The Hong Kong Polytechnic University

Electric and Information Engineering Department

EIE3360 Integrated Project

Project Final Report

Game Title: Java Mart

Target Course: EIE3320 JAVA Programming

Submission Date: 2023/4/12

Group 2

DING Jiaqi	20075618D
HAO Jiadong	20084595D
ZHANG Chengcheng	20076355D
FAN Zhendi	20101448D

Page list

1. Abstract 3

2. Objective 3

3. Background study 3

4. Introduction 4

UI design and Functions 4

Scene Interaction Animations 5

Level content introduction 6

5.Methodology 8

Development hardware and software architecture: 8

Technical implementation 9

6. Project impact on Internet and Multimedia Technologies 16

7. Resource and teamwork management 16

8. Difficulties encountered 16

9. Conclusion 18

10. Distribution Table 18

11. Project schedule 20

12. Reference 20

1. Abstract

This report introduces a 3D mobile application that designed to assist in teaching EIE3320, Java programming, particularly data structures (list and stack), sorting algorithms (bubble sort and insertion sort) and the java GUI and Graphics. The application simulates a shopping mall environment where players complete tasks as part of a roleplay-game like teaching process. These tasks include sorting shopping baskets, arranging delivery orders, helping the cashier sort coupons and painting on the advertisement board. The game provides an interactive and engaging way to learn Java concepts, with a final quiz to test the player's understanding on topics delivered.

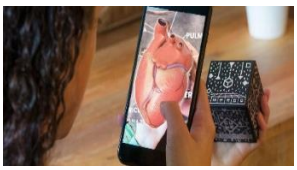
2. Objective

The objective of this project is to create a 3D mobile application that provides an engaging and interactive learning experience for JAVA programming students. By simulating real-world scenarios and providing gamified missions, the application aims to increase student engagement and comprehension of difficult concepts. Additionally, the application aims to provide immediate feedback and assessment through the games in each level and the final quiz, allowing students to identify areas for improvement and reinforce their learning.

3. Background study

In recent years, the mobile app market has experienced a significant rise in the use of 3D applications for interactive education, making learning more engaging and effective. Among these popular applications are Merge Cube, Anatomy 4D, and Metaverse, each offering unique learning experiences.

Anatomy 4D



Merge Cube

Metaverse

- Merge Cube is a groundbreaking augmented reality tool that allows users to interact with virtual 3D objects through their smartphones or tablets. This innovative app transforms abstract concepts into tangible, interactive experiences, enhancing students' understanding of various subjects. *[1]*
- Anatomy 4D is an immersive 3D application that brings human anatomy to life. By visualizing and interacting with the human body's various systems, students can develop a deeper understanding of complex anatomical structures and their functions. *[2]*
- Metaverse, on the other hand, enables users to create, explore, and interact with virtual worlds. This versatile platform has applications in education, storytelling, and entertainment, offering students an engaging and interactive way to learn and collaborate. *[3]*

Despite the success of these applications, there is still room for improvement and expansion, particularly in the field of Java programming. Java is a widely used programming language with applications in various domains, such as web development, mobile applications, and data analytics. To address the need for effective Java programming education tools, our project aims to create a 3D interactive application focused on topics like data

structures, sorting algorithms, and Java GUI and Graphics.

By incorporating innovative techniques such as gamification and real-time assessment, our 3D application seeks to enhance students' learning experiences, making Java programming concepts more accessible and engaging. In conclusion, the development of this 3D mobile application aims to provide an interactive and engaging way for students to learn complex Java programming topics, building on the success of existing 3D educational applications.

In conclusion, the development of a 3D mobile application for teaching Java programming will offer an interactive and engaging way for students to learn complex topics such as data structures, sorting algorithms, and Java GUI and Graphics. By combining gamification and real-time assessment, our application aims to enhance the learning experience for Java programming students significantly.

4. Introduction

Learning JAVA programming can be challenging, especially when it comes to understanding complex topics such as data structures and sorting algorithms. To help students better comprehend these concepts, we have developed a 3D mobile application that simulates a shopping mall environment. By presenting programming concepts in a gamified setting, students can immerse themselves in real-world scenarios and practice critical thinking skills.

The application provides various missions that require players to organize shopping baskets, sort delivery orders, assist store clerks with coupon sorting and design a shop poster on the advertisement board. By completing these tasks, students can embody and understand the knowledge they have learned. Additionally, a final quiz is set to provide an assessment of the player's understanding and allows for immediate feedback, helping students identify areas for improvement and reinforcing their learning.

UI design and Functions

The UI adopts an uncomplicated design and uses display questions and button options as user interaction methods. This is an easy-to-operate UI that takes into account the limitations of the mobile phone screen.

The task list allow player to see the total tasks of this application. In each game level, the student will get hints on solving the question by clicking the hint button (question mark icon) at the top right corner. After completing each level in scene, below three indicating signals will emerge. The detailed animation is demonstrated in the demo video we submitted. Including prompting of you-win pop-up and updating of the task list as shown in the next part, and the question mark converts to check mark.

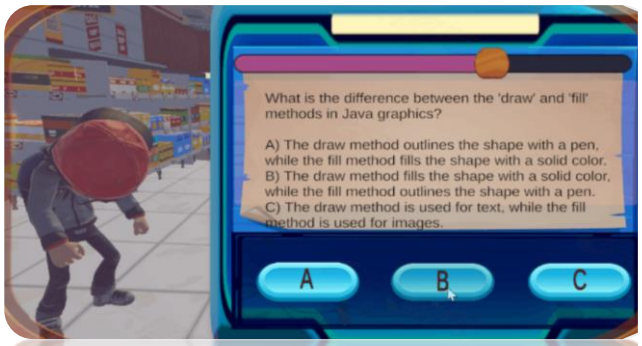
Scene Interaction Animations

Below is an overview of our application's character and environment interaction designs. These designs are crafted by us to enhance the user experience and provide an immersive learning environment.

Character animations design

Character interaction design includes a variety of animations and features that bring our characters to life.

We have already implemented walking animation for our character, and we have now added a celebratory jumping animation for when the user completes a level, as well as a frustrated gesture when the player provides a wrong answer in the final quiz.



Frustrating

Cheering

Environment interactions design

Environment interaction design is also an essential part of the user experience. We have implemented specific animations for entering various levels, including the Basket Level, Coupon Level, Advertisement Board Level, and List Level.

- For the Basket Level, we have added a carrying basket animation to indicate when the user enters the shopping cart level.
- For the Coupon Level, a greeting animation with the cashier has been implemented to indicate when they enter the coupon sorting level.
- For the List Level, the character performs a grabbing box animation to showcase the entry into the level.

