VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D HRA V UNITY S VYUŽITÍM ANALÝZY HUDBY

3D GAME IN UNITY WITH MUSIC ANALYSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAKUB PETRJANOŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Petrjanoš Jakub**
Program: Informační technologie
Název: **3D Hra v Unity s využitím analýzy hudby**
3D Game in Unity with Music Analysis
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s metodami analýzy zvukových signálů.
2. Seznamte se s enginem Unity a procedurálním generováním grafiky.
3. Popište vybrané metody a algoritmy pro analýzu hudby a pro procedurální generování.
4. Implementujte 3D hru pomocí Unity, která bude využívat analýzu hudby.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek/video pro prezentování projektu.

Literatura:

- Dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce pojednává o zpracování hudby a jejím využití při tvorbě hry v herním engine Unity. Pomocí analýzy hudby je ve hře řešeno hledání cesty přes procedurálně generovaný terén. Při vývoji byly použity experimentální postupy a výsledkem je hra, která podle vybrané skladby vygeneruje trať v procedurálně vytvořeném terénu a vizuálně reaguje na prvky hudby.

Abstract

This thesis is about music analysis and it's usage in game development in game engine Unity. With help of music analysis there is implemented path finding in procedural generated terrain. In development were used experimental methods and result is game, which generates path based on chosen song through procedural generated terrain and visualizes music.

Klíčová slova

3D hra, Unity, herní engine, herní mechaniky, hudba, beat, analýza hudby, Fourierova analýza, Fourierova transformace, spektrální odečítání, procedurální generování, terén, cesta, hledání cesty, shader, collider, zorné pole, Bézierova křivka, C#

Keywords

3D game, Unity, game engine, game mechanics, music, beat, music analysis, Fourier analysis, Fourier transform, spectral flux, procedural generating, terrain, path, path finding, shader, collider, field of view, Bézier curve, C#

Citace

PETRJANOŠ, Jakub. *3D Hra v Unity s využitím analýzy hudby.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

3D Hra v Unity s využitím analýzy hudby

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Petrjanoš
12. května 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu panu Ing. Tomáši Miletovi za rady, věcné připomínky, ochotu, čas a směrování při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvoření 3D hry v herním engine Unity s využitím analýzy hudby a na základě této analýzy hledat cestu terénem a vizualizovat prvky hudby v samotné hře.

Hry všeho druhu byly odedávna součástí lidské kultury. Před vznikem a rozmachem elektronických zařízení to byly převážně hry deskové, které se i nyní těší stále velké oblibě. V dnešní době má téměř každý člověk přístup k zařízení, na kterém se dají hrát hry, ať už je to počítač, telefon, tablet, konzole... Tím se zvyšuje počet lidí, kteří tyto elektronické hry hrají, a proto taktéž vzniká velké množství nových her. Aby se hra v této velké záplavě prosadila, je potřeba něčím zaujmout. Některé hry zaujmou nádherným vizuálem (např. české Machinarium), některé nápadem (např. Hellblade: Senua's Sacrifice) a některé zase třeba zakomponováním hudby (např. Audiosurf).

Hudba je také neodmyslitelnou součástí lidské kultury. A stejně jako hry se rozmachem elektronických zařízení rozšířila mezi širokou veřejnost. Dříve, před vznikem audiozáznamů, mohli lidé hudbu poslouchat pouze naživo. Dnes má každý člověk v kapse zařízení, do kterého si může nahrát nezměrné množství hudby, nebo si ji může poslouchat z rádia či hudebních služeb (např. Spotify). Na hudbě je nádherné to, jak je rozmanitá.

Tato rozmanitost hudby je skvělá vlastnost pro využití hudby při tvorbě hry. Žádné dvě skladby nejsou totožné, pouze v případě coveru mohou být velice podobné. V kombinaci s procedurálním generováním to dává příležitost k nezměrnému množství unikátních úrovní hry, aniž by samotná hra zabrala velké místo v paměti.

Pro tvorbu samotné hry je vhodné využít herní engine. Herní engine usnadní vývoj hry. Je jich celá řada a za zmínku stojí právě Unity, které je vhodné pro tzv. nezávislé vývojáře, Unreal Engine, který je velmi oblíbený a rozšířený, CryEngine (např. Crysis) a Frostbite (např. Battlefield).

Kapitola 2

Teorie

V této kapitole, respektive v jejích podkapitolách, budou popsány a vysvětleny potřebné pojmy k pochopení dalšího textu.

2.1 Procedurální generování

Procedurální generování obsahu ve hrách je zde [6] definováno jako automatický způsob vytváření obsahu hry pomocí algoritmů. Obvykle bývá zapotřebí minimální interakce od člověka, například spuštění generování.

Procedurálně generovaným obsahem ve hře může být např. terén, vegetace, města, úrovně hry atd. Vstupem pro procedurání generování jsou předem definovaná data, která jsou pomocí algoritmů transformována a výsledkem je vygenerovaný obsah.

Dříve vývojáři ve hrách museli pracovat s omezenými zdroji paměti uložiště, proto se někteří snažili vytvářet herní úrovně až při načítání například Diablo 1 z roku 1996. Herní úrovně zde byly generovány „náhodně“ (v uvozovkách proto, že nejsou náhodné, ale pseudonáhodné, o tom ale až dále). Dnes se procedurální generování využívá k tomu, aby se ušetřila spousta práce vytváření rozsáhlé mapy, kdy se mapa/prostředí vygeneruje a designér pak ručně doladí. Nebo se používá například pro vytváření „náhodných“ úrovní, které u hráče prověří reakce v předem neznámém prostředí.

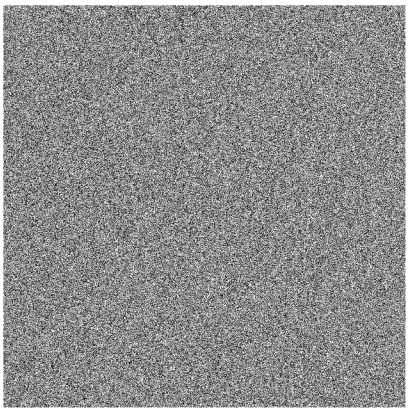
Techniky procedurálního generování můžeme obecně rozdělit do dvou kategorií:

- bodově orientované techniky - tyto techniky pracují s jednotlivými body v N-rozměrné matici (např. šumy)
- strukturově orientované techniky - pracují s abstraktními strukturami (tyto techniky nejsou obsahem této práce)

2.1.1 Šumy

Obecně se jedná o náhodný signál, někdy se mu rozumí jako znečištění (obrazu, zvuku...). Tento signál je náhodný například v elektronice, ale v informatice se většinou jedná o pseudonáhodné signály, protože na skutečně stochastické jevy je potřeba specializovaný hardware. Vizualizace šumu – obrázek 2.1.

[1] https://www.publicdomainpictures.net/en/view-image.php?image=201932&picture=tv-noise



Obrázek 2.1: Vizualizace šumu. Obrázek převzatý z internetu[1].

2.1.2 Generátory pseudonáhodných čísel

Generátor pseudonáhodných čísel je deterministický algoritmus, který generuje posloupnost čísel. Je snaha, aby se tato posloupnost nedala statickými testy rozlišit od skutečně náhodné. Po nějaké době se však začne generovaná sekvence pro dané inicializační vstupy opakovat. Tento problém se dá řešit tím, že se inicializace generování provádí častěji, než začnou být sekvence čísel periodické. Počet čísel před periodickým opakováním je vcelku velký, takže pro většinu případů jsou tyto sekvence čísel dostačující. Vstupní data pro inicializaci generátoru jsou nazývána semínko (seed), pokud jsou vstupem pouze tato data, pro stejné dvě vstupní data bude výsledná sekvence čísel totožná. Tento jev nemusí být někdy žádoucí, ale v tomto případě je tato vlastnost vítána (o tom ale až dále).

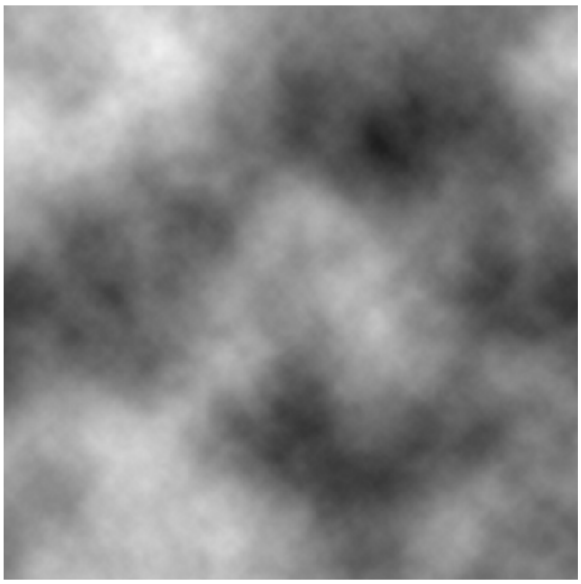
2.1.3 Perlinův šum

Problém u pseudonáhodného šumu je, že pro realisticky vypadající objekty, terén atd. není příliš vhodný z toho důvodu, že hodnoty nemají mezi sebou žádnou spojitost. Výsledek pak vypadá nepřirozeně a nerealisticky. S řešením tohoto problému přišel Ken Perlin, když vyvinul Perlinův šum (Perlin noise) [3].

Perlinův šum (obrázek 2.2) je založen na šumových funkcích. Vzniká součtem stejné šumové funkce o různé frekvenci a amplitudě. Používá se v prostoru (x,y,z), nebo s časem (x,y,t). Perlinův šum v čase je zajímavou možností jak nasimulovat přirozený pohyb například trávy[2].

Počtu šumových funkcí se říká oktávy (octaves), pořadí této funkce je oktáva (octave). Frekvence dané oktávy je většinou základní frekvence krát mocnina dvou. Amplituda je pak určena základní amplitudou a perzistencí (persistence). Perzistence se volí menší než 1, protože chceme, aby oktávy o větší frekvenci měly menší amplitudu, a díky tomu pak nemají na výslednou hodnotu takový vliv a tvoří spíše drobnější nerovnosti (u terénu). Jednotlivé oktávy (obrázek 2.3) a výsledek složených šumových funkcí (obrázek 2.4).

[2] https://www.youtube.com/watch?v=L_Bzcw9tqTc
[3] https://blog.mbedded.ninja/mathematics/perlin-noise
[4] https://flafla2.github.io/2014/08/09/perlinnoise.html
[5] https://flafla2.github.io/2014/08/09/perlinnoise.html



Obrázek 2.2: Perlinův šum. Obrázek převzatý z internetu[3].

2.1.4 Bézierova křivka

Bézierova křivka [4] (obrázek 2.5) je parametrická křivka. Tvar křivky je určen pomocí řídících bodů. Bézierova křivka těmito body neprochází, ale vytváří aproximační křivku. U těchto bodů lze nastavit váhu, která udává, jak moc je křivka ovlivněna právě tímto bodem.

