Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Hra Minecraft v Unity3D

2019

Bc. Tomáš Polák

Vedoucí práce:
Mgr. Martin Trnečka, Ph.D.

Studijní obor: Aplikovaná Informatika,
prezenční forma

Bibliografické údaje

Autor:	Bc. Tomáš Polák
Název práce:	Hra Minecraft v Unity3D
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Aplikovaná Informatika, prezenční forma
Vedoucí práce:	Mgr. Martin Trnečka, Ph.D.
Počet stran:	31
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Tomáš Polák
Title:	Minecraft game in Unity3D
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Martin Trnečka, Ph.D.
Page count:	31
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce pojednává o vývoji hry typu Minecraft za použití enginu Unity3D a modelovacího nástroje Blender. V práci je popsán vývoj jednotlivých stěžejních mechanismů vyskytujících se ve hře a následné rámcové porovnání s originální hrou Minecraft.

Synopsis

This thesis is about creating version of Minecraft game using engine Unity3D and modeling tool Blender. There is described development process of all essential game mechanics and comparison with original Minecraft game.

Klíčová slova: Unity3D; Blender; Minecraft

Keywords: Unity3D; Blender; Minecraft

Děkuji Martinu Trnečkovi za odborné vedení diplomové práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Už od střední školy se věnuji tvorbě her v Unity3D [1]. Většina mých projektů vždy spočívala ve snaze pokusit se vytvořit kopii nějaké veřejně známé a oblíbené hry. Vyzkoušet si implementovat hlavní mechaniky hry a tím si rozšířit rozhled v tomto odvětví.

Většinou to byly jednoduché 2D hry, které získaly svou popularitu díky jednoduché a návykové hratelnosti, a tak jednotlivé mechaniky nebyly složité na implementaci.

Za poslední dva roky jsem nasbíral velké množství zkušeností, proto jsem se rozhodl pro svou diplomovou práci vytvořit už trochu rozsáhlejší a složitější hru.

1.1 Hra Minecraft

Minecraft [2] díky svým skvěle navrženým herním mechanikám a takzvanému sandbox[1] stylu udělal velkou díru do světa a od doby vzniku této hry můžeme na internetu najít nespočet následovníků, kopií a variací této hry.

Minecraft je hra o přežití a budování v procedurálně vygenerovaném světě, který je vytvořen jen pomocí kostek. Každá kostka v tomto světě může být vytěžena, uložena do inventáře hráče a následně taky umístěna zpátky do světa. Tato mechanika umožňuje hráči stavět z kostek libovolné stavby.

Obrázek 1: Ukázka stavby vytvořené ve hře Minecraft[11]

[1] za sandbox se označují hry, které hráče nijak zvlášť nelimitují v tom co může ve hře dělat. Zpravidla si hráč může měnit virtuální svět hry podle sebe.

Každá kostka v Minecraftu také reprezentuje jeden materiál, ze kterého hráč může vytvářet jiné předměty. Například základní stavební prvky pro domy jako jsou, dveře, schody, žebříky nebo také bedny pro úschovu vytěženého materiálu, louče, koleje, vozíky, různé zbraně a nástroje, oblečení a brnění a další.

Podle Minecraft Wiki se v základní verzi hry nachází 967 předmětů, se kterými může hráč pracovat [3]. Ve světě se pak vyskytují zvířata, které může hráč zabíjet pro získání jídla a nebo různí nepřátelé, kteří hráče ohrožují hlavně v noci.

Ve hře se také nachází určitý typ předmětů, který funguje stejně jako logické obvody v počítači. V Minecraftu je tedy možné vytvářet si zautomatizované procesy jako třeba automatické zalívaní polí, následné sklizení a uložení do beden, nebo si v Minecraftu můžete „naprogramovat“ vlastní kalkulačku.

Obrázek 2: Plně funkční 16-bit kalkulačka vytvořená ve hře Minecraft.[12]

Minecraft se již od svého vzniku těší obrovské oblibě, a tudíž není divu, že studio Mojang, které tuto hru vyvíjí, v roce 2014 odkoupil Microsoft za 2,5 miliardy dolarů [4].

1.2 Jak začít

Z početného množství mechanizmů, které se v Minecraftu nacházejí, mě nejvíc zaujalo procedurální generování světa z kostek. S procedurálním generováním jsem neměl žádné zkušenosti, proto jsem se rozhodl začít právě s tím. Při vývoji jsem pak přidával další základní herní mechanismy originálního Minecraftu jako je ukládání celého světa na disk, vytváření předmětů, umělou inteligenci zvířat, bojový systém a další.

2 Použité technologie

Pří výběru technologií jsem vsadil na nástroje, které dobře znám a mám s nimi pokročilé zkušenosti, abych se nemusel zabývat studováním pro mě nových nástrojů ale tvorbě samotných mechanik. I tak se ale v mém seznamu objevil jeden nováček a tím je A* Pathfinding project [5].

2.1 Unity3D

Unity3D je pro mě dobře známý herní engine, se kterým pracuji již sedm let, a tak mám poměrně bohaté zkušenosti. Věděl jsem, že mě při vývoji jádra hry nebude nic brzdit, proto bylo Unity3D jasná volba. Unity3D je multiplatformní herní engine s podporou vývoje jak 3D, tak i 2D her.

V současné době je kolem tohoto nástroje obrovský boom a uchytil se hlavně v indie scéně hned z několika důvodů. Unity3D poskytuje zdarma prémiové funkce, které jsou u konkurence zpoplatněny.

Dále je Unity oproti jeho konkurentům uživatelsky velmi přívětivé, a i naprostí začátečníci jsou schopni se během chvilky v tomto softwaru zorientovat.

Pro vývoj her je Unity3D naprosto soběstačný nástroj. Pokud chcete začít vytvářet hry, stačí si na oficiálních stránkách Unity stáhnout nejnovější verzi instalátoru (v současné době existuje verze pro Windows a Mac, vývojáři však pracují i na verzi pro Linuxové distribuce), se kterou se do počítače nainstaluje i základní vývojové prostředí MonoDevelop a vy můžete ihned začít vydávat hry na více než 25 platforem [1].

