CS4240 INTERACTION DESIGN FOR VIRTUAL AND AUGMENTED REALITY

AY21/22 Semester 2

Project Report

Team 08

Student Name	**Student No.**
Bu Wen Jin	A0205129M
Hong Yi En, Ian	A0217685U
Koh Zheng Qiang Shawn	A0185892L
Siti Nurul Syasya Bte Azman	A0205371N
Wei Chen Kun	A0188758A

1. Project Description

Problem

Physically practising a sport has many implicit requirements and obstacles that may be hard to solve. Examples include the booking of sporting facilities, the difficulty scheduling with friends, the hassle of meeting COVID safe distancing requirements, the long travel time to facilities and more (Physiopedia, n.d.).

Value Proposition

By bringing table tennis into VR, our target audience, everyday non-competitive players, is able to overcome all of the previously stated difficulties. They can practice the sport anytime they want from the comfort of their own homes. Furthermore, they get to train consistently, while having fun in our gamified environment.

2. Project Design

Intuitive 3D UI

Every interaction with a menu is done using the ball and the paddle (with the exception of the Gesture System). By making the player interact directly with objects, the game seamlessly builds their mental model and teaches users how to play even before they start the tutorial. This interaction pattern was inspired by Cloudlands VR, a VR golf game.

Familiar and Interactive Menu

Upon launching the game, the player enters the start menu scene and is greeted with the option to play a:

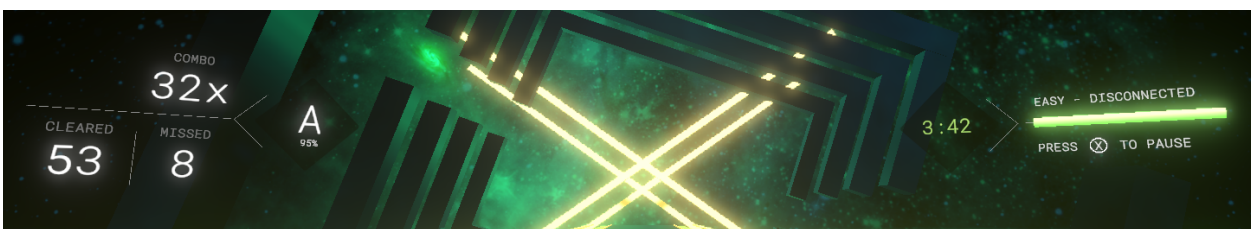
1) Tutorial to get to grips with using the ball and paddle
2) Short, easy song
3) Hard song which challenges the player

By designing the menu as part of the environment, the player gets the option of playing ping pong ball using the credits wall for fun.

Level Design

There are two songs with varying difficulties: easy and hard. It describes the frequency and number of targets that appear with the music.

When the song starts, the player is faced with hoops coming towards them that are synchronised with the beat of the music. The player needs to spawn a ping pong ball, and successfully shoot it through the hoop with the paddle in order to score a point. Players can measure how well they perform using the displays on the sides of the stage.



The performance metrics include:

- The accuracy of the player, and a corresponding letter grade for an immediate understanding of how they are doing
- A combo tracker, which resets to 0 when a player misses a hoop
- The number of hoops cleared and missed
- Song name
- Time left, displayed as a bar and in minutes and seconds

There is an element of strategy and risk in the game as well. A player can try to clear multiple hoops with a single ball. But if they are not careful, their combo will be reset if they let a hoop fly past them.

