Bartendu

Project report

Jean Bou Raad
Lou Lefebvre
Vincent Thirouin
Emeline Tichit

EPITA 2024

https://areas0.github.io/website/

Contents

Introduction

This report is the final one that will be handed in alongside our game *Bartendu*.

Bartendu is a cooking bar simulation game where players cook and make drinks to satisfy customers in a restaurant. The goal of the game is to get as many points as possible in a limited amount of time. In order to earn points, players have to successfully meet the needs of their customers. As the overall gameplay was already explained in the book of specifications and has not changed since, we will not explain it in this final report.

Previously, a list of tasks was assigned to each group member. The aim of this report is to show how far each student has come, and present our achievements as well as our impressions on the results and this whole journey.

1 Book of specifications follow-up

The following is the tasks that were originally assigned to each student at the beginning of the project. Some of these tasks were picked up by another group member than the one who was supposed to complete them originally. We will explain later the shifts that happened.

Task	**Main**	**Substitute**
UI graphics	Lou	Emeline
UI implementation	Vincent	Lou
Character actions	Vincent	Jean
Multiplayer	Vincent	Lou
AI	Vincent	Jean
Tutorial	Emeline	Vincent
Level design	Emeline	Jean
Level implementation	Jean/Vincent	-
Object implementation	Emeline	Vincent
Particle Effects	Emeline	Vincent
3D modeling	Lou	Emeline
3D animation	Emeline	Lou
Music	Jean	Lou
Sound Effects	Emeline	Jean
Writing (Storyline)	Jean	Emeline
Website	Jean	Emeline
LaTeX	Emeline	-

Here is the update on all the tasks.

- **UI graphics**: The main student changed, Vincent took charge of it. Most of the art has been done.
- **UI implementation**: All the UI that were created were implemented.
- **Character actions and movements**: Already implemented for the first defense. They were fixed over the course of the semester.
- **Multiplayer**: Functional.
- **AI**: Implemented but not polished.
- **Tutorial**: Not done.
- **Level design**: Done, 3 levels designed.
- **Level implementation**: Done, 3 levels implemented.
- **Object implementation**: The main student changed, Vincent took charge of it. All objects were implemented.
- **Particle Effects**: Done but some of them were not implemented.
- **3D modeling**: Done and implemented.
- **3D animation**: Done and mostly implemented.

- **Music**: Done. Two partitions missing but as compensation a music manager was implemented into the project.
- **Sound Effects**: Not done.
- **Writing (Story-line)**: Done.
- **Website**: Up and running.
- **LaTeX**: Done.

2 Jean's Achievements

2.1 The website

During this semester, I developed a website. It is an essential tool for our project because it allows the user to discover the project and the game. It must be a great experience. Therefore, I worked a lot to create a functional and beautiful website.

2.1.1 Key steps of creation & features

The process of creating our website has been quite a great journey.

The first step was choosing between using a template or going from scratch. The second was the option chosen because it offered an occasion to deepen my knowledge of CSS and HTML. Indeed, before the project, I already had some experience with website building for personal projects and the TPE for the BAC. Hence, this project was a perfect occasion to learn even more about website building.

However, starting from scratch meant choosing at least some fundamental tools to work with: a skeleton. This was Bootstrap's role. Bootstrap offers some premade layouts and some tools to customize the layout of a website. But it also allows the developer to take liberties on the theme.

The design of our website was decided by the whole group. We wanted something simple and standard: a black theme with little contrast on the pages to keep everyone's eyes comfortable.

With those ideas in mind, I built a website from scratch with little iterations on it after the first release for the first defense.

The website features many pages:
— A classic homepage
— A download page with some documents
— A features page featuring some videos of the game (created for the last defense)
— A task issues page (reworked for the last defense)
— A page for the credits
— An "about us" page (reworked for the last defense)

Our website fits all the requirements for the last presentation and offers a diverse and rich experience with the latest improvements done for this defense.

The homepage features a small news section, a menu with some additional and useful links, and a small Easter-egg capable of changing the theme on it: it involved writing a little bit of JavaScript which I found would be interesting.

The download page is a place for all documents, executables, music appendixes made during the semester.

The feature page is a showcasing tool for our game with small videos and a gallery.

The task and issues page offer some details concerning how we have progressed overtime on the project. It includes some information on the bugs we have fixed.

The "about us" page includes a description of the group and all group members.

2.1.2 Latest updates

Some updates have been done to improve the website for this defense and some challenges arose.

The biggest difficulty being the creation of a layout that could provide enough information without becoming overloaded. This explains some reworks for the last defense. For example, the "about us" page, for the two first defenses, was filled with text as you can see here:

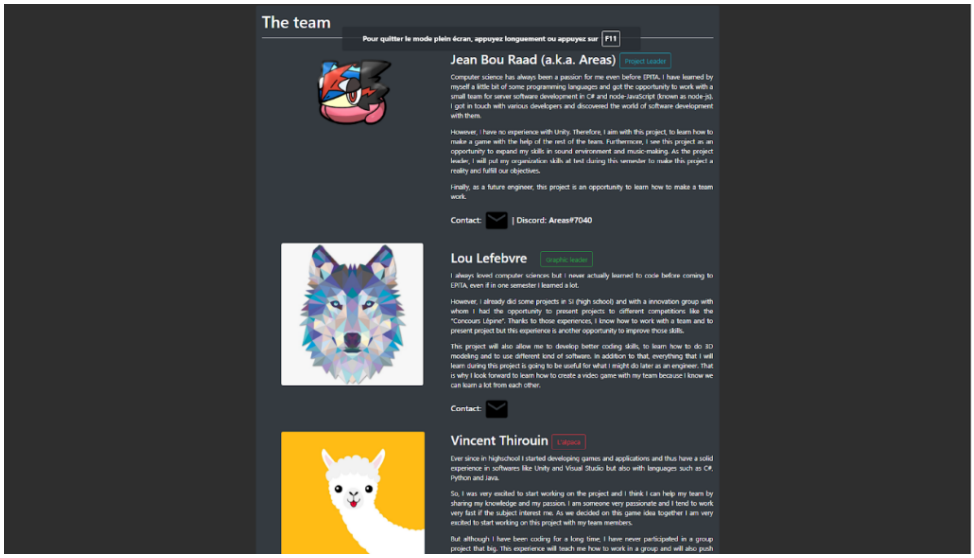

Figure 1: Old presentation layout (partial screenshot)

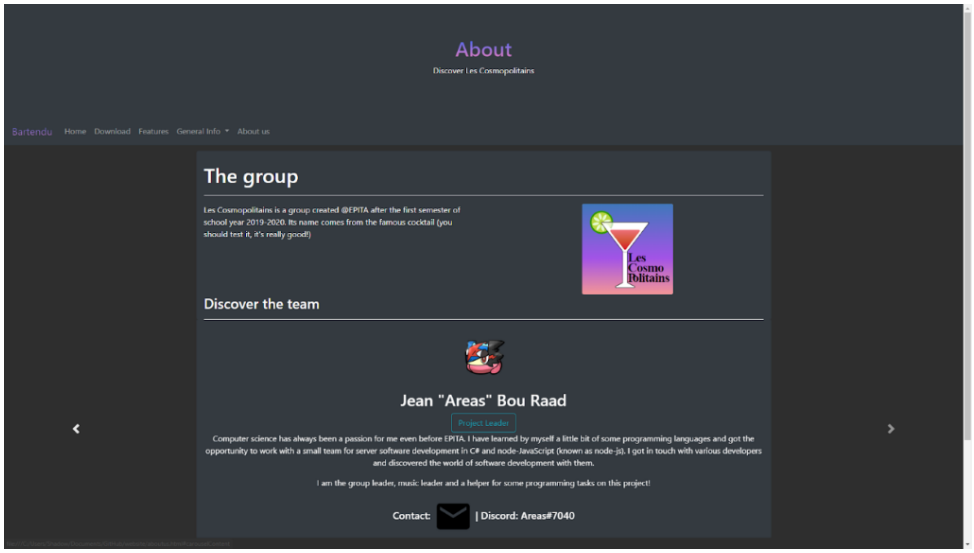

Figure 2: New layout with a dynamic carousel

The second one seems way more satisfying with less text on the page, a carousel division from Bootstrap framework to make one slide per group member. Overall, it is clearer and easier to read. But it raised many difficulties because the carousel division in Bootstrap is made to show images, not text-only content. So, I had to find a way to manipulate the CSS behind the scenes to adapt the existing division to our needs. This was possible thanks to some information found on the internet and Bootstrap's documentation.

We also introduced a features page. It has a simple layout featuring first a small gallery (carousel division of Bootstrap) with some screenshots from the game. It also displays some videos showcasing the game and giving some information on the installation.

Finally, we added an issues section in our "task and schedule" page with some details on them.

2.1.3 Hosting and technical aspects

The website is hosted on GitHub Pages[1]. It offers reliable and easy-to-use service to host a simple website that requires no dynamic changes. You have probably guessed it, but the website must be in a GitHub repository to be hosted and updated. Updates are done by following the commits on the repository's master branch. So, even though I was the only one working on the website, it has been quite a new thing to force myself to commit regularly, using branches, discovering pull requests with the conflict that might come with them. It is not a lot, but no less than 96 commits have been made, 4 additional branches have been created iteratively, 2 formal pull requests have been introduced successfully on our main branch. Finally, you can find on the repository a snapshot of the website for each defense (and even more).

2.1.4 Conclusion

To conclude, the journey to create this website has not been easy, but it fits all the requirements that we had in mind while writing our book of specifications. The group feels like it has charm with its simple but complete design. Thanks to GitHub pages, it will stay online for quite a long time to showcase our game. On a personal note, I have gotten better at creating a website almost from scratch by only using a framework and its documentation during this project.

[1] `https://pages.github.com/`

2.2 Music

Creating a sound environment for a game is a huge and complicated task. It requires some work on various parts of the project. From development to music sheet creation, it needs a guiding principle to keep the intention behind the music written and the output in-game. So, in this part, I will explain the guiding principles I have chosen along with the explanations on how it has been done.

2.2.1 Music Creation

"Music creation" or "composing" seem like pretty and nice words. But they bear with them a huge responsibility for the one creating the music: making it sound good for most people while keeping its freedom of creation. It might be a very philosophical approach to music creation, but it was in my mind during all this semester. Hence, it explains the time spent on this task.

As a reminder, you can find all the sheets, audio samples mentioned in this text on our website:[1]

Main menu theme
For the main menu, we have chosen a plain kind of nostalgic style inspired by the *Life is Strange* main menu soundtrack[2]. Then an acoustic guitar was mandatory. The piano was also chosen. Thanks to my experience as pianists, I was easily able to write sheets for this instrument. With the nostalgic approach, it was clear that tempo should be slow and a minor scale. But an only nostalgic approach would have been too sad for the game that *Bartendu* aims to be, so I brought some light to the theme at the end by going on a major scale and with a more rhythmic approach. As for all games, the track is a loop that never ends until the player does something else.

First level: main and rush themes
For the in-game music, it was obvious that we had to write something dynamic and swinging. To do that, we have chosen only piano instrumentation, which offers great flexibility. The main loop has a fast tempo (we would call it in music Allegro). With that in mind, offbeat was also used to create some tension for players. For the melodic approach, we did not have any major source of inspiration except for a small part inspired by the *Dialga & Palkia battle theme* from *Pokémon Diamond Pearl Platinum*[3].

For the rush loop of this theme, we wanted to reuse some parts from the previous theme while also creating more tension. Hence, the tempo is going a little faster in some parts.

[1] `https://areas0.github.io/website/download.html`
[2] `https://www.youtube.com/watch?v=d9ENy1v3Dyg`
[3] `https://www.youtube.com/watch?v=I_57ptO3TKc`

Second Level: Boat music
A stage on a boat is a special location to cook. Hence, I wanted the music to reflect this special situation.

The melodic line has been chosen to fit with the pirate theme because we also have a story mode level with pirates in it. The inspiration mainly came from a very known piece of music of the film *Pirates of the Caribbean*[1][2]. It is very rhythmic with an ostinato on part for the left hand.

A combination of triplets with regular quavers makes the listener unsettled. The fast tempo combined with these elements makes the music sound good. The result is satisfying. The structure is great because It eases the work for the rush part. Indeed, with such a melodic line, it was easy to rework and integrate it differently in another sheet.

The rush sheet keeps all the elements of the previous theme while exaggerating some aspects. The rhythmic aspect is intensified because the player has no time to breathe during a rush period. There is also more counterpoint integrated in the music sheet at the beginning. But I have decided to introduce some contrast with a slowdown at the middle and at the end with a reference to a very famous theme of *Zelda*[3].

2.2.2 In Unity: Audio Management

Creating a script to handle music correctly was mandatory to make a good-sounding game.

It involved some technical skills and some work on various parts of the project.

Features & Guidelines
The team wanted a sound system that could easily adapt to any kind of situation. That meant being able to make it play music when there is or even turn it off for the lite version while keeping some features of it that are useful for sound effects.

I identified two major parts to make this system:
– Volume Management is the script in charge of handling volume sliders across scenes, getting those values, and handing them over to the Music Management. This one is dependent from the music management to retrieve those values in a new scene.
– Music Management is the script in charge of playing music and make transitions between tracks. This script is dependent on audio management to setup correctly the volume of tracks. It also interacts with some of the most important scripts of the game (e.g. Game Manager).

