MP Survey Completion Evidence

Aliah	Temasek POLYTECHNIC Thank you for your feedback. Response ID: 96487753 You will receive an acknowledgement email with your response ID. Do check your Inbox/Junk/Spam folder.
Fabian	Dear student, Thank you for completing the MP Student survey. Please include the Response ID below in your MP Report, and save this email for future reference. Response ID: 96650808 Best Regards ENG MP Admin Support Response ID: 96650808
Rynie	Temasek POLYTECHNIC Thank you for your feedback. Response ID: 96555596 You will receive an acknowledgement email with your response ID. Do check your Inbox/Junk/Spam folder.

Table of Contents

2M22340 Gamifying Disaster Simulation

Aliah Umairah Reduan

Lim Rui How, Fabian

Nur Aishuarynie A'qidah

Supervisor: Mr Tommy Tai

Abstract

National Mass Disaster training is held for medical students so they would know what to do when there is a mass casualty nation-wide. Since COVID struck, national mass disaster training for medical students has come to a halt since they are not able to conduct the training in real life. So they are forced to resort to online methods to conduct the training.

This project is aimed to allow residents to learn on how to help casualties with the proper triage and what they should do in a disastrous setting in a 2D Pixelated Mobile game where they can have fun while learning.

1. INTRODUCTION

1.1 Background

DUKE–NUS Medical School (DUKE-NUS) and Singapore General Hospital (SGH) are working together for this project and they explained to us what they were currently using to teach their medical residential students, a medical residency programme where it transforms a medical school graduate into a full-on doctor, on what they should do when a disaster strikes.

Before COVID-19, the students would have disaster training where they are taught what they should do when there is a national mass disaster and they would be equipped to give patients needing emergency, critical care, and/or burns management with safe, wholesome, and high-quality care. equipped with the necessary knowledge, skills, and experience.

Since COVID-19 struck, it is difficult for the students to have that experience. It would be difficult for them to conduct it in real life back when there were restrictions and even after the lifted restriction, it would be still difficult as it is hard to find the resources for it - finding 20 mannequins for the simulation. So because of this,

the idea of creating a game where the students can learn how to help casualties in a disaster area came about.

1.2 Objectives

To simulate a disaster scenario in a 2D android game. It has interactive hotspots that represent casualties. In the scenario, the player is presented with casualties and would need to perform triage when approached near the hotspots. They would be faced with the problem and be required to select the correct triage in a Multiple-Choice Question format. There will be a scoring system in the game which would be dependent on the players correctness and speed of diagnosis.

2. PROJECT DESCRIPTION

The objective of this project is to simulate a disaster scenario in a 2D Pixelated Android Game. It features interactive hotspots that represent the casualty where when the player approaches the casualty, they will be presented a series of questions for them to answer and they have to answer correctly and timely as they are rushing against time and have to save other casualties in the scene. There are also hazards scattered across the map such as fire and exposed wires which will damage the player. The player is given 5 HP and once they run out of health, it would be game over for them and they would have to restart. There are also bandages scattered across the map so that when the player finds it and picks it up, their HP gets restored and they can use this method as a way to not die/ get injured from the hazards. When the player successfully helped the casualty by answering all the questions in the MCQ correctly, they would receive a point per casualty. There is a total of 20 casualties scattered across the map and there is a system that tracks the scores for them. If the player gets any questions wrong, at the end of the question, there will be a button to display the correct answers so that the students are able to learn and understand what and where they did wrong. There will also be an overall timer of about 10 minutes where the player has to save all the casualties in the map within the given time frame to create a sense of urgency in the game.

3. PROJECT DEVELOPMENT

3.1 Gather Town

[Refer to Appendix O]

Their original game was a website called Gather Town where players, at close proximity, are able to communicate with one another. In the game, casualties are represented by a table so when they are near the casualty, they would need to press ‘X’ to interact, to ‘help’ the injured. They would be presented with a photo and then the host of the game would then question the player on what they should do. There are also hazards such as fire in a form of a firework where if you get hit by it, your points get reduced. It does not have its own score system where when the user “helped” a casualty, they need to keep track of their own scores instead of having the system to track their scores.

3.2 First Step

Since the game is a website, we decided to make a mobile game version of the game where it is made accessible

on an Android phone. We decided to use Unity3D and for designing the graphics of the game, we used Aseprite to design the background and characters. The game is split into 2 sides, coding and design. Coding would refer to the movement of the character, interaction between the player and the Non-Player Character (NPC), the Multiple-Choice Question (MCQ) when the player helps the casualty.

We were given resources beforehand by our supervisor on how to start on a 2D Mobile Game - Ruby Tutorial [1], which helped us greatly in the making of our game. We wanted to keep the 2D Pixelated look of the game that Gather Town had so we based the entire graphics on that. We followed the tutorial for the most part as it applies to what we have to do strongly.

3.3 Designing

3.3.1 Avatar & Characters

[Refer to Appendix P]

We first started designing the main avatar to start off so there was a character we could play around with. The software used for this is Aseprite. The first original designs of the avatar were light blue but the client requested the avatar to be a more 'Singhealth' color, which is a darker shade of blue. The end design is an avatar with dark blue scrub.

For the NPCs, since there are different types of casualties, there are different designs for each casualty depending on what type of injury they have.

3.3.2 Background

[Refer to Appendix Q]

So for the backgrounds, we used tile maps - where you design part of a 32x32 box and you can use that as a template and use the paintbrush tool and design in Unity however you like. For most of the base background such as the sea, sand, planks, grass and roads, they were mainly designed by us, as well as the fences. For the detailing of the map such as the lamp posts, buildings, benches, trees, cars, they were imported from an external website [2] which was free of use and the creator themselves allowed it. This was the best way to design the background as there is a lot more freedom while only designing a small chunk of sprite.

3.4 Game Logic

[Refer to Appendix for codes]

To start the game, proceed to any NPCs and trigger the help button when directly facing the front of the NPCs to represent talking to a real life patient. This will enable the 'help' script to be activated which references the 'AvatarController' script setting off the dialog box which includes all the scenario information the player needs to know to answer the relevant questions. If the player clicks on the button when they are not in front of the NPC, the button will be inaccessible.

The game is coded so that the player is unable to pass through obstacles such as buildings, benches, trees, etc to create a sense of realism. To further establish realism, the hazards present in the game would damage the player's health gradually when the player is within close proximity. It can be seen by the decreasing health bar status and when the bar reaches zero, the game will end. The health bar can be restored before the game ends by picking up the bandage found scattered around the map.

3.5 Meeting & Requests

[*Refer to Appendix S*]
We held a meeting with SGH & DUKE-NUS to showcase what we have done so far. They noted that everything was well beyond their expectations but they also gave some feedback. Firstly, they mentioned that it would be great that the buttons and everything else would not be in the way when the player is answering the question, as it is hard to read the question. Not only that, they added that it would be nice if the buttons, health, timer and volume would not be cramped up in the middle and that it should be more spaced out.
They also requested that when the player is done with the MCQ, they should be provided the correct answers so that they know where they went wrong and learn from their mistakes. They also mentioned that it would be nice if the player is able to know what their score is throughout the entire game. They also wished that there would be a map overview of where the player is, something similar to Figure , so that the player knows where in the map they are. They also feedback that the tutorial button available in the start menu to be easily accessible in the gameplay so that the player can easily refer back to the proper triage whenever they need to.

3.6 Improvements

[Refer to Appendix R]
Our group agreed on working with the answers at the end of the MCQ, having the tutorial page being easily accessible in the gameplay, having a game over scene when the player dies or when the timer runs out and having a congratulations scene when the player successfully saves all the casualties within the given time frame. For where the player is in the map, we put that as our least priority as by then it was reaching the end of our Major Project Journey so we gave more attention as to how to make the current game a much better experience for the player.

We first worked on having an answer key at the end of the MCQs then we worked on adding more variations of damage zones other than fire such as exposed wires on wet grounds, we also added collectible items(bandages) for the players to recover some of their health points upon collecting and lastly we implement a suitable font for the texts in all the canvas to give the player an easier time reading when playing the game.

4. CONCLUSION

We were able to get everything fully functioning and even have small details to make the game experience

enjoyable for anyone who plays it. Creating a game from scratch and having it actually put to use and help students just like us is definitely a wonderful experience and hopefully will be improved better in the future so that future medical students will have an even better learning experience playing the game.

Acknowledgments

The author wishes to thank SGH and DUKE-NUS for working with us. We would also like to thank our supervisor, Mr Tommy Tai, for guiding and helping us throughout the whole of our Major Project Journey.

References

[1] Ruby Tutorial - https://learn.unity.com/project/ruby-s-2d-rpg
[2] Modern City Tile Map - https://shatteredreality.itch.io/modern-city

Other Sources

Appendices

Appendix A:

AnswerScript.cs

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

public class AnswerScript : MonoBehaviour
{
    public bool isCorrect = false;
    public QuizManager quizManager;

   public void Answer()
    {
        if(isCorrect)
        {
            Debug.Log("Correct Answer");
            quizManager.correct();
        }
        else
        {
            Debug.Log("Wrong Answer");
            quizManager.wrong();
```

```
        }

    }
}
```

Appendix B:

AvatarController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;

public class AvatarController : MonoBehaviour
{
    public float speed = 3.0f;

    public int maxHealth = 5;
    public float timeInvincible = 2.0f;

    public int health { get { return currentHealth; } }
    int currentHealth;
    public Slider healthSlider;
    public TextMeshProUGUI healthText;

    bool isInvincible;
    float invincibleTimer;

    Rigidbody2D rigidbody2d;

    public float horizontal;
    public float vertical;

    public int Avatar;
    Animator animator;
```

```
    Vector2 lookDirection = new Vector2(1, 0);

    // Start is called before the first frame update
    void Start()
    {
        rigidbody2d = GetComponent<Rigidbody2D>();
        currentHealth = maxHealth;
        animator = GetComponent<Animator>();
        healthSlider.maxValue = maxHealth;
        healthSlider.value = currentHealth;

    }

    // Update is called once per frame
    void Update()
    {
        //horizontal = Input.GetAxis("Horizontal");
        //vertical = Input.GetAxis("Vertical");
        Vector2 move = new Vector2(horizontal, vertical);

                if  (!Mathf.Approximately(move.x,  0.0f)  ||
!Mathf.Approximately(move.y, 0.0f))
        {
            lookDirection.Set(move.x, move.y);
            lookDirection.Normalize();
        }

        animator.SetFloat("Look X", lookDirection.x);
        animator.SetFloat("Look Y", lookDirection.y);
        animator.SetFloat("Speed", move.magnitude);

        if (isInvincible)
        {
            invincibleTimer -= Time.deltaTime;
            if (invincibleTimer < 0)
                isInvincible = false;
```

```
    }
     /*if (Input.GetButtonDown("Help")) //When user presses h on keyboard, will interact with NPC
    {
                                RaycastHit2D hit = Physics2D.Raycast(rigidbody2d.position + Vector2.up * 0.2f, lookDirection, 1.5f, LayerMask.GetMask("NPC"));
        if (hit.collider != null)
        {
                            NonPlayerCharacter character = hit.collider.GetComponent<NonPlayerCharacter>();
            if (character != null)
            {
                character.DisplayDialog();
            }
        }
    }*/
  }

  public void help()
  {
                                RaycastHit2D hit = Physics2D.Raycast(rigidbody2d.position + Vector2.up * 0.2f, lookDirection, 1.5f, LayerMask.GetMask("NPC"));
    if (hit.collider != null)
    {
                            NonPlayerCharacter character = hit.collider.GetComponent<NonPlayerCharacter>();
        if (character != null)
        {
            character.DisplayDialog();
        }
    }
  }
```

```
    void FixedUpdate()
    {
        Vector2 position = rigidbody2d.position;
            position.x = position.x + 3.0f * horizontal *
Time.deltaTime;
             position.y = position.y + 3.0f * vertical *
Time.deltaTime;
        rigidbody2d.MovePosition(position);
    }

    public void ChangeHealth(int amount)
    {
        if (amount < 0)
        {
            if (isInvincible)
                return;

            isInvincible = true;
            invincibleTimer = timeInvincible;
        }

            currentHealth = Mathf.Clamp(currentHealth +
amount, 0, maxHealth);
        healthSlider.value = currentHealth;
          healthText.text = currentHealth.ToString("F0") +
"/" + maxHealth.ToString("F0");
        Debug.Log(currentHealth + "/" + maxHealth);
        if (currentHealth <= 0)
        {
            SceneManager.LoadScene("Harbour");
        }
    }
}
```

Appendix C:

BackgroundMusic.cs

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

public class BackgroundMusic : MonoBehaviour
{
   private static BackgroundMusic backgroundMusic;

   void Awake()
   {
      if(backgroundMusic == null)
      {
         backgroundMusic = this;
         DontDestroyOnLoad(backgroundMusic);
      }

      else
      {
         Destroy(gameObject);
      }

   }

}
```

Appendix D:

DamageZone.cs

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

public class DamageZone : MonoBehaviour
{
```

```
    void OnTriggerStay2D(Collider2D other)
    {
                            AvatarController    controller    =
other.GetComponent<AvatarController>();

        if (controller != null)
        {

          controller.ChangeHealth(-1);

        }
    }

}
```

Appendix E:

GameTimer.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class GameTimer : MonoBehaviour
{
    public float timeRemaining = 360;
    public bool timerIsRunning = false;
    public Text timeText;
    private void Start()
    {
        // Starts the timer automatically
        timerIsRunning = true;
    }
    void Update()
    {
        if (timerIsRunning)
```

```
        {
            if (timeRemaining > 0)
            {
                timeRemaining -= Time.deltaTime;
                DisplayTime(timeRemaining);
            }
            else
            {
                Debug.Log("Time has run out!");
                timeRemaining = 0;
                timerIsRunning = false;
                SceneManager.LoadScene("Menu");
            }
        }
    }
    void DisplayTime(float timeToDisplay)
    {
        timeToDisplay += 1;
         float minutes = Mathf.FloorToInt(timeToDisplay /
60);
          float seconds = Mathf.FloorToInt(timeToDisplay
% 60);
            timeText.text = string.Format("{0:00}:{1:00}",
minutes, seconds);
    }


}
```

Appendix F:

Help.cs

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

public class Help : MonoBehaviour
```

```
{
    AvatarController ac;

    public void helpButton()
    {
        ac = FindObjectOfType<AvatarController>();
        ac.help();
    }
}
```

Appendix G:

MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene("Harbour");
    }

    public void QuitGame()
    {
        Debug.Log("Quit successful");
        //SceneManager.LoadScene("Quit");
    }

    public void BackToMain()
    {
        SceneManager.LoadScene("Menu");
    }

    public void Tutorial()
```

```
    {
        SceneManager.LoadScene("Tutorial");
    }
}
```

Appendix H:

Navigation

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

public class Navigation : MonoBehaviour
{
    GameObject go;

    public void moveLeft()
    {

gameObject.GetComponent<AvatarController>().horiz
ontal = -1;
    }

    public void stopMove()
    {

gameObject.GetComponent<AvatarController>().horiz
ontal = 0;
    }

    public void moveRight()
    {

gameObject.GetComponent<AvatarController>().horiz
ontal = 1;
    }
```

```
    public void moveUp()
    {

gameObject.GetComponent<AvatarController>().vertic
al = 1;
    }

    public void moveDown()
    {

gameObject.GetComponent<AvatarController>().vertic
al = -1;
    }

    public void stopMoveV()
    {

gameObject.GetComponent<AvatarController>().vertic
al = 0;
    }
}
```

Appendix I :

NonPlayerController.cs

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

public class NonPlayerCharacter : MonoBehaviour
{

    public GameObject dialogBox;
    public float displayTime = 4.0f;
    float timerDisplay;

    // Start is called before the first frame update
```

```
    void Start()
    {
        dialogBox.SetActive(false);
        timerDisplay = -1.0f;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Y))
        {
            dialogBox.SetActive(false);
        }
    }
    public void DisplayDialog()
    {
        dialogBox.SetActive(true);
    }
}
```

Appendix J:

QuestionAndAnswers.cs

```
[System.Serializable]

public class QuestionAndAnswers
{
    public string Question;
    public string[] Answers;
    public int CorrectAnswer;
}
```

Appendix K:

QuizManager.cs

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

```
using UnityEngine.UI;

public class QuizManager : MonoBehaviour
{
    public List<QuestionAndAnswers> QnA;
    public GameObject[] options;
    public int currentQuestion;
    public GameObject Quizpanel;
    public GameObject GoPanel;

    public Text QuestionTxt;
    public Text ScoreTxt;

    int totalQuestions = 0;
    public int score;

    private void Start()
    {
        totalQuestions = QnA.Count;
        GoPanel.SetActive(false);
        generateQuestion();
    }
    public void GameOver()
    {
        Quizpanel.SetActive(false);
        GoPanel.SetActive(true);
        ScoreTxt.text = score + "/" + totalQuestions;

        if (score == totalQuestions)
        {

GameObject.Find("RescueBoard").GetComponent<Res
cueBoardManager>().rescue += 1;
        }
    }
    public void correct()
```

```
    {
        score += 1;
        QnA.RemoveAt(currentQuestion);
        generateQuestion();
    }
    public void wrong ()
    {
        QnA.RemoveAt(currentQuestion);
        generateQuestion();
    }
    void SetAnswers()
    {
        for(int i = 0; i < options.Length; i++)
        {

options[i].GetComponent<AnswerScript>().isCorrect  =
false;

options[i].transform.GetChild(0).GetComponent<Text>
().text = QnA[currentQuestion].Answers[i];

                if(QnA[currentQuestion].CorrectAnswer ==
i+1)
            {

options[i].GetComponent<AnswerScript>().isCorrect  =
true;
            }
        }
    }

    void generateQuestion()
    {
        if (QnA.Count > 0)
        {
                    currentQuestion  =  Random.Range(0,
```

```
QnA.Count);

                                        QuestionTxt.text    =
QnA[currentQuestion].Question;
            SetAnswers();
        }
        else
        {
            Debug.Log("End of quiz");
            GameOver();
        }




    }
}
```

Appendix L:

RescueBoardManager.cs

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

public class RescueBoardManager : MonoBehaviour
{
    public int rescue;
    public Text RescueTxt;
    public GameObject rescueBoard;
    void Start()
    {
        rescueBoard.SetActive(false);
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.S))
```

```
        {
            rescueBoard.SetActive(true);
            DisplayScore();
        }
        if (Input.GetKeyDown(KeyCode.R))
        {
            rescueBoard.SetActive(false);
        }
    }
    public void DisplayScore()
    {
        RescueTxt.text = rescue + "/" + "12";
    }


}
```

Appendix M:

SoundManager.cs

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

public class SoundManager : MonoBehaviour
{
    [SerializeField] Slider volumeSlider;
    // Start is called before the first frame update
    void Start()
    {
        if(!PlayerPrefs.HasKey("musicVolume"))
        {
            PlayerPrefs.SetFloat("musicVolume", 1);
            Load();
        }
        else
```

```
        {
            Load();
        }

    }
    public void ChangeVolume()
    {
        AudioListener.volume = volumeSlider.value;
        Save();
    }
    private void Load()
    {
                                        volumeSlider.value      =
PlayerPrefs.GetFloat("musicVolume");
    }
    private void Save()
    {
                        PlayerPrefs.SetFloat("musicVolume",
volumeSlider.value);
    }

}
```

Appendix N:

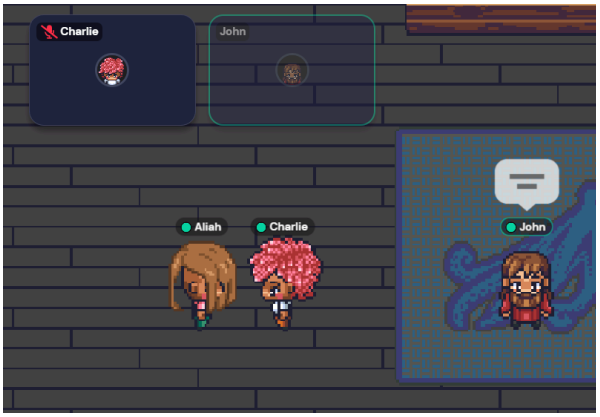
Switch.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class Switch : MonoBehaviour
{

    public GameObject[] background;
    int index;
```

```
void Start()
{
    index = 0;
}

void Update()
{
    if (index >= 4)
        index = 4;

    if (index < 0)
        index = 0;

    if (index == 0)
    {
        background[0].gameObject.SetActive(true);
    }

    if (Input.GetKeyDown(KeyCode.T))
    {
        background[index].SetActive(false);
    }
}

public void Next()
{
    index += 1;

    for (int i = 0; i < background.Length; i++)
    {
        background[i].gameObject.SetActive(false);
        background[index].gameObject.SetActive(true);
    }
    Debug.Log(index);
}
```

```
    public void Previous()
    {
        index -= 1;

        for (int i = 0; i < background.Length; i++)
        {
            background[i].gameObject.SetActive(false);
            background[index].gameObject.SetActive(true);
        }
        Debug.Log(index);
    }

    public void Back()
    {
        SceneManager.LoadScene("Menu");
    }

    public void BackToGame()
    {
        SceneManager.LoadScene("Harbour");
    }

    public void close()
    {
            background[index].SetActive(false);

    }
}
```

Appendix O:



Interacting with other players

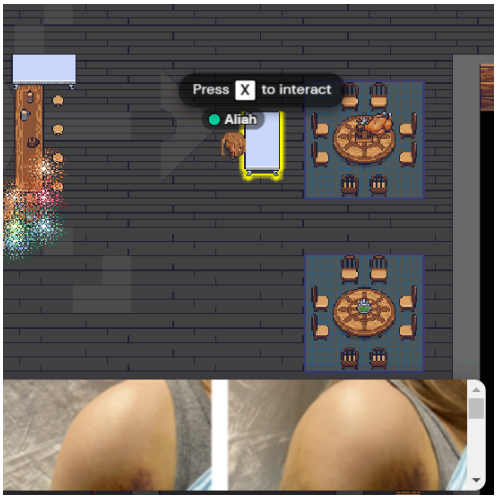
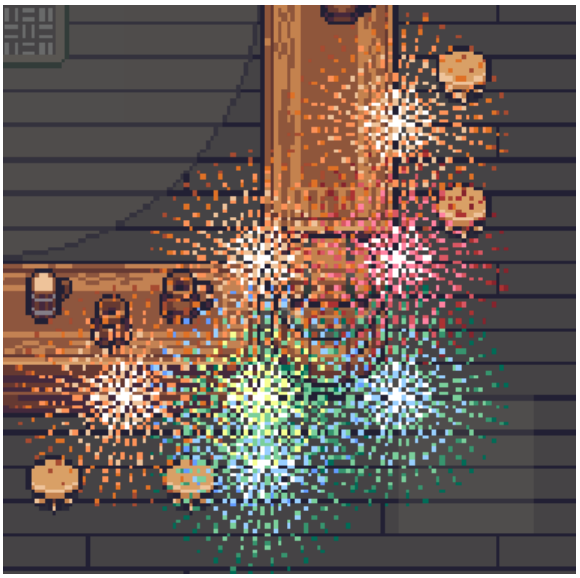
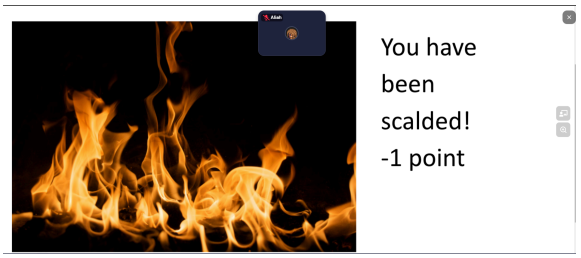


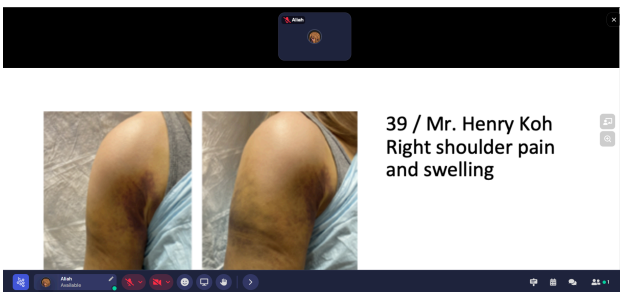
Table represents as casualty



Fireworks to represent fire



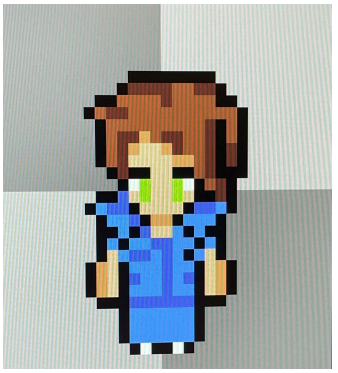
Fire Damage



Pop-up when interacted with casualty

Appendix P:

Final Design

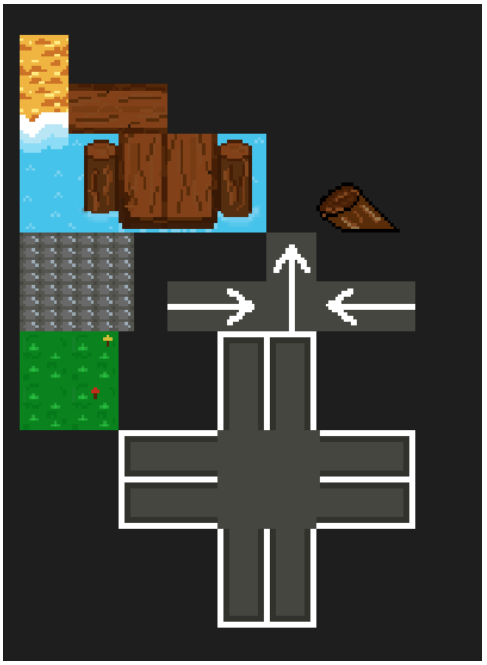


Initial Design

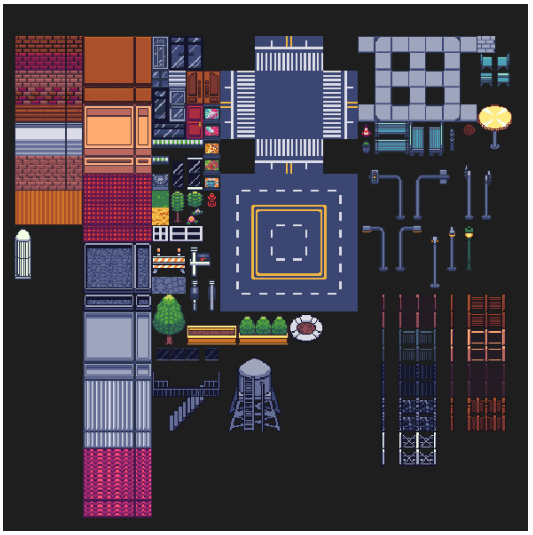
Crying Girl - Casualty

Casualty

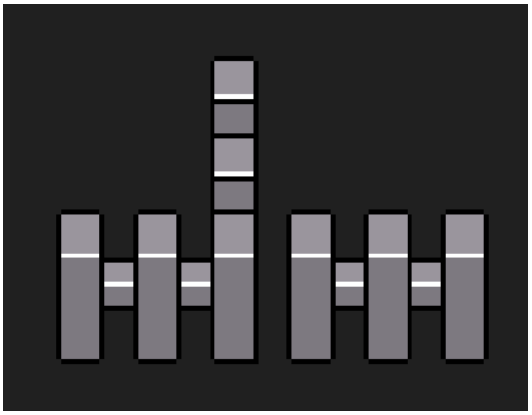
Appendix Q:



Figure



Figure

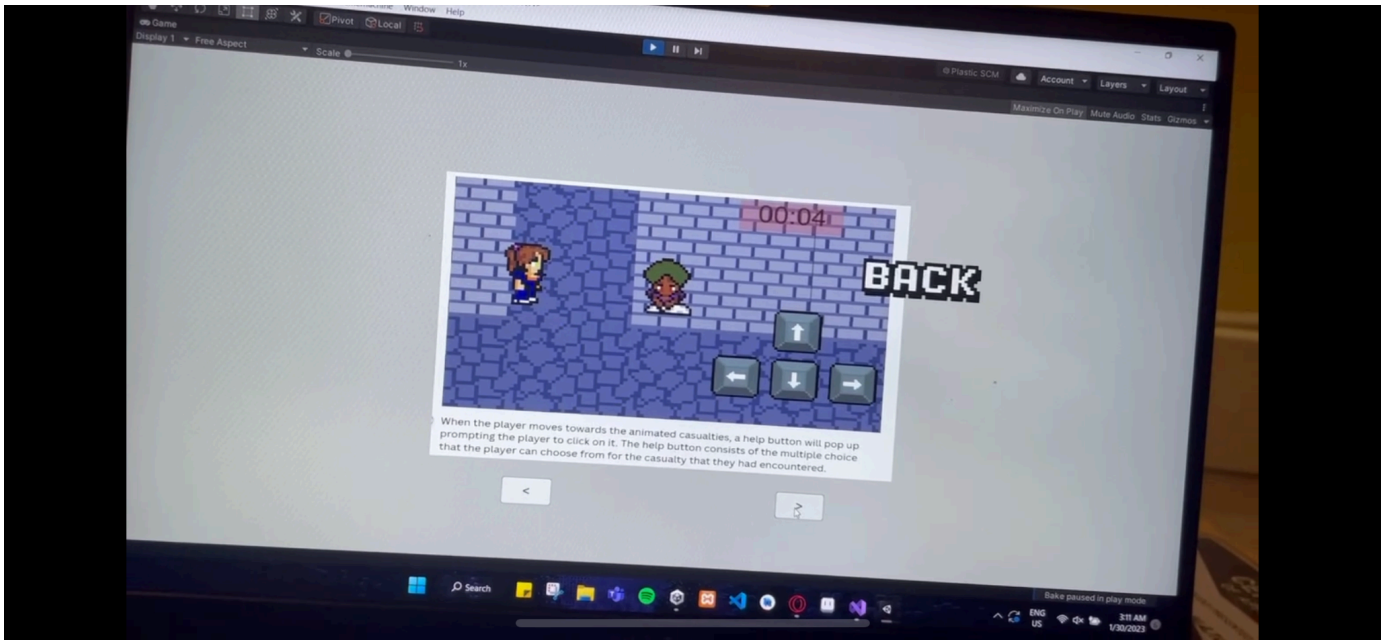


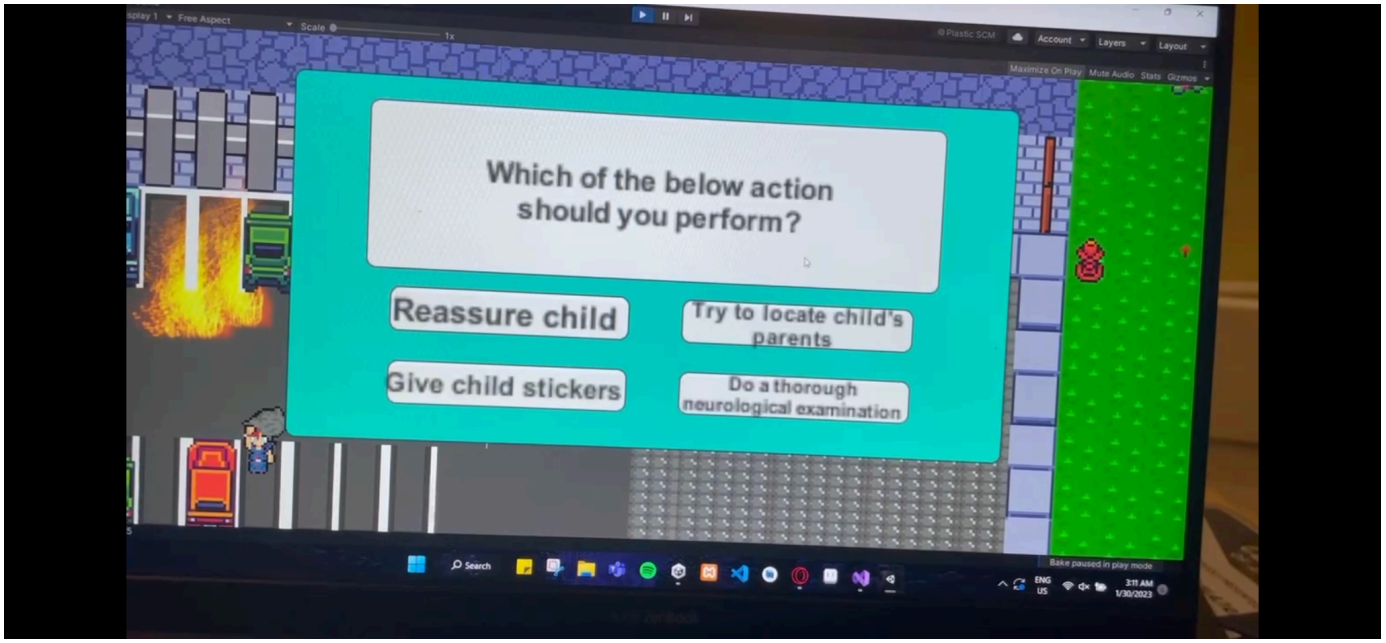
Figure