Unity3D se pyšní obrovskou komunitou, která neustále vytváří nové assety, články a návody jak s Unity nejlépe pracovat, proto pro mě nebyl problém najít řadu článků zabývajících se právě tvorbou Minecraftu.

2.2 A* Pathfinding Project

A* Pathfinding project je komerční balíček obsahující řešení hledání cest vyvíjen pro Unity3D. Tento balíček obsahuje také omezenou verzi, která je zcela zdarma a tu jsem použil pro svůj projekt [5]. Pathfinding v tomto projektu používám pro pohyb zvířat po vygenerovaném světě. Zvířata se náhodně pohybují po světě a pokud na ně zaútočíte, tak se před vámi snaží utíkat a některé se vás naopak

snaží ulovit. Abych tohoto chování mohl docílit musel jsem vytvořit systém hledání cest. Samotné Unity3D obsahuje své vlastní řešení. Jejich pathfinding systém byla moje první volba. Tento systém je ale stále ve vývoji, a tak jsou některé funkce nedostačující a rychle jsem narazil na problémy, kvůli kterým jsem musel vybrat jiné řešení. Těchto systémů pro Unity existuje celá řada. Na Unity Asset Storu jsem narazil na 31 různých balíčků. Hlavní kritérium při výběru byla možnost dynamického generování navigační mapy, a také aby byl balíček přístupný zdarma. Nejlépe hodnocený balíček na AssetStoru byl právě A* Pathfinding project, který splnil oba požadavky.

2.3 Blender

Na modelovací software jsem od začátku projektu neměl moc velké nároky, protože všechny modely ve hře Minecraft jsou vytvořeny jen pomocí krychlí a kvádrů. Blender jsem si vybral také hlavně proto, že s tímto softwarem mám už základní zkušenosti a je dostupný zcela zdarma [6]. Díky tomu, že je Blender open-source projekt, vznikla kolem tohoto nástroje početná komunita, a tudíž není problém nalézt značné množství návodů, které mi tvorbu modelů do hry velmi urychlily. Nevýhodou tohoto softwaru je poměrně neintuitivní prostředí, které je z mé zkušenosti pro začínající uživatele dost těžkopádné.

2.4 Gimp

Stejně jako na modely, tak ani na samotné textury není Minecraft moc náročný. Všechny textury ve hře jsou nakresleny v takzvaném pixel artu. Ani já jsem při tvorbě projektu neměl na textury velké nároky, a tak jsem sáhl open-source softwaru Gimp, který obsahuje všechny potřebné funkce pro tvorbu textur, které jsem potřeboval [7]. Gimp mohu doporučit všem začátečníkům při tvorbě her, protože je velmi snadné v tomto programu vytvářet základní 2D grafiku, proto pokud hledáte nástroj pro nějaké rychlé „prototypování“, je Gimp ideální volba.

2.5 Audacity

Pro úpravu zvukových efektů do hry jsem si vybral taktéž open-source řešení, a to software Audacity [8]. V této oblasti jsem úplný začátečník a Audacity jsem používal jen pro základní úpravy již hotových zvukových efektů.

3 Herní mechanismy

V této kapitole podrobně popisuji implementaci jednotlivých herních mechanizmů, které jsem v rámci diplomové práce do hry zahrnul. Základní stavební kámen pro celou práci je procedurální generování světa, ke kterému jsem poté přidával další mechanismy, aby výstupem práce byla kompletní hra. Od začátku jsem při návrhu a samotné tvorbě hry dával velký důraz na možnost rozšíření hry o nové prvky. V současném stavu je tedy možné do hry přidat libovolné množství druhů bloků, předmětů, které může hráč vyrábět nebo jednotek, které se po světě pohybují, a to bez jakéhokoliv programování skriptů. Všechny nové prvky je možné vytvořit a umístit do hry jen pomocí Unity Editoru[2].

3.1 Procedurální generování světa

Procedurální generování světa vytvořeného pomocí bloků bylo pro mě hlavní a největší výzvou. Jak jsem již zmiňoval, s touto problematikou jsem na začátku práce neměl žádné zkušenosti, a tak jsem strávil několik hodin studováním různých způsobů implementace. První článek, který se zabýval generováním světa z kostek v Unity3D, popisoval způsob klonování jedné předem vytvořené kostky. S tímto způsobem jsem ale poměrně rychle narazil na výkonnostní problémy, proto jsem musel začít úplně od znovu.

3.1.1 Struktura světa

Když jsem se dostal do této problematiky hlouběji, zjistil jsem, že správné řešení je vytvořit svět z takzvaných chunků, což jsou objekty o určitém rozměru. V mém případě je rozměr jednoho chunku $16 \cdot 16 \cdot 16$ bloků. Samotné bloky ale v chunku fyzicky nejsou. Chunk pouze obsahuje informace o tom, na které pozici je pevný blok (blok hlíny, kamene, dřeva) a jaké strany jsou u jednotlivých bloků viditelné. Samotný algoritmus pak prochází vygenerované chunky a na pozicích pevných bloků vykresluje textury a vytváří collider[3].

3.1.1.1 Třída Block

Třída *Block* definuje strukturu bloku, ze kterých jsou složeny jednotlivé chunky.

Tato třída je rodičovská a samotné třídy bloků jako je blok kamene, blok hlíny nebo blok dřeva z této třídy budou dědit.

Mimo jiné bude tato třída obsahovat:

- Informaci o globální pozici ve světě.
- Parametr *Update* určující, jestli potřeba tento blok aktualizovat.
- Funkci *IsSolid*, která vrací informace o tom, které stěny bloku jsou pevné.

[2]Unity Editor je hlavní software dodávaný s Unity3D enginem

[3]collider je oblast, která pokrývá objekt a zajišťuje kolize s ostatními objekty ve scéně.

- Funkci *GetUVs*, která vrací pozici textury na atlasu textur.

Tato třída reálně obsahuje poměrně velké množství dalších parametrů a funkcí, které ale nejsou pro generování světa tak důležité.