Two guidelines guided the creation of those scripts: stability adaptability. The first one means being able to run the script in any situation, any scene. Adaptability means that I added many parameters to adapt the script to the music but also to the game itself. These explain some choices made and explained in the second part (implementation).

[1] `https://en.wikipedia.org/wiki/Pirates_of_the_Caribbean_(film_series)`
[2] `https://www.youtube.com/watch?v=27mB8verLK8`
[3] `https://www.youtube.com/watch?v=cGufy1PAeTU`

Implementation
Before going into the details of the how, we must remind you that the sound value of any track in Unity is a float between 0 and 1. Also, all the scripts mentioned are MonoBehaviors scripts[1].

A) Audio Management

With the second defense coming up, Vincent reworked the UI of *Bartendu* to integrate essential features and settings. In those settings, there were three volume sliders:
1) a general volume slider
2) an audio effect volume slider
3) a music volume slider

They follow a classic linear relation: the general volume is what you could call a master volume for the game. All sound cannot go above this limit. The music slider then has a value between 0 and 1 (float) that represents a percentage of the general volume's value. So, behind the scenes, there is another value: the real volume of the track. Once calculated, it must be kept in memory. Therefore, communicated to the music management which stores all those values.

Why is it stored in Music Management? Well, as you can imagine, in a game, sliders are not everywhere but music is. The Volume Management was useless in some parts of the game without any settings. Hence, in opposition to Music Management, Volume Management can be destroyed (because it is a game object) while transitioning from scene to scene. Music Management acts as a memory save of the current state of sliders. Once destroyed, any new Volume Management can retrieve the old values.

This functioning eased the process of the creation of this script. Indeed, if the game object is destroyed and spawns when we need it, we can beforehand attach predefined sliders in serialized fields[2] in the scenes when needed. That would have been more complicated if the script was kept across scenes. Indeed, that would have meant detecting the sliders, guessing which one is the general volume slider, which one is the music volume slider, etc...

Finally, the script works as follows:

1. The player arrives in a scene with sliders. A volume management object is initialized. It gets the current and only Music Management. Then It attaches the values from the attributes sliderValue and generalVolume to their sliders attached in advance to the object.

2. If the player interacts with the slider, via an OnValueChange event[3], it calls a function in charge of updating music management's attributes to reflect those changes. It follows the formula introduced above.

3. If the player leaves the scene, the object is destroyed, but the settings remain intact because Music Management stores these values and is not destroyed.

[1] `https://docs.unity3d.com/ScriptReference/MonoBehaviour.html`
[2] `https://docs.unity3d.com/ScriptReference/SerializeField.html`
[3] `https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.InputField.OnChangeEvent.html`

To conclude, the implementation was pretty smooth because Vincent introduced to me all the technical aspects of OnChangeEvents and Sliders[1].

The script is efficient and provides most of the features that were missing to the game.

B) Music Management

Music Management is Audio Management's heart. Hence, it was the first script implemented after the first defense.

It was implemented feature by feature from the most basic one to some trickier.

Starting with playing an audio file and switching to another track. This was implemented successfully thanks to Unity's Audio Source[2] objects making the process easier. By putting an audio file in our assets folder, I was able to attach it to an Audio Source object and play music. Hence, Music Management works with those objects and manipulates them to play new Audio Sources. This was done by editing the Audio Source attached to the object holding the Music Management. The only difficulty being to transfer some properties such as the boolean telling if the track is muted or not and transferring its volume. There was also an additional case to handle: if we did not want to play a new track. This was solved by attaching a null audiosource to the music management.

Secondly, an essential feature: transitions. For us, good sounding means no brutality while switching between pieces of music. Hence, it required a function to handle the transition. This function was quite complicated to conceive for a beginner on Unity because it was a time-dependent function. It must update volume every few milliseconds and continue running. By looking at Vincent's way of handling the timer in-game, I discovered the coroutines. They do exactly what has been described before. To make it work, it requires partial returns, technically speaking: yield returns. These returns hold a timer for unity that pauses the function via a WaitForSeconds[3] object.

The transition works with a few parameters such as the total time of the transition and the mode (decreasing volume or increasing). The first one follows the first guideline: adaptability. Different pieces of music require different timings to produce good transitions. Hence, this parameter was created. The second one is important because the relation and the iteration are not the same to decrease or increase the volume. Most of the technical aspects being ready, the only remaining difficulty was to determine a good mathematical model for transitions. The formulas are linear. And, as a continuity of Volume Management, the volume cannot go higher than the value defined by the sliders. Hence the following formula:

$$\frac{totalSecs - seconds}{totalSecs} \cdot maxVolume$$

(maxVolume being the value defined by the users via the sliders)

[1] `https://docs.unity3d.com/Packages/com.unity.ugui\spacefactor\@m{}1.0/manual/script-Slider.html`

[2] `https://docs.unity3d.com/ScriptReference/AudioSource.html`

[3] `https://docs.unity3d.com/ScriptReference/WaitForSeconds.html`

Last feature but not least, preparing the lite version of our game by adapting all our functions to make them safe from the missing tracks due to possible NullReferenceException[1]. I introduced a boolean to determine if the game is built in a way that does not include music files (what a shame). It is used in other scripts to avoid useless calls to music management such as the gamemanager which updates the music.

Finally, I used the DontDestroyOnLoad[2] function to assure continuity across scenes of music.

2.2.3 General conclusion

Overall, music was one of the most prominent tasks I had to work on during this semester. I have learned a lot from making music sheets but also by creating the system that would play them. The only regret being maybe a lack of diversity in the music made due to a paucity of instruments on my music rendering software and the quality of this software (they cost a lot of money and hacking is bad :)).

[1] `https://docs.microsoft.com/fr-fr/dotnet/api/system.nullreferenceexception?view=netcore-3.1`

[2] `https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html`

2.3 An additional feature: Discord Rich Presence

For a game that aims to reach some standards of the gaming industry, Discord Rich Presence has been chosen as an additional feature to add to our game.

As you may know with the current context, Discord is a communication application for gamers and more... So, bringing to players the possibility to show that they are playing the game was considered by the team as a nice add-on.

2.3.1 Implementation

The feature was not planned on our book of specifications. However, this was not a problem because it was not a major feature that would require tons of development.

Indeed, Discord Rich Presence is easily integrated in Unity games because it is included in Discord SDK. Discord SDK is a very complete development kit which allows developers to communicate with the discord app for various tasks.

After downloading the development kit, I was already able to integrate it to our gamefiles. With the documentation and the knowledge acquired by making the music management, I was easily able to create a new script: Status Management.

It handles the connection to Discord and the Rich Presence by preserving it across scenes with the DontDestroyOnLoad function of Unity.

Now with the game being able to communicate with Discord, I had to tell Discord what to show. Generally speaking, a game is an activity. It is the same with Discord: we can tell discord that a player is playing our game by sending activities. This data is created and formatted by the game to fit our needs.

However, I had to think about the way to create generic formats for our game.

I have come up with three main activities in our game:
– Being in the main menu
– Being in a lobby
– Playing a game including intro and outro scenes

Each activity has its own function. It allows us to define some presets and add their customizable fields:
– A title: e.g. “In a game”
– A field for details: e.g. “Playing on Level 1” or “Waiting for players 1/4”
– A timestamp: if there are 5 minutes left on the timer of your game it will show 5:00 remaining
– An image: we chose to show only a default one so it will be set to a value by default

```
var newActivity = new Activity
{
    // state and details are two standard fields to describe the activity
    State = "In the Main Menu",
    Details = "",
    Timestamps =
    {
        Start =  curTimeStamp, //epoch value of current time
    },
    Assets =
    {
        LargeImage = "bartendulogoingame", //name of the asset to display, here the logo of our game
        LargeText = "Getting in the lobby!", // text displayed when the mouse is on the image
    },
    Instance = false,
};
```

Figure 3: Standard template of an activity object of Discord's SDK

After creating these functions in a script, most of the job was done. The only task remaining being the implementation of calls to the script in already existing scripts such as GameManagers, or even LobbyManagers. I had to deal again with some PhotonView and their RPCs (remote procedure call) to activate some changes remotely. For example, the master client (the one in charge of computing timers, scores, and orders) was the only client knowing whether the game was lost or won. Hence, it is the only one able to tell Discord in which scene the other clients would end up after the game and update correctly the activity.

The activities are updated thanks to a method Discord.RunCallbacks[1] in the Update function. And to avoid errors, I have created a safe structure of try and catch in case Discord failed to destroy the gameobject and disconnect it from Discord if it is closed while the player is in-game. After that, the implementation was complete and functional:

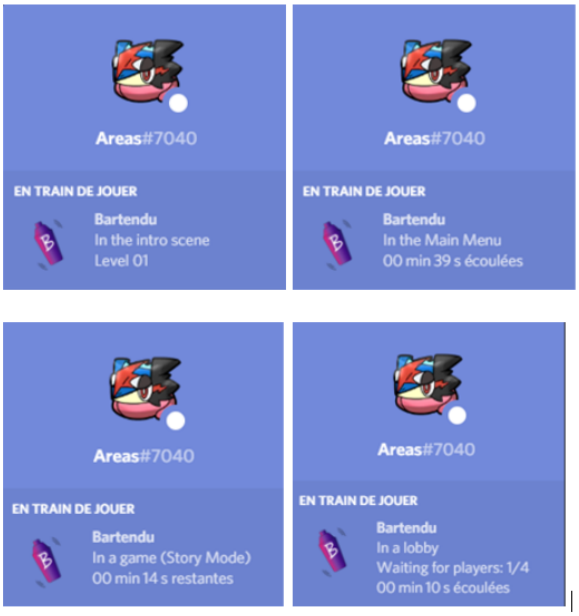

Figure 4: Activities displayed by Discord when playing *Bartendu*

[1] `https://discord.com/developers/docs/game-sdk/discord#runcallbacks`

2.3.2 Issues

Even though it seemed easy, the implementation created a few difficulties.

Firstly, an interesting one: I was not able to setup correctly timers in activities due to my lack of knowledge on the way computers calculate time. Indeed, right now when I am writing those words, computers consider that we are on the timestamp 1589138931 (It was 10/05/2020 21:28). Those are Unix Epoch seconds format to calculate time. It represents the number of seconds elapsed since 01/01/1970. It is used by Discord in their activities to calculate the beginning and the end of an activity. And without any real explanation in the documentation, I had some trouble with that. That said, it is easy to calculate time with this format thanks to System.DataTime[1] class of C# and a simple calculation you have below:

```
// Getting the Unix timestamp value to start a timer (GMT elapsed seconds since 1/01/1970)
var epochStart = new DateTime( year: 1970, month: 1, day: 1, hour: 0, minute: 0, second: 0, DateTimeKind.Utc);
var curTime = (long)(DateTime.UtcNow - epochStart).TotalSeconds; // time elapsed between now and 01/01/1970
```

Figure 5: Code to calculate current timestamp in Epoch format

There was another issue with my implementation of Rich Presence. Unity and the game were crashing if the player did not have Discord opened on their laptop. This bug was created by a single line of code at the initialization of Discord object.

The createflag is a parameter used to identify the type of Discord client (Stable, Beta, Canary) the player has. Hence, it requires discord to be opened and, if not, generated a crash in Unity because Discord was missing. The documentation not stating clearly the implication of using such flags, I had a lot of trouble to find the cause and the fix. However, after much trouble, I changed the parameter to one that does not require Discord at all. And now the script runs flawlessly.

2.3.3 Conclusion

This feature, which was not planned at all, is today working perfectly and fulfills its objectives. Moreover, it helped me learn that some bugs cannot be debugged in a classic way and that documentation may provide the solution if I search properly. The feature looks great on a profile and brings a more professional and polished aspect to the game.

[1] `https://docs.microsoft.com/fr-fr/dotnet/api/system.datetime?view=netcore-3.1`

2.4 Level Design & Level implementation

In this game, stages are a major task because it is the heart of our gameplay. It showcases every piece of work done behind the scenes from the little assets such as the plate to the biggest one: the boat. Also, it is supposed to provide a good gaming experience to the players. That is clear: if level design is bad, a player will stop playing the game. Hence, it requires loads of work to create the concept behind the levels, create sketches, implement them, etc.. . .