Preventing Motion Sickness

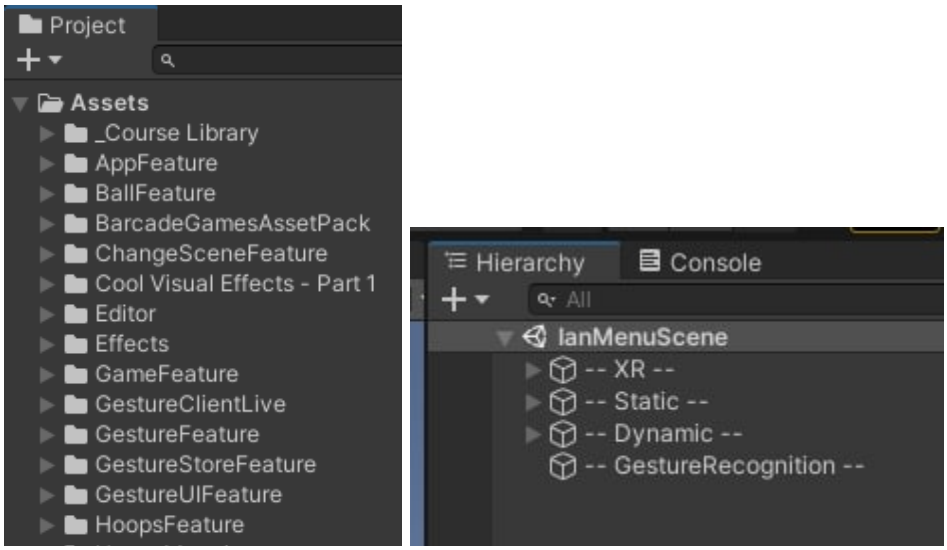
To reduce motion sickness, all locomotion in the game is accomplished by the player moving their real body. The scenes, controls and gameplay is designed such that the player does not need to travel at all. The game has many static reference points, with the targets being the only moving objects. Even then, they move towards the player so that players can have maximum comfort.

3. Project Implementation

The following shows how we standardised how we built the project so as to facilitate integration.

Folder Structure

To facilitate decoupling and good organisation, the code is organised into feature folders. All related scripts and prefabs needed for the feature are placed in the feature folder. Similarly, the feature modules are then grouped under headers in each scene in the figures below.



Software Architecture

Overall, we have 28 feature folders that are responsible for various functionalities in the game. They can be roughly divided into:

- Spin Gesture recognition system: **AppFeature, IdleFeature, TrainingFeature, GameFeature, GestureFeature, GestureStoreFeature, GestureUIFeature, InputFeature, PlayingFeature, RecordingFeature, and WristFeature**

- Scene changing system: **ChangeSceneFeature**
- Tutorial scene behaviour: **TutorialFeature**
- Model View Controller pattern for score behaviour: **ScoringFeature**
- Music playback control system: **SongControlFeature**

- Prefabs for player's hands: **PlayerFeature**
- Prefabs and behaviour of paddle: **RacketFeature**
- Prefabs and behaviour of ball: **BallFeature**
    - triggers scoring after interacting with paddle
- Prefabs and behaviour for hoops and mapping hoop timings to a song: **HoopsFeature, HoopsMappingFeature**

Event Channels

To coordinate events across features, scenes and prefabs, we developed our own custom Event Channels, which are scriptable objects comprised of C# Actions. These events can be invoked through APIs we provided in the scriptable object, and relevant methods can subscribe to them. This drastically reduced the complexity of code, because classes in different features do not need to know the names of methods in the classes of distantly-related features. This idea was taken from [this conference].

An example of what an Event Channel ScriptableObject looks like

```
using System;
using UnityEngine;

namespace ScoringFeature__MVC_pattern_
{
    /// <summary>
    /// Parent class for all other score events
    /// </summary>
    [CreateAssetMenu(menuName = "ScoreFeature/ScoreEventPublisher")]
    1 asset usage  3 usages  Ian Hong *  2 exposing APIs
    public class ScoreEventChannel : ScriptableObject
    {
        public Action OnScoredOnePoint;
        public Action OnScoredTwoPoints;
        public Action OnHoopMissed;

        1 usage  Ian Hong *
        public void ScoreOnePoint()
        {
            OnScoredOnePoint?.Invoke();
        }
        1 usage  Ian Hong
        public void ScoreTwoPoints()
        {
            OnScoredTwoPoints?.Invoke();
        }
        1 usage  Ian Hong
        public void MissHoop()
        {
            OnHoopMissed?.Invoke();
        }
    }
}
```

