AzTech - Cooperative Portal Puzzle Games

Jeanne Beyazian	Email id: ucabje1@ucl.ac.uk	SN: 21204497
Yicheng Zhang	Email id: ucaby83@ucl.ac.uk	SN: 18052846
James Harding	Email id: ucabjs9@ucl.ac.uk	SN: 21162043
Noorpreet Soni	Email id: zcabnks@ucl.ac.uk	SN: 17050830

Brief The goal of the game revolves around solving puzzles as a team that involve interactions with objects in the scene in order to reach a specific location.

Description We imagine that the players are impersonating explorers walking through some ancient mystical ruins generally themed as belonging to an antique prehispanic civilization (such as the Aztec). Throughout their journey, the characters discover a set of magical equipment and other interaction locations that will be key to solving the three-dimensional puzzles we have set up in the scene. These objects are described later on in this report.

1. Overview & Social Task

The sample level we have created presents a variety of game mechanics that are arranged in a specific way to motivate and even force multiplayer collaboration.
To complete a challenge, it is imperative that players not only **discuss** together but also **synchronise their actions** and use each other's tools to solve the puzzles.

The level takes the following form: several players are in the same room. They have access to a number of tools they can interact or equip themselves with. Once a portative tool is selected by a player, the player will be able to use its power. Since there are different kinds of tools, everyone's participation is essential to move forward in the game. A selected tool can be

dropped if a player would like to exchange with another player's tool. All players have the ability to interact with levers, pressure plates and can teleport themselves to a nearby location.

We have implemented a **collaborative** game experience through a client-server model, which will allow up to 4 players to connect to the server using the UCL ubiq network library. The players share a common scene and are all connected to the server to appear in each other's worlds. Data regarding changes on networked objects within the scene is shared immediately with the server via a JSON message system. This allows the connected players' worlds to be updated in a synchronised way. The shared message may include information about creation/destruction of portals, triggering of interaction objects (which in turn may cause a moving platform to transform its location) as well as location of other players in the scene. This model attempts to ensure that local plausibility is maintained throughout the duration of the game.
The social aspect of the game is crucial. We designed the game map in a way that encourages player discussion. Through the ubiq connection, they have access to a proximity chat and can discuss their next actions to move forward in the game. Moreover, we created different kinds of wand with each defining a specific playstyle. All these roles complement each other. For example, if players would like to create a portal in the world, they will start by using the entry wand to create an entry portal, and then the exit wand to create the exit portal. This is especially relevant when both players are in different locations due to puzzle constraints (e.g. a player must control a moving platform for another player through a lever or a pressure plate).

2. Mechanics & Remarkable Features

Wands

- **Entry Portal wand**: wand that creates entry portals on walls tagged with 'Wall'
- **Exit Portal wand**: wand that creates exit portals on walls tagged with 'Wall'
- **Destroy wand** : wand that deletes a portal it lands on. Given the limit of inactive portal at once in a game
- **Alternator wand** : wand that creates entry and exit portals in alternation.

Interaction tools

- **Moving platforms** : unlike the floor objects, these specific platforms can move from one static anchor to another. The moving platform bundle was defined as a prefab constituted of two platform objects, a start anchor that indicates the start position of the platform, and an end anchor**.**
- **Waypoints** : to facilitate navigation around the map when a player falls off, waypoints serve as respawn locations and are scattered along the path to the end. Walking on them is enough to activate them.
- **Levers** : the levers can be activated by any player and can control moving platforms. Pulling a lever once moves a platform to its end position. Pulling the lever again moves the platform back to its original position. Each lever has a cooldown to make sure players do not spam the interaction and that the animation is generated properly.

- **Portals** : a portal can be of two types : entry or exit. A portal can be either active if it is linked to another portal of the opposite type, or inactive if it is alone. An active portal has green borders, an inactive portal has red borders. If a player enters through a linked entry portal, they will arrive at the location of its corresponding exit portal in a smooth way. Entering an exit portal will teleport the player to where they already are. All portals show the view at their destination using a portal camera child in the portal prefab.

Implementation

Throughout this project, we have put emphasis on maintainability as well as extendability. We started with only one wand that would alternate between entry and exit portals. However, this behaviour gives too much power to a single player, who could potentially take the role of two or more people. Using class inheritance, we designed 3 more wands with the purpose of making the game more complex and more collaboration-based.