2.4.1 Level 01: a simple kitchen with its restaurant

This first level was implemented for the first defense with the first assets available in-game thanks to the work done by Lou and Vincent. However, it went under many reworks to add the remaining assets, rework layout to adjust it to the scale and solve some perspective problems. Below you have the result:

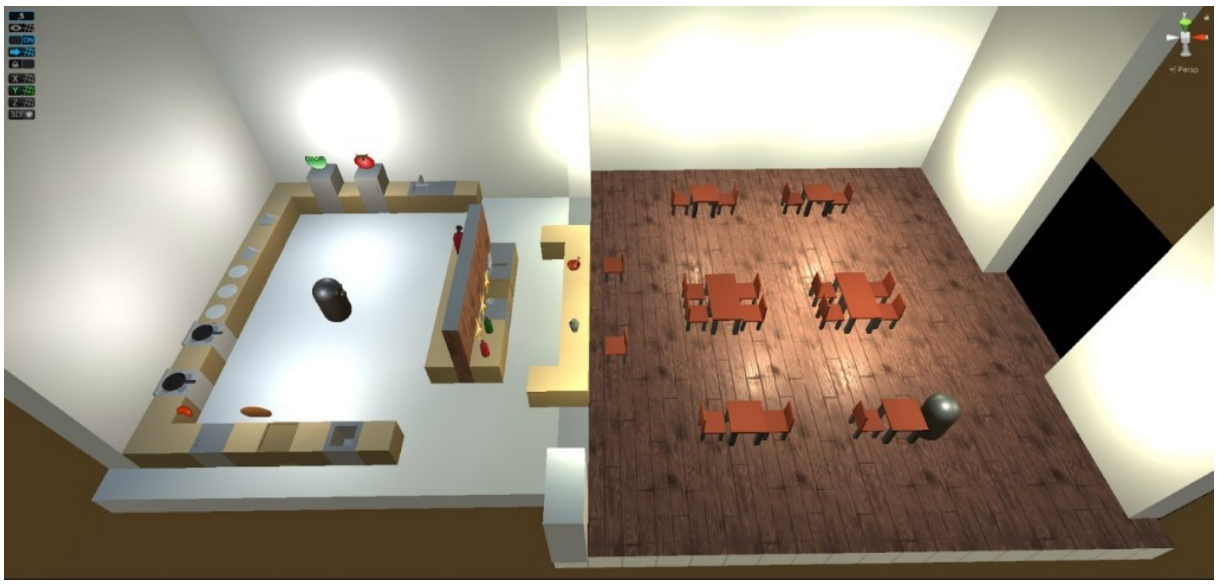

Figure 6: Result for Kitchen Basics stage

Conception:
A cooking game without its basic stage composed of a simple kitchen and its restaurant would be a bad game. Therefore, we decided that It would be the first level created and implemented.

As you can see on the sketches below, the scale was way smaller compared to the result. Each square was supposed to be a $1m^2$ object. This is explained by the fact that at the time of conception, we had no idea of what the game would look like. The same goes for the scale. We also did not have any idea of how many tools we would be needing to make the bar. It was decided late in the project. Basically, for this step, I had to deal with many variables that did not have a value.

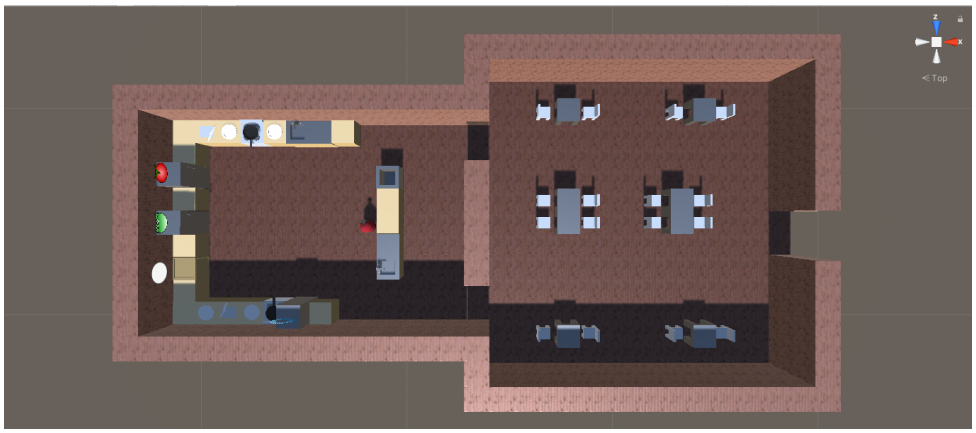

Figure 7: A screenshot of the old kitchen level in Unity scene view

These difficulties aside, the layout is stripped down with only the necessary. In the kitchen-bar, there are only the necessary tools and no space is supposed to be useless. The level is also symmetric because we wanted to ease the movement on this level. To conclude, the player has every tool he needs to get started with the game and discover the gameplay without any additional difficulty from the environment.

Implementation:
The first implementation of this level following the sketches was not good at all because It was too small. Also, due to my lack of experience, the level was not meeting the quality standards wanted by the team.

Hence, the first rework was an opportunity to make the level bigger and add some useful space for players. Thanks to the addition of a new tool: ProGrids[1], I was able to meet the quality standards decided by the team. I also created the solo version of this level. Indeed, all assets are handled differently (without Photon) in this mode. This implied that assets would be a bit different and this could not be solved by a copy-paste of the level to make the offline version. Also, the layout has been slightly modified to adapt to scriptwriting constraints. But the bar items were still missing.

So, the last rework for the last defense was mandatory. Adding the bar with its items. It did not go as planned because I did not anticipate this number of assets for a very limited space in the solo version of this level. So, I had to rework a little this space on the solo version. For the multiplayer version, there was enough space to add all the required items as you have seen on the result above. Also, I reworked the layout a little bit to adapt to the camera angle used. Indeed, the players could not see the cutting progress bar correctly with the angle chosen. Hence, I exchanged the dispensers with the cooking tools.

[1] `https://www.procore3d.com/progrids/`

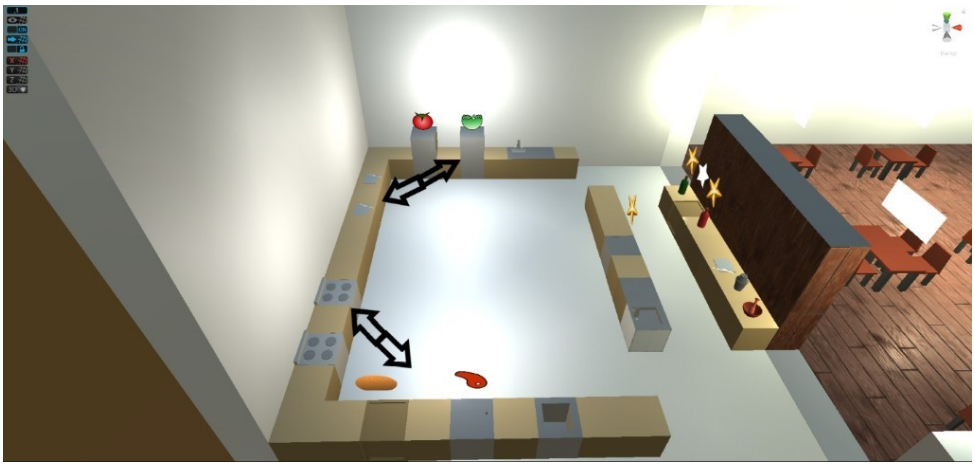

Figure 8: Changes made in the Kitchen Basics stage

2.4.2 The boat

A restaurant on a boat? The idea seems crazy, right? Well, *Bartendu* is a game and we did it. The creation of this level raised many technical and interesting challenges behind the scenes. For example, Lou had to create the biggest model for the game. Vincent and I had to work on how to make the player navigate between different levels on the boat. As usual, it went through the 2 major creations steps.

Conception:
The implementation, except for the scale, is faithful to the sketches. Basically, the stage has 3 levels: the hold, the deck, and the top. The hold holds the kitchen part. The deck acts as the restaurant part and the top as a bar. There is also a pontoon to link the deck to the port. Basically, the players will have to navigate between all those parts which raises a little bit the difficulty compared to the first level.

Implementation:
With the use of a model (the boat), the implementation changed a little bit because I had less freedom on the space I would want to use. I had also many constraints such as the door leading to the hold being static, or even I was forced to leave space for some stairs to go to the top of the boat. Except that, this level was done more easily than the previous ones because everything behind the scenes was ready (all assets). The notable feature of this level being the teleportation of players using the door to go to the hold. There was also additional work in the artistic team to get a sea effect (see Emeline's part) and some work on lighting. We chose a night lighting for this level because we thought that there was some interesting work around lighting at that time of the day. I used an asset pack to modify the skybox[1]. It allowed us to change the overall lighting of the level and make the level look good. This is the final result:

[1] `https://assetstore.unity.com/packages/2d/textures-materials/sky/10-skyboxes-pack-day-night-32236`

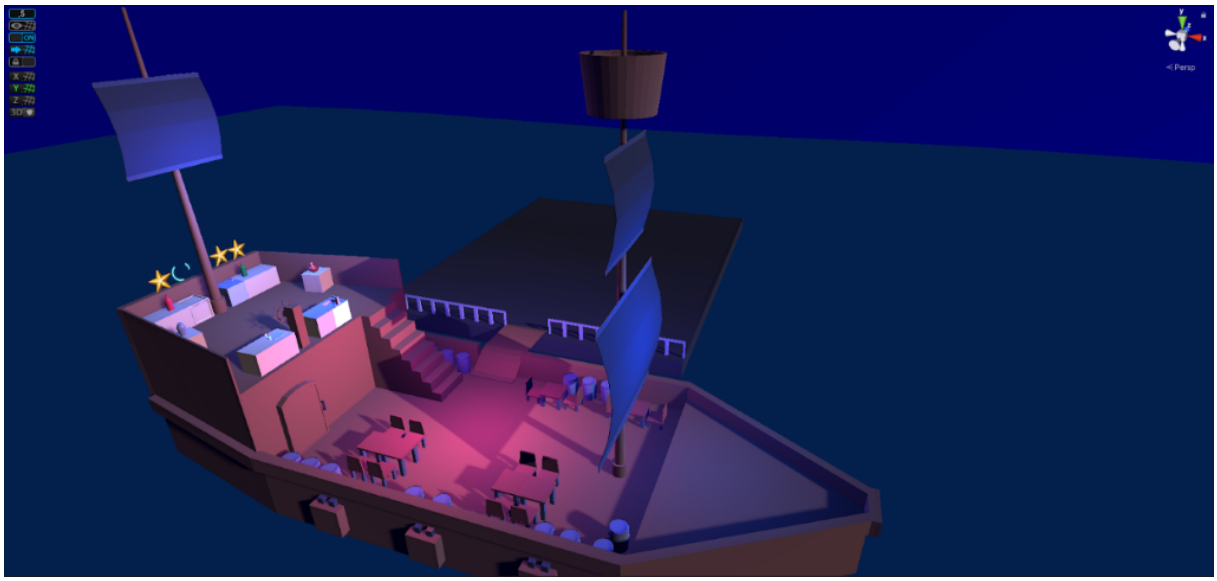

Figure 9: Boat stage, final result (top and bar)

Figure 10: Boat final result: hold's kitchen

2.4.3 The classroom

The classroom is, in my opinion, the funniest level I had to work on during the semester. It runs in an environment everyone knows: a classroom with its teachers. The players hold the role of being the students. They are cooking during a game while the teacher is writing at the board. The students must not get caught cooking by the teacher. Occasionally, he looks at them after writing for a while on the board.

Conception:
This was the first idea created before we even handed in our book of specifications. Hence, the sketches were incomplete or even invalid when I had to implement this level. They were useless. We only kept the concept.

Implementation:
The implementation of this level was fast because It was the last level implemented. As explained before (see Boat), with all assets and tools ready, it was much easier to do my job and finish this level with a few hours of work.

Firstly, I created a few weeks before the real implementation a draft of the level to have an image of the scale of the level and its objects to help Lou creating them. As you can see here, the skeleton was done with it:

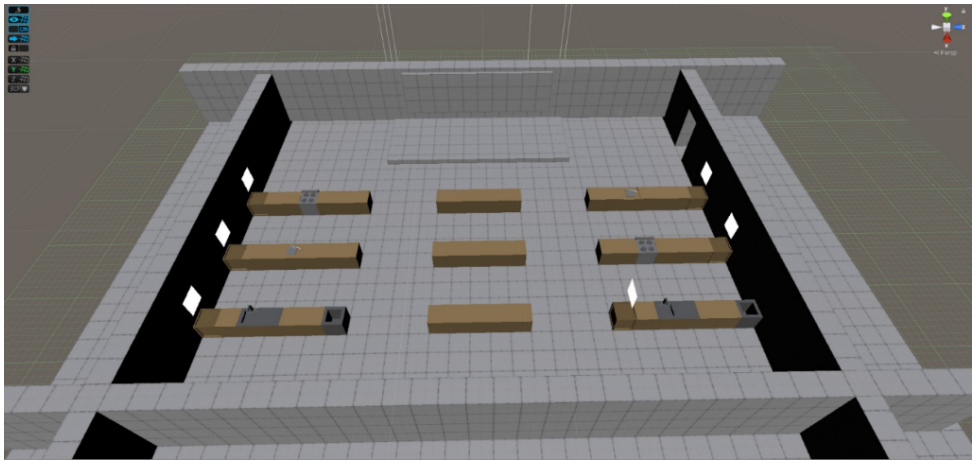

Figure 11: View of classroom in Unity, greybox stage

With that done, we decided together to make some assets to make the level look like a real classroom: class desks, chairs, books (opened and closed), a blackboard, etc...

A few weeks later, I was able to build again the level with the new assets (including bar assets). This is the first result after the first build of the level:

I polished the level a little bit later. It was made possible thanks to Unity Student Pack assets. It included a set of school assets[1]. It was an opportunity to make a classroom even more realistic (and a bit like EPITA's classrooms).