Podle počtu těchto řídících bodů Bézierovy křivky rozdělujeme na:

- lineární Bézierovy křivky - dva řídící body
- kvadratické Bézierovy křivky - tři řídící body
- kubické Bézierovy křivky - čtyři řídící body

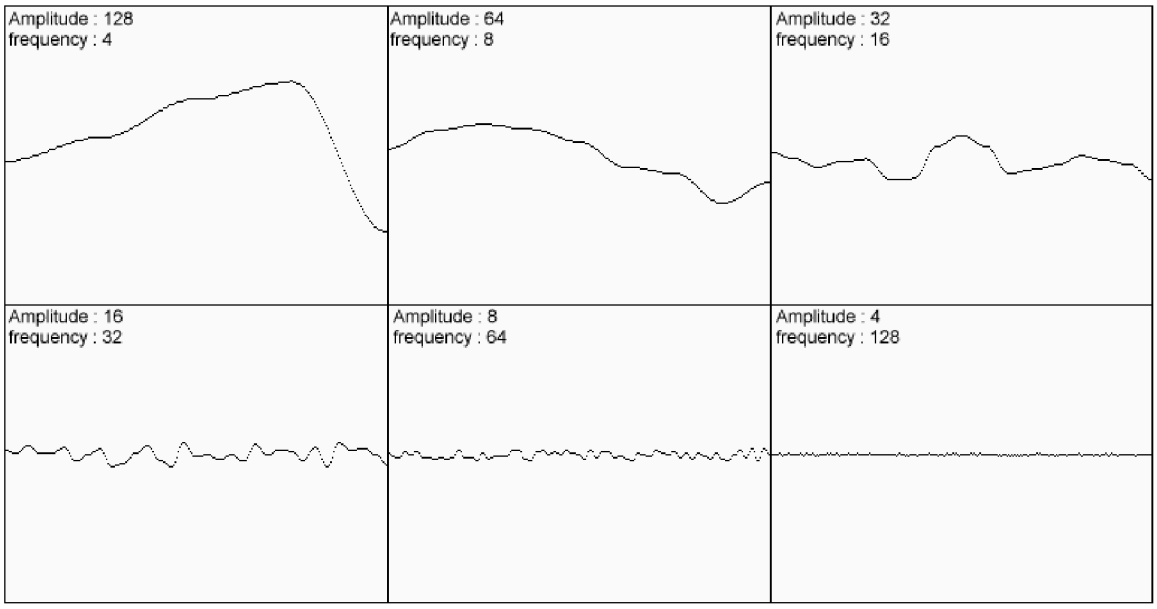
2.2 Analýza hudby

Běžný signál se skládá z více signálů o různých frekvencích. Pro podrobnější pohled na tento signál je zapotřebí od sebe tyto frekvenční složky oddělit a k tomu slouží spektrální analýza. Algoritmy, které slouží ke spektrální analýze se nazývají Fourieorova analýza [2].

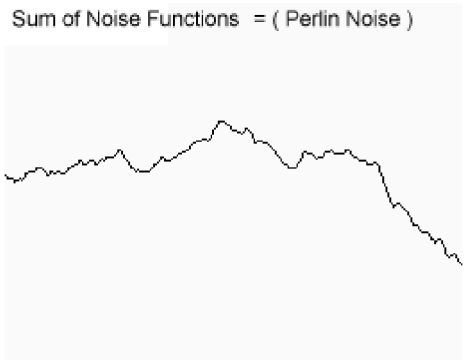
2.2.1 Fourierova analýza

Fourierova analýza je jeden z nejčastěji používaných postupů ve zpracování signálů a je využíván ve více vědeckých oborech. Fourierova analýza je metoda, pomocí které se převádí signál vzorkovaný v čase na informace o tom, které frekvenční složky jsou přítomny v daném signálu.

[7] https://en.wikipedia.org/wiki/B%C3%A9zier_curve



Obrázek 2.3: Na obrázku jsou vidět vizualizace jednotlivých oktáv. Převzato z internetu[4].



Obrázek 2.4: Výsledek složení jednotlivých oktáv. Převzato z internetu[5].

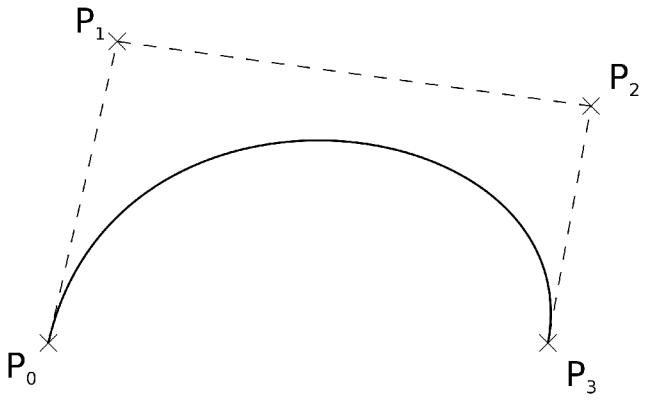
Můžeme ji rozdělit na 4 kategorie:

- pro periodický signál ve spojitém čase
- pro periodický signál v diskrétním čase
- pro neperiodický signál ve spojitém čase
- pro neperiodický signál v diskrétním čase

2.2.2 Fourierova transformace

Fourierova transformace nám umožňuje pracovat s neperiodickým signálem nebo funkcí. Umožňuje to tím, že se na neperiodický signál/funkci dívá jako na periodickou, s periodou s časem nekonečno.

Výsledkem jsou informace o tom, které frekvence se v daném signálu vyskytují, ale chybí informace o tom, kdy se daná frekvence v signálu vyskytla.



Obrázek 2.5: Na obrázku je vidět kubická Bézierova křivka a její řídící body. Převzato z internetu[7].

2.2.3 Diskrétní Fourierova transformace a Rychlá Fourierova transformace

Tato Fourierova transformace pracuje v diskrétním čase.

Rychlá Fourierova transformace (FFT) je vysoce efektivní výpočet Diskrétní Fourierovi transformace. Je založena na postupném rozdělování vstupní posloupnosti na poloviny až k dílkům o délce 1. Z tohoto důvodu je FFT nejoptimálnější, pokud je na vzorku o délce mocniny 2.

Nyquistův teorém nám říká, že maximální hodnotu, kterou můžeme detekovat je polovina vzorkovací frekvence. Pokud máme vzorkovací frekvenci 48 000 Hz, pak je možné detekovat maximální hodnotu 24 000 Hz.

2.2.4 Spektrální odečítání

Spektrální odečítání (dále v textu spectral flux) je odečítání (difference) každé složky vzorků v daném kusu vzorků, s odpovídající složkou vzorků v předchozím kusu vzorků spektra.

Po provedení tohoto postupu na celý signál, získáme spectral flux funkci.

Beat

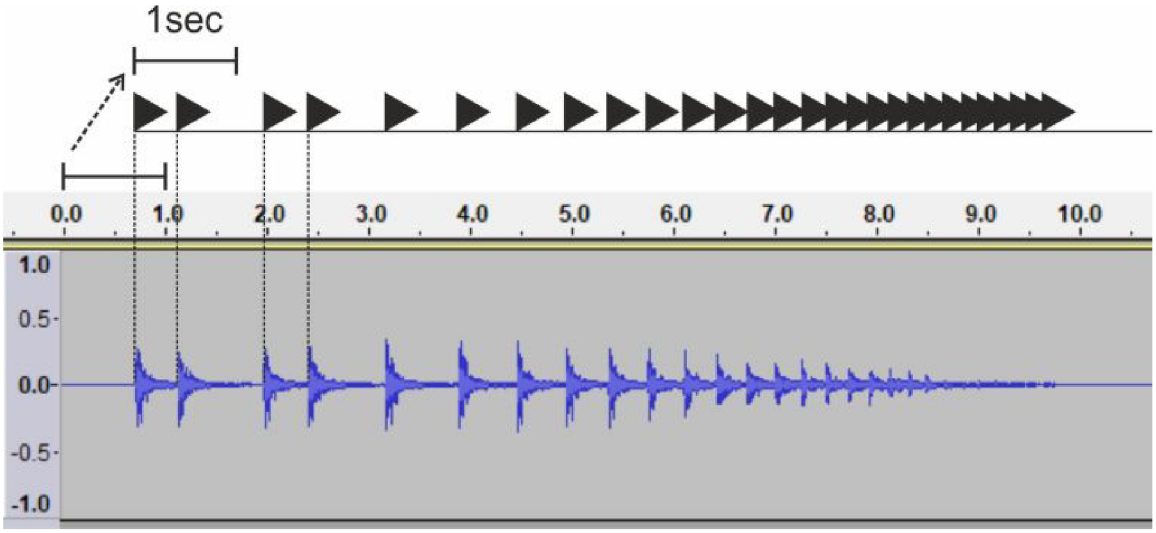
Beat lze definovat jako rytmický úder nebo pohyb. V hudbě časový rozestup mezi beaty určuje tempo písně (obrázek 2.6).

2.3 Herní engine

Vývojáři herního enginu Unity jej definují jako [7]: Herní engine je software, který poskytuje herním vývojářům potřebnou sadu funkcionalit k tomu, aby vytvářeli hry rychle a efektivně.

Herní engine je v podstatě jakýsi framework, který vývojářům umožňuje se odprostit od základních věcí ve hře, jako jsou například vykreslování scény, vykreslování objektů nebo například osvětlení. Tím dovoluje to, aby se mohl vývojář soustředit na tvorbu samotné hry a jejích mechanik. Moderní herní enginy nabízejí možnost vytvořit ohromující grafiku

[9] http://globalnotation.org.uk/measuring-onset-timing-with-a-waveform-graph/



Obrázek 2.6: Na obrázku jsou vizualizované beaty a časové rozestupy mezi nimi. Tyto rozestupy určují tempo. Obrázek převzatý z internetu[9].

ve hře (například Frostbite engine – Battlefield V, obrázek 2.7). Nemusí být použit jen na tvorbu hry. Velké využití herních enginů se nachází taktéž při tvorbě videosekvencí, ať už jako cutscene v samotné hře, nebo pro tvorbu filmu jako takového[10].

Obrázek 2.7: Ukázka ze hry Battlefield V. Hra je vytvořena ve Frostbite engine. Převzato z internetu[12]

[10]Příklad videosekvence vytvořené v UnrealEngine https://www.youtube.com/watch?v=0GLbwkfhYZk
[12]https://www.techspot.com/review/1754-battlefield-5-cpu-multiplayer-bench/

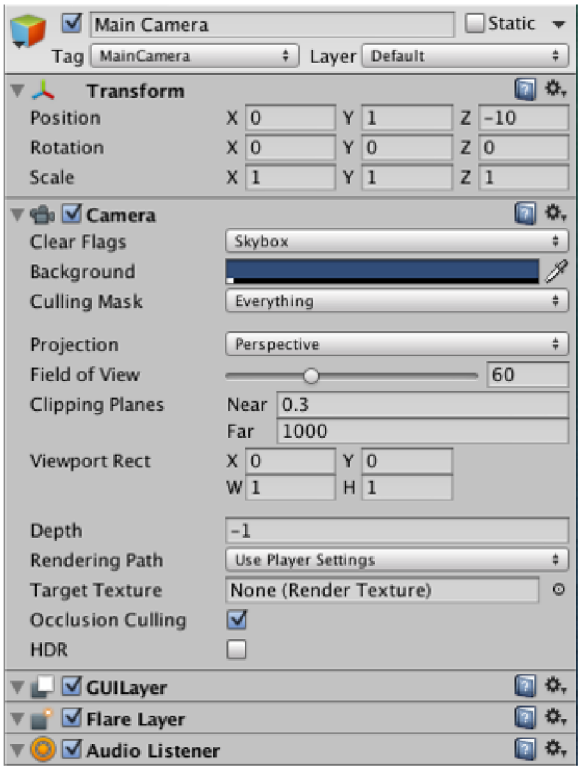
2.3.1 Unity 3D

Jak bylo nastíněno v úvodu, herní engine Unity patří mezi oblíbené a rozšířené enginy. Unity bylo poprvé představeno v roce 2005 na konferenci Apple Worlwide developers conference jako herní engine exkluzivně pro OS X.

Herní engine Unity umožňuje vytvářet hry pro různé platformy: smartphone (iOS, Android ...), herní konzole (například Xbox, Playstation ...) nebo pro počítač (maxOS, Windows, Linux). Je možné v Unity vytvářet i hry pro virtuální realitu, a to pro různé typy, například Oculus Rift, HTC Vive nebo PlaystationVR.

Komponenty

Komponenty (obrázek 2.8) jsou nejdůležitějšími prvky objektů v herním engine Unity. Slouží k přidání vlastností, vzhledu a funkcionalit daným objektům ve scéně. Jednotlivé komponenty reprezentují třídy. Tyto komponenty mohou být buď již součástí engine Unity, nebo mohou být implementovány komponenty vlastní.