3.1.1.2 Třída Chunk

Třída *Chunk* definuje strukturu jednotlivých dílů, ze kterých je svět postaven.

Tato třída už je poměrně složitější než třída bloku a obsahuje daleko náročnější funkce na implementaci. Nejdůležitější funkce a parametry této třídy jsou:

- Parametr globální pozice ve světě.
- Trojrozměrné pole obsahující instance třídy bloku.
- Funkci, která aktualizuje všechny bloky nacházející se v tomto chunku.
- Funkci, která vytvoří samotný mesh pro renderování.
- Funkci, která vytvoří kompletní collider chunku.

Stejně jako třída bloku i tato třída obsahuje řadu dalších parametrů a pomocných funkci, které ale v tuto chvíli nejsou důležité.

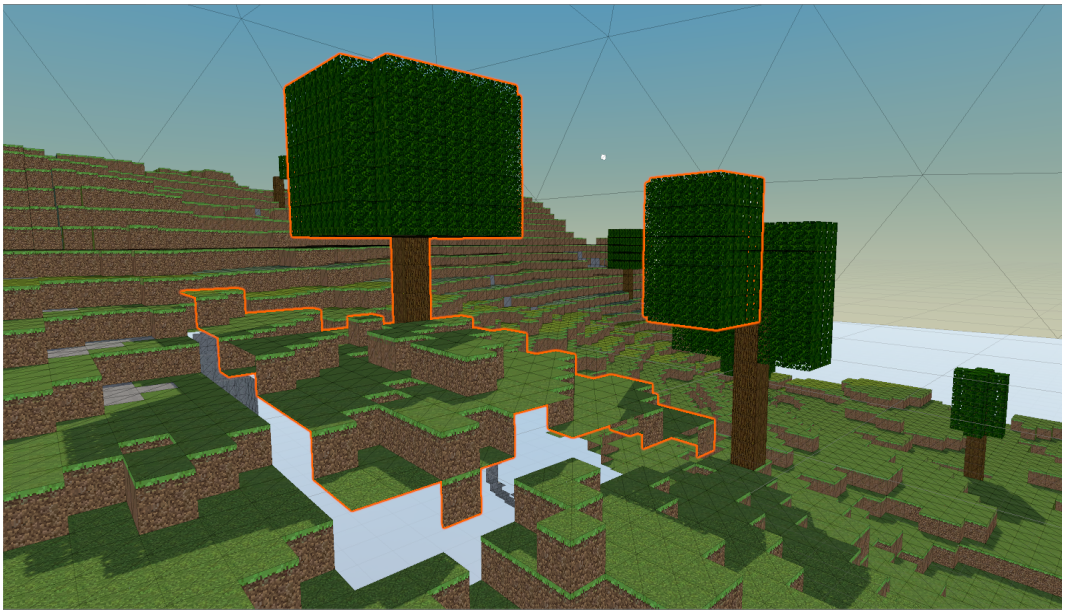
3.1.1.3 Třída World

Třída *World* definuje strukturu samotného světa poskládaného z chunků.

Tato třída má pro generování světa v podstatě jediný důležitý parametr a tím je seznam jednotlivých instancí třídy *Chunk* a jejich globální pozice ve světě. Pozice je v tomto seznamu důležitá hlavně z hlediska vyhledávání potřebných chunků při postupném načítaní světa buď z disku nebo z paměti počítače.

Funkce této třídy, které jsou důležité pro generování světa jsou:

- Funkce *SetBlock*, která vloží blok na určitou pozici ve světě.
- Funkce, která vrátí blok na určité pozici.
- Funkce, která vytvoří chunk na určitě pozici.



Obrázek 3: Jeden chunk oddělený od zbytku světa

3.1.2 Algoritmus vykreslování bloků

Algoritmus, který vykresluje samotné bloky ve světě je už poměrně jednoduchý. Algoritmus postupně prochází seznam chunků uložený ve třídě *World*.

U každého chunku se zjistí, jestli je potřeba chunk aktualizovat. To zjistí tak, že si projde seznam všech bloků tohoto chunku, a pokud v seznamu existuje blok s kladným parametrem *Update*, tak celý chunk aktualizuje.

Aktualizace chunku probíhá tak, že si algoritmus u každého bloku zavolá funkci *IsSolid* – tak bude algoritmus vědět, jaké strany bloku má vykreslit a kde má vytvořit collider. Poté si zavolá funkci *GetUVs*, aby věděl, jakou texturu má na jednotlivé strany vykreslit.

Z těchto informací pak algoritmus vytvoří samotný mesh pro tento chunk, ten vykreslí a také vytvoří jeho collider.

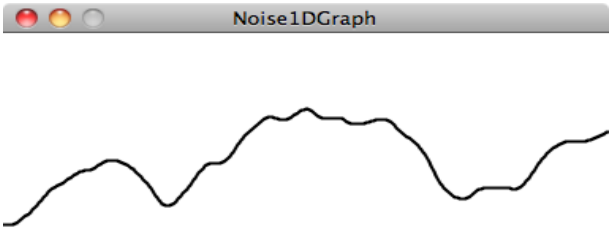
Tento algoritmus se spouští pokaždé, když je ve třídě *World* volána funkce *SetBlock*.

3.1.3 Perlin noise

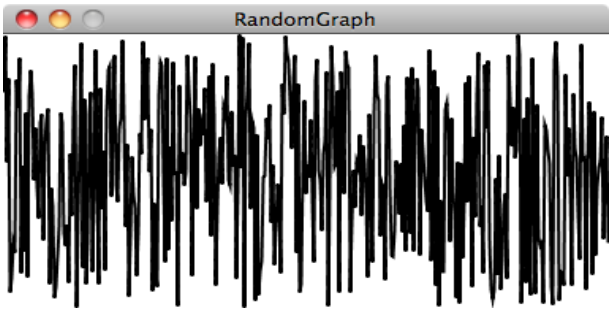
Pokud nechceme, aby náš vygenerovaný svět byl jenom obrovská krychle a chceme docílit něčeho, co aspoň trošku připomíná reálnou krajinu, potřebujeme jednotlivé chunky vytvářet s jakousi náhodnou deformací. K tomu můžeme využít grafický šum – konkrétně tedy Perlin noise.

