ODD ONE OUT - humans not allowed

REFLECTIVE PRACTICE

06/06/2015

Author: Hien Quy Tran
Student Number: 543800

Semester: 4th Semester

Lecturers: Prof. Thomas Bremer
Prof. Susanne Brandhorst
Oliver Langkowski
Walther Sänger

Lecture: Project B. 3 Weeks Experimental Game Jam I
Project: Development of a full body game with Microsoft's Kinect V2.0 support

Content

1. Introduction to the Project

Unlike previous projects, which have taken place throughout the whole semester, this project is an experimental game jam with the duration of three weeks. Not only the time has been cut down by a significant amount, but also the size of the group has been halved from around six to three members per group. Having said this; the scope of this project is strongly limited compared to previous projects and our time schedule had to be planned more carefully than ever.

The topic for our group project is: “Full Body Game”.

1.1 The Development Team

Hien Quy Tran
Game Concept
Programming
Sound Design

Bettina Ladányi
Art Concept
Character Design

Anna Gofman
Environment Design
UI Design

1.2 Concept
Looking at existing Kinect games, we were not able to find one single game, which would meet our expectations in terms of: “making good use of body tracking”.

My mission was therefore, to work out a game concept, which would not support body tracking as an optional input device, but furthermore, requires body tracking as something truly irreplaceable. I also did not want to create yet another Kinect game, in which the core mechanic is just about mimicking specific gestures in order to trigger yet another specific event. Meaning; it maybe could be also triggered by any other input system like for example a gamepad.

Thinking about this led to the conclusion, that the main advantage of the Kinect is not to trigger events by doing specific gestures, but the capability to track “minimal” body movement and translate them directly onto an object in game and alter its state directly, which again could directly influence the outcome of the game.

Our game, “Odd One Out - No Humans Allowed”, is a four player game, in which the players have to disguise themselves as non-player characters (NPC). The motion of each player is being tracked by the program and projected onto one of the ten characters on screen. The remaining six characters are computer controlled NPCs. The goal is to identify the avatars of the other players and kick them out first.

fig.1: in-game screenshot of the dancing avatars, 4 of them are mimicking the players

1.3 Game Mechanics

As soon the players are lining up in front of the Kinect sensor, the game starts automatically. Each player gets a random character which is mimicking her/his body motion. The players are now trying to identify themselves and the other players. Once an opponent player is identified, her/his character can be selected by moving the own cursor over its head and is excluded from the game by pressing the “Out” button. But beware: if the player was selecting a NPC, she/he will lose her/his cursor and will not be able to mark other characters anymore. Being unable to mark other characters, the players chances to win the game are dropping tremendously. If everyone loses her/his cursor by getting fooled into selecting the NPCs, the game will end as a draw. If someone manages to be the last survivor by kicking out the other players, she/he will win the game.

1.4 Art Style

It was important to us, to find a setting, in which we could easily design very distinguishable characters while keeping them in same topic. At the same time the characters had to stay very close to the human in body structure. That is why, we have chosen a mythical theme with animal like creatures. The initial plan was to design 10 different character meshes. Each of them would have traits of a specific animal, making them easily to distinguish from each other. That could have helped the in-game communication and amplify the metagame layer. Due to the time restriction we had to cut it down to one mesh with 10 different diffuse maps. The overall art style is simple and clean.