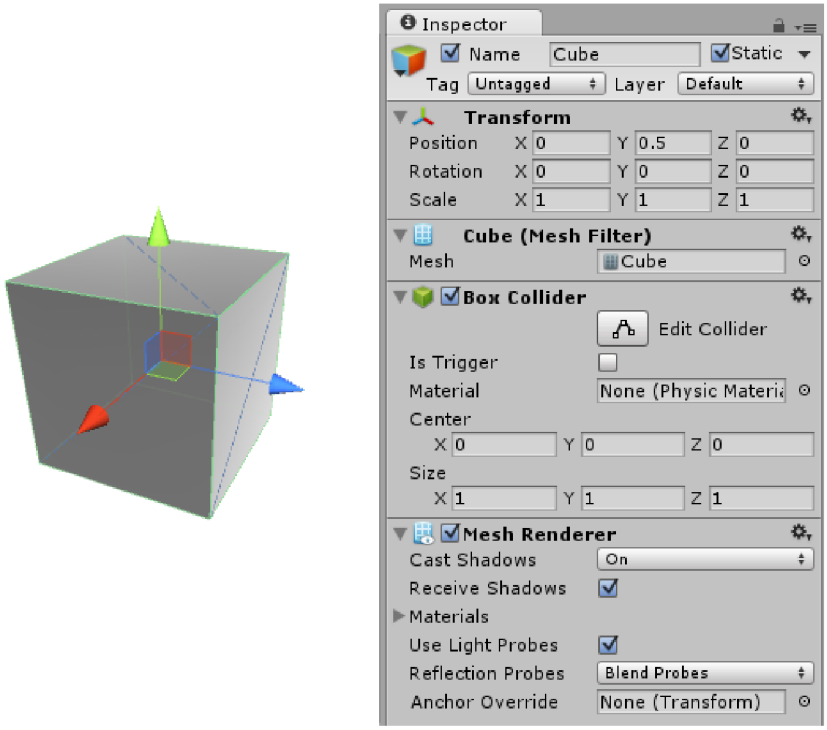


Obrázek 2.8: Na obrázku je objekt s různými komponentami a zobrazení nastavení těchto komponent. Jako první komponenta je Transform, která určuje pozici objektu v herním světě, jeho rotaci a měřítko/velikost. Dále je zde vidět komponenta Camera. Tato komponenta slouží k vytvoření herní kamery, která zobrazuje herní svět hráči na display. V této komponentě se vyskytuje mimo jiné i nastavení Field of View (2.3.5). Pod komponentou Camera jsou další komponenty. Poslední je důležitá, protože se stará o přehrávání zvuků hráči.

GameObject

GameObject [8] je základní objekt v engine Unity. Jedná o bázovou třídu, která obsahuje základní metody. Samotný GameObject mnoho nezmůže, ale v Unity slouží jako jakýsi kontejner pro další komponenty (třídy) – obrázek 2.9. Vždy obsahuje komponentu Transform, která slouží k určení umístění objektu ve scéné, jeho rotaci a měřítko. Tuto komponentu nelze z objektu odebrat.

GameObject lze změnit jméno, přiřadit Tag (označení) nebo Layer (vrstva), ve které se nachází. Tyto informace o objektu se dají využít při hledání objektů, jejich vzájemných interakcí a kolizí.



Obrázek 2.9: GameObject reprezentující kostku. V tomto objektu se nachází komponenty Transform (pro pozici, rotaci a velikost objektu), Mesh Filter, Box Collider a Mesh Renderer. Mesh Filter a Mesh Renderer slouží k vykreslování objektů. Mesh Filter určuje tvar objektu a Mesh Renderer určuje, jak bude povrch objektu vypadat. Box Collider slouží k přidání collideru (2.3.4) ve tvaru kostky.

Scéna

Scény [8] obsahují prostředí a nastavení pro hru. Každá scéna má vlastní soubor a reprezentuje vlastní úroveň (level) hry. Do scény lze umisťovat různé objekty, osvětlení, terén atd. Slouží tedy k nadesignování hry. Celá hra může být v jedné scéně nebo se hra může rozdělit do více scén. Oba postupy mají svá pozitiva a negativa.

RigidBody

Komponenta RigidBody [8] slouží ke schopnosti GameObjektu chovat se podle fyzických zákonů. Díky RigidBody může reagovat na vnější síly a podléhat tak třeba gravitaci.

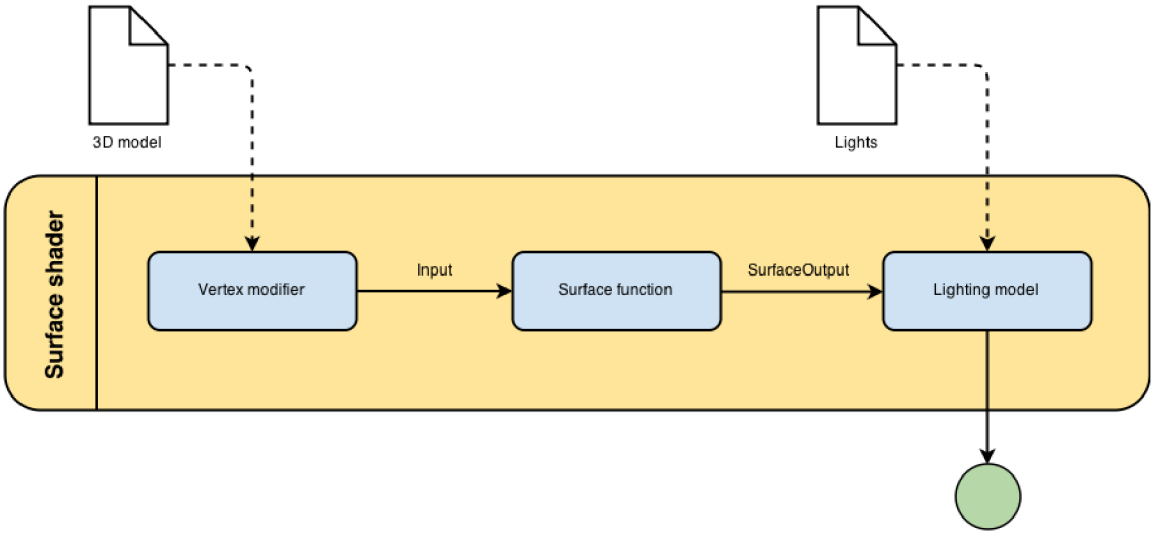
Dále RigidBody slouží ke schopnosti objektu spouštět reakci při kolizi. Například když nějaký objekt s komponentou RigidBody spadne do určité oblasti, kde je collider (více kapitola 2.3.4), může se spustit nějaká událost, přepnout scéna... Pro tento efekt je potřeba, aby objekt s RigidBody měl i collider.

Při nastavení RigidBody jako Kinematic objekt nereaguje na vnější síly, fyzikální kolize ani jiné fyzikální jevy.

2.3.2 Surface shader

Shader je speciální program, který se vykonává na grafické kartě. Ve zkratce je shader kód, který vykresluje objekty. Příkladem shaderu je Surface shader [10].

Surface shader slouží k definování, jak bude daný materiál objektu vykreslen. Na obrázku 2.10 je nakresleno zjednodušené schéma surface shaderu.



Obrázek 2.10: Zjednodušené schéma surface shaderu. Na začátku je předán 3D model do funkce, která může upravit geometrii modelu pomocí vertexů (body modelu). Tento upravený model je dále předán funkci, která definuje „vzhled“ objektu (tato funkce je základním kamenem surface shaderu). Její výstup je použit v osvětlovacím modelu, který určuje, jak bude povrch objektu ovlivněn okolními zdroji světla. Výsledkem jsou informace, jakou barvu má každý jednotlivý pixel. Převzato z internetu[14].

Pomocí surface shaderu lze též aplikovat texturu na daný model.

2.3.3 Herní mechaniky

Herní mechaniky mohou být definovány jako metody, které jsou volány agenty (objekty ve hře) pro interakci s herním světem [5]. Hry jsou obvykle založeny na principu objektově orientovaného programování, který dovoluje mít různé objekty s atributy a metodami. Metody, které reagují na stav hry (například hráčovy životy), jsou obvykle vyhodnocovány ve smyčce (loop). U příkladu se životy hráče může být například zavolána metoda, která ukončí hru nebo hru vrátí na nějaký checkpoint (záchytný bod), pokud hráči dojdou životy.

[14] https://www.alanzucconi.com/2015/06/17/surface-shaders-in-unity3d/

Pro hráče to může být například reakce na stisknutí nějaké klávesy nebo pohyb myší. Tyto mechaniky jsou často navrženy k tomu, aby jejich pomocí hráč například překonal nějakou překážku, pokročil ve hře, nebo nějakým jiným způsobem změnil stav hry.

Herní mechaniky můžeme rozdělit do dvou kategorií a to:

- Primární - ty, které slouží pro ovládání, interakci s předměty atd.
- Sekundární - ty, které reagují na stav hry

2.3.4 Collider

Collider [8] objektu určuje jeho tvar pro účely fyzických kolizí. Je neviditelný, proto tento collider nemusí být stejného tvaru jako model objektu (například model auta v collideru ve tvaru kvádru). Někdy je dokonce vhodné, aby měl jiný tvar, než je tvar objektu.

Tvar collideru může být jednoduchého geometrického útvaru například koule, kvádr nebo krychle. Nebo může mít tvar daného modelu, a to buď kopíruje tvar přesně, nebo přibližně.

Kolize je situace, kdy se setkají dva nebo více colliderů. Když k této kolizi dojde, je možné určit co se má stát (například hráč narazí do nějakého objektu a ten se rozbije).

2.3.5 Field of view

Field of view (zorné pole) ve hrách určuje rozsah viditelného světa na display v daném okamžiku. Obvykle se měří v úhlech.

Hry na počítači mají většinou větší Field of View než konzole, z toho důvodu, že silnější počítače dokáží vykreslovat více objektů než konzole. A také protože u počítače monitoru sedí člověk blíže, než jak je u konzolí obvyklé, u televize, na kterou je připojena konzole. U některých her právě kvůli náročnosti na vykreslování lze v nastavení Field of View měnit, a tím tak snížit nároky na vykreslování. Nebo naopak si Field of View může zvýšit a mít větší přehled, co se ve hře děje.

Kapitola 3

Návrh

V této kapitole bude rozebrán návrh terénu, využití analýzy hudby a samotné hry.

Cílem bylo vytvořit hru, která využije informace z hudební analýzy. Rozhodl jsem se vytvořit hru, ve které využiji tyto informace k tomu, abych vytvořil trať podle hudby. Na podobném principu funguje hra Audiosurf[1]. Já jsem ale chtěl vytvořit hru, ve které bude auto a pojede přes horskou krajinu.

3.1 Terén

Pro tvorbu samotného terénu jsem se rozhodl využít procedurálního generování. Bylo to z toho důvodu, že jsem chtěl docílit toho, že každým spuštěním bude hra o něco jiná. Ke tvorbě horského terénu dobře poslouží Perlinův šum.

Jelikož jsem vytvářel horský terén, nastane situace, kdy bude hráč na vyšší úrovni povrchu, než je okolí hory a uvidí odtud, jak v dáli terén končí v prázdnu. Z tohoto důvodu jsem mapu navrhoval větší, než by bylo potřeba pro samotnou trať.

Pro různé rozsahy výšky terénu jsem volil jinou barvu a texturu povrchu.

3.2 Analýza hudby

Hudbu lze zpracovávat dvěma způsoby. Prvním způsobem se hudba analyzuje při běhu hry nebo při přehrávání hudby. Druhým způsobem je analýza hudby, která proběhne dopředu, ještě před samotným startem hry nebo přehrávání hudby.

Pro můj způsob využití je první varianta nevhodná. Analýzu hudby využívám ke generování trati a pro vizualizaci hudby na objektech ve hře.