Perlin noise je algoritmus, který vyvinul Ken Perlin, když v 80. letech minulého století pracoval na filmu Tron. Chtěl vymyslet způsob, jak vytvořit pro tento film reálnější vzhled textur pro počítačově generované efekty ve filmu.

Tento algoritmus spočívá v tom, že vrací náhodné hodnoty závislé na prostoru nebo čase, ale s takzvaně uhlazenější postupností než kdybychom použili obyčejný algoritmus pro generování náhodného čísla. Zjednodušeně funguje tak, že pokud bychom generovali náhodná čísla závislá na prostoru, tak pro dvě různé pozice nacházející se vedle sebe, algoritmus vygeneruje náhodné hodnoty, které ale nejsou od sebe moc vzdáleny [9].



Obrázek 4: Graf hodnot funkce Perlin noise.[9]



Obrázek 5: Graf hodnot funkce pro náhodné číslo.[9]

Tímto způsobem pak generujeme jednotlivé sloupce bloků každého chunku. Postupně procházíme pozice všech sloupců chunku a zadáváme je jako parametry funkce Perlin Noise, která nám vrátí výšku toho určitého sloupce. Tímto docílíme postupného stoupání kopců a dalších nerovností v terénu jako jsou třeba jeskyně nebo náhodný výskyt stromů.

Unity3D obsahuje vlastní implementaci funkce Perlin noise, která je naprosto dostačující pro generování světa jako je tento. Já jsem použil jinou open-source implementaci tohoto algoritmu, kde jsem se mohl podívat, jak samotný algoritmus vypadá, a tak lépe pochopil jak tato funkce funguje.

Teď už jen zbývá určit druh bloků, kterými jednotlivé sloupce vyplníme. Stejně jako v originální hře Minecraft, určuji druh bloku podle výšky.

Bloky, které mají větší výšku než je hodnota vygenerována funkcí Perlin noise, budou vzduch. To znamená, že ani jedna jejich stěna nebude vykreslena a nebude pro ně vytvořen collider. Bloky s výškou o jednu nebo o dvě jednotky nižší, než je hodnota funkce, budou hlína a ostatní nižší bloky budou kámen. Nejnižší řada bloků pak musí být nezničitelná, aby se hráč nemohl „prokopat“ pryč ze světa.

3.2 Uložení světa na disk

Ukládání světa na disk je užitečné hned ze dvou důvodů. Prvním důvodem je ukládání rozehraného světa, který můžeme později zase nahrát z disku a pokračovat ve hraní. Další důvod je ,že pokud budeme ukládat svět rozdělený do jednotlivých chunků, můžeme tuto funkcionalitu využít při postupném načítání světa a tím ušetřit výkon.

Postupně budeme procházet všechny chunky, které chceme uložit. Pro každý chunk si vytvoříme soubor, který bude mít v názvu globální pozici tohoto chunku (tento způsob ukládání nám později pomůže při postupném načítání světa, kdy budeme načítat chunky podle jejich pozice). Poté algoritmus prochází jednotlivé bloky, které chunk obsahuje a v pořadí, v jakém jsou uloženy v seznamu, postupně zapisuje do souboru ID, které určuje druh bloku (kámen, hlína, dřevo nebo vzduch).

Nahrávání chunků funguje velmi obdobně. Vytvoří se nový prázdný chunk na určené pozici ve světě. Podle této pozice se potom nahraje soubor z disku a postupně chunk naplní novými bloky podle ID uložených v souboru. Stačí nám tedy dodržovat stejné pořadí, v jakém byly bloky ukládány a nahraný chunk bude vypadat úplně stejně, jako ten původní ukládaný.

3.3 Postupné načítání / ukládání částí světa

Další velmi důležitá mechanika, bez které bychom nemohli efektivně udržovat tento nekonečný generovaný svět. Aby byla hra dobře hratelná bez znatelných záseků, musíme ve scéně zobrazovat jen ty chunky, které jsou od hráče v určité vzdálenosti. S tím jak se hráč pohybuje po světě se zobrazují nové chunky, a ty které hráč nechává za sebou se zase skrývají.

3.3.1 Princip algoritmu

Jak je zmíněno v kapitole 3.3, ve světě se zobrazují jen chunky do určité vzdálenosti od hráče. Ostatní chunky, které jsou mimo tento okruh kolem hráče, musí být někde uloženy a čekat než se k nim hráč přiblíží, aby mohly být zobrazeny.

Skryté chunky můžeme jednoduše nechávat v paměti počítače a odtud jej také znovu načítat a zobrazovat. Tento způsob je nejrychlejší. Nemusíme se starat o žádné ukládání, žádnou serializaci, jen si udržujeme reference na každý chunk.

Takhle fungoval můj algoritmus při první iteraci. Tohle řešení je ale velmi omezené, pokud by se hráč pohyboval světem dost daleko, vygenerované chunky by postupně alokovanou paměť zaplnily a hra by přestala fungovat. Proto se musí některé chunky ukládat na disk počítače. Algoritmus tak funguje se dvěma vzdálenostmi kolem hráče.

První vzdálenost určuje, které chunky budou ve světě aktivní a viditelné.

Druhá vzdálenost, která je trochu větší, určuje, které chunky budou neaktivní, uložené v paměti.

Ostatní chunky, které se nachází ještě dál budou uložené na disku počítače.

3.3.2 Ukládaní chunků

Při ukládání nepotřebných chunků do paměti, a následně na disk, algoritmus postupně prochází vytvořené chunky, a ty které jsou od hráče dál než je první okruh, jednoduše vypne a chunky, které jsou ještě dál, smaže z paměti a uloží na disk.

