Game Specialism - Programming

By Corey Shepherd

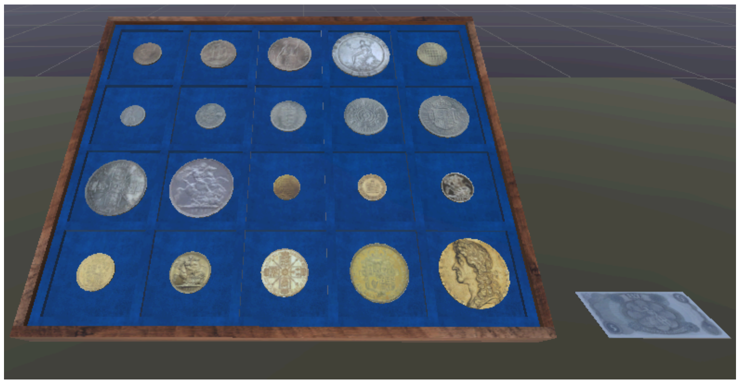
This reflective document is to outline the development process of my projects and my skills. Creating these projects was really enjoyable and I was really motivated to get my projects completed to the best of my ability.

Old Money System

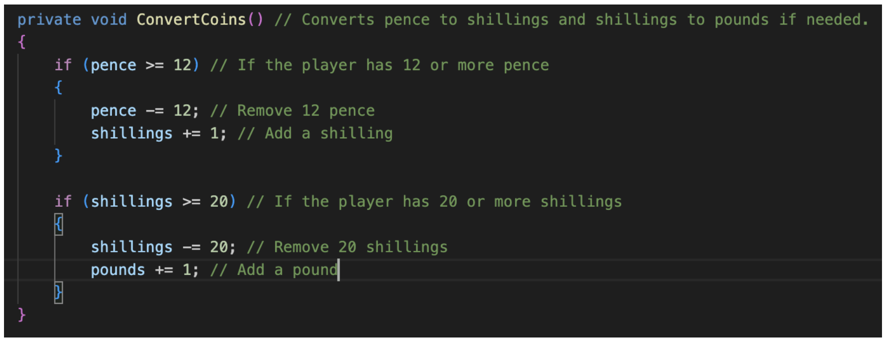
To get started, I did some research into the pre-decimal currency system in the UK. I found some information on Wikipedia which was quite useful for understanding how the system worked and how to notate it.

I started off by taking a first person player controller I made previously and put it into the scene. This script also included a way to interact with objects, so I set that up quickly as I have done for previous projects, and was able to detect the player's interaction with the coins easily.

Next, exiting my comfort zone, I created my own 3D models in order to have something visual and physical to use when I was programming and debugging. I did some research on all the specific coins and tried to make each one as size accurate as possible.



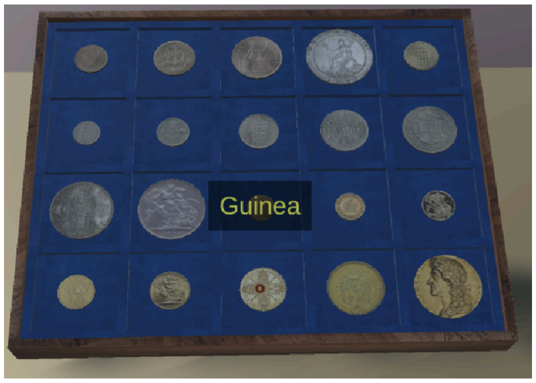
After this, I needed to give the player the correct amount of money when a coin was interacted with. To do this, I figured out how much each of the coins were worth in pennies and assigned each coin their respective value in pennies. In a Singleton class called GameManger, I was able to increase the players' pennies by the value in which they picked up from the table. I then created a function in order to convert the players pennies into shillings and pounds so it can be displayed correctly later.



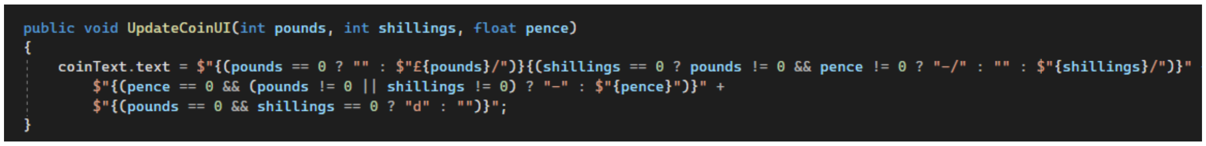
```
private void ConvertCoins() // Converts pence to shillings and shillings to pounds if needed.
{
    if (pence >= 12) // If the player has 12 or more pence
    {
        pence -= 12; // Remove 12 pence
        shillings += 1; // Add a shilling
    }

    if (shillings >= 20) // If the player has 20 or more shillings
    {
        shillings -= 20; // Remove 20 shillings
        pounds += 1; // Add a pound
    }
}
```

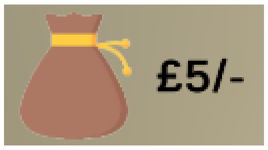
I realised that I couldn’t tell what coins were what just from looking at the design. In order to resolve this I created a tooltip system and added a crosshair in the centre of the screen so you could see where you are aiming and what coin you are looking at.



