Video Game Design--Unknot

by

Jixiang Chen, Collin

(1930026011)

Shijie Cheng, Dean

(1930026019)

A Final Year Project Thesis (COMP4004; 3 Credits)

submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Science (Honours)

in

Computer Science and Technology

at

BNU-HKBU

UNITED INTERNATIONAL COLLEGE

December, 2022

DECLARATION

I(We) hereby declare that all the work done in this Project is of my(our) independent effort. I(We) also certify that I have never submitted the idea and product of this Project for academic or employment credits.

Mr. Chen: In charge of game process development, puzzle producing, puzzle solving algorithm development and bug fixing. Responsible for Chapter 1.1 and Chapter 4.

Mr. Cheng: In charge of partial detailed design, map construction, animation and character construction, puzzle producing, puzzle solving algorithm development and bug fixing. Responsible for Chapter 1.2, Chapter 2, Chapter 3 and Chapter 5.

MAD students: In charge of most of the detailed design and the producing of map materials.

_____

Jixiang Chen, Collin (1930026011)

Shijie Cheng, Dean (1930026019)

Date: _____

BNU-HKBU

United International College

Computer Science and Technology Program

We hereby recommend that the Project submitted by Chen and Cheng and entitled " Video Game Design--Unknot" be accepted in partial fulfillment of the requirements for the degree of Bachelor (Honours) of Science in Computer Science and Technology Program.

_____ _____

Date: _____ **Date:** _____

ACKNOWLEDGEMENT

I(We) would like to express my great gratitude towards my supervisor, Dr. Zhiyuan Li, Goliath who had given me invaluable advice to this project.

Video Game Design--Unknot

by

Jixiang Chen, Collin

(1930026011)

Shijie Cheng, Dean

(1930026019)

A Final Year Project Thesis (COMP4004; 3 Credits)

submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Science (Honours)

in

Computer Science and Technology

at

BNU-HKBU

UNITED INTERNATIONAL COLLEGE

December, 2022

ABSTRACT

The origins of puzzle games can be traced back thousands of years, when people already created several interesting and challenging puzzles. Nowadays, with the advent of new digital media such as video games, puzzle games started to come into their own, and became one of the mainstream game categories on the market. However, as a game type with a high degree of fun and playability, there is still a lot of potential in puzzle games waiting to be explored.

Over the past 30 years, game design has become a discipline with a systematic theory and methodology. By developing game prototypes, game designers test and evaluate to come up with interesting game design solutions, including but not limited to level design, system design, world view design, etc. This thesis introduces the design process of a knot puzzle game, a type of puzzle that has not yet been transplanted to video games. Through this attempt, we look forward to more innovation in the puzzle game type, including but not limited to adding new gameplay, developing more puzzles that can be transplanted to computer games, and expanding the gameplay, artistry and commerciality of puzzle games.

Contents

1 Introduction

1.1 What is Puzzle Games?

Puzzle games, also known as puzzle adventure games, are a type of intelligence game that solves various puzzles and discovering clues by analyzing and processing the information or plots that appear in the game. As one of the game categories with the widest range of users, puzzle games have been created and played by people for a long, long time. Some of the better known ones are Tangrams, Klotski and Lupin Locks.

Nowadays, with the advent of new digital media such as video games, puzzle games started to come into their own, and became one of the mainstream game categories on the market. Take single-player computer games as an example, puzzle game or puzzle element appears in several games genres. Adventure games are full of puzzles, frequently about obtaining inaccessible objects or getting information from other people; and even first-person shooters offer the occasional puzzle, figuring out how to get past locked doors and other obstacles (Adams, 2014). Until now, Puzzle games can be generally divided into two categories: traditional puzzle games and adventure puzzle games.

Traditional puzzle games are generally based on a certain type of puzzle, then design the game and gameplay around that puzzle type. Some of the classic ones are Angry Birds, Where's My Water, Cube Escape and so on. With the continuous development, traditional puzzle games are constantly being added with a variety of new gameplay and systems. For example Mirror 2, which adds animation elements and text adventure gameplay to traditional Match-3 puzzle games.

Figure 1 Visual reference of Mirror2

As the more popular category of puzzle games nowadays, adventure puzzle games focus on players controlling the character to reach unknown places, solve puzzles and trigger other plot and exploratory interactions. This type of game is mostly based on the theme of following the clues set in the game to solve the final mystery of the game. It generally has a gorgeous plot, beautiful graphics, and emphasizes the discovery of story clues, which mainly tests the player's observation and analysis ability. Some of the more famous examples are Braid, Inside, Monument Valley and so on.

Figure 2 Visual reference of Inside

1.2 Inspiration: Combination Plus New Puzzle

However, there are only few puzzle games that combine these two categories. For this reason, we want to try to combine the traditional puzzle game with the adventure puzzle game: a puzzle game with a specific type of puzzle as the core gameplay, combined with explorable puzzle elements to drive the game process and story plot. So that players can not only enjoy the traditional form of puzzle solving, but also the explorable collection elements can bring more immersion and freedom for players.

For the choice of puzzles, we choose a form of puzzle that has never been transplanted to video games before: the knot puzzle. In our game, the player needs to manipulate the knot freely (including dragging, pulling, rotating, etc.) to make it into its simplest state and eventually unravel it. The final unraveled state of the rope will be displayed on one side of the screen. Only when the player unknotted the knot, the whole game process will be possible to continue to move forward.

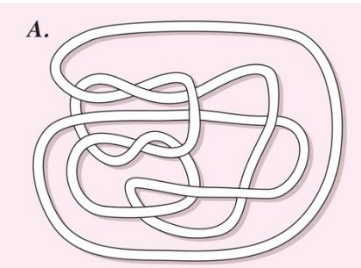

Figure 3 Knot Puzzle

From the perspective of game design, knot puzzles are interesting and challenging enough as a game in themselves. Secondly, we believe that mathematical knot puzzles are closely related to the "puzzles" in life, and essentially they are the same, and such characteristic also provides us with more possibilities to explore the artistic connotation of knots, including but not limited to adding new gameplay, expanding the artistry and commerciality of puzzle games. For example, the combination of knots and geometry (cylinders, squares, etc.) is also a good way to try out this puzzle format.

From the perspective of game programming, the knot puzzle itself is complex enough, and the flexibility and complexity of the rope itself involves complex puzzle solving algorithms and performance optimization. This requires a high level of mastery of both the game engine and programming language. We thought it would be challenging and worthwhile to try this as our final year project. It was also a lot of fun: creating a type of puzzle that has never been seen in a video game before!

In order to complete this project to a higher degree, we contacted the Media Art and Design students to work together. In this project, the MAD students are responsible for the global design of the game and the production of the game materials, while the CST students are responsible for the programming of the game. The project will be divided into two phases: in the first semester of the senior year, the CST students will implement and verify the techniques based on the basic design plan, such as the simulation of the rope physics system and the knot puzzle algorithm; in the second semester of the senior year, the complete game will be constructed based on the detailed design plan and game materials provided by the MAD students.

2 Merits From Existing Puzzle Games

2.1 Braid: Innovative Level Design and “Rewind” Gameplay

Braid is an standard 2D puzzle-platform indie game developed by Number None. The player controls Tim, who is a little guy in a business suit, to travel through 6 worlds and try to save the princess from monsters.

The basic gameplay of Braid is classic. Arrow keys make your character Tim move left, right, up and down, and the space key makes your character jump. Players control Tim as he runs, jumps and crawls through the game's various levels. Tim defeats enemies by stomping on them and is able to collect items like keys to open doors or pull levers to

activate some platforms. Player must collect the pieces contain in each level, to unlock the last level and create the jigsaw puzzles that tell Braid's main story.

Figure 4 Visual reference of Braid

The most special, interesting and attractive part of Braid is its "rewind" gameplay. Tim travels to six unique worlds and each world has its own time mechanics. Players can use these time-based game mechanics to reverse time and rewind the action to solve the level's puzzles, and the entire game process is driven by such a game mechanic. Such imaginative and flexible mechanic makes the whole game playful and philosophical and creates thoughtfulness and harmony between its various aspects, which is the factor to make Braid a really interesting project (Hellman, 2008).

Figure 5 The shadow performs the actions that player rewound

What really makes Braid a magnificent art is the way to connect all the different elements together. The story feeds into game mechanics that change your perception of time, and this new perception affects your reading of the story. The story ideas also act as a metaphor for the overall structure of the game, which in turn fills in the ending. Braid is like a giant jigsaw puzzle made up of some of the most interesting and innovative gameplay in stages. All of this is packed into a very compact and elegant package, an ideal format (Rohrer, 2007).

2.2 Inside: Fascinating World View and Atmosphere Building

Inside is a standard 2.5D puzzle-platformer adventure game developed by Playdead. Its control mode is quite similar to Braid, arrow keys make your character move left, right, up and down, and the space key makes your character jump. There is no main objective for the player in this game. All the player needs to do is to keep moving, avoiding death and gradually uncovering the hidden truths of the world as he goes along.

Figure 6 Huddle, the amalgam of body parts that the player controls at the end of the game

The most intriguing aspect of Inside is its antiutopian worldview setting based on puzzle solving and the environment created to reinforce it. The whole story begins with a scenario in which a boy slides down a rocky incline, as the player controls the boy's actions to advance the game, the player will encounter a great variety of level elements, such as masked guards, fierce guard dogs, parasitic worms cause pigs to run rampant, lines of zombie-like people, a large factory of flooded rooms, a laboratory environment where scientists perform underwater experiments on bodies, and Huddle (Tamburro, 2016).

In Inside, the player needs to interact with these elements to find out the way to move on as they progress through the levels, and gradually uncover the secrets hidden deep within the world. To reinforce this world view narrative, the game has put a great effort in the creation of the atmosphere. The game is dark, with color used sparingly to highlight both the player or certain parts of the environment. The light is used cleverly, and there are even several puzzles involving shadows occlusion. The game is mostly silent, with the exception of occasional musical cues, the boy's vocals, dog's bark, equipment and sound effects, so that the oppressive and tense atmosphere of the game is well represented (Rignall, 2016).

Figure 7 Visual reference of Inside

2.3 Monument Valley: Innovative Labyrinth and Visual Stimulation

Figure 8 Visual reference of Monument Valley

Monument Valley is an indie 2.5D labyrinth puzzle game based on architecture, geometry and illusions developed and published by Ustwo Games. In this game, the labyrinth is composed of optical illusions and impossible objects, which are referred to as "sacred geometry" in-game (Stinson, 2013). Players needs interact with environment by creating bridges, rotating or moving platforms and pillars etc. to find hidden exit routes from the map for the main character, Ida. The game's colors and other design elements will provide indirect hints to the player's puzzle solving process, while the crows blocking Ida's path will also provide direct hints to the player to prevent the frequent occurrence of stuck levels.

The most attractive part of Monument Valley is the clever integration of visual deception and spiritual aesthetics into the design of the labyrinth. “People have an innate sense of geometry, patterns and space,” says Wong, the lead game designer of Ustwo Games. “We find cubic forms and staircases and interlocking tiles beautiful. There's also something satisfying and even a bit mystical about shapes lining up perfectly (Stinson, 2013).” Therefore, we can find many geometries and graphics based on different colors of patchwork in the labyrinth design within the game.