[1] `https://assetstore.unity.com/packages/3d/environments/urban/snaps-prototype-school-154693`

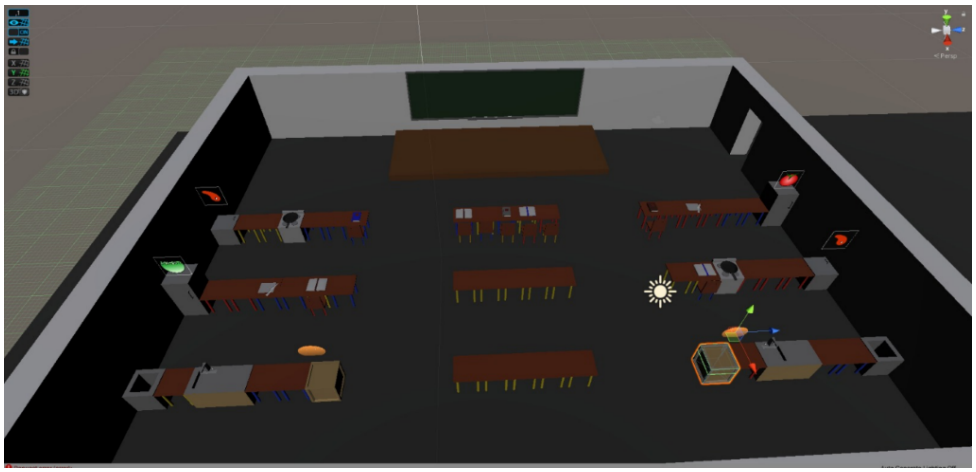

Figure 12: Classroom level after the implementation of the assets made by Lou

As you can see below, the result is good, and the level looks like a real classroom. It is our biggest kitchen and bar layout in the game. Hence, we adopted a dynamic approach during the implementation of cameras. There is not any restaurant part because we considered that It would break immersion. Hence, we adopted another approach with orders having to be delivered at the door.

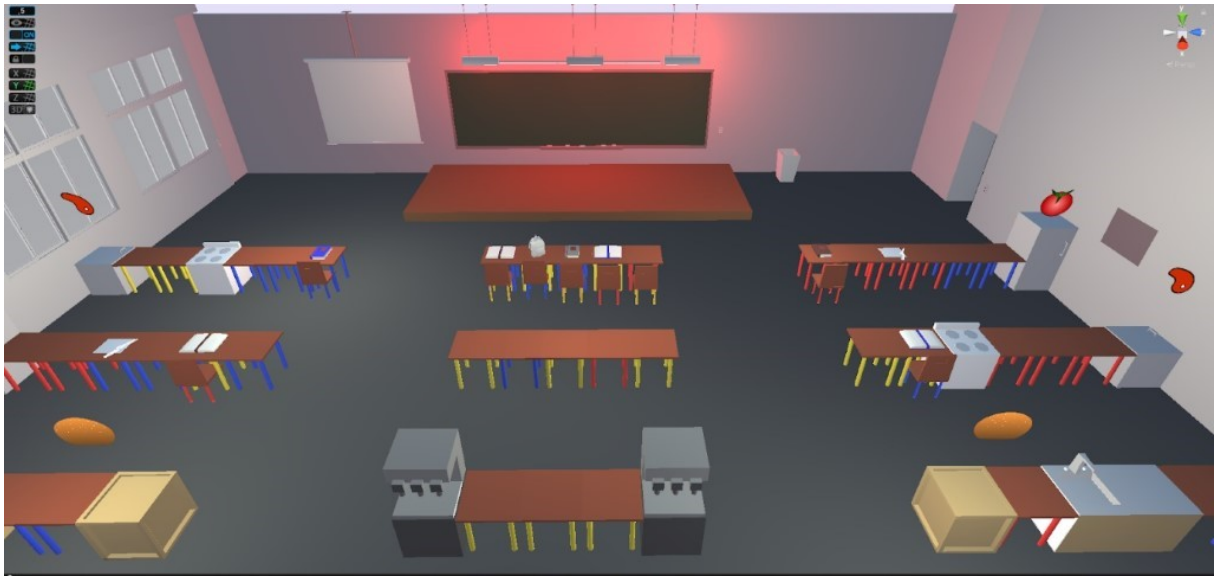

Figure 13: Classroom stage view in Unity (final result)

2.5 Story Writing

In addition to the multiplayer we have implemented a solo mode. It includes three original stories. My task was to write and design those levels. Once done, Vincent was the one in charge of implementation of cutscenes and scripting for this level.

Our game includes three original storylines:

1. A food-critic comes in the restaurant to evaluate its quality. He is the only client and if there is any delay you will lose many points.
2. At the grand opening of the boat restaurant, a pirate comes to defy your team. If you lose against the pirate's team you will lose your boat.
3. A student needs to earn some money to repay his debt. Hence, he must cook in the classroom and deliver orders to Duber Eats.

For each of these stories I had to create a script then export it in an excel file (csv format), and to create the graph around this script. This system was created by Vincent (note to his part).

I had a lot of fun creating these stories and their gameplay concepts. However, I and the team had many more ideas we were not able to implement due to a lack of time even though we satisfied the requirements made in the book of specifications.

2.6 Experience in the group

During this project, I was also the group leader. Hence, I was the one in charge of making sure everything worked out well between the different people in the group. I also had to make sure everyone was dedicated to the project. Obviously, I had some tasks around organization. It meant organizing meetings to see how things progressed, creating deadlines, keeping track of everything done, etc... Because doing such a project during a semester is not easy, it was mandatory to keep a strong organization of tasks in the group.

For example, during some weeks the team had too much (regular/school) work to do. So, I had to reorganize our schedule to make sure we would be meeting the deadlines for the presentation. Or even some tasks being late due to unforeseen complexity.

Overall, I can say it was an amazing experience. Having to deal with some problems, but also with group members who are fully dedicated to their tasks made my job even more interesting. Moreover, the quarantine made things even more interesting because everything had to be done online. Discord (`https://discord.com/`), Unity Teams, GitHub, were great tools to help us with that.

3 Vincent's Achievements

3.1 Introduction

We have been working on Bartendu for 6 months now and I have enjoyed it so far. It is a record for me to have worked on a project for this long. I always drop my projects about a month after starting them, so this was a new experience to me. At some point I just did not feel like working on this project, but I had no choice. My teammates relied on me and I relied on them. This was also totally new because I had always worked alone. So, sometimes they would ask me to implement something and they could not continue what they were doing because they needed me to implement it. It was sometimes frustrating because I was already working on something else. But overall, this experience was very rewarding. I learned a lot, I was sometimes pushed to my limit (thanks to a lot of bugs) but I am, and I think we all are, proud of presenting this game.

My tasks for this project concerned all the technical part and the implementation in Unity. When we distributed the different tasks, we tried to give to everyone a field of work that the person enjoyed or had a bit of experience in. I had a lot of experience in Unity and I loved programming so we decided that I would choose those tasks. I ended up also doing some sprites for the game like the logos or even the different 2D graphics you can see while ordering or the graphics representing the food. I had no prior experience, but someone had to do it, so I took it upon myself to provide those graphics. Can I say I learned how to do 2D art? Probably not... I am not an artist and my art skills have not evolved nor improved and that can be seen in the game... But I still am very proud of the work that I have made.

Each phase of the project had a different focus. I will review my work theme by theme instead of chronologically because it will ease comprehension as I did not do everything at once but rather a little bit each phase.

The first important theme I tackled is the multiplayer. We were advised to start with this, and I am grateful for that advice. It was (not by far) the hardest part of the project for me. I had no prior experience with the multiplayer and was new to these concepts of server synchronization, server objects, clients, and master client. I learned RPCs which can send information to the server and I also learned about custom serialization which allows you to send custom data through the server. My lack of understanding led to a handful of bugs which was devastating for me at that time, but I do believe it made me stronger and taught me not to give up as soon as a difficulty arose.

The second theme was the singleplayer. We had from the beginning mentioned the singleplayer. For us it was important to implement this as not everyone can play online and as it would give us the possibility to script some levels which would have been difficult online due to synchronization and different internet connections. This part was not easy per say but it was the part of the project I most enjoyed working on. When I say easy, I say that because we already had the base finished. All that was left to do was to change the multiplayer to the singleplayer. Again, thanks to the advice we got, it was way easier to go from a multiplayer to a singleplayer rather than the other way around. During this part I also implemented scriptable levels, a cinematic system and a dialogue one.

The final part was all about refining the details and the AI. I originally planned on implementing it during the second phase of our project. But I had underestimated the amount of work to go from a multiplayer to a singleplayer. Refining details included bug fixes (and they were a lot of them), adding more customization for the player and making the UI more user-friendly. Now the AI was not that easy to implement. It was probably the second most difficult task and it took a lot of sleep away from me... But I will tell you all about it later in this report. Let us go back to the multiplayer implementation.

3.2 Base of the game and multiplayer

This was our first big theme. First, we had to create a very quick multiplayer. This version of the multiplayer would allow us to join a lobby and then connect to a room. This was maybe the easiest thing to implement. With a lot of research and my programing experience this was implemented in a heartbeat. It was functional but not even close to the game we have today.

What lacked was the gameplay. This is where we implement the base of the game. It includes the player controller and core functionalities like cooking, cutting, merging and serving.

3.2.1 Base of the game: Player controller

Again, my experience really helped me here. With a lot of games done – or partially at least – within my 3 years of programming with Unity, I learned that physics-based movement were to favor over simply translating the player on the coordinate axis. This is because collisions will stop any force. Meaning even if we boost in front of a wall the collider will stop the player. Whereas if you boost in front of a wall with a none physics-based movement, there is a chance that the player will just teleport through the wall if the player tries to move past the wall. Again, and I will never stress this enough, with my experience in programing we were able to have a moving character in a day due to the countless character controllers I had to implement in past projects.

But even if moving is important it is not the only thing a character controller must do. We must implement the different actions the player can do. There were 4 actions we wanted to implement. One is to take or drop an item, the second action is to use an item like the cutting board, the shaker or the extinguisher, the third action is to throw an object (this is still very experimental, I tested it thoroughly but I am not sure if this action should stay in the game) and lastly I implemented the boost action. It allows you to boost to give you additional speed in the forward direction but only occasionally every now and then.

When it came to bugs, I saw none at that time. But it turned out that the pick and drop action was the main source of the problem. That is why even if the script should be fairly simple (only movements and actions in this script) it ended up being over 1000 lines long and about 2900 words (More stats will come in the conclusion about the amount of lines of code in this game).

3.2.2 Multiplayer: Spawning objects on the server

I think this was the first problem I encountered. The ability to spawn objects on the server. Spawning objects on a single computer was not very hard. Spawning it on 2 to 4 computers is a different topic. I was very new to the concept of server synchronization. So, every client would spawn this object and we would end up with 4 objects at the same place causing all sorts of strange bugs. Plus, testing at that time was very difficult because I had to build the game every time (at that time the project was not very big, but it still took about a minute to build the game just to play 5 seconds and realize that the game is just not working).

I realized I needed to spawn the object on the server instead of on each client. I also discovered after some research that we had to use special components to synchronize the position and the physics. Another problem showed up. While instantiating an object on the computer return the object spawned, it is not the case for the Photon Network Instantiate. This was terrible news because it would complicate everything even more. I had to create yet another script to handle the instantiation. If a viewID was given during the instantiation, the object would search this id and get the parent object. With this object, the newly instantiated object would set its parent to be the object found and we could then get the instantiated object by getting the first child.

Spawning objects on the server remained an important theme throughout the entire development of this game but thanks to the "ObjectHandler" script I was able to correctly spawn objects without worrying about if they would synchronize with other players.

3.2.3 Base of the game: Core mechanics

Now that we could spawn objects, it was time to add the very base of the game. The very first building block of our game was the SnapToSurface script. Indeed, we wanted a grid-based game meaning the objects would be able to move freely anywhere but as soon as they encountered a surface (like a working surface, cutting board and more) they would snap to a grid resetting their rotation and physics. When I say that this script is the building block of our game, I mean it. Every surface has this script, I quoted some before, but the list goes on and on. One particular type of object in this list I want to talk about is the movable surface. It is a surface, but the player can also pick this surface. This type of object was useful regarding plates, glasses and pans. They all are surfaces meaning objects can snap in it. But they also are movable as the player and pick them and put them on any surface.

Movable surfaces are essential to the game: you need to be able to serve a dish on a plate, you need to be able to cook a steak and then remove it from the stove and you also need to be able to serve a drink in a glass. The huge problem was to move the 2 objects at the same time, and this caused all sorts of weird bugs... The player would start to fly, a giant tomato would turn into a huge disk, players would clip through the floor, objects would not follow the intended holder... All these bugs took so much time and effort to fix. Up until the last defense some of the countless bugs were still coming from the movable surfaces. This was a really low point for me and at that time I lost all desire to work on the project. And working on plates for a month without knowing the source of the problem is very frustrating...

Thankfully, I slowly started to work again on the project due to the deadline coming up and it turned out just fine in the end. The problem was collisions and physics messing with the object's scale and doing all sorts of weird stuff. It was an unfortunate problem of the 3D asset of the plate. Lou had just gotten the hang of Blender and together we were able to fix the bug.

