Aquarium Nightmare

3D Narrative Puzzle Game
Made with Unreal Engine

Abstract

In this immersive first-person room escape game, player will get chance to explore an abandoned aquarium. Through this journey, players uncover their own forgotten past as a former worker in the aquarium, and with each discovery, they gain a deeper understanding of the environmental contamination caused by humanity's destructive actions. The game's setting reflects a world scarred by warfare and pollution, featuring contaminated water tanks, remnants of diary entries, and devastated scenes that vividly depict the aquarium's struggle against the consequences of toxic chemicals, harmful algae blooms, and plastic waste in an once-vibrant ocean ecosystem.

Team

I was responsible for coding, designing, and making 3D models. My team member Junting Zhu worked on the art, and Peilin He worked on designing and making videos.

Inspiration Board

What Remains of Edith Finch

Escape Simulator

Inception

Night at the Museum

Cube Escape: Theater

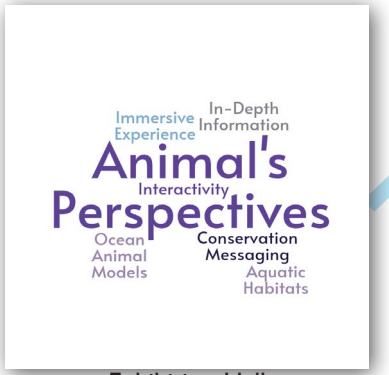
News & Photos
Recently, there have been frequent news reports about ocean pollution, and people often express their lamentations about the loss of clear seawater.

Prototype

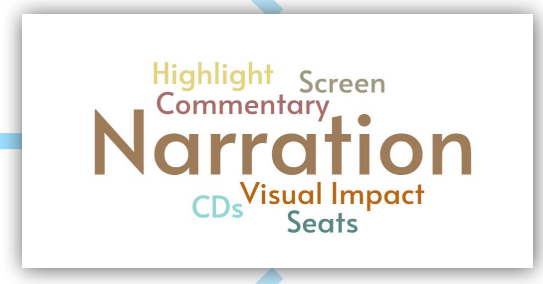
In our vision, the entire story will be told from three main perspectives: the general public, experts, marine animals. Accordingly, we placed these perspectives in corresponding rooms:

Lobby

Laboratory



Exhibition Hall



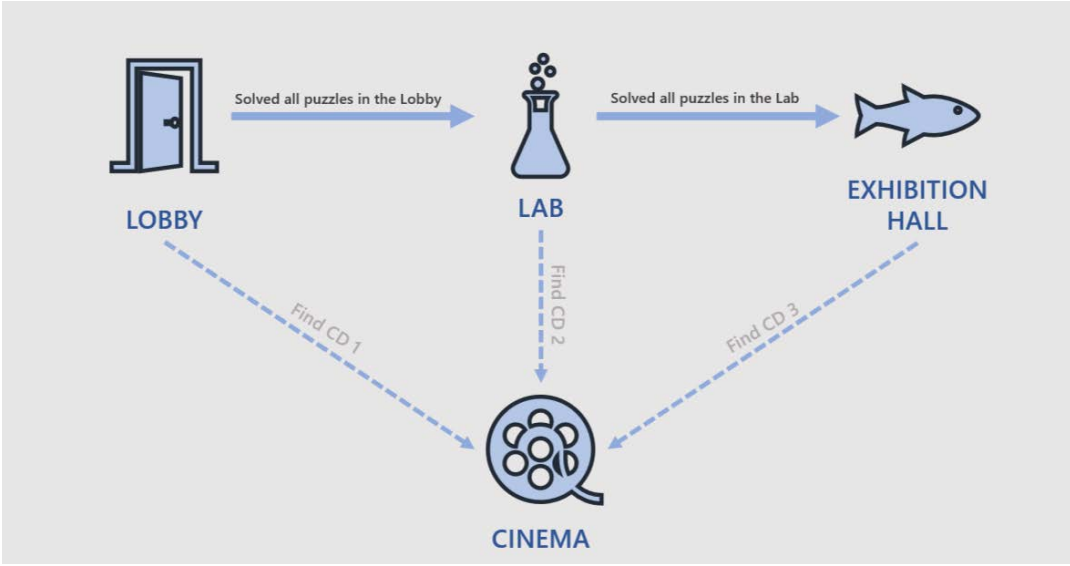
To highlight the corresponding perspectives in each room, we designed a cinema where players receive a CD after completing each room, which plays a corresponding news report based on the perspective of the completed room.

Representation of Ocean Pollution

The trickiest part of an escape room game is how to connect multiple rooms in a logical way. I decided to use a water tank, which represent the ocean condition. As the game progresses, the overall atmosphere gradually becomes eerie, and the large water tank on the first floor becomes increasingly murky and dark, symbolizing the worsening of ocean pollution over time.

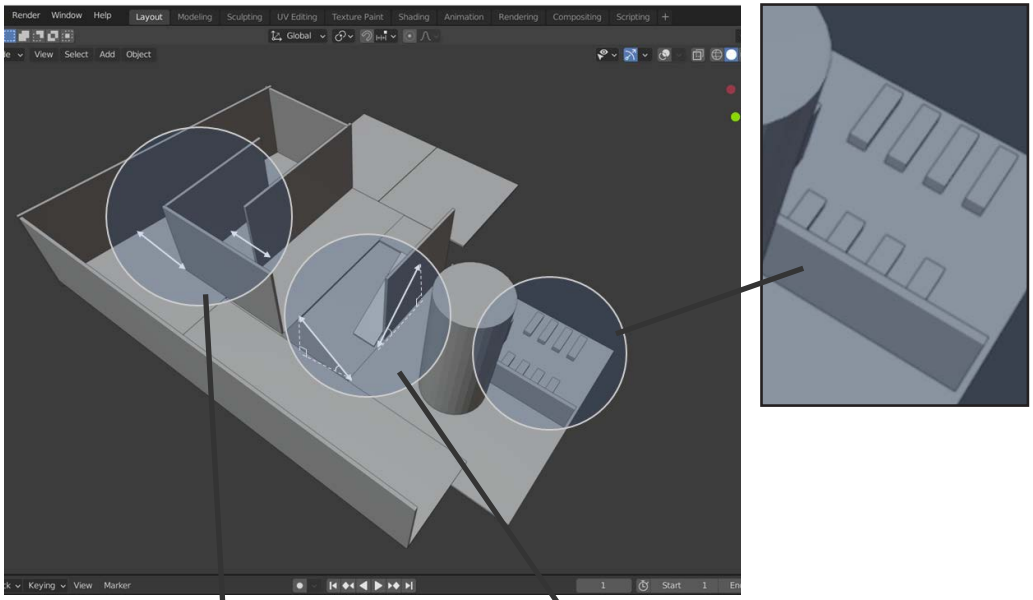
(Ocean Pollution Simulated by Blender)

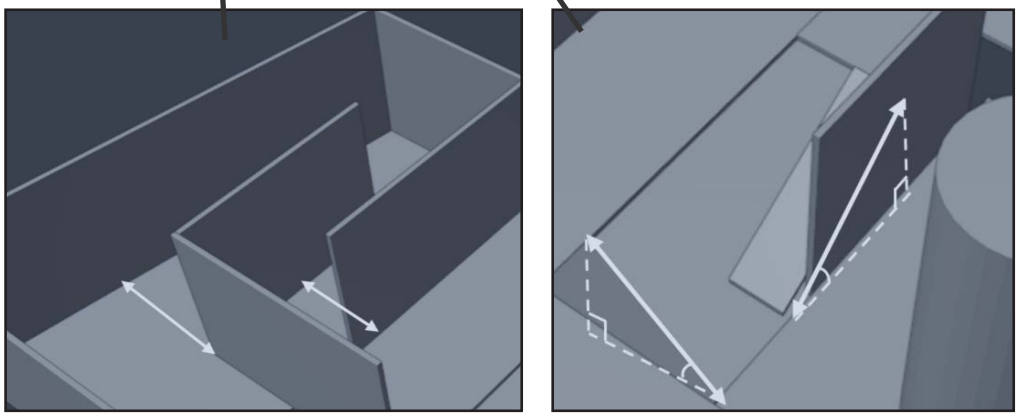
Abstract Game Flow



To allow players to observe the changes in the tank, it should be visible both at the beginning and end of the game. As the cinema serves as a thematic tool for each room, it needs to be easily accessible—players should be able to reach it quickly and conveniently after unlocking each room, with minimal detours. Therefore, we need to position the cinema in a relatively central location.

We didn't want to design the rooms as a single-level flat plane, as that would make the layout feel empty and monotonous. I created a simple layout model using Blender. In this model, a cylindrical tank, serving as a visual connection between the beginning and end, spans two floors and is located at the center of the entire aquarium.

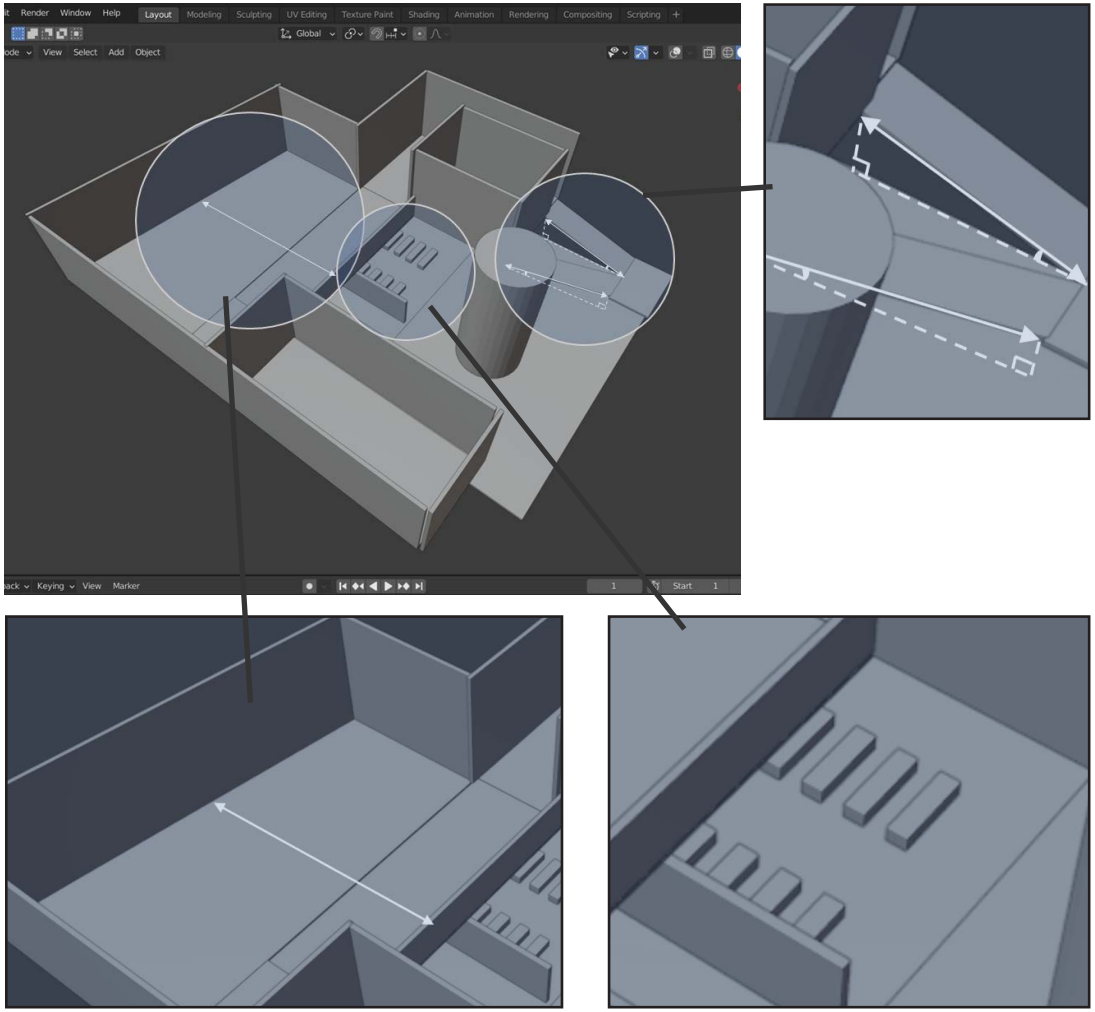


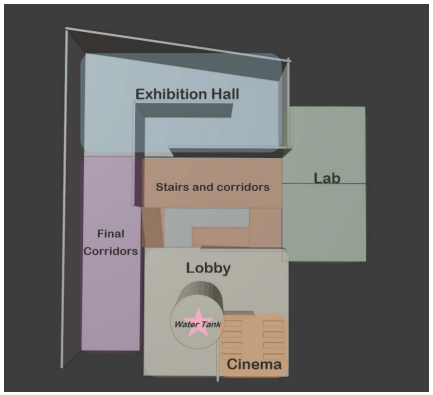


Therefore, I implemented this design in UE and attempted to play through it, but encountered some issues.

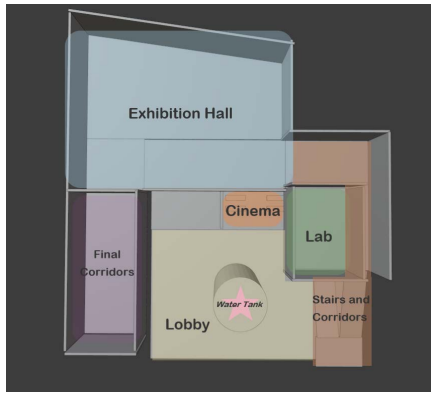
1. The stairs were too steep, resulting in excessively long stairs
2. If we kept both sets of stairs as shown, players might enter the exhibition hall while ignoring the laboratory
3. Furthermore, placing the cinema in this location made it challenging to layout the main hall and did not achieve the desired central position we envisioned.

After making modifications, we decided to keep only one set of stairs and relocated both the stairs and the cinema. We also removed the partition walls and altered the shape of the rooms to create a more flexible layout.



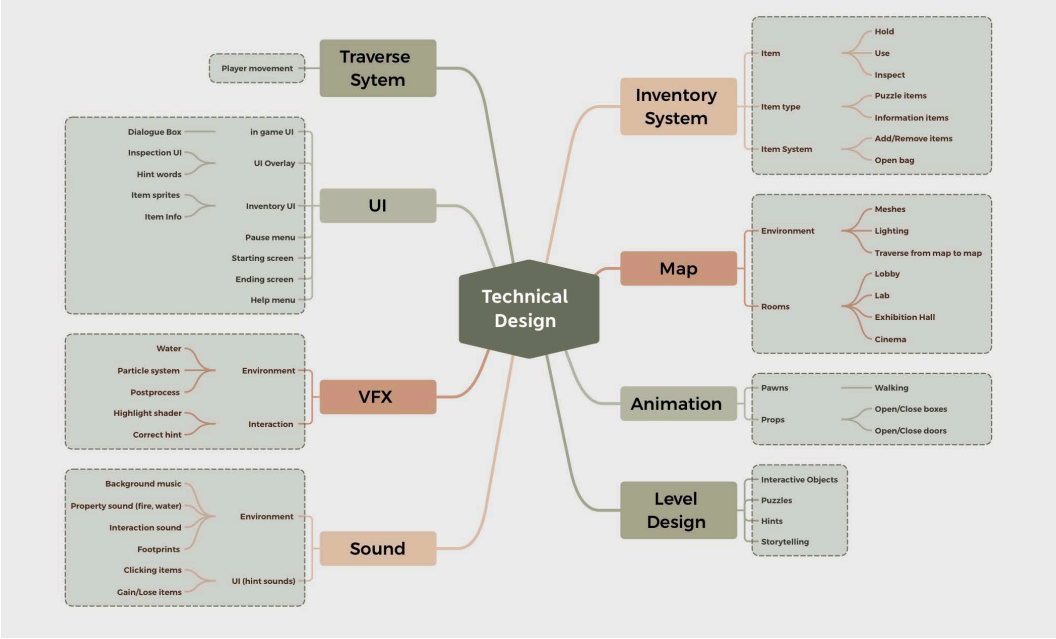


Floorplan After Modification

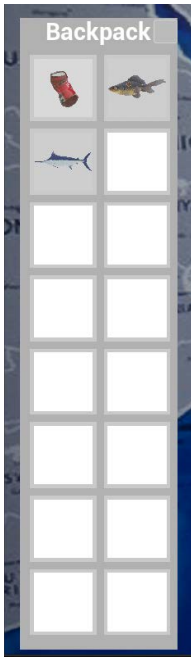


Development

According to the functions, we separate work into 8 main features and assign subtasks with priority, difficulty, and risk.



Inventory



I created the interaction system of this game, which
primarily involves two functions: holding items and inspecting items.

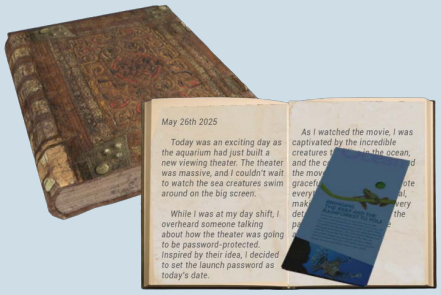
Hold: Right click the item sprite in the backpack to hold item on right hand.

Inspect: Left click the item sprite in the backpack to inspect the infomation of the item.

Items

I created and categorized game items into three main types:
1. Clue items: The clue items include a diary book and diary pages.
2. Puzzle-solving items: These items serve as the "keys" to some puzzles, such as the lab ID card and brochure.
3. Consumable items: They provide certain effects when consumed.

1.



Journal

"Explore the mesmerizing world of underwater creatures with our aquarium brochure and learn fun facts about marine life."

2.

Brochure

"Explore the mesmerizing world of underwater creatures with our aquarium brochure and learn fun facts about marine life."

Lab ID Card

"Unlock the secrets of the aquarium's discarded lab with our ID card"

3.

Mutated Fish

"A mutated fish, with no difference in taste from an ordinary dead fish. How long has it been since I last had the chance to taste a fresh and lively fish?"

Dead Fish

" The pangs of hunger twist and turn within me, threatening to consume my very being. But alas, all I have is this lifeless fish, staring back at me with empty eyes. Perhaps if I close my own eyes and take a bite, it will taste like the finest of delicacies, fit for a king!"

Polluted Water

" Are you sure you haven't drank it

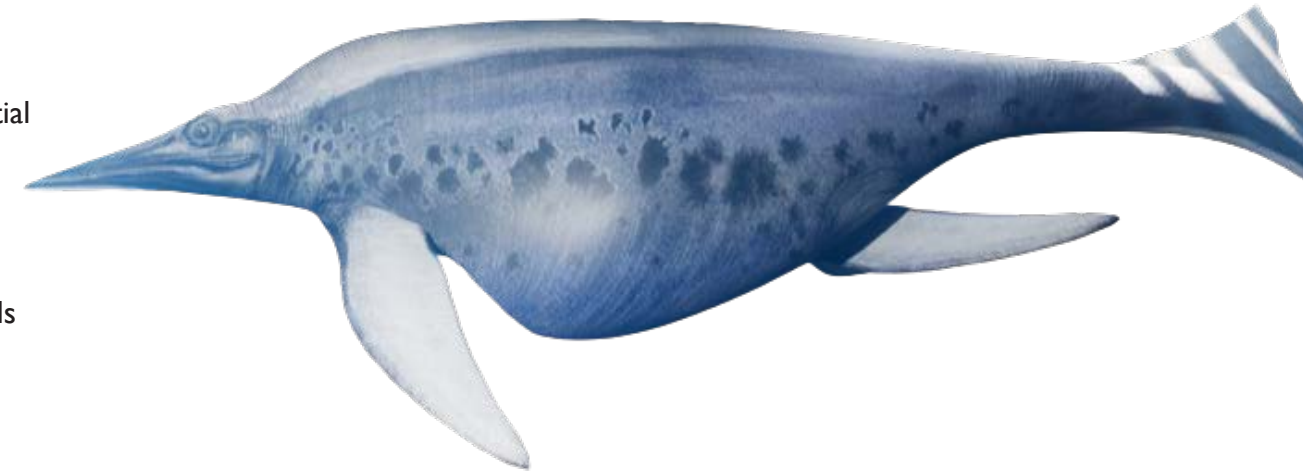
Trash

" The ocean's trash is endless."

