TEMPORAL TETHER TENNIS

Above: Environment Blender Render

Grads in Games

SEARCH FOR A STAR

2023 Programming Challenge Entry

by Joe Bevis

Contents

Introduction

Welcome to the supporting documentation for *Temporal Tether Tennis,* my original game submission for the *Search for a Star 2023* programming competition.

For this challenge, participating students were given a base project made with Unreal Engine 5 & C++ and were expected to expand and adapt on the original code base to create a finished game.

It is expected of industry professionals to be able to work with code authored by colleagues. Being accustomed to creating games from scratch, I had never properly experienced adapting existing code before and I was excited to begin.

The Base Project

At first, the base project was an enigma to me. There were two circles and a sphere and a box. One of the circles moved around very slowly. Titled *STB,* I genuinely had no clue what the game was meant to be.

Later, I discovered that *STB* stands for 'Spot the Ball' which is a type of gambling game in which the player must identify the location of the invisible ball using hints from visible characters' body language. This began to add up given that the project included lots of animations for a football character.

Above: STB Game Example [1]

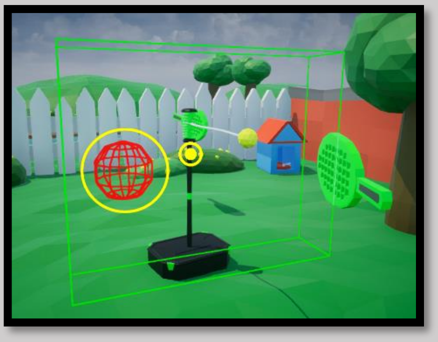
However, this is not the type of game that I wanted to make for the challenge. By the point of discovering the true meaning of *STB*, I had already decided that I wished to create an action game designed to test the player's reflexes.

Enter *Temporal Tether Tennis*, a *Swingball* inspired game with time dilation where the player's goal is to hit the ball back before it gets past and ultimately get the string to the top of the Archimedes screw.

Above: Swingball Set [2]

Design Choices

After seeing the circle moving about very slowly, I decided early on to increase its movement speed. There is something innately fun about moving a circle around the screen quickly, so why not turn this into a game mechanic?


Above: Debug Shape Draw Turned On

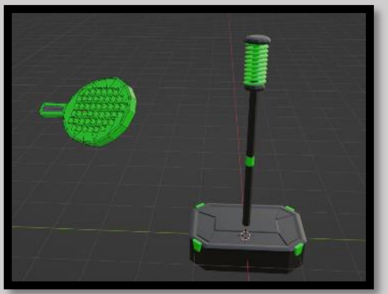
While wrapping my head around the existing code of the base project, I discovered that the current game aim was to move the player-controlled circle into an invisible circle, which has its position determined by the randomised location of a ball represented by a debug sphere.

I wanted to reuse as much of the existing code base as possible. This led to the idea of the moving the cursor into a *visible* target location in a limited amount of time. Could this be an exciting mechanic?

Next, I found myself thinking *'Why would the player need to keep moving themselves into a randomised location in limited time?'*. The circular pattern of *Swingball* made sense to me. In real life, the ball comes round in a wildly varied orbit which requires the player to adapt. This seemed to fit the design I wanted perfectly.

Asset Creation

While I realise it is an unconventional approach to development, once I knew the design for the game, I first went about 3D modelling a low-poly garden environment and *Swingball* set in Blender.


Above: Swingball Set Models

There are multiple reasons why I did this. First, I knew that I needed bespoke assets for this unique game idea. Second, I enjoy having art assets to work with during development because it reminds me what I am working towards. Third, I was slightly intimidated by Unreal engine C++ and wanted to do something else before jumping straight into the deep end…

I also created **audio assets** for the game:

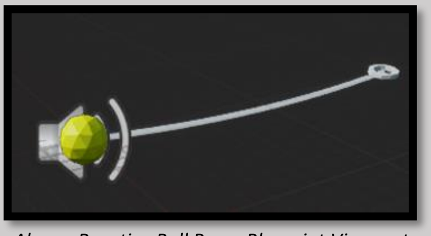
For the air sound effects, I used *DSP Motion* by *Tsugi* to draw the sounds that I needed. This process is very satisfying and I would highly recommend the software.

I made the background music using a MIDI keyboard and *ACID Music Studio* with plugins by *AAS (Applied Acoustics Systems).*

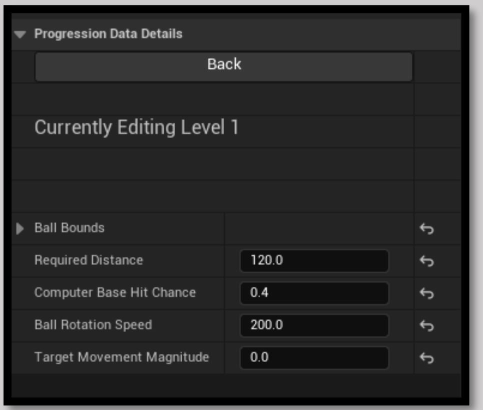
Above: Drawing Sound Effects in DSP Motion

Development Challenges

The focus during the first stages of development was getting the ball to rotate around the pole and be able to change direction. To solve this problem, I created a custom *C++ pawn* class that contains a static mesh, offset in a derived blueprint. The actor is then rotated around its root centred on the pole at runtime. To create the string, I set up a cable component in the blueprint and adjusted the settings.