Another script I added that would complement the SnapToSurface is the script UseSurface. The name is self-explanatory, it allows you to use the surface by reducing the health of objects held by the SnapToSurface. This script is used for the cutting board, sink, pestle and shaker. I also added a trash can which uses the SnapToSurface script and then destroy any object held by it. Of course, I then had to add a lot more functionalities like not being able to destroy important objects like the extinguisher or a plate. For that I used a tag that allows me to check if the item cannot be trashed. One last small mechanic I added was the fire and the spreading of the fire. Fire occurs when you leave something on the stove for too long. You then must take the extinguisher fast to take out the fire before it can spread to nearby surfaces. The spreading is random but deadly as you will not be able to use the surfaces on fire.

The last big feature I had to implement was the room part. A huge inspiration for our game was Overcooked. But we wanted to get away from the game and create our own. So, we decided to add the room part. Here the goal is to serve the food to the clients as well as seat them and clean the table. The first thing to implement was a client generator capable of generating clients with orders for the player to complete. This was a whole new kind of problem to solve because this time it was not a technical one but more like a level designing problem. I worked together with Jean to implement this feature. He designed an order manager with diagrams and I tried to implement it in the game. At the end of the first deadline we had a somehow working (but not really) order manager and room management. But I decided to do it completely again from scratch because I was not happy with how it turned out. The whole thing was unstable and was working strangely in multiplayer. But first let me give you the basics of client management.

The clients spawn and then use a spot to wait for one of the players to seat them. To move around they use a NavMeshAgent, Unity's AI movement controller. For that to work we had to bake a map giving the different surfaces the AI can walk on (This method of moving non-player objects will later be used for the AI). Each client of the group shares the same properties (they are synchronized - on the server for multiplayer - at the beginning of the game and then each time a player interacts with a client). The shared properties allow the players to interact with any client of the group, so they do not have to find the client with all the attributes. One of those attributes is the state of the client. The different states include Waiting, FollowingPlayer, SeatAtTable, ReadyToOrder, WaitingForMeal, Eating and Leaving.

When they are waiting, the player can interact with one of them to switch them all to "following player". Then if he interacts with one of them again, they will look in a 2 wide radius to see if there is an empty and cleaned table nearby. If there is, they will seat randomly, if not they will go back to "Waiting" and just stay in place. Seating clients at the table was another fun thing to do. We had the plan to have one table of 2 people and one table of 4 people. But the goal here was to serve a dish for each table of two. Meaning that the table of 4 would be split into 2 tables of 2 each requesting a different (or not) dish. When I say this was fun it is because I implemented a surely useless feature but somewhat fun that seat the client randomly. For example, if the client is alone and you decide to seat him at a table for 4 people, then he will not seat at the same place if you start the experiment again. The placement is random, and this allows the restaurant to feel more realistic.

Then the player does not have to interact with the clients anymore. The Table script alongside the PassSurface script will handle the rest. First, when a client group is spawned, they randomly choose a number within the bounds given by the GameManager script to know how long they will take to order and how long they will take to eat. This is also a feature implemented for realism and breaking the monotony of the game. So as soon as the clients are seated, they will start a countdown. When this count down reaches 0, the group will order a dish. The menu where they can choose a dish from is also on the GameManager script. Once you receive the order you have a certain amount of time, depending on the difficulty of the preparation of the dish and time to make the dish, to serve it to the clients. If the timer runs out, you lose points and the timer resets. If you serve the wrong food, you also lose point but if you serve the right food before the timer runs out you get points and the clients start eating. As for ordering they will take a certain amount of time to eat.

Once they are done eating, they will leave the restaurant letting a dirty plate (or glass) on the table. It will then be your job to clean the table before you can seat another client group at that table. Doing the dishes in *Bartendu* is a bit particular. You clean dirty plates in the sink, but you trash glasses to get a new one. The idea behind is that the glasses are made of faux glass and you can trash it (of course it gets recycled because *Bartendu* is an eco-friendly game) and get a new one clean. This is basically how the room management works.

I previously said I would talk about why I chose to re write the whole order management from scratch. The problem was that the clients would not synchronize their properties causing bugs when interacting with another client from the one you interacted at the beginning. Doing it all over again was the best option rather than fixing and building over an already unstable script.

Finally, I implemented an endgame screen. It would become very different in the singleplayer version of the game, but the functionalities remain the same. The end panel will compare your score to the thresholds given again by the GameManager. If you exceed at least the first threshold then the level is considered complete. You also have stars to collect that represent the different points thresholds. The star system as well as the highscore mechanics are built to make the players try again and again to beat their highscore and collect all stars and try to do a perfect run of the level.

3.3 Singleplayer

Almost everything said previously was implemented before the first defense. Except for all the reworks and the cocktails mechanics.

The second focus of *Bartendu* was the singleplayer. And this took an unexpected amount of time. I first thought that it would be done in a week, but it turned out to take the entire 6 weeks. Indeed, all the assets implemented previously all had multiplayer components. I had to entirely redo the entirety of the assets to adapt them in the singleplayer. And this was not fun. It was not complicated but time consuming. So, let us first talk about the assets in the game.

3.3.1 Base of the game: Food properties and Food recipes

One important thing we wanted was the ability to combine the food in any way possible or imaginable. This led to the creation of huge amount of properties and recipes to make sur that every way of combining the food was possible. For example, to make a BreadSteakLettuce you can either add Bread to a SteakLettuce or Steak to a BreadLettuce or even Lettuce to a BreadSteak. But this had to be done for every single property and for the 4 dishes that a client could order: a salad, a hamburger (5 possible), a cosmopolitan or a mojito. The salad was easy because it has only 2 ingredients, but all of the other orders have 4 ingredients, meaning each of them have 15 different properties and 25 recipes. Now you can imagine the amount of time to make those in the first place and you can then imagine having to do them a second time.

The different attributes of the food property are: the object to display (one for the singleplayer and one for the multiplayer), the ingredients that the food is composed of (to be able to show the composition of the object to the player), the recipes you can make with this food, the health of this object (to be able to use it in a surface [-1 for invincible; -2 for burned], the targeted layer surface (if the food requires a special surface to be transformed in (like the ice or the cut lemon needed to be transformed in the pestle), the transformed property (when the object's life has reached 0), the object to spawn if the food is given to a client (either a dirty plate or a dirty glass [3 types of glass]) and the cocktail object (only for cocktails to put the cocktail in an invisible plate so you can serve the drink to a client)

That being said, even if it took a humongous amount of time, it was definitely worth it as now the game is fluid and you do not have to think of the order the ingredients need to go in.

3.3.2 Singleplayer: rewriting scripts and recreating prefabs

A lot was done on the server especially sending information. Since we did not have a server anymore, we had to find a way to pass this information. In fact, this was super easy (way easier than going through a server). But even if it was easy, I still had to go through every single script and change function by function. Another tedious task was to entirely recreate all prefabs. Not only did I have to recreate every property and recipe I also had to recreate prefabs to remove the server components (Photon View, Photon Transform and Photon Rigidbody). This took so much time and yet a lot more had to be done before the second defense.

3.3.3 Singleplayer: Cinematics

What do you remember in a game? The graphics? Lou had us covered with beautiful assets. The animations, particle effects and sound effect? Emeline carried that out. The Levels? Jean created 3 magnificent levels. How about all together with beautiful camera work? This is where cinematics come into place. The first step of creating a beautiful cinematic with dialogues is to be able to display the dialogues in different languages. Here English and French.

Cinematics: The localization system
I already explained in detail what the localization system is in the previous report, so I will not go into so much detail but here is a summary:
The localization system is a script responsible for handling the different languages in your game. Here I decided to use Excel as an input for my localization system so that Jean, who is implementing the dialogues and their translations, can have an easy and fast way to use and implement the dialogues and their translations into the game. Indeed, its interface is very easy to use and comprehend. On the leftmost column we would put the identifier key, the one unity can understand and look for when trying to display a text. On the second column would be the English version and on the third the French version.

Now having all this written in a block is not very readable. We had to find a way to determine which cell has a text we need to integrate to the localization tool and which text is just a header or commentary. For that we decided to use a special character, the tilted double quote.

Combine with REGEX expressions (Regular Expression) we can extract the text within those special characters and give it to the localization tool.

As soon as the entire Excel document is formatted, we can then proceed to add it to the project. One problem, Unity does not accept ".xls" files; instead, it does accept ".cvs" files which is perfect because Excel allows us to transform a sheet into a CVS file. In short, a CVS file is a file where every cell is delimited by a comma. (CVS stands for Comma-separated values)

Now that we have a CVS file, we must be able to read it and then create the tool. We get the texts given by the file and we put them in the according dictionary: either the English one or the French one. The text value goes in with its according key. For example: ([greetings_intro_1, "Hello World"] for the English dictionary and [greetings_intro_1, "Bonjour le monde"] for the French one). Then we just have to call the localization tool and ask for the text with the key "greetings_intro_1" and we get "Hello World" if the language is set to English and "Bonjour le monde" if the language is set to French.

Now that we have a working localization tool it is now time to work on the dialogue system.

Cinematics: The dialogue system
The dialogue system is a simple script that asks for the localized text from a list of keys and displays it. When the player presses the space bar, the system changes the current key to the next one and so on and so forth. But we wanted to add more character to our dialogues... More interaction... We thus decided to implement a possibility to influence the dialogue we get. This is what we called "Dialogues options": You get multiple options and according to which one you chose you are greeted with a different dialogue.

Before the second defense we would have handled the dialogue system with scriptable objects. But it was very inefficient, not visual at all and time consuming. I thus recreated this system using Unity's Graph View. This allowed Jean to create Dialogues that look like a horizontal tree. Very visual and practical, each node represents a dialogue; you can then insert in a list of string with the name of the speaker and then the dialogue key and the dialogue system will correctly display the dialogue. When a choice can be made you can add more output ports to the node to then connect this node to different dialogue nodes. This allows the player to choose a different dialogue, here representing the different branches of the tree.

Finally, Jean asked me to add a way to trigger functions using a key. I thus created a special code (if the key starts with then it will be a trigger key). The key does not go to the localization tool as it has no translation but rather investigate a dictionary on the Dialogue system to see if there is a corresponding key and a function associated. If there is, it just runs the function and continue his way into the tree.

Cinematics: Moving cameras
Cameras are a key point for successful cinematics, they are what makes a cinematic look great. Smooth movement, right camera angles, everything is made to deliver the best shot of the game. When I first tried implementing moving cameras, I decided to create my own script. The result was fine but not that great. The big problem was that my movement were a bit abrupt and sharp which was the opposite of what we wanted to achieve. I was using key points creating a path for the camera and the camera had to try to match the rotation of each point it encountered on the path. However, we wanted something more professional and better looking. With some research on the internet I found an amazing unity package: Cinemachine. This package allows us to have very smooth rotation but also to change camera during the cinematic to get the best shots. This was perfect. It took a bit of time to get used to this new powerful tool but in the end, I think the results look great.

Cinematics: Cinematic manager
All we need to do now is wrap everything up together. Which means launching the dialogue at the right time, having the cameras match the client's animation, stopping the dialogue during animations (when the barman prepare the drink for the client, the dialogue must stop so that the animation plays correctly) and at the end of the game loading the right scene (Victory or Defeat). We have not mention yet but each cinematic take place in a different scene. For example, because the camera in the defeat and victory scene does not move, an entire part of the level has been removed because it would never be seen. This can greatly improve performances and was easier for us to work in these levels. We decided to label the different scenes as follows: Level X + (KEYWORD) [X represents the number of the level and the KEYWORD can either be Intro, Victory, Defeat or blank for the actual playable scene]. With that we can just load the right scene with its build name.

3.3.4 Singleplayer: Scripting the gameplay

Another decision we took was to script the gameplay for the singleplayer so that we could tell a story. As I went into details for the first level, I will quickly explain it and go on with the other levels.

Scripting the gameplay: Level 1
The concept for the level 1 was to give an overview of the game to the player. Meaning we removed the bar part (no drinks must be done) as well as the room part (you do not have to handle client seating). Instead you have one client, Mr. Insatiable. The catch behind is that he never stops ordering food. As soon as you give him the correct dish, he will order another one. The only way you can make him stop eating is if your level timer runs out. In that case your points will be evaluated according to the thresholds and you will know if you won or not. But in order to make the level last a little bit longer, each time you correctly complete an order, some time will be added in you level timer. Mr. Insatiable can order from an exhaustive list of 5 burgers (the hamburger, the burger steak, the burger steak lettuce, the burger steak tomato and the burger lettuce tomato).

This level was interesting when talking about technical specifications because it required to rework the order manager and to add a few methods. While not being complicated nor time consuming it was still a very amusing change of pace after coding all the previous more complicated scripts. But overall, I did not change much, and the core gameplay remains the same.

Scripting the gameplay: Level 2
On to the second level. This one was not very difficult to implement because it required no scripting. The goal of this level is to engage on a battle against a pirate to see who will control the sea. Here we added core gameplay features such as the bar and clients. They can order any food implemented in the game and you must get as many points as possible within the time limit. When the timer runs out you get the end scene.