Hints

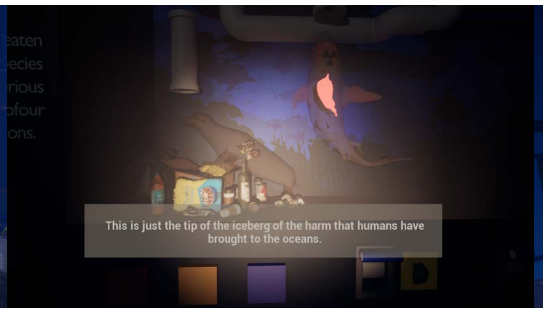
As a puzzle-solving game, players may get stuck during gameplay due to difficulty in finding clues or knowing how to use items. It is essential for a good escape room game to have a well-designed hint system that provides players with helpful hints without disrupting the game environment.

Although our game takes place within the player's dream, where everything can be absurd yet logical, we still strive to design the levels with a sense of coherence. In this game, I employed 3 types of hint systems:



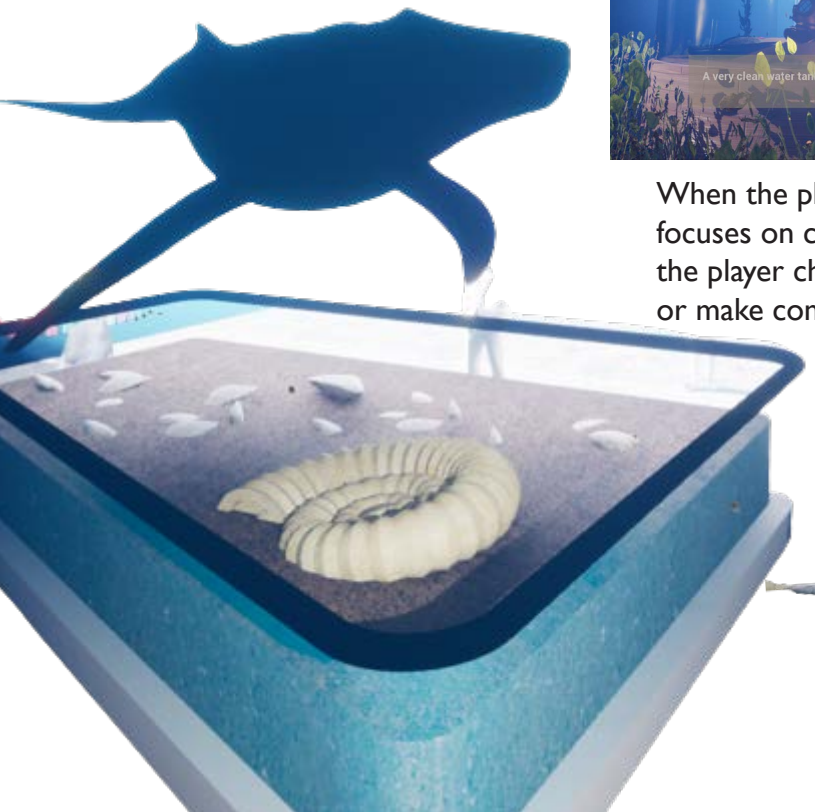
These hint systems aim to provide subtle guidance to players while maintaining the immersive and logical nature of the game's setting.

When the player's gaze first focuses on certain mechanisms, the player character will exclaim or make comments, such as...

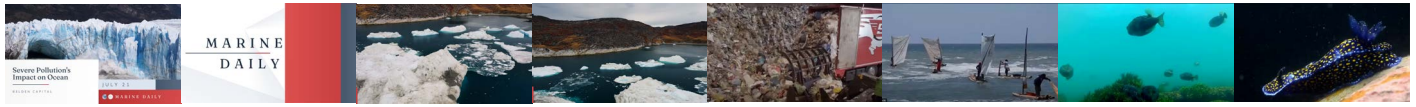


When the player enters a specific area for the first time, the player character may mutter to themselves.

If the player spends a long time without making progress, the character will speak aloud to give the player a hint.



The player can also watch a series of video about ocean pollution when the cinema is unlocked:



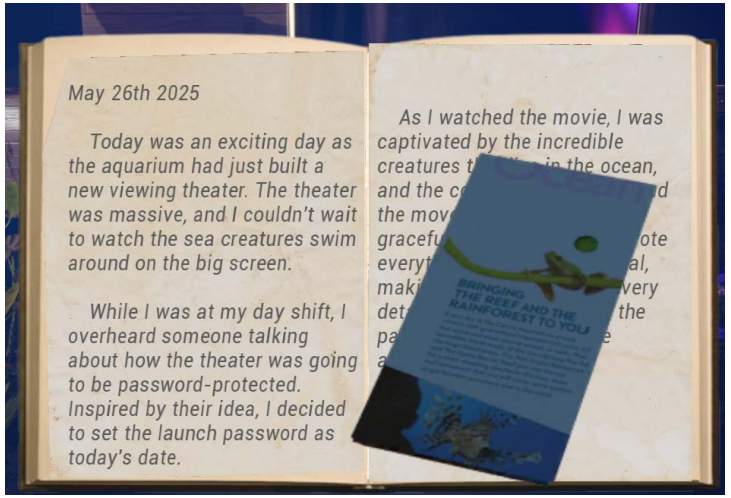
Puzzles

Lobby

As the player's starting point, the player's understanding of the entire game is limited. In the lobby, we need to guide the player's actions, so I designed and created these two straightforward puzzles in the lobby.

1. Hanging out in the Lobby, the player will notice the cylindrical water tank placed in the center of the hallway. It serves as a symbol of game progress and gradually becomes murky as the game progresses.

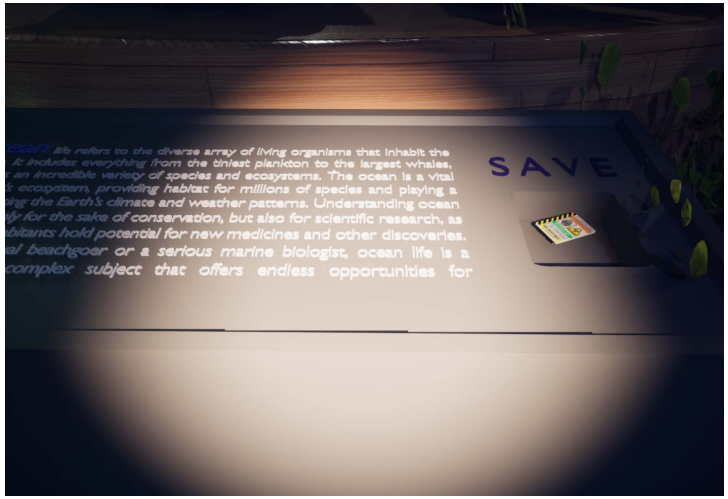
2. The first challenge involves observing the number of crabs/ eels/fish in the tank to unlock a cabinet and obtain a diary. After unlocking the cabinet in the lobby, the player will obtain a diary.



3. Upon opening the diary for the first time, a brochure from the aquarium will fall out.

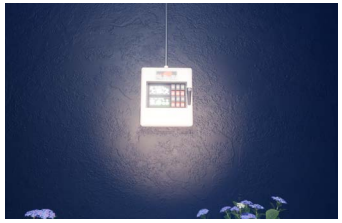
4. Upon closer examination of the brochure, the player will notice that it has half of the text and some holes, and these half-texts correspond to the word "OCEAN" on the central display in the hallway.

5. By holding the brochure towards the display, the player will place it on top, and the four letters corresponding to the holes are the solution to this puzzle.



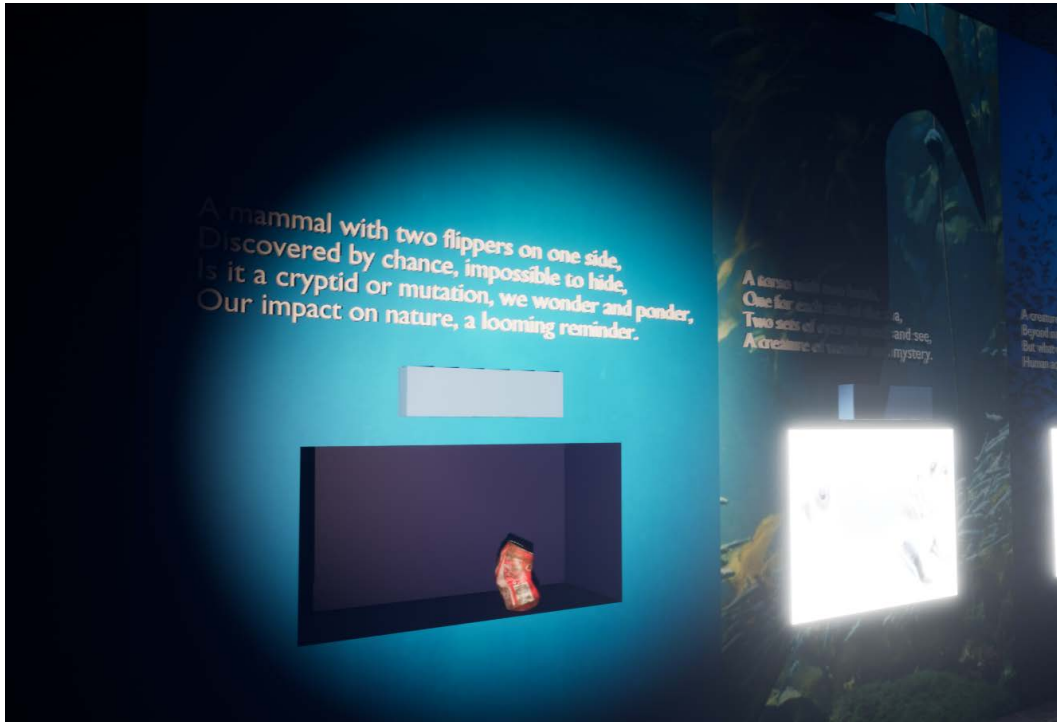
6. Entering "SAVE" as the answer, the player will receive a CD and a lab ID card.

Lab: Inside the lab, there is a broken water purification device that the player needs to repair. Once successfully repaired, the player will obtain a CD and the password to enter the Ocean Animal Exhibition Hall. The password is the date of World Oceans Day. This part is created by my teammates.



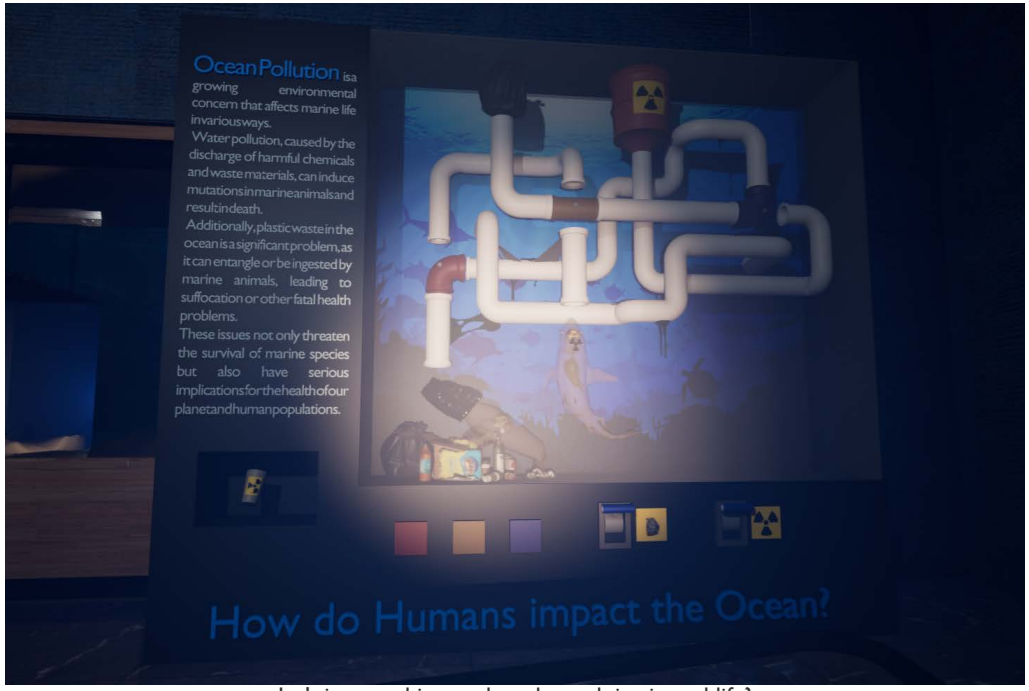
Ocean Animal Exhibition Hall: This hall presents the phenomenon of ocean pollution from the perspective of marine animals.

Due to ocean pollution, marine creatures are experiencing widespread death and mutations. Player needs to match them with the corresponding descriptions on the wall using poetry. Choosing the correct images will reward the player with "Dead Fish," "Mutated Fish," and "Trash," which can be consumed by the player.



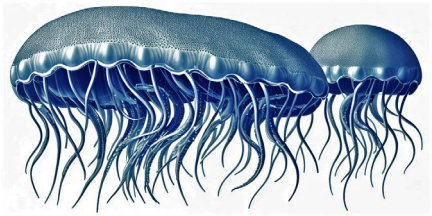
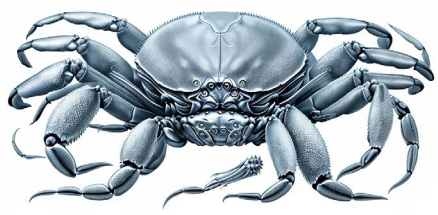
Ironically, aren't the marine animals living in polluted waters feeding on these very things?

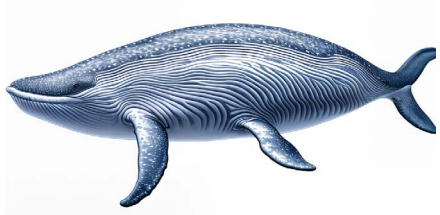
On the other side of this room, I created a water pipe device inspired by the Rust Lake Theater. The player needs to rotate the pipe connections to transport the garbage and nuclear-contaminated water to a sea lion and a shark.



Isn't it something we have been doing in real life?

After successfully completing this level, the player will receive a cup of nuclear-contaminated water. Considering that players may be confused about the use of these items in this room, we have implemented hints in the form of player self-talk such as "I'm so hungry" and "I'm the great shark of the sea" when the player has not used the items in this level for a long time. This hints at the player gradually transforming into a marine creature in this level and consuming these items, implying the plight faced by marine life today.

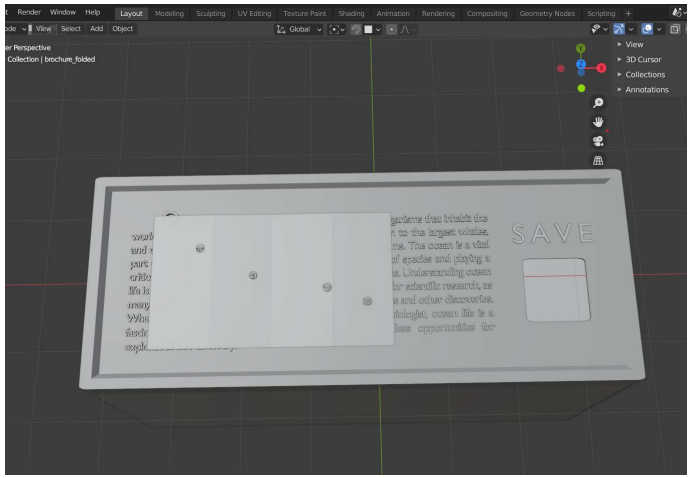






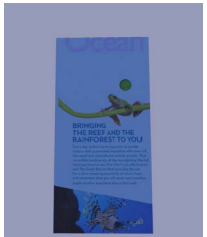
Models

Model: Information Panel

This infomation panel hides a puzzle in the words. When the player place the brochure on it, the word "SAVE" will show up in these holes.



(folded brochure)

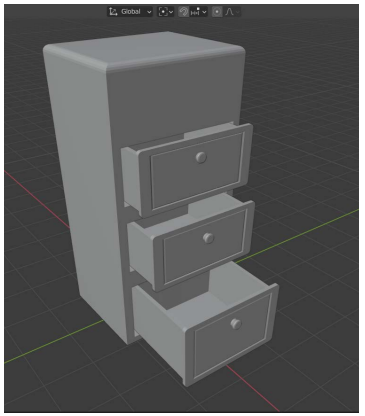


(unfolded brochure)

(brochure placed on the infomation panel)

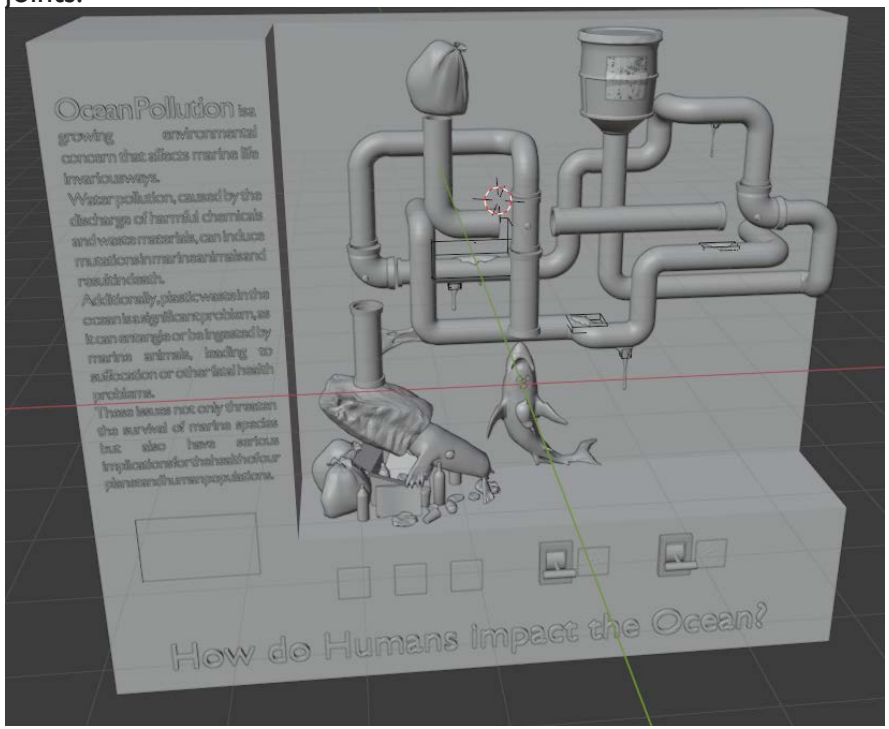
Model: Openable Drawer

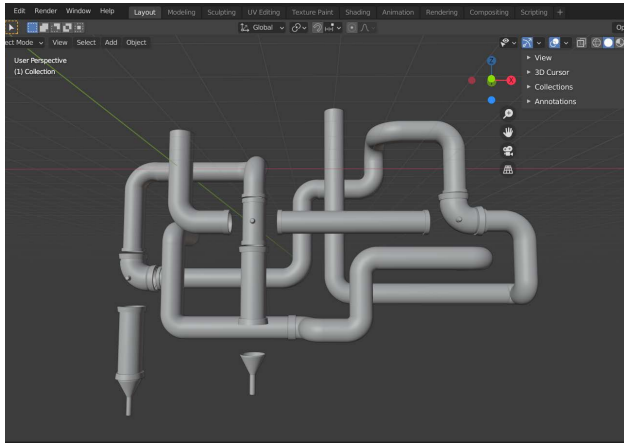
This openable drawer can be implement in a puzzle with a digit lock. It is used to hold a journal in the top drawer. To make it less dull, I design it wide at the top and narrow at the bottom.