At the same time, in order to provide enough visual stimulation for the player, each scene in the game has a new visual twist through interaction with the labyrinth. Although

the first level teaches us how to move Ida by tapping our fingers, the future exploration is about changing the world itself. By tapping certain parts of the game's colorful levels, we can rotate or elevate platforms to open up new paths. For example in one particularly challenging level, there is a columnar creature with a shiny face that could be moved by sliding our fingers. By moving this creature to form new bridges - much like Escher's paintings create a logical way to go without a path - we were able to move on (Farokhmanesh, 2014).

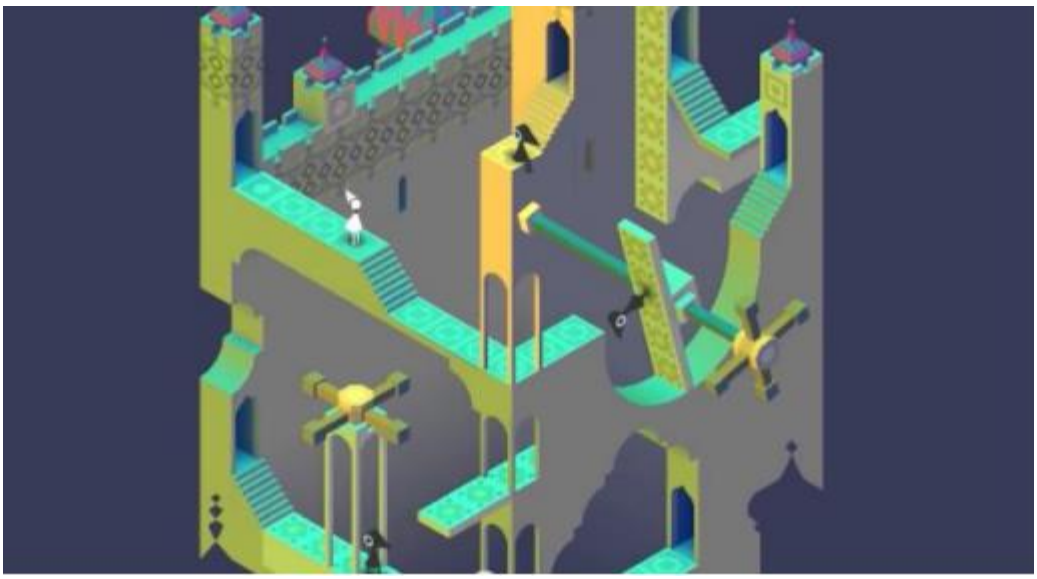

Figure 9 A wide variety of interactive modes in Monument Valley

3 Detailed Design

As an interdisciplinary cooperation project, the detailed design part is mostly achieved by Media Art and Design students. Of course, in order to avoid technical bottlenecks caused by some wild imagination and add some ideas of our own, Computer Science and Technology students also participated in part of the design. To provide a more comprehensive and detailed design documentation thus making the overall design thought of the game more clear and concise, in this section, the detailed design of the entire game will be divided into six parts: overall statement, world view design, narrative design, level design, puzzle design and interactive flowchart.

3.1 Overall Statement

This final year project is to design and implement a video game: Unknot. Unknot as an adventure puzzle game, the whole game can be divided into two main parts: the adventure

part and the puzzle part. The adventure part consists of a main story line and several story branches, the main level is the game process that players must play, and as some of the main levels is finished, some story branches will also be unlocked, providing players the freedom to choose to explore them or continue the main level. The puzzle part is the component of each level, each level is a knotted puzzle, players must complete the knotted puzzle within the level to be able to advance the game progress and unlock the subsequent levels.

In Unknot, the player will play as a spirit with the ability of "shadow seeing" who travels around the inanimate earth. By collecting the diffuse human consciousness, the player unravels the "neural net" (which is the knot puzzle in our game) attached to each key item, so as to obtain the memories of the owner of the item related to this item. In the process, the entire world view, the motive of the genie to find memories, and a touching memory related to the past Earth will be shown to the player.

Figure 10 Design reference, from OPUS: The Day We Find Earth

3.2 World View Design

The future Earth due to rampant infectious diseases, nuclear crisis, war, alien invasion and other factors, the sea level rises, the earth's axis distorted, natural disasters take place frequently, economic disintegration, the Earth's population plummeted to 10% of the early 21st century. Even though it suffered such a major blow, human beings did not become extinct like the dinosaurs and still live strongly on this planet.

In order to strengthen the bonds between existing humans and maintain the fragile human society, post-disaster humans created the "Cyberspace", which linked the consciousness of human beings scattered all over the Earth with the mechanical intelligence existing on each object through digital interconnection technology. People's memories and consciousnesses began to merge and could be shared with each other and gradually

visualized in digital form. People can choose to upload their memories and consciousness to Cyberspace by fusing them with the mechanical intelligence on their personal belongings, or they can choose to add a "Cyberlock" to their private memories that they do not want to share, which is commonly known as a "neural net" among humans. Cyberspace is composed of neural networks, and each neural thread is interconnected to form a vast network of data. It not only carries the current consciousness of people, but also stores the history of the Earth and the traces that each departed life once left on the network. In this way, humans rely on each other in a virtual space, surviving on a planet that has lost its vitality (Lan, 2021).

But eventually, the cosmic pulse from the Milky Way reached Earth, shattering the data structure of Cyberspace, and the human consciousness and memories stored in Cyberspace gradually dissipated with the data, and the connection between humans and others disappeared from then on. The arrival of the cosmic wave shattered most people's lives in an instant, and the survivors were enveloped by the despair of losing their emotional attachment, human society began to become fragile and soon humanity disappeared from Earth...

3.3 Narrative Design: Fishbone Narrative Mode

For the story telling design, we choose fishbone narrative mode to present our story and world view. There is a main story that runs through the entire game, which serves as the player's goal and presents the general worldview in front of the player while driving the player complete the game. At the same time, in each section of the main story there will be one or several side stories, the content of the side story has nothing to do with the main story, but mainly serves to enrich the game content, improve the concepts not mentioned in the main story, and enrich the game world view. Players are free to choose to explore the side story or continue with the main story, such freedom of exploration to ensure that players can choose their preferred way of playing the game at the same time, but also in the process of exploring the side story to enhance the immersion of the game, while complementing the worldview of the main story design.

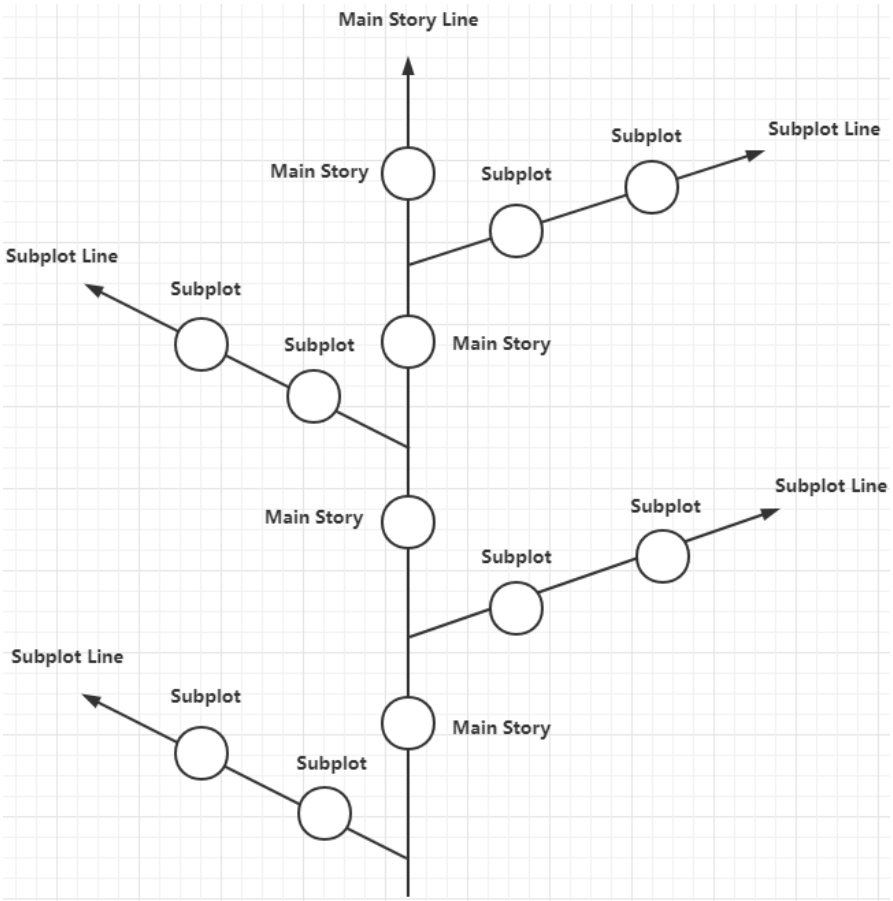

Figure 11 Story telling design reference: fishbone narrative mode

3.4 Level Design: Lighthouse Attraction Point

Cause we have a relatively free story telling mode, so for the level design, We introduce the concept of "Lighthouse Attraction Point" from open-world games into puzzle games, thus enhancing the degree of freedom and explorable elements of the genre. By setting up eye-catching strong attraction objects in the game, the player is guided to plan a linear path through this eye-catching abruptness. The distribution of each attraction point is designed to be balanced, so that the next attraction point can be easily found at one attraction point, as a continuous attraction force invisibly pushing the player to explore (Zhang, 2020). In Unknot, we use lighthouse attraction point to indicate where the player should go next. Key items in the main story will emit different type of light from key items in the side story, thus providing the player with information to decide their next action.

At the same time, in order to encourage players to go deeper into the puzzle, we set a reward mechanism: if the puzzle is perfectly solved, additional side stories will be unlocked, so that players who invest more time can understand the story and world view more

comprehensively, thus giving players enough positive feedback.