Scripting the gameplay: Level 3
This time scripting had to be done, here is the idea behind this level. You are a student behind on the bills, so you have to cook even in class (a classroom that strangely resembles EPITA's) and make sure you do not get caught by the teacher (a teacher that strangely resembles someone I know). Here the scripting is fairly simple, we just have to randomly and every now and then make the teacher look back and check if the players are moving. If they are, deduce points and give an audio feedback to let them know they were caught. I also implemented, after a suggestion by Jean, a red light to know when the teacher is going to move to make it a bit easier. But because it would then be too easy, sometimes the teacher will give a false alarm and will not completely look back.

Scripting the gameplay: Level 1 and 2
Overall, this was very fun to add but I thought that the first 2 levels were missing some fun gameplay mechanics... So, here is what I added. For the level 1 I added mice that will spawn and steal unsupervised food. You must take the food away before they do because after that it will be too late. And for the second level, as we are on a boat some surfaces will move due to the ocean. This caused a lot of bugs but greatly improved the gameplay overall.

3.3.5 Saving data

This was the final part of the singleplayer: being able to save your progression. I will not go into too much details as it is a pretty simple part, but we created a script to handle the save and load of the star count and best score for each level. I honestly had very little experience in this field, so I mainly took inspiration from tutorials online and forums. A few lines of code later and the system was correctly working. It basically uses serialization to convert data into text and then saves that file in the game's file. I am pretty sure that with enough knowledge or a bit of research anyone could understand the text file and change the values, but we decided to turn a blind eye and avoid any attempt at encoding our data for time reasons.

3.4 AI and final touches

As said previously the AI should have been implemented for the second defense. Even though it was not implemented, some research was made, and I had a rough idea on how to implement it. Needless to say, I struggled a lot and that probably was the second lowest point for me. Stress was building up; we were very close to the deadline we gave ourselves and this AI almost got the best of me. Here is my journey.

3.4.1 AI: Basic actions

The AI is a player. It just does not use the player input but anything the player can do, the AI can too. That meant implementing the same actions as the player but this time for the AI. Fairly easy you would say. Well it was. The actions were implemented in the PlayerController script, so I just copied that out and pasted it into the AI script. I then added the movement for the AI to navigate through the level. I used coroutines and NavMeshAgent to move the AI because coroutines allowed me to wait for the AI to arrive at destination.

Here are the actions I implemented: Pick (to pick an object from the ground or a surface), Drop (to drop an object on a surface [cannot drop an object on the floor]), Use surface (keep using until the object inside the surface has changed), Cook (wait for the object inside the surface to cook and then remove the pan from the stove).

All these actions allowed the AI to do anything the player does. And it was a breeze to implement. I was fairly happy and thought to myself "Now this is easy!". I was such a fool.

3.4.2 AI: Brain

Now that we have actions, we need to put them together to make the AI do coherent things. Not just random actions. This is where I implement the AI brain.

Brain: Get the correct recipe
It is at that point that I realize this was not easy and that we shot ourselves in the foot when we decided to create about 70 recipes. How will the AI know which recipe to make? I would be tempting to take the first one and just go with it. But I am a man of challenge (it probably caused my doom). I wanted to make an AI that you would be happy to play with. An AI intelligent not just following one recipe but instead analyzing its environment and choosing the correct recipe. What a mistake.

First step is to decide what is a food that the AI can interact with. This was done so that the AI could not interact with a food the player touched (allowing the player to play without the AI just taking the food he is making because it would not be much fun). So, I gave the ability for the AI to pick items only if they are in spawners. When it does that, if the food does not have an AI tag on it, it will add one. An AI tag is a script that does nothing, but the AI can look for object with this tag and know that it can interact with it.

Once you know what objects your environment is composed of, you can start looking at the recipes. The AI looks for the first order and tries to make it. It first gets all the possible recipes to make a dish from scratch. Then the AI evaluates which recipe has the most potential (if you already have the ingredient in your environment the recipe has potential because the food is already ready to use). Once the AI has the perfect recipe it will start cooking. And every time it finishes (or not for example if the player took the food the AI wanted to pick or if the player uses the surface the AI wanted to use) it will reevaluate the best recipe possible and continue.

One thing that my AI struggled with is knowing how to get basic food. Because sliced lettuce or cooked steak do not have a recipe, the AI does not know how to make that. I first implemented the action to pick from a dispenser (like a fridge, crate or a cold spawner) then I added the action of getting a tomato or lettuce then cutting it and then the action of getting a steak and then cooking it.

Finally, I added the action of plating, serving, cleaning the table and doing the dishes. I had trouble knowing exactly when to clean the table and do the dishes so the AI will just wait for the clients to eat the food then it will clean the tables and do the dishes. I already had a hard time implementing this AI I did not want to add the room management, so the AI is not capable of handling clients (seating them).

Brain: Bugs, bugs and bugs
Probably the biggest source of bugs for our project. It took one test run to completely take down the AI. The problem is that when I implemented the actions, it did not take into consideration the fact that there would be players around and that players would also use the same environment as the AI. The main source of bugs was coming from the fact that players would steal the food that the AI was working on, or use the surface the AI wanted to use... All these bugs had to be fixed in order to present a game as bug free as possible and it took my sanity with it. At the end of the day this AI is far from perfect, but it is still somewhat working so this is a huge pride for me.

3.4.3 Final touches: Customization and bug fixes

Final touches were mainly focused around bug fixes and the rework of certain features like the order management or the dialogue management. We implemented the player name over the player so that players could tell each other apart. We also added colors to the names as well as a skin selection as promised. We finally added the level selection for multiplayer and reworked the interfaces to be more user-friendly. The language was generalized on the entire game not just the level and I created an installer for the game.

3.5 UI work

I really had no experience creating sprites, but I had fun doing that. I created a logo for the company and a logo for the game. Then I started to create the sprites for the game that are used for the orders and the display to know what the item is made of. Overall, more than 30 sprites made it into the game, they are not beautiful, but I love them very much.

3.6 Small tasks

I had a lot of experience with Unity, so I took the lead when it came to implementing 3D assets. I also implemented the 2D assets I created and designed the entirety of the user-interface for the game. From the main menu (with its animation), lobby to the in-game UI, I tried to make it as small as possible, as well as comprehensible and beautiful. I also added some camera effects for all scenes to try to make the game even more beautiful. Some of those effects include colored vignette, chromatic aberration, shadows enhancing, color correction and color temperature and overall color balance, contrast, exposure.

Finally, I helped some of my teammates to get used to Unity, but I only had to do this once and they got the gist of it very quickly!

3.7 Conclusion

I really loved working on this project. Even if there were ups and downs, I am really proud of what we have made, proud of my teammates and a bit proud of myself. I did not think that it would be such an adventure to work on this project, but I was lucky to be working with such a great team. We got along pretty well, and we worked hard for this project. I do not really know if I will ever work on this project again, but I heard that at least one of us is, and I think it is pretty cool.

I learned a lot from that game. Technically, I got to create a multiplayer, an AI and tackle a lot of fields I never did before like the user-interface and sprites creation. Mentally I learnt a lot too. I felt really bad at some points in the making of this game but I never gave up (not that I could have because I would have) but it also taught me how to work in a group (which I had never done while working on a game) and also helped me deal with my stress issues.

When I look back at this project, I feel proud. I caught myself playing instead of working because I loved what my teammates did. The levels are beautiful, the objects are impeccable and all the effects (animations, particles and music) make the player immerse themself so well in the game. We have been working on this project for so long and even though I am happy to turn the page I will still miss this project as I put so much into it. I worked very hard and I always wondered how much I wrote.

So, here are some fun stats:

- I created 74 scripts
- The longest is PlayerController with 1000+ lines
- The shortest is AiTag with 0 lines
- All scripts combined make up:
  – 8.000+ lines
  – 20.800+ word
  – 175.000+ characters without space

4 Lou's Achievements

4.1 3D modeling : Introduction

3D modeling was a very important part of this project, we wanted our game to look "cartoonish" and so we decided to use Blender in order to create our own assets. Plus, we thought that it would be funnier to have our own assets, that we would have built for our game. This was challenging because none of us had any experience with this software. So, I was charged with the task to learn how to use Blender and to create all our assets in order to create a game that would look exactly how we would imagine it.

4.2 Blender

So, as I mentioned before, I used Blender in order to create the assets of the game. First, I had to learn how to use this software. We did not need any complicated shapes for our assets, but the first ones were very hard to make. The closest thing I had to a 3D modeling experience was the use of the software SolidWork in high school and I remember having trouble using it. Blender was different, it seemed easier to use but was not, there were so many options on how I could modify an object or create an object. It took time to learn how to extrude, add objects and shape them the way I wanted to but also on how to join different objects.

Finally, with the help of many tutorials, YouTube videos and Google pages, I managed to create assets, but I had a few difficulties along the project, to make objects, to export them in Unity, etc...

Those difficulties were the following:

- One of the first difficulty I encountered was learning the different shortcuts and how to use the different tools in Blender
- Then I had to figure out how to assemble different elements I was creating, at first I used a modifier called Boolean but then I discovered that I could just right click on the items and join them.
- I also had to figure out how to add colors on different faces, we decided to use color instead of texture (except for the glasses) so our assets looked more vintage and "cartoonish".
- One difficulty that I encountered was the rotation of the asset and its location that were different in Blender and Unity. I managed to fix the problem by applying the rotation and the location on the object. I had the same issue with the scale and size and fixed it using the same method.
- I struggled with exporting the assets with the correct scale,... then I found out that I had to transform each asset into an FBX file and check the box Apply all Transforms otherwise even if the transforms were applied in Blender, the result in Unity wasn't the same.
- Then I found it hard to find the correct mesh and form to obtain what we needed, but with practice it became easier.

- Another issue I had was that some faces were invisible on Unity but not on Blender so I fixed the problem by creating a new modifiers at the object called solidify in order to add thickness to the faces that were "invisible" and became transparent in Unity.
- I then discovered that the colors in Blender and Unity seemed different so while I was adding the color I also added a light (a sun) in Blender in order to pick the right colors then I would remove the light to export the assets (so that we would not have too much light in our scene).
- I struggled with adding bones and weight to an asset so that we could animate it later on but I managed to do it after a few tries.

4.3 First Assets

The first assets to be created were the one that would be used in all the levels and the ones we needed the most.

The following assets were made:
— A tomato and a slice of tomato to represent what the tomato becomes after being cut
— A lettuce and a slice of lettuce to represent what the lettuce becomes after being cut
— An extinguisher that would be used in case some meat is left to cook for too long
— Steaks: burned, cooked and uncooked to represent the different states they could be in
— Some glasses to put soft drinks in
— A pan
— Bread
— Plates: clean and dirty (once the customers are done eating
— Hamburgers with all the different possibilities (bread, tomato and steak, bread, lettuce and steak. . .)
— A salad
— A Ragdoll to represent the customers

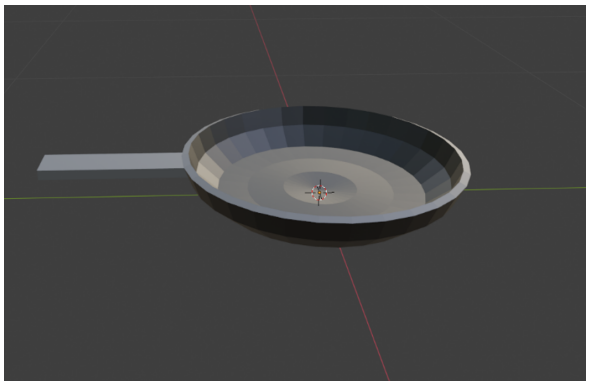

Figure 14: A pan

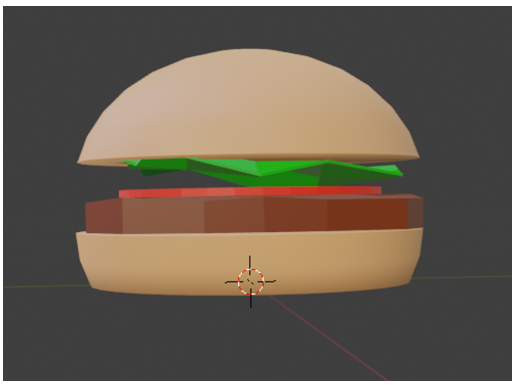

Figure 15: A hamburger

Those assets were hard to make because I learned to use Blender while creating them, but it turned out okay and their looks matched our vision of what the game would look like.

4.4 Ragdoll to represent the customers

One of the most difficult asset to realize during the first part of the project was the Ragdoll that would represent the costumers. I spent a lot of time doing it and it was beneficial because I learned more by creating this asset than in creating the first ones.