Model: Pipeline

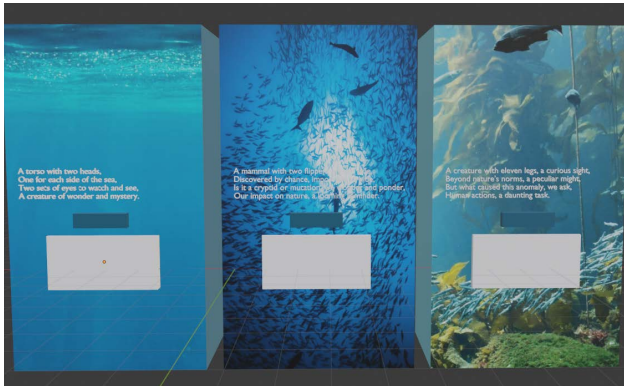
This pipeline model contains 3 rotatable pipe joints. Player will be able to transport 2 sources to destinations by adjusting the pipe joints.



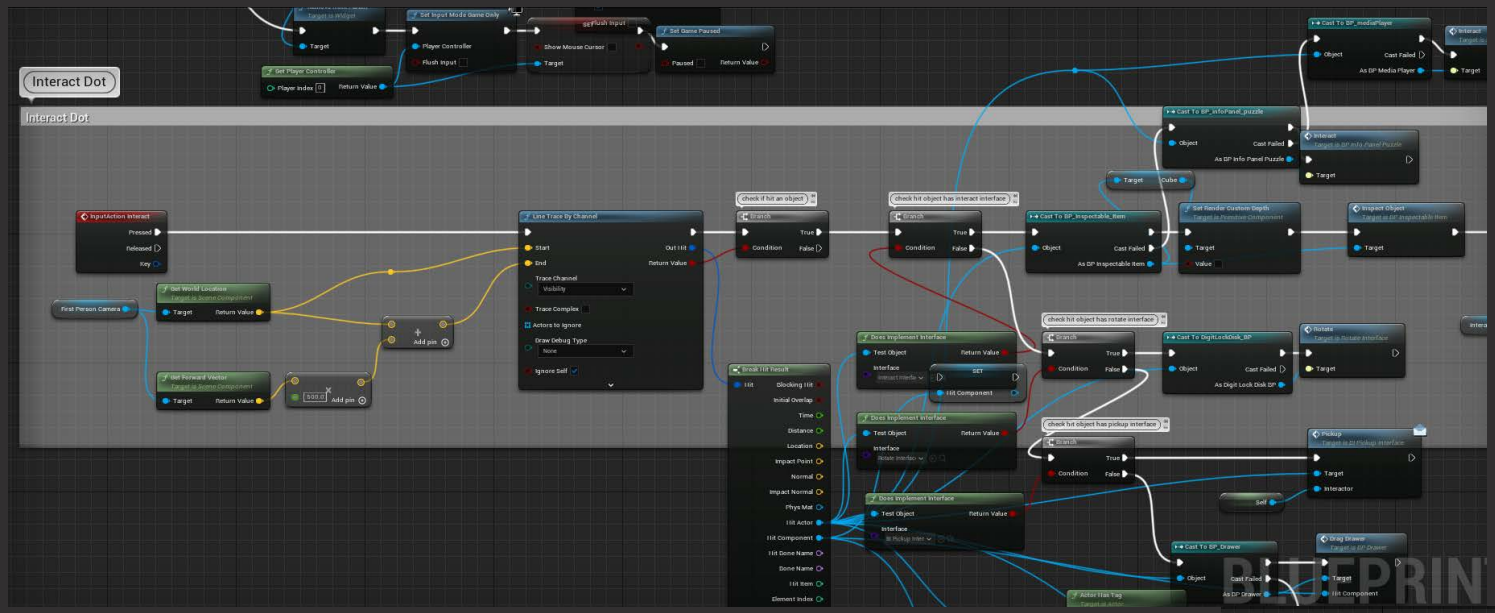


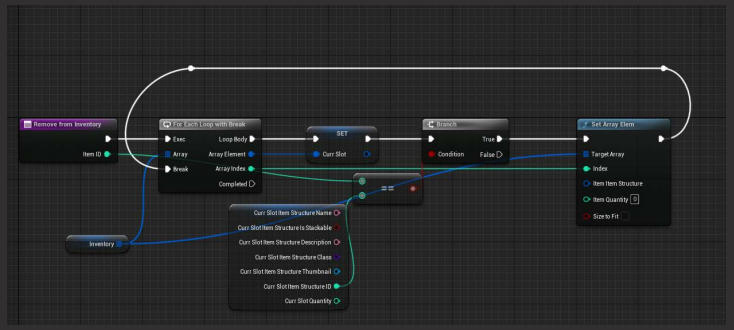
Model: Panels in Exhibition Hall

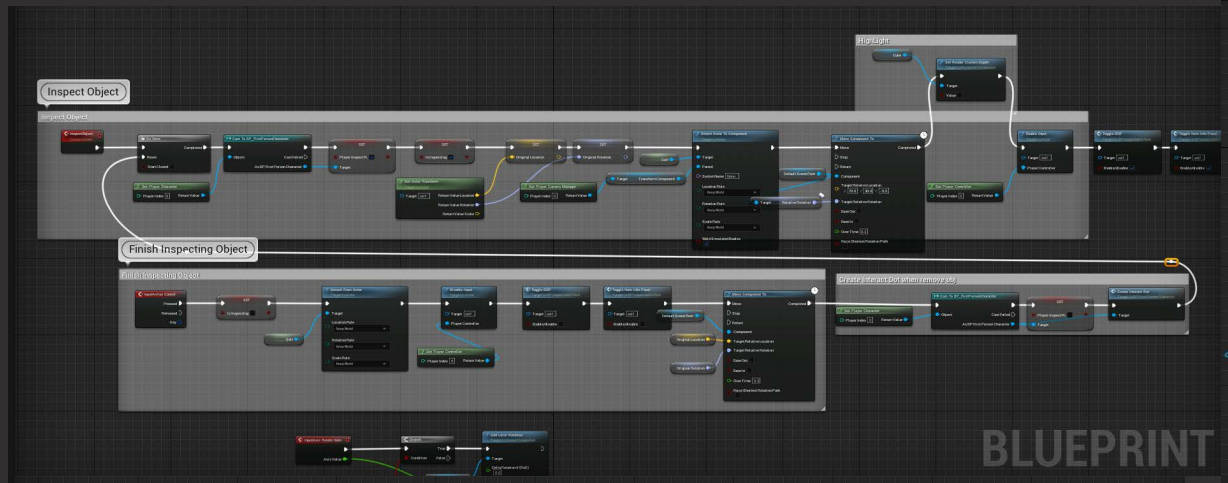
Each panels contains a ironic 4-line poem, which depicts the appearance of a kind of mutated marine animals.

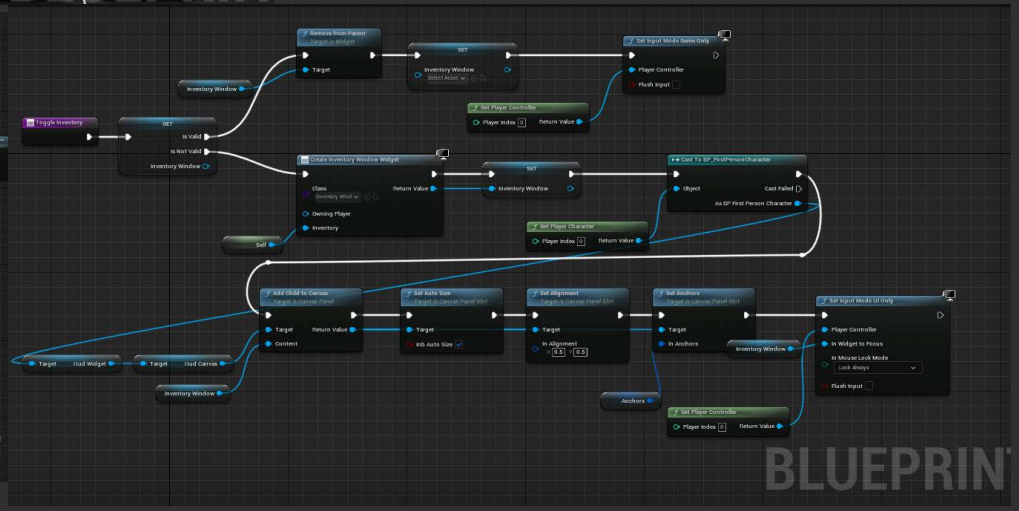


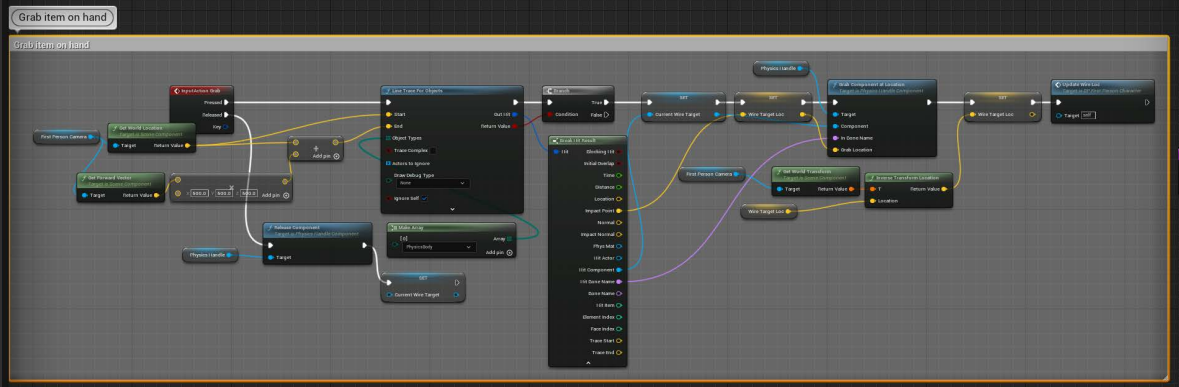
Code

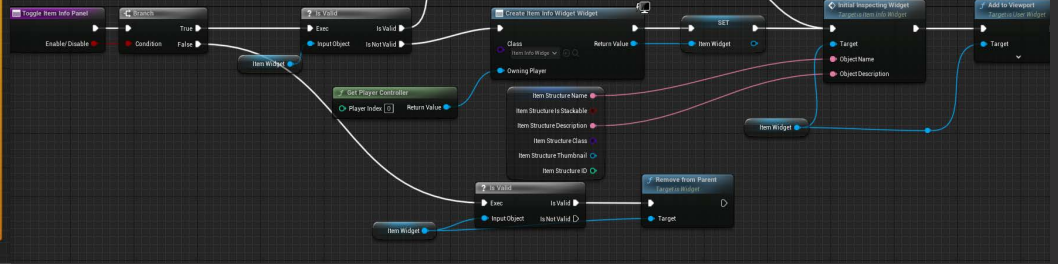

Interact Dot
Interact Dot
InputAction Interact
Line Trace By Channel
Branch
Cast To BP_InspectableItem
Cast To BP_InfoPanel_puzzle
Cast To BP_MediaPlayer
Cast To DigitLockDisk_BP
Cast To BP_Drawer
Does Implement Interface
Break Hit Result
Impact Object
Pickup
Drag Drawer
check if hit an object
check hit object has interact interface
check hit object has rotate interface
check hit object has pickup interface

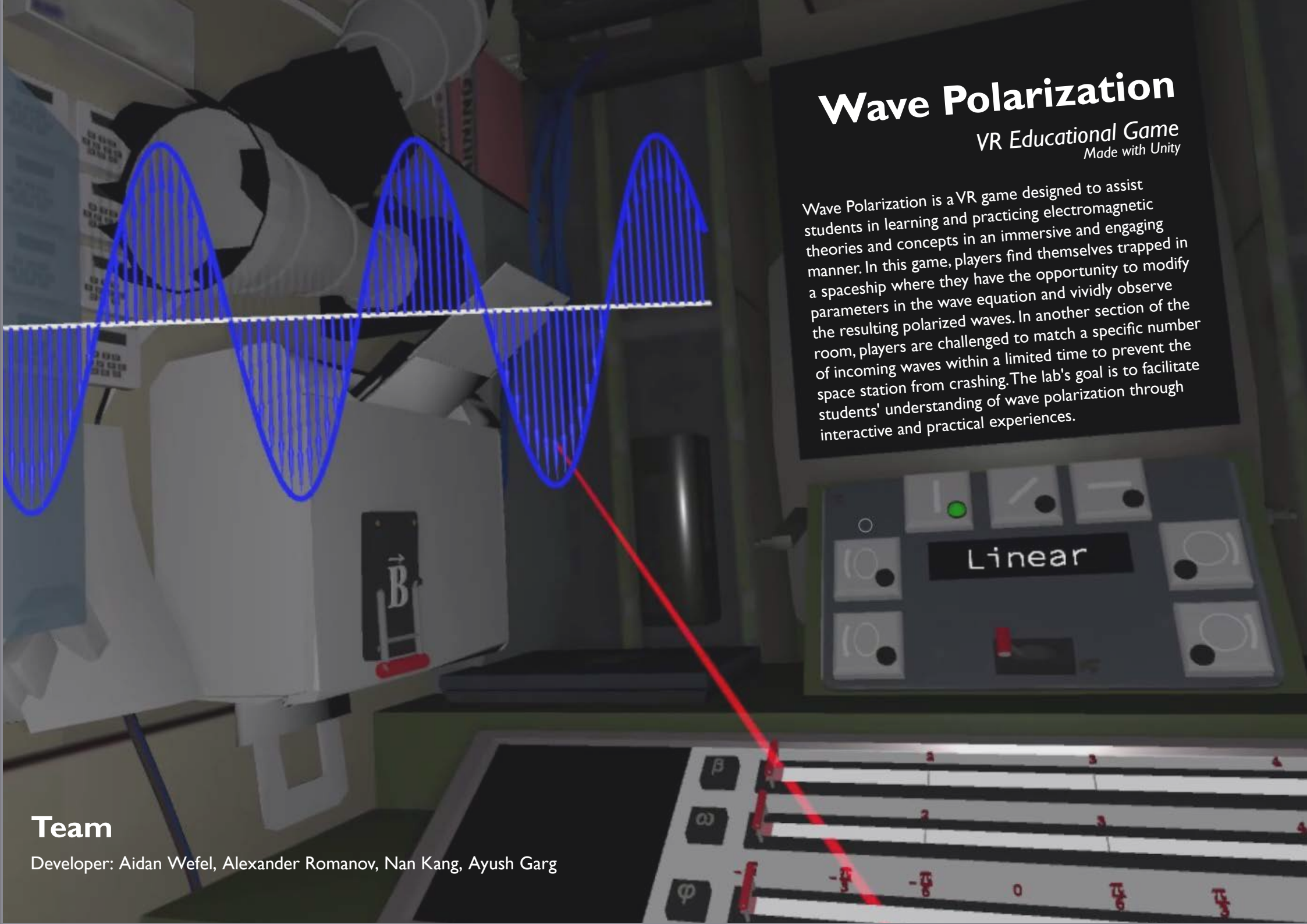

Remove from Inventory
For Each Loop with Break
Set Array Elem
Inventory


Inspect Object
Finish Inspecting Object
Highlight
Create Interact Box when remove obj
BLUEPRINT


Toggle Inventory
Create Inventory Window Widget
Add Child to Canvas
Set Input Mode UI Only
BLUEPRINT


Grab item on hand
Grab item on hand
BLUEPRINT


Toggle Item Info Panel
Create Item Info Widget Widget
Initial Inspecting Widget
Add to Viewport



Wave Polarization

VR Educational Game

Made with Unity

Wave Polarization is a VR game designed to assist students in learning and practicing electromagnetic theories and concepts in an immersive and engaging manner. In this game, players find themselves trapped in a spaceship where they have the opportunity to modify parameters in the wave equation and vividly observe the resulting polarized waves. In another section of the room, players are challenged to match a specific number of incoming waves within a limited time to prevent the space station from crashing. The lab's goal is to facilitate students' understanding of wave polarization through interactive and practical experiences.

Team

Developer: Aidan Wefel, Alexander Romanov, Nan Kang, Ayush Garg

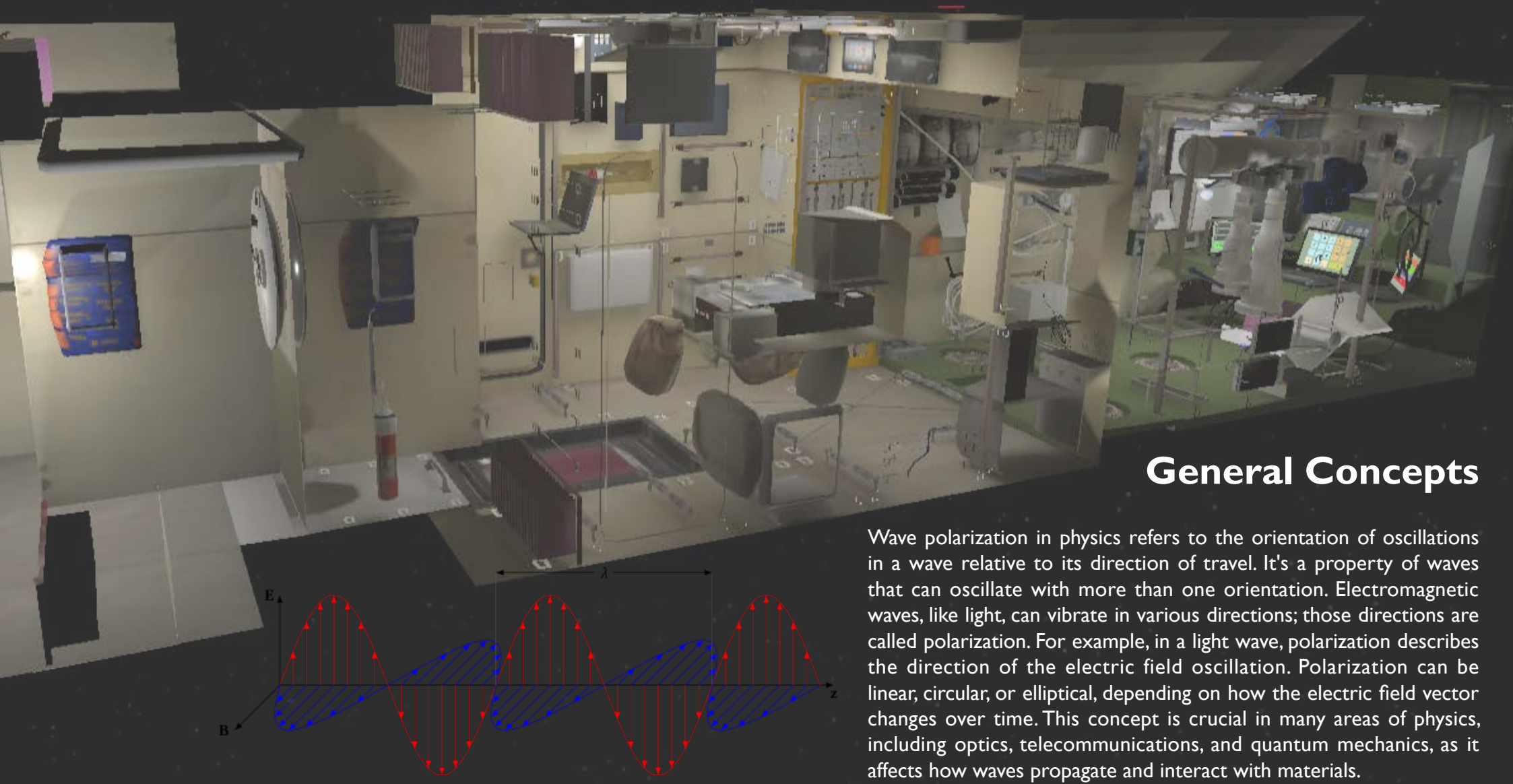
Core Mechanic

Learn Wave Type

Players can modify the parameters in the wave equation, change the propagation direction, and turn the magnetic field wave on or off. They will learn how these parameters affect the wave type and the relationship between electric and magnetic waves and the propagation direction.

Match Wave Type

Polarization can be linear, circular, or elliptical, each with its own polarized angle or orientation. In total, there are seven different types of polarization waves. After sufficient learning, players need to test their understanding by matching the incoming waves to their correct types.