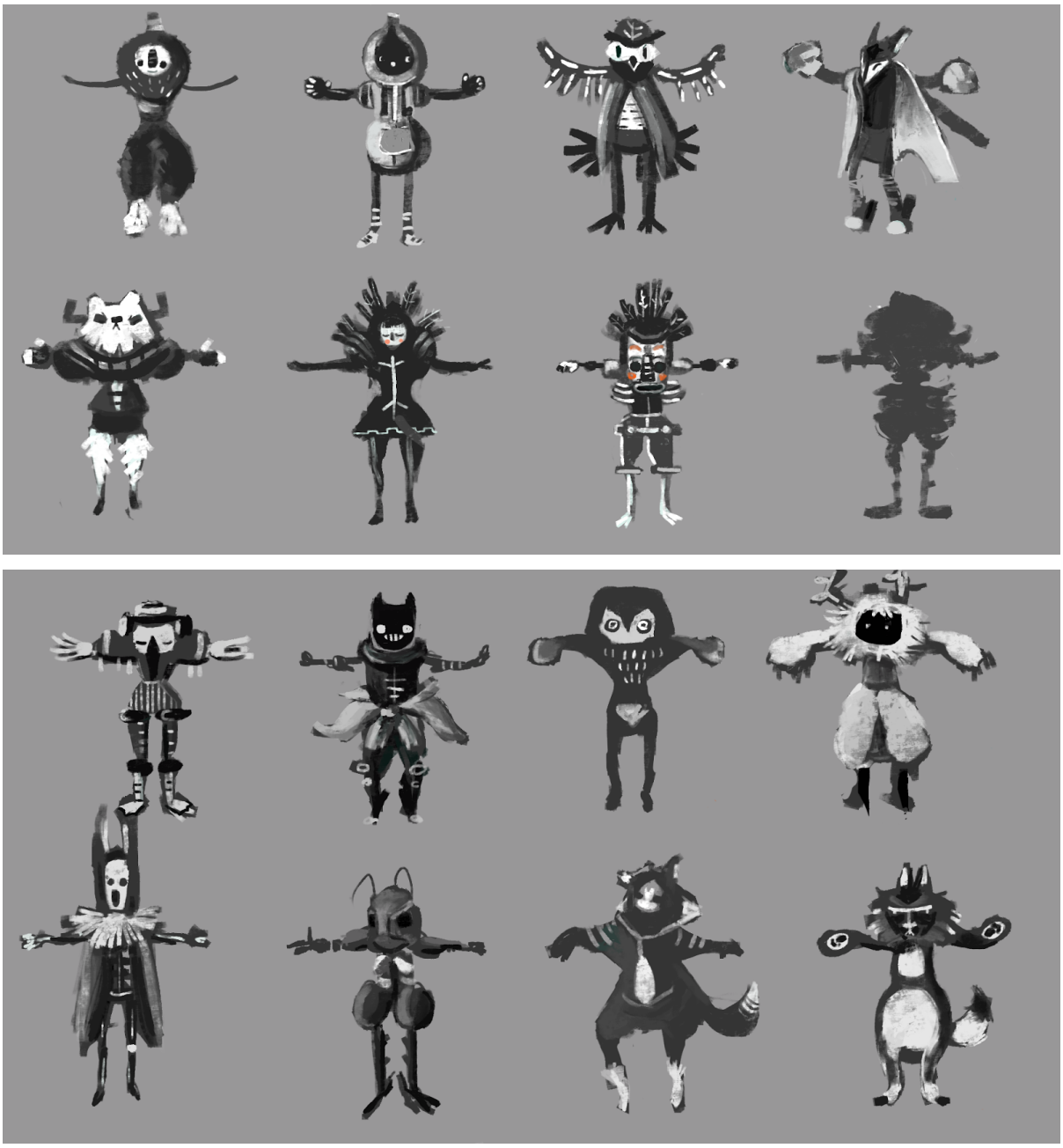

fig.2&3: early sketches of possible characters (sketched by Bettina)

fig.4: concept for different texturing for same mesh (sketched by Bettina)