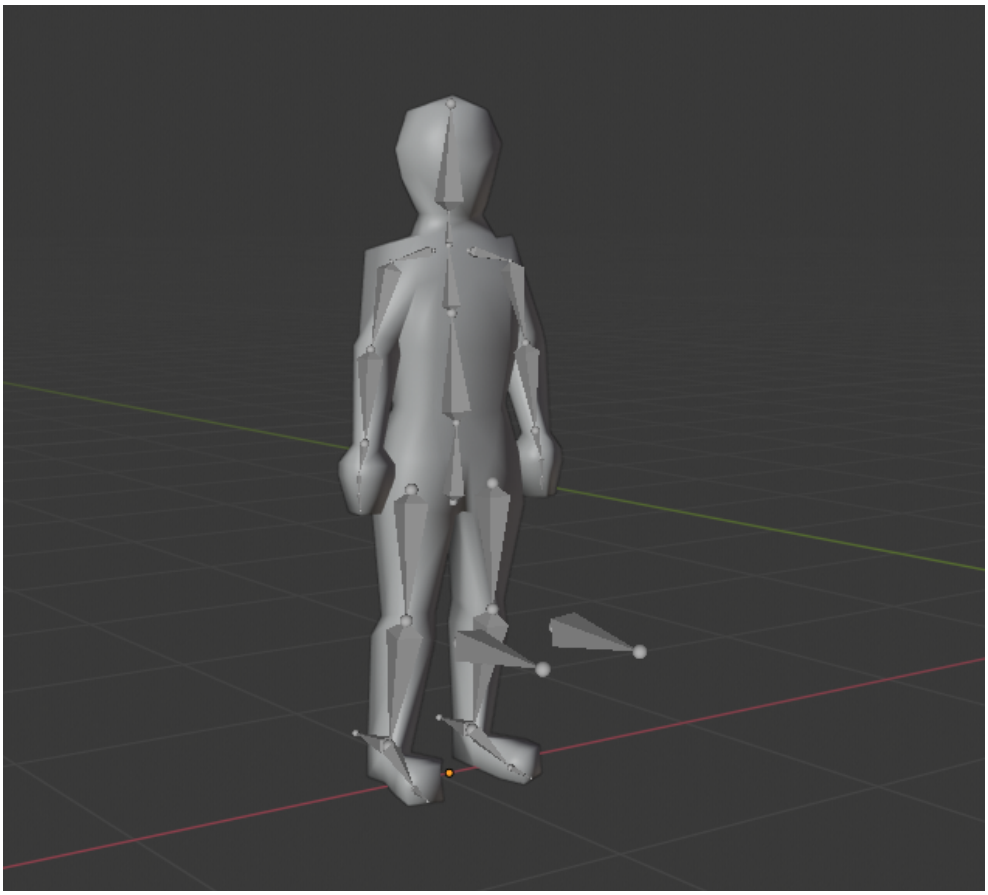

Figure 16: Ragdoll

Once I finished building the model, I then had to create an armature (with bones) and add weight so that Emeline could animate it. The difficult part was to add weight because I was not going to do the animation and I was not sure which movements the Ragdoll would need to be able to do. After finishing the Ragdoll we decided that Emeline would make the bones and add the weight on the next assets we would have to animate as she knew which movements they would have to do, hence it would be less time consuming. Moreover, adding the bones and weight felt like a part of the animation process.

4.5 Other Assets

For the second defense new assets were added to the game, those new assets were meant to be used in the bar part of the game, among them: two cocktails, Mojitos and Cosmopolitans, as well as soft drinks, tools and ingredients to create those cocktails.

Here is the list of those assets:
— Mojito's glasses empty and full
— Cosmopolitan's glasses empty and full
— A shaker
— A Machine to create some soft drinks (coke...)
— Mint
— Lime
— Ice cubes
— Cranberry juice
— Bottle of alcohol
— A mortar and pestle (to prepare some ingredients)

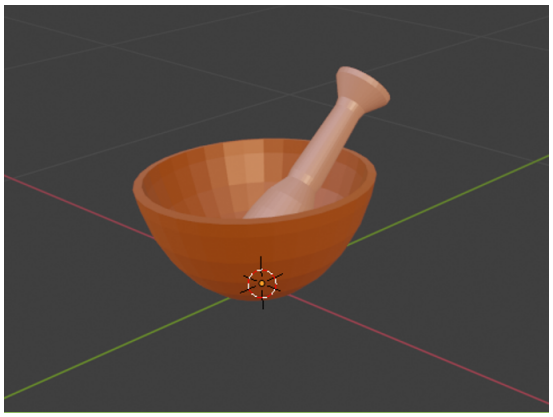

Figure 17: Pestle and Mortar

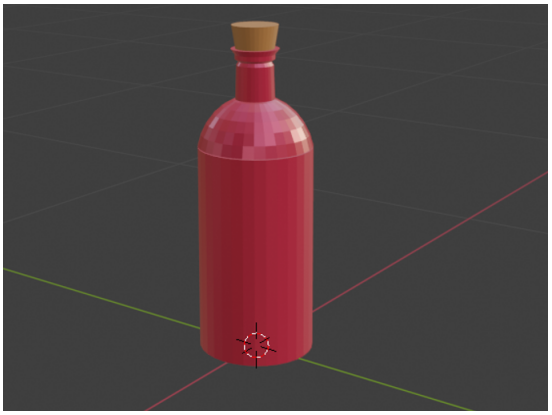

Figure 18: Cranberry juice

The texture of the glass and ice have been added in Unity as it was easier to do so and looked better.

Plus some other assets were replaced from the ones previously made in Unity:
— A Chair
— A Table
— ...

4.6 Assets specially made for some levels

For the need of some levels we decided to create new assets that would be used only in those particular levels in order to make our game better looking and funnier to play.

So for the level where a food critic comes into the player's restaurant, a new Ragdoll Mr.Insatiable has been made. This ragdoll is bigger and has different colors than the ragdoll used to represent the customers in order to be recognizable.

Then for the level that takes place on a pirate's boat, assets have been created to respect this pirate themed level such as:
— Barrels (to decorate the ship)
— A bridge so that the customers could board on the boat
— A boat
— Another character representing a pirate, this character is not a ragdoll, it was better looking and easier to realize it using the same design as the characters the players can impersonate.

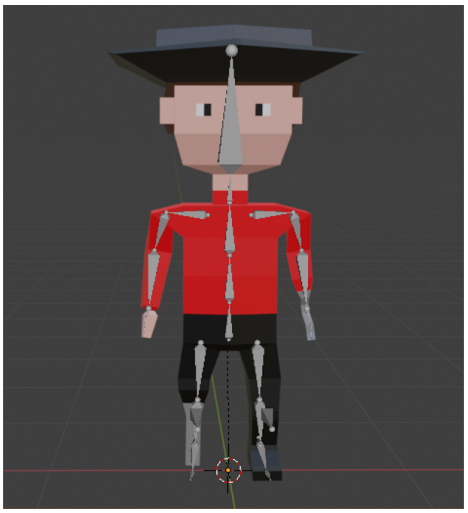

Figure 19: Pirate

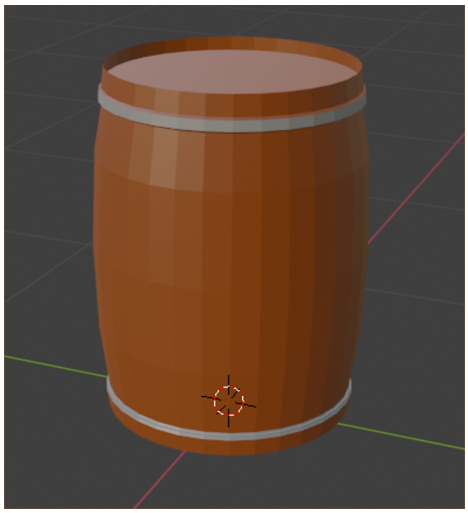

Figure 20: Barrel

Other assets have been made for other levels, such as the level in a classroom:
— New tables with different colors (red, blue, yellow) like the ones we can find in EPITA
— New chairs with different colors (red, blue, yellow) like the ones we can find in EPITA
— A black board with chalks
— Open Books
— Closed Books
— A new ragdoll to represent the Teacher of the class (this ragdoll like the one named Mr. Insatiable is different (colors, size, glasses,...)

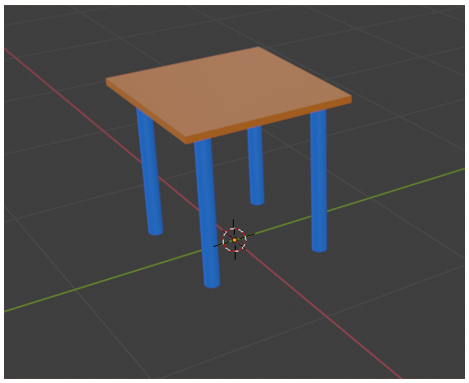

Figure 21: Classroom chair

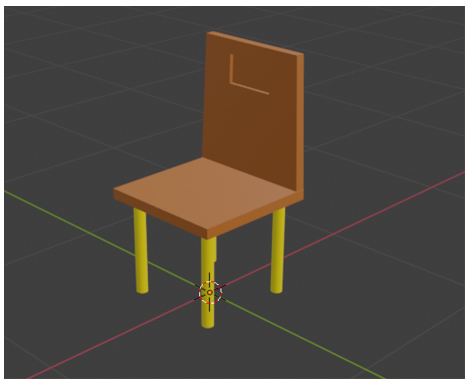

Figure 22: Classroom table

4.7 The boat

The boat was the biggest asset I had to make for the game, and it took a long time to create. It is composed of a deck, a top deck with a ladder to access this part of the boat, and a door to access the boat hold (that Jean designed on Unity).

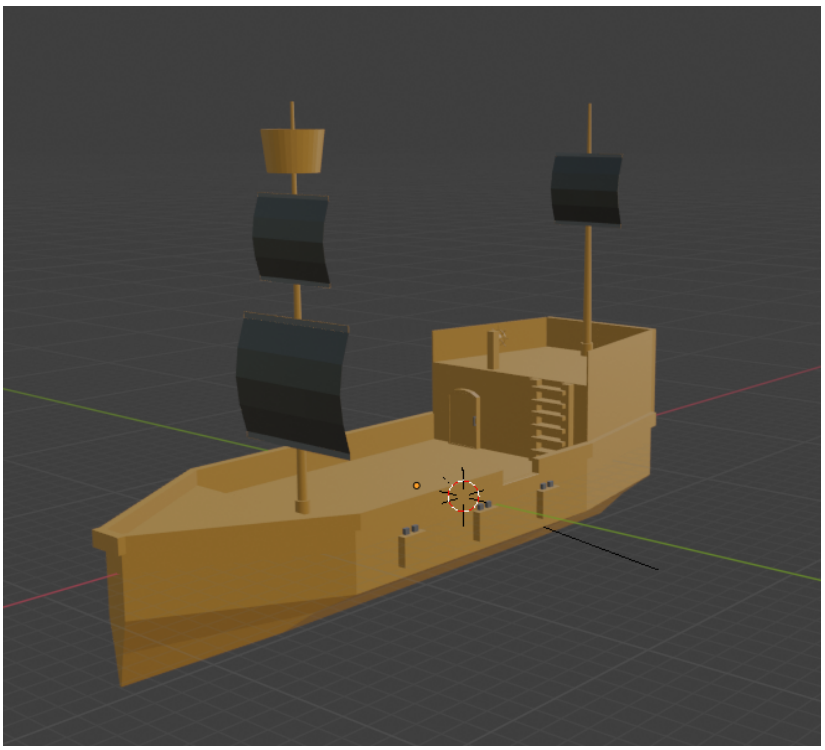

Figure 23: The boat, named "Le Loup de Mer"

The goal was to create this asset big enough so that Jean could build the restaurant on the deck with the table for the clients on it and the bar on the top deck. The kitchen would then be in the boat hold. Then the boat would be put at sea near the coast.

One of the biggest challenge to make this asset was making sure that it would be big enough to fit the restaurant's rooms but that it was on a scale similar to the one of the players and customers so that it would not look excessive. Then all the details like the canon, the windows were not hard to make, it just took some time. I am really happy with the result as it was definitely one of the biggest challenges of 3D modeling for this project.

4.8 Scan and players' characters

Since the design of our game is "cartoonish", we had this idea that the characters used by the players could be Funko Pop's. So we had this idea to scan in 3D those figurines and polish them in Blender, add bones and weight, animate them and then implement them in Unity. Then each one of us would choose a figurine that we would scan. That way we could have original characters for the players and we would learn a new technique.

To do so I first tried to take more than 30 photos of a figurine and then use this software called Mushroom to create a 3D model. After a few tries with different numbers of photos, different lights, and backgrounds, it still did not work, I had nothing and the process stopped before even creating a part of the model. I could not figure out why it was not working. Maybe we did not have a camera good enough to have the quality needed by this software.

So instead, after a lot of research, I found different smartphone applications that would supposedly allow us to create a 3D model and with which we should not have any camera issues. I tried two applications.

The first one was called display. land, and the goal was to film the object while moving around it. After a lot of different tries, it was still not working. This time the process was but the final product was not what we wanted, the model was flat and did not look like the figurine, I still managed to model a stool but it seemed like the model we wanted to model was too small Because when I tried to model something bigger it worked (the final product was not perfect but we had something that we could have used).

The second application was called SCANN3D, and you had to take at least 20 photos and the application would guide you with point of colors to take them. I tried a lot of different ways to model the figurine, different lights (daylight, led, lamp,...), different backgrounds (white, normal background, in a black box,...), different way to take the pictures,... But it was still not working, the model created was looking weird and it had modeled only the front of the figurine, there was no back and the shape was deformed.