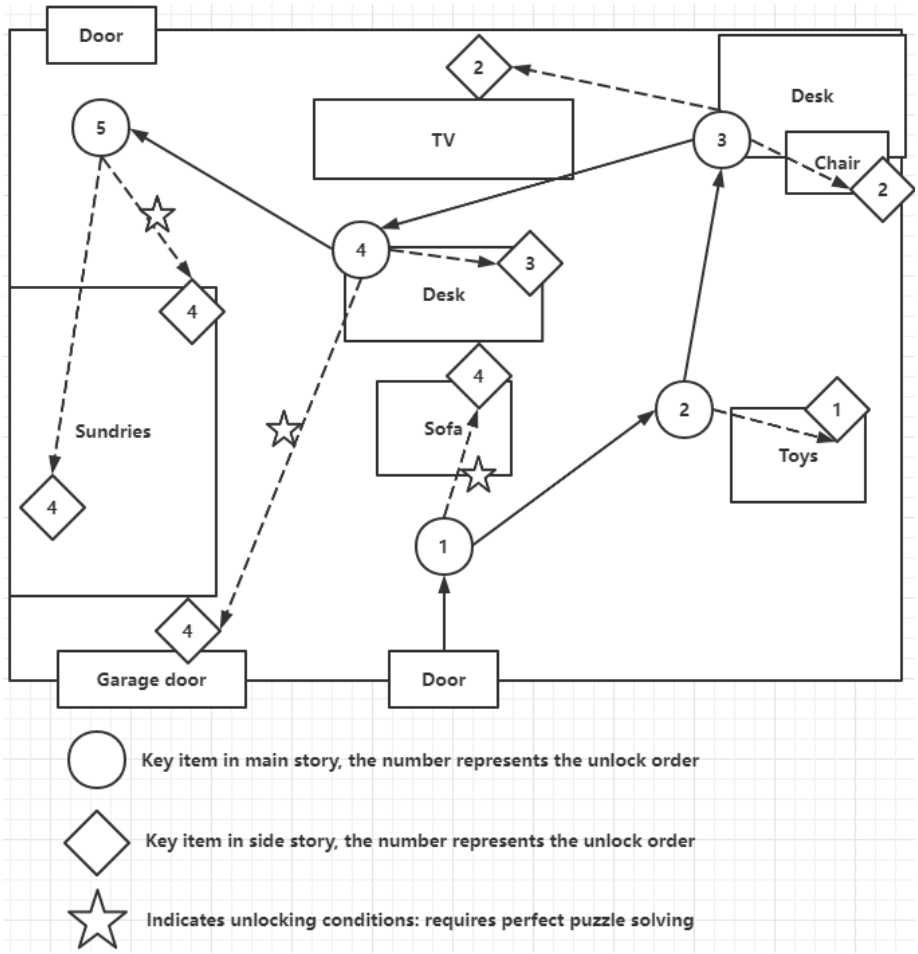

Figure 12 Level design reference

3.5 Puzzle Design: Knot Puzzle

For the puzzle part, we use the knot puzzle. Such form comes from a mathematical puzzle called knot puzzle. We believe that mathematical puzzles are closely associated to puzzles in life that there is no fundamental difference between them. There by we chose the knot puzzle and ported it to the game. The player has to manipulate the knot in dragging, pulling, flipping and so on to turn it into its simplest state that eventually unravels, the final untied state of the string will be shown on the side of the screen. When player try to unknot the puzzle, the camera will show player the knot from a bird's-eye view. Light and shadow will tell the player which piece of rope is on top and which piece of rope is on the bottom. Players can perform several fixed operations.

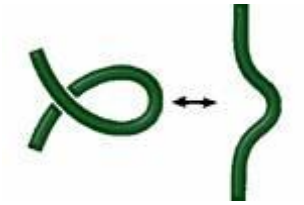

Figure 13 Take out or put in

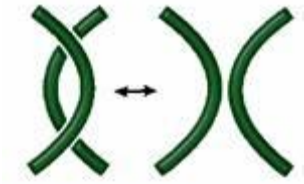

Figure 14 Add or remove two crossings

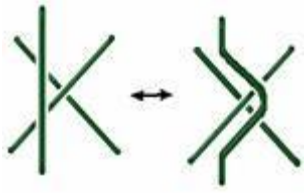

Figure 15 Slid a strand from one side of a crossing to the other

3.6 Interactive Flow Chart

For the interaction flow chart of our project, it can be roughly summarized as a loop of exploration, interaction, puzzle solving, and then exploration. Players enter the level and explore freely in the scene, interact with key items and then enter the scene to solve the puzzle, and then return to the level scene to continue exploring after a successful or failed puzzle. In the following prototype development part, we will also focus on these three aspects: explore, interact and puzzle to elaborate our technical implementation.

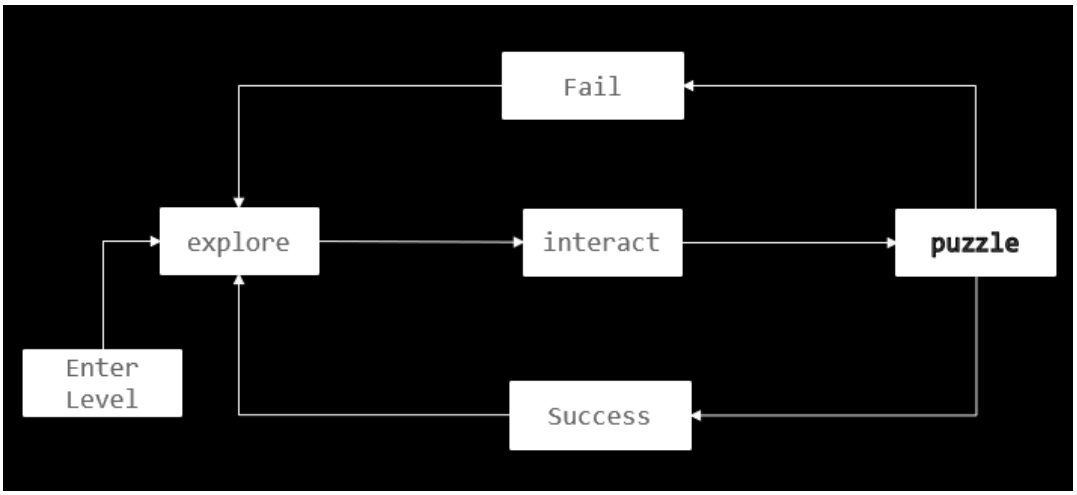

Figure 16 A brief interactive flowchart of this game

For the whole game process, the player, act as a spirit who have a special ability, can explore the scene freely. Player can move around in a small scene, usually a room or an area with restricted size, with many interactable items. Each item will display a text after the player interacts with it. Some items are just normal item, and the text will tell player what item it is and the story related to this item and the owner of this item. But some items, which have special characteristics, are called key item. They will be signed with a corresponding story telling the world view, and the something once happened related to this item. Except from reading the story, part of those key items are designated with knot puzzle. Player will be guided to solve the puzzle one by one to forward the game process. Other key items which have not been signed with knot puzzle will be signed with some collect points called “consciousness”. Reading the story on such kind of item will collect the consciousness. Player can only go to the next key item with knot puzzle when both forward puzzle is solved and the consciousness is collected enough. And when all the knot puzzles are solved, player are able to finish this level.

4 Prototype Development

As an interdisciplinary cooperation project, the prototype development part is mostly achieved by Computer Science students. In this section, the entire development structure of the game prototype will be shown based on the categories in the interaction flowchart mentioned above. The overall structure will be divided into three parts: exploration, interaction and puzzles to show the whole game prototype implementation process, thus making the overall structure of the game more clear and concise.

4.1 Explore

4.1.1 Map

The map of our game is designed and built by the MAD students. The demo scene is a small bedroom with some objects likes desk, bed, vase and so on. According to the world view, this room is a small corner in the huge cyberspace. A human being used to live here and leave some memory and consciousness. Our character enters and is trapped in these small room, which is a fragment floating in the universe, by accidently and try to find out what has happened in these room in order to get out. The human being’s memory is left on

some item in this room. Each memory corresponds to a story, and all the memory in this room build up a fragment of this human being's life. Knowing his life, our character can find the way to get out from here. But there are some private memories, which is locked by their owner with a "Cyberlock", cannot be seen easily. So, our character should come over some difficulties to reach those key memories.

While playing, player can explore freely in the small room by controlling the character and interact with the objects, some of which are signed with main story or side story. While player interact with the objects signed with main story, they can either enter the knot puzzle. By solving the puzzle, player can advance the game process and the plot.

And in order to provide players with a more engaging scene, thus enhancing their gaming experience as well as the sense of immersion, we added a global rendering to the environment. Through these two comparison images we can see that after this layer of rendering, our scenes become more realistic and the color tones are softer and closer to reality.

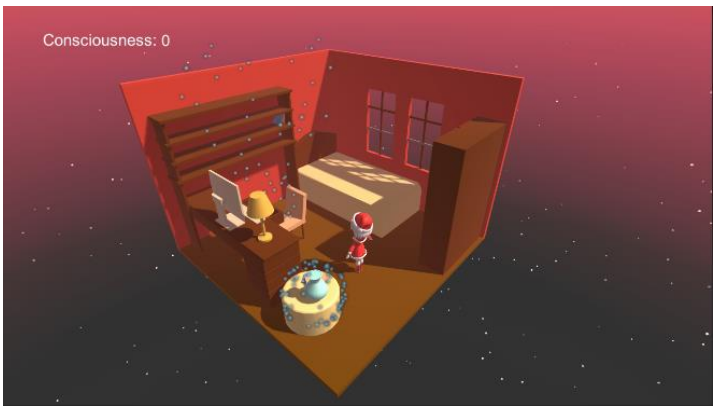

Figure 17 Scene before rendering

Figure 18 Scene after rendering

4.1.2 Character Movement: Navigation, Mouse Manger

As a puzzle adventure game, player adventuring is an important part. Player

adventuring means that player can control the character moving in the scene. So, we should achieve the character movement. We achieve character movement with automatic wayfinding and mouse manager. The navigation system provided by unity allows us to create characters that can navigate through the game world and bake the passage and collision relationships between the various elements on the map and the characters. The Unity Navigation system consists of the following pieces: NavMesh, NavMesh Agent, Off-Mesh Link and NavMesh Obstacle.

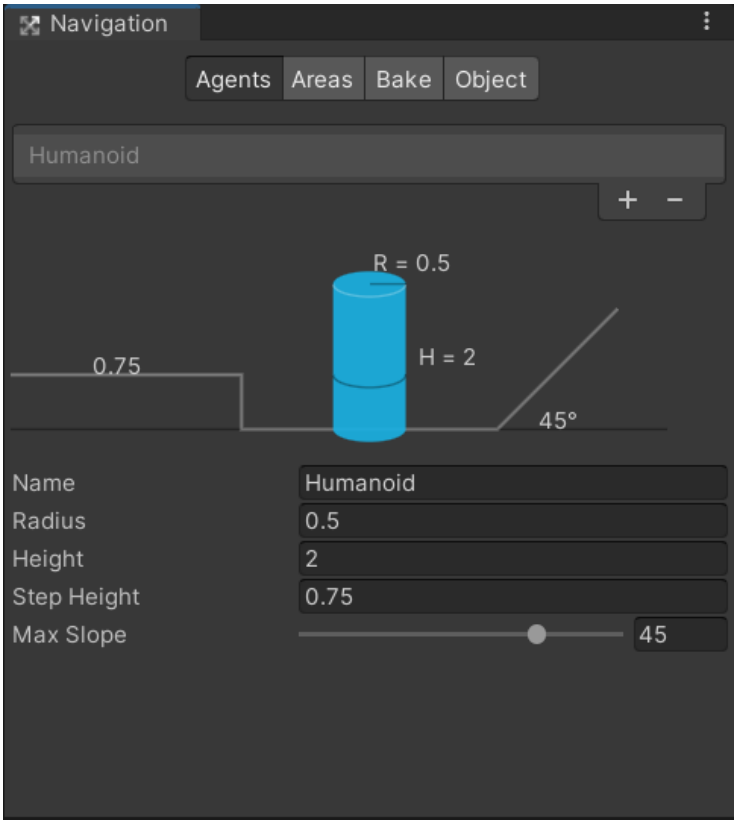

Figure 19 Navigation

NavMesh describes the walkable surfaces of the game world and allows to find path from one walkable location to another in the game world. (Unity, 2022) This tool allow us to mark the place whatever the player can walk on to or walk through, and place in our game like the bed, the desk and the chair where player cannot simply “walkthrough” is not marked.