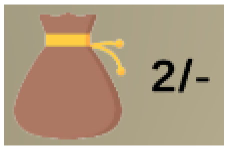
Adding on to the user interface, I created some text in order to display the amount of money the player has. I made sure the notation was correct by using string interpolation and ternary operators to change the output depending on what the player has. Creating this took a bit of time as I found it quite complex to wrap my head around.



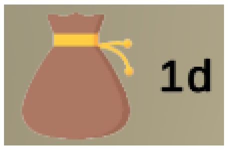
```
public void UpdateCoinUI(int pounds, int shillings, float pence)
{
    coinText.text = $"{(pounds == 0 ? "" : $"£{pounds}/")}{(shillings == 0 ? pounds != 0 && pence != 0 ? "-/" : "" : $"{shillings}/")}"
        $"{(pence == 0 && (pounds != 0 || shillings != 0) ? "-" : $"{pence}")}" +
        $"{(pounds == 0 && shillings == 0 ? "d" : "")}";
}
```



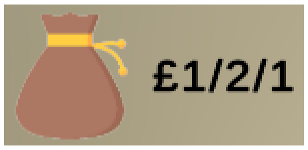
Five pounds



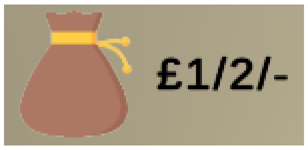
Two shillings



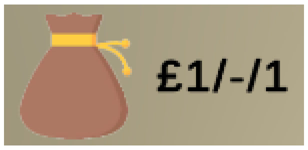
One pence



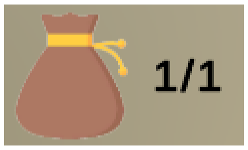
One pound, two shillings, and one pence



One pound and two shillings



One pound and one pence



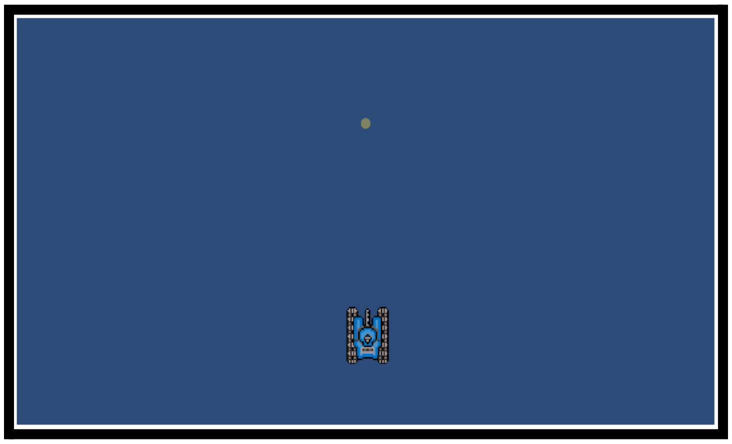
One shilling and one pence

To further improve this system, I could add functionality for the system to know the exact coins the player has. For example, if they pick up a shilling and a florin then the system knows that's what they have as opposed to just knowing they have 3 shillings.

Hit a Moving Target

I decided to increase the challenge a little by going for an advanced brief. I chose this one as it was very maths oriented and would help me in developing my skills even further.

Firstly, I created a scene with a tank and a player that could be controlled by getting the horizontal and vertical inputs. Next I created a bullet prefab and created a script to make the bullet constantly move forward at a set speed. I then did some research on how to get this system working and got to work on making the script for the targeting system.

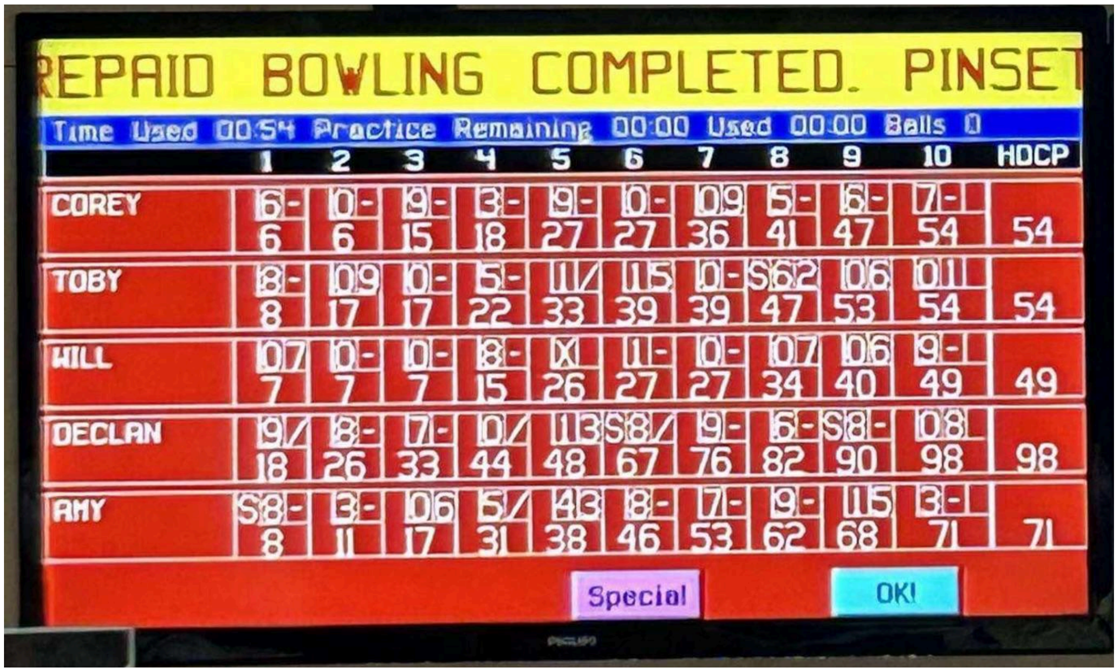


The first thing I did was to make the barrel aim directly at the player. To calculate the direction from A to B you have to take away B from A. So to get the direction of the player from the tank I had to take away the tanks position from the players. At first I did this calculation the wrong way around so all the next calculations were also done slightly incorrect. Eventually, I fixed this and now I have the correct direction vector, but I need to now calculate the angle for the barrel to turn.

To do this I used an inbuilt unity function: Mathf.Atan2. Simply, if you pass in the y and x value of the direction (in that order) you will get back the angle in radians. To convert radians into degrees I multiplied the return value by another inbuilt unity function, Mathf.Rad2Deg. Now, this float value can be used to rotate correctly to target ahead of the player.

Figuring out how to use these maths functions properly took me a lot of trial and error and I was constantly tweaking values. One thing I had the hardest time with was getting the tank barrel in the correct orientation.

Ten Pin Bowling
I chose this as my last brief as it was a bit more of a fun one. The first thing I did was take me and my friends bowling to get some scores I could use as an input to the system for testing.


REPAID BOWLING COMPLETED. PINSET

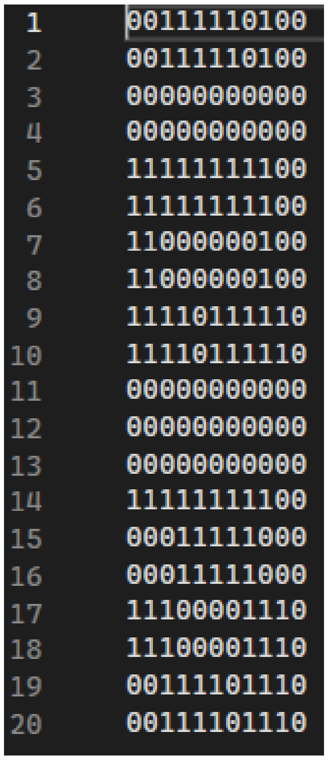
Time Used 00:54 Practice Remaining 00:00 Used 00:00 Balls 0

	1	2	3	4	5	6	7	8	9	10	HDCP
COREY	6 -	0 -	9 -	3 -	9 -	0 -	0 9	5 -	6 -	7 -	
	6	6	15	18	27	27	36	41	47	54	54
TOBY	8 -	0 9	0 -	5 -	1 /	1 5	0 -	S6 2	0 6	0 1	
	8	17	17	22	33	39	39	47	53	54	54
WILL	0 7	0 -	0 -	8 -	X	1 -	0 -	0 7	0 6	9 -	
	7	7	7	15	26	27	27	34	40	49	49
DECLAN	9 /	8 -	7 -	0 /	1 3	S8 /	9 -	6 -	S8 -	0 8	
	18	26	33	44	48	67	76	82	90	98	98
AMY	S8 -	3 -	0 6	5 /	4 3	8 -	7 -	9 -	1 5	3 -	
	8	11	17	31	38	46	53	62	68	71	71

The first thing I did was research how ten pin bowling was officially scored. After doing this, I realised that it was a lot more complicated than I initially thought.

I was really determined to use recursion for this project as this way made the most sense to me given how bowling is scored. To see if this way would work I decided to attempt to correctly score the "perfect game" which is 12 strikes in a row. I managed to do this quite easily but realised that I made it harder for myself to score everything else.

So, keeping what I had just done in mind, I restarted and started by scoring a game with no spares or strikes. This was fairly easy to do, and I managed to figure out how to do it recursively fairly quickly. I ran into a few overflow errors from where I was wrongly iterating over the string of 0s and 1s.


```
00111110100
00111110100
00000000000
00000000000
11111111100
11111111100
11000000100
11000000100
11110111110
11110111110
00000000000
00000000000
00000000000
11111111100
00011111000
00011111000
11100001110
11100001110
00111101110
00111101110
```

I fixed that issue and then reimplemented the strike scoring from the system I had before and further implemented scoring for spares which was extremely similar.

Overall this brief was really tough as I decided to use iteration but it really helped actually playing a bowling game at the start.