K využití hudby pro generování trati mi stačí amplituda signálu. Pro vyzualizaci hudby na objektech jsem se rozhodl využít detekci beatů. Při beatu objekt změní svou velikost.

3.2.1 Získání dat

Jako první věc je potřeba získat data hudby. Kvůli rozmanitosti hudby a chuti lidí jsem se rozhodl, že si uživatel může svou skladbu vybrat. Pro vybírání skladby jsem navrhl řešení, že hráčovi se na začátku hry zobrazí dialogové okno, kde si bude moct vybrat vlastní skladbu uloženou na počítači, disku atd.

[1] https://store.steampowered.com/app/12900/AudioSurf/

Jako formát souboru jsem vybral MP3, z důvodů implentačních (viz dále kapitola Implementace, podkapitola 4.3.1). MP3 formát jsem zvolil také z toho důvodu, že pokud má člověk v dnešní době stále nějaké uložené skladby, bude to pravděpodobně MP3.

3.2.2 Detekce beatů

Tento postup je inspirován z [1] a [9].

Po nahrátí dat skladby se v iteracích projde vzorky po kusech, které mají velikost mocniny dvou (kvůli efektivitě). Velikost 1024 vzorků by měla být optimální.

Na tento kus vzorků se aplikuje škálovací Hanningovo okno Rychlé Fourierovy transformace (FFT).

Poté se aplikuje Rychlá Fourierova transformace k získání hodnot spektra v komplexních číslech. Velikost bude polovina kusu vzorků. Výsledek FFT je tedy rozdělen při velikosti 1024 vzorků na 512 binů. Každý z těchto binů obsahuje určitý rozsah frekvence, který reprezentuje. Při 48000 Hz / 2 / 512 je rozsah 46.875 Hz.

Poté je potřeba tato komplexní čísla převést na normální čísla.

Na tento výsledek se použije škálovací faktor okna FFT a vypočítá se aktuální čas hudby reprezentován tímto kusem vzorků.

Dále se pro tento kus vzorků vypočítá spectral flux. Tedy z každého binu FFT současného kusu vzorků se odečte ekvivalentní bin z předchozího kusu vzorků. Rozdíly všech těchto binů (512 binů) se sečtou do sumy a výsledek této sumy je spektral flux. Tento spektral flux se pak vypočítá pro všechny kusy vzorků.

Všechny spectral fluxy dohromady pak reprezentují funkci. Pokud má aktuální kus vzorků celkovou energii větší než předchozí kus vzorků, spectral flux funkce bude v tomto rozmezí rostoucí a pokud bude celková energie menší, funkce bude v tomto rozmezí klesající.

Před detekováním vrcholů (které reprezentují beaty) je ještě potřeba se zbavit negativních hodnot spectral fluxů. Pro detekci beatů nejsou důležité. Tomuto procesu se říká rektifikace.

Pro zbavení se šumu („znečištění“ v podobě malých vrcholů) lze využít vyhlazení funkce, nebo si vytvořit threshold funkci (určuje zda-li hodnota překročila určitou mez). V tomto případě bohatě vystačí treshold funkce.

Mez, podle které tato funkce rozhoduje, se získá průměrem hodnot v okně několika hodnot spectral flux funkce. Tato mez se dále vynásobí násobitelem, který je větší než jedna (například 1.5).

Treshnold funkce se využije k vytvoření tzv. Pruned Spectral Flux funkcí. Pokud je hodnota flux menší než mez treshold funkce, hodnota pruned spectral flux je 0. Pokud je větší než mez, pak je pruned spectral flux roven spectral flux.

Nakonec se pak provede k získání beatů porovnávání bezprostředních sousedů vzorků této pruned spectral flux funkce. Pokud je vzorek větší než předchozí vzorek a zároveň větší než následující vzorek, pak je tento vzorek vrcholem a hledaným beatem.

3.3 Trať

První návrh na tvorbu tratě byl vytvořit nejdříve trať, potom vytvářet terén okolo tratě. Nakonec jsem se rozhodl nejdříve vygenerovat terén a cestu najít v něm.

K vytvoření tratě pro auto přes horský terén využívám informace z analýzy hudby. Stál jsem před otázkou, jak hudbu při tvorbě trati využít. Mohl jsem použít princip, jaký je využit ve hře Audiosurf. V ní je, jednoduše popsáno, trať podle hudby tvořena tak, že trať

klesá nebo stoupá mezi hlavními body písně. Tyto body jsou například výraznější změna síly signálu. Právě podle síly signálu je určeno, jestli trať klesá nebo stoupá. V případě slabšího signálu trať stoupá a raketka letí pomaleji a při silnějším trať klesá a raketka letí rychleji.

Ačkoliv je má hra Audiosurfem inspirována, nechtěl jsem vytvářet její kopii. Pro mé řešení jsem zvolil využití takové, že trať hledá ve vygenerovaném terénu přibližně body, které kopírují křivku amplitudy signálu. Těmito body je pak provedena Bézierova křivka, ze které se vytvoří model silnice.

3.4 Herní mechaniky

Jak bylo uvedeno v kapitole 3.3, nechtěl jsem vytvářet kopii hry Audiosurf. Nakonec jsem navrhl herní mechaniku, kterou jsem v žádně hře ještě neviděl. Netvrdím, že neexistuje, jen jsem se s ní nesetkal.

3.4.1 Start hry

Po spuštění aplikace se zobrazí jednoduché menu s položkami Play (hrát) a Quit (ukončit). Po kliknutí na položku Play se hra spustí. Při načítání se zobrazí dialogové okno pro volbu skladby. Zde si může hráč vybrat vlastní skladbu nebo zavřít dialogové okno. Pokud dialogové okno zavře, pro hru se vybere výchozí skladba. Po inicializaci a vygenerování objektů je hráč na výchozí pozici.

3.4.2 Cíl hry

Cílem hry je dorazit do cíle dříve, než do cíle dorazí čára znázorňující skladbu. Tato čára jede konstantní rychlostí a celou trať ujede za dobu trvání zvolené skladby.

Čára znázorňující píseň má na začátku malý náskok před hráčem. Aby hráč tuto čáru předehnal, dorazil do cíle první, a tím vyhrál, musí po cestě sbírat modré koule. Ty hráče zrychlí a naopak se musí vyhýbat koulím červené barvy, které po sebrání hráče prudce zpomalí.

Osobně nejsem žádný fanoušek skóre ve hře, proto v řešení není žádné počítadlo skóre za sebrané koule, ať už plusové body za modré objekty, či mínusové za červené objekty.

3.4.3 Konec hry

Jak bylo zmíněno v 3.4.2, hra končí buď výhrou, nebo prohrou. Pokud hráč dojede do cíle jako první, vyhrává a zobrazí se menu s nápisem, že vyhrál, s tlačítkem pro restart (spustí hru znovu, vygeneruje se nový terén, lze zvolit stejná, ale i jiná skladba) a s tlačítkem pro konec aplikace. V případě, že do cíle dorazí první čára reprezentující skladbu, zobrazí se stejné menu jen s tím rozdílem, že hra oznámí prohru. Tlačítka a jejich funkce jsou stejné.

3.4.4 Hráč

Pro ovládání hráče, respektive auta na trati, se nabízely v podstatě dvě možnosti. Tou první bylo dát hráči plnou kontrolu nad autem, kdy by mohl ovládat plyn, brzdu a další. V tomto případě by nebylo možné rozumným způsobem zajistit pohyb hráče v souladu s hudbou.

Druhou variantou bylo dát hráči možnost ovládat auto pouze do stran. Pohyb vpřed by pak zajišťovalo konstantní následování trati v čase.

Pro své řešení jsem vybral druhou možnost, z důvodu větší kontroly plynutí hry a také tento způsob více vyhovoval mým návrhům herních mechanik.

3.4.5 Objekty na silnici

K docílení zmíněné mechaniky cíle hry 3.4.2, jsem se rozhodl generovat objekty na trati. V prvotním návrhu jsem místo barevných koulí zamýšlel použít model kanistru (místo modré koule) a kámen (místo červené koule). Tento návrh jsem se rozhodl nepoužít, protože jsem chtěl vizualizovat hudbu na těchto objektech.

Pro vizualizaci hudby na těchto objektech jsem využil změny velikosti (scale) objektu při beatu podle síly signálu v daném okamžiku. Mezi těmito beaty se velikost objektu postupně zmenšuje na původní velikost.

Objekty jsou generovány náhodně po trati a je náhodně volen i druh objektu.

Objekty jsou tedy dvojího druhu:

- Modré koule - hráče mírně zrychlí na 2 sekundy
- Červené koule - hráče prudce zpomalí na 1 sekundu

Kapitola 4

Implementace

V předcházející kapitole byl probrán návrh řešení a v této kapitole bude řeč o implementaci tohoto řešení.

Již z názvu je zřejmé, že pro tvorbu hry je použit herní engine Unity. V Unity je možné psát sktripty v jazyce C# nebo JavaScript. Pro své řešení jsem si vybral jazyk C#.

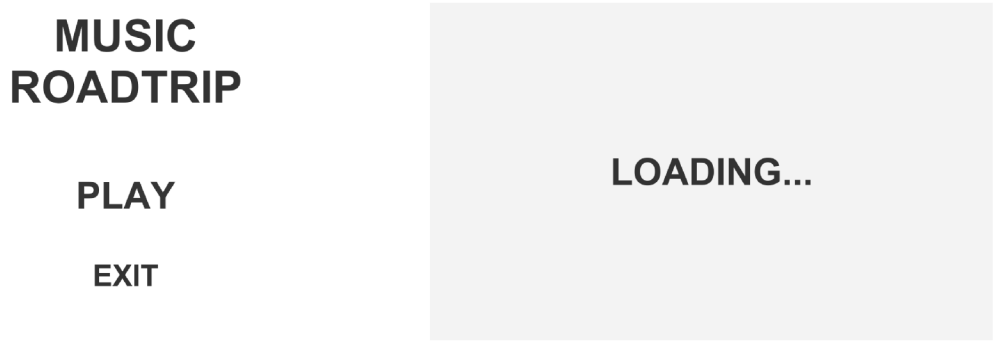
Hru jsem implementoval v Unity verze 2017. Unity je sice už vyšší verze, ale kvůli obavám z kompatibility jsem hru dokončil ve stejné verzi, v jaké jsem začal.

Pro editaci skriptů, některých konfiguračních souborů a sestavení řešení jsem využil Visual Studio 2017.

4.1 Menu

Grafické rozhraní menu jsem vytvořil pomocí UI elementů. Konkrétně to byl panel, text a tlačítka.

Hlavní nabídku (Main menu) tvoří panel. Tento panel je vyplněn nadpisem, který je vytvořen prostým Text elementem. Dále je zde tlačítko Exit (Ukončit), které zavolá metodu pro ukončení aplikace. Tlačítko Play (Hrát) volá metodu, která přepne scénu na následující scénu, kterou je scéna Načítání (Loading).



Obrázek 4.1: Vlevo je ukázka scény Hlavní nabídky, která je načtena jako první po startu aplikace a která slouží k zapnutí hry nebo ukončení aplikace. Vpravo je ukázka scény Načítání. Scéna Načítání je načtena po kliknutí na tlačítko Play ve scéně Hlavní nabídky.

Na scéně Načítání je opět panel a v něm je text „Loading...“. Kromě panelu s UI je na scéně objekt, který automaticky po startu scény scénu přepíná na scénu následující, kterou je samotná hra. Scéna Načítání je zde proto, aby hra hráče ujistila, že se zareagovalo na

jeho stisknutí tlačítka. V Unity je vidět scéna, ze které se přechází než je načtena scéna, na kterou se přechází.

Po scéně samotné hry jsou zde ještě dvě scény, na kterých je UI. Je to scéna Výhry a scéna Prohry. Obě scény jsou tvořeny opět panelem, textem a dvěma tlačítky. Na scéně je informace v Text elementu o tom, že hráč vyhrál a naopak u scény Prohry je v tomto Text elementu informace o prohře. Dále jsou na obou scénách 2 tlačítka. První tlačítko je tlačítko Restart, které volá metodu na přepnutí scény na scénu Načítání. Druhé tlačítko je Exit, které volá metodu na ukončení aplikace.