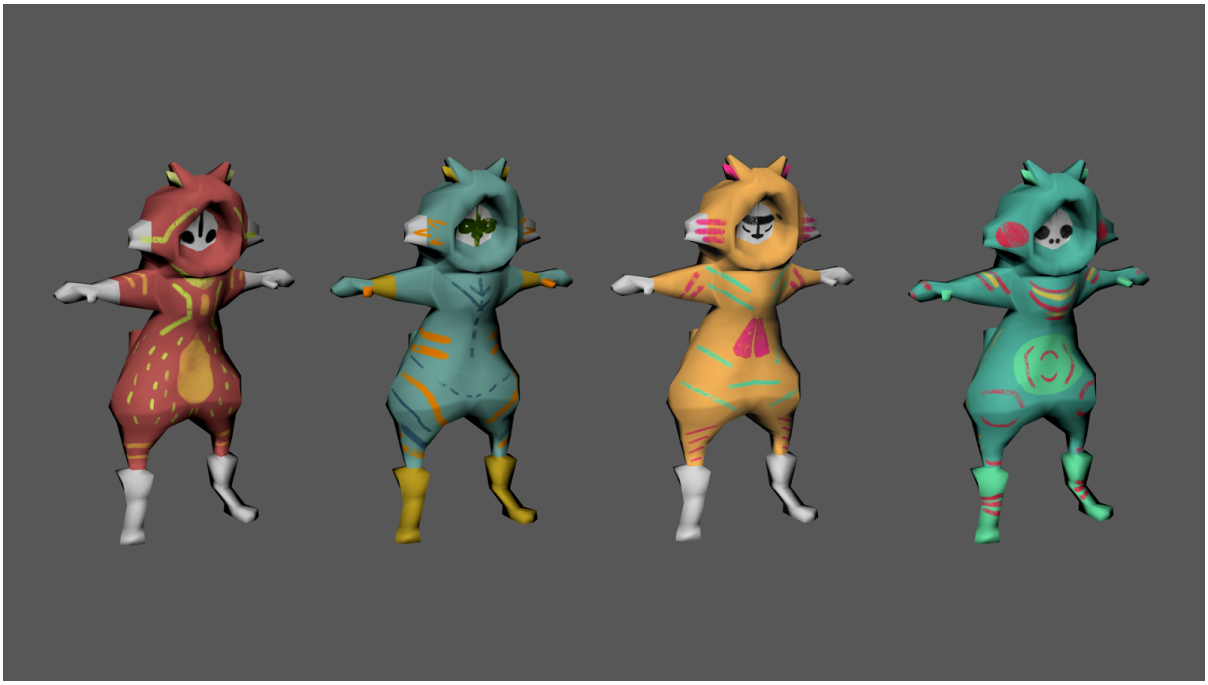

fig.5: first final characters (finalized by Bettina)

2. Personal Goals and Milestones

Because I was unsatisfied with my work-life balance during previous projects, I wanted to focus on finding the equilibrium. This includes: knowing the scope of the project, careful planning of tasks as well as identifying potential delays and progression blockers.

Furthermore, I still had the impression, that I had only scratched the surface of my programming capabilities and wanted to improve in that area. Especially, because I felt, like I have finally understood the basics pretty well, I would be now able to dig even deeper into this area.

Also as a sound enthusiast, I was really looking forward in designing the sound and improve my capabilities in sound modulation.

3. My Task Area

As the only programmer, my main task was to script the entire game mechanics. I had free choice over which game engine to choose. Since the last project was created with the Unreal Engine 4, I felt most comfortable with it.

fig.6: rough timetable

The timetable served as a rough sketch up for the weeks, it was difficult to estimate how long a task would take, because I have never worked with the Kinect before and the approach is sometimes very different than with conventional input devices.

3.1 Understanding Mircosoft's Kinect V2.0 (10h)

To say it was the first time for me to work with Microsoft's Kinect V2.0, would be an understatement. In fact, it was the first time for me to work with any input device other than keyboard or gamepad. Having this said; it was also the first time for me to even think about how to work with anything other than what the game engine would support out of the box. I had no idea what to expect and was really curious how my efforts in getting the Kinect to communicate with the game engine would pay out.

First, out of all steps, it was essential to understand how the Kinect works and what its capabilities are.

I started off by installing the Kinect SDK 2.0 and had a look at its performance in detecting human bodies and calculating the bones. The SDK is not only providing the position and orientation of the human bones, but is also capable of providing a depth map of the camera feed in a rate of 30 frames per second. It became very obvious, that the software (SDK) had problems in calculating bones, which were out of sight (e.g.: leg hidden behind the other leg), which would result in weird body twitching. Also the optimum results were achieved from a frontal perspective, which would expose all limbs most clearly to the camera. Body tracking from the side or back view did not work very well. We also tried a top down view (hanging the Kinect to the sealing), which did not work at all.

