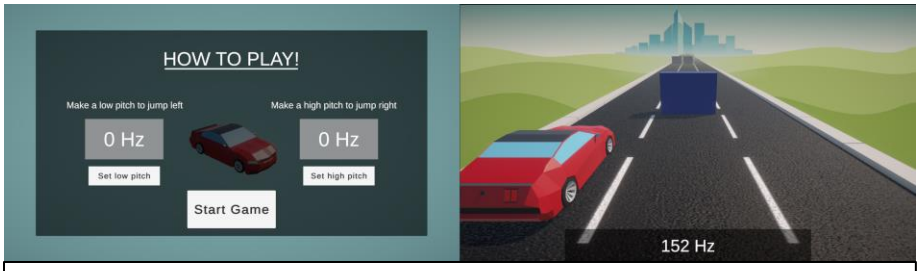
Adam Nilsson, Adani731
Anton Martic, Antma001
Marcus Skoglund, Marsk090

Användning av tonhöjds-detektering för styrning i ett bil-spel

1. Inledning

Ljudfysik är ett område som utforskar ljudvågors beteende och interaktioner, inklusive deras generering, fortplantning och perception. Denna vetenskap har inte bara teoretiska tillämpningar, utan även praktiska inom teknologins värld. Ett exempel är ljudanalys i digitala applikationer, där tekniker som röstigenkänning och tonhöjdsdetektion (tonhöjds-detektering) används. I detta projekt har vi använt ljudfysik för att skapa ett interaktivt bil-spel där bilen fördas på en trefilig väg och möts av olika hinder som ska undvikas genom röststyrning. Innan spelaren startar spelet ska en kort kalibrering utföras för att få så bra och personlig spelupplevelse som möjligt för spelaren. Genom att analysera tonhöjden i spelarens röst med en tonhöjds-detekterings-algoritm kan spelarens vokala frekvens kopplas till spelets dynamik: en högre ton för att få bilen att byta till körfältet höger om bilen och en lägre ton för det vänstra. En bild bestående av spelet och dess meny visas i Figur 1. Syftet med projektet är att undersöka hur effektivt röststyrning kan tillämpas i spel och för att få en fördjupad förståelse av realtidsanalys av ljuddata.



Figur 1. Visar menyn till vänster och spelet till höger

2. Material och teknik

För att realisera idén användes Unity, en mångsidig och kraftfull spelmotor som är populär inom spelutveckling. Unity erbjuder inte bara ett omfattande bibliotek av verktyg och komponenter för spelutveckling, utan även stöd för ljudhantering, vilket var behövligt för projektet. Programmeringsspråket C# används för att hantera spelets logik, då det är Unitys inbyggda språk och ger stor flexibilitet i att skriva både logik och ljudbehandling.

Blender som är ett kraftfullt och öppet 3d-modelleringsprogram användes i projektet. Programmet erbjuder verktyg för att skapa detaljerade och realistiska objekt med olika former och texturer, vilket gav oss stor flexibilitet i att skapa en spelvärld som både såg bra ut och fungerade tekniskt.

3. Metod

3.1 Skapandet av spelmiljön

Först skapades en menyskärm där spelaren kunde kalibrera spelet utefter spelarens låga och höga toner. Detta gjordes enkelt med 2st knappar där den första var avsedd för att ge en direkt feedback på den låga tonen spelaren gjorde, medan knapp 2 gjorde detsamma för den höga tonen. Det skapades även en tredje knapp som startade spelet.

Därefter skapades själva spelmiljön i Blender, där både bilen och vägen modellerades. Blender var det primära verktyget för att designa de visuella elementen i spelet. Bilen designades med enkla former för att underlätta designen och för att möjliggöra bättre prestanda i spelet, medan vägen modellerades med texturer som en bana där hindren senare skulle placeras.

I Unity skapades sedan hindren i form av boxar som placerades procedurellt på vägen, och därefter skapades en funktion som flyttade vägen och hindren mot spelaren med en hastighet som ökade med tiden.

3.2 Mikrofon input

För att kunna styra bilen i spelet med röst behövdes ljud fångas upp i realtid. Detta gjordes genom Unitys inbyggda AudioSource-komponent som tog in spelarens röst via mikrofonen. Med hjälp av AudioSource-komponenten kunde sedan den insamlade ljudsignalen bearbetas direkt i spelet för att sedan kunna analysera och extrahera tonhöjden (pitch).