Obrázek 4.2: Vlevo je ukázka scény Výhry. Tato scéna je načtena, pokud hráč dorazí do cíle jako první. V opačném případě je načtena scéna Prohry, která je vidět na obrázku vpravo.

4.2 Terén

Jak již bylo zmíněno, terén je generován pomocí procedurálního generování, konkrétně Perlinovým šumem.

Pro generování výškové mapy pomocí Perlinova šumu slouží třída `Noise` a její metoda `GenerateNoiseMap()`.

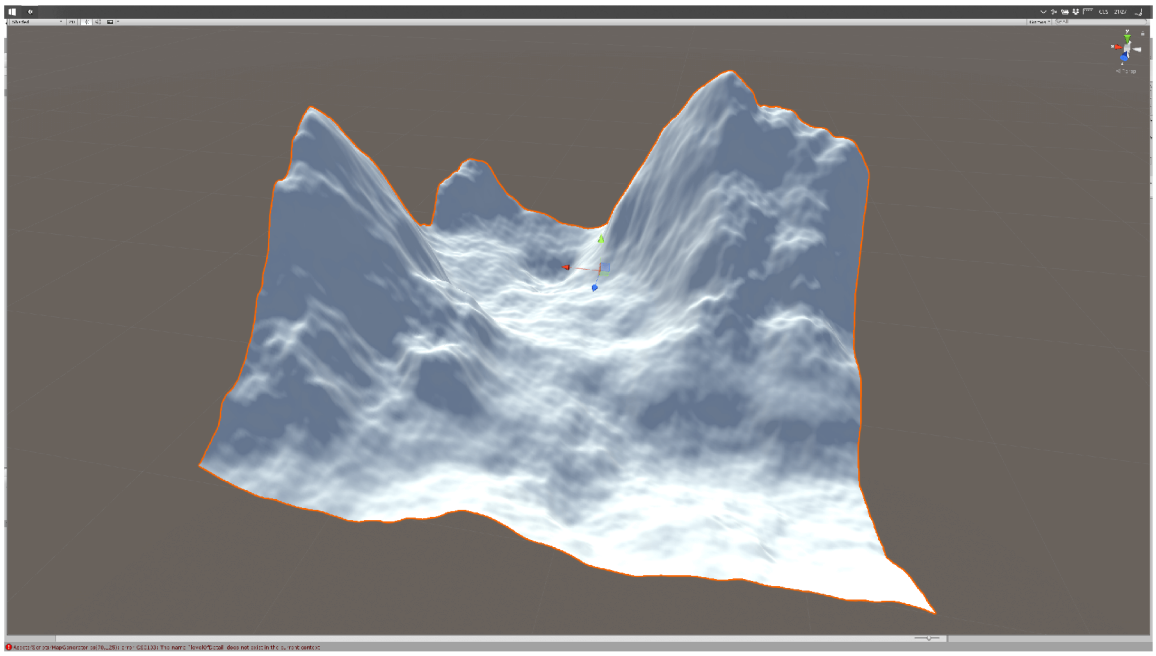
Třída `MapGenerator` má vytvořený svůj objekt ve scéně pro jednoduché a rychlé změny proměnných. Tato třída se stará o zavolání vykreslení terénu. Toto vykreslení obstarávává komponenta (třída) `MapDisplay`, která je přidána stejnému objektu jako je `MapGenerator`.

Povrchu je potřeba dát nějaký vzhled, jinak by terén byl celý bílý (obrázek 4.3). Prvotní zkouška byla použití obarvení trojúhelníků (triangles) podle toho, jakou měly výšku. Výsledek nebyl vizuálně přívětivý (obrázek 4.4). Mezi barvami byl velmi hrubý přechod a samotné barvy vypadaly uměle.

Nakonec jsem využil tvorby vlastního surface shaderu. Pomocí jej jsem mohl na terén nanést textury, jejich podkladové barvy a navíc mezi jednotlivými vrstvami textur vytvořit plynulý přechod (obrázek 4.5).

Terén je potřeba generovat po kusech (chunks), protože engine Unity dovoluje mít na jednom kusu meshe (viditelný model objektu) 65535 vertexů (vrcholů). Je to z toho důvodu, že mesh index buffer je 16 bitový. V nové verzi Unity údajně lze tento buffer změnit na 32 bitový.

Tyto kusy terénu je na sebe potřeba plynule napojit. Zde se dostávám k tomu, proč je dobrá vlastnost, že se při použití stejných vstupních dat vygeneruje stejná sekvence pseudonáhodných čísel 2.1.2. Aby na sebe tyto kusy terénu navazovaly, je zapotřebí mít vygenerovaný stejný Perlinův šum s tím rozdílem, že se nastaví odsazení (offset). S tímto



Obrázek 4.3: Na obrázku je ukázka vygenerovaného jednoho kusu terénu bez textur, pouze s nastaveným základním materiálem.

odsazením, pokud bude rovno šířce kusu mapy, lze generovat pokračující Perlinův šum, který bude navazovat.

Tímto principem generuji matici kusů terénu o rozměrech 3x3 – obrázek 4.5.

4.3 Hudba a její zpracování

Jak bylo uvedeno v kapitole 3.2, hudba se zpracovává dopředu. Veškerou logiku potřebnou pro hudbu obsahuje objekt SongController. Tento objekt obsahuje komponenty `AudioSource`, `SongController` a `SpectralFluxAnalyzer`. Komponenta `AudioSource` je součástí Unity a slouží pro samotné přehrávání hudby a obsahuje důležité informace o skladbě.

`SongController` slouží k získání skladby, spektrální analýze hudby, pro ovládání objektu reprezentujícího hudbu a reakce kamery na hudbu.

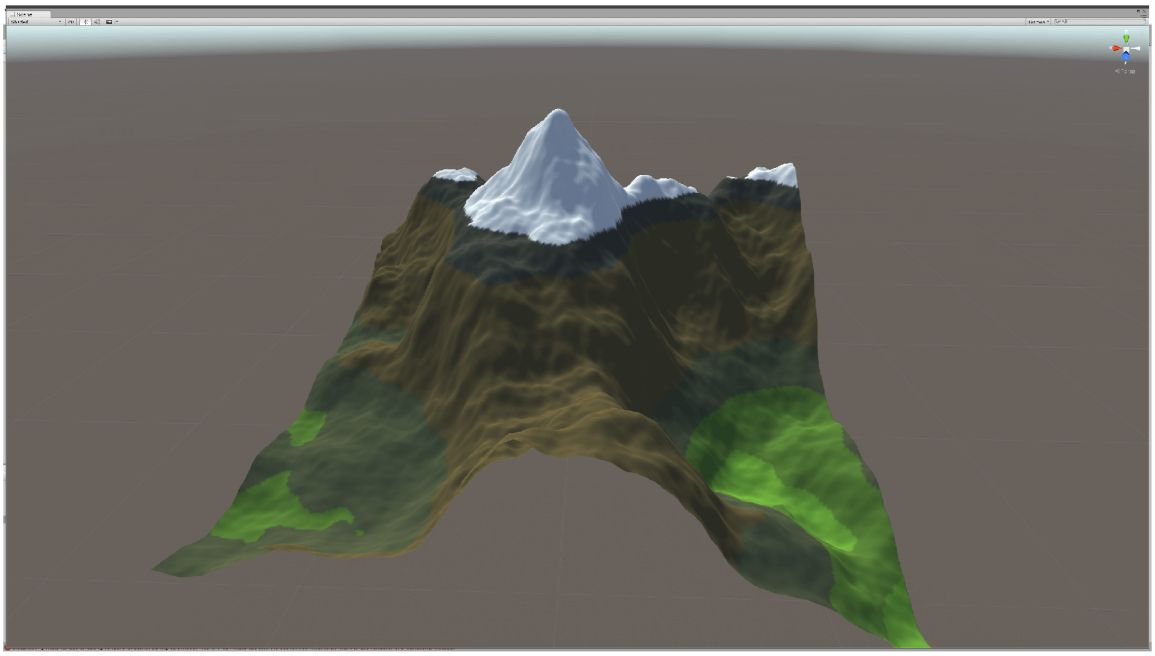
`SpectralFluxAnalyzer` je pomocná komponenta k analýze hudby.

4.3.1 Získání skladby

Po vygenerování základního terénu je potřeba získat skladbu k analýze.

Pro výběr vlastní skladby jsem se rozhodl použít dialogové okno. Zde jsem narazil na skutečnost, pro kterou jsem nenašel odpověď. Unity obsahuje dialogové okno pro výběr souboru, ale funguje pouze při spuštění hry v editoru Unity. V sestavené hře už toto dialogové okno nefunguje a navíc při sestavování způsobuje chybu.

Toto řešení je tedy k ničemu a je potřeba si vytvořit vlastní dialogové okno nebo využít asset (nástroj v Unity) třetí strany. Rozhodl jsem se použít asset File Browser, který je volně dostupný na Asset store přímo v Unity. Z tohoto assetu jsem využil funkci `OpenSingleFile()`, která vrací cestu k vybranému souboru jako `string`.



Obrázek 4.4: Na obrázku je ukázka vygenerovaného kusu terénu s obarvenými trojúhelníky meshe podle výšky.

Před tuto cestu je dále potřeba ještě přidat řetězec `file:///`. Z takto vytvořené celé cesty se vytvoří `WWW`.

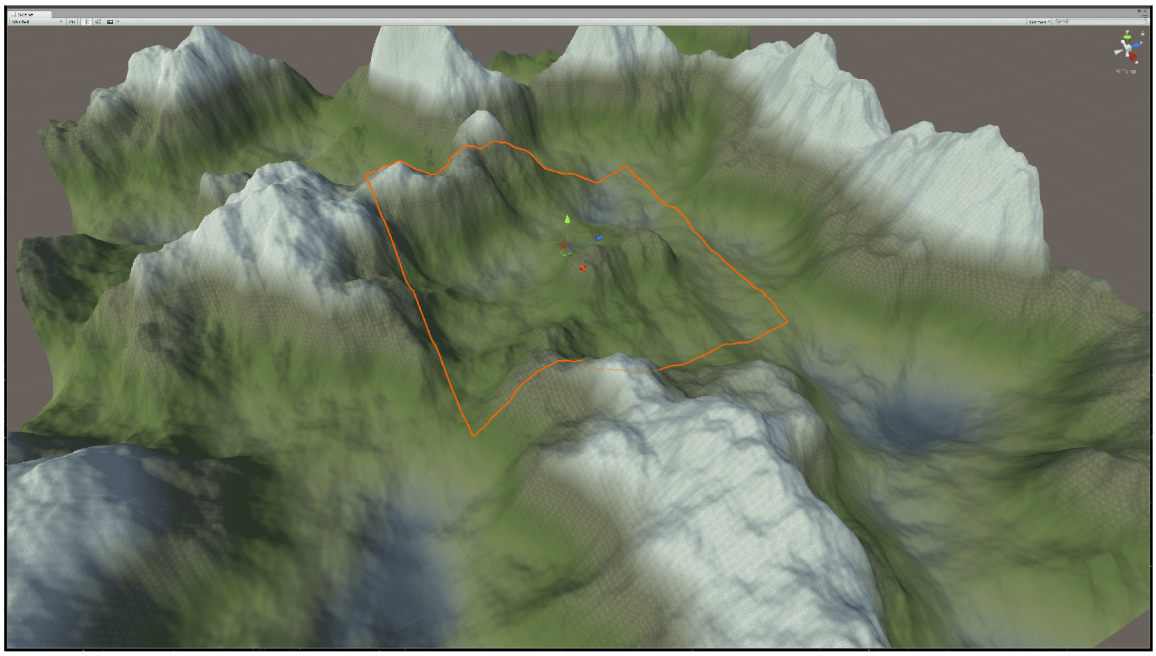
Zde se vyskytl druhý problém s načítáním. Unity z licenčních důvodů umí zpracovávat pouze formáty ogg a wav. Opět je zde tedy potřeba sáhnout po řešení třetí strany. Zde jsem využil knihovnu NAudio, ze které jsem využil funkci `FromMp3Data()`. Tato funkce ze zadané cesty k souboru ve tvaru WWW získá potřebná data. Tato knihovna způsobila menší problém, vyžaduje totiž .NET verze 4.0 a ve verzi Unity 2017 je stabilní verze 2.0. Bylo zapotřebí použít experimentální verzi 4.6, která je ve verzi 2017. Toho se docílí přepnutím v nastavení sestavení, přepsáním konfiguračního souboru na sestavení a přenastavením projektu ve Visual Studiu.

4.3.2 Zpracování hudby

Získaná syrová data jsou potřeba upravit, protože jsou ve stereu. Pro zpracování je nutné stereo zvuk převést na mono. Převedení se docílí zprůměrováním hodnot po dvou v cyklu. Tato operace je vcelku náročná. Pro představu: 5 minut dlouhá skladba (300 sekund) se vzorkováním 48 000 vzorků za vteřinu je celkem 14 400 000 vzorků v mono. Ve stereu je to pak 28 800 000 vzorků. Tolik vzorků převést na mono chvíli zabere.

Po převedení skladby na mono je možné začít skladbu analyzovat. V kapitole 3.2.2 je popsán algoritmus k detekci beatů. Zde budou popsány spíše konkrétní detaily a nástroje.

Velikost chunků (kusů vzorků) je tedy 1024 vzorků. Tato hodnota je vyhovující a docílí se s ní uspokojivých výsledků. Pro implementaci FFT má Unity nástroje pouze pro zpracování v běhu aplikace. Zde jsem se rozhodl použít knihovny DSPLib. Tato knihovna je vcelku nenáročná a má uspokojující výsledky. Jako škálovací okno jsem použil Hanningovo okno, které je implementováno i v této knihovně.



Obrázek 4.5: Na obrázku je ukázka generovaného terénu sestaveného z více kusů a s texturami. Uprostřed je vidět označený jeden kus terénu.

Unity poskytuje Mono .NET, ve kterém není obsažena System.Numerics knihovna, ve které je datový typ Complex, který je potřeba pro DSPLib. Proto je potřeba přidat Complex.cs (získán na githubu Microsoftu).

Výsledkem FFT je pole floatů o velikosti 512. Každý prvek v poli reprezentuje rozsah 48000/2/512 = 46.875 Hz. Nultý prvek pole reprezentuje rozsah 0-46.875 Hz, další prvek 46.875-93.75 Hz a tak dále.

4.4 Trať

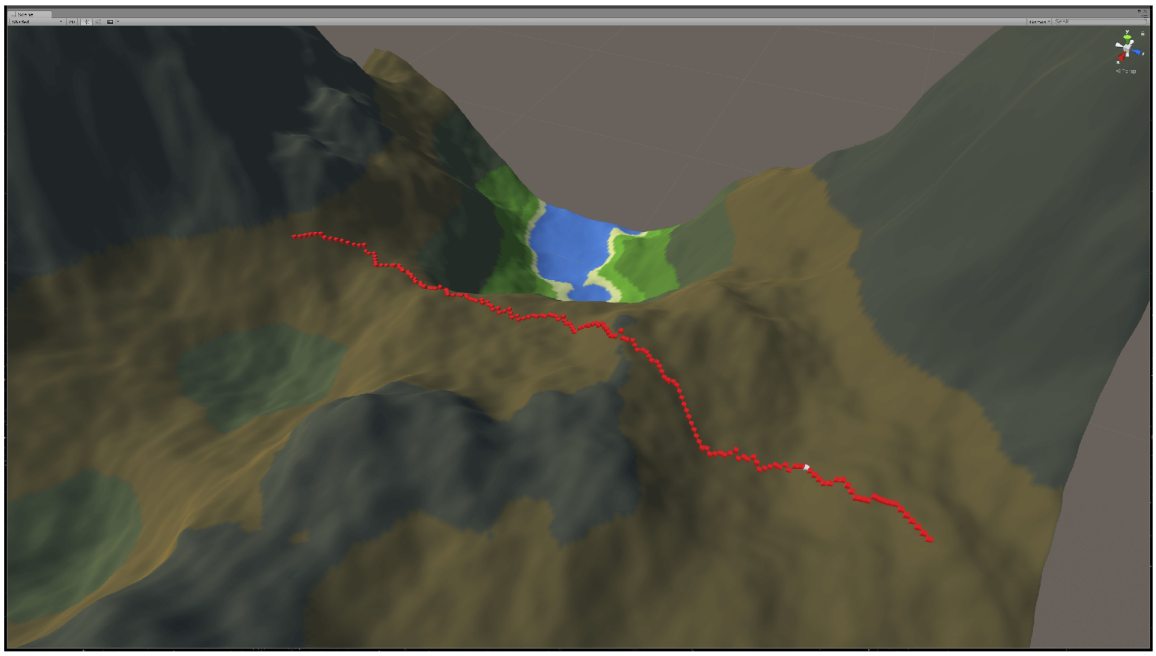
Trať je generována podle skladby, kterou zpracovává `SongController`.

Pro hledání dalšího bodu trati jsem chtěl využít hledání v osmiokolí. Problém nastal v tom, že v písních se určitý rytmus většinou alespoň chvíli opakuje. Když jsem hledal v celém osmiokolí, trať se vracela tam a zpět, jak se rytmus opakoval.

Nakonec jsem omezil hledání v osmiokolí jen na 3 body, aby se trať nevracela a pokračovala dále. Rozhodně by se dalo najít lepší řešení, ale z důvodu času jsem přistoupil na tuto metodu. Výsledná cesta vizualizována body – obrázek 4.6.

V konečném výsledku je téměř i tak nemožné v procedurálně generovaném terénu najít přesnou křivku skladby, protože terén podléhá určité náhodnosti a není generován na míru zvolené skladbě.

Těmito body je pak vedena Bézierova křivka. Z této křivky se vytvoří mesh a přiřadí se mu textura silnice a materiály včetně materiálu pro spodní stranu silnice. Výsledná silnice – obrázek 4.7.



Obrázek 4.6: Na obrázku jsou vidět malé červené kostky reprezentující nalezené body trati v terénu. Napravo je mezi červenými kostkami jedna bílá, která reprezentuje hráče. Tato bílá kostka se posouvá v čase po trase znázorněné červenými kostkami.

4.5 Hráč a jeho pohyb

Pro model hráče jsem použil model vozu Mercedes-Benz 190SL[1]. Pro tento model jsem si vytvořil vlastní materiály.

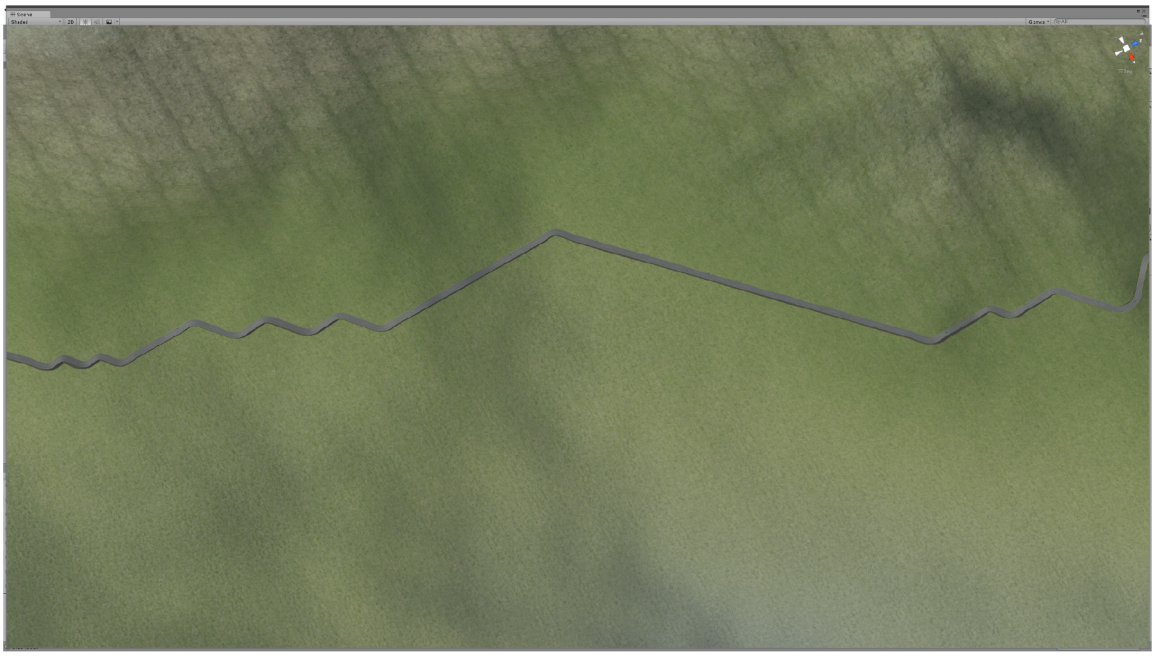
Jak bylo zmíněno v 3.4.4, rozhodl jsem se pro variantu, že hráč bude moci ovládat auto pouze do stran a auto bude v čase následovat trať. Pro následování trati a zároveň možnosti zatáčet (z pruhu do pruhu) bylo potřeba hráče zanořit do objektu, který má stejnou šířku, jako silnice a následování trati udělat pro tento zanořující objekt.

Pomocí tohoto objektu lze taktéž zajistit to, aby hráč nevyjížděl ze silnice. Kolem tohoto objektu jsou vytvořeny mantinely v podobě colliderů (Collider Box komponenta - dostačuje tvarem kvádru) a hráči je přiřazen taktéž collider a navíc RigidBody komponenta. U komponenty byly vypnuty rotace v osách kvůli nežádoucím rotacím.

Pohyb hráče je ovládán pomocí šipek na klávesnici. Pro pohyb auta jsem využil toho, že je stisk šipek do stran uložen v `Input.GetAxis("Horizontal")`. Tuto hodnotu použiji jako hodnotu osy x pro vytvoření `Vector3` pro reprezentaci pohybu do stran. Tento vektor se zmenší vynásobením koeficientem `0.05`. Tato hodnota je získána experimentálním způsobem a pomáhá zpomalit pohyb, aby auto přejíždělo plynulou a uvěřitelnou rychlostí.

Tento pohyb do stran je ještě potřeba upravit tak, aby se auto pohybovalo po normále silnice v daném bodu. Bez této úpravy auto pak zajíždí do silnice nebo levituje nad silnicí. Pro samotné posunutí auta používám funkci `Vector3.Lerp()`. Tato funkce má argumenty počáteční pozice, pozice na kterou se posouvá a čas, za jaký se posune. Jelikož je vhodné mít pohyb nezávisle na počtu snímků za vteřinu (FPS), používám u času hod-

[1] https://www.turbosquid.com/3d-models/free-duesen-bayern-mystar-190-3d-model/1062796



Obrázek 4.7: Ukázka vygenerované výsledné silnice v terénu.

notu `Time.deltaTime`, která zaručí, že se bude auto pohybovat stejnou rychlostí pro různě výkonné počítače.

K pohybu do stran jsem ještě přidal animaci natočení auta o pár stupňů ve směru zatáčení. Tuto animaci jsem udělal ručně pomocí odčítání a přičítání malých hodnot do určité úrovně a následné vracení do původní hodnoty.