3.3.3 Nahrávaní chunků

Při nahrávání chunků zpátky do hry už je postup trochu komplikovanější. Pokud by algoritmus pracoval jen s pamětí, tak je to jednoduché. Algoritmus projde všechny neaktivní chunky, a ty, které jsou dostatečně blízko hráči jednoduše zapne. V případě chunků, které jsou uloženy na disku by byl tento proces velmi náročný. Algoritmus by si musel nahrát do paměti všechny existující chunky a ty potom procházet tak, jak je popsáno výše.

Ideální řešení tedy je, aby algoritmus z disku načítal jen ty chunky, které jsou potřeba.

Při vytváření tohoto algoritmu jsem nejprve používal řešení, které spočívalo v tom, že jsem si vypočítal všechny pozice chunků, které leží mezi hráčem a určitou vzdáleností. Tyto pozice jsem poté procházel a podle názvu načítal chunky ze souboru.

Tento způsob fungoval poměrně dobře, nicméně jsem chtěl tento algoritmus víc zefektivnit, protože algoritmus tímto způsobem procházel mnoho zbytečných pozic.

Finální řešení tohoto algoritmu nakonec vypadá tak, že si algoritmus při každém průchodu zjistí, které chunky leží v tuto chvíli nejdál od hráče. Poté postupně projde všechny tyto hraniční chunky a pro každý z nich si vypočítá pozice jeho potencionálních bezprostředních sousedů. Pokud se některý z těchto sousedů ještě nachází dostatečně blízko k hráči, bude načten.

Výsledek tohoto algoritmu je následující.

Při pohybu hráče se nejbližší chunky postupně deaktivují a ty deaktivované zase ukládají na disk. Druhým směrem se pak nejvzdálenější chunky nahrávají

z disku do paměti a nechají se deaktivované. Ty deaktivované, které se nacházejí blízko hráči se poté zobrazují.

3.3.4 Paralelizace

Poslední věc, kterou jsem při postupném ukládání a nahrávání světa řešil, byla snaha tento celý proces nějakým způsobem paralelizovat. Obecně způsob, kterým se svět generoval a ukládal na disk, fungoval správně. Problém ale nastal, když se v jednu chvíli naráz načítalo / ukládalo mnoho chunků. Načítání třeba dvou set chunků zapříčinilo zaseknutí hlavního vlákna vždy třeba na jednu až dvě sekundy. Řešení, které se nabízelo bylo rozdělit celý tento proces do více vláken, kde by se každé vlákno staralo o část z celkového počtu chunků, které je potřeba zpracovat.

Rozdělení takové úlohy do více vláken v Unity však není jednoduché. Unity sice dovoluje vytvoření vláken, které poskytuje jazyk C#, ale veškeré funkce z Unity API musí být volány v hlavním vlákně.

To znamená, že všechny potřebné výpočty by bylo možné rozdělit do více vláken, ale tyto operace zaseknutí hlavního vlákna nezpůsobovaly. Zaseknutí hlavního vlákna způsobilo až samotné aktivování většího množství chunků naráz a aktivování objektu ve scéně vyžaduje volání funkce z Unity API.

Samotné Unity má ve své API vlastní implementaci práce s vlákny, která je zase omezena na práci výhradně se strukturami, což by můj problém také nevyřešilo [10].

3.3.5 Coroutines

Další věcí, kterou se dá v Unity simulovat paralelní běh programu jsou Coroutines.

Běžné funkce, které se při programování skriptů v Unity používají, proběhnou vždy celé v jednom framu. To znamená, že stejně jako v mém případě, pokud je provedení dané funkce náročné, tak se hlavní vlákno zasekne, dokud celá funkce neproběhne až do konce. Tomuto chování se můžeme vyhnout, pokud použijeme coroutinu. Průběh těchto funkcí můžeme totiž pozastavovat na libovolnou dobu, nebo průběh funkce rozdělit do více framů [10].

A právě této funkcionality jsem využil při postupném načítání světa.

Funkce, která postupně prochází seznam chunků, které mají být zprácovány je implementována pomocí coroutiny a její výpočet je navrhut tak, že funkce aktivuje / deaktivuje vždy jen určitý počet chunků v jednom framu.

Postupným testováním jsem pak přišel na ideální počet chunků, který může být za jeden frame zpracován. Tuto hodnotu nebylo jednoduché najít, protože příliš nízký počet chunků za jeden frame způsoboval pomalé načítání chunků a hráč tak měl možnost spadnout do prázdnoty. Vysoká hodnota zase způsobovala zaseknutí hlavního vlákna. V současné verzi algoritmu je zpracováno 25 chunků v jednom framu.

3.4 Druhy bloků

V originální hře Minecraft se nachází početné množství různých druhů bloků, ze kterých je svět vytvořen, od základních druhů, přes kámen, hlína, písek, dřevo až po vzácné kovy a drahokamy.

Implementace těchto druhů bloků byla jednoduchá. Jednotlivé bloky se od sebe liší jen pozicí textury na atlasu textur a dobou těžby. V Minecraftu existují ještě bloky, které mají nějaké speciální vlastnosti - například písek oproti ostatním blokům ve hře podléhá gravitaci.

V současné verzi mé hry se nachází jen tři různé druhy bloků, ale celý systém je navrhnut tak, aby bylo kdykoliv možné další druhy bloků přidávat, bez zásahu do samotného enginu hry.

3.5 Těžba materiálů

Těžba materiálů je klíčový prvek hry. Celá hra funguje na principu, kdy si hráč natěží potřebné materiály k vytvoření různých předmětů. Každý druh bloku po vytěžení vytvoří předmět materiálu, který si hráč může umístit do svého inventáře.

Ke každému bloku jsem tak přidal možnost výběru, jaký materiál se po vytěžení vytvoří a jaké množství.

Každý blok má také různou dobu potřebnou k vytěžení. Tyto doby lze zkracovat použitím vhodné předmětu, který si hráč může vyrobit. V originální hře Minecraft jsou také bloky, které je možné vytěžit jen použitím určitého předmětu. Tuto funkcionalitu jsem do své práce nezahrnul.