General Concepts

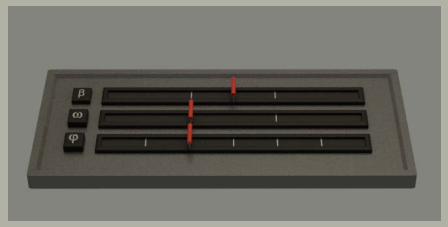
Wave polarization in physics refers to the orientation of oscillations in a wave relative to its direction of travel. It's a property of waves that can oscillate with more than one orientation. Electromagnetic waves, like light, can vibrate in various directions; those directions are called polarization. For example, in a light wave, polarization describes the direction of the electric field oscillation. Polarization can be linear, circular, or elliptical, depending on how the electric field vector changes over time. This concept is crucial in many areas of physics, including optics, telecommunications, and quantum mechanics, as it affects how waves propagate and interact with materials.

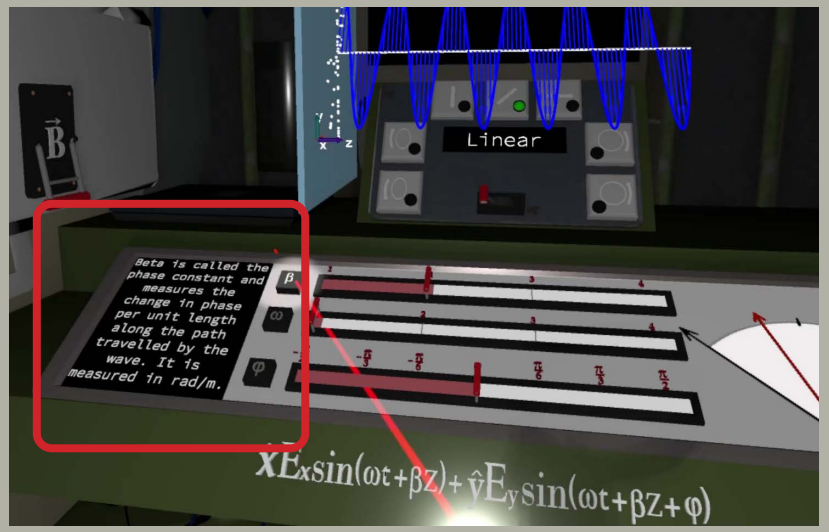
Development

Experiment Panel

I created this model in blender. There are 4 sliders on this panel for players to select the parameters' values. The button on the left of the slider can be pressed to show the description of the parameter.

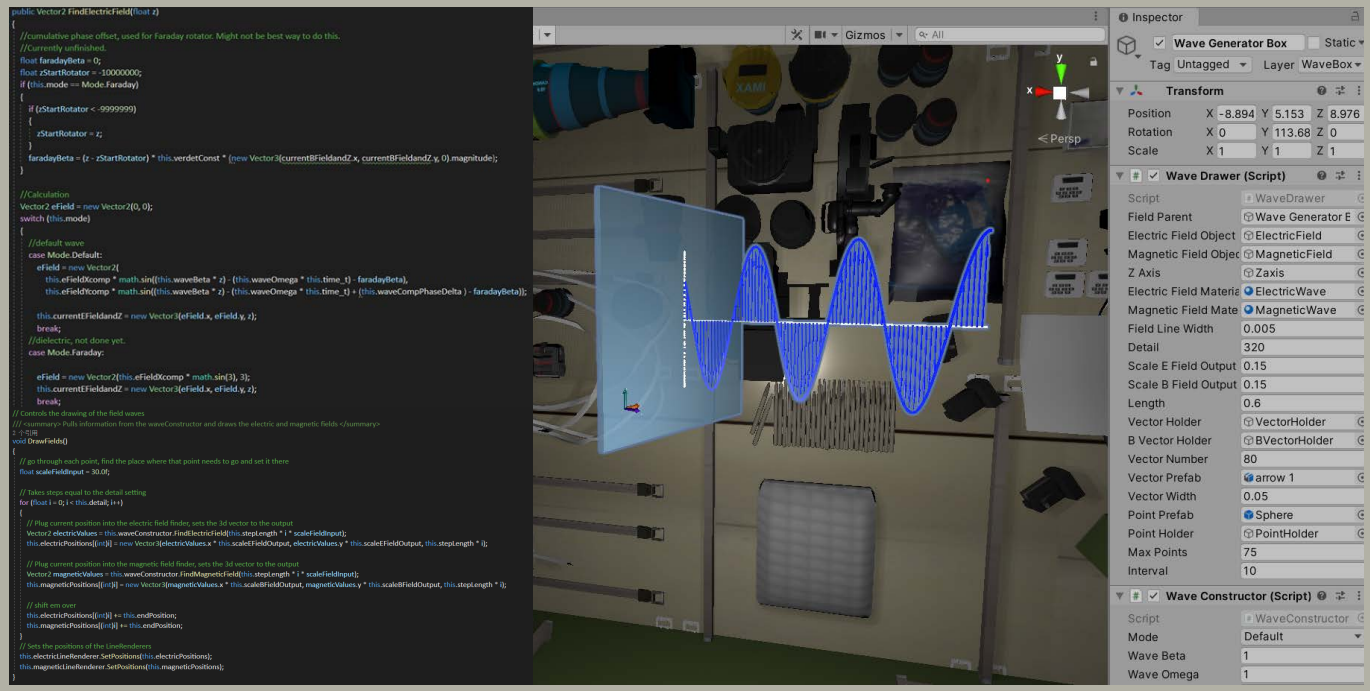
The button on the left of the slider can be pressed to show the description of the parameter.





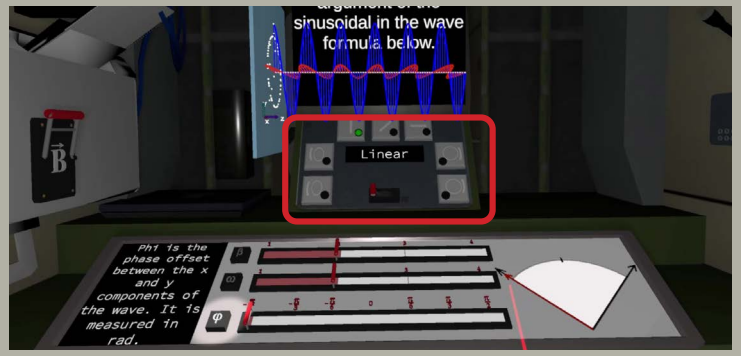
Wave Generator

The wave generator in the game can be moved and rotated by the player. It generates electrical and magnetic waves based on various parameters in the wave equation and the direction of wave propagation.

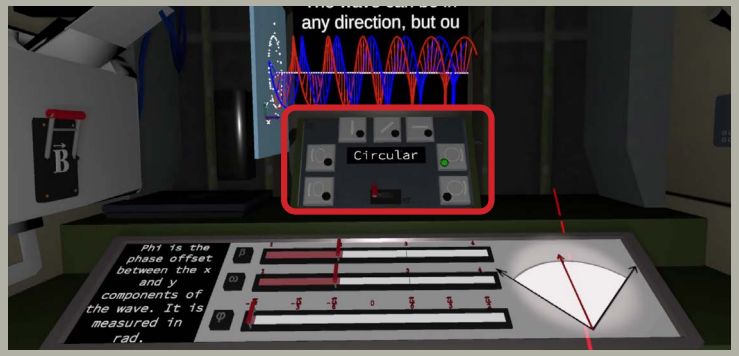


This panel features 7 indicators, each displaying the corresponding polarized type of the generated wave.

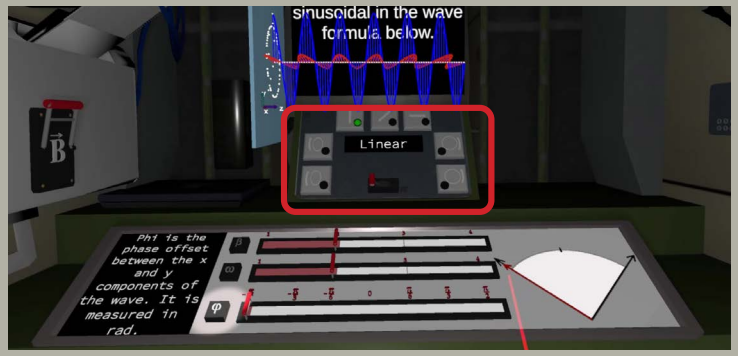
Linear Polarization



Circular Polarization

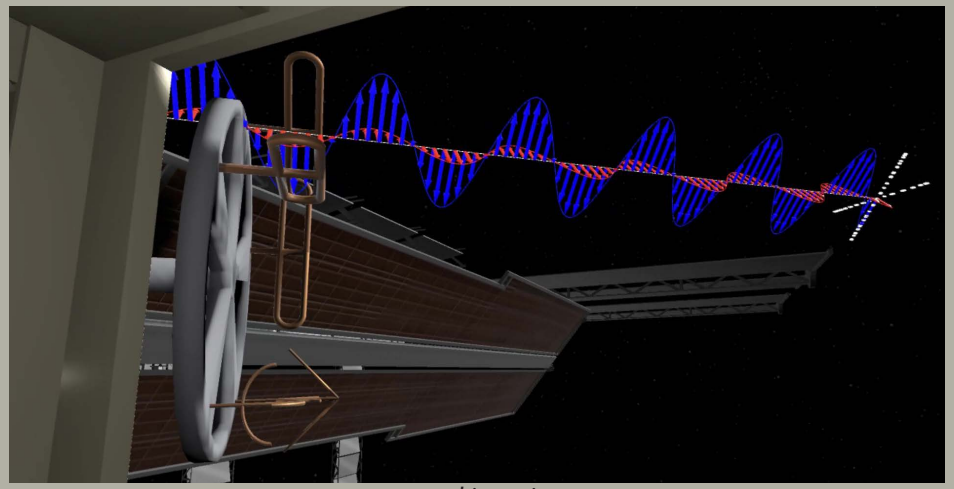


Eliptical Polarization

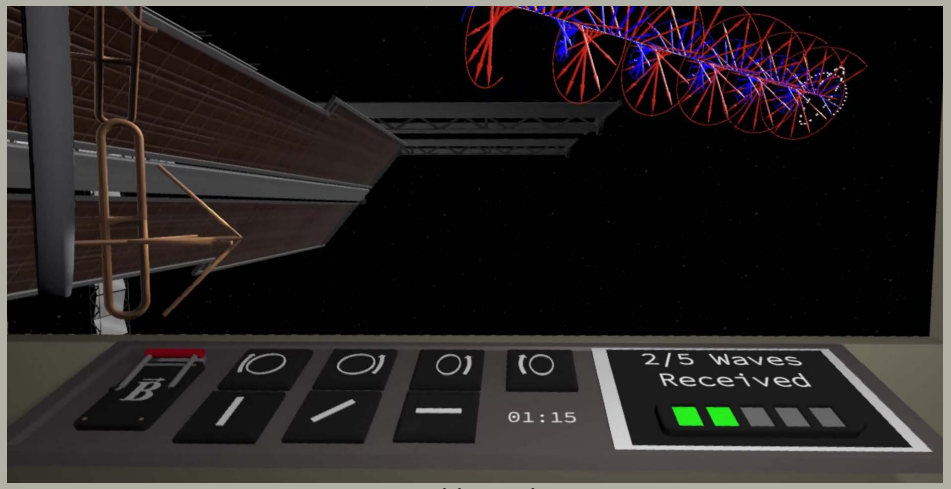


Wave Receiving Station

After acquiring sufficient knowledge about polarized waves, players can teleport to the other side of the space station to test their understanding. We created several incoming wave for player to correctly match 5 waves to corresponding polarized types within 2 minutes. Successfully completing this challenge is crucial to prevent the space station from crashing.

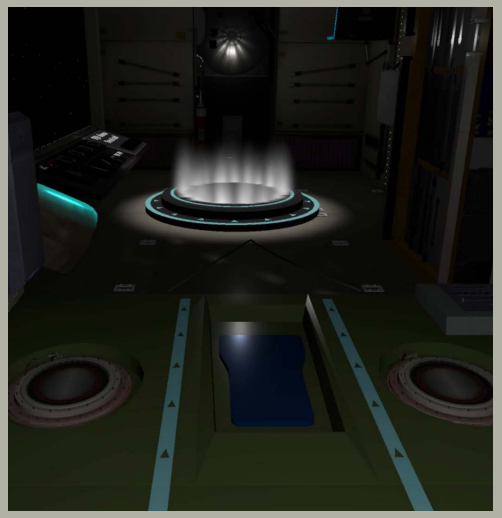


antenna and incoming wave

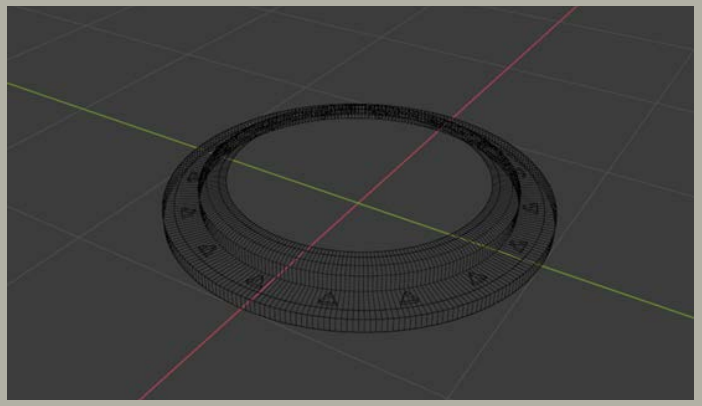


receiving station

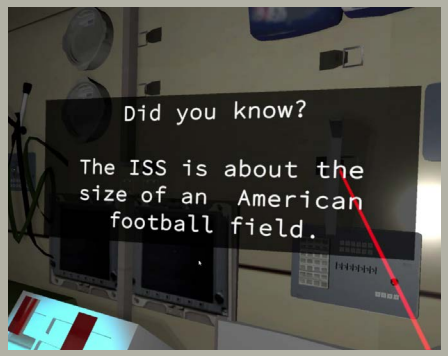
Teleportation Station



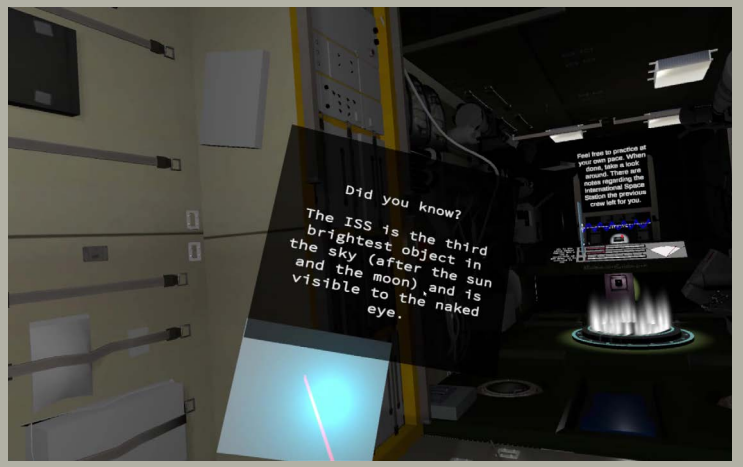
I designed the teleportation station model in Blender and then programmed a script to make it periodically flash a blue light.

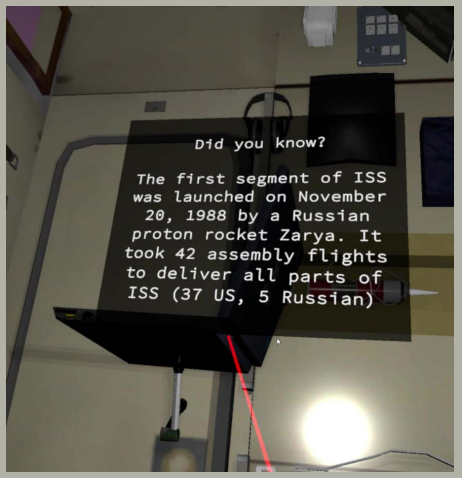


Clickable Fun Fact Objects



In this space station, players will find various objects suspended in the environment, which they can move and rotate using their controllers. Interacting with these objects by clicking on them reveals fun facts about the space station.





Code

```
void CallWaveCrash() {
        Debug.Log("Wave Crashed");
        this.StartCoroutine(this.FlashLightsRed());
    }

    void CallWaveSuccess() {
        Debug.Log("Wave good");
        this.wavesReceived++;
        this.StartCoroutine(this.TextScrolling());
        this.audioSource.Play();
    }

    void Awake()
    {
        waveTransform = wave.
GetComponent<Transform>();
        transmissionSettings = wave.GetComponen
t<TransmissionSettings>();
        rotateWheel = wheel.
GetComponent<RotateWheel>();
        scoreText = scoreTracker.GetComponent<
TextMeshProUGUI>();
        timerText = countDownTimer.GetCompon
ent<TextMeshProUGUI>();
        narratorController = narrator.GetCompon
ent<NarratorController>();
        waveSound = waveSoundObject.
GetComponent<AudioSource>();
        wavesReceived = 0;
        waveCrashed += CallWaveCrash;
        waveSuccess += CallWaveSuccess;
        lt1 = light1.GetComponent<Light>();
        lt2 = light2.GetComponent<Light>();
        scoreText.text = wavesReceived.ToString()
+ "/" + howManyWaves.ToString() + " Waves\
nReceived";
        SetBattery();
    }

    public void callOnTeleport()
    {
        if (!playedStartupSound) {
            StartCoroutine(DelayGame());
            playedStartupSound = true;
        }

    }
    public void startGame()
    {
        //Debug.Log("Game started");
        scoreText.text = wavesReceived.ToString()
+ "/" + howManyWaves.ToString() + " Waves\
nReceived";
        SetBattery();
        if (gameStarted == false)
        {
            gameStarted = true;
            wave.active = true;
            currentWaveTime = timePerWave;
            ResetWave();
            this.StartCoroutine(this.OneMinWarning());
            narratorController.Play2MinWarning();
        }

    }

    // Update is called once per frame
    void Update()
    {
        if (!gameStarted)
        {
            return;
        }
        if (overallTimeValue > 0)
        {
            overallTimeValue -= Time.deltaTime;
            currentWaveTime -= Time.deltaTime;
            if (wavesReceived < howManyWaves)
            {
                if (currentWaveTime > 0)
                {
                    //Debug.Log("It works");
                    waveTransform.position +=
waveTransform.TransformDirection(Vector3.fwd) *
waveTranslateSpeed * Time.deltaTime;
                } else {
                    //enums are pain
                    if ((int)transmissionSettings.waveType
== lookupList[rotateWheel.state])
                    {
                        if(rotateWheel.state == 0 ||
rotateWheel.state == 1) {
                            if((clockwise &&
transmissionSettings.outphi < 0) || (!clockwise &&
transmissionSettings.outphi > 0)){
                                waveSuccess?.Invoke();
                            } else {
                                waveCrashed?.Invoke();
                            }
                        } else {
                            // checks if null, if its not, calls it
                            waveSuccess?.Invoke();
                        }

                    } else {
                        waveCrashed?.Invoke();
                    }
                    ResetWave();
                    waveSound.Play();
                }

            } else if (wavesReceived == howManyWaves)
            {
                wave.SetActive(false);
                narratorController.PlayGameSuccess();
                gameStarted = false;
            }
            float minutes = Mathf.
FloorToInt(overallTimeValue / 60); // convert game
time to minutes and seconds
            float seconds = Mathf.
FloorToInt(overallTimeValue % 60);
            timerText.text = string.
Format("{0:00}:{1:00}", minutes, seconds); // black
magic string formatting

        } else if (wavesReceived < howManyWaves)
        {
            waveCrashed -= this.CallWaveCrash; //
unsubscribe the functions from the event, without it
there were errors changing scenes
            waveSuccess -= this.CallWaveSuccess;
            SceneManager.LoadScene("SpaceStation_
Exterior_Night"); //load the crash scene
        }

    }
    void ResetWave() {
        waveTransform.position = waveSpawnLocation.
GetComponent<Transform>().position;
        currentWaveTime = timePerWave;
        transmissionSettings.AdvanceWave();
    }

    void SetBattery()
    {
        for (int i = 0; i < howManyWavesBattery.
transform.childCount; i++)
        {
            howManyWavesBattery.
transform.GetChild(i).gameObject.
GetComponent<Renderer>().material.color =
wavesReceived >= i + 1 ? Color.green : Color.gray;
        }
    }

    IEnumerator TextScrolling() {
        string targetText = wavesReceived.ToString()
+ "/" + howManyWaves.ToString() + " Waves\
nReceived";
        //Debug.Log(targetText);
        string currentText = "";
        while (currentText != targetText) {
            currentText = targetText.Substring(0,
currentText.Length + 1);
            scoreText.text = currentText;
            SetBattery();
            yield return new WaitForSeconds(0.1f);
        }
    }

    IEnumerator FlashLightsRed() {
        int count = 0;
        float startTime = Time.time;
        while (count < 3) {
            float t = Mathf.PingPong(Time.time,
flashDuration) / flashDuration;
            lt1.color = Color.Lerp(colorRed,
colorWhite, t);
            lt2.color = Color.Lerp(colorRed,
colorWhite, t);
            if (Time.time - startTime > 3) {
                lt1.color = Color.white;
                lt2.color = Color.white;
                yield break;
            }
            yield return null;
        }
        yield break;
    }

    //IEnumerator PromptForGame()
    //{
    //    if (!gameStarted)
    //    {
    //        narratorController.PlayStartingTalk();
    //    }
    //}

    IEnumerator OneMinWarning()
    {
        yield return new WaitForSeconds(61f); //
delayed by one second so that it won't play over the
success sound if the player
        // wins on their 6th wave
        if (gameStarted)
        {
            narratorController.Play1MinWarning();
        }
    }
```

Phantom Seascape

2D Puzzle Game

Made with Unity

On the surface, it exudes luxury and allure, but once aboard, a series of unsettling and enigmatic events transform your idyllic getaway into a chilling ordeal.

Abstract

In the 2D puzzle-adventure game "The Phantom Seascape", players are immersed in a Cthulhu-themed narrative. Trapped aboard the sinking "Phantom Seascape," players must navigate through four levels and seven intricately designed rooms. As water levels rise and oxygen dwindles, players must gather items and decipher riddles to make their escape, all while the eldritch atmosphere intensifies the urgency.

Team

Art: Bojun Gu, Yuanyue Tao, Xintong Chen
Code: Nan Kang
Game design: Yuanjin Fang, Yuanyue Tao, Nan Kang
Technical art: Han Chen
Narrative: Yuanyue Tao

Inspiration

The ocean embodies the perfect paradox for an escape game. Its limitless horizon contrasts sharply with the claustrophobic feel of being trapped within its depths. This tension, coupled with the ocean's inherent mysteries, heightens the gameplay experience.

Shipwrecks, with their maze-like structures and untold stories, further enrich this setting. They symbolize humanity's vulnerability against nature and provide intricate backdrops for puzzles and clues.

In essence, the ocean offers a unique blend of exploration and urgency, making it an ideal canvas for a captivating escape game experience.

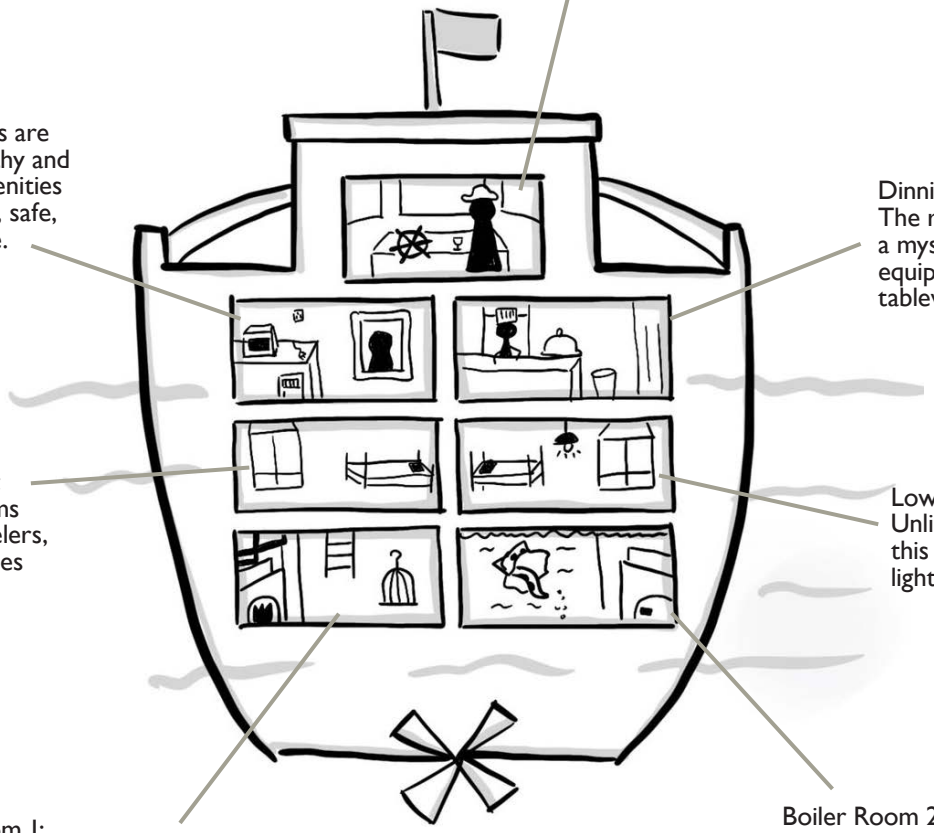
Map Design

We designed 7 rooms, including: Boiler Room 1, Boiler Room 2, Lower Class Room 1, Lower Class Room 2, Upper Class Room, Dining Room, and Captain's Room. These rooms are arranged across 4 decks of the ship. To visualize these rooms better, I created a floorplan below:

Captain's Room:
The locked room cannot be entered until all keys are collected. Inside, there's an evil captain that must be defeated in order to obtain the lifeboat.

Upper Class Room:
The upper-class rooms are designed for the wealthy and are equipped with amenities such as TV, telephone, safe, oil paintings, and more.

Dinning Room:
The restaurant features a mysterious chef and is equipped with an array of tableware for diners.



Lower Class Room 1:
The lower-class rooms cater to budget travelers, offering basic amenities such as a bed and a wardrobe.

Lower Class Room 2:
Unlike the room on the left, this room has a switchable light.

Boiler Room 1:
The boiler room burns fuel, with some pipes transporting steam, located at the very bottom of the ship.

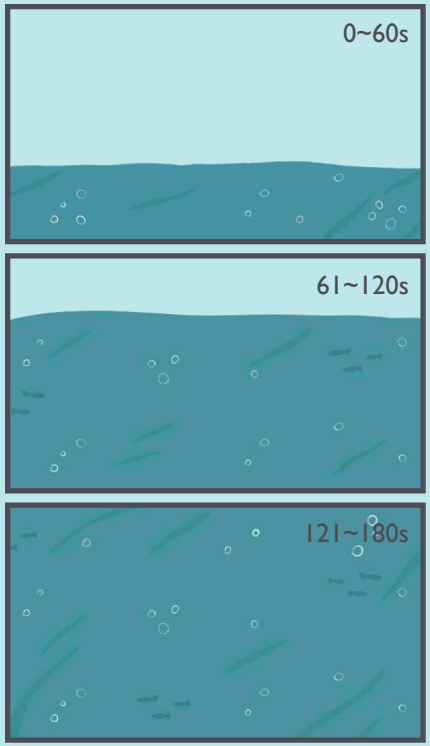
Boiler Room 2:
The boiler room is non-functional due to an unexplained hole, through which seawater is continuously pouring into the cabin.

Development

- Water Rising

Unlike typical escape room games, this game has a time limit. As the game progresses, the water level will flood a room every three minutes. If the player doesn't escape before the entire ship is submerged, they will drown.

To depict the rising water levels, I initially created three water level diagrams to overlay on the rooms, updating every minute.



We've introduced a clock as a health indicator for the player. In water-filled rooms, their oxygen depletes in 2 minutes. If the clock becomes entirely red, the player succumbs to drowning. Exiting these submerged areas allows their oxygen to gradually restore.

After multiple tests, I felt that this mode couldn't give players an intuitive sense of urgency. So, I adopted a water shader and attached code to refresh the water level at intervals. To strike a balance between the desired effect and operational efficiency, after repeated testing, I set the refresh interval to 0.1 seconds.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Game2DWaterKit;

Unity 脚本(1 个资产引用)|0 个引用
public class WaterRise : MonoBehaviour
{
    public float sec;
    public int min;
    public float waterPercent;
    public float tenCount;
    public float floorCount;

    public GameObject waterEffect;
    public GameObject waterEffect2;
    public bool waterinLevel2;

    public GameObject fireEffect;

    Unity 消息|0 个引用
    public void Start()
    {
        sec = 0;
        min = 0;
        waterPercent = 0;
        tenCount = 0;
        waterinLevel2 = false;
    }

    public void FixedUpdate()
    {
        sec += Time.fixedDeltaTime;
        tenCount += Time.fixedDeltaTime;
        floorCount += Time.fixedDeltaTime;

        //water between 2 levels
        if (floorCount >= 180f)
        {
            floorCount = 0;
            sec += 90;
            waterinLevel2 = true;
        }

        //water rise per 0.1s
        if (tenCount >= 0.1f)
        {
            tenCount = 0;
            waterPercent = sec / 18;
            SetWaterShaderLevel(waterPercent);
            if (waterinLevel2)
            {
                SetWaterShader2Level(waterPercent);
            }
        }

        if ((sec > 10f) & (sec < 10.5f)){
            fireEffect.SetActive(false);
        }
    public void SetWaterShaderLevel(float y_size)
    {
        waterEffect.GetComponent<Game2DWater>().MainModule.SetSize(new Vector2(18f, y_size));
        waterEffect.transform.position = new Vector3(0, -5 + y_size/2, 0);
    }
```

- Inventory System

The player can pick up and use items in the scene. As players may accumulate many items in their inventory during the game, I designed a scrollable inventory system. Players can click the flip buttons at the top and bottom to scroll through the inventory slots.

```
public class ScrollInventory : MonoBehaviour
{
    public RectTransform Grids;
    private float posX;
    private float posY;

    0 个引用
    public void ScrollDown()
    {
        posX = Grids.anchoredPosition.x;
        posY = Mathf.Clamp(Grids.anchoredPosition.y + 120f, 0, 600);
        Grids.anchoredPosition = new Vector2(posX, posY);
    }

    0 个引用
    public void ScrollUp()
    {
        posX = Grids.anchoredPosition.x;
        posY = Mathf.Clamp(Grids.anchoredPosition.y - 120f, 0, 600);
        Grids.anchoredPosition = new Vector2(posX, posY);
    }
}
```

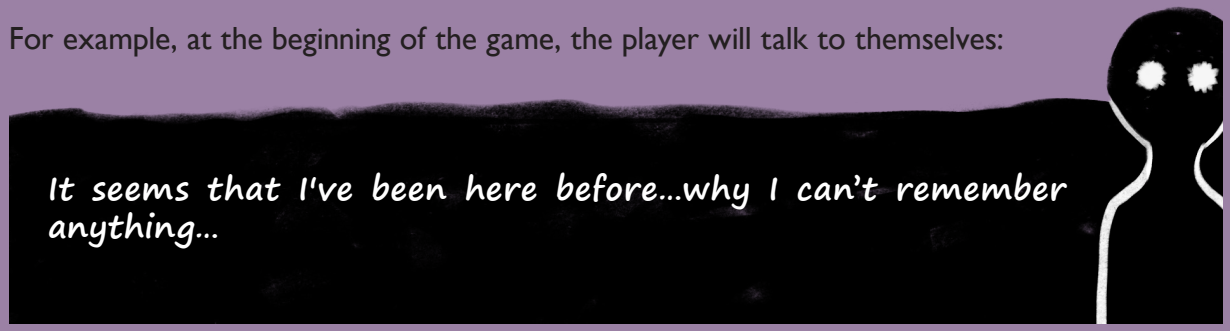
```
public class InventorySystem : MonoBehaviour
{
    [System.Serializable]
    5 个引用
    public class InventoryItem
    {
        public GameObject obj;
        public int stack = 1;
        1 个引用
        public InventoryItem(GameObject o, int s = 1)
        {
            obj = o;
            stack = s;
        }
    }
    //list of item picked up
    public List<InventoryItem> items = new List<InventoryItem>();

    //inventory system window
    [Header("UI Item Selection")]
    public bool isSelected = false;
    public int selectedIdx = -1;
    public GameObject ui_window;
    public Image[] items_images;
    public Image[] select_frames;
```

```
public void PickUp(GameObject item)
{
    if (items.Count <= 12)
        {
            InventoryItem i = new InventoryItem(item);
            items.Add(i);
        }
    Update_UI();
    item.SetActive(false);

}

1 个引用
public bool CanPickUp()...

5 个引用
public void DropFromInventory(int id)
{
    if (items[id] == null) return;
    items[id].stack = 0;
    items.RemoveAt(id);
    Update_UI();
}
void Update_UI()
{
    HideAll();
    for (int i = 0; i < items.Count; i++)
    {
        items_images[i].sprite = items[i].obj.GetComponent<SpriteRenderer>().sprite;
        items_images[i].gameObject.SetActive(true);
    }
}
1 个引用
void HideAll()...
2 个引用
void HideAllSelect()...
5 个引用
public void SelectItem(int itemidx)
{
    //select empty grid or deselect current selection
    if ((itemidx >= items.Count) || (itemidx == selectedIdx)|| (itemidx < 0))
    {
        isSelected = false;
        selectedIdx = -1;
        HideAllSelect();
    }
    else
    {
```

- Dialogue System

During the puzzle-solving process, players are very likely to encounter difficulties. For instance, they might not know what the item they've obtained is, or how to use it. At such times, the inclusion of a dialogue box allows us to explain some background and intentions to the player.

For example, at the beginning of the game, the player will talk to themselves:



When obtaining certain items, the character will hint to the player about what it is:

> "A jar of moldy chili sauce? That's a one-way ticket to the afterlife..."
>
> "Huh, a messy bunch of hair."

When clicking on an NPC to initiate a conversation, a dialogue box from the NPC will also pop up, indicating to the player what items they need to give them. This dialogue box will disappear automatically after a period of time.

Puzzles and Interactive Items

Clock Puzzle

Players can turn the hands of the clock by clicking on the clock face. When all the hands are in the correct position, the puzzle will be unlocked.

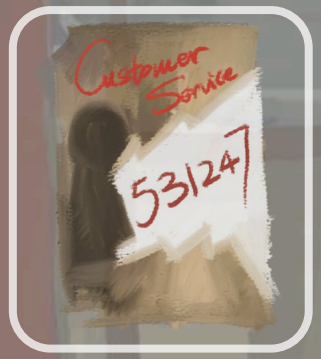
Cage Covered by Cloth

Remove the cloth with a knife, and a human head is revealed inside the cage. One of his eyeballs has the digits "123" on it.

Item Hidden in Wardrobe

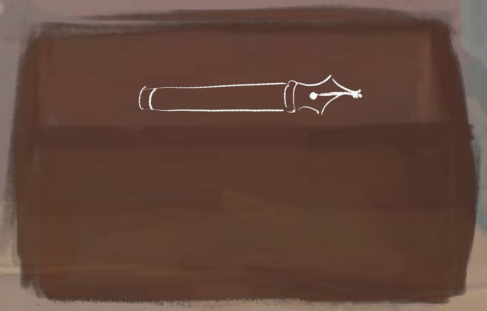
Poster Fragment

If the player find the lower half of the poster in the restaurant, they can combine it to form a poster with a customer service number.



Lipstick

Upon opening the cabinet door, player will find a lipstick, one of required items for completing the woman's portrait.



Pen

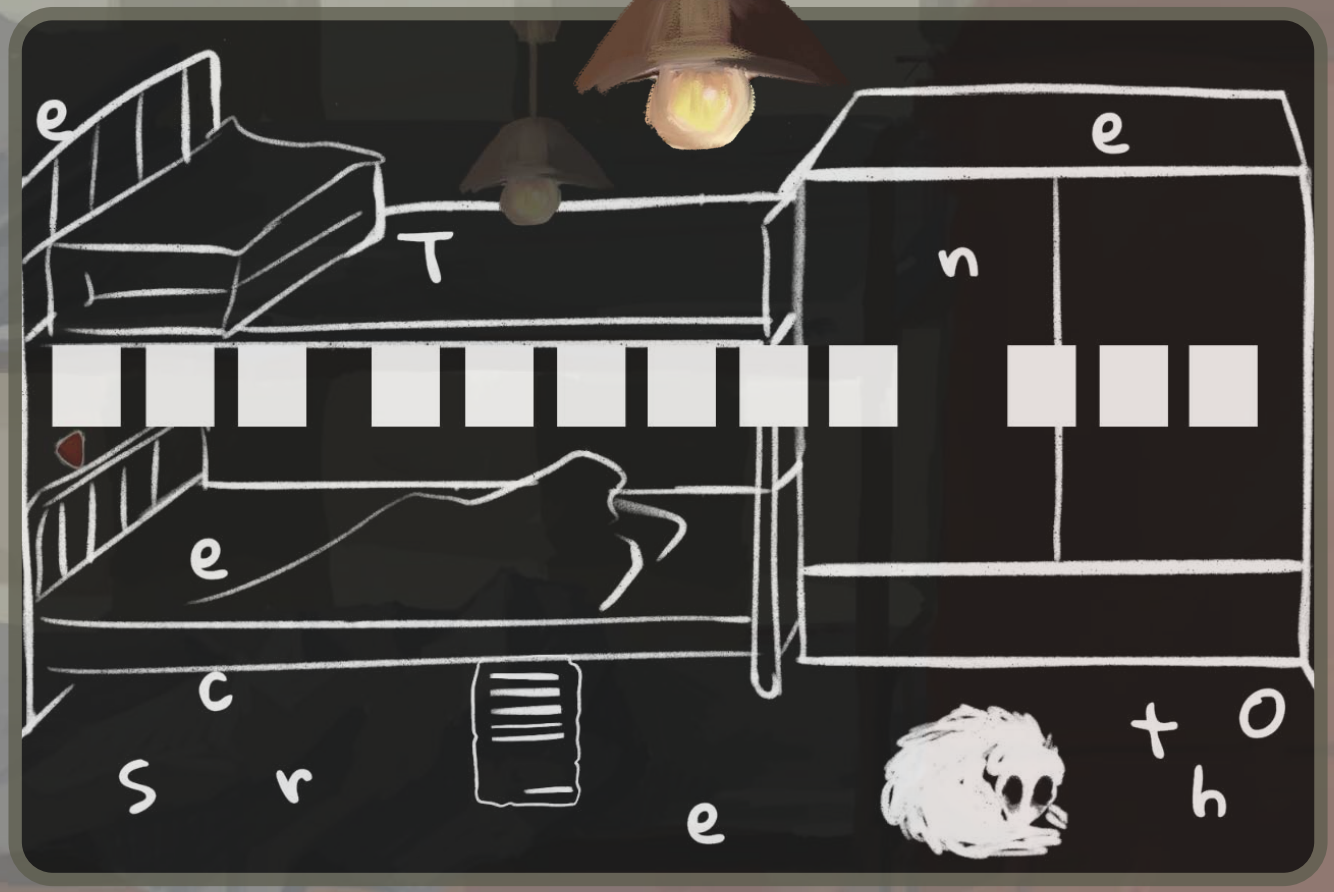
A pen outlined in white, designed for signatures in a monochrome world.

Crying Baby

A constantly crying baby that can only be quieted by giving it a book. Give him the book, he will give player a piece of a photo fragment in return.

Switchable Lamp

Once player switches off the light, darkness envelops the room, revealing new mysteries within this shadowed place.



Broken photograph

By assembling the three collected photo fragments, player can recreate a full group photo. It contains the answer for the boiler room's clock face.

Passenger Satisfaction Survey Form

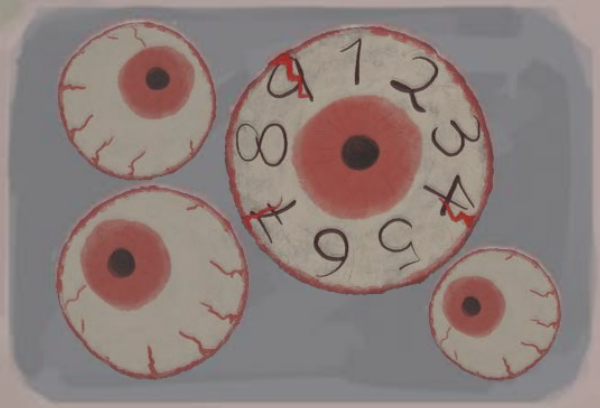
It requires a white pen to be filled out in the dark. Annoyingly, it forces its customers to give it a positive review.

A Dying Dog

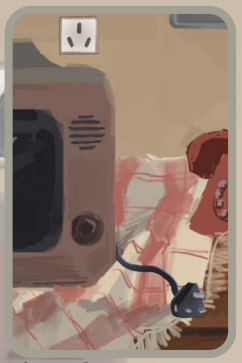
It's nearing its end. Give it a gentle pat, and player will hear a soft whimper. Offer it a bone, and in return, it will expel a woman's hand.

TV Puzzle

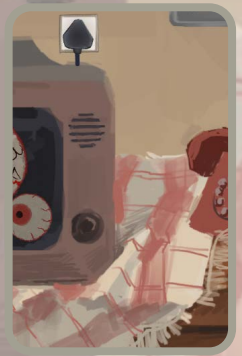
By interacting with the smaller eyeballs around, the blood streaks on the main eyeball shift in response. Drawing from the hint we gathered in Boiler Room 1, these blood streaks should align with positions 1, 2, and 3.



3. Solving Puzzle



1. Before Plugging



2. After Plugging

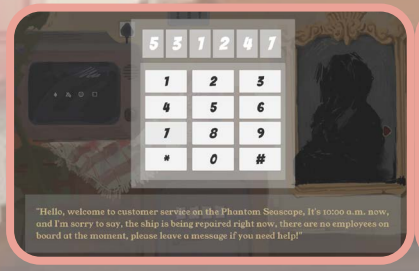
4. Solved

Captain Room Lock

Need 3 keys to unlock it.

Customer Service Telephone

Players have the option to dial a number. Entering the correct customer service number (531247) provides the current time.
However, dialing any other number connects them to a malevolent stranger, misleading them with false information about a life vest in the boiler room.

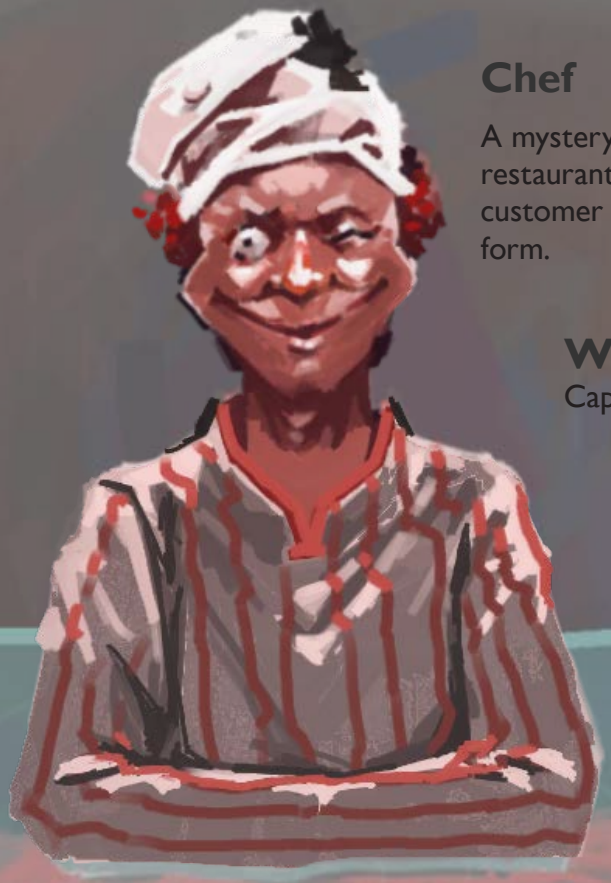


Lock Box

Match the patterns as the answer shown on the TV. A key to the Captain Room is here.

A Portrait of a Woman

Initially, the portrait is shrouded in darkness. As player restore her missing elements (hand, hair, lipstick), her image gradually becomes visible. She'll express her hunger, and player can satiate it by offering her a cake. In gratitude, she provides another key leading to the captain's room.

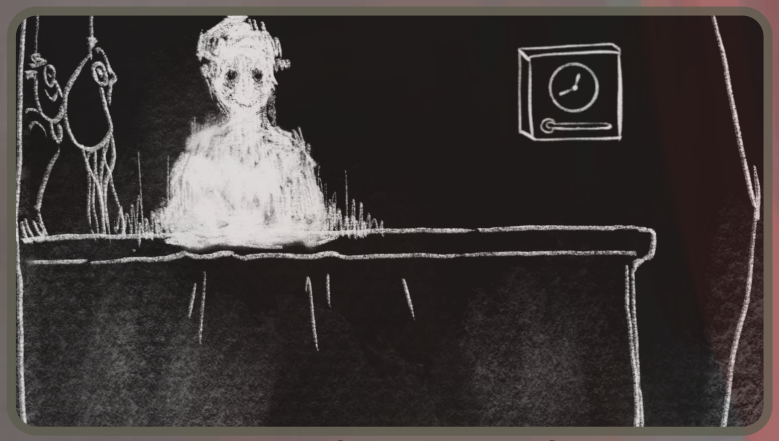


Chef

A mystery chef in the restaurant. He is collecting the customer satisfaction survey form.

Wine

Captain's favourite wine.



Restaurant After an Electrical Short

Cake

The hungry woman want to eat this.

Plate Cover

A bone is hidden below it.

Socket

As the area gets flooded, the restaurant experiences a power outage caused by an electrical short.

Secret Under the Tablecloth

If the player try to lift the tablecloth, they will discover a knife and chef legs shackled in handcuffs.

Clock Puzzle

As the restaurant plunges into darkness, a lockbox featuring a clock puzzle emerges. By setting the clock to the current time (10 am) as instructed by customer service, the lockbox unlocks, revealing a bottle of moldy spicy sauce inside.

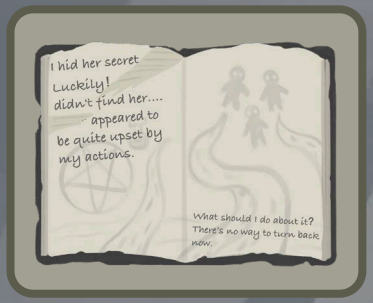
Trash Can

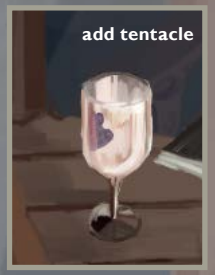
Here, player can discover a fragment of the poster.

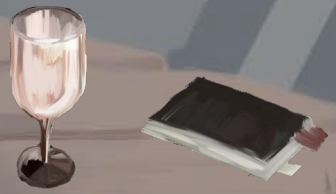
The tentacles of the enigmatic being have been severed

Captain's Notebook

The captain's wicked intentions come to light. Is he planning to sacrifice the ship's guests to the enigmatic being in hopes of a trade-off?







Glass

Craft a poisoned concoction of red wine for the malevolent captain. Mixing in different ingredients will yield distinct alterations to the drink.

Captain is drinking

Captain was poisoned to death

Evil Captain

Lifeboat

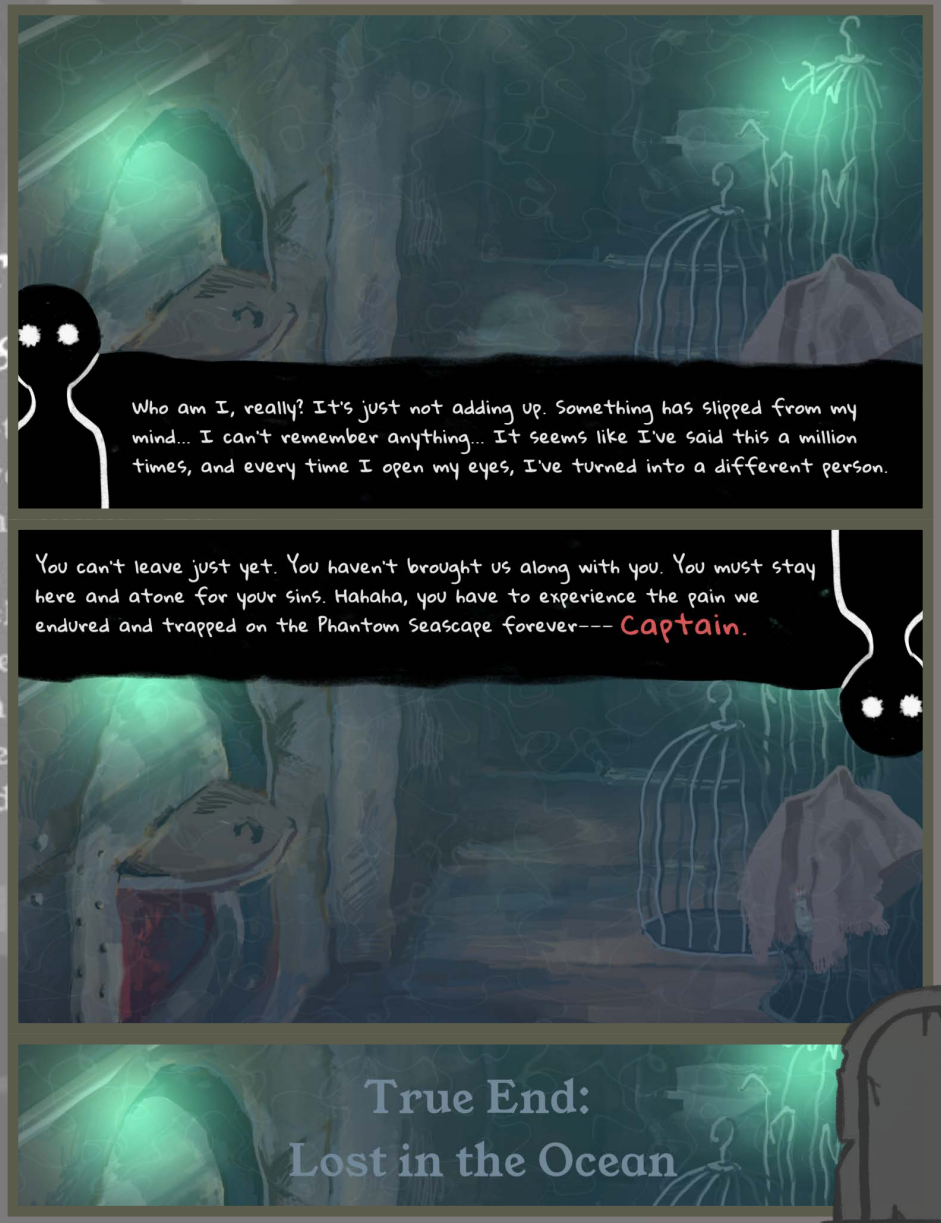
The only lifeboat on the ship, reserved exclusively for the captain.

Game Endings

I created two types of endings. If the player fails to escape from the Phantom Seascape and drowns in the water, they are presented with the life narrative of one randomly chosen individual who met their end aboard the ship. There are ten unique outcomes, with the deceased's roles ranging from fisherman and nobleman to sailor, writer, farmer, and so on.


```
public class EndGameDescription : MonoBehaviour

public TMP_Text DName;
public TMP_Text DDate;
public TMP_Text DDescription;

private string[] Names = new[] { "Thomas Whitman", "James Fletcher", "Margaret Belle", "Alexander Grant", "Evelyn Stoker",
    "Robert Keene", "Geoffrey Hayes", "Harrison Loomis", "Beatrice Dowd", "Daniel Stone " };
private string[] Dates = new[] { "34", "28", "20", "30", "27", "40", "19", "36", "18", "45" };
private string[] Describs = new[] { "     A curious apothecary who dreamt of discovering exotic herbs from distant lands. His dream turned into a nightmare
    "     A courageous fisherman from the coastal town of Cornwall. Driven by the tales of treasure, he fell prey to Captain Thorn's deceit. The captain had hi
    "     The daughter of a nobleman, Margaret sought adventure beyond the aristocratic life. Enticed by the promises of exotic voyages, she found herself a
    "     A seasoned sailor from Liverpool, Alexander was desperate to prove his mettle. He was cold-bloodedly murdered by Captain Thorn who then fed his
    "     An aspiring writer, Evelyn was charmed by the tales of treasures and mysteries of the sea. Captain Thorn locked her in a cage and left her to starve, h
    "     A blacksmith seeking fortune to support his ailing wife. He drowned amid a staged shipwreck orchestrated by the malevolent Captain Thorn, the ech
    "     The young bard seeking inspiration for his magnum opus got entrapped in Thorn's treacherous tales. His voice was silenced forever as he was drown
    "     A cartographer seeking to map uncharted waters, Thorn's deceit led him to a gruesome end. Strangled by the captain in a heated confrontation, his
    "     A girl from a small village, the allure of an adventurous life led her to the deathly grips of Captain Thorn. She met her end falling from the ship, her s
    "     The humble farmer hoping for a fortune to save his drought-ridden farm was betrayed by Thorn. Drowned by the captain, his dreams sunk into the

public int randomNum;
Unity 消息|0 个引用
void Start()
{
    RandomEnd();
}
1 个引用
public void RandomEnd()
{
    randomNum = Random.Range(0, 10);
    DName.text = Names[randomNum];
    DDate.text = "Age: " + Dates[randomNum];
    DDescription.text = Describs[randomNum];
}
```

Upon poisoning the malevolent captain and fleeing the Phantom Seascape via a lifeboat, players are confronted with the true ending. Yet, the escape is illusory. The twist reveals that the player is, in fact, the captain, ensnared in an eternal loop on the ship. Doomed to perpetually experience the last moments of all those he once left behind.



Background Story

Once upon a time, in a small coastal town, rumors ran wild about a ghostly ship named the Phantom Seascape. The ship was known far and wide for its luxurious exterior that promised adventure and treasure. However, only those who had dared to step aboard knew the eerie reality that lay within. Its haunting legend began with its notorious captain, who was known to the townsfolk as Captain X. The captain was famed for his unyielding courage in the face of tempests and his uncanny ability to always return laden with treasure from every voyage, often with a new tale of gold and glory.

But unbeknownst to the world, a sinister secret brewed within the heart of Phantom Seascape. The ship carried the souls of the ambitious treasure seekers who were beguiled by the captain's promise of wealth. However, their greed became their undoing. Captain X had made a nefarious pact with the sea god, Cthulhu, offering the lives of his crew in exchange for protection and treasure from the abyss. Each voyage, the unsuspecting adventurers met with a grisly fate as they were either murdered or drowned by the merciless captain, their spirits forever trapped within the phantom vessel, swirling in a dance of despair and vengeance.

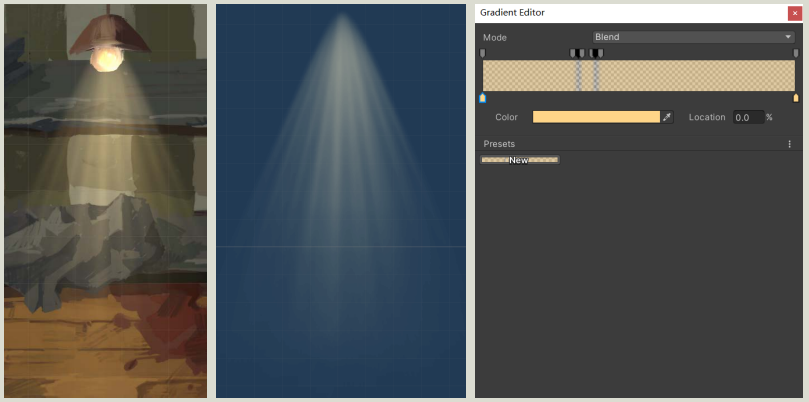
One eventful voyage, a woman aboard the ship caught the captain's eye. Their fleeting romance birthed a child, yet her presence and the resultant infant altered the sinister pact the captain had with Cthulhu. The next time the Phantom Seascape set sail, the wrath of the sea god was a tempest that knew no mercy. It sought out the captain, claimed his life, and thereby shattered the pact. The merciless captain's spirit was now bound to the very ship where he committed his dark deeds.

Years passed and tales of the Phantom Seascape turned into ghostly whispers among the townsfolk, yet its legend rekindled when a curious soul stumbled upon it anchored at the eerie shores. As the adventurous spirit, armed with nothing but desperation to escape the mundane 9 to 5 grind.

Special Effects

Here are some special effects we adopted in the game.

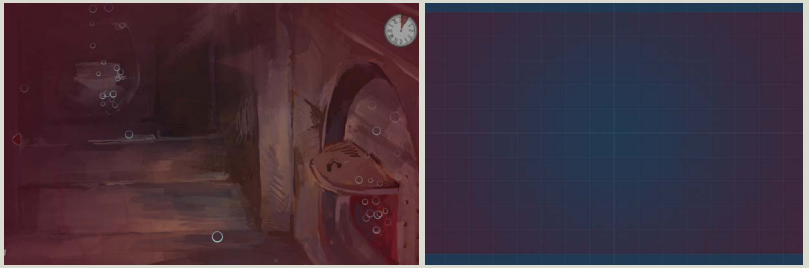
Lamp



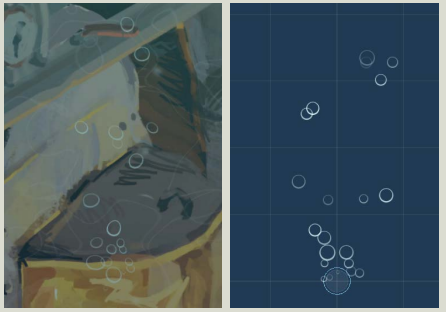
Ghost

Dust

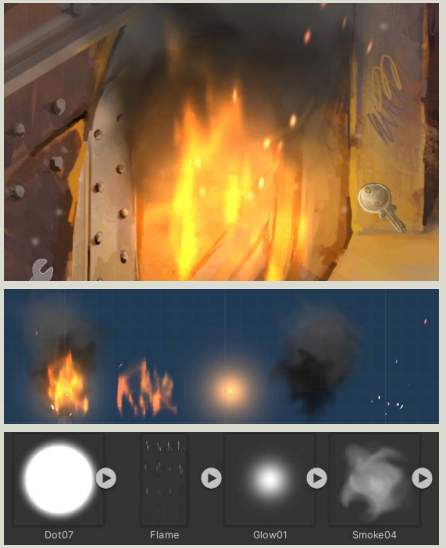
Drowning



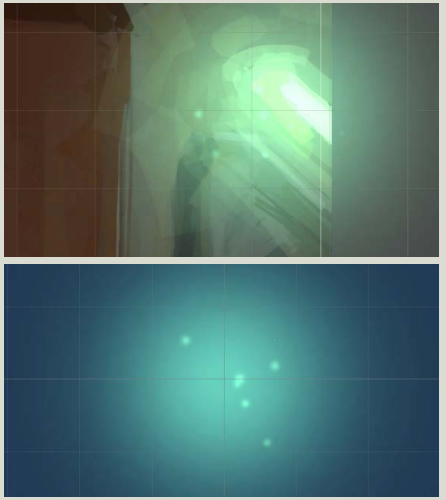
Bubbles



Fire



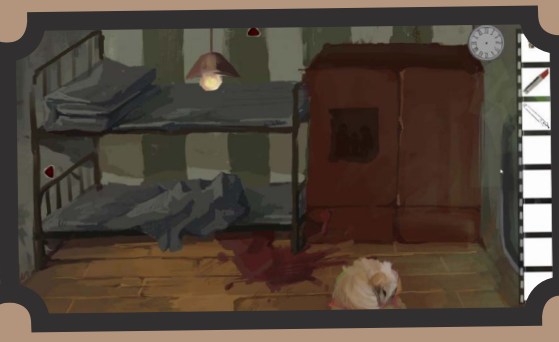
Fluorescence

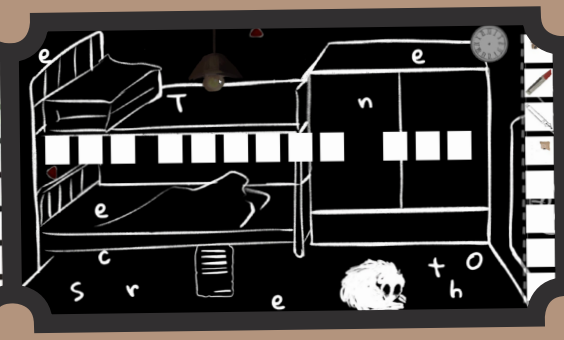


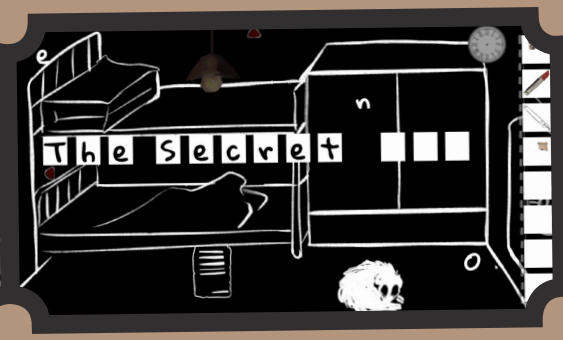
Gallery

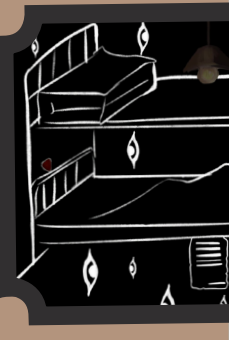

Phantom Seascape
Start
Exit

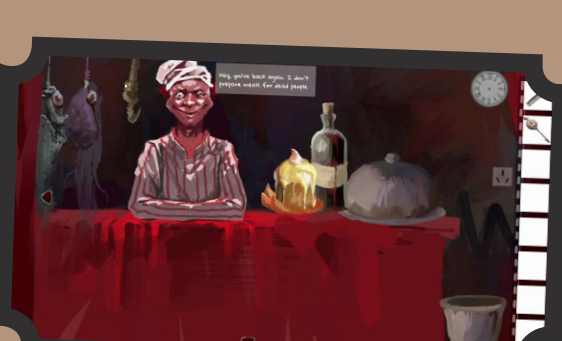

It seems that I've been here before...why I can't remember anything...

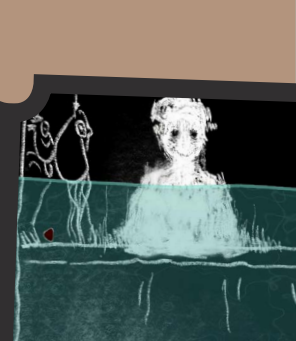


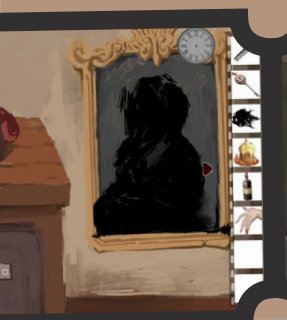


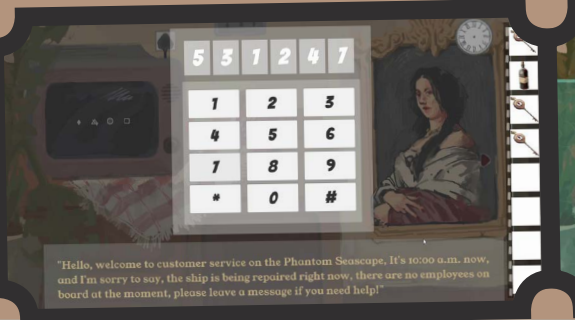

The Secret










5 3 1 2 4 7
1 2 3
4 5 6
7 8 9
* 0 #
"Hello, welcome to customer service on the Phantom Seascape, It's 10:00 a.m. now, and I'm sorry to say, the ship is being repaired right now, there are no employees on board at the moment, please leave a message if you need help!"

Cats Crossing Guard

Arcade game
Made with Vivado

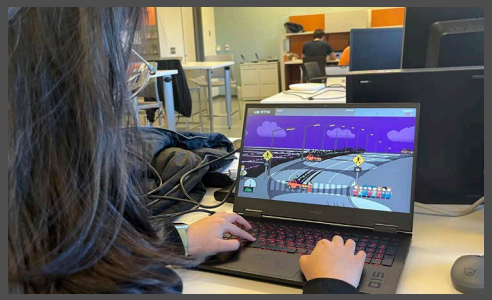
Abstract

In 'Cats Crossing Guard,' players strategically guide cats across a busy highway using an FPGA board. The goal is to safely navigate as many cats as possible past fast-moving cars to the other side. Timing and strategy are crucial in this fast-paced game, demanding quick reflexes and careful planning from players. Each level increases in difficulty, challenging players to continuously adapt their strategies for the cats' safe passage.

Team

Game design & Code:
Nan Kang, Yixiao Fang

Inspiration



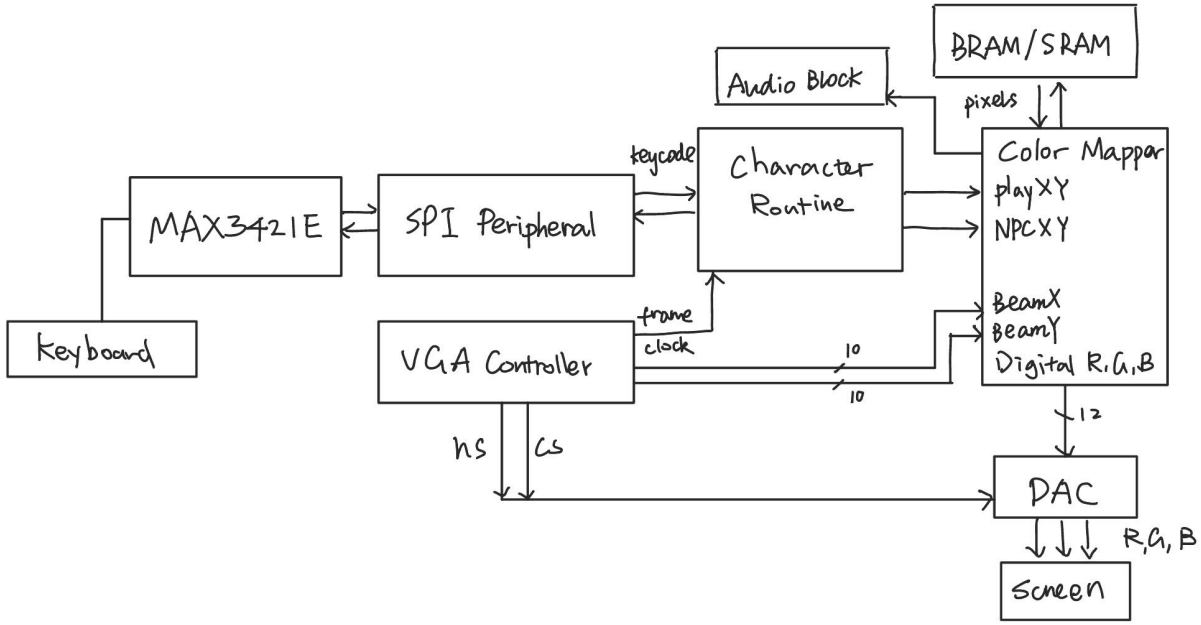
My inspiration came from Crossing Guard Joe. Instead of guiding children across roads in cities where drivers often disregard traffic laws and pedestrians, I believe it's more common for drivers to accidentally hit innocent animals while driving through forests. Through this simple arcade game, we hope players will become more cautious and avoid hitting animals crossing the road.

Core Mechanic

Apart from moving, players have two types of interactions to control NPCs within a certain range. If the player shows a 'go' sign, NPCs in range will follow the instruction and continue crossing the highway; however, if the player shows a 'stop' sign, they will stop immediately. If NPCs are out of the control range, they will maintain their current state, either continuing to move until reaching the other side of the road or staying in their current position.

Techinical Details

Our design is entirely based on the Spartan7 CPU and its peripherals on the provided FPGA board. We will utilize the MicroBlaze block design to construct the SoC. The block components will include, but are not limited to, the system bus, RAM, SPI peripherals, VGA controller, VGA to HDMI module, and basic I/Os. Our goal is to demonstrate a playable game using a USB keyboard and HDMI monitor.



block diagram

SPI Peripheral and GPIO

MicroBlaze can read input and output signals through GPIO, where inputs come from the FPGA board, and outputs can be displayed on the board. GPIO can be configured as input, output, or both, allowing the MicroBlaze processor to interact with external devices. Within the GPIO blocks, the structure consists of several control registers that store the input and output data. These control registers are memory-mapped to addresses assigned by the block design editor.

VGA Operation

Essentially, four modules are involved in the VGA operation: 'VGA_controller.sv', 'player_controller.sv', 'animals_controller.sv', and 'Color_Mapper.sv'. Within the block diagram context, 'player_controller.sv' and 'animals_controller.sv' are categorized as part of the Character Routine.

The VGA_controller provides four essential signals for drawing shapes on the screen: hsync, vsync, drawX, and drawY. The hsync and vsync signals control the horizontal and vertical drawing rates on the screen, respectively. As the electron beam moves horizontally from left to right and vertically from top to bottom to display pixels, hsync becomes high at the start of a new row, and vsync becomes high at the start of a new column. The drawX and drawY signals indicate the current pixel being drawn on the screen.

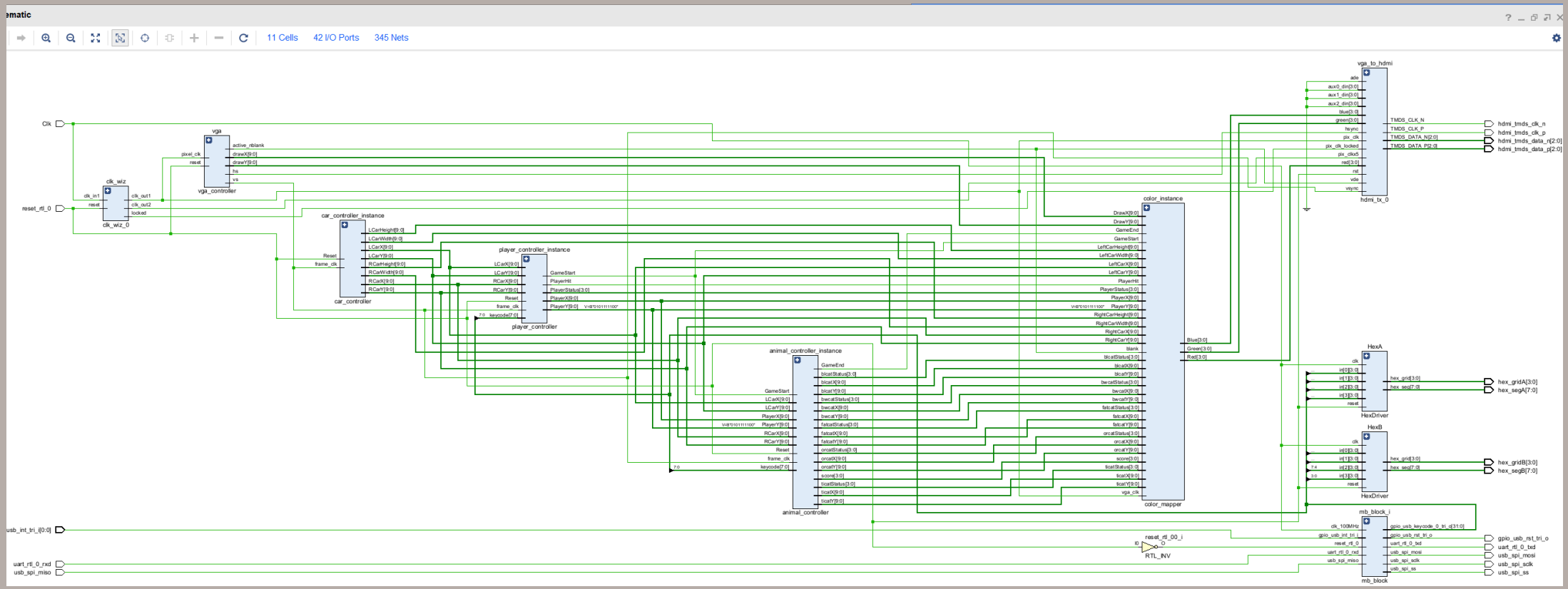
With drawX and drawY, we can determine the RGB value of the current pixel based on the positions of the cats and player on the screen, which are determined by the Character Routine reading from keyboard codes. As multiple objects need to be displayed on the screen, Color_Mapper.sv includes a MUX to select which object occupies the current pixel. The ROMs storing character sprites are accessed directly in the color mapper, with corresponding addresses determined by drawX, drawY, and the top-left pixel position of the character.

Finally, the hsync, vsync, and RGB signals are all outputted to the actual VGA to HDMI converter to draw the actual pixel on the screen.

ROMs for Sprite Storage

All sprites are stored in separate ROMs, which are provided IPs in Vivado. We convert desired images into .COE files and load them directly into the ROMs. To display a sprite without a background, we use a distinct hot pink color for the background, which is omitted during rendering. Initially, the sprites displayed with a pink border due to similar shades in the image processing. We refined the code to select the most common colors from the original image, avoiding any averaging of pixel intensities.

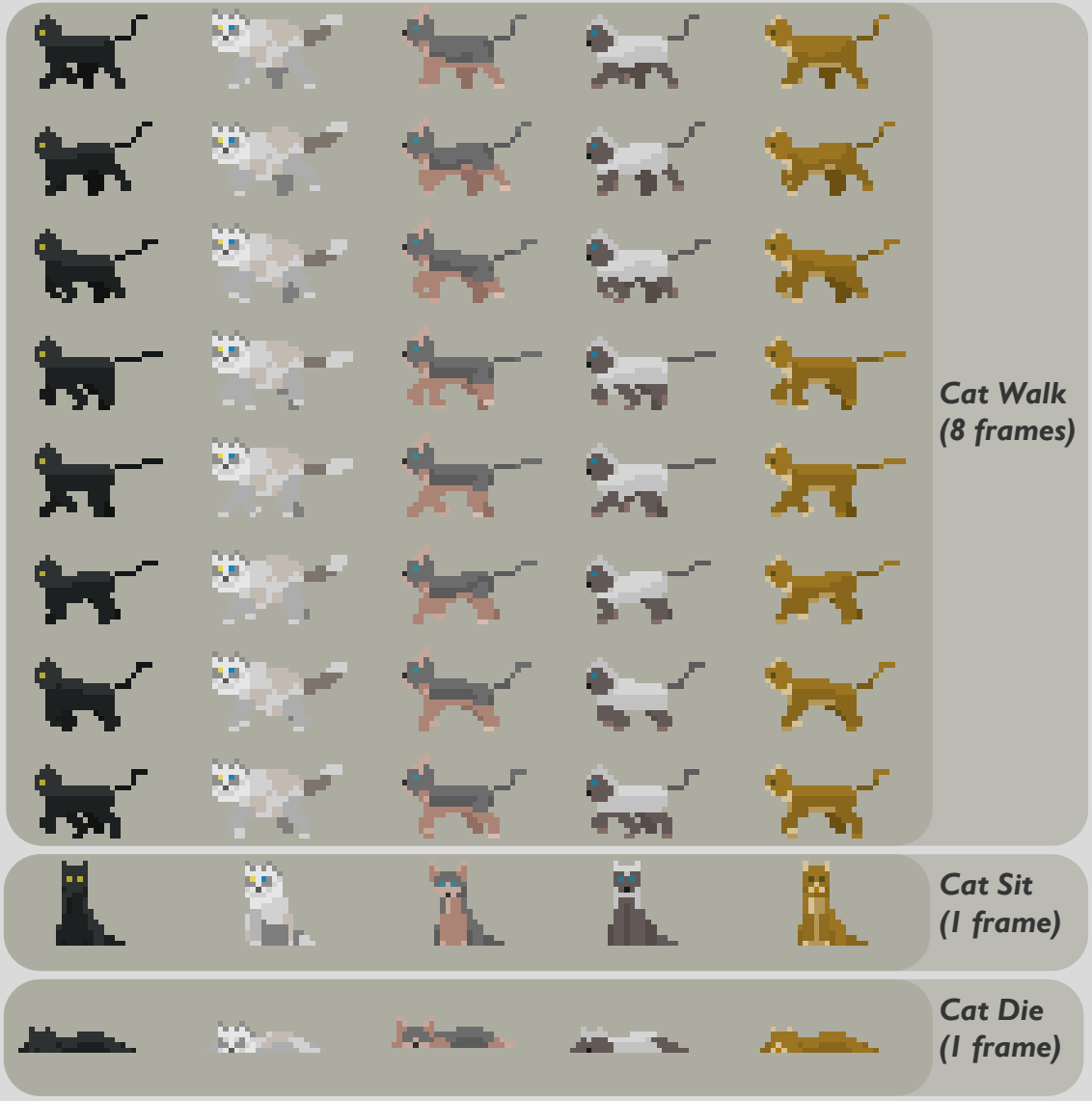
For characters with animations, we store all potential poses in a single ROM by concatenating them. This allows us to access different character states based on designed logic, reducing the need for multiple ROMs.



Spotlight Features

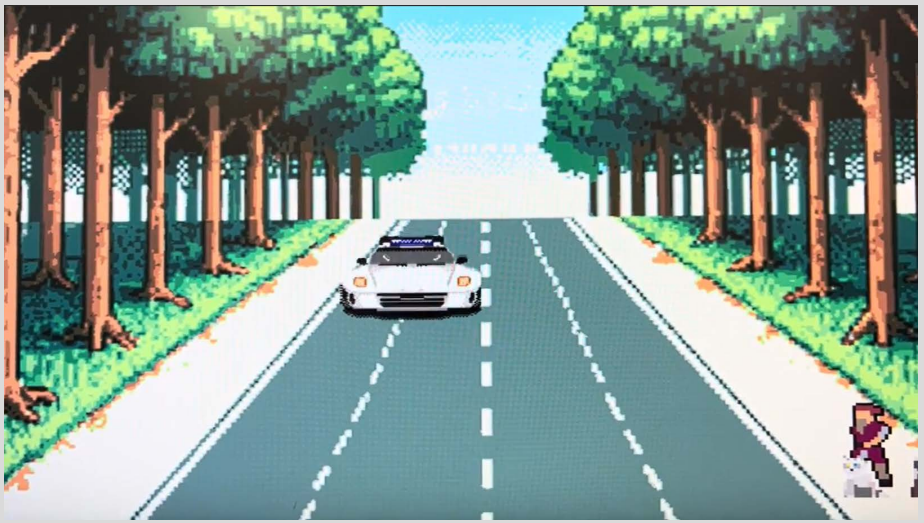
Character Animation

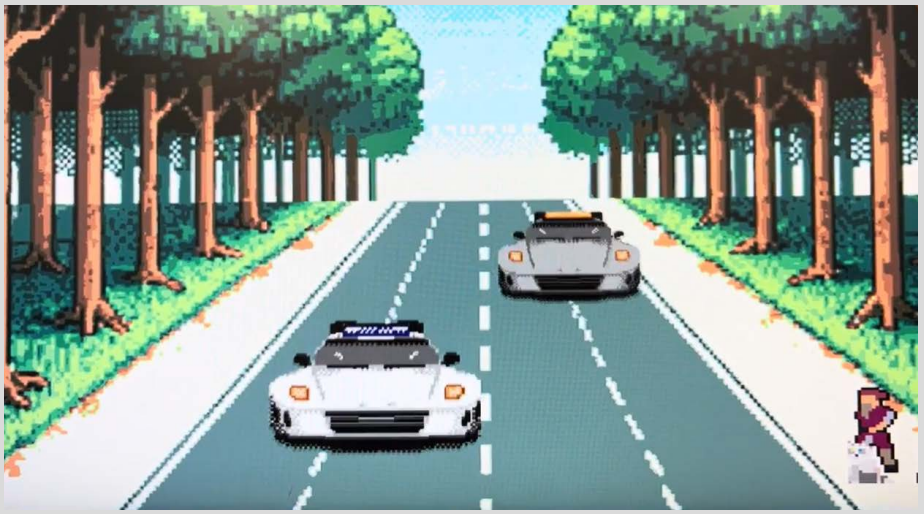
To animate cats and players, we use a combination of stored poses in ROMs and logic to alternate sprite displays. Characters have various poses, like a cat's walking, sitting, or being hit, totaling eight poses. The game identifies the cat's status (walking, sitting, or being hit) through signals generated by player_controller.sv or animals_controller.sv. For walking animations, a counter tracks the number of frames since the cat started walking, updating the pose with each clock's rising edge to create the animation effect.



2.5D Visual Effect of the Cars

Regarding the 3D effect of the cars, specifically, the cars appear larger as they approach the players and cats. The car will show up in the middle of the screen and then move down gradually. To achieve this effect, we first determined a range within which the cars can move. Then, we calculated how the height and width change with each specific value of the car's y-position change. We also adjusted the car's moving route to ensure it moves along the middle of the highway.





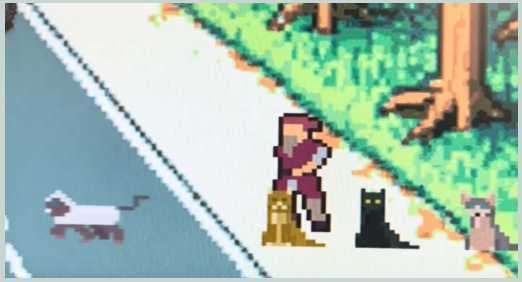
NPC control during the waiting stage and crossing stage

Players can control cats within a specific range; beyond that, cats act based on their position. As movement is along the x-axis, we monitor their x-axis positions to set statuses. Initially, player and cat speeds were equal, but to reflect the player's relinquished control after releasing cats, we doubled the player's speed. To add complexity, we spaced cats apart initially. Once the first cat leaves the waiting zone, the cats behind will walk to their next left position and then stop and wait to be guided. To achieve this, each cat should check the previous cat's position and their status will be set as walk status temporarily before they reach the desired waiting position.

1. All cats sit at waiting zone

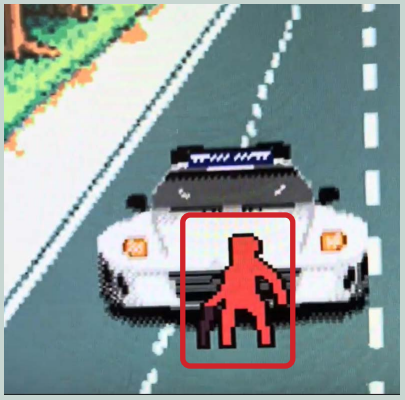
2. The first cat leave the waiting zone, the subsequent cats move to their waiting points

3. The first cat leave the waiting zone, the subsequent cats reach their waiting points



4. The second cat leave the waiting zone, the subsequent cats move to their waiting points

Collision between Characters and Cars



If the player is hit by a car, they turn red and cannot move until the car passes. If a cat is hit, it will die and move along with the car.

Score

The upper left corner of the screen displays a score, indicating how many cats have successfully crossed to the other side of the highway. The score is calculated in the animal controller module by evaluating if the cat is hit and comparing their x position.

Code *(system verilog)*

animal_controller.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/new/animal_controller.sv

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////////


module animal_controller(
    input logic Reset, frame_clk,GameStart,
    input logic [7:0] keycode,
    input logic [9:0]  LCarX, LCarY, RCarX, RCarY,
    input logic [9:0]  PlayerX, PlayerY,
    output logic [9:0] fatcatX, fatcatY,
    output logic [9:0] bwcatX, bwcatY,
    output logic [9:0] orcatX, orcatY,
    output logic [9:0] blcatX, blcatY,
    output logic [9:0] ticatX, ticatY,
    output logic [3:0] fatcatStatus, bwcatStatus, orcatStatus, blcatStatus, ticatStatus, score,
    output logic GameEnd
    );

    logic [9:0] fatcat_X_Motion;
    logic fatcatWalk;
    logic fatcatHit;
    logic [3:0] fatcat_Walk_counter;
    logic [5:0] fatcat_motion_counter;

    parameter [9:0] fatcat_X_Size = 60; // cat sprite width
    parameter [9:0] fatcat_X_Max = 600; // Rightmost location for the player on X axis
    parameter [9:0] fatcat_X_Min = 10 ; // Leftmost location for the player on X axis
    parameter [9:0] Right_Car_Y_Max = 400; // Right Car reach hit zone
    parameter [9:0] Right_Car_Y_Min = 340 ; // Right Car reach hit zone

    //assign fatcatY = 10'd420;

    always_ff @ (posedge frame_clk or posedge Reset) //make sure the frame clock is instantiated correctly
    begin: Move_Ball
        if (Reset)  // asynchronous Reset
            begin
                score <= 4'd0;

                fatcat_X_Motion <= 10'd1;
                fatcatX <= 10'd519;
                fatcatY <= 10'd420;
                fatcatStatus <= 4'd0;
                fatcatWalk <= 1'b1;
                fatcatHit <= 1'b0;
                fatcat_Walk_counter <= 4'b0;
                fatcat_motion_counter <= 6'd4;
        end
        else
            begin
                //fatcat section begin
                if (fatcatHit == 1'b1)
                    begin
                        if (fatcatY < 10'd800)
                            begin
                            fatcatStatus = 4'd0;
                            fatcatY = fatcatY + 1;
                            end
                    end

                else if (fatcatX == 10'd1)
                    begin
                        score = score + 1;
                        fatcatX = 10'd0;
                    end
                //reach other side
                else if (fatcatX < 10'd1)
                    begin
                        fatcatStatus = 4'd1;
                    end

                else
                    begin
                    fatcatY <= 10'd420;
                    //check collision between right car and player
                    if ((fatcatX > 10'd320 ) && (fatcatX < 10'd460) && (RCarY > Right_Car_Y_Min) && (RCarY <
                        begin
                            fatcatHit <= 1'b1;
                        end
                    //check collision between left car and player
                    else if ((fatcatX > 10'd160 ) && (fatcatX < 10'd300) && (LCarY > Right_Car_Y_Min) && (LC
                        begin
                            fatcatHit <= 1'b1;
                        end
                    else
                        begin
                            fatcatHit <= 1'b0;
                        end

                    //player input
                    if ((GameStart == 1'b1)&&((PlayerX - fatcatX < 10'd80) || (fatcatX - PlayerX < 10'd80)))
                        begin
                            if (keycode ==  8'h1A) //W:Go
                                begin
                                    fatcatWalk = 1'd1;
                                end
                            else if (keycode ==  8'h16) //S:Stop
                                begin
                                    fatcatWalk = 1'd0;
                                end
                        end

                    //cat walk
                    if ((fatcatWalk == 1'b1) && (fatcatX < 10'd840))
                        begin
                            fatcatX = fatcatX - fatcat_X_Motion;
                        end
                    else if (fatcatWalk == 1'b0)
                        begin
                            fatcatX = fatcatX;
                            fatcatWalk = 1'd0;
                        end
                    else
                        begin
                            fatcatX = 10'd0;
                            fatcatWalk = 1'd0;
                        end
```

```
                    //cat animation
                    if ((fatcatWalk == 1'b1)&&(fatcat_motion_counter == 6'd0))
                        begin
                            if (fatcat_Walk_counter > 4'd0)
                                begin
                                    fatcatStatus = fatcat_Walk_counter;
                                    fatcat_Walk_counter = 4'd0;
                                end
                            else
                                begin
                                    fatcatStatus = fatcat_Walk_counter;
                                    fatcat_Walk_counter = fatcat_Walk_counter + 1;
                                end

                            fatcat_motion_counter = 10'd0;
                        end
                    else if ((fatcatWalk == 1'b1)&&(fatcat_motion_counter != 6'd4))
                        begin
                            fatcat_motion_counter = fatcat_motion_counter + 6'd1;
                            fatcatStatus = fatcatStatus;
                            fatcat_Walk_counter = fatcat_Walk_counter;
                        end
                    else //not walking
                        begin
                            fatcat_motion_counter = 6'd4;
                            fatcatStatus = 4'd1;
                            fatcat_Walk_counter = 4'b0;
                        end
                end
                // fatcat section end
```

player_controller.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/new/player_controller.sv

```
module player_controller(
    input logic Reset, frame_clk,
    input logic [7:0] keycode,
    input logic [9:0]  LCarX, LCarY, RCarX, RCarY,
    output logic [9:0]  PlayerX, PlayerY,
    output logic PlayerHit,GameStart,
    output logic [3:0] PlayerStatus
);
//status: 0 - walk; 1 - hurt; 2 - go; 3 - stop;
    logic [9:0] Player_X_Motion;
    logic PlayerWalk;
    logic [3:0] Walk_counter;
    logic [5:0] motion_counter;
    logic enable_KB;

    parameter [9:0] Player_X_Size = 50; // Player sprite width
    parameter [9:0] Player_X_Max = 610; // Rightmost location for the player on X axis
    parameter [9:0] Player_X_Min = 10 ; // Leftmost location for the player on X axis
    parameter [9:0] Right_Car_Y_Max = 400; // Right Car reach hit zone
    parameter [9:0] Right_Car_Y_Min = 320 ; // Right Car reach hit zone

    assign enable_KB = 1'b1; //default state: accept KB input
    assign PlayerY = 10'd380;

    always_ff @ (posedge frame_clk or posedge Reset) //make sure the frame clock is instantiated correctly
    begin: Move_Ball
        if (Reset)  // asynchronous Reset
        begin
            GameStart <= 1'b0;
            Player_X_Motion <= 10'd2;
            PlayerX <= 10'd300;
            PlayerHit <= 1'b0;
            PlayerStatus <= 3'd4;
            PlayerWalk <= 1'b0;
            Walk_counter <= 4'b0;
            motion_counter <= 6'd0;
        end
        else
        begin
            if (PlayerHit == 1'b0)
            begin

                if (PlayerX + Player_X_Size > Player_X_Max )  // Player reach rightmost point
                    begin
                      PlayerX <= Player_X_Max - Player_X_Size;
                    end

                else if ( PlayerX < Player_X_Min )  // Player reach leftmost point
                    begin
                      PlayerX <= Player_X_Min;
                    end

                else if (keycode ==  8'h44) //space
                    begin
                        GameStart <= 1'b1;
                    end
                else if ((keycode ==  8'h41)&&(GameStart == 1'b1)) //A
                    begin
                        PlayerX <= PlayerX - 10'd2;
                        PlayerWalk <= 1'b1;
                    end
                else if ((keycode == 8'h27)&&(GameStart == 1'b1)) //D
                    begin
                        PlayerX <= PlayerX + 10'd2;
                        PlayerWalk <= 1'b1;
                    end

                else if ((keycode ==  8'h1A)&&(GameStart == 1'b1)) //W : Go
                    begin
                        PlayerStatus <= 3'd2;
                        PlayerWalk <= 1'b0;
                    end
                else if ((keycode == 8'h16)&&(GameStart == 1'b1)) //S : Stop
                    begin
                        PlayerStatus <= 3'd3;
                    end
                //check collision between right car and player
                if ((PlayerX > 10'd300 ) && (PlayerX < 10'd440) && (RCarY > Right_Car_Y_Min) && (RCarY < Right_C
                    begin
                        PlayerHit <= 1'b1;
                    end
                //check collision between left car and player
                else if ((PlayerX > 10'd140 ) && (PlayerX < 10'd280) && (LCarY > Right_Car_Y_Min) && (LCarY < Ri
                    begin
                        PlayerHit <= 1'b1;
                    end
                else
                    begin
                        PlayerHit <= 1'b0;
                    end
```

car_controller.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/new/car_controller.sv

```
module car_controller(
    input logic Reset, frame_clk,
    output logic [9:0]  LCarX, LCarY,RCarX, RCarY,
    output logic [9:0]  LCarWidth, LCarHeight, RCarWidth, RCarHeight
);

    logic [9:0] LCar_Y_Motion;
    logic [9:0] RCar_Y_Motion;

    parameter [9:0] LCar_X_Size = 82; // Car sprite width
    parameter [9:0] LCar_Y_Size = 66; // Car sprite height
    parameter [9:0] RCar_X_Size = 82; // Car sprite width
    parameter [9:0] RCar_Y_Size = 66; // Car sprite height
    parameter [9:0] Car_Y_Max = 500; // Bottom location for the car on Y axis
    parameter [9:0] Car_Y_Min = 200 ; // Top location for the car on Y axis

    always_ff @ (posedge frame_clk or posedge Reset) //make sure the frame clock is instantiated correctly
    begin: Move_Ball
        if (Reset)  // asynchronous Reset
        begin
            RCar_Y_Motion <= 10'd1; // Car_Y_Step
            RCarY <= 10'd200;
            RCarX <= 10'd320;

            LCar_Y_Motion <= 10'd1; // Car_Y_Step
            LCarY <= 10'd400;
            LCarX <= 10'd260;
        end

        else
        begin
                // Right Car
                if (RCarY > 10'd700 )  // Car reach bottom point
                    begin
                        RCarX = 10'd320;
                        RCarY = Car_Y_Min;
                        RCarWidth = 10'd80;
                        RCarHeight = 10'd48;
                    end

                else
                    begin
                        RCarY = RCarY + RCar_Y_Motion;
                        RCarX = ((RCarY - 10'd200)/6) + 10'd320;
                        RCarWidth = ((RCarY - 10'd300)>>1) + 10'd80;
                        RCarHeight = ((RCarY - 10'd200)>>1) + 10'd48;
                    end

                // Left Car
                if (LCarY > 10'd600 )  // Car reach bottom point
                    begin
                        LCarX = 10'd260;
                        LCarY = Car_Y_Min;
                        LCarWidth = 10'd80;
                        LCarHeight = 10'd48;
                    end

                else
                    begin
                        LCarY = LCarY + LCar_Y_Motion;
                        LCarX = 10'd260 - ((LCarY - 10'd200)/10'd3);
                        LCarWidth = ((LCarY - 10'd300)>>1) + 10'd80;
                        LCarHeight = ((LCarY - 10'd200)>>1) + 10'd48;
                    end
        end
    end

endmodule
```

Color_Mapper.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/imports/design_source/Color_Mapper.sv

```
module  color_mapper ( input  logic [9:0] DrawX, DrawY,
                       input logic vga_clk,
                       input logic blank,
                       input logic [9:0] PlayerX, PlayerY,
                       input logic [9:0] fatcatX, fatcatY,
                       input logic [9:0] bwcatX, bwcatY,
                       input logic [9:0] orcatX, orcatY,
                       input logic [9:0] blcatX, blcatY,
                       input logic [9:0] ticatX, ticatY,
                       input logic PlayerHit,
                       input logic [3:0] PlayerStatus, fatcatStatus, bwcatStatus,orcatStatus, blcatStatus, t
                       input logic [9:0] LeftCarX, LeftCarY,LeftCarWidth, LeftCarHeight,
                       input logic [9:0] RightCarX, RightCarY,RightCarWidth, RightCarHeight,
                       input logic [3:0] score,
                       input logic GameStart,GameEnd,
                       output logic [3:0]  Red, Green, Blue
                       );
    logic [3:0] R_bg, G_bg, B_bg;
    logic [3:0] R_sc, G_sc, B_sc;
    logic [3:0] R_num, G_num, B_num;
    logic [3:0] R_car, G_car, B_car;
    logic [3:0] R_car_l, G_car_l, B_car_l; // Left Car RGB
    logic [3:0] R_player, G_player, B_player;
    logic [3:0] R_fatcat, G_fatcat, B_fatcat;
    logic [3:0] R_bwcat, G_bwcat, B_bwcat;
    logic [3:0] R_orcat, G_orcat, B_orcat;
    logic [3:0] R_blcat, G_blcat, B_blcat;
    logic [3:0] R_ticat, G_ticat, B_ticat;
    logic [3:0] R_title, G_title, B_title;
    logic [3:0] R_end, G_end, B_end;
    logic ena_bg;
    logic ena_sc;
    logic ena_num;
    logic ena_car;
    logic ena_car_l; // Left Car Enable
    logic ena_player;
    logic ena_fatcat;
    // Used for hit flashing effect
    logic flash_count;

    always_comb
    begin
        if ((GameStart == 1'b0)&&(ena_title == 1'b1)&&(R_title != 4'hf || G_title != 4'h0 || B_title != 4'hf
            begin
                Red = R_title;
                Green = G_title;
                Blue = B_title;
            end
        else if ((GameEnd == 1'b1)&&(ena_end == 1'b1))&&(R_end != 4'hf || G_end != 4'h0 || B_end != 4'hf))
            begin
                Red = R_end;
                Green = G_end;
                Blue = B_end;
            end
```

greycar_example.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/imports/greycar/greycar_example.sv

```
module greycar_example (
    input logic vga_clk,
    input logic [9:0] DrawX, DrawY,
    input logic blank,
    input logic [9:0] PosX, PosY,RightCarWidth, RightCarHeight,
    output logic [3:0] red, green, blue,
    output logic ena
);

logic [12:0] rom_address;
logic [3:0] rom_q;

logic [3:0] palette_red, palette_green, palette_blue;
logic [20:0] calc_temp; //used to hold temporary data

logic negedge_vga_clk;

logic [9:0] DistX, DistY;
assign DistX = DrawX - PosX;
assign DistY = DrawY - PosY;

// read from ROM on negedge, set pixel on posedge
assign negedge_vga_clk = ~vga_clk;

// address into the rom = (x*xDim)/640 + ((y*yDim)/480) * xDim
// this will stretch out the sprite across the entire screen

always_comb
begin
    if (DistX < RightCarWidth && DistY < RightCarHeight)
        begin
            calc_temp =  ((( DistY * 21'd96) / RightCarHeight)) + 21'd82 + (((DistX * 21'd82) / RightCarWidth)
            rom_address = calc_temp;
            ena = 1'b1;
        end
    else
      begin
        rom_address = 13'd0;
        ena = 1'b0;
      end
end


always_ff @ (posedge vga_clk) begin
    red <= 4'h0;
    green <= 4'h0;
    blue <= 4'h0;

    if (blank) begin
        red <= palette_red;
        green <= palette_green;
        blue <= palette_blue;
    end
end

greycar_rom greycar_rom (
    .clka   (negedge_vga_clk),
    .addra (rom_address),
    .douta      (rom_q)
);

greycar_palette greycar_palette (
    .index (rom_q),
    .red   (palette_red),
    .green (palette_green),
    .blue  (palette_blue)
);

endmodule
```

greycar_palette.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/imports/greycar/greycar_palette.sv

```
module greycar_palette (
    input logic [3:0] index,
    output logic [3:0] red, green, blue
);

localparam [0:8][11:0] palette = {
    {4'h5, 4'h5, 4'h5},
    {4'hF, 4'hF, 4'hF},
    {4'h0, 4'h0, 4'h0},
    {4'h0, 4'h0, 4'h0},
    {4'h5, 4'h5, 4'h5},
    {4'hF, 4'hF, 4'hF},
    {4'h5, 4'h5, 4'h5},
    {4'hF, 4'hF, 4'hF}
};

assign {red, green, blue} = palette[index];

endmodule
```

mb_usb_hdmi_top.sv

D:/D/Fall2023/ECE385/lab6/lab6 srcs/sources_1/imports/design_source/mb_usb_hdmi_top.sv

```
module mb_usb_hdmi_top(
    input logic Clk,
    input logic reset_rtl_0,

    //USB signals
    input logic [0:0] gpio_usb_int_tri_i,
    output logic gpio_usb_rst_tri_o,
    input logic usb_spi_miso,
    output logic usb_spi_mosi,
    output logic usb_spi_sclk,
    output logic usb_spi_ss,

    //UART
    input logic uart_rtl_0_rxd,
    output logic uart_rtl_0_txd,

    //HDMI
    output logic hdmi_tmds_clk_n,
    output logic hdmi_tmds_clk_p,
    output logic [2:0]hdmi_tmds_data_n,
    output logic [2:0]hdmi_tmds_data_p,

    //HEX displays
    output logic [7:0] hex_segA,
    output logic [3:0] hex_gridA,
    output logic [7:0] hex_segB,
    output logic [3:0] hex_gridB
    );
```