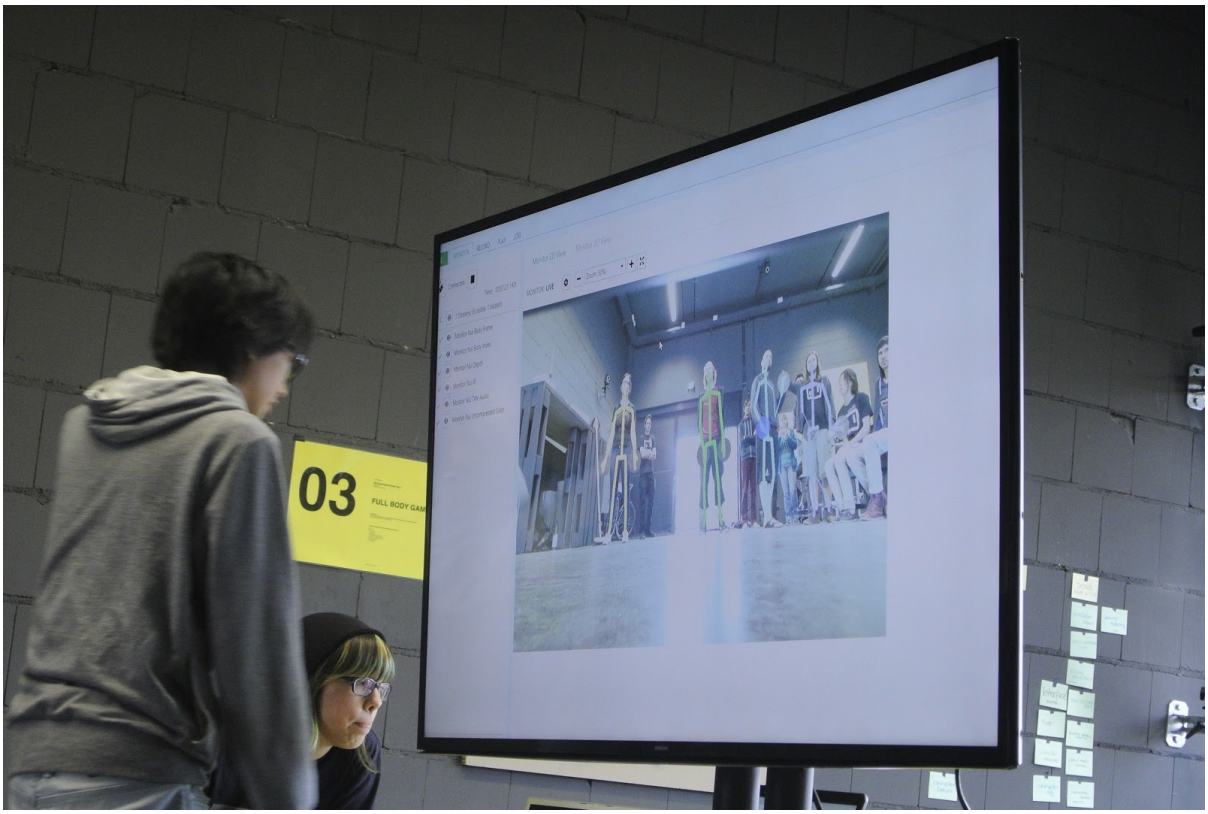

fig.7: testing the capabilites of the Kinect in combination with the SDK

We were positively surprised about the responsiveness of the Kinect. There was almost no lag noticeable. Before running the tests, we were already expecting the Kinect to be unable to track bodies from the top down view, but the incapability of calculating single hidden bones was a little bit disappointing.

3.2 Avateering (20h)

I had no idea how to make Microsoft's SDK2.0 to communicate with Epic's Unreal Engine 4. The short project period convinced me to look for a third party solution. I used a plugin called "Kinect 4 Unreal" by Opaque Multimedia, which provides a Kinect interface in Unreal Engine 4. This Kinect interface gives access to all the information provided by the SDK, which enables the function to read out the position and orientation of a specific joint of a specific body. Having the orientation of the bones enables avateering by passing the orientation of the actual bone onto the bone of the avatar.

Thanks to the third party solution, avateering was an easy task to achieve. Knowing that I most likely would not have been able to achieve this result within three weeks without using third party software, reminds me of how much I actually can still improve in my programming skills. Nonetheless, I did not expect from myself to be able to write a Kinect interface for Unreal Engine 4 yet and I am satisfied with this third party solution for now.

3.3 Assigning Positions and Meshes (10h)

Most of the gameplay is based on the fact, that the players do not know, which avatars are assigned to whom and which avatars are computer controlled.

In order to mix them all up, I have created a blueprint, called *BP_PlayerRep*. This *BP_PlayerRep* is representing one out of the ten avatars. Now, ten of these blueprints are being dragged into the scene and are positioned accordingly. They all have a fixed position for now, which will be mixed up at a later stage. For that reason, I used the construction script to save each position of the ten blueprints into an array called *allPossiblePositions*. This way I can drag and adjust the position of the blueprints in the scene whenever I want, while the array holding the positions is getting updated at the same time. Having this array of all possible positions, makes it an easy one to mix the positions up. The array itself is getting shuffled up randomly and each *BP_PlayerRep* is getting one random position out of this shuffled array.

