DDU2 spiludvikling reflektionsaflevering

Jeg har i løbet af dette forløb lavet både den grønne tutorial, hvor man skulle få en bil til at køre, den grønne tutorials ’challenge’, hvor man skulle få et fly til at flyve rundt, og den orange opgave, hvor man efter at have fulgt tutorialen til at lave spillet ’John Lemon’s Haunted Jaunt’, så skulle ’udvide’ det eksisterende spil og ’gøre det til sit eget’.

Den grønne tutorial viste sig at være lidt for nem, men jeg lærte dog stadig noget, og den forståelse jeg fik for Unity igennem denne tutorial gjorde mig i stand til bedre at se, hvorfor diverse ting i den orange tutorial blev gjort som de blev, og hvordan man selv kunne have gjort det anderledes.

Et eksempel på forskellene mellem de to tutorialer er kameraet. I den orange tutorial blev jeg instrueret i at bruge ’cinemachine’-pakken til at lave et kamera, der blandt mange andre funktioner kunne sættes til at følge efter spillets hovedperson, John Lemon.

I den grønne tutorial gik det lidt anderledes til. Hvor John Lemons implementering af et kamera der kan følge spilleren var programmeringsmæssigt high-level, så var den grønne tutorials tilgang til problemet mere lowlevel, mere kodenær. Her skulle man nemlig ’selv’ (tutorialen hjalp selvfølgelig) skrive et behaviourscript i C#, som skulle styre kameraets position ift. bilen, som man skulle køre.

Alt i alt var resultatet det samme: Et kamera, der følger spillerkarakteren med en bestemt afstand, og som ikke vender sig sammen med spillerkarakteren. Jeg har som min udvidelse til begge spil (ikke at det var et krav til den grønne opgave - der havde jeg bare lyst til at gøre det) gjort kameraerne i stand til at rotere sammen med spillerkarakteren, og i John Lemon-spillets tilfælde har jeg også i den forbindelse ændret spillets perspektiv og måde at bevæge sig på fra en slags ’top-down’/fugleperspektiv (som man f.eks. ser i spil som præ-3D pokemon) til et tredjepersonsperspektiv a la Dark souls.

I den grønne opgave vidste jeg endnu ikke noget om hvordan forskellige spilobjekter kunne gøres til hinandens børn og derved nedarve lokale koordinatsystemer, hvor origo er sat til forældreobjektets position i det globale koordinatsystem, hvilket ville have gjort det fuldstændig unødvendigt at lave et kamera-script til at starte med. I alt fald endte jeg med at finde ud af, at når man ganger en normaliseret rotation (som har datatypen quaternion, hvilket jeg efter at have læst lidt om det fatter mindre af, end da jeg startede) med en 3D-vektor, får man den samme vektor roteret til at pege i samme retning som rotationen, hvilket er smart, når man styrer afstanden mellem kameraet og bilen med selvsamme 3D-vektor.

At implementere et tredjepersons-kamera i John Lemon-spillet var noget sværere. På dette tidspunkt havde jeg via et uheld med at gøre spøgelsernes waypoints til børn af et spilobjekt, hvis position ikke var i det globale origo, lært på den hårde måde, hvordan børns positioner defineres ift. deres forældres, og det at sætte kameraets position og rotation ift. John Lemon kunne dermed let gøres ved bare at gøre mit kamera-objekt til et af John Lemon-objektets børn. Dette er sikkert dårlig praksis, og det gjorde kortvarigt hele konceptet med at bruge cinemachine-pakken lidt ubrugeligt, men det virkede.

Dette kom dog med nye problemer. John var sat til altid at gå i samme retning, når man trykkede på en tast, f.eks. ville han gå langs med den globale x-akse, hvis man trykkede på a- eller d-tasterne, uanset hvad vej han og kameraet så vendte.

I bil-spillets tilfælde var dette blevet fikset ved at lade a- og d-tasterne styre bilens rotation, og så køre i den retning bilen pegede med w og d. Dette var dog utilfredsstillende når spillerkarakteren var et menneske (eller i hvert fald en menneske-citrusfrugt-hybrid med arme og ben), og dermed i modsætning til en bil sagtens burde kunne gå sidelæns. Derfor følte jeg mig nødsaget til at implementere to nye features: Evnen til at strafing, altså til at gå sidelæns ift. kameraet, og evnen til at bruge musen til at kigge rundt med.

Det gjorde jeg, kort sagt, ved at lege lidt med de indbyggede variabler transform.right og transform.forward, der, i Unity’s egne ord er enhedsvektorer der beskriver de pile, som man ser når man highlighter et spilobjekt i Unity. Efter man finder ud af, hvad vej spilleren trækker musen, roteres transform.forward mod enten transform.right (hvis musen trækkes mod højre) eller transform.right \* -1 (hvis musen trækkes mod venstre). Så laves der en rotation ud fra denne delvist roterede vektor, Johns rotation sættes lig med denne, og da kameraet er barn til John, så følger det også med. Johns position opdateres også dejligt let ved at gange outputtet fra ’a-d’ og ’w-s’ tasteparrene (disse værdier går hver især fra -1 til 1 afhængig af hvilken tast man trykker på og hvor længe, og er lig 0 hvis man hverken trykker på den ene eller den anden tast) med hhv. transform.right og transform.forward.

Til sidst tilføjede jeg en cameracollider-komponent til mit virtual camera-objekt for at undgå at kameraet fløj gennem vægge, og man dermed ikke kunne se John. Dermed viste cinemachinepakken endelig sit værd med en funktionalitet, som ville tage mere end 20 sekunders makværk for mig at implementere (om end jeg med den viden om raycasting som John Lemon-tutorialen har givet mig nok godt *kunne* stable noget lignende på benene).

Note til hvis du åbner bilspillet: Ja, det ser ret scuffed ud, men jeg ville gerne have mindre compiletid, så jeg fjerene