För att undvika att bakgrundsljud påverkade spelet implementerades en GetLoudness-funktion. Denna funktion analyserade ljudnivån genom att den hämtade data från ljudklippet i tidsdomänen och därefter beräknade den genomsnittliga ljudstyrkan över ett bestämt antal sampels. Funktionen säkerställde att endast ljudnivåer som översteg en viss tröskel användes för styrningen.

3.3 Tidsdomän till frekvensdomän

När ljuddatan togs emot från mikrofonen behöves den omvandlas till frekvensdomänen för att möjliggöra olika operationer på frekvensinnehållet. Detta gjordes med hjälp av en Fast Fourier Transform (FFT), som konverterade ljudet från tidsdomänen till frekvensdomänen. I Unity användes den inbyggda funktionen “GetSpectrumData()” för att utföra denna

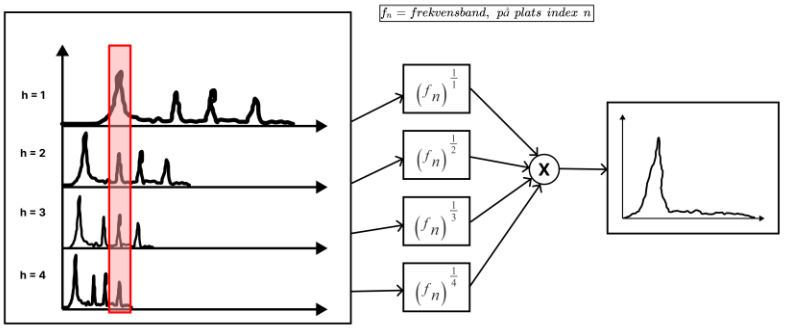
omvandling på det inkommande mikrofonljudet, vilket resulterade i ett frekvensspektrum [1]. Denna operation utfördes med en samplingsfrekvens på 48 kHz (mikrofonens upptagningsfrekvens), och resultatet returnerades som en array, “spectrumData”, med 4096 element som representerar olika frekvensband.

För att effektivisera beräkningarna ytterligare valdes det att filtrera bort samtliga frekvensband som inte var nödvändiga. Mer specifikt är det de frekvenser som inte kunde skapas av människan och kunde bestå av brus. På grund av detta valdes en nedre tröskel på 70Hz och en övre på 17 kHz [2].

3.4 Harmonic Product Spectrum (HPS)

För att förbättra precisionen ytterligare användes analysmetoden Harmonic Product Spectrum (HPS). HPS är en metod för att identifiera den fundamentala frekvensen (grundfrekvensen) i spelarens röst genom att analysera de harmoniska frekvenserna. Detta gav en tydligare bild av den dominerande tonen trots eventuella övertoner, vilket är särskilt viktigt eftersom mänskligt tal ofta innehåller många övertoner som kan påverka analysen.

Arrayen “spectrumData” skalades ner till nya arrayer (i detta fall är det 4 arrayer) där varje array representerade ett harmoniskt steg av frekvenser (f/2, f/3, f/4). Dessa nyskapade arrayer multiplicerades sedan element för element med den ursprungliga arrayen “spectrumData”. Denna process förstärker den fundamentala frekvensen genom att dämpa effekten av övertoner, vilket ger en tydligare bild av grundtonen. Detta resulterade i fungerande tonhöjdsdetektion, men det visade sig ge vissa störningar i detektionen. För att undvika detta introducerades en exponent i multiplikationen av de harmoniska stegen. Vid varje harmoniskt steg h multiplicerades varje element med exponenten $\frac{1}{h}$. Detta gör att varje frekvenskomponent justeras proportionellt till sitt harmoniska steg, vilket visade sig ge en mer stabil detektion. Figur 2 illustrerar denna process.