NavMesh Agent helps us to create characters which avoid each other while moving towards their goal. (Unity, 2022) This tool give character an attribute like the one collider component does, but it is specially designed to combine with this navigation system.

Off-Mesh Link component allows you to incorporate navigation shortcuts which cannot be represented using a walkable surface. (Unity, 2022) When meeting area like

broken bridge, where player cannot reach by walking straightly but can be reached by detouring, this tool will be useful.

NavMesh Obstacle component allows us to describe moving obstacles the agents should avoid while navigating the world. (Unity, 2022) This tool is useable for creating some NPC in the game who may occupy some walkable surface (usually a little). And is like the collision between collider.

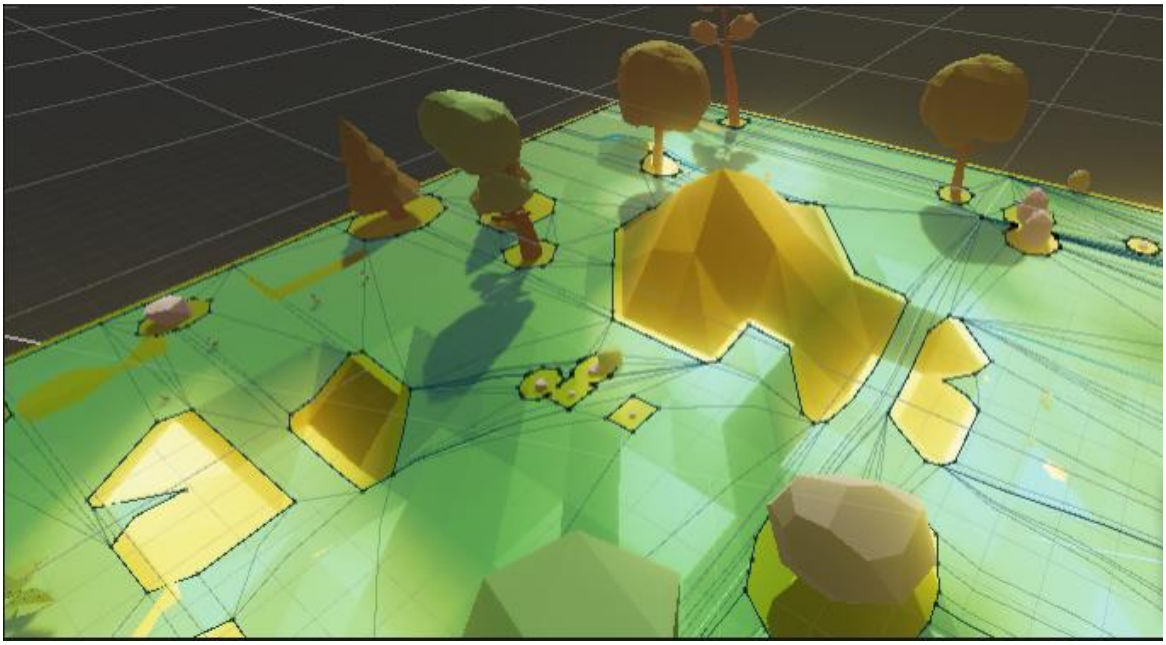

Figure 20 Setting walkable area

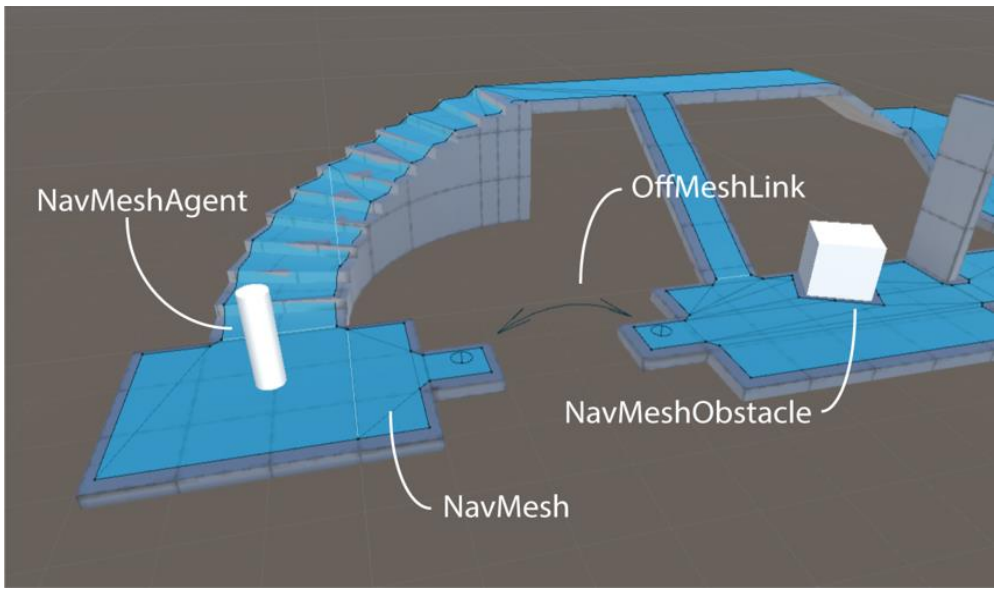

Figure 21 Visual reference of navigation system

And we use a mouse manager to allow player to control the movement of the character by clicking mouse. The mouse manager can change the appearance according to the tag, which is signed by us to the game object during creating them, and get the 3D coordinate

of the point that player click on, which is the intersection point of ray coming out from camera and the object being clicked.

4.1.3 Character Animation

As one of the most important aspects of video games, animation always plays a pivotal role in the impact of interactive media on the player's interactive experience. In order to enhance the interactive experience of the game and to make our characters more vivid and lovely, we added a variety of advanced animations to the characters.

First is the animation blend tree. In games, the game character is sometimes not precisely in a certain state, such as the transition between standing and walking. However, state transitions are a process of smoothly changing from one state to another. For the character movement part. Because the speed of our character movement will change according to the movement state, and different speed corresponds to different movement state, so if we just use the normal state machine to control the animation during the character movement, then the switching of character animation during the movement will become very stiff, which will bring very bad impact to the player's playing experience. in order to achieve such smooth transitions, we introduce an animation technique: blend trees. The blend tree can mix multiple animations together and add interpolation values to the switch between animations, so that the whole animation playback process becomes smoother.

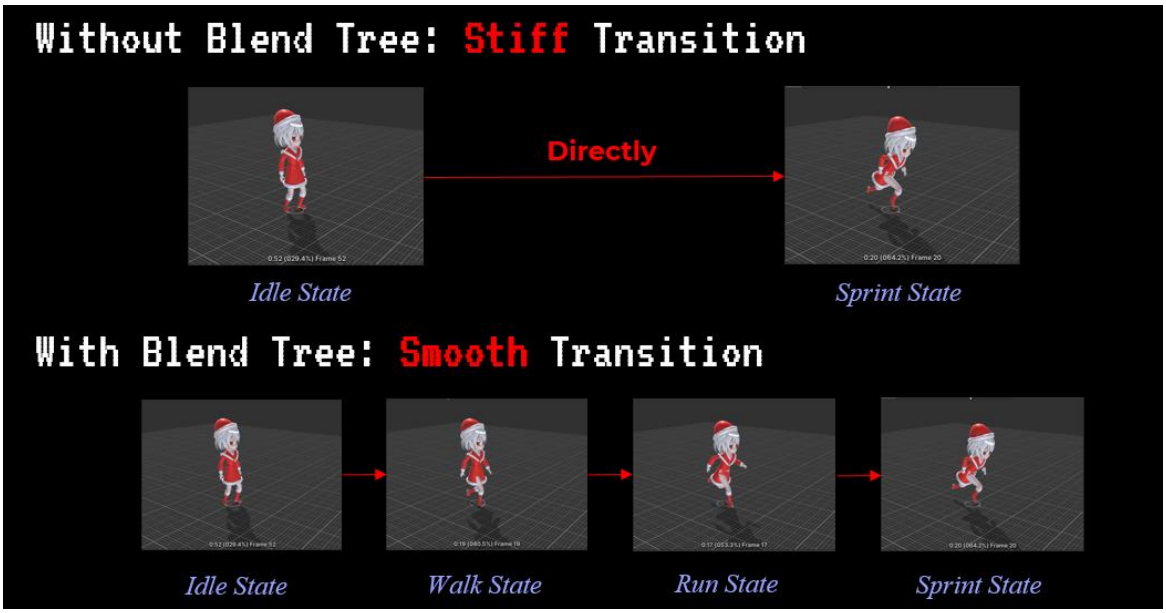

Figure 22 The difference between with and without an animation blend tree

In this project, we first create a blend tree called LocalMotion and put all the animations that will be played during the character's movement into the blend tree. In order to control the state of the animations in the blend tree, then we introduce a variable "Speed"

representing the character's movement speed at that moment. When the speed of the character is 0-10, the character will keep idle, when the speed is 10-20, the blend tree controls the character to enter the walk state, when 20-30, the running animation will be played, when 30-40, play the sprint animation. The overall effect can be approximately seen as the frame filling technology in video processing. By inserting interpolation values in the process of switching between several different animations, the whole process of character animation switching becomes much smoother.