3.6 Stavba bloků

Vytěžené materiály, které má hráč umístěné ve svém inventáři nemusí sloužit výhradně k výrobě jiných předmětů. Samotné vytěžené bloky lze totiž umístit zpátky do světa.

Stavba bloků je dalším z klíčových prvků hry a umožňuje hráči stavět z bloků libovolné stavby.

3.7 Předměty

Za předmět ve hře považujeme všechny věci, které si může hráč uložit do inventáře. I samotné vytěžené bloky jsou předměty. Mimo vytěžené bloky a jídlo, které může hráč získat zabitím zvířat, si všechny předměty musí vyrobit.

Jak bylo napsáno v úvodu, v originální hře Minecraft existují stovky různých předmětů. V mé verzi jsem připravil jen pár předmětů, na kterých je demonstrován celý koncept. Stejně jako to bylo u různých druhů bloků, tak i tady je systém vyrábění předmětů navrhnut tak, aby bylo možné další předměty přidávat bez zásahu do kódu. Nyní jsou ve hře implementovány 4 kategorie předmětů:

1. Block
2. Item
3. Weapon
4. Consumable

Do první kategorie patří všechny vytěžené bloky, jejich specifikací je to, že jdou umísťovat zpět do světa, tím se liší od druhé kategorie, která je mimo tento rozdíl úplně stejná.

Do druhé kategorie patří materiály, které může hráč vyrobit v inventáři.

Do třetí kategorie patří všechny zbraně a nástroje pro těžení. Specifický parametr těchto předmětů je *Damage*. Tento parametr u zbraní určuje poškození, které bude způsobeno při útoku na nepřátele. U nástrojů pro těžení tento parametr určuje čas, o který se zkrátí doba těžení, pokud se bude tento nástroj používat na správný druh bloku.

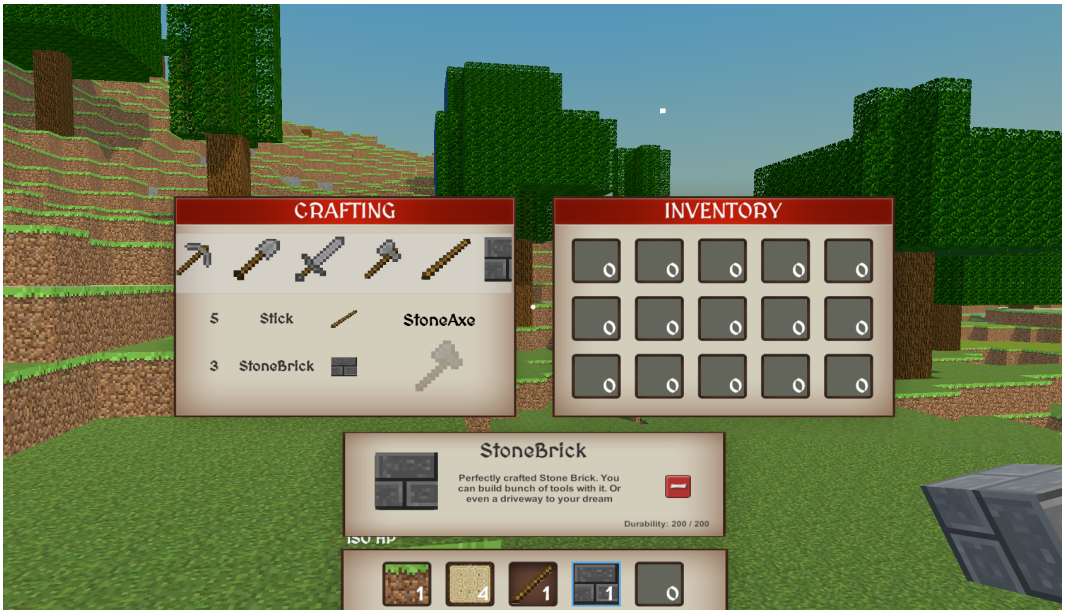
Dalším specifickým parametrem je pak možnost určit právě ten druh bloku, pro který je nástroj určen.

Poslední specifický parametr je durabilita předmětu. Každý předmět z této kategorie, který hráč vyrobí, se umístí do inventáře s určitou durabilitou. Ta se potom snižuje, když hráč předmět používá. Když durabilita klesne na nulu, předmět se zničí.

U každého bloku, který se světě nachází je nastavený parametr kvality. Kvalita určuje množství durability, kterou předmět ztratí po vytěžení tohoto bloku. Ztráta durability se pak ještě počítá následovně. Pokud hráč vytěží blok, na který je použitý nástroj určen, tak se z durability předmětu odečte polovina kvality bloku. Na druhou stranu, pokud hráč vytěží blok, na který tento použitý nástroj není určen, tak se z durability předmětu odečte dvojnásobek kvality bloku.

Do čtvrté kategorie patří předměty, které může hráč zkonzumovat. Tyto předměty pak mají specifický efekt, který nějakým způsobem ovlivní hráče. V tuto chvíli se ve hře nachází dva takové předměty - kuře a steak, které po konzumaci hráči zvýší počet životů o určitou hodnotu.

U každého předmětu, který je možné vyrobit přes hráčův inventář, je možné nastavit libovolný počet materiálů a jejich množství, které je zapotřebí pro výrobu předmětu.



Obrázek 6: Inventář

3.8 Inventář

Inventář ve hře poskytuje hráči místo, kam si může uskladňovat všechny předměty, které se ve hře nacházejí.

Inventář je rozdělen do dvaceti políček (míst) pro uskladnění předmětu. Pět políček je viditelných po celou dobru hraní a měly by sloužit jako místo pro nejpoužívanější předměty. Zbytek políček je hráči zpřístupněno až po otevření inventáře.

V inventáři se vždy nachází jediný aktivní předmět. To je předmět, který má hráč zrovna v ruce. Z toho důvodu musí pro každý předmět existovat jeho fyzický model, který se hráči v ruce zobrazí, pokud je předmět aktivní.

Aktivní předmět lze jednoduše vybírat kliknutím na políčko v inventáři. Po aktivování předmětu se zobrazí jeho základní informace. Základní informace obsahuje ikonu předmětu, název, krátký popis a durabilitu.