The same principle applies to the meshes, which is realized by an array holding all possible character meshes. This array is being shuffled up as well and each *BP_PlayerRep* is getting one random character mesh out of this array.

Fig.8: testing the shuffle functions of all possible positions and meshes. (early footage)

3.4 Recording Movement (40h)

In order to make it difficult to distinguish between human players and NPCs, the players must behave very NPCs like or the NPCs very human like. I chose the later one, because I did not want the players to lose a game caused by malfunction or misinterpretation of the Kinect software and the resulting weird body twitching.

The idea was to record real human movements while they are playing and feed these data to NPCs in later games in order to achieve very natural body movements for the computer.

This task was a particular difficult one for me and took me by far the longest to achieve. In order to manage all the information for a record and playback management system I needed to structure the data very carefully.

I was laying out the structure as follow:

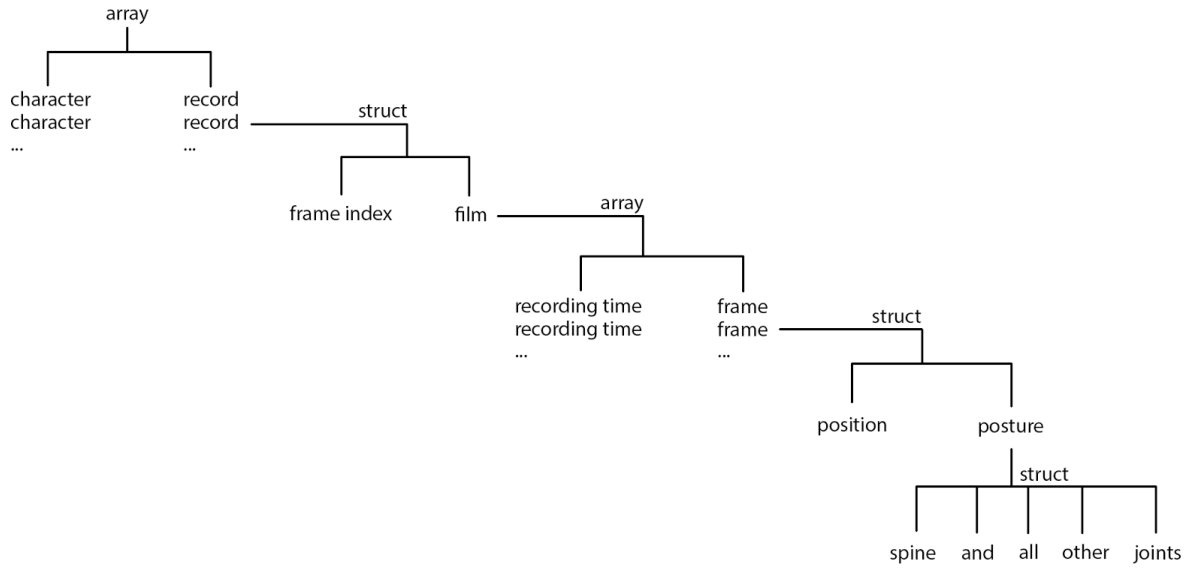

Fig.9: structure of the data for the recording and playback function

There is an array containing each *character*. Each of them is getting a *record*. This *record* is a struct containing a *frame index* and the actual *film*. The film is an array with the *recording time* for each saved *frame*. One single saved *frame* is a struct holding the information of the *position* and the *posture* of the skeleton. And finally, one *posture* is a struct containing all the orientation data of *all joints*, which together are forming the posture of the character.

While saving a *record* of a *character* the program is filling the array called *film* with *frames* and the according *recording time* of the *frame*. The *frame index* of the *record* is ignored for now, it will be used for later playback. The *recording time* is required to make the playback later independent from the frame rate. One saved *frame*, saves the *position* and *posture* of the character in this actual frame of the game. And finally, one saved *posture* contains the orientation of all *joints*, which are forming the actual posture of the character.

When playing the *record* on a NPC, all the data has to be feeded to the animation blueprint of the NPC. In this case it follows the same principle, just instead of saving, it reads out the *film*. The *frame index* starts at 0. Each in-game frame the program is looking into the array called *film* and starts at the given *frame index*. It then compares the *recording time* of the *frame* at the *frame index* with the actual game time. If the *recording time* of the *frame* is lower than the actual game time, the program will skip the the saved *frame* and set the *frame index* one integer higher until it reaches the point, where the *recording time* is higher than the actual game time. At this point, the program starts to dig into the saved *frame* for information about the *position* and *posture*, which is then feeded to the animation blueprint of the *character*.