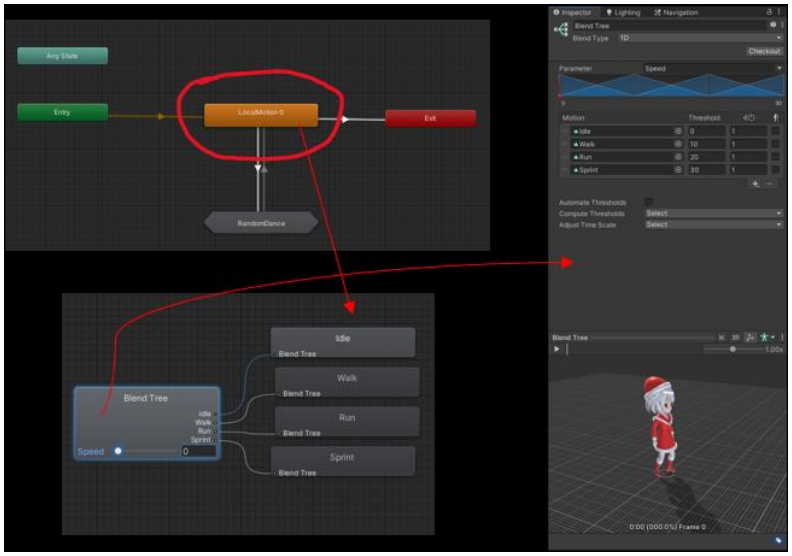

Figure 23 The blend tree in this game project

Additionally, To make our character cuter and more absorbing, we added idle animations to the characters under the idle state. But it's not just ordinary idle animation, it's random idle animation. As players usually see in most RPG games like Genshin Impact or The Legend of Zelda, when the character stays still for a certain amount of time, the character will start playing some idle animation, such as dancing, stretching themselves and so on. The same goes for our character. If the character stays idle for a while, she will start dancing. As for what she dances, it is completely random. Character's dance will always continue, unless the player gives a move command to Kiki, then the dance state will be interrupted and goes to the animation state of character movement, which is also what we have

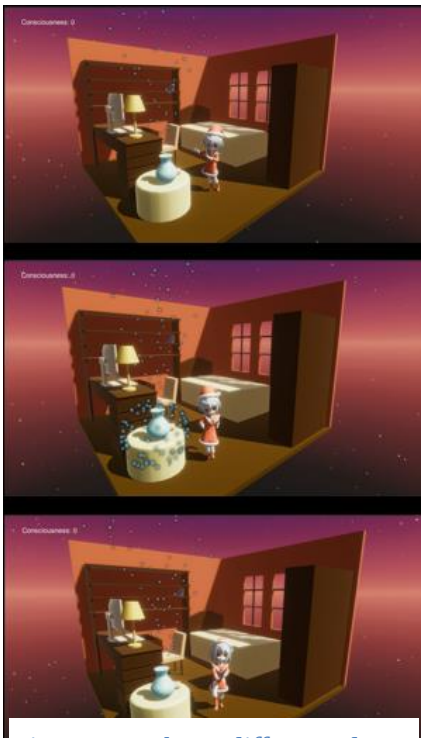

Figure 24 Three different dance states

described above.

To achieve it, we first create a sub-state machine that hosts all our possible dance animations. Also we introduce an int variable "DanceState", which determines the type of the following dance. To achieve the random effect, we added a script for the RandomDance sub-state machine to decide which dance the character in the idle state does, which is determined by the variable “DanceState”. Before the animation playing state entering this sub-state machine, we will give the variable "DanceState" a value between 0 and 3, and pass this variable into the sub-state machine, which will then determine which dance animation the character will play.

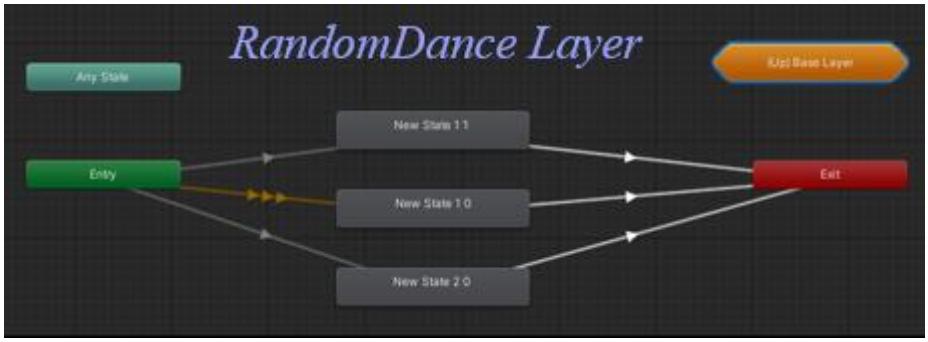

Figure 25 The state machine controls all dance states

4.1.4 Game Process Monitor

When the game is running, we must make sure that all things are in their right state. For example, object A should appear only when we allow it appear and NPC B should tell what we want him to tell. In other words, we need the game processes correctly, by knowing how far the player has gone on the preset game process. So, we need a global control. This global control will record the information like which puzzle has been solved, which object player has already interacted with even what operation player has done and other object in the game will change their state, like the state of some object, if the player can enter a puzzle scene or if the player can enter next level, according to the data stored in this global control unit. And we design to use a game object to be the global control and store the data. Because we put different puzzle in the different scenes, it means that when we switch to a different scene, all the game object in the original scene will be destroyed and some data may be lost. To solve the problem, fortunately, unity provide a method called “Don’t destroy on load”.

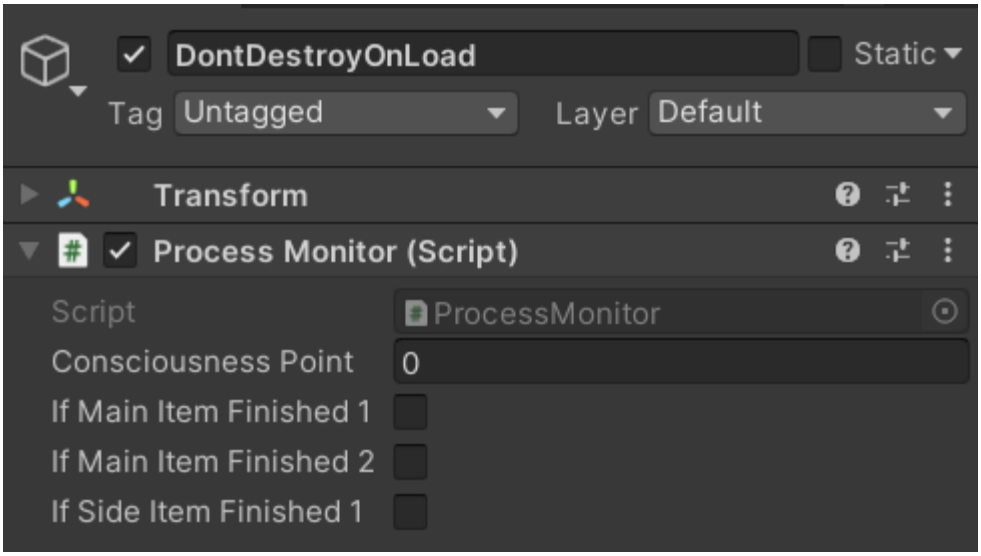

Figure 26 Don't Destroy On Load game object

The game object being set by the "Don't destroy on load" will not be destroyed while unity loading new scene and is very useful for saving global data and player data. But if we use this method directly, there will also be a problem: when we load a new scene, we will find that there is another new game object has been created also call "Don't destroy on load". To solve this problem, we use a simply way combine with another method provided by unity, the function awake. This function is similar to function start, but with a little different. Both of these two functions are called during the game object is being initialized, and both of them will only be called once. But the different is that the start function is called after the initialization and before the update function first called. But the awake function is called as soon as the initialization is finished. Simply speaking, the awake function will be called no matter if the game object is enabled once it has been created. But the start function will only be called after the game object is enabled.

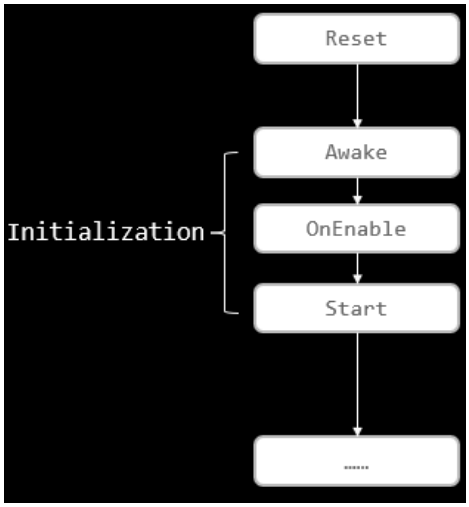

Figure 27 Process of unity calling the function

And in the awake function we let the unity check if such kind of game object is already existed in the system. If it is existed, we immediately destroy the new don't destroy on load game object we just created. And in this way, we avoid creating new "Don't Destroy On Load" game object and make sure that there exist only one "Don't Destroy On Load" game object in the system all the time.

```
private void Awake() {
    if (Instance != null)
    {
        Destroy(gameObject);
    }
    else {
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
}
```

Figure 28 Do not destroy on load

And we set different kinds of variables to mark different stage of game process. We use Boolean value to store the information of if a puzzle has been solved. And we use float value to store how many consciousness the player has collected.

```
public float consciousnessPoint = 0;

public bool ifMainItemFinished1;
public bool ifMainItemFinished2;
//public bool ifMainItemFinished3;
//public bool ifMainItemFinished4;
//public bool ifMainItemFinished5;

public bool ifSideItemFinished1;
//public bool ifSideItemFinished2;
```

Figure 29 Variables to mark game process

Other game objects will check the game process by getting data from this "Don't Destroy On Load" object to decide their own stage. The following picture shows a game object decide if it should be active and show a button while the player finish the first main item and collect more than 20 consciousness.

```
private void OnTriggerEnter(Collider collision)
{
    if (collision.tag == "Player" && other.ifMainItemFinished1 == true && other.consciousnessPoint >= 20)
    {
        Button.SetActive(true);
    }
}
```

Figure 30 Code in a game object Decide

4.2 Interaction

4.2.1 Object Interaction

As we mention before, player can interact with some object in the scene. But there could be so many objects in a scene, if we give no tips to the player, they may wander aimlessly. And this may cause players to be impatient. So, to avoid this problem and to follow the concept of "Lighthouse Attraction Point", we use different particle effects provided by unity to high light the items which signed with main story or side story, in order to attract player to get close to the story line. By providing information to the player through eye-catching strong attractors in the game, the player is guided to decide their next move and plan a linear path through these eye-catching protrusions.

Figure 32 Particle effect on object with main story

Figure 31 Particle effect on object with side story

After the player is attracted by the object with the particle effect, player will get close to the corresponding object. And next is to implement the interaction. To achieve the effect of trigger event, we use OnTriggerEnter function and OnTriggerExit function provided by unity. When two game object that signed with collider component collide with each other, unity will detect it and will call the these OnTriggerEnter function (if the function exists in the script). And when two game objects leave each other after the collision, unity will also detect it and call the OnTriggerExit function (if the function exists in the script). So, if the character getting close to the particular object, the reminder sign will be shown to tell the player that there is a story exist on this object. And when character get far away from the object, the sign will be disabled.

Figure 34 Object with no reminder sign

Figure 33 Object with reminder sign

```
private void OnTriggerEnter(Collider collision)
{
    if (collision.tag == "Player")
    {
        Button.SetActive(true);
    }
}

private void OnTriggerExit(Collider collision)
{
    if (collision.tag == "Player")
    {
        Button.SetActive(false);
    }
}
```

Figure 35 Trigger function

4.2.2 Reading Story

As a puzzle adventure game, the adventure part is the most important part for telling the story and showing the world view, since the information provided by the puzzle part is limited. So, the problem of how to tell the story should not be ignored. In our game, we design the stories to be shown in the form of text which is signed to the corresponding object as we mention before. The story is placed to the designated object in advance. At the beginning, the game object who carry the story is not enabled. And only when the player gets close to the and press the key to trigger the object, the story text will be shown in the following form and in the front of the screen. Player can read the story many times and whenever they want while getting close to the corresponding object even they have already finished the puzzle.