Pohybu v čase podle trati je dosaženo počítáním aktuální pozice v čase a podle této pozice hledání souřadnic trasy. Aktuální pozice ve skladbě se získá: aktuální čas / celkový čas skladby. Hodnota aktuálního času je zvyšována v proměnné každým snímkem. Unity v každém snímku volá funkci `Update()`, pokud nějakou objekt má. Problém je, pokud je potřeba zvyšovat v této funkci hodnotu v čase, že nezaručí na všech strojích stejné zvyšování hodnoty. K tomu slouží opět `Time.deltaTime`, která se postará o zvyšování podle reálného času.

4.6 Objekt reprezentující skladbu

Objekt reprezentující skladbu je vytvořen jako `GameObject` s komponentou `Trail Renderer`. Tato komponenta vykresluje stopu za objektem. Tento objekt reprezentující skladbu je protivníkem hráče. Pohybuje se uprostřed silnice a jede konstantní rychlostí. Objekt cestu urazí za délku času skladby.

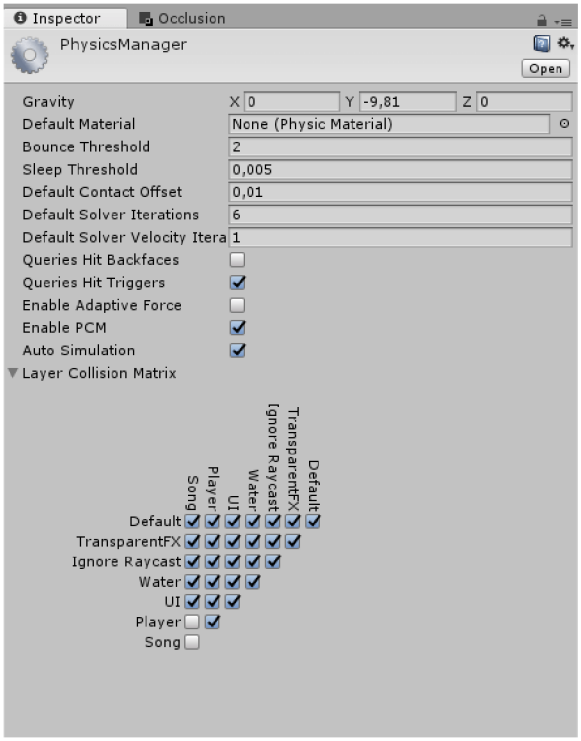
Aby bylo možné mít reakci na projetí tohoto objektu cílem, je potřeba objektu přiřadit komponentu `RigidBody` a spolu s ní collider. Vystačí základního tvaru například Box Collider. Komponentu RigidBody je vhodné nastavit jako kinematickou (volba `Is Kinematic`), kvůli nechtěným rotacím a pohybům.

Přidáním těchto komponent je však způsoben jeden problém. Pokud nastane situace, že auto a tento objekt jedou vedle sebe, auto nemůže zatočit, protože mu v tom tento objekt

brání. Řešení tohoto problému je vypnutí kolizí mezi autem a tímto objektem reprezentujícím skladbu (obrázek 4.8).

Trail renderer vyžaduje materiál, který má shader zvolený jako Particles/Alpha Blended.

Pro vizualizaci hudby je při beatu změněna šířka této stopy. Mezi těmito beaty je šířka stopy plynule zmenšována každým snímkem o malou hodnotu.



Obrázek 4.8: Nastavení fyziky ve hře. Dole na obrázku je vidět matice, ve které lze nastavit, které vrstvy hry budou mít mezi sebou kolize. V tomto případě je vypnuta kolize mezi hráčem a objektem znárožňujícím skladbu.

4.7 Objekty na trati

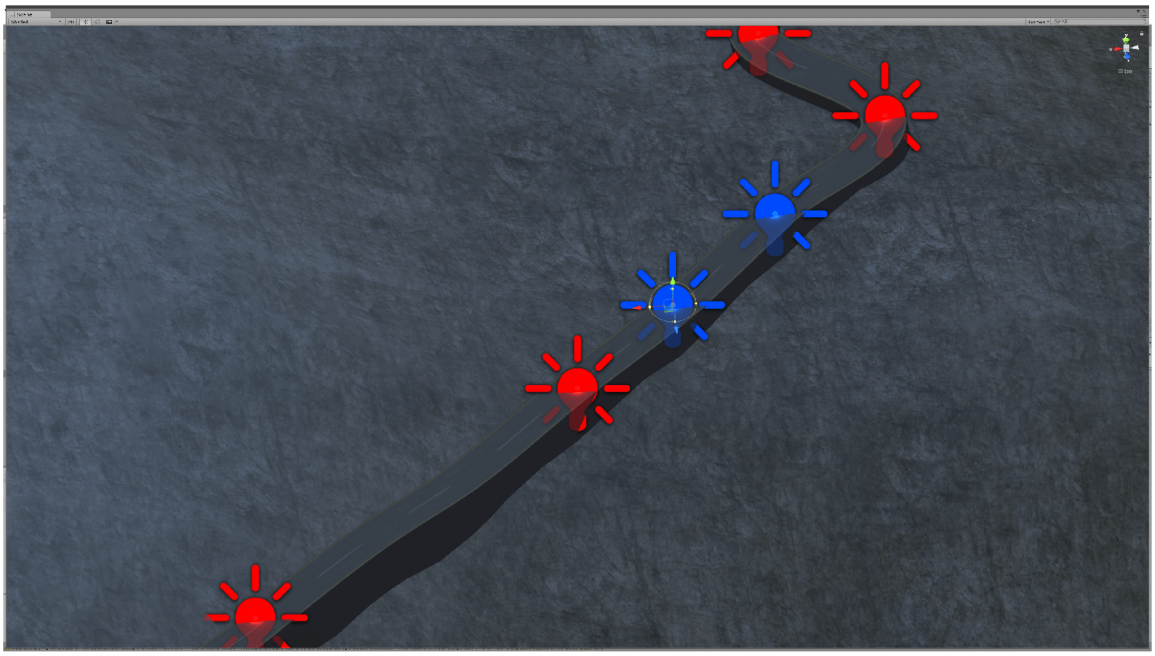
Objekty na cestě jsou generovány primitivním způsobem. Funkcí `Random.Range(0,3)` se generuje celé číslo v intervalu <0,3) a podle tohoto čísla se určí:

- 0: objekt se nevygeneruje
- 1: objekt se generuje jako překážka
- 2: objekt se generuje jako zrychlení

Body pro počáteční lokalitu vygenerovaných objektů jsou body trasy.

Dále se funkcí `Random.Range(-0.1f,0.1f)` určí posunutí do strany na silnici. Pro toto posunutí je stejně jako v případě posunu auta potřeba získat normálu v daném bodě silnice. Následuje samotné vytvoření instance (`Instantiate()`), v případě, že se generuje nějaký objekt.

Objekty vygenerované na silnici – obrázek 4.9.



Obrázek 4.9: Na obrázku jsou objekty na trati, které vizualizují hudbu a které slouží ke zrychlení a zpomalení hráče. Pohled je z editoru Unity a jsou zde vidět ikony svítící žárovky, která dává najevo, že v daném místě je zdroj světla. Červené objekty jsou zpomalující a modré zrychlující a jsou náhodně vygenerované po trati.

Funkcionalita

Vygenerované objekty hráče buď zpomalí, nebo zrychlí. Zrychlení je dosaženo tím, že pozice hráče v čase skladby roste 2 vteřiny o malou hodnotou rychleji, než běží čas. U zpomalení je to naopak, zde na 1 vteřinu roste pozice hráče v čase pomaleji, než běží čas. Míru zrychlení a zpomalení ovlivňuje délka skladby. Kdyby byla míra zrychlení a zpomalení ve všech skladbách stejná, tak by u krátkých skladeb bylo velmi těžké vyhrát a naopak u delších skladeb by bylo zase velmi jednoduché vyhrát.

Ukázka řádku kódu, který pracuje s pozicí v čase hráče. Je zde vidět také práce s `Time.deltaTime`

- `currTime -= 0.6f * (1 / clipLength) * Time.deltaTime;` // pro zpomalení
- `currTime += 0.15f * (1 / clipLength) * Time.deltaTime;` // pro zrychlení

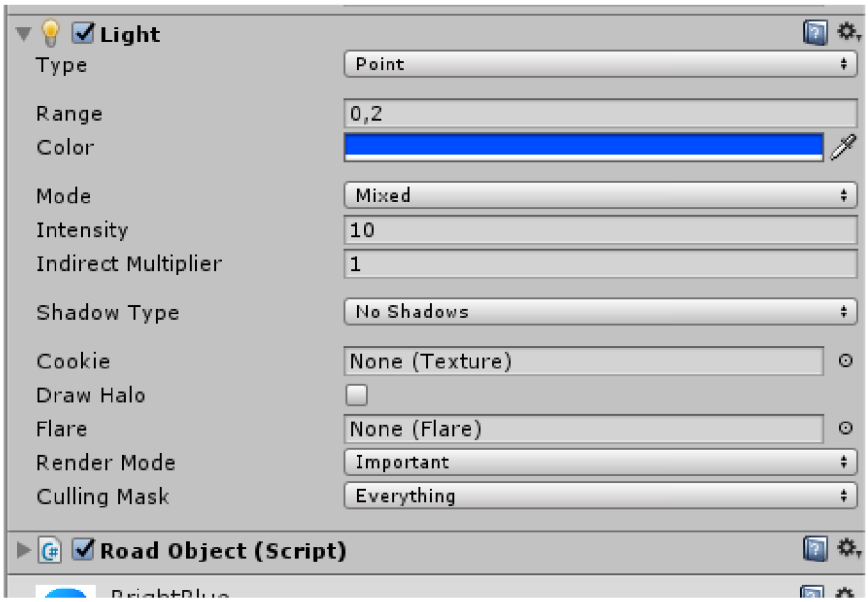
Hodnoty byly získány experimentálně.

Model objektu je základní objekt `Sphere` (Koule) a má nastavený Sphere collider. Materiály pro tyto objekty mají nastavenou luminiscenci (vyzařování světla) v požadované barvě a navíc mají přidáno do nitra lokální osvětlení (komponenta Light s nastaveným typem `Point`), aby osvětlily i své okolí. Takto nastavený objekt je uložen do složky `Resources`, kde je uložen jako prototyp pro generátor objektů na trati (`RoadObjectGenerator`).

4.8 Osvětlení

Aby vynikla vizualizace objektů na trati, je ztlumeno osvětlení, které reprezentuje přirozené okolní světlo. Toto světlo je v Unity implementováno jako `Directional light`. Po zatmavení scény jsem se rozhodl přidat modelu auta světla. Tato světla jsou vytvořena jako prázdné objekty s komponentou `Light` – obrázek 4.10 – a nastaveným typem světla na `Spot`. U zadních světel je přidána pouze luminiscence materiálu.

Při implementaci jsem narazil na problém, kdy světla různě blikala, když měla svítit. Převážně se to stávalo, když se potkala dvě světla (například když auto osvětlilo objekt). Tento problém se vyřeší tím, že se u komponenty `Light` nastaví render jako `Important`.



Obrázek 4.10: Příklad Komponenty Light u objektu, která vytvoří zdroj světla. Tomuto zdroji lze nastavit typ, dosah, barvu, intenzitu a další. Důležitou vlastností v případě více světel je Render Mode. Další důležitým atributem je, jaké stíny dané světlo vrhá.

4.9 Kamera

Kamera je v objektu zvaném `Main Camera` a je umístěna v určité poloze za hráčem. Tento objekt s kamerou je v hierarchii zanořený pod hráče. Nikoliv pod model samotného auta, ale pod objekt, pod kterým je zanořen i model auta. Kamera pak zůstává ve stejné poloze relativně k objektu, ke kterému je připnuta (zanořením v hierarchii). Pokud by byla kamera připnuta modelu auta, pak by měla relativní pozici k tomuto autu a při každém zatočení (kde je zahrnuta i rotace auta) se pak otočí i kamera, což způsobuje zmatek a nepřehlednost.

Pokud je kamera připnuta objektu, který auto zanořuje, pak plynule následuje hráče a nerotuje při každém pohybu auta.