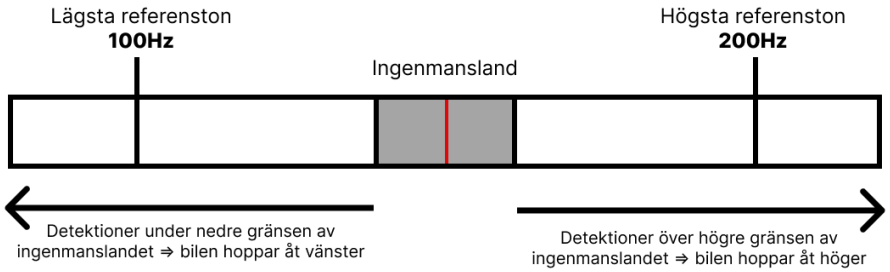


Figur 2. Visar processen av HPS, tillsammans med en viktning för varje harmoniskt steg.

Efter att alla harmoniska steg hade beräknats, analyserades den nya arrayen för att hitta den maximala amplituden och dess index. Indexen representerade dess frekvensband som den maximala amplituden låg i, det vill säga vilken hertz det motsvarade [3].

3.5 Mappa detekterad tonhöjd till bilens styrning

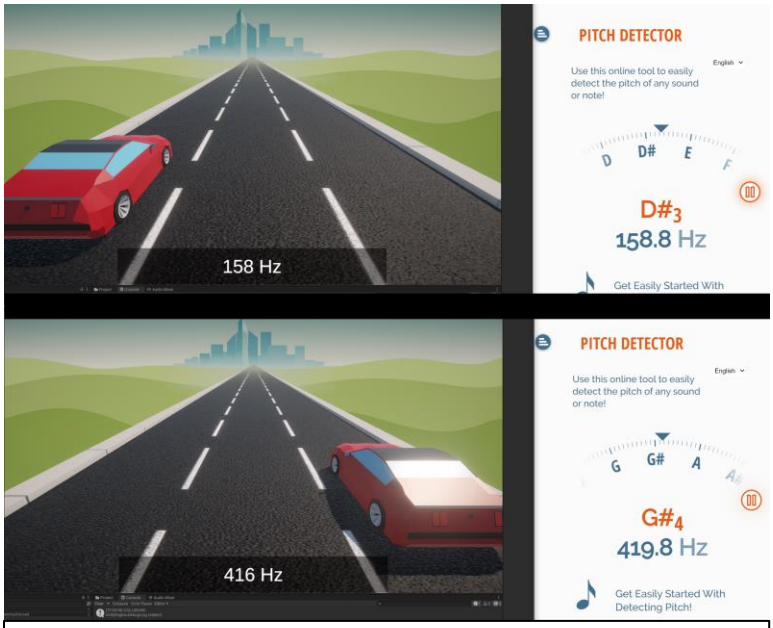
Den detekterande tonhöjden kunde sedan användas för att styra bilen i spelet. Detta genom att beräkna avståndet mellan de två kalibrerade referenstonerna som definierades i meny-delen innan spelets start. Referenserna bestod av en lägre och en högre frekvens. För att förbättra upplevelsen och inte få en frekvens där bilen svängde beroende på vilken sida tonen var på just en enskild frekvens, infördes ett ingenmansland. Ingenmanslandet konstruerades genom att skapa ett frekvensområde på 20% utav distansen mellan de kalibrerade referenstonerna som placerades i mitten. Om den detekterade tonhöjden var lägre än Ingenmanslandet, hoppade bilen åt vänster, och om den var högre än Ingenmanslandet, styrde bilen åt höger. Se Figur 3 för ett förtydligande. Denna dynamik gjorde det möjligt att intuitivt parera mellan hindren med spelarens egen röst.



Figur 3. Representerar frekvensmappning till styrningen av

4. Resultat

Resultatet av projektet är ett bil-spel där spelaren styr bilens rörelser med sin röst. Den utvecklade algoritmen för tonhöjds-detektion gör att spelet kan reagera på spelarens röst i realtid: en hög ton får bilen att byta körfält åt höger, medan en låg ton får den att byta körfält åt vänster. Efter testning konstaterades det att algoritmen detektera tonhöjder i intervallet 70 Hz till 1000 Hz, vilket täcker de flesta spelares röstfrekvenser. I Figur 4 visas två experiment som utfördes för att utvärdera tonhöjds-detektionens noggrannhet vid olika frekvenser. Figuren visar 4 stycken olika tonhöjds-detektioner där 2 olika frekvenser använts, en i spelet jämfört med en befintlig tonhöjds-detekterare som finns tillgänglig online [4].