Some features that are worth mentioning are the following:

- We implemented cooldowns on the firing of portal projectiles from the wand, on the levers and on the teleporting system (how frequently a player can teleport). This keeps a more realistic balance of actions and stops players being stuck in an endless portal loop.
- We added a lifetime on portal projectiles which defines how long they bounce around in the map without reaching a portal wall to keep the map clean, and reduce clutter.
- A remarkable feature is the portal linking process: we keep 4 static lists of portals - inactive entry and exit portals, and active entry and exit portals. When a new portal is instantiated, it is added to its corresponding inactive list. If there is a portal in the **opposite inactive list** with the **same index**, they can be **linked**. The entry portal gains the **reflection** of the exit portal and both are moved to their corresponding active lists.
    - Keeping track of the portals also allows us to set a **limit** on the number of portals that exist in the world at a time. If a new portal is instantiated but the limits on the lists are already reached, the oldest portal the scene is destroyed and the new one is added. When a linked portal is targeted with a destroy projectile and if the limit of inactive portals is already reached, both linked entry and exit portals will be deleted, otherwise its pair will simply become inactive.
    - When a projectile creates a portal, it also sends a message to the server to update other players about the creation of the new portal and its location. This is in case client-side collision does not occur so everyone knows to create a portal and maintain synchronisation.
- The portal search algorithm is crucial to keep the players in synchronisation when a portal is deleted. We have encountered an issue when introducing the deletion wand: we think that due to variations in physics engine calculations or latency from machine to machine, a collision may happen for one player but not for another, therefore a message is sent to the server containing the portal ID of the destroyed portal so that every client knows to delete it.
- For interaction items such as levers or pressure plates, we implemented the Triggerable virtual class. This allows any kind of interaction item to keep a list of the objects they can

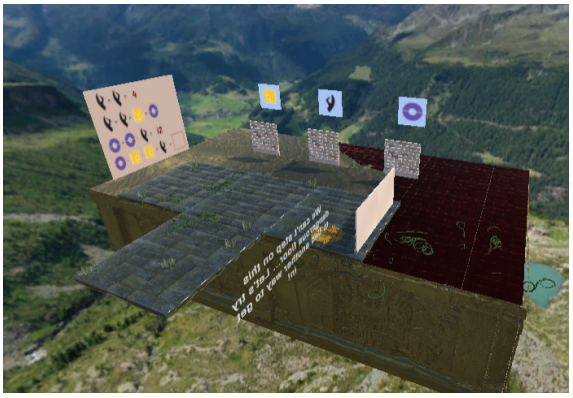
trigger. For example, the lever registers the moving platforms it is responsible for, as the moving platform script inherits Triggerable. This gives the `isTriggered` boolean to the platform which controls its status. When an interaction occurs, `isTriggered` is updated in the virtual method `beTriggered.` This gives freedom when the effects of the interaction are implemented.

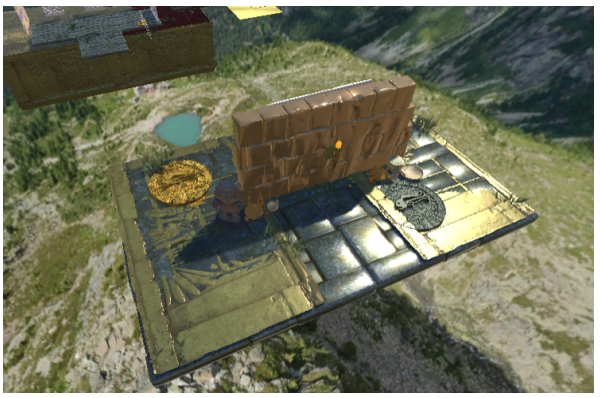
3. Design

The design and modelling part of the project was crucial to create a sense of immersion. The bigger elements were kept as 'low-poly' as possible but we allowed some more realistic smaller items to give the idea of a more realistic Aztec ruins.

We found our assets on free 3D models websites, such as SketchFab.com or we created them ourselves in Blender. The licences for imported assets are included in the Assets folder. Some of these models come from cultural institutions and are reproductions of real-life prehispanic artefacts. Notably from the National Natural History Museum of Chile (Museo Nacional de Historia Natural de Chile) or the Peru Ministry of Culture (Ministerio de Cultura Perú).

We opted for floating platforms as they are ideal for vision across the map and seemed like the right choice for a portal based game. Players can see points of interest from a distance which helps them figure out the direction.





4. Software engineering process

For this group project, we followed agile methodologies as it was our first time building a game in Unity for all of us and we knew we would have to experiment a little bit. We worked iteratively using Kanban boards(on Trello.com) to assign tasks and used GitHub to work collaboratively on the source code and assets. We dedicated the first two weeks to defining what the project entails, analysing the requirements and storyboarding our ideas. In the following weeks, we started creating possible scenarios and puzzles. We also focused on the design of the game and started gathering assets or modelling them using Blender.