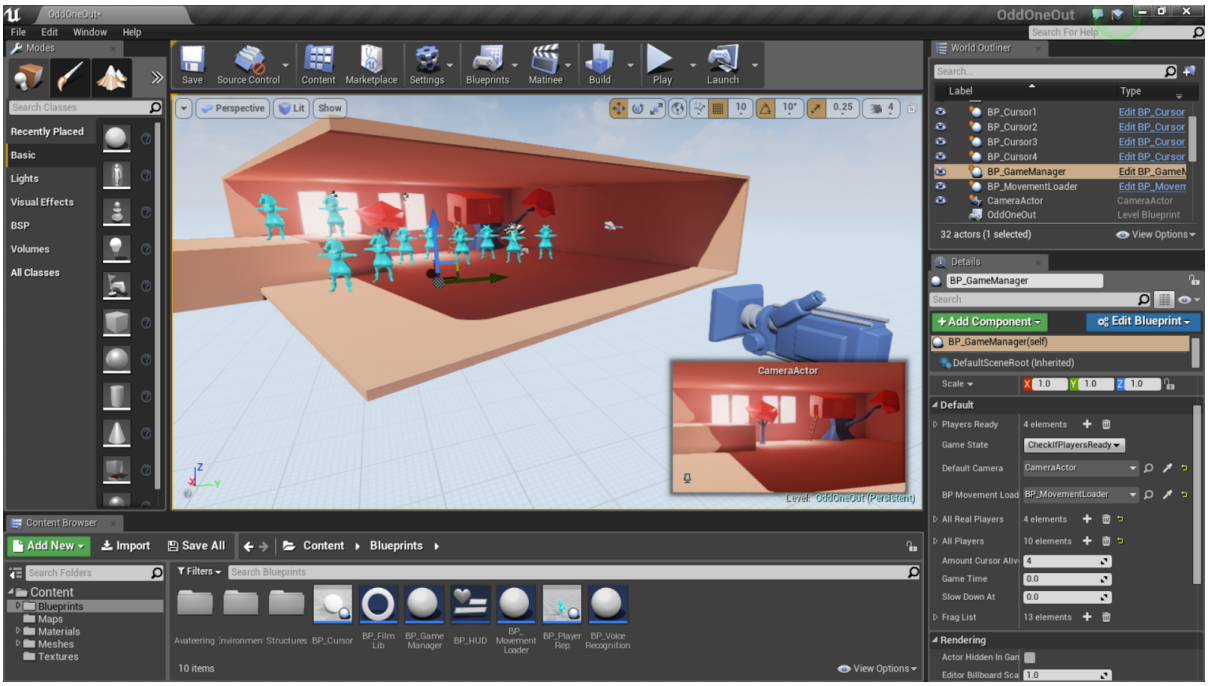

Fig.10:arranging the scene and the character positions. (early footage)

3.5 Implementing “Kick Out” (20h)

The players need the ability to somehow point out other players without getting exposed. My initial plan was to use voice recognition in order to call out players. The program would not only have to understand which avatar to reveal, but also from whom of the players the call out came from. The Kinect has four microphones and theoretically the ability to detect the direction of the noise source. But after several tests, I was not sure, whether it would work consistently. The uncertainty and the lack of time to implement proper voice recognition, forced me to look out for another solution. Another idea was to use hand gestures to select avatars and kick them out. This worked well for two players, but did not work at all for more than two, because the Kinect V2.0 is restricted in the number of hands it can track. All these circumstances forced me to implement the character selection through a cursor which is controlled by gamepads. This was a compromise solution and I am not happy about it. If I will find time, I will try to implement the voice recognition for kicking out suspicious characters.

I have created a blueprint called *BP_Cursor*. This blueprint contains an array with all positions from left to right. Each position is paired with an integer: position 0-9. This way the cursor can jump from one position to the next one or back. After the cursor has moved its position, it immediately looks up which character (*BP_PlayerRep*) is sharing the same position and follows its character mesh.