Basket Level

Coupon Level

List Level

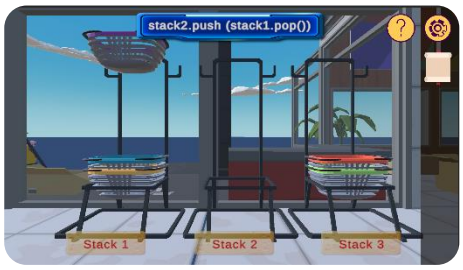
Advertisement Board Level

- For the Advertisement Board Level, the character takes out a pen and walks towards the billboard, signifying the beginning of the level.

In the game, players need to tap on the question mark floating in the scene to play the animation and enter the teaching interface. These animations add a layer of fun and engagement to the learning process, creating a more immersive experience for the user.

Level content introduction

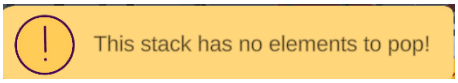
Basket Level



In the basket level, we're trying to use the placement of the baskets to illustrate the concept of stack in java because they are in the same FILO (First In Last Out) manner. The whole level is divided into two parts: teaching part and game part.

During the teaching part, the subtitles will pop up from the bottom to explain the connections between the baskets and the stack concept and what the functionalities of the ".pop()" and ".push()" in java.

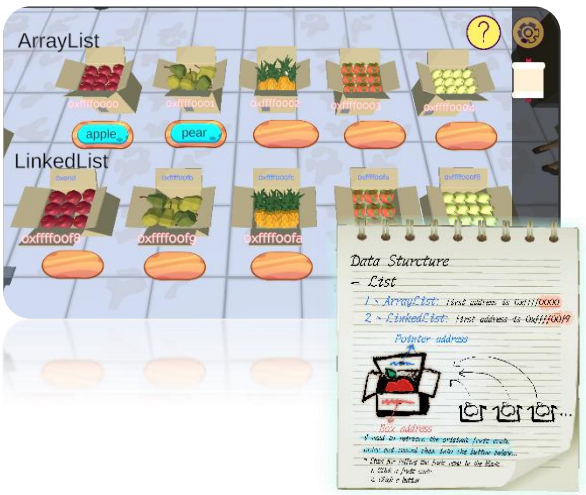
During the game, the player is required to use ".push()" and ".pop()" to place the baskets in a sequence in the middle stand. In each round, the firstly clicked stack name will be filled before ".push()" and the secondly-clicked stack name will be filled before ".pop()". After two blanks are filled, an animation of basket will be displayed to show the effect. If the player tries to pop a basket from an empty stand, an error message will be displayed.



List level

In the list level, our aim is to deliver the basic concept of the ArrayList and the Linkedlist.

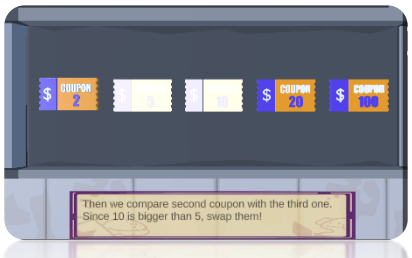
This game has two parts. The first part provides a find order game that expect learner to get familiar with the storage rules of these two kinds of list. Users can click the box of fruit first and the click the blank to fill in the fruit name into the blanks. The second half is to teach how to insert an element into the list with a specific index. After following the animation step by step, there is a checkpoint to make sure the learner fully understands the operation process and perceives the pros and cons between these two kinds of data structures.



Coupon Level

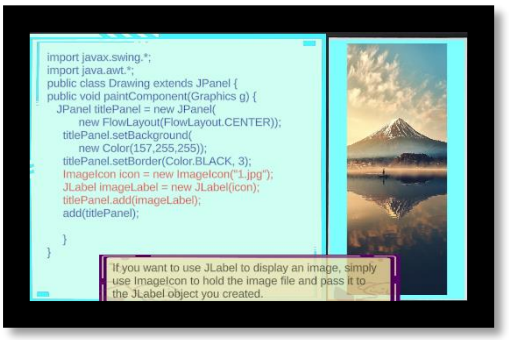
In the coupon level, we're trying to deliver the knowledge of bubble sort and insertion sort. The whole level is divided into two parts: teaching part and game part.

In the teaching part, we will sort the 5 coupons step by step using the bubble and insertion sort with subtitle explanations. There will be a flashing effect added to the two coupons under comparison to draw the player's attention. In the game phase, the aim is to answer a question regarding to the working principal of the two sorting algorithms. If the player finds it's a bit hard to answer, he can press the hint button on the top right corner to ask for hints.



Advertisement board Level

In the advertisement level, we're trying to deliver the basic java GUI and Graphics knowledge. The whole level is also divided into the teaching part and the game part.



During the teaching part, the subtitles will pop up and the corresponding codes will be highlighted in red. The effect of the codes will be displayed on the right panel. For example, the picture shows we're trying to illustrate how to display an image using "ImageIcon" and "JLabel".

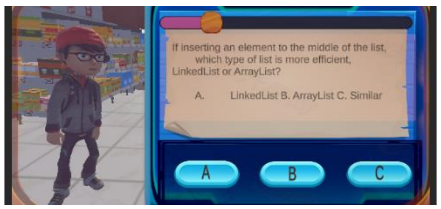
For the game part, the aim is to create a poster for the store. The player should choose the correct option from the drop-down list to create a title bar, a smiley face and a picture onto the poster. After completing one part, the player can check whether all the answers for that part are correct by clicking the corresponding button on the right panel. If correct, the corresponding effect will be displayed accordingly.

Final Quizzes level

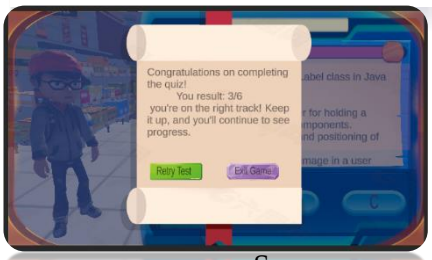
This is a test designed to test a player's understanding on the knowledge delivered in the game and EIE3320.

After completing 4 tasks in scene, a vibrating cue signal starts to play on the task sheet, at which point the final test session can be accessed via the button in the bottom right corner of the task sheet. There are six questions in total, each with an ABC option. A wrong choice is indicated by a beep and the character on the left plays a remorseful animation, while the animation of a correct choice is the celebration animation. There is a progress bar at the top of the screen. After completing the 6 questions, the result will be displayed in the summary page. At the end of a quiz, player can choose to restart the test or return to the main game scene.

Before→ Later



Question screen



Summary page

5.Methodology

Development hardware and software architecture:

Hardware	PC (develop) and Android phone (test)
Build Models and Animations	3DS Max, Blender
Image Design	Procreate, Photoshop
Video Edit	Final cut pro, Adobe Premiere Pro
Game Engine	Unity
Code Editor	Visual Studio Community 2017
Work Collaboration	Plastic SCM

Technical implementation

1. Character moving and rotating (*PlayerController.cs*):

Firstly, we start by getting the input values from a joystick that provides the horizontal and vertical values of the input. Then we get the movement direction of the character based on the camera angle and rotate the character to face the direction of movement.

Next, we calculate the current speed of the character and update its position using the Lerp method to smoothly interpolate between its current position and the target position.

Finally, we set two animator parameters based on the magnitude of the vector "*moveDirection*" which allows for the animation of the character's movement.

```
void FixedUpdate()
{
    // Get the input from the joystick
    float horizontal = joystick.Horizontal;
    float vertical = joystick.Vertical;

    // Calculate the move direction based on the camera angle
    moveDirection = mainCamera.transform.TransformDirection(new Vector3(horizontal, 0, vertical));
    moveDirection.y = 0;

    // Rotate the character to face the direction of movement
    if (moveDirection != Vector3.zero)
    {
        targetRotation = Quaternion.LookRotation(moveDirection);
    }

    // Smoothly rotate the character towards the target rotation
    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime * 5f);

    // Move the character in the direction of the joystick input
    float currentSpeed = Mathf.Clamp(moveDirection.magnitude, 0, 1) * speed;
    Vector3 targetPosition = transform.position + moveDirection * currentSpeed * Time.deltaTime;
    transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime);

    animator.SetBool("isWalking", moveDirection != Vector3.zero);
    animator.SetFloat("Speed", moveDirection.magnitude);
}
```

Character movement and rotation reference: *Rotating a Character in the Direction of Movement (Unity Tutorial) - YouTube [4]*

2. Main camera following the character (CameraController.cs):

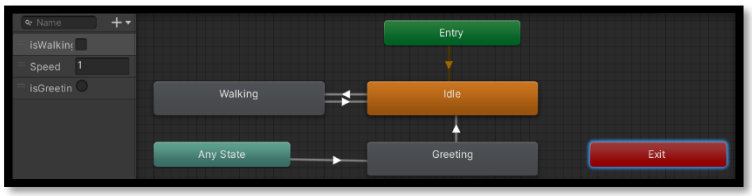
```
void FixedUpdate()
{
    float wantedRotationAngle = target.eulerAngles.y + yaw;
    float wantedHeight = target.position.y + height;

    float currentRotationAngle = transform.eulerAngles.y;
    float currentHeight = transform.position.y;

    currentRotationAngle = Mathf.LerpAngle(currentRotationAngle, wantedRotationAngle, rotationDamping * Time.deltaTime);
    currentHeight = Mathf.Lerp(currentHeight, wantedHeight, heightDamping * Time.deltaTime);

    Quaternion currentRotation = Quaternion.Euler(pitch, currentRotationAngle, 0);

    Vector3 newPosition = target.position - currentRotation * Vector3.forward * distance;
    newPosition.y = currentHeight;

    if (Physics.Raycast(target.position, newPosition - target.position, out RaycastHit hit, distance, obstacleLayer))
    {
        newPosition = target.position - currentRotation * Vector3.forward * hit.distance;
        newPosition.y = currentHeight;
    }

    transform.position = newPosition;
    Vector3 lookAtPosition = target.position + Vector3.up * targetHeightOffset;
    transform.LookAt(lookAtPosition);
}
```

The CameraController script allows the main camera to smoothly follow the character while maintaining a specified distance and height. In this script, we first set up the public variables for target, distance, height, rotationDamping, heightDamping, pitch, yaw, and targetHeightOffset.

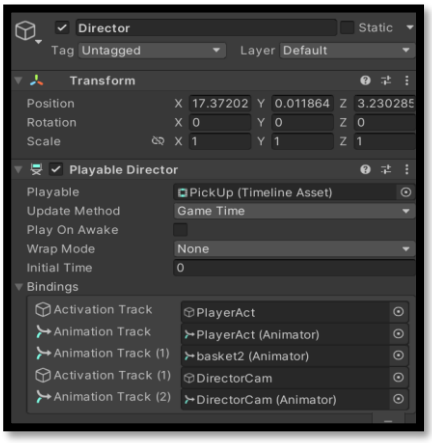
Inside the FixedUpdate() method, we calculate the desired rotation angle and height based on the target's position and the yaw and height values. We then use Mathf.LerpAngle and Mathf.Lerp to interpolate between the current rotation angle and height and the desired values, resulting in smooth camera movement.

A Quaternion is created to store the current rotation, and a new position for the camera is calculated based on this rotation. The camera is positioned at the calculated newPosition and is set to look at a point offset from the target's position by the targetHeightOffset value. This ensures that the camera follows the character and smoothly adjusts its position and rotation in the game environment.

3. Character animations



The character animations are controlled using three parameters in the Animator: *isWalking, Speed,* and *isGreeting*. The *isWalking* boolean triggers the transition between Idle and Walking animations. *Speed*, a multiplier, adjusts the walking animation's pace, and *isGreeting*, a trigger, initiates the Greeting animation regardless of the current animation.



For the PickUp basket animation, a GameObject called *Director* is created, containing a *PlayableDirector* component linked to a 'pickup' Timeline. This Timeline connects to a camera, basket, and player duplicate. When activated, the player duplicate becomes visible and picks up the basket, while the camera switches to *DirectorCam*, capturing the animated sequence.

Timeline Reference: *Intro to Unity Timeline- YouTube [5]*

4. Transition effect

To enhance the user's game experience, we add many transition effects to our game to make the transition more smoothly, for example, the fading-in and fading-out effects when the player enters a level and gets back to the main scene, and also when a text or image appears on the screen. To make our code more efficient, we apply overloading mechanism.