Figure 36 Story text shown

```
void Update()
{
    if (Button.activeSelf && Input.GetKeyDown(KeyCode.R))
    {
        storyText.SetActive(true);
    }
    else if (storyText.activeSelf && Input.GetKeyDown(KeyCode.Escape))
    {
        storyText.SetActive(false);
    }
}
```

Figure 37 Code to implement showing the story text

4.2.3 Consciousness Collection

To meet the design of the fishbone story narrative, we construct the consciousness collection system. Player cannot trigger more main story object and enter the corresponding

knot puzzle unless player collect enough consciousness. When player trigger the object signed with side story and read the side story, the script of this object will change the value of consciousness in the “Don’t Destroy On Load” object. And the “Don’t Destroy On Load” object will store the amount of the consciousness the player has collected. This system makes that the player must read a certain amount of side stories that signed to the object which helps them know better of the world view, so that they will interact with the environment instead of just solving the puzzle one by one.

4.3 Knot Puzzle Solving

4.3.1 Rope Simulation: Obi Rope

To complete the implementation of the rope knot puzzle, how to simulate the shape and physics of the rope is a very critical factor. To simulate a real rope with complete physical effects, we choose Obi Rope, a plug-in developed by Virtual Method. The rope created by ObiRope is made out of small spheres called particles, and particles can interact with each other, affect and be affected by other objects. Oriented particle technology allow Rods to offer torsion/twisting effects. Ropes are more lightweight and can be cut/resized. Bones can be used to add secondary motion to character bone hierarchies. All of them can collide with the environment and with each other, can be attached to rigid bodies and offer two-way coupling with them (VirtualMethod, 2019). It is also because that the rope is made up of particles that it also causes some trouble for us at a later stage, which will also be mentioned later in Topology Knot Puzzle section.

With Obi Rope we can create ropes and rods in a matter of seconds, with complete control over their shape and behavior. It also provides us with a lot of properties and components to simulate the physics of the rope and add various interactions to the rope. For example, Self collisions and Surface-based collisions, which are provided to handle collisions between different ropes.

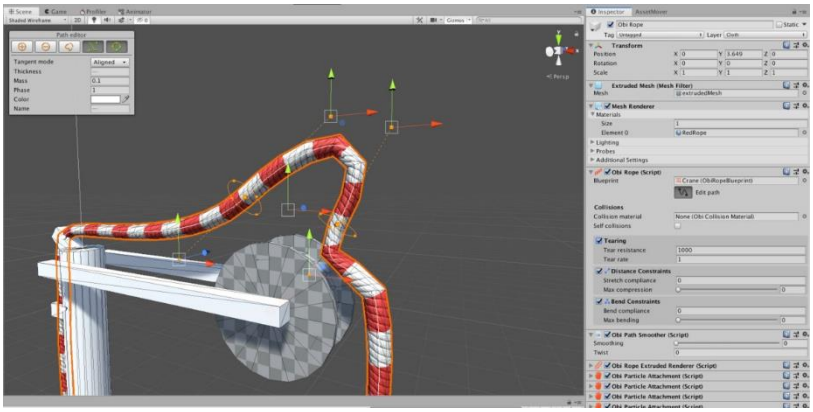

Figure 38 The Rope Created by Obi Rope

4.3.2 Drag Operation

For simulating rope dragging, we use obi particle picker to handle this. With this component, we are able to pick a specific particle that makes up the rope and manipulate it. Then we add a script to handle the dragging operation, and the whole rope dragging function is implemented. In order to make the dragging operation more visible to the player, we also added a line render to render the player's operation into a colored line to enhance the player's interactive experience.

Figure 39 Obi Particle Picker component

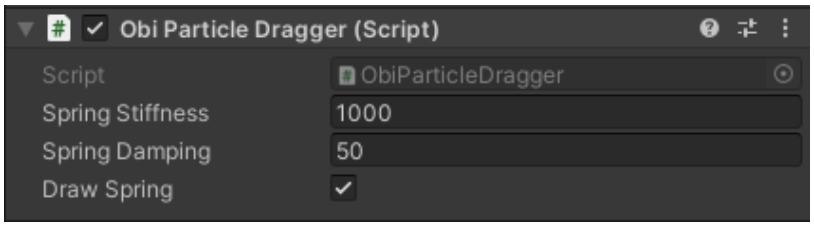

Figure 40 Obi Particle Dragger component

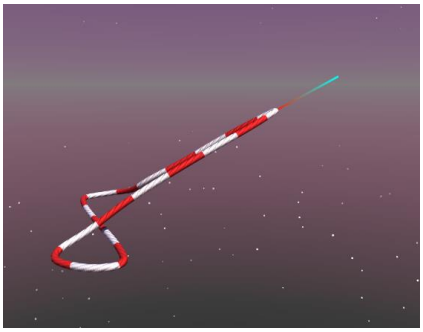

Figure 41 Line render shows while being dragged

4.3.3 Camera Control

In the process of solving puzzles, the freedom to manipulate the camera view is crucial to the player's gaming experience. In most puzzle games that require meticulous manipulation like Moncage and The Witness, free camera view switching is mandatory, because if the viewpoint cannot be moved based on the demand of player, the player will have a blind spot during the process of solving puzzle, which means that the players are very limited in what they can do with the puzzle and will greatly affect player's playing experience. So in our game, we added a camera control script for the camera, which allows the player to freely move the camera on the XYZ axis, rotate the camera and zoom in and out of the field of view.

4.3.4 Topology Knot Puzzle

Topology knot puzzle is one kind of knot puzzle. It consists of a rope and an irregularly shaped geometry, players need to turn on their brains and figure out how to get the rope out of a complex geometry. But if it is simply like this, the puzzle game may be a little bit easy, so In order to increase the difficulty a bit and make players have more fun, we've added some new mechanisms based on this kind of knot puzzle which will be mentioned later.

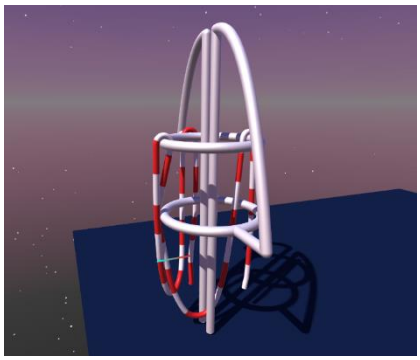

Figure 42 visual reference of topology knot puzzle

Regarding how to solve the problem of determining whether the rope is taken out, there are two factors that can help us to make the determination: collision and judgment zone.

Firstly, if there is no collision between the rope and the geometry, then the rope must been taken out. So we use Obi Rope's own component Wrappable to perform collision detection, Wrappable component is able to be bound to both the rope and the geometry respectively, so that the wrapping relationship between the rope and the object can be detected. By this method , we are able to ensure that there are no collisions between the rope and the geometry. As you can see from the figure below, the Wrappable script is able to perform collision detection, and all geometries that collide with the rope will turn yellow, while those that do not collide will remain green.

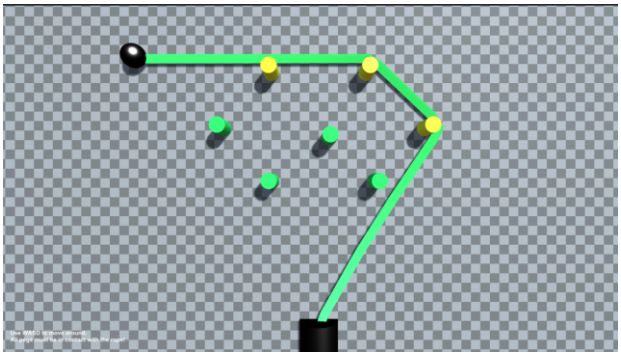

Figure 43 Collision detection by Wrappable component

And, to avoid some unexpected bugs, we added a double insurance for this judgment: the judgment zone. We have added 3 checkpoints to the front, middle and end of the rope. In the process of player operation, all three checkpoints need to be in the range of the decision zone at the same time, in which case the rope will be judged to be solved.

Figure 44 The judgment zone

When all of the above two conditions are met, the puzzle will be solved.

However, in the process of implementation, we have encountered some problems. We already know that the rope is made up of a large number of small particles, which means if you apply a large enough force to the particles which form the knot so that they can be held apart, then gaps must appear between each pair of two consecutive particles. Because of the presence of such gap, the model will directly go through the rope in certain cases, which means a renowned serious bug will appear in our game: model penetration.

Actually Obi Rope offers a solution for us to this situation: surface collision, which ignores the gaps between particles. However, Since our rope will collide and iterate with the geometry many times, using surface collision in this case would greatly consume the computer's performance, since it needs to handle a very large number of collision simulations between particles and geometries simultaneously. As a result, once surface collision is turned on, the game gets stuck, so we have to abandon this approach.

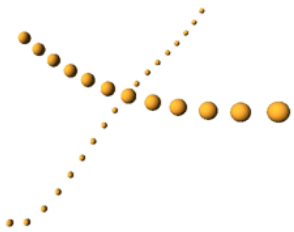

Figure 45 Turn off Surface Collision

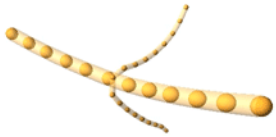

Figure 46 Turn on Surface Collision

If we think about it in another way, since the rope will pass through the geometry model, all the problems will be solved if we let the rope break directly while passing through the geometry. In order to solve the bug of model penetration, we have added a new mechanism to the rope: tearing. When the rope is subjected to a force sufficient to make it pass through the geometry model, the rope will break.

The tear of rope is a feature provided by the plugin itself, when the player applies a certain amount of force to the rope (the magnitude of this amount is self-defined by the developer), the rope will trigger a break. However, this function does not additionally provide additional determination like judging the failure of the game. So in order to make the rope tear the same as a game failure, we need to make some changes to the source code of plugin so that once the rope break is triggered, the game failure will also be determined.

By searching the user manual and API guideline of ObiRope, we found that when the rope breaks, it will trigger the Tear() function in the source code of the Obi Rope plugin. So we find the function called by the plugin when Tear() is triggered, and edit it so that when it is triggered, a game failure UI will pop up, and the only operation the player can do is to confirm that the game has failed and choose to restart or go back, which is the effect that we want to achieve with a failed puzzle solving process.

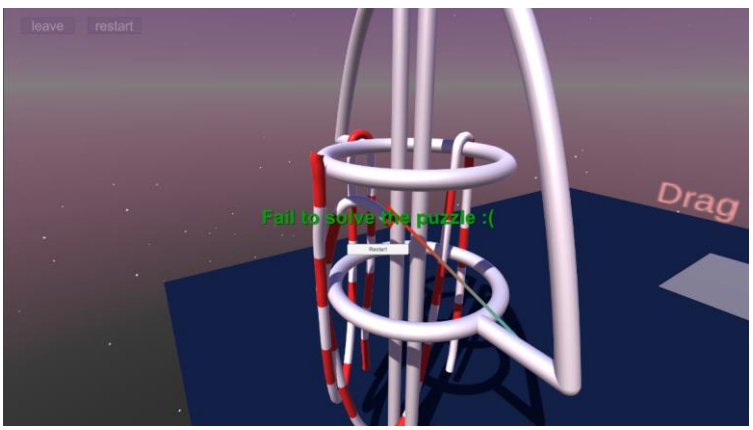

Figure 47 The broken rope and failure UI

4.3.5 Free Knot Puzzle

A knot is a knotted loop of string, except that we think of the string as having no thickness, its cross-section being a single point. The knot is then a closed curve in space that does not intersect itself anywhere. The 2 pictures below show two knots, the first one is an unknotted circle and the second is a trefoil knot.