To split up the work, we defined a few different categories of assets:

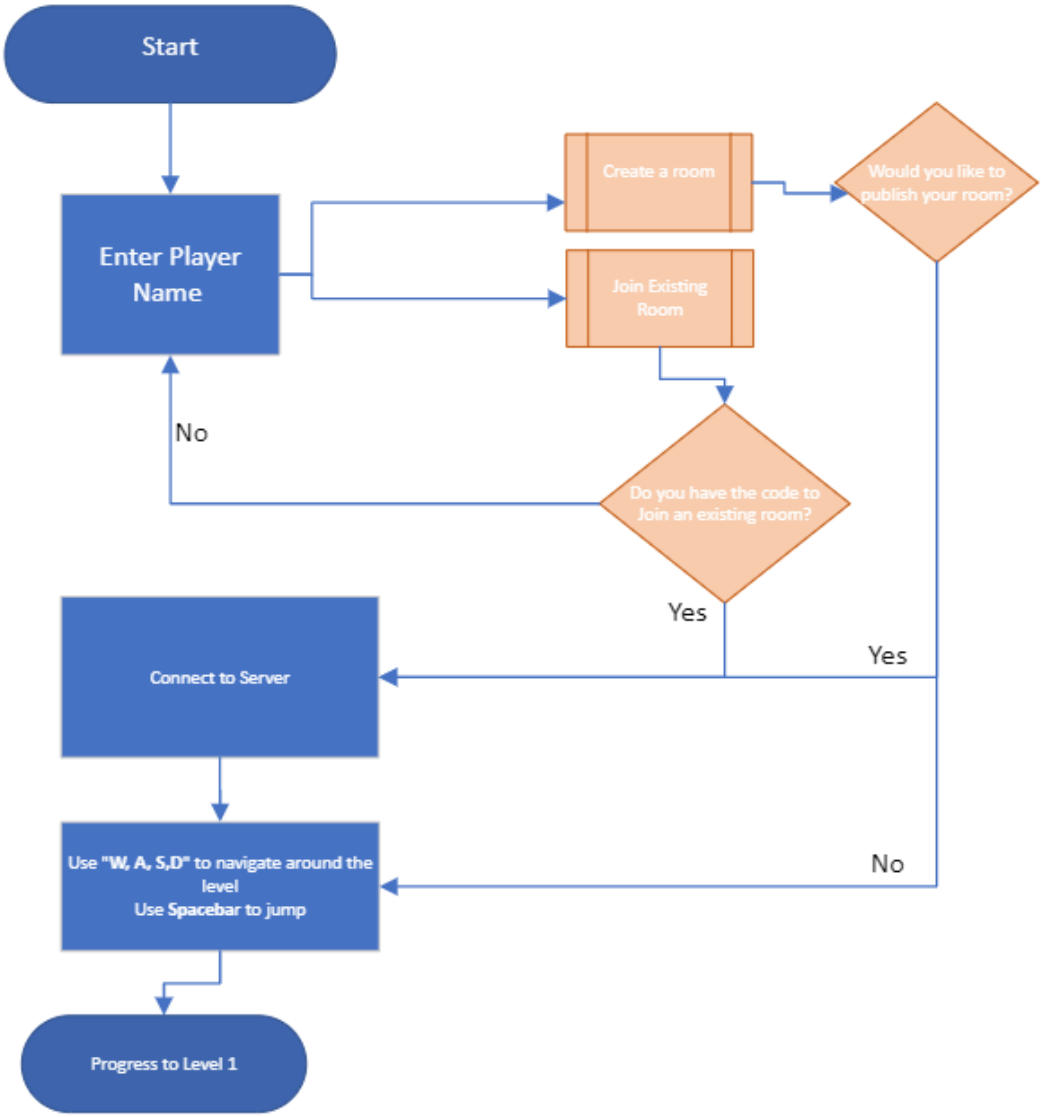
- Walls and floors: portal and non-portal walls, static floor and moving platforms

- Interaction objects: levers, pressure plates, waypoints.
- Portative equipment: portal wands
- Decorative objects.

In the last week of February, we started experimenting with Unity and implemented the smaller assets scripts.

In March, we moved on to the implementation of the most important parts such as class hierarchies for the wands and the implementation of the portal spawning and linking system.

APPENDIX : GAME GUIDE



On a Desktop

- W - Forward
- A - Right
- D - Left
- S - Backward
- Scroll Wheel click - wand pickup
- Spacebar - Jump
- Shift - Sprint

Oculus

- Use Thumbsticks to navigate through the virtual environment.
- Trigger (on the front of the controller), A and X buttons select objects in your environment.
- Grip button (on the side of the controller) grabs objects or makes a fist when using your virtual hands.

Move Platform

- Use a lever to activate the moving platform or step on a pressure plate.

Create portal

- Use the Entry portal, aim at a wall and shoot projectiles to create Entry portal.
- Use Exit portal , aim at a different wall or part of the wall and shoot projectiles to create Exit portal.
- Ensure both exit and entry portals exist before using a portal.

Destroy Portal

- Grab the Destroy portal wand.
- Shoot projectile at either entry or exit portal to destroy it.

Move between floors

- Pull Lever to bring the platform down, push lever to move platform up.
- Ensure that the user is close enough to the lever to pull or push it.