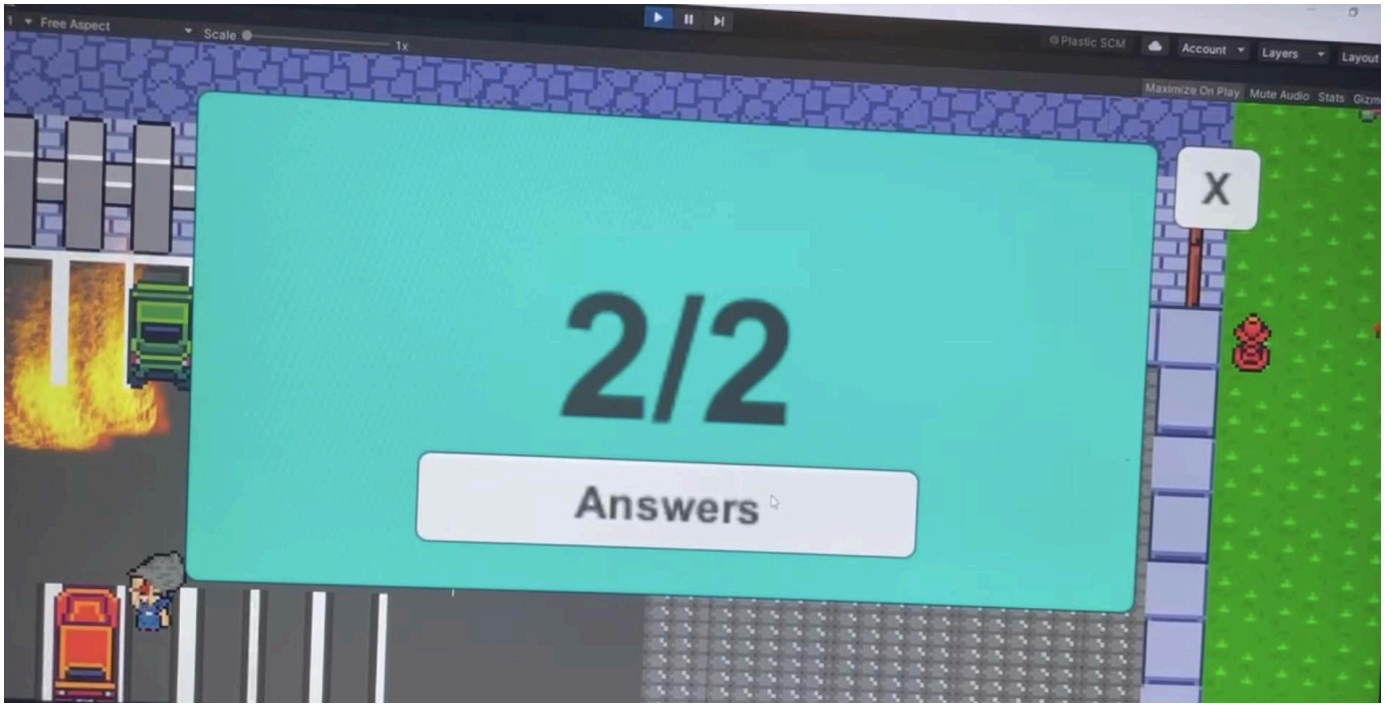
Figure

Figure

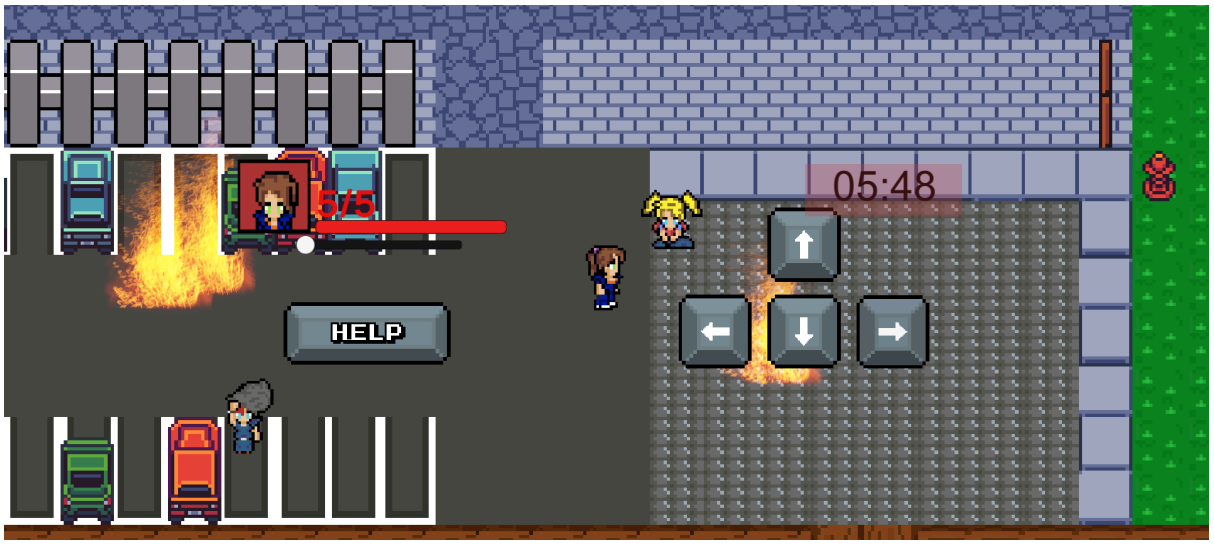
Appendix R:

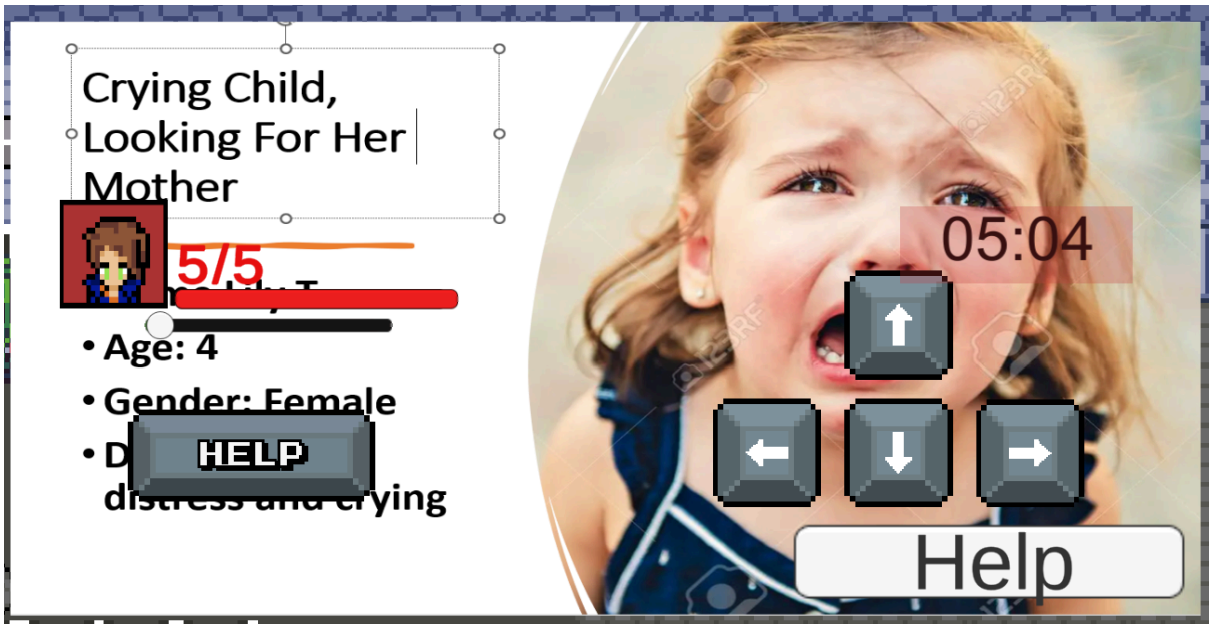

00:04
BACK
When the player moves towards the animated casualties, a help button will pop up prompting the player to click on it. The help button consists of the multiple choice that the player can choose from for the casualty that they had encountered.

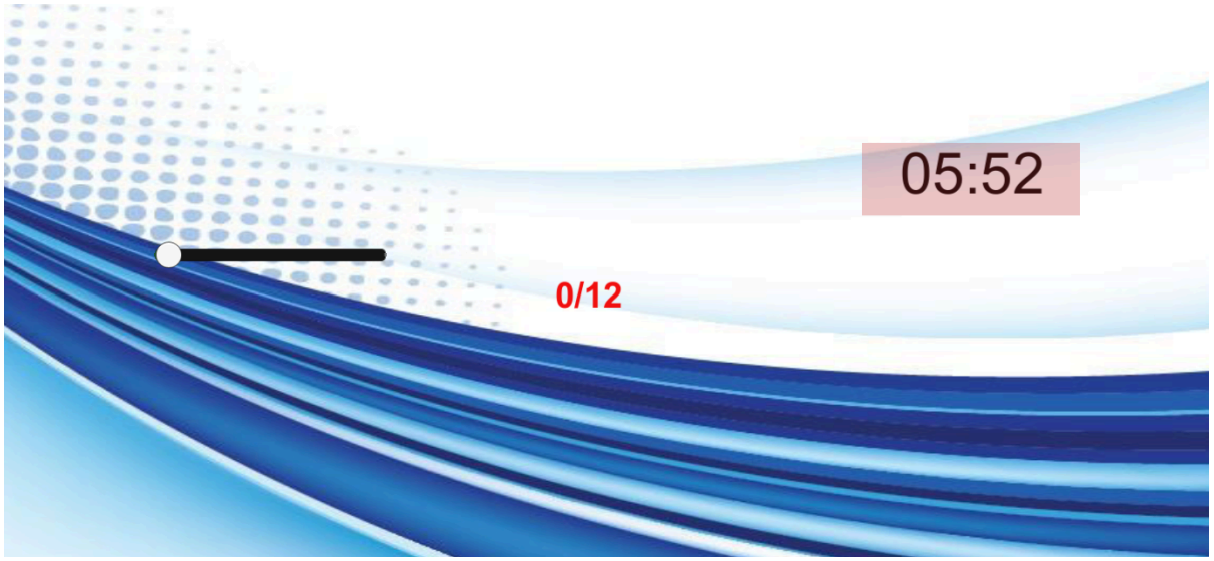

Which of the below action should you perform?
Reassure child
Try to locate child's parents
Give child stickers
Do a thorough neurological examination


2/2
Answers
X

Appendix S:


05:48
5/5
HELP


Crying Child,
Looking For Her
Mother
5/5
• Age: 4
• Gender: Female
HELP
05:04
Help


05:52
0/12


Inventory
Bokoblin Fang
12:55