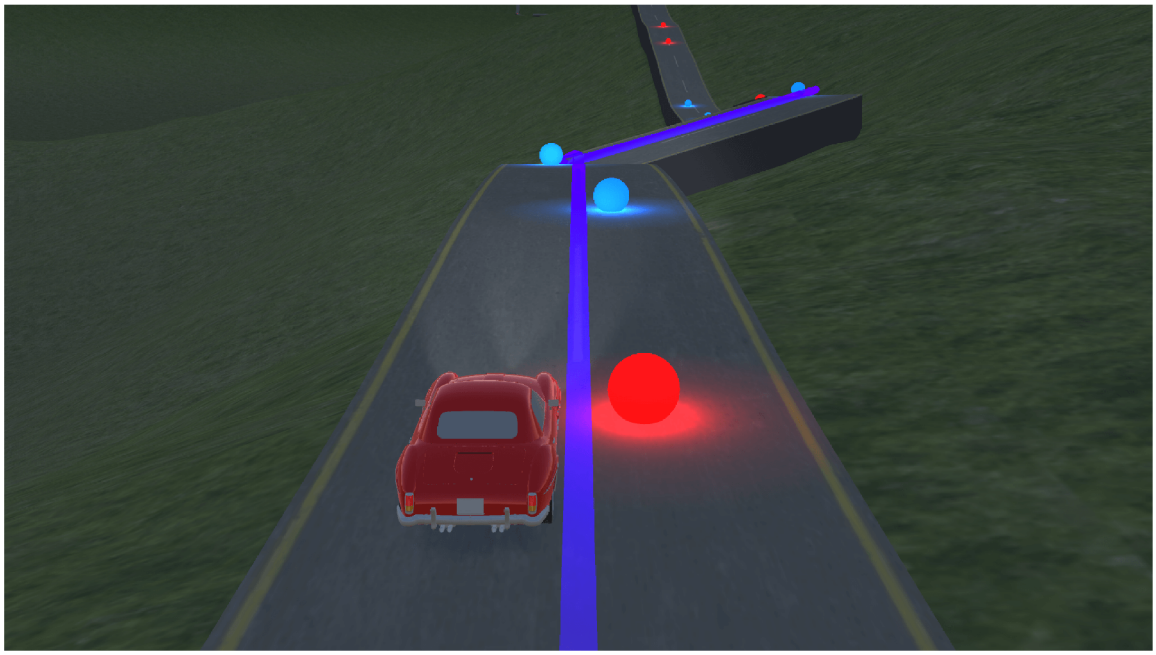
U kamery se podle síly signálu mění Field of View (FOV). Tato změna Field of View navozuje dojem, že auto jede rychleji nebo pomaleji. Cílem je měnit tuto vizuální rychlost auta podle energie signálu.

Hodnota Field of View jde změnit v komponentě `Camera`. Použití různých metod plynulé změny mi nefungovalo, vždy se hodnota Field of View měnila rázově. Pro plynulou změnu

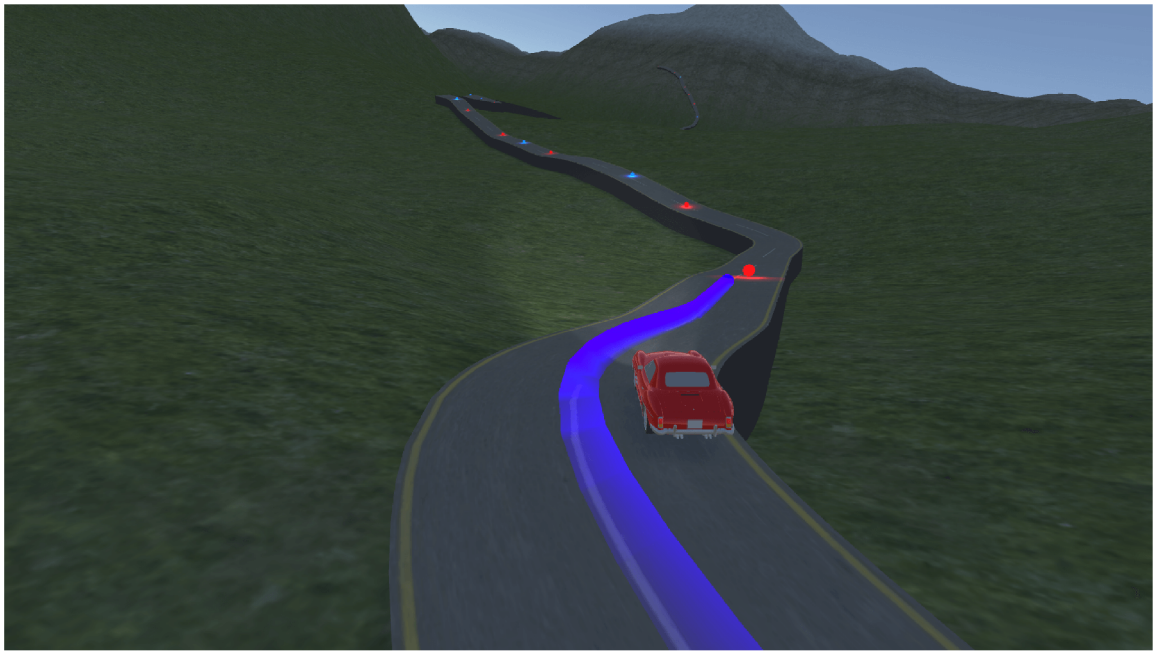
této hodnoty každým snímkem (v metodě `Update()`) je odečítána nebo přičítána malá daná hodnota.

Kamera se nedá ovládat. Proběhly pokusy s přidáním ovládání, ale se současným použitím rotace kamery a změny Field of View hra byla nepřehledná a zmatená.

4.10 Výsledná hra

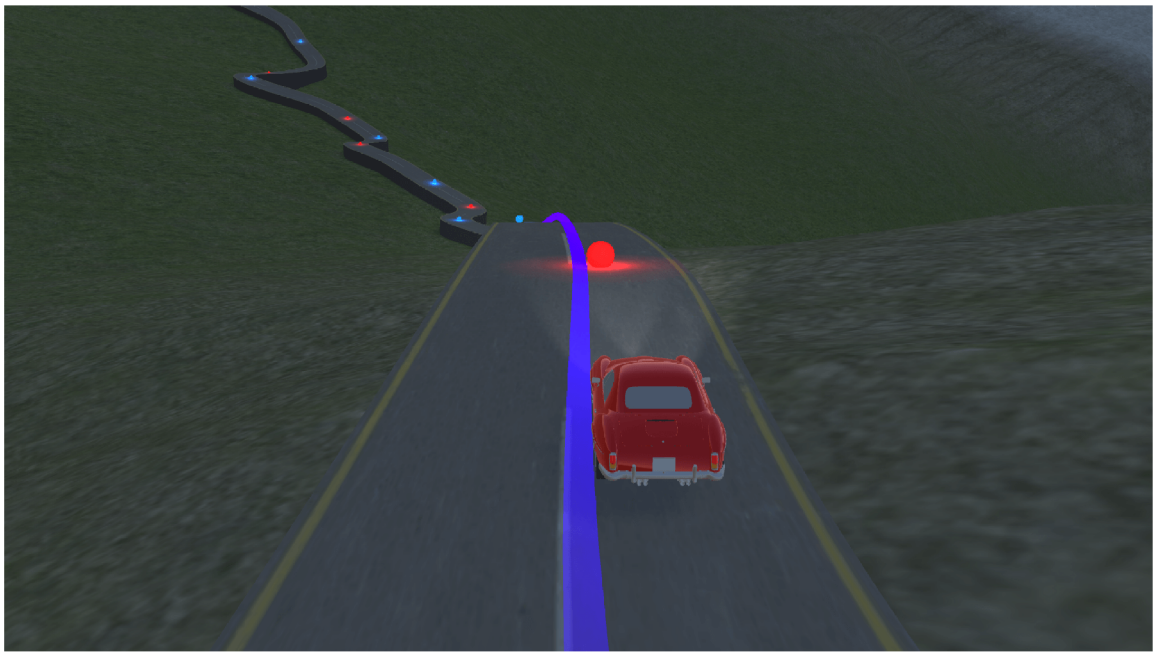


Obrázek 4.11: Ukázka ze hry, na obrázku lze vidět, jak koule se zdrojem světla osvítí model auta.



Obrázek 4.12: Ukázka ze hry, na obrázku lze vidět delší kus silnice a objekty se zdrojem světla.

Obrázek 4.13: Ukázka ze hry, na obrázku je objekt reprezentující skladbu a model auta následující tento objekt.



Obrázek 4.14: Ukázka ze hry.

Kapitola 5

Závěr

V práci je popsán princip využití analýzy hudby při tvorbě hry v herním engine Unity. Analýza hudby byla použita při tvorbě trati v terénu a při vizualizaci hudby na objektech ve hře. Pro detekci beatů bylo popsán postup a využití Fourierovy transformace v detekci beatů. Pro tvorbu terénu byl popsán a využit způsob procedurálního generování.

Pro další pokračování práce bych ve hře přidal generování vegetace, která by byla generována v závislosti na výšce terénu. Dále bych hru převedl do nové verze Unity a upravil způsob vykreslování osvětlení s možnostmi nové verze pro dosažení lepšího vizuálu. Po těchto úpravách bych hru publikoval jako nezávislou hru.

Způsob detekce beatů fungoval spolehlivě pro využití ve tvorbě hry, proto se jako další možnost pokračování nabízí vytvoření assetu na detekci beatů pro Unity, který by byl zveřejněn v Assets store přímo v Unity editoru. Pro hru vytvořenou v rámci této práce nebylo potřeba detekci provádět ve více vláknech, ale pro zveřejnění v Assets store by bylo vhodné provést úpravu, aby se prováděla ve více vláknech.

Všechny body zadání byly splněny a výsledkem je implementovaná hra, která vytvoří procedurálním generováním základní terén, podle vybrané skladby tímto terénem najde cestu a při hraní vizualizuje beaty ve skladbě na objektech na silnici.

Literatura

[1] Bello, J. P.; Daudet, L.; Abdallah, S.; aj.: A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, ročník 13, č. 5, Sep. 2005: s. 1035–1047, ISSN 1063-6676, doi:10.1109/TSA.2005.851998.

[2] Illing, L.: Fourier analysis. 2008.
URL https://www.reed.edu/physics/courses/Physics331.f08/pdf/Fourier.pdf

[3] Perlin, K.: Improving Noise. *ACM Trans. Graph.*, ročník 21, č. 3, Červenec 2002: s. 681–682, ISSN 0730-0301, doi:10.1145/566654.566636.
URL http://doi.acm.org/10.1145/566654.566636

[4] Salomon, D.: *Curves and Surfaces for Computer Graphics*. Berlin, Heidelberg: Springer-Verlag, 2005, ISBN 0387241965.

[5] Sicart, M.: *Defining Game Mechanics*. [Online; navštíveno 01.05.2019].
URL http://gamestudies.org/0802/articles/sicart

[6] Togelius, J.; Kastbjerg, E.; Schedl, D.; aj.: What is Procedural Content Generation? Mario on the borderline. 01 2011, doi:10.1145/2000919.2000922.

[7] Unity: *Game engines—how do they work?* [Online; navštíveno 01.05.2019].
URL https://unity3d.com/what-is-a-game-engine

[8] Unity: *Unity User Manual*. [Online; navštíveno 01.05.2019].
URL https://docs.unity3d.com/Manual/

[9] Zechner, M.: *Onset detection tutorial*. [Online; navštíveno 01.05.2019].
URL https://www.badlogicgames.com/wordpress/?cat=18

[10] Zucconi, A.: *Surface shaders in Unity3D*. [Online; navštíveno 01.05.2019].
URL https://www.alanzucconi.com/2015/06/17/surface-shaders-in-unity3d/

Příloha A

Plakát