Figur 4. Visar den implementerade tonhöjdsdetektionen vid två tillfällen men frekvenserna

5. Diskussion/Slutsats

Som resultatet visar blir tonhöjdsdetektionen bra om man jämför med redan existerande tonhöjdsdetektioner. Men när vi integrerar den i spelet, får vi störningar när spelaren gör höga frekvenser. Dessa störningar är att programmet alltid kommer detektera en låg frekvens innan den detekterar den rätta höga frekvensen och detta är inte optimalt för spelet då spelaren vill ha korrekta uppfattade signaler för att enkelt kunna styra sin bil. Problemet uppstår då vi som människor ska producera ett ljud, vi kommer automatiskt börja i de låga frekvenserna och sedan höja dem till vår önskade slutfrekvens. Detta lyckades vi visa genom att starta spelet med micken avstängd samtidigt som vi producerade en ljus ton och när vi väl startade micken uppfattade spelet tonen direkt som en ljus ton då vi eliminerade början av ljudet genom att ha micken avstängd.

Under projektets gång hade vi stora problem med att använda en fullständig HPS och få den att detektera korrekta tonhöjder. När detektionen kördes fick vi alltid frekvenser mellan 12 kHz och 20 kHz, vilket inte är nära ett korrekt resultat. Därför testades en annan metod som vi började kalla för HSS (Harmonic Sum Spectrum). Den presterade betydligt bättre än HPS:n, där den gav korrekta detekteringar men var mycket mer ostabil än den version vi har nu. För att lösa problemet med HPS:n krävdes det att vi initierar spectrumData-arrayn precis innan FFT genom “GetSpectrumData()”. Det krävdes även att

vid varje harmoniskt steg multiplicera varje element med exponenten $\frac{1}{h}$ för att dämpa inverkan av de högre harmoniska frekvenserna. Efter att vi ändrade detta verkar det som att vi får ett mer stabilt resultat med en feldetektion mellan 0.8Hz och 3.8Hz, vilket syns i Figur 4 i resultatavsnittet.

Som tidigare nämnt, uppstår problem när spelaren gör ljusa toner, då kommer en mörk ton att detekteras först. Då detta har en inverkan på spelet så skulle vi behöva göra en lösning som inte tar hänsyn till den första detekterade tonen men som ändå håller spelet responsivt. Ett sätt att eventuellt lösa detta skulle kunna vara att slänga bort det första detekterade värdet när man uppnått tröskelvärdet för upptag. En annan lösning skulle kunna vara att jämföra den nuvarande detekterade tonhöjden med den föregående och se om de ligger inom ett godtyckligt intervall från varandra och då tillåta det nuvarande värdet att påverka bilens körning.

Avslutningsvis visar projektet att röststyrning potentiellt kan vara en effektiv metod för att styra spelmekaniken. Genom att analysera spelarens röstfrekvens i realtid och mappa den till bilens styrning. Även om vi stötte på utmaningar, såsom detektering av felaktiga frekvenser vid snabba förändringar i tonhöjd, lyckades vi utveckla en algoritm som fungerar godtyckligt. Projektet har gett oss en fördjupad förståelse av realtidsljudanalys, speciellt vad gäller användningen av tonhöjds-detektering och dess tillämpning i spelutveckling.

6. Referenser

[1] Unity Technologies (2024), *AudioSource.GetSpectrumData,* Tillgänglig: https://docs.unity3d.com/ScriptReference/AudioSource.GetSpectrumData.html (Hämtad: 2024-10-03)

[2] Sound Engineering Academy (2024), *FREKVENSOMRÅDE FÖR HUMAN RÖST*, Tillgänglig: https://seaindia-in.translate.goog/blogs/human-voice-frequency-range/?_x_tr_sl=en&_x_tr_tl=sv&_x_tr_hl=sv&_x_tr_pto=rq (Hämtad: 2024-10-10)

[3] Smyth, T. (2019), *Harmonic Product Spectrum (HPS)*, Tillgänglig: http://musicweb.ucsd.edu/~trsmyth/analysis/Harmonic_Product_Spectrum.html (Hämtad: 2024-10-05)

[4] Online Mic Test. (2024), *Pitch Detector*, Tillgänglig: https://www.onlinemictest.com/tuners/pitch-detector/ (Hämtad: 2024-10-12)