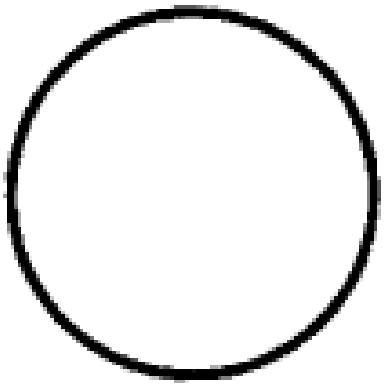

Figure 49 Unknotted circle

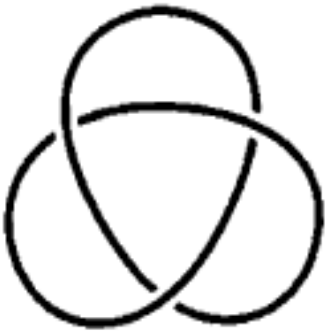

Figure 48 Trefoil Knot

Actually, they are different knots, in another word, we couldn't untangle the trefoil knot into the unknot circle without using scissors and glue, and vice versa. There are numerous kinds of knots. All of those knots are known to be distinct. If we made any one of them out of string, we would not be able to deform it to look like any of the others.

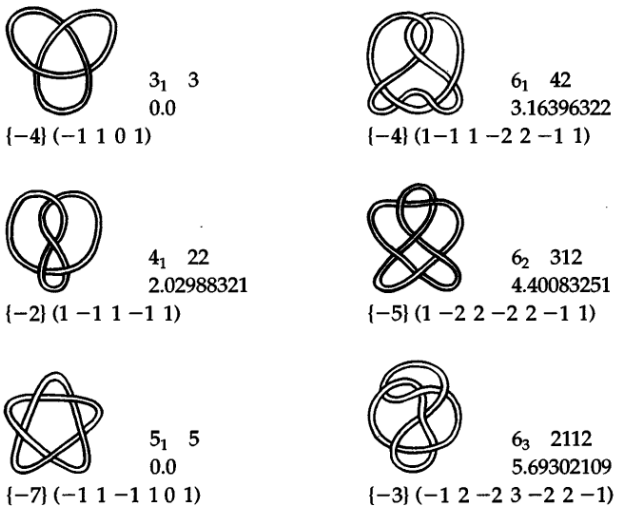

Figure 50 Different kinds of knots

And on the other hand, there are different shapes of the same knot. Anyone of them can deform to other one by certain operations. And we can say they are isomorphisms and that is why we have knot puzzle. The picture below is actually a trefoil knot, even though it looks completely different from the previous of a trefoil knot.

Figure 51 A nonstandard picture of Trefoil Knot

And the places where the knot crosses itself in those pictures are called the crossings of the projection. Take the example of trefoil knot, we say that the trefoil knot is a three-crossing knot. Because there is a projection of the trefoil knot with three crossings, and there are no projections of it with fewer than three crossing.

To solve an unknot knot, we need some operation on the knot. The picture below shows the steps to solve a simple knot puzzle. The left most one is an isomorphism of the unknot circle with one crossing point. Since its simplest projection has no crossing point, the knot in the left most picture hasn't been solved. And if we drag the red point in middle picture on the knot and pull up, it will finally come to the state in the right most picture which has no crossing point. And the puzzle has been solved.

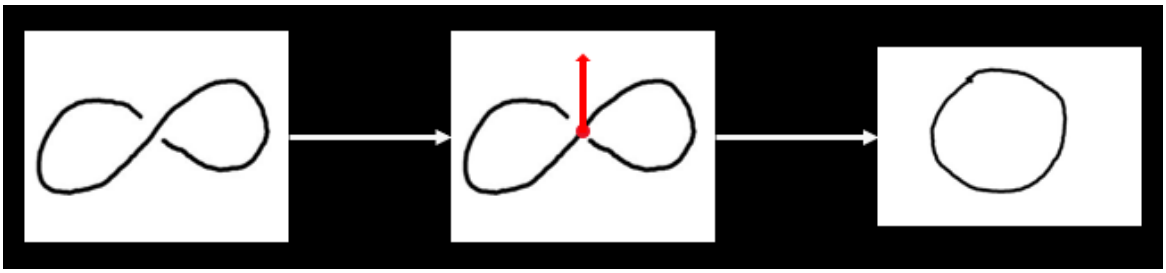

Figure 52 Step to solve a knot puzzle

The free knot puzzle is the most normal kind of puzzle. In our game, we place such kind of puzzle in 3D space and player can operate the rope by clicking and dragging their mouse. If we look with our eyes and think with our brain, we can easily determine that weather a knot has been solved. But it is difficult to let Unity know if the knot has been solved.

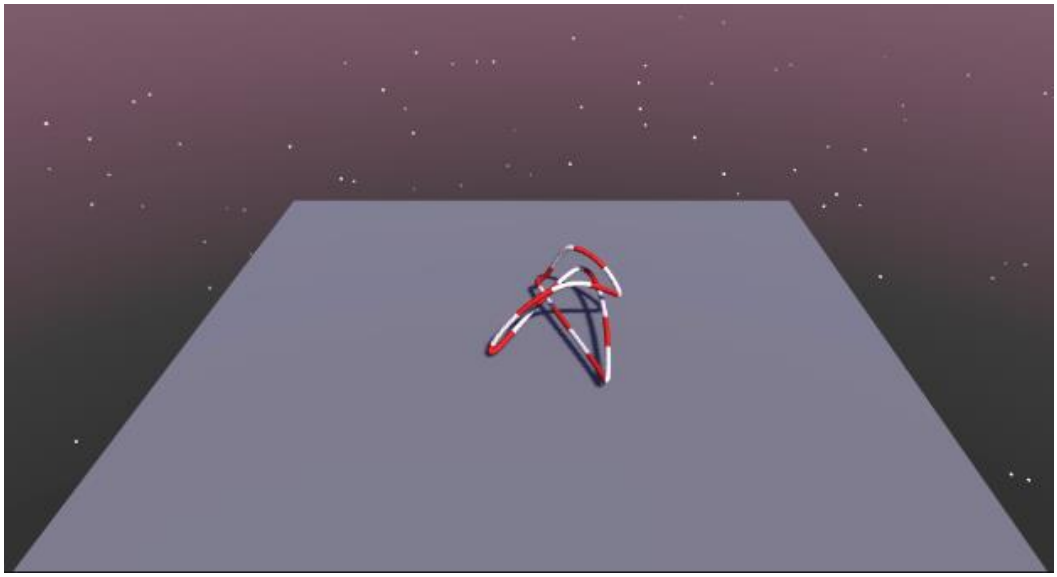

Figure 53 Puzzle game screen

So, the following parts we will talk about how we implement the detection system.

At the very beginning, we have our initial thought on how to detect the knot if solved: we divide the rope into serval continues segments, and we check the amount of the segments that cross with others. If the amount of the cross points is the same as the simplest projection's, we say that this knot has been solved.

To achieve such thought, firstly, we use a component provide by obi rope, which is called obi rope attach. It is a script component and it can help us simply locate the points on the rope, as we change the value of this M variable, which represent the position on the rope from start to end. When the M equal to 0, the point is on the start point of the rope. And when M equal to 1, the point is on the end point of the rope. And for the close loop rope, value 1 and value 0 are at the same place.

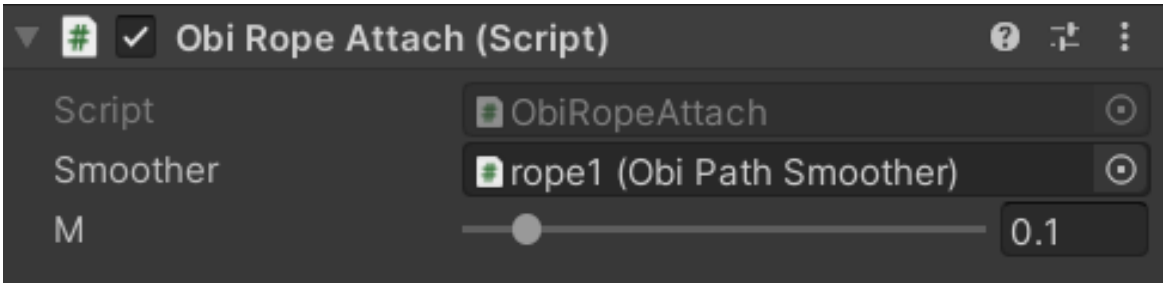

Figure 54 Obi Rope Attach component

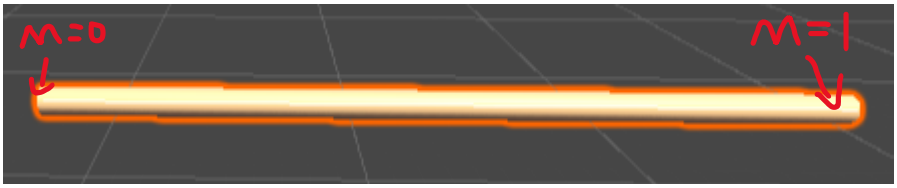

Figure 55 Locate the point on rope

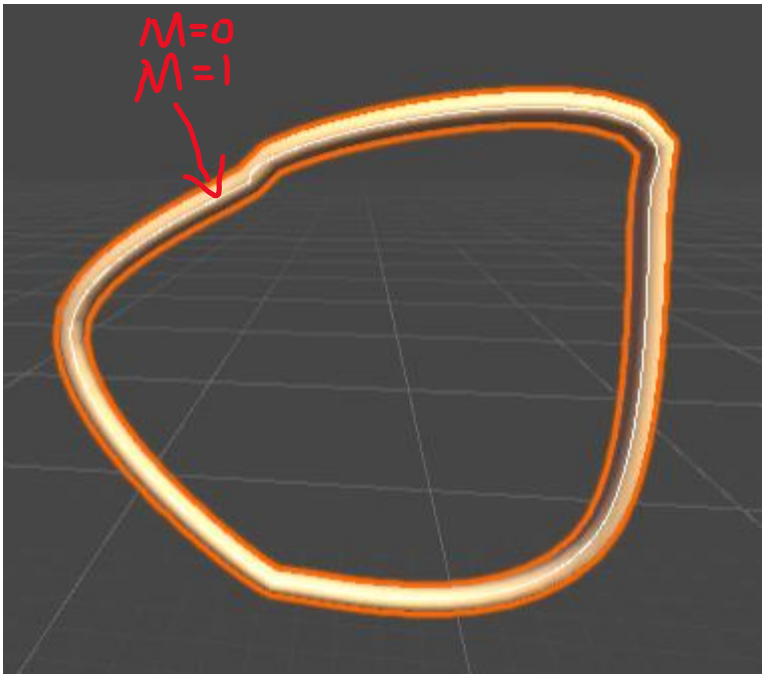

Figure 56 Locate the point on close loop rope

We sign this component to different blank game object to make sure we can get the coordinate value of the point. This type of game object can have any number according to the length and requirements of the rope. And what we need to do is just divide 0 to 1 equally by the number of points and sign to variable M.

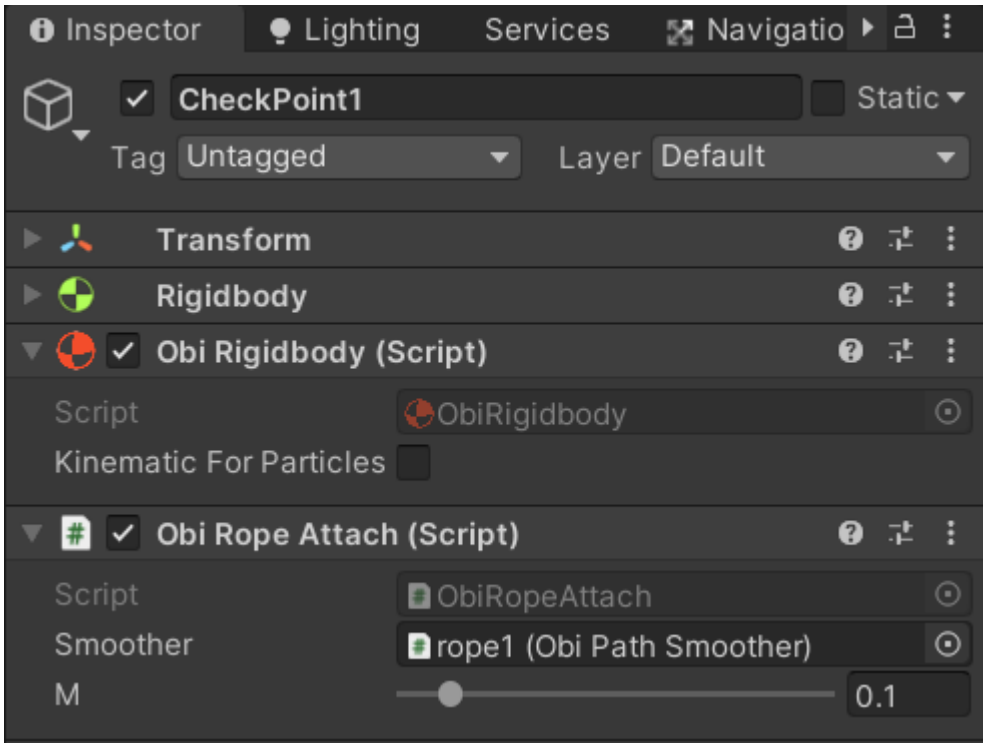

Figure 57 Blank game object signed with Obi Rope Attach

With the point upon, we can easily construct the line segment virtually, selecting two consecutive points. And we use a structure to store the information of a segment. The line sign integer is used to avoid two consecutive line segments being detected, since there is a common point and will be treat as crossing.

```
public struct Line
{
    public float p1_x;
    public float p1_y;

    public float p2_x;
    public float p2_y;

    public int lineSign;

    public Line(float[,] p1_x, float[,] p1_y, float[,] p2_x, float[,] p2_y, int lineSign)
    {
        this.p1_x = p1_x[0, 0];
        this.p1_y = p1_y[0, 1];

        this.p2_x = p2_x[0, 0];
        this.p2_y = p2_y[0, 1];

        this.lineSign = lineSign;
    }

}
```

Figure 58 Structure to store line segment information

Since the rope is placed in 3D space, it will be complicated while we checking the crossing point. So, we just simply throw away one coordinate axis and use the projection of the rope to do the detection. And we give a gravity to the rope to let it drop on a plane in order to make sure the coordinates in the vertical direction will not affect the detection.

For the detection part, we use the following function:

$$Given\ 2\ line\ segments\ AB\ and\ CD,$$

$$we\ have\ AB = A + r(B - A)\ and\ r \in [0,1], CD = C + s(D - C)\ and\ s[0,1]$$

$$A\ and\ B\ are\ the\ endpoints\ for\ AB, C\ and\ D\ are\ the\ endpoints\ for\ CD,$$

$$the\ value\ of\ r\ and\ s\ repersent\ the\ position\ on\ between\ AB\ and\ CD$$

If AB and CD have crossing point, we have function:

$$A + r(B - A) = C + s(D - C)$$

Represent in coordinate form is:

$$x_A + r(x_B - x_A) = x_C + s(x_D - x_C)$$

$$y_A + r(y_B - y_A) = y_C + s(y_D - y_C)$$

Transform:

$$r = \frac{(y_A - y_C)(x_D - x_C) - (x_A - x_C)(y_D - y_C)}{(x_B - x_A)(y_D - y_C) - (y_B - y_A)(x_D - x_C)}$$

$$s = \frac{(y_A - y_C)(x_B - x_A) - (x_A - x_C)(y_B - y_A)}{(x_B - x_A)(y_D - y_C) - (y_B - y_A)(x_D - x_C)}$$

If the value of r and s are both between 0 and 1, which means that the crossing point is on line segment AB and CD.

And then we use the thought of divide and conquer to optimize the performance. Firstly, we store all the point into a set. And the way we divide the problem is that we draw a line to divide all the point into two parts. This line is determined in this way: we calculate the average value of all the point in the current set to get the coordinates of the first point. And the second point is decided by the current number of recursions, make vertical lines to the x axis and y axis respectively and take the intersection point with the coordinate axis. If this time makes vertical line to the x axis, next time will make the vertical line to the y axis. According to the position of two points on each line segment, we divide the line segment into 3 parts: the whole line segment is on the left side of the divide line, the whole line segment is on the right side of the divide line and the line segment across the divide line. For the left-side part combined with the across part, the right-side part combined with across part, the across part, each of those part we put into a set and do the recursion. And when the number of the line in the set is less than 3, we reach the base case and will use the upon check function to check if there are any crossing points. If there exist a crossing point, we will return and add 1 to the return value. And for the final return value, we check the return number. If the value meets the number of crossing points in the simplest state of current knot, we tell the system that this knot has been solved. Otherwise, the knot is treated unsolved.

```
base case
according to the divide line,
divide the line into 3 parts

recursion(left + mid)
recursion(right + mid)
recursion(mid)
```

Figure 59 Pseudocode of the detection function

5 Conclusion and Future Work

In general, the above sections introduce the preliminary shape of the whole game from four aspects: introduction, existing puzzle games, detailed design and prototype development. The introduction section introduces our game type, where the inspiration for the game comes from and where our innovation lies; existing puzzle games section introduces the outstanding puzzle games available on the market now, analyzing their merits and what aspects we can learn from them; detailed design section introduces the detailed game design of our game from 6 aspects comprehensively to provide a more comprehensive and detailed design documentation thus making the overall design thought of the game more clear and concise; prototype development section divides the entire game process into three major modules, and explains in detail the role of each module function and the process of implementation to make the overall structure of the game more clear and concise.

However, the whole game has only completed the technical validation part: the physics simulation of the rope, the algorithm of the knot puzzle solving and the running of the game process flow are already fully proven, while the detailed game flow and story line are not added to this current prototype, and this part is closely related to our partners: the students of Media Art and Design.

So in the next semester, we will interface with the MAD students to get their specific process and story design, then construct the whole game process based on their design documentation and the proven technique we already have, take our game from a technical validation prototype to its ultimate and fully fledged form. And at the end of next semester, we expect to exhibit our game at MAD's final year project exhibition to give the students of UIC an opportunity to experience our final product.

6 References

Adams, C. C. (1994). The Knot Book. In C. C. Adams, *The Knot Book* (p. 2). American Mathematical Society.

Cabebe, J. (2013). *Candy Crush Saga for Android review: Great alternative to Bejeweled.*

Douglas, J. (2013, 10 28). Journey Review. *Game Spot*.

Farokhmanesh, M. (2014, 1 1). *Mounment Valley and the art of visual trickery.* Retrieved from Polygon.

Hellman, D. (2008). *"Braid"*. (Davidhellman.ne) Retrieved from https://web.archive.org/web/20080329043805/http:/www.davidhellman.net/braid.htm

Introduction to Knots - Knot Theory. (n.d.). Retrieved from Oglethorpe: http://sites.oglethorpe.edu/knottheory/introduction/

Kollar, P. (2014, 3 18). *Monument Valley: the quest for a game everyone can finish.* Retrieved from Polygon.

Lan, J. (2021). 走出人类世：人文主义的终结和后人类的降临. *内蒙古社会科学(01)*, 35-43+2.

Long, N. (2014). Wgake Trauk studio Ustwo's next game revealed: Mounment Valley. *Edge*.

Miller, M. (2012, 6 4). Journey Review: Beauty Trumps Complexity. *Game Informer*.

Puzzle - Wikipedia. (2022, 6 3). Retrieved from Wikipedia: http://en.volupedia.org/wiki/Puzzle

Rignall, J. (2016, June 13). *"Xbox One Inside is a Superlative Platform Puzzler"*. Retrieved from US Gamer: https://www.usgamer.net/articles/inside-xbox-one-preview

Rohrer, J. (2007, February 7). *"Exclusive Preview: Braid"*. Retrieved from Arthouse Games: http://www.northcountrynotes.org/jason-rohrer/arthouseGames/seedBlogs.php?action=display_post&post_id=jcr13_1170707395_0&show_author=1&show_date=1

Staff, E. (2014). Monument Calley review. *edge*.

Steven, R. (2014, 1 2). Mounment Valley: a beautiful new app from ustwo. *Creative Review*.

Stinson, L. (2013, December 13). *"This Might Be the Most Beautiful iPad Game of 2014"*. Retrieved from Wired: https://www.wired.com/2013/12/monument-valley-a-

gorgeous-game-thats-like-an-m-c-escher-come-to-life/

Stubsib, L. (2014, 1 2). This Might Be the Most Beautiful iPad Game of 2014. *Wired*.

Tamburro, P. (2016, July 6). *"Inside's Ending and Why it Puts Other Video Games to Shame"*. Retrieved from CraveOnline: https://web.archive.org/web/20180316232949/http://www.craveonline.com/entertainment/1006325-insides-ending-puts-video-games-shame

Unity. (2022). *Navigation System in Unity.* Retrieved from Unity Documentation: https://docs.unity3d.com/Manual/nav-NavigationSystem.html

VirtualMethod. (2019). *Obi*. Retrieved from Virtual Method Studio: http://obi.virtualmethodstudio.com/

W., T. (2014, 1 2). *Trailer: ustwo's Monument Valley.* Retrieved from IndieGames.

Zhang, R. (2020). 探索自由的边界：开放世界游戏的控制与认同. *科技传播(15)*, 114-118.