Above: Reactive Ball Pawn Blueprint Viewport

Next, I made the AI opponent. While testing the game, something felt wrong at first because the chance of the ball being hit back was a fixed value. Taking inspiration from real life Swingball, I realised that on subsequent circulations the ball is much easier to hit back, so after the first pass the chance of the computer hitting back the ball now increases.



Currently Editing Level 1

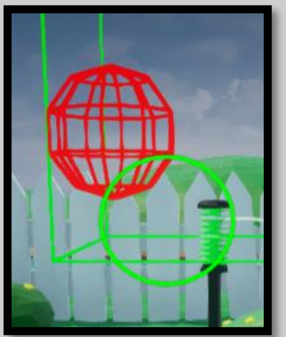
Ball Bounds	
Required Distance	120.0
Computer Base Hit Chance	0.4
Ball Rotation Speed	200.0
Target Movement Magnitude	0.0

Above: Level Data Editing Process

I carefully considered the design behind the difficulty curve – I wanted to be able to change the rotation speed easily as well as other factors such as the bounding box and AI bat hit chance. I made sure certain properties were easily accessible, and then implemented level data loading functionality to allow me to build a difficulty curve. I successfully modified the existing custom editor code for level progression data as seen on the left.

After that, I focused on making the UI clean and accessible, to be used easily with keyboard & mouse OR any gamepad. Building on the existing code and deriving from the base *UScreen* class, I created a Credits and Win screen. I also added button click functionality by hooking up the buttons in each UMG blueprint, making use of the *UPROPERTY(BlueprintCallable)* macro.

One of the biggest hurdles I came across during development was the seemingly random offset of the target circle from the ball's *DebugSphere* location on screen. I eventually figured out that this was being caused by the window scale: *ProjectWorldLocationToScreen(…)* converts the 3D location as viewed into a 2D screen space vector given in pixels that it would be **if** the game were running at fullscreen 1080p. To fix this, I adapted the original code to factor in the window scale as well as set the moving image anchor points to the top left (origin) instead of the centre.


Above: Target Offset Issue

To make the game feel more satisfying to play, I added crunchy sound effects and a particle effect when the ball is hit, all of which are played or activated via C++. On top of this, I added custom attenuation settings for the ball air-displacement looping sound to make it sound like the ball is whizzing in front of the player's face as it comes past. A continuous modulation node lowers the pitch in accordance with the time dilation of the rotating pawn – otherwise this sound feels out of place during the slow-motion effect.

Game Testing

My university accommodation flatmate was kind enough to playtest the game in front of me. It was extremely valuable to see how a non-gamer / casual gamer would play the game. While it was picked up quite quickly, there were some useful feedback points:

1. **Without realising enter hits the ball back** Why can't I use spacebar?
2. Keyboard is so much harder than gamepad!
3. Would it be cool to be able to use the mouse?
4. How many levels are there?
5. How do I win? **Wins** Oh, it's like real *Swingball*!

I made changes based on these points:

1. Added the spacebar as another select button in project settings.
2. For keyboard play, ensured that required distance (target size) is not too small, and slowed down the rotation speed slightly for levels that feature a smaller target.
3. Mouse control was considered: it was not implemented as I believe it would make the game too easy / fast paced.
4. Added total levels to the current level indicator.
5. I plan to highlight the win conditions on the *itch.io* page.

Summary & Conclusion

Overall, taking part in the *SFAS Programming challenge 2023* has been excellent. I have learned how to adapt existing code and I've improved my *Unreal Engine 5* and C++ programming knowledge. On top if this, I believe I have managed to turn the base project into a fun, polished and playable game.

Throughout the development journey, I always kept a notepad beside me to jot down ideas and improvements before implementing them. While this worked well for me, I would like to be able to plan out the project better if I were to take part in a competition like this again.

If I were to continue development, I would consider adding a multiplayer mode to the game: in real life Swingball, part is the fun is playing against someone you know! Thank you for reading.

Image References

[1] Coote, A., Kissane, E., Manchester, S., & Taylor, R. (2014, July 18). *How We Made "Spot the Ball"*. Retrieved from Source: https://source.opennews.org/articles/spot-ball/

[2] Mallon, J. (2012, June 26). *Review: Hey ho, Let's go Swingball Pro*. Retrieved from Opposable Thumbs: https://opposablethumbsblog.com/2012/06/review-hey-ho-lets-go-swingball-pro/