Figure 24: 3D scan made by SCANN3D

Finally, we decided not to use 3D Scanning to create the players, we did not have the time nor the equipment to create something that was good looking and working with the environment of the game and the other applications I found were rendering something worse.

So I decided to create a new character in Low poly using Blender, the same way I created the ragdoll but with a "cartoonish" look, the players would have the possibility to choose between different characters with different looks (different colors and hairstyles).

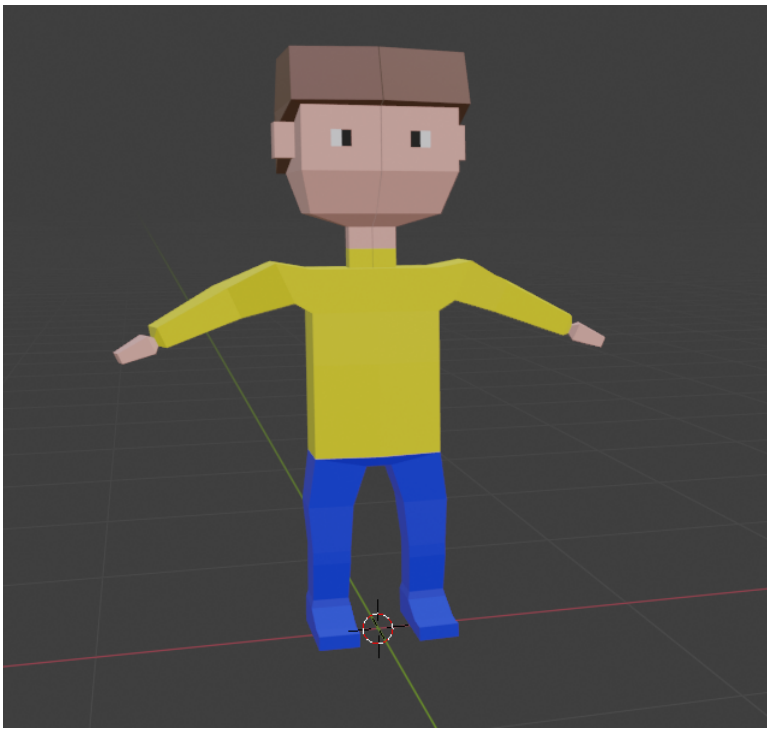

Figure 25: Player character

4.9 Last assets and improvements

Then it was time to create a few more assets that we needed for our game to be complete.

Those assets were:
— Mice that steal unsupervised food and ingredients
— More players with different styles
— Dirty Mojito's glasses
— Dirty Cosmopolitan's glasses
— Dirty Soft drinks' glass
— A ladder (on the boat and as an independent asset)
— ...
Some adjustments have also been made, for instance some scales have been modified and size of some assets as well as some colors in order for our game to be better looking and for our assets to better fit into the levels.

4.10 Conclusion on 3D modeling

To conclude, all the assets we needed for our game and we wanted have been made on time, the specifications have been respected and the graphical/art part of the project is now complete. Learning how to realize 3D modeling and Blender was really difficult but I am really proud of the result and of the look of our game.

4.11 My experience in this project

This experience was definitely challenging but also very interesting. I was able to learn new skills and improve those that I already had. I learned how to do 3D modeling and how to work with a team on a project that big and long.

Being a part of "Les Cosmopolitains" was amazing, I had a lot of fun and the all team was dedicated to this project.

We were well organized, and everyone respected the schedule whether it was for deadlines or meetings.

Finally, I am really happy about what we did and hope that I will be able to work again with Jean, Vincent and Emeline.

5 Emeline's Achievements

5.1 Particle effects

Particle effects were quite tricky for me at first, as I had no experience on Unity. It took me a lot of time to get used to using this tool. It was only after the second presentation that I had achieved to make convincing-looking particle effects. Here is a comparison of the fire effect between the first presentation and its current state in the game:

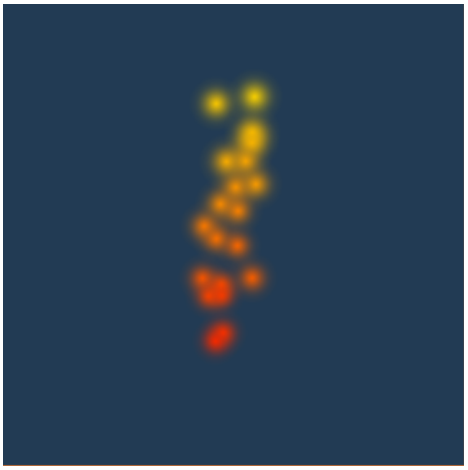

Figure 26: First presentation

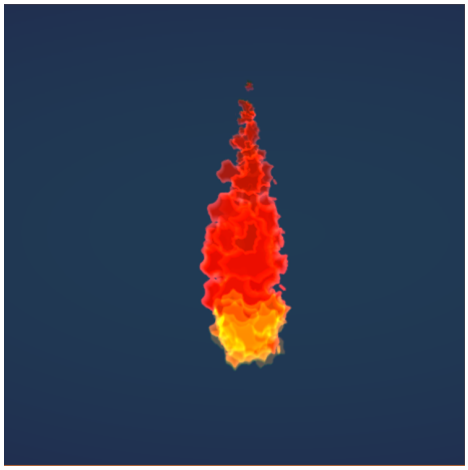

Figure 27: Last presentation

There was definitely a huge improvement in the way I made particle effects while I started to fully understand all the parameters and options Unity had to offer. The one thing that specifically helped improve the quality of the particles was to play around with their textures and the materials. I would draw textures in Gimp and apply them to Particle Systems, and repeat the same process over and over again until I was satisfied with the results.

I did make other particle effects for the second defense, but no matter what I tried I could not seem to manage to implement them in the actual project. These were the foam effect and the boost effect.

The only other particle effect that made it into the project was the fire extinguisher foam effect, which remained the same since the first presentation.

Figure 28: Fire extinguisher foam effect

5.2 Sea effect

It really is not much but it was still interesting to make: the sea effect for the Boat map. We needed it to be opaque so that it would hide the hold of the ship. We also needed it to be a plane so that it would not bother the rest of the gameplay, while also making wave-like movements. Some very convenient tutorials found online helped make this sea, which is a shader that was made with a script.

5.3 3D animations

I had to animate many different characters for the game; to name them, the customers, the player characters, Mr.Insatiable and the Teacher.

Getting used to all the different shortcuts in Blender was difficult, but in the end it worked out fine. Some other difficulties caused me a lot of trouble though, as I had a hard time finding a way around them.

5.3.1 Customers

The customer animations were done for the previous defense. As a reminder, these animations were:
— Idle, for when the customers wait in line outside
— Walking, for when the customers enter the room and go to their seats
— Sitting down, when the customers reach their table
— Ordering, which happens when an order is sent on the board for orders
— Eating, which customers use once they are served
After making these customers animations, I got used to making animations and figuring out how to portray all the movements that would be shown in the game. I did not need to record myself doing these different movements anymore.

5.3.2 Mr.Insatiable

Mr.Insatiable is a special customer; therefore, he has only 3 animations. The first one is called "Sat", it's a looped animation of a single keyframe where Mr.Insatiable is only sat at his table and not doing anything. Then he has the classic "Ordering" and "Eating" animations that other customers have.

5.3.3 Player Characters

When Lou gave me the model she had made for the player characters, an unexpected situation arose. What we had decided on was that Lou would take one of ragdolls I had already animated, and change the shape of the ragdoll to make it look like a player; that way, the armature would keep its animations and I would be able to reuse the animations I had already made for the customers. However, when I would try to play the animations on the player character, it would look very deformed and the movements looked awkward. I tried to change the weights on the character, modify the rig a bit, but nothing would work. So I ended up deleting the rig from the player character and making a brand new one, and made the weights around it. I then copied the "Walking" animation from the customers onto the player. It still looked strange, so I changed the animation on the player a bit to fix it.

After that I made a series of animations for the players:
— Idle, the player is not moving
— Walking
— Running, for when the player is boosted
— Grabbing, the player takes an object from a surface
— Putting down, the player puts an object on a surface or drops it onto the floor
— Cutting, the player cuts an ingredient
— Shaking, the player shakes the shaker
— Washing, the player washes the dishes

A few things are to be noted there:

1) The cutting, shaking, and washing animations were each seperated into 3 parts. First, the player starts interacting with the object corresponding to the action; second, the player perfoms the action, so this bit could be looped; finally, the player stops doing the action.

2) For the idle, walking and running animations, a second version of them was made so they are adapted to whether the player is holding an object or not.

5.3.4 The Teacher

The Teacher has his own animations since he's a very specific character that only appears in one level. The basic animation for this character makes him look like he is writing on the classroom's black board. Then two different animations can be triggered. The Teacher is able to turn around during class to check if the students are behaving or not. He is also able to feign turning around to scare the players; what I did was take the beginning and the end of the "Turning Around" animation, but instead of letting him turn all the way around in the middle, I made him stop and only shake his head a bit while he looks at the board. This was just a way of making the level a bit more fun.

5.3.5 Implementation

I then had to implement the animations in Unity. This is when a major issue arose, which I did not expect. In older versions of Unity, you only had to export the FBX file of your animated object from Blender, and once you dragged and dropped it into Unity, you would have access to all the animations. But in my case, I would only have access to the one animation that was playing in the Blender scene from which I had extracted the FXB. I searched far and wide for a solution, and could not seem to find one. Until one day I realized that I could mix together two things I had read about online. So what I did was put all the keyframes of all the different animations end to end in the same animation clip in Blender. I deleted all the other animations in this file, and exported the FBX again. I then had one very long clip containing all my animations. All I had to do was cut this clip under Unity according to the keyframes at which each animation started and ended.

Once I had access to the animations, I could set the relations and the conditions under which each animation would be triggered in the Animator Controller. It looked like that for the player animations:

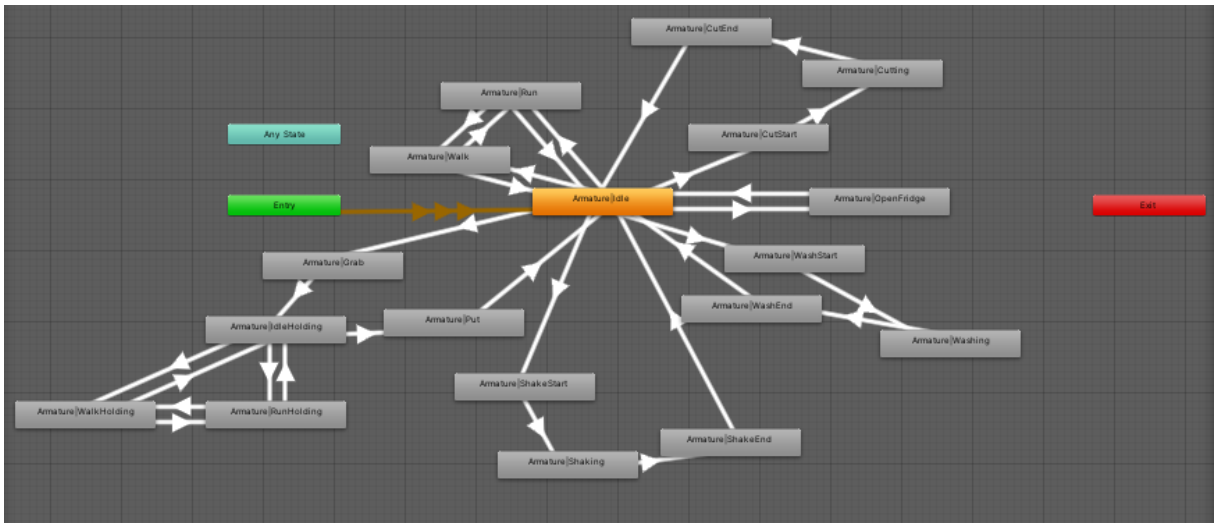

Figure 29: Player Animator Controller

After that, I had to look through the scripts of the game to find where to change the values of my conditions so that it would trigger the desired animation. It was complicated at first, but thanks to the help of Vincent and Jean I got to understand the code enough to implement the animations.

5.4 Tutorial and sound effects

Tutorial and sound effects, but mostly their lack thereof. I did not complete these tasks. I did have some sounds recorded on my computer, but never ended up putting them in the project. So they are basically non-existant in the game.

5.5 Personal experience

I learnt to use some software I had never used before and managed to get some understanding of Unity. This experience was definitely enriching and it highlighted some personal issues I have, mostly concerning organization. I am very thankful for my group mates and can only feel apologetic towards my group mates, as I did not complete my tasks properly, which I was assigned at the beginning of the semester.

Conclusion

This project allowed us to learn to use many different tools and software that we may not have used otherwise. It taught us to be more thorough and meticulous in our programming while we kept trying to stabilize *Bartendu*. A lot of research was done on all ends, as most of us had little to no experience in making video games. We improved our ability to work in group, which will be very advantageous in our future projects, at EPITA but also in our future careers.

Even if some things were not done or polished, we are still proud of the result. Some of us might even continue working on the project during the holidays.

This is the end of the journey for Les Cosmopolitains.

Thank you for playing.