As soon as the “Out” button is pressed, the cursor is checking whether the selected *BP_PlayerRep* is the blueprint of a real player or of a NPC. If it is a real player, it will call the “Out” function on the enemy cursor and mark the *BP_PlayerRep* as “Out”. The “Out” function causes the controller of the owning player to vibrate and destroys the cursor. If it is a NPC, the cursor will destroy itself, leaving the player without a cursor and therefore without the chance to kick out other players.

This task was a very easy one, but the solution with gamepads is unsatisfying and I am looking forward to at least test it with voice recognition when implemented.

3.6 UI (20h)

The UI has to provide information about the game without being too distractive. For our game we wanted a minimalistic UI. The Unreal Engine 4 Widget function allowed me to blend in, blend out, color in and set opacity of UI elements through conditions.

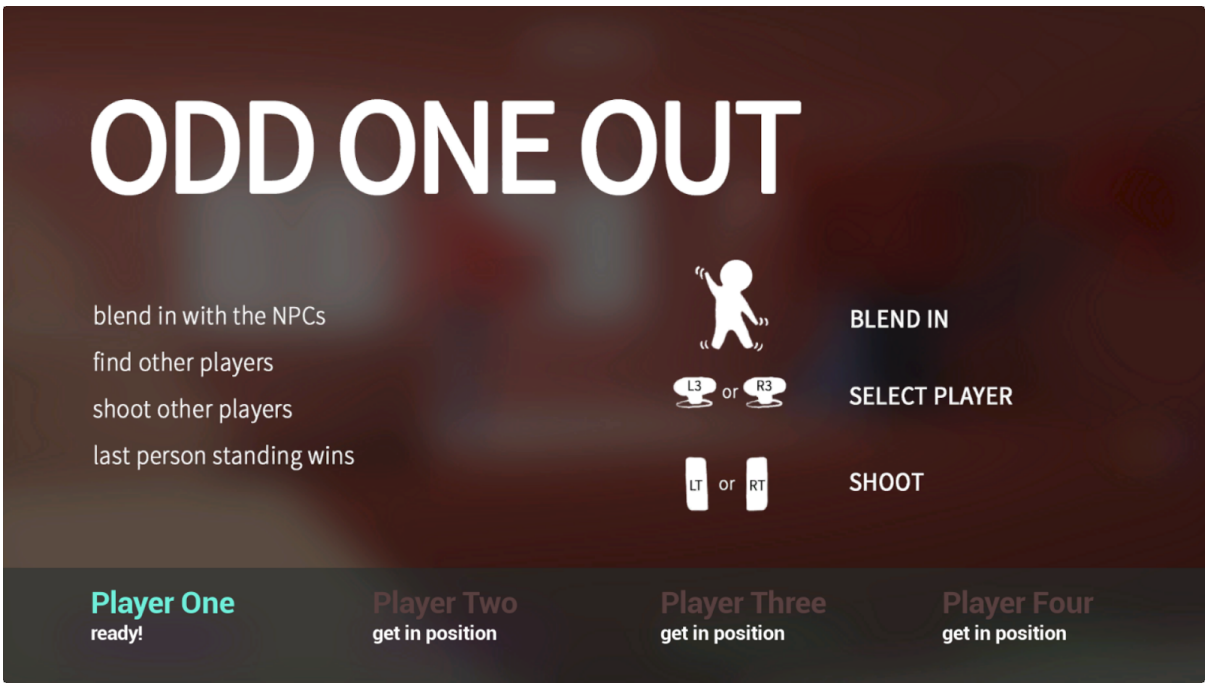

Fig.11: program is waiting for players (upper instruction screen made by Anna)

The winning screen contains strings, which are feeded by the blueprint *BP_GameManager*. That way, the winning string can adapt to the game outcome.