4. User Experience Evaluation

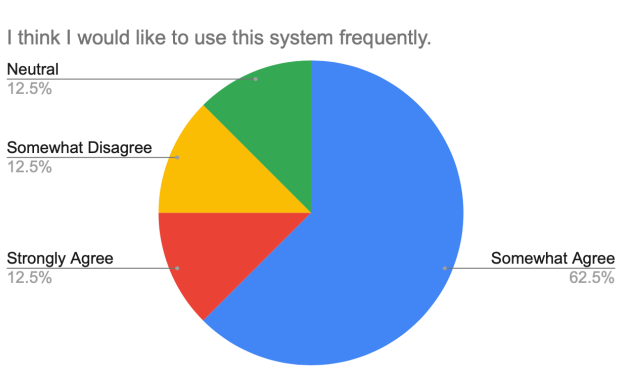
Using the SUS (System Usability Scale) Usability Questionnaire, we engaged peers from other teams to playtest HyperPong and received 8 responses in total. We are humbled by the positive majority responses despite our project having many aspects that were yet to be polished.

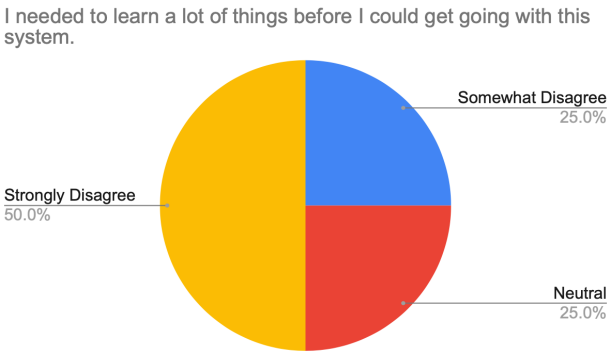
In combination with verbal elaborated feedback about the gameplay and user experience, we were glad that most of the users found the controls to be easy to use (see some of the visualised responses below). They experienced little to no difficulty with picking up the paddle and spawning ping pong balls. However, some of them did find it difficult to hit the ping pong ball through the hoops accurately. We felt that this was an important aspect to improve on, especially since our target users are looking to transfer their skills to a real game of table tennis. Hence, after the play testing sessions, we prioritised the implementation of a tutorial scene to give users a walkthrough of how to hit the ping pong balls with the paddle, so that they can perform well during the actual gameplay.

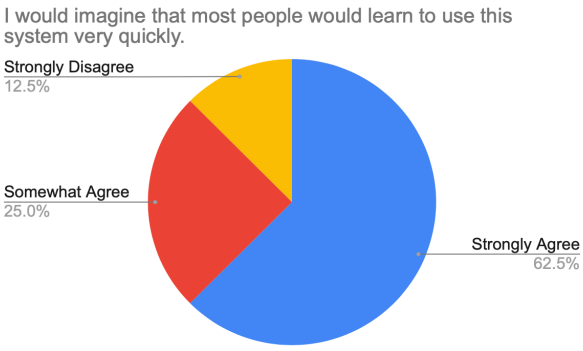
Our peers also provided notable feedback on the functionality of our system design. For example, users could hold the ball and score through the hoops without having to project it by hitting with the paddle. We then polished our system by requiring the ball to touch the paddle before it could score a point.

Overall, we received positive responses regarding the user experience which included aspects such as the music, environment and user controls. As a result, we decided to dedicate resources to build on the mentioned aspects to make our project even more immersive for users.

Below are some of the positive responses received from the questionnaire.







5. Challenges Faced

Version Handling and Collaboration

Due to the size of the working files and the size of the team, we decided to use GitLab with Git Large File Storage (LFS) to develop our project. However, since there were functions that not everyone was familiar with, we had to work through various difficulties using this platform, while we learnt as we went. We adopted the feature-branch workflow and only merged working branches to master, such that we always had a functional version of the game at any time.

Large Amount of Self Learning, Differing Programming Skill Levels

Most of us were new to VR development. Therefore, to materialise our vision for this project, a lot of self learning had to be done within the time that we had. We broke our vision into features and split the work. Some only handled post processing and vfx, while the others coded functionality for the game. Since each of us did very different things, we often did not have the expertise to understand and debug problems that we encountered in each other's features. It was also a challenge to integrate our features. For instance, one of our teammates coded in a reactive programming style, which was unfamiliar to the others. When it was time to integrate their feature into the other features, the other teammates could not understand, debug, or link up with their code on their own.

Underestimation of Task Difficulty

We vastly underestimated the complexity involved in building our original set of functionalities.

An example would be to improve paddle skills by practising gestures for trick shots. Training the model requires the typical ML attribute and model selection work. In an attempt to simplify the task, we sought to use a gesture recognition plugin that was supposed to offer a "plug and play" experience. However, their feature set was not designed to support our mechanics, so we had to create support mechanics to collect, modify and delete samples for hitting the ball with the paddle. These support mechanics alone were a huge undertaking due to the complex state management required.

In our initial attempt to build the support mechanics, we tried to adopt an event-oriented architecture, modelled after

- Unity Open Project #1: Chop Chop
- Unite Austin 2017 - Game Architecture with Scriptable Objects
- Unite 2016 - Overthrowing the MonoBehaviour Tyranny in a Glorious Scriptable Object Revolution

We chose this approach because we thought that it would strike a balance between the team (aside from Shawn) not having experience in reactive programming, and offering some semblance of primitive "streams" (through C# delegates) and event channels as scriptable objects. However, this got really messy, largely because of our inexperience with Unity and also because of the lack of established state management frameworks that serve this workflow.

Despite spending significant effort in building the support mechanics, progress was extremely slow and there were many bugs (treating delegates like streams, without having tools to actually manage the flow of streams was a recipe for disaster)

We found ourselves inevitably building tools that sought to manage the streams, essentially recreating slices of Rx.NET. At this point, the deadlines were fast approaching and a decision was made to abandon and rebuild the gesture feature from scratch, using an established reactive programming framework, UniRx.

This allowed us to finally focus on building the gesture mechanics rather than building menial micro-architecture tools to support our version of “streams”.

However, it was still difficult to manage the gesture mechanics because of the complex state management required, so we re-implemented a unidirectional state management framework in C# (see the UnityHelpers folder) based on swift-composable-architecture using emulated sum types via OneOf.

This truly unlocked progress and we rebuilt all the mechanics to support the training phase in only one day, and even managed to implement the ball curve when gesture recognised mechanics shortly after.

Just as we were wrapping up the gesture feature, we faced a *fatal*, unexpected issue with the gesture recognition plugin - we could not reliably train the model, because the plugin’s DLL (it was written in C++) very frequently did not invoke our callbacks. It appears to be an issue that is still unresolved in Unity, as mentioned in Problem with callbacks - Unity Forum.
We tried our best to resolve this but time was up - it was due for playtesting (hence why we requested to postpone our slot).

In hindsight, we should have put this feature aside despite it being our unique selling point when it started showing signs that it was too large an undertaking for a project of this short length, and instead, focused our efforts on other core mechanics of the game.

6. Future Work

If given the opportunity to extend this project, we would like to work on the following areas:

- Refine the holding position of the paddle
- Refine the physics applied to the ball so that it better emulates the Magnus Effect, which is the creation of a sideways force when a ball spins in the air (*Encyclopedia Britannica, 2020).*
- Encourage new players to practice spinning the ball in the tutorial scene
- Make the tutorial more engaging
- Automatically generate hoop patterns based on the amplitude and beats of new songs
- Increase variety by adding more songs and design more stage environments
- Add more feedback. This may include playing a sound when the ball hits the paddle, or making the stage light up when the player scores a point
- Defining modules via assembly definitions in order to remove cyclic dependencies.

7. Conclusion

Having scored 1st runner up for STePS assured that our efforts had not gone to waste despite encountering numerous failures along the way that taught us to be better developers after each one. This experience was enriching and had certainly opened our eyes to VR/AR development.

8. References

Barriers to physical activity. (n.d.). Physiopedia. Retrieved 20 April 2022, from https://www.physio-pedia.com/Barriers_to_Physical_Activity

Britannica, T. Editors of Encyclopaedia (2020, February 6). *Magnus effect. Encyclopedia Britannica*. https://www.britannica.com/science/Magnus-effect