge in level 1 did still work as was described in the project plan. In this playtest I wanted to find out if the game is actually playable. My brother and his girlfriend happily volunteered to play the game on two separate computers, and were able to see the other's screen. In order to prepare for more formal playtesting session that I would have in the future, I started the session by telling them how playtesting is going to help me, that it's an unfinished prototype and that all their mistakes are going to help me. I also asked them to think out loud. I asked them questions about their experience with computer games, and initially this information appeared to be irrelevant, until I realized it's importance towards the end of the

playtesting phase. Opon loggin in, they were confused by the font which is all capital letters. I later changed the authentication system to treat everything as lower case letters, which solved this issue and has the benefit that players with usernames that only differ in case cannot be confused anymore, an issue that occurs with almost every system that supports usernames. To my surprise, when trying to send out an invitation and starting the game through the lobby, they managed to break the system in ways I never imagined to be possible. In the end, I had to intervene and start the game for them. Just as every other pair I did playtesting with later on, they started the game by trying to work out how to control the game, and quickly realized that they could jump on top of each other. However, they mentioned that it would be great to have a screen that shows the available controls, however I chose to not implement this suggestion yet. They were also trolling each other by stepping off the button when the other player was on the bridge or staying on the button when they other player wanted to leave the whole under the bridge, a behaviour that would have been great to encourge, however I didn't encounter this in the following playtesting sessions.

In the second level, they found a way to use the spring board in a way that wasn't intended. Due to the way the code worked at that time, they could simply stay on the board and their character would always jump higher once he touched the board.

Moreover, they had the assumption that they'd both needed to enter the exit at the same time, which was not the case but was changed just afterwards. As the level was a bit hard, they needed around seven attempt to reach the exit, but when I asked them some questions after they played the game, they confirmed that they were positively surprised by the difficuly, which motivated them to finish the level.

The next day, I was doing a less formal, but remote playtest. This time I wanted to find out if it was possible to play the game with someone who's not on the same network as I am. I did setup port forwarding on my router at home so that the game servers could be reached from outside and so that I could share the executable client application with little effort. The playtester showed her screen so that I could see what she was doing. Upon reaching level 2, she didn't understand how the spring board would work, so in an attempt to explain this concept within the game, another spring board which didn't require the player to jump as high as the second one was added right at the beginning.

Moreover, as her screen was quadratic, she couldn't see certain elements that I could see, an issue that I haven't thought about before.

I then organised another two playtesting session, however due to organizational issues they were both on the 23th, one at noon, one in the afternoon.

This time I was doing a playtest with people that normally don't play games. Surprisingly, they didn't understand that the buttons were colour coded, so in order to make this more clear, I did put two buttons next to one another right in the beginning of level 1, which solved this issue. The also had issue with spring boards because were too tiny for them and they didn't quite understand the way they work. Moreover, when they realized that they had to try the second level again because they fell off, their motivation was immediately gone, which is why the second level is now a bit easier to satisfy all potential players.

Another issue that I noted was that on one of the playtesters screen, the other player's character was not at the position where it was supposed to be but rather move all over the place. I attributed this issue to a bad internet connection

When doing the next playtesting session, the same issue that the player was all over the place occured, which actually made it impossible to properly test the game. After starting the game several times I found out that the game was running soothly on my computer, which was also running the server, but on some clients, not on all, it was impossible to play the game because the other player's character would always change the position. I decided to not fix this issue because it would have been a very difficult endeavor to spot this issue as I don't have the capacities to test on another network.

In this version of the game, I added coins, increased the size of the spring board and changed the second level to better explain how spring boards work, which was doing a good job.

I also tried to playtest the game with people that I don't know by putting on a post in the playtesting channel in the Module's Teams Group, however I didn't get any replies, which is why this game was only tested with people I know.

2 Changes

Firstly, on the playerList screen, I only included a list that shows players which are currently logged in and not playing a game. The reason for this change is that it took significantly more time than I expected to set up the aforementioned list properly.

Moreover, I also removed the entire lobby screen because I realized that the lobby wasn't finished at all and it was causing issues when more than two players were logged in because if someone would enter or leave the lobby, the player list on the previous screen wouldn't always update. I decided to decouple and refactor the entire server code, which allowed me to at least get the invitations to work properly.

On the playerList screen, there is now a text which shows the currently logged in user, which is helpful for debugging purposes and for players alike

Regarding the game itself, I did modify the levels quite strongly based on how players interacted with the levels. Moveable boxes, like the ones planned in level 2 were also removed as I chose to focus on other issues. As mentioned in the playtesting section, I added the option that platforms can appear and stay visible if two players simultaneously step onto the buttons. The reason for this change will be explained in the playtesting section.

3 Design Patterns

The Client Project uses Godot's build-in architectural design pattern, which, compared to a classical ECS, uses Nodes instead of entities with components, favours inheritance over composition, while still allowing the later to be used on a higher level, and, similar to systems in ECS, uses what the Godot developers call servers and lets Nodes have their own logic in the form of scripts.

This architectural pattern was mainly chosen because Godot is build around this pattern, thus being well integrated with Godot's design philosophy, making it very userfriendly and leightweight, while still providing good performance. Moreover, as there is no destinction between scenes and prefabs - everything is a node in godot - it become easier to combine and compose nodes which

increases the reuse of parts of the game. Considering the complexity of the game there should not be any performance issues compared to a traditional ECS approach, however if optimization was required in the future, it would easily be possible to do so, for instance by only processing visible objects or by directly interfacing with Godot's low level servers for critical game code.

This pattern can be seen almost everywhere in the Client project. For instance, the scenes in the Levels folder make heavy use of this pattern. They all inherit from the superclass Level, which provides funcionality common to all levels. Composition can still be found in the individual level nodes, which hold different nodes that make up the level, such as platforms or pressure plates.

In order to decrease coupling between objects, I used Godot's build in signaling system, which acts as an observer pattern. It is used everywhere for communication between nodes where a Composition or Aggregation cannot be assumed, or when making calls to a Singleton.

This can be found in the Services.gd Singletons of both the Services and the Client project. When receiving a remote procedure call, they simply emit a signal. Before refactoring the entrie Services project, thich was done with Singletons and simple function calls, which made it very difficult to get invitations to work properly or to extend anything.

Moreover, every pressure plate that inherits from PressurePlat.gd emits a custom signal when a player enters or exits it's area. This signal is then connected with an object that includes Platform.gd, such as Bridge.tscn, to trigger the Platform to be shown or hidden based on it's properties. As a side note, Platform.gd also makes use of the Command pattern by using a FuncRef to call the perform the correct action, hide or show, based on the objects properties.

Moreover, I made use of the Singleton pattern. Initially, I used way too many Singletons, which resulted in high coupling, especially in the Services project. Stritcly speaking, an Autoload in Godot, which I'm using as a Singleton, can be instanciated multiple times and is therefore, strictly speaking, not a Singleton.

Singletons are used for every script that has a NetworkMultiplayerENet: Gateway.gd, Authentication.gd, Services.gd and TokenTransfer.gd. Using Singletons in this context makes sense as the rpc's can only call functions on a remote machine that have the same Node path, and Singletons guarantee this. Moreover, the networking functionality is expected to stay available during the entire game and needs to be accessible by every other Node. It might make sense to only call functions on the Networking Singletons only with signals, however as these nodes are assumed to always be available they're called directly.

Singletons are also used to hold global information, such as the game_id or the players username in Global.gd in the client project, or to provide an interface to another API, such as Database.gd in Authenticate which provides methods to manipulate the underlaying database (Please note that in this case there in no database connected, but all data is stored in json plaintext on the same machine, however as the interface stays the same, one could easily connect the project with a database)

4 Networking

5 Security Considerations

6 If I were to tackle this project again