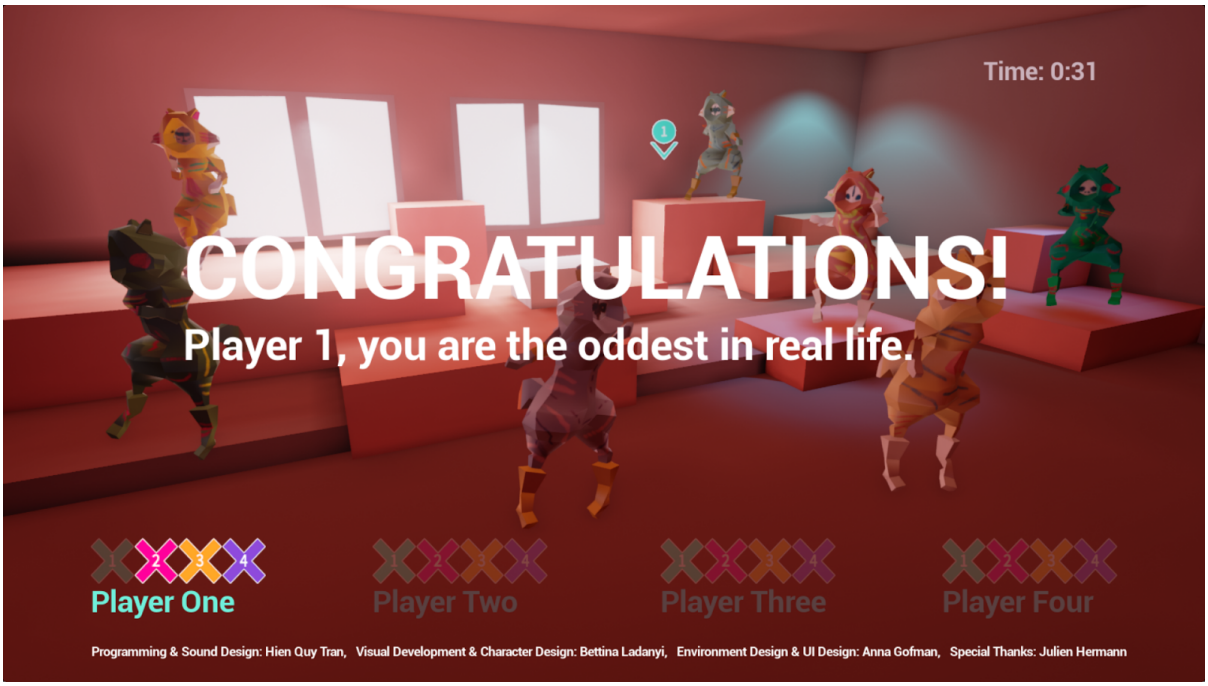

Fig.12: winning screen (cross icons made by Anna)

3.7 Sound Design/Implementation (5h)

As a sound enthusiast, I was looking forward to design the feedback sounds and the soundtrack for this game. After designing the soundtrack and feedback sounds for "Armville - Archie's Book Chase", I wanted to improve in this area even more.

Unfortunately, the time management did not went as well as expected, because the record body movement function was more difficult than expected and therefore took more time to complete. Because the recording function had a higher priority than the sound design, I had to cut down the time for designing the sounds drastically.

I used Ableton Live to create the soundtrack. I wanted to give the soundtrack a childish character, which is a little bit off the beat sometimes. I wanted the sound to be somewhere between of a wooden xylophone and a ukulele. For feedback sounds for the cursors I took notes within the harmony of the soundtrack. I choose a cymbal crash for kicking out another player and a loud off tune noise for trying to kick out a wrong character.

Considering, that I have spent about 5 hours into designing the feedback sounds and the soundtrack, I am very satisfied with the result. The sound fits the art style and is not irritating after several loops.

Fig.13: designing the sound

4. Recap

Looking back at this project, I would mark it as a great success. I made a great leap forward in programming, while maintaining a reasonable work-life balance. Especially the recording function forced me to keep my code tidy and structured. (Something I was always struggling in previous projects.) I have also gained a lot of new knowledge in programming by using structures and enums. The sound design part went well, although I was worried about the brief time at the end. The efficiency in sound design made me even more confident for future projects.

Although my work-life balance was great compared to previous projects, I still see room for improvement. The last three days of the three week project have been very exhausting, which is 15% of the project time. I want to aim for a even better schedule in which the crunch time is reduced to a maximum of 10%.

There are two things, which did not fit into the scope of a three week project: the proper implementation of voice recognition and the implementation of a Kinect interface without any third party software. These are the things, I am planning to look into sooner or later.

Also, I am still very interested in the programming of AI and would love to improve in this area at next possible chance. This time I had no chance, because there was no AI needed.

Overall, I am very happy with the result of the project and am very satisfied with my performance. I am looking forward to the next project!

Some more impressions of our studio work:

Fig.14: Betti and Quy working at the studio (from left to right)

Fig.16:Anna and Quy working at the studio (from left to right)

Fig.17: people playing our game at the studio

Fig.18: Betti and Anna working at the studio (from left to right)

Fig.19: people playing our game