Další možností inventáře je drag and drop funkce[4], která hráči umožňuje si inventář třídit podle vlastního uvážení. Tato funkce slouží hlavně pro výběr předmětů, které chce mít hráč v prvních pěti políčkách, a tak je mít na dosah po celou dobu hraní.

Dále inventář obsahuje možnost výroby předmětů.

Okénko výroby předmětů obsahuje seznam všech předmětů, které lze vyrobit. Po kliknutí na jeden předmět z nabídky se hráči zobrazí informace o tom, jaké materiály hráč pro výrobu potřebuje, název a ikonu předmětu.

Předměty vkládané do inventáře se liší parametrem *Stackable*. Tento parametr určuje, jestli se víc kusů daného předmětu může skládat na sebe, na jedno políčko.

[4]drag and drop funkce, umožňuje kliknutím a tažením myši přesouvat elementy

Obecně lze na jedno políčko skládat předměty z první a ze čtvrté kategorie.

Poslední užitečnou funkcí je možnost jakýkoliv předmět odstranit z inventáře pryč. Odstraněný předmět se pak objeví před hráčem na zemi a hráč si jej pak může znovu vzít.

3.9 Boj

Bojový systém je implementován velice jednoduše. Hráč může útočit s jakýmkoliv předmětem v ruce, nebo pouze rukama.

Pokud má hráč aktivní předmět ze třetí kategorie a bude útočit na nějakou jednotku, tak poškození, které bude dané jednotce způsobováno je přesně dáno nastavením parametru *Damage* daného předmětu. Pokud má hráč aktivní jakýkoliv jiný předmět, bude jednotce způsobovat vždy stejné poškození, které je nastavené na dvacet jednotek životů.

Hráč může útočit pouze na objekty, které spadají do vrstvy *Enemies* a pouze na určitou vzdálenost.

Při kliknutí levým tlačítkem myši se ze středu kamery vystřelí paprsek o určité délce (délka, na kterou může hráč útočit), nastavený tak, aby kolidoval jen s objekty patřící do vrstvy *Enemies*. Prvnímu objektu, do kterého paprsek narazil se zavolá funkce *TakeDamage*.

Funkce *TakeDamage* sníží aktuální hodnotu životů jednotce. Pokud hodnota životů bude nižší nebo rovna nule, zavolá se funkce *Death*.

3.10 Jednotky

Jediné jednotky, které se ve hře nachází jsou zvířata.

Jednotky jsou rozdělené na dva typy:

1. Pasivní
2. Aktivní

Pasivní jednotky mají jen náhodný pohyb po světě. Pokud na ně hráč zaútočí, tak před ním začnou prchat, dokud bude hráč v blízkosti. Aktivní jednotky mají také náhodný pohyb po světě, ale pokud se hráč ocitne v dostatečné blízkosti, tak na něj samy zaútočí. U každé jednotky rozlišujeme některé parametry:

- Rychlost pohybu
- Rychlost útěku
- Schopnost překonávat překážky
- Počet životů
- Velikost poškození (jen u aktivních jednotek)

Každá jednotka po své smrti zanechá na zemi nějaký předmět. V případě pasivních jednotek je to předmět ze čtvrté kategorie, v případě aktivních je to předmět ze druhé kategorie.

3.10.1 Útok jednotek

Útok aktivních jednotek je založený jen na vzdálenosti od hráče. Všechny instance aktivních jednotek si udržující referenci na pozici hráče a pokud se hráč ocitne v určité vzdálenost tak na něj jednotky zaútočí.

V první fázi útoku se jednotky snaží dostat až ke hráči, v této fázi se jim zdvojnásobí rychlost pohybu. Pokud hráč bude od jednotky pořád v určité vzdálenosti, tak jednotka zůstane v této fázi, kdy se snaží dohonit hráče. Pokud se hráč dostane dál, než je tato vzdálenost, tak se jednotka vrátí do pasivní fáze a rychlost pohybu se vrátí do normálu.

Druhá fáze útoku probíhá, pokud je jednotka bezprostředně blízko hráči. V tomto případě se jednotka zastaví a začne na hráče útočit. Samotné útoky probíhají v určitém intervalu. Jednotka bude útočit, dokud hráč bude v bezprostřední blízkosti a je naživu. Pokud se hráč vzdálí, tak se jednotka přepne zpátky do první fáze.

3.10.2 Pohyb jednotek

Samotný pohyb jednotek byl na implementaci nejsložitější. Aby se mohly jednotky pohybovat co nejvíc přirozeně, musel jsem implementovat nějaký pathfinding[5]. Protože ale tato práce nebyla na tuto problematiku zaměřena rozhodl jsem se, že použiji už hotový systém.

3.10.2.1 Unity Navigation

Samotné Unity nabízí svůj pathfinding systém, se kterým jsem už měl předchozí zkušenost, proto to byla moje první volba. Bohužel jsem rychle narazil na problém, kvůli kterému jsem se musel poohlédnout po něčem jiném.

Většina pathfinding systémů funguje tak, že objekty ve scéně (podlahy, schody a podobně) označíme nějakým parametrem *Walkable*. Pathfinding systém pak projde všechny objekt ve scéně a z objektů s parametrem *Walkable* si vytvoří navigační mapu, na které pak bude pro agenty[6] počítat a hledat cesty.

Tato operace se provede před samotným spuštěním hry, aby měl pathfinding systém tuto mapu předem nachystanou a mohl ji používat. Tento způsob funguje bezchybně u statických scén, kdy se povrch světa nemění, a tudíž navigační mapa bude vždy odpovídat reálné struktuře scény.

Problém ale nastává, pokud se scéna, jako v mém případě, neustále generuje nebo mění. V takovém případě pak nelze navigační mapu generovat před spuštěním ale musí se generovat průběžně při hraní, aby mapa odpovídala skutečnosti.

