COMP0113: Virtual Environments
Coursework 1: Group Project Report
Group 19

March 29th, 2023

Introduction

Exquisite corpse is a game that involves creating a drawing on a single paper where each participant contributes to a part, without seeing the other parts. The resulting creation is a collaborative and often bizarre piece of art.

We have developed this concept in a virtual environment that allows players to create exquisite corpse designs with other players in VR on a shared canvas. This provides players with the opportunity to collaborate and express their creativity in a digital realm.

This report will explore the development process of the game, including its design and implementation. The report will also highlight the part of the code or model that we found to be the most interesting in the development process.

Approach & Goals

Our group approached the task by first examining the traditional game and exploring ways to translate its mechanics into a virtual reality experience. We then split the development into different tasks:

- Networking - Developing the networking functionalities of the game.
- Canvas - Developing the drawing canvas.
- Menu - Developing menu panels for different functions of the game.

We then split these tasks among the members of the team.

Development & Highlights

Networking

We have taken into consideration the late join and early leave behavior of players in addition to network synchronization. This allows players to join and leave the room seamlessly at any point during gameplay.

Late Join

Late join features were implemented by listening for `OnJoinedRoom`. The newcomer requests information from players in the current room and updates the objects' states according to the data received.
Any player joining into the room after a game has begun will be considered a spectator. As shown in the left most image in Figure 2, a spectator will see both players drawing on the canvas. Upon request, current players in the room will fetch the current drawing from the `RenderTexture` object in the GPU, convert it to a `Texture2D`, encode the texture as a `PNG` to reduce size, then convert the bytes to `Base64 String` for transport. The newcomer decodes this information, recreates the `Texture2D`, then updates its corresponding

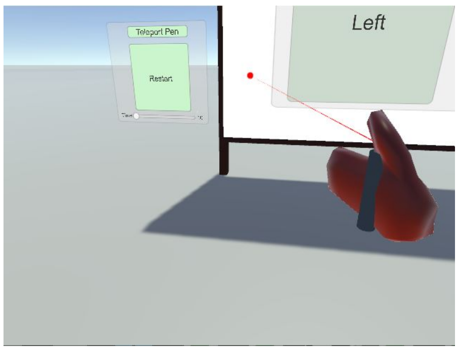
`RenderTexture` object in the GPU, which renders the current drawing onto their canvas. Late join behavior was also implemented for the shared menus.

- If a player has already selected a side to draw on, then a second player joins the room, said option would become unavailable for the second player.
- Any newcomers to the room will see the most current state (value) of the timer slider

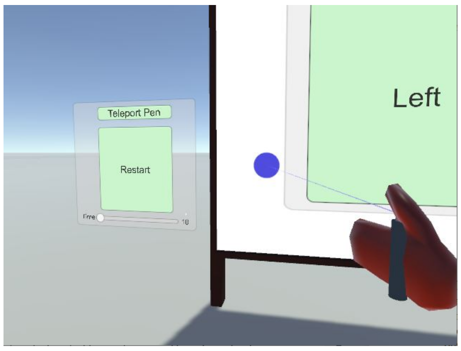
Early leave

A spectator leaving the room would not impact game play. A participating player leaving the room during an ongoing game will restart the game. The feature was implemented by listening for `OnPeerRemoved` and resetting the game if the removed peer's `NetworkID` is a participating player's.

Canvas



(a) Smaller brush size with color set to red



(b) Larger brush size with color set to blue

Figure 1: Drawing Pen, Laser Effect, and Laser Pointer

The traditional game involves players creating a drawing without seeing the other parts. We decided to translate this experience to virtual reality by creating a shared canvas to increase its social and collaborative aspects.

As shown in the left and middle images in Figure 2, our group decided to incorporate a shared region in the canvas where both players could see each other's drawing to facilitate the connection between the different parts. This shared region is 10% of the overall canvas.

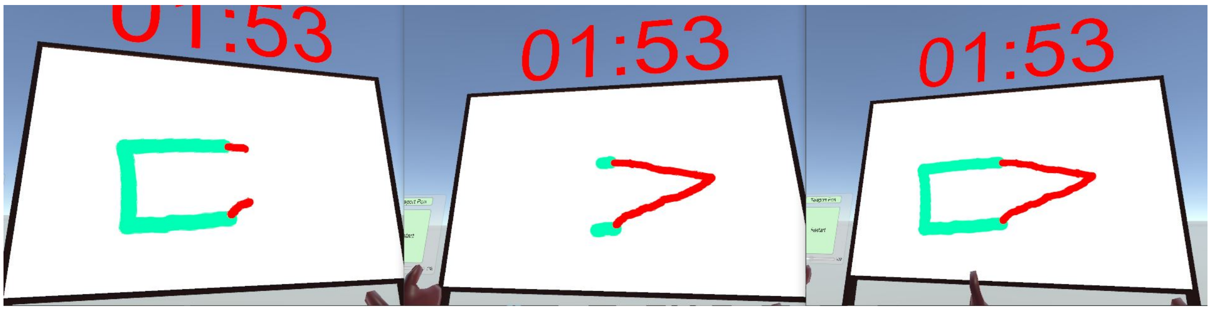


Figure 2: Game Play Demo
Left & Middle images are from participating players. Right image is from a spectator.

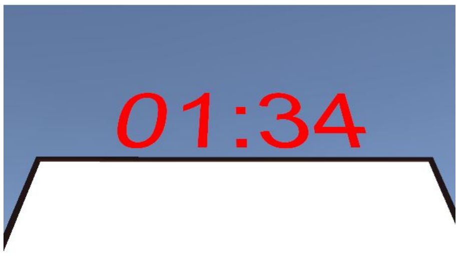
The drawing on a canvas is achieved through the use of a `RenderTexture` coupled with a fragment shader. During `LateUpdate` a ray is cast from the pen, if the ray hits the canvas, then the hit position is recorded. Once two points have been recorded, the shader is invoked with:

1. Start Position
2. End Position
3. Brush Colour
4. Brush Size

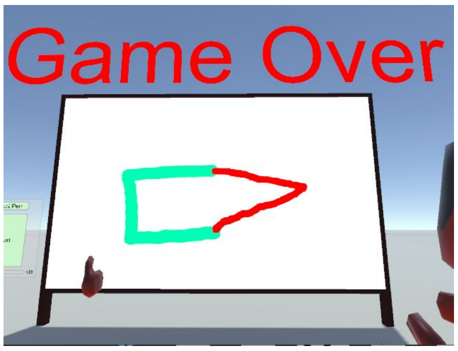
The fragment shader then checks whether the distance between the current UV coordinates and the line defined by the start and end positions is less than the brush size. If the fragment is within the defined distance, a blend between the existing texel and the brush colour is returned. The blend is done by using a linear interpolation between the two colours, however, the blending only occurs if the distance from the line is at least 80% of the brush size. This results in nicely anti-aliased lines. The shader fills in all texels along a line, instead of around a single point, to compensate for the speed of a player's strokes. If the shader was to only take a single point and the player were to make a fast stroke then there is potential for there to be gaps in their stroke. By recording two points, we can at least ensure a consistent line.

To handle shared and private regions, each player's canvas is managing two separate `RenderTextures`. All of the player's local brush strokes are rendered to both textures. Whenever a message containing a brush stroke is received from another player, one of the two textures will be updated with the command. The other texture will only be updated if the stroke is within the predetermined shared region. During gameplay, the players can only see the texture containing their strokes and the publicly shared strokes. Once the game ends, the rendered texture is swapped to the texture containing all brush strokes to reveal the final result.

Once the number of participants registered with the canvas matches the `playerCount` variable, the canvas will begin emitting `DrawingBegunEvent` whenever a participant attempts to draw whilst the game is still in the preparation phase. The `GameSystem.BeginGame` is connected to this event, which in turn changes the game's state to `State.InProgress`. The canvas will receive the notification that the game state has changed and begin allowing the participants to draw. During gameplay, if any of the canvas's recorded participants leaves the room the canvas will trigger an `ActivePlayerLeftEvent`. This allows for other components to make decisions when this happens. For our implementation, this is connected to `GameSystem.ResetGame`, such that once a player leaves the game will be reset.



(a) Remaning Time Countdown



(b) Game Over Screen

Figure 3: Game Status

Menu

We have implemented several menu panels for the game namely:

1. Brush Settings Control Panel

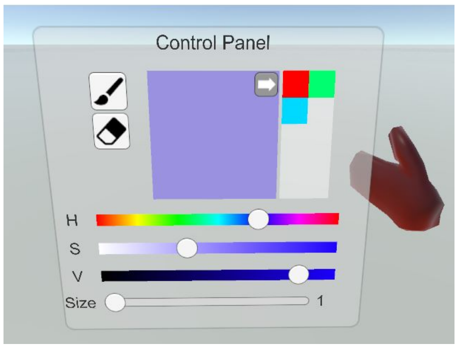


Figure 4: Brush Settings Control Panel

The panel allows players to choose between two tools: the brush and the eraser. The brush tool allows players to paint their designs on the canvas in a range of colors and sizes, while the eraser tool lets them selectively erase parts of their drawing by changing the colour of the brush to white. Players can also fully customize their colours and add them to a palette. To create a specific color, players can adjust the three sliders - Hue, Saturation, and Value - until they achieve their desired color. In addition, the size of the brush can be adjusted using the size slider.

2. Restart Menu

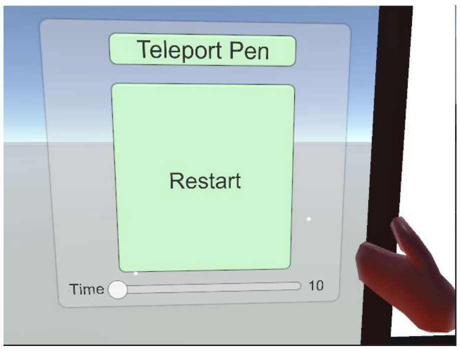
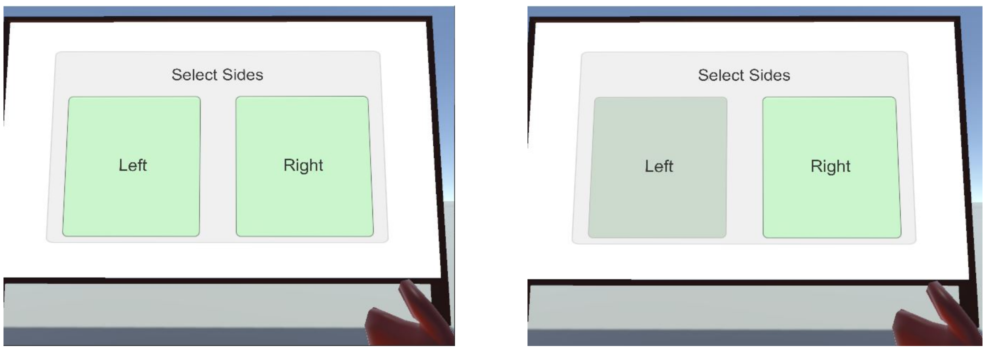


Figure 5: Restart Menu

The restart menu allows players to restart the game manually. Players can also adjust the duration of the game using the time slider as seen in Fig 5. In addition to this, they can also teleport the pen to its original position using teleport pen button.

3. Player Selection Menu



(a) Initial state (b) Upon selection

Figure 6: Player Selection Menu

Upon entering the game, a Select Sides panel will appear on the canvas. This allows players to choose which side they want to draw on. Once a side has been selected, its box will turn grey to indicate that it has already been selected by another player as seen in Fig 6.

Conclusion

In conclusion the team has successfully implemented all the planned features. Looking to the future, we believe that there is great potential for the game to expand and evolve. Further developments could include additional tools and features, such as a wider range of brush types and more complex interaction methods.