Below shows the logic of a coroutine called "GradualAppearEffect", which is used to create the fading-in effect of a TextMesh Pro game object. We firstly set the initial color of the text object to have an alpha value of 0. Then we use a while loop to gradually change the alpha value from 0 to 1 over the specified duration using "Mathf.Lerp".

```
IEnumerator GradualAppearEffect(TextMeshProUGUI textMeshPro, float duration)
{

    Color color = textMeshPro.color;
    color.a = 0;
    textMeshPro.color = color;

    // Gradually change the color to be fully opaque
    float startTime = Time.time;
    while (Time.time - startTime < duration)
    {
        float alpha = Mathf.Lerp(0, 1, (Time.time - startTime) / duration);
        color.a = alpha;
        textMeshPro.color = color;
        yield return null;
    }

    // Set the final color to be fully opaque
    color.a = 1;
    textMeshPro.color = color;
}
```

We also have some overloading coroutines to create similar fading-in effects for images and canvas as the following figures show. This kind of overloading technics make our code more readable and easier for debugging.

```
IEnumerator GradualAppearEffect(Image image, float duration)
```

```
IEnumerator GradualAppearEffect(Canvas canvas, float duration)
```

Fade-in effect reference: *c# - Using a Coroutine in Unity3D to fade a game object out, and back in. Looping infinitely - Stack Overflow [6]*

5. Subtitle display (Subtitle_advertisement.cs, Subtitle_coupon.cs, Subtitle_basket.cs)

The desired effect of the subtitles is that when there are no words displayed, a left clicking will start the displaying process, which gradually shows the words character by character. If the player wants to see the whole sentence immediately, he can left click during the character-by-character displaying process.

```
void Start()
{
    subtitles = new string[6];
    subtitles[0] = "Look! Shopping baskets are neatly stacked on a stand, waiting for customers to take them.";
    subtitles[1] = "Normally, customers will only take the top basket and put the unused baskets back on top.";
    subtitles[2] = "That perfectly illustrate the concept of stack, " +
        "where elements are always added and removed from the top in a last-in, first-out (LIFO) manner.";
    subtitles[3] = "In Java, the \"stack\" class provides two main methods for adding (.push(item)) and removing (.pop()) elements from the top " +
        "of the stack.";
    subtitles[4] = "Now you may get the basic idea about the stack. Let's play a game to test your understanding!";
    subtitles[5] = " Choose the correct stack number to fill in the blanks of the codes to create a rainbow-colored sequence of shopping baskets! ";
    GetComponent<TextMeshProUGUI>().text = "";
}
```

First, in the Start() function, we initialize an array called subtitles to store the sentences.

Then, in the Update() function, if a left-click is heard, we further check the variable “isDisplaying” to see whether the subtitle is being displayed or not. If (isDisplaying==false), then we start a coroutine “displaySubtitle” to display the subtitles character by character. If the canvas is not blank (isDisplaying == true), a left-click will display all the subtitle at once, hence we stop the “*displayCoroutine*” and directly put the subtitle onto the “*TextMeshPro*” object.

```
void Update()
{
    if (Input.GetMouseButtonDown(0) && !isDisplaying)
    {
        if (currentSubtitleIndex < subtitles.Length)
        {
            isDisplaying = true;
            displayCoroutine = StartCoroutine(DisplaySubtitle(subtitles[currentSubtitleIndex]));
            currentSubtitleIndex++;
        }
        else
        {
            //currentSubtitleIndex = 0;
            isPlaying = true;
            hint_btn.gameObject.SetActive(true);
            buttons.gameObject.SetActive(true);
            subtitle_canvas.gameObject.SetActive(false);// All subtitles have been displayed, do something else here
        }
    }
    else if (Input.GetMouseButtonDown(0) && isDisplaying) // If player clicks while subtitle is displaying
    {
        StopCoroutine(displayCoroutine); // Stop coroutine
        GetComponent<TextMeshProUGUI>().text = subtitles[--currentSubtitleIndex]; // Display full subtitle
        isDisplaying = false; // Set isDisplaying to false to exit coroutine
        currentSubtitleIndex++;
    }
}
```

```
IEnumerator DisplaySubtitle(string subtitle)
{
    currentText = "";
    for (int i = 0; i < subtitle.Length; i++)
    {
        currentText += subtitle[i];
        GetComponent<TextMeshProUGUI>().text = currentText;
        yield return new WaitForSeconds(letterDelay);
    }
    isDisplaying = false;
}
```

The above is just the implementation of the "*DisplaySubtitle*" coroutine to display the sentence character by character, the "yield return" lets the coroutine waits for "*letterDelay*" seconds before displaying the next character.

Subtitle display reference: *Display rich text character by character - Unity Answers [7]*

6. Basket game implementation (*BasketStand.cs*):

In the basket game, we ask the player to fill in two blanks in the code segment to control the movement of baskets and place them in a correct order. The two blanks in the code segment are "___.push(___.pop())". The desired effect of the basket is that the firstly clicked stack name will be filled in the first blank of the code segment (identifying the stack the popped basket should go) and the subsequently-clicked stack name will be filled in the second blank (identifying which stack will pop a basket). After both blanks are filled, show a movement of the basket.

The variable "*push_Stack*" is just a flag to indicate whether this stack name is firstly-clicked or secondly-clicked. If it equals 0, the stack name will be put on the first blank. If it equals 1, we should further check if the stack that is going to pop an element indeed has an element. If it is an empty stack, we should pop up some error messages by "*Instantiate(errorPrompt)*". If it has a basket to pop, we should get the top basket using "*pop_Stack_trans.GetChild(0)*" and start the movement of that basket using the coroutine called "*WorkingSequence*";

```
void ButtonClicked(Button button, int i)
{
    if (push_Stack == 0)
    {
        push_Stack = i + 1;

        textMeshPro.text = "stack" + push_Stack + ".push (___.pop())";
    }
    else
    {
        textMeshPro.text = textMeshPro.text.Replace("___", "stack" + (i + 1));
        Transform pop_Stack_trans = stacks[i].gameObject.transform;
        Transform push_Stack_trans = stacks[push_Stack - 1].gameObject.transform;
        if (pop_Stack_trans.childCount > 0)
        {
            Transform pop_basket = pop_Stack_trans.GetChild(0);

            StartCoroutine(WorkingSequence(pop_basket, push_Stack_trans));
        }
        else
        {
            Debug.Log("stack" + (i + 1) + " has no elements to pop!");
            Instantiate(errorPrompt);
            push_Stack = 0;
            textMeshPro.text = "___.push(___.pop())";
        }
    }
```

```
yield return StartCoroutine(TransitionCoroutine(pop_basket, pop_basket.localPosition, new Vector3(0, 1.7f, 0)));
pop_basket.SetParent(push_Stack_trans);
pop_basket.SetAsFirstSibling();
yield return StartCoroutine(TransitionCoroutine(pop_basket, pop_basket.localPosition, new Vector3(0, 1.7f, 0)));
int num_of_children = push_Stack_trans.childCount;
yield return StartCoroutine(TransitionCoroutine(pop_basket, pop_basket.localPosition, new Vector3(0, 0.4f + 0.1f * (num_of_children - 1), 0)));
```

In "*WorkingSequence*", the key code responsible for the movement of a stack is shown above. The "*TransitionCoroutine*" is another coroutine that is used to move the basket smoothly from one position to another. The first line moves the basket up along the current basket stand, the second and third line is to reset the basket's parent in the hierarchy so that it's convenient to do the following animations. The fourth and sixth line is to move the basket horizontally and downwards respectively.

The "*isWinning*" function is to check whether all the baskets are stacked on the second basket stand in order. The logic is to get the child of the second basket stand one by one to check whether they are in the same sequence of a predefined objects array called "*basket[]*".

```
private bool isWinning()
{
    if (stacks[1].gameObject.transform.childCount == 5)
    {
        bool win = true;
        for (int m = 0; m <= 4; m++)
        {
            if (stacks[1].gameObject.transform.GetChild(m).gameObject != baskets[m])
            {
                win = false;
                break;
            }
        }
        return win;
    }
    return false;
}
```

7. Coupon game implementation(*Subtitle_coupon*):

```
if (ii < couponsArray.Length - 1)
{
    if (jj < couponsArray.Length - ii - 1)
    {
        if (numbers[jj] > numbers[jj + 1])
        {
            script1.Swap(jj, jj + 1);
        }
        prejj = jj;
        jj++;
        if (jj == couponsArray.Length - ii - 1)
        {
            ii++;
            jj = 0;
        }
```

```
if (ii < couponsArray.Length)
{
    if (jj >= 0 && numbers[jj] > numbers[jj + 1])
    {
        script1.Swap(jj, jj + 1);
        jj--; // Move to the previous element
    }
    else
    {
        ii++; // Move to the next element
        jj = ii - 1; // Reset jj to the previous e
    }
}
```

The above two codes show the logic of the bubble sort and insertion sort. We maintain ii and jj to simulate bubble sort and insertion sort, and each time we receive a mouse click, ii and jj will change accordingly. If we detect that the order of the two coupons under comparison needs to be switched, we call "Swap" function to smooth the movement of coupons.

8. Coupon flashing effect (FlashingEffect.cs)

A coroutine called "*FlashCoroutine()*" is used to implement the flashing effect of the coupons. The flashing is implemented by changing the emission color of the material using two while loops (turning brighter and turning darker process). However, simply apply the "FlashCoroutine" to two coupons under comparison can cause problem of flashing out of sync. The reason behind is that we use many times of Time.time to calculate the time elapsed, which is updated every frame and might not be perfectly synchronized between the two coroutines. The solution will be discussed in detail in the "Difficulties encountered" session.

```
private IEnumerator FlashCoroutine()
{
    while (isFlashing)
    {
        float startTime = Time.time;
        while (Time.time < startTime + flashSpeed)
        {
            float t = (Time.time - startTime) / flashSpeed;
            Color targetColor = Color.Lerp(originalColor, flashColor, t);
            mat.SetColor("_EmissionColor", targetColor);
            yield return null;
        }

        startTime = Time.time;
        while (Time.time < startTime + flashSpeed)
        {
            float t = (Time.time - startTime) / flashSpeed;
            Color targetColor = Color.Lerp(flashColor, originalColor, t);
            mat.SetColor("_EmissionColor", targetColor);
            yield return null;
        }
    }
    mat.SetColor("_EmissionColor", originalColor);
}
```

Flashing effect reference: *https://www.youtube.com/watch?v=DNMdu3kylec [8]*

9. Basic operations animations and its reusability (ListGame.cs)

The insert box teaching animation flow is mainly consisting of three different operations (Moving, Scaling and Flashing). Writing these operations in individual methods correspondingly is the best solution to make the entire code as concise as possible. For instance, the Move method, which have five parameters as input. It allows to customize which object is about to move (gameobject), to move in which direction (direction), the moving distance (distance), operation delay after the coroutine is invoked (waitingTime) and if allow subtitle to continue (notMove).

```
private IEnumerator Move(GameObject gameobject, Vector3 direction, float distance, float waitingTime = 0f, bool notMove = true)
```

```
private IEnumerator Scale(GameObject gameObject, float targetScaleFactor, float waitingTime = 0f, bool notMove = true)
```

```
private IEnumerator FlashCoroutine(GameObject gameObject)
```

Therefore, we can call these methods with some parameters input directly to realize a complex animation effect.

10. Important Codes Emphasized in red (CodingSpace.cs)

```
IEnumerator BlinkLine(int[] shouldFlash, float duration, string[] line)
{

    Color originalColor = codeSpace.color;
    Color blinkColor = Color.red;
    string[] displayedLines = (string[])line.Clone();

    // Set the color of the specified lines to red
    foreach (int index in shouldFlash)
    {
        displayedLines[index] = $"<color=#{ColorUtility.ToHtmlStringRGBA(originalColor)}>{line[index]}</color>";
    }

    float elapsed = 0;
    // Gradually change the color of the lines from red to the original color
    while (elapsed < duration)
    {
        elapsed += Time.deltaTime;
        float lerpValue = Mathf.Clamp01(elapsed / duration);
        Color currentColor = Color.Lerp(originalColor, blinkColor, lerpValue);
        for (int i = 0; i < shouldFlash.Length; i++)
        {
            int index = shouldFlash[i];
            string displayedLine = displayedLines[index];
            displayedLine = Regex.Replace(displayedLine, "<color=#([0-9A-Fa-f]{8})>", "");
            displayedLine = Regex.Replace(displayedLine, "</color>", "");
            displayedLine = $"<color=#{ColorUtility.ToHtmlStringRGBA(currentColor)}>{line[index]}</color>";
            displayedLines[index] = displayedLine;
        }
        codeSpace.text = string.Join("", displayedLines);
        yield return null;
    }

    // Set the code space text to the final displayed lines with the blink color
    codeSpace.text = string.Join("", displayedLines);
    blinkLineFinished = true;
}
```

In the advertisement board game, when the subtitles are explaining some lines of codes, the codes will be highlighted in red to draw the player's attention. A coroutine called "*BlinkLine()*" is used to implement this effect, which takes in three parameters, shouldFlash array (the line numbers of the TextMesh Pro game object that should be highlighted), duration (transition time) and the line array (the text of codes to be displayed). In the first for-loop, we add rich-text tags to the original text to indicate the color of specific lines. In the while-loop, we use interpolation value between the original text color and the red color to make a gradual turning effect. After we use the "Lerp" function to identify the "currentColor", we iterate each line in the TextMesh Pro that should be highlighted, firstly deleting its original rich-text tags using "Regex.Replace", then adding a new pair of rich-text tags with the "currentColor". Finally, we use the "Join" method to set the texts in TestMesh Pro to the newly created lines with new rich-text tags to achieve the goal of "gradually changing color to red" effect.

Codes turning red reference: *http://digitalnativestudios.com/textmeshpro/docs/rich-text/ [9]*

11. Final test function: condition control

- The Start() method initializes the quiz by creating the Question objects and setting up the answer and summary buttons. It then displays the first question.
- The OnClick() method is called when a player clicks an answer button. It determines which button was clicked and judges whether the answer is correct or not. It then updates the progress slider and displays the next question or the summary panel, depending on whether the quiz is complete. The OnSecondClick() method is called when a player clicks a button on the summary panel. If the player chooses to retry the quiz, it resets the progress and displays the first question. If the player chooses to return to the game scene, it starts a transition animation and then loads a new scene.

```
private void Start()
{
    summaryButtons = new Button[2];
    questions = new Question[6];
    questions[0] = new Question(
    @"Consider the following sequence of elements stored in st
    After executing the commands below,
    what will be the top 3 element ?
    Command 1 : stack.pop();
    Command 2: stack.pop();
    Command 3: stack.push(1);

    A. 5  B. 6  C 7
    ", 3);
    questions[1] = new Question(
    @"If inserting an element to the middle of the list,
h type of list is more efficient, LinkedList or ArrayList?
```

```
public void OnClick() //when ABC button be clicked
{
    var button = UnityEngine.EventSystems.EventSystem.current.currentSelectedGameObject;
    int choice = (button.name == "A") ? 1 : button.name == "B" ? 2 : 3;
    determine(choice, questions[currentQuestionIndex].answer);// judge corectness

    if (currentQuestionIndex < questions.Length)// show question if quiz not done
    {
        DisplayQuestion(currentQuestionIndex);
    }
    else //show summary page
    {
        for (int i = 0; i < 3; i++)
        {
            answerButtons[i].interactable = false;
        }
        ShowSummaryPage();
    }
```

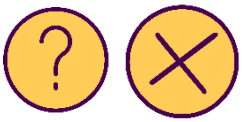
```
private void determine(int choice, int answer)
{
    progressBar.value++;
    if (choice == answer)
    {
        GetComponent<AudioSource>().clip = right;
        GetComponent<AudioSource>().Play();
        correctAnswers++;
        questions[currentQuestionIndex].correct = true;
        currentQuestionIndex++;
        animationController.PlayCheeringAnimation();
    }
    else
    {
        GetComponent<AudioSource>().clip = wrong;
        GetComponent<AudioSource>().Play();
        questions[currentQuestionIndex].correct = false;
        currentQuestionIndex++;
        animationController.PlaySadAnimation();
    }
```

- The determine() method is called from OnClick() to check if the answer is correct or not. If it is, it updates the progress and correctAnswers variables, sets the current question to be marked as correct, and plays a cheering animation. If it is not, it updates the current question to be marked as incorrect and plays a sad animation.

UI function:

Code reuse function for the UI. I made the hint button have two functions at once: one, to provide in-game hints, and two, to act as a close button for all overlay information pages (hint pages, task list pages). This took some time and thought to implement but improved the efficiency of the use of resources. Specifically, the individual levels were discussed in terms of situations and the current function of the hint button in each case was determined, such as the transition between the hint icon and the close icon.

```
(button.name == "hintButton")
{

    if (Question_mark.level_is_coupon == true)
    {
        if (hint_btn.image.sprite == close)
        {
            canvas_large.gameObject.SetActive(false);
            hint_btn.image.sprite = hint;
            taskbutton.interactable = true;
            if (Subtitle_coupon.Subtitling == true)//wring
            {
                hint_btn.gameObject.SetActive(false);
            }
            StartCoroutine("playClip_2");
            ok = false;

        }
        else if (hint_btn.image.sprite == hint)
        {
            StartCoroutine("playClip");
            ok = true;

        }
```



.

Game process control

Use multiple bool variables to monitor each level for completion, and use fixedupdate (no update, as there is no need for excessive detection frequency) for single detection (by only opening variables single time at the right time and closing them after they have been detected and operated accordingly.) Process. Each time a detection is made, the task list is updated accordingly.

```
void Start()
{
    doneCount = 0;
    basket =false;
    coupon1=false;
    coupon2=false;
    list1   =false;
    list2 = false;
    canvas =false;
    doneMark = Resources.Load<Image>("Prefabs/doneMark");

}
Unity Message | 0 references | Changed by 20676355d@connect.polyu.hk on Wednesday, April 12, 2023
private void FixedUpdate()
{
    if (basket == true)
    {
        doneCount++;
        Image newDM = Instantiate(doneMark, large.transform);
        newDM.GetComponent<RectTransform>().anchoredPosition = new Vector3(-256, 109, 0);
        newDM.transform.SetParent(large.transform);
        basket = false;
    }
    if(list1 == true)
    {
        TextMeshProUGUI newTM = Instantiate(list1_text, large.transform);
        newTM.GetComponent<RectTransform>().anchoredPosition = new Vector3(-492, -35, 0);
        newTM.transform.SetParent(large.transform);
        list1 = false;
    }
```

6. Project impact on Internet and Multimedia Technologies

This application is an innovative approach to teaching complex data structure concepts in a way that is engaging and relatable to students. By using a market scenario, the application can help students understand and retain knowledge better. The use of a real-life scenario that is commonly seen in our daily lives makes it easier for students to comprehend the concepts and apply them to various scenarios. The benefit of this approach is that abstract knowledge can be challenging to remember, and students can easily mix up different concepts. By incorporating game logic, learners can associate different concepts with specific game scenarios, which can reduce their memory load and help them retain the knowledge better. In addition to being useful for students who are currently taking the subject, the application is also accessible to individuals who have limited programming experience and are eager to expand their knowledge of data structures. The use of a market scenario provides a familiar context that can make the learning process less intimidating and more enjoyable.

In summary, this application offers an innovative and creative way of teaching complex data structure concepts that can help all the potential learner to understand and retain knowledge better. By connecting abstract concepts to real-life scenarios and game logic, learners can develop a deeper understanding of data structures and their applications.

7. Resource and teamwork management

DING Jiaqi	2D art assets for the game; UI design and implement and refining; Final quizzes function
HAO Jiadong	basket and coupon and advertisement board levels; level switching system; BGM; movement of characters.
ZHANG Chengcheng	list level; testing and debug; export to apk and test on the mobile device.
FAN Zhendi	3D model assets; camera system; behavioral characters animations.

8. Difficulties encountered

Problem 1: The basket cannot move properly

In the basket game, the expected animation of a basket moving from one basket stand to another should be divided into three stages, i.e., moving up, moving horizontally to the target basket stand and finally moving down. To make the transitions smoothly, we use the coroutine called “TransitionCoroutine”. However, we found that if we directly write the three coroutines in sequence in the update function, the transitions will be combined into one step and the effect is not what we want. By searching relevant webpages, we know that the problem is the three coroutines in the update functions will be initiated at the same time, so the moving position of the basket is mixing. How to separately execute the three coroutines one after another? We further looked for the website and found out the answer [10]. We can use another coroutine “WorkingSequence” to hold these three “TransitionCoroutine” by “yield return StartCoroutine(). This statement waits for the coroutine to finish before continuing with the next line. Hence, we realize the desired effect.

```
private IEnumerator WorkingSequence(Transform pop_basket, Transform push_Stack_trans)
{
    for (int a = 0; a < buttons.Length; a++)
    {
        buttons[a].interactable = false;
    }
    yield return StartCoroutine(TransitionCoroutine(pop_basket, pop_basket.localPosition, new Vector3(0, 1.7f, 0)));
    pop_basket.SetParent(push_Stack_trans);
    pop_basket.SetAsFirstSibling();
    yield return StartCoroutine(TransitionCoroutine(pop_basket, pop_basket.localPosition, new Vector3(0, 1.7f, 0)));
    int num_of_children = push_Stack_trans.childCount;
    yield return StartCoroutine(TransitionCoroutine(pop_basket, pop_basket.localPosition, new Vector3(0, 0.4f + 0.1f * (num_of_children - 1), 0)));
```

Problem 2: Unity Version Control (teamwork collaboration function) has raised a lot of conflictions when merging files from each team member. There are many times that we cannot merge two members work successfully so one member had to redo all his work which wastes a lot of human resources and time. However, as the project progresses, we gradually found out that the conflicts will occur only when two or more members are modifying the scene at the same time. Hence, by dividing the labor reasonably, we ensured that at a time, only one member should be working on the scene, others may work on UI design and script writing, hence, the conflicts were significantly reduced.

Problem 3: Camera obstruction by obstacles

When the camera follows the character, it may encounter obstacles such as walls, causing the view to be blocked. To address this issue, we modified the CameraController script to include obstacle detection using Raycast. By adding an obstacleLayer public variable, we can specify the layer containing the objects that may obstruct the camera view. In the FixedUpdate() method, after calculating the newPosition for the camera, we use Physics.Raycast to detect any collisions with objects in the obstacleLayer. If a collision is detected, we adjust the camera's newPosition to avoid the obstacle while maintaining the desired height. This ensures that the camera follows the character smoothly and avoids any obstacles in the game environment.

Problem 4: Coupon flashing effect out of sync

To highlight the two coupons under comparison, we use a coroutine called “FlashCoroutine” to add flashing effects on them. However, if we simply write the coroutine as the code we provided previously, the two coupons may not flash synchronously, which may downgrade the user experience. To solve this problem, we set a public variable “shareTimer” each time we start the two coupons’ flashing coroutines. And we calculated the elapsed time only based on the current time and the “shareTimer”, which greatly reduce the error as we use too many times of “Time.time” in the coroutine to do the calculation, which is updated in each frame and hard to synchronize in the coroutine.

```
private IEnumerator FlashCoroutine()
{
    while (isFlashing)
    {
        float elapsed = Time.time - sharedTimer; // Calculate the ela
        float t = Mathf.PingPong(elapsed, flashSpeed) / flashSpeed; /
        Color targetColor = Color.Lerp(originalColor, flashColor, t);
        mat.SetColor("_EmissionColor", targetColor);
        yield return null;
    }
    mat.SetColor("_EmissionColor", originalColor);
}
```

Problem 5: The synergy between subtitles and animation

There are several animation demonstrations together with subtitle illustration for each step in our game. To achieve this effect, we can divide it into two parts. Firstly, set a restriction that the subtitle number cannot exceed the step number. Secondly, avoid the flow continue when the object is keep moving or the subtitle is still popping out.

For the first restriction, we can allow subtitle class to get access to the steps parameter in ListGame class. And do nothing when the subtitle index is larger than steps number.

```
if(currentSubtitleIndex <= ListGame1.steps)
```

For the second restriction, we can set two parameters, isDisplaying and isMoving, to observe if subtitle is popping out or if object is moving corresponding. If any of them is true, do nothing even if user click on the screen.

9. Conclusion

In conclusion, our Unity 3D teaching aiding project presents an innovative approach to helping students learn Java programming. By utilizing a 3D mobile application that simulates a shopping mall environment, students can better comprehend complex topics. The application's gamified setting allows students to immerse themselves in real-world scenarios and practice critical thinking skills. The various missions provided by the application require players to apply their JAVA skill to real life cases, providing an interactive and engaging learning experience. Furthermore, the final quiz provides students with immediate feedback, helping them identify areas for improvement and reinforcing their learning. Overall, our project provides a unique and effective way to aid students in their EIE3320 Java course learning.

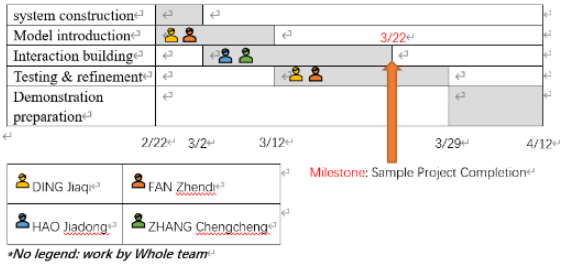
10. Distribution Table

Time	Member	Work
2/17~21	Whole team	Project brainstorm
2/21~22	Whole team	Proposal writing
2/23	FAN Zhendi	Imported 3D model construction for preliminary result
3/6	Whole team	Rough work distribution: HJD & ZCC: scripting for stack; FZD: character animation; DJQ: UI
3/8~11	DING Jiaqi	Imported asset: UI buttons, some backgrounds, Icons. Stack: 1, creation of basket (by Blender); 2, Hint image (by iPad Procreate) and button script.
3/8-12	HAO Jiadong	Complete building the first mini-game, that is used to the shopping baskets to illustrate the idea behind the stack in java.
3.13	HAO Jiadong	Character movement development: 1. Control movement by the joystick. 2. Walking animation. 3. Add collision detection between the character and the environment. 4.Add background music to the game 5.Add a camera following the character.
3.14	HAO Jiadong	1.Improve the game flow of the basket game. Add music effects and ending scenes. Change the question mark to a check mark after completion.

		2. Add distance limitation to the clickable objects "?"
3.14	DING Jiaqi	All the points include artworks (include online resource and hand-made images) and scripts: Create incorrect operation prompt for stack level; Level success animation; Optimize stack level logic; Create more button images; View all tasks function.
3/8-14	ZHANG Chengcheng and FAN Zhendi	Do Lab 1&2 and write the report.
3.15	Whole team	Fix the UI display problem on Android devices and camera jittering problem due to the camera following mechanism and tiny movements of the character after collision. Discussion on the implementation detail of the teaching of Array list and Linked list module.
3.25-28	DING Jiaqi	2D element updated; UI prefabs and script updated; Various bug fixed; adjustment applied.
3.24-28	HAO Jiadong	Design and build the entire coupon game, which illustrates the concepts of sorting algorithms (bubble sort and insertion sort). Work includes: 1. subtitle functions. 2. Flashing effect to the coupon so that students can see more intuitively the two objects being compared. 3. Animation of exchanging two coupons. 4. The little game implementation, the functionality of the hint button.
3.24-28	Zhendi FAN	1. Optimize the camera control script and character movement script, allowing the movement direction to be adjusted according to the camera angle, and fixing the bug of the camera. 2. Add idle and greeting animations 3. Add a picking up basket animation, and added a transition animation timeline.4. Modeled the tickets and fruit baskets. 5. Optimized some of the scene transition code.
3.19-28	ZHANG Chengcheng	Design and realize a fruit crate game to illustrate the characteristic and operation of the ArrayList and the LinkedList: 1. Find the order of the fruit crates by observing the location of a series of crates. 2. Use animation and narration to introduce the add and remove logic of ArrayList and LinkedList. Lastly, some questions to examine the learner's understanding. 3. Find the element in the list by a given index.
4.1	HAO Jiadong	Use a "shartTimer" to solve the problem that the flashing effects of the two coupons under comparison are not synchronized. Fix other bugs like the subtitle won't disappear after the game ends, the music is not properly played and muted and so on.
4.2	DING Jiaqi	Bugs fixed; Imported Canvas 3D model; designed final test and replay function
4.3	Whole group	Have a group meeting and make the work distribution of the following two weeks: hjd: canvas Teaching and Gaming zcc: list teaching and games fzd: Interactive animations at the beginning of two new games, celebrating and regretting animation djq: The final quiz
4.4	HAO Jiadong	Design the whole game of using the advertisement board to teach the JAVA GUI. Write down the JAVA code for drawing the market poster and test it on BlueJ.
4.9	DING Jiaqi	Help refine Canvas level (update UI images tasklist and canvas exceptation.png, add game global flow, sound effects (1) https://sc.chinaz.com/yinxiao/200112039932.htm; (2) https://sc.chinaz.com/yinxiao/230109284910.htm)

		Final test function done
4.5-4.9	HAO Jiadong	Finish the advertisement board game
4.8-10	ZHANG Chengcheng	Synthesize and export the needed model from blender and complete the second level of list game
4.11	FAN Zhendi	Make and assign the character animation for list game, advertisement game and final test

11. Project schedule



The right image is the schedule before. We followed the schedule with some flexible adjustments.

By March 29, we have done the two main levels and the UI design. From April 5th to 12, we continued working on the project, completed the remaining works (GUI and Graphics, List, Final test) while making necessary changes to ensure the feasibility of the project, also check in with team members to ensure everyone is aligned on the project's progress. Then we finalized the Unity project, re-export the apk file and prepared for the demonstration. Detailed contribution with time stamp please see the part 10.

12. Reference

[1] Merge cube official website: https://mergeedu.com/cube
[2] Anatomy 4D official website: https://www.visiblebody.com/apps/anatomy-and-physiology
[3] Metaverse official website: https://studio.gometa.io/landing
[4] Character movement and rotation: https://www.youtube.com/watch?v=BJzYGsMcy8Q
[5] Timeline reference: https://www.youtube.com/watch?v=G_uBFM3YUF4
[6] Fade in effect reference: https://stackoverflow.com/questions/64510141/using-a-coroutine-in-unity3d-to-fade-a-game-object-out-and-back-in-looping-inf
[7] Subtitle display reference: https://answers.unity.com/questions/1093166/display-rich-text-character-by-character.html
[8] Flashing effect: https://www.youtube.com/watch?v=DNMdu3kylec
[9] Codes turning red: *http://digitalnativestudios.com/textmeshpro/docs/rich-text/*
[10] Coroutines in sequence: https://forum.unity.com/threads/difference-between-yield-return-ienumerator-and-yield-return-startcoroutine.432571/
[11] UI buttons: https://www.vecteezy.com/vector-art/14070924-game-ui-buttons-for-app-interface-cartoon-plaques
[12] Some UI backgrounds: https://lovepik.com/download/detail/400329113?byso=&type=0
[13] Button icon: https://pngtree.com/freepng/info-icon-design--essential-icon-vector-design_4168845.html
[14] Character model resource: https://www.mixamo.com/#/
[15] Low Poly Fruits Pickups: https://assetstore.unity.com/packages/3d/props/food/low-poly-fruit-pickups-98135

13. Peer Evaluation

20075618d
The total amount of workload distribution is 22.86 %.

20084595d
The total amount of workload distribution is 22.86 %.

20101448d
The total amount of workload distribution is 30 %.

20076355d
The total amount of workload distribution is 24.28 %.

20075618d
The total amount of workload distribution is 30 %.

20101448d
The total amount of workload distribution is 20 %.

20076355d
The total amount of workload distribution is 20 %.

20084595d
The total amount of workload distribution is 30 %.

20084595d
The total amount of workload distribution is 28.57 %.

20101448d
The total amount of workload distribution is 23.57 %.

20076355d
The total amount of workload distribution is 23.57 %.

20075618d
The total amount of workload distribution is 24.29 %.

20075618d
The total amount of workload distribution is 22.86 %.

20084595d
The total amount of workload distribution is 22.14 %.

20076355d
The total amount of workload distribution is 30 %.

20101448d
The total amount of workload distribution is 25 %.