Unity pathfinding systém je samozřejmě pro tady ten případ připraven a umožňuje navigační mapu aktualizovat v průběhu hraní. V případě Unity pathfinding systému je tato operace až moc náročná a celou hru každé generování této mapy zaseknulo.

[5] jako pathfinding se označuje systém pro vyhledávání cest v prostoru
[6] agenti jsou objekty, které se pohybují po navigační mapě

3.10.2.2 A* Pathfinding project

Tento problém vyřešil hned další systém, který jsem vyzkoušel. A* Pathfinding project disponuje řadou variant jak tuto navigační mapu generovat. Pro můj případ bylo nejvýhodnější používat takzvaný *Grid Graph*.

Grid Graph řešení spočívá v tom, že je kolem hráče vytvořený obrovský čtverec, a pro jakýkoliv objekt, který se bude nacházet v tomto čtverci bude vypočítána navigační mapa.

Tohle řešení má mimo jiné výhodu v tom, že nemusíme každý objekt označovat, že pro něj chceme počítat navigační mapu. Při použití *Grid Graphu* tedy stačí posunovat tento čtverec zároveň s hráčem a v nějakém krátkém intervalu aktualizovat navigační mapu. Tento proces se ukázal jako ideální. Počítání navigační mapy, i přes to, že byla aktualizována každou sekundu, nemělo žádný dopad na běh celé hry.

A* Pathfinding project obsahuje spoustu ukázkových scén pro různé případy použití a také spoustu hotových skriptů, které vám usnadní práci. Jeden takový příklad byl právě skript, který se stará o to, aby hráč, který se pohybuje po světě, byl vždycky uprostřed výše zmiňovaného čtverce, a s každým posunutím čtverce pak přepočítá navigační mapu.

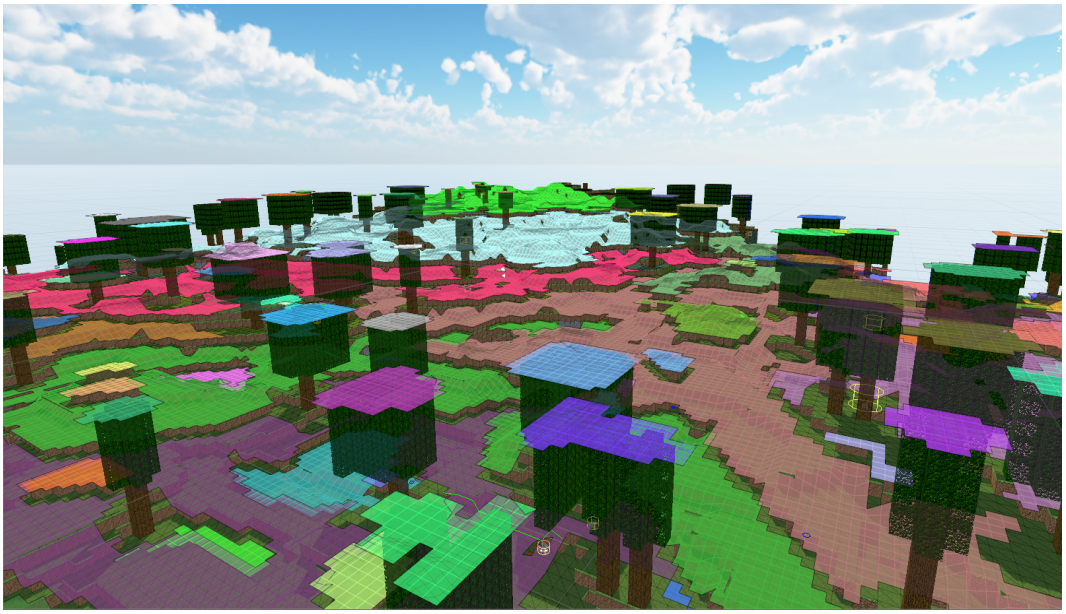
Tohle řešení přesně pasovalo na můj problém, proto už stačilo jen správně nastavit parametry, aby vygenerovaná navigační mapa správně kopírovala povrch světa a systém byl připraven na použití.

Na agenty pak už stačilo vložit další z hotových skriptů, kterému se nastavovala pozice kam se má agent dostat, a pohyb jednotek byl připraven. Jediná nevýhoda A* Pathfinding project, konkrétně verze dostupné zdarma, je to, že v této verzi není dostupný jejich *Local Avoidance* systém, který se stará o to, aby se jednotliví agenti mezi sebou vyhýbali. Tento problém jsem musel vyřešit vlastními skripty, ale i přes mou snahu se ve hře může někdy stát, že bude jedna jednotka chodit po jiné.

Náhodný pohyb po světě je vyřešen tak, že se při vytvoření jednotky vygeneruje náhodná pozice, která je však omezena určitou vzdáleností, a tato pozice se předá skriptu, který se stará o pohyb po navigační mapě.

Dále se kontroluje, jestli se jednotka nezasekla někde v terénu. Při náhodném pohybu se testuje podmínka, jestli se jednotka nachází dvě sekundy pořád na stejném bodě, pokud se tato podmínka splní jednotce se vygeneruje jiná náhodná pozice.

Stejně tak to funguje i při dosažení cíle. Pokud se jednotka nachází dostatečně blízko zadané pozice, tak se vygeneruje nová náhodná pozice. V případě útoku u aktivních jednotek se skriptu, který se stará o pohyb předá hráčova pozice a zvýší se rychlost. Stejně jako u náhodného pohybu se pak testuje dosažení cíle a zaseknutí v terénu.



Obrázek 7: Vygenerovaná navigační mapa

4 Minecraft vs. Mycraft

V této kapitole budou popsány největší rozdíly mezi originální hrou Minecraft a mou verzí, kterou jsem pracovně pojmenoval Mycraft.

Generování světa
: Nejznatelnější rozdíl je v generování a celkové struktuře světa. Zatímco v mé verzi jsem použil základní Perlin noise, který jsem upravoval jen pro generování stromů a jeskyní, a tak celá krajina vypadá jednotvárně, v Minecraftu tento systém prošel několika lety vývoje, proto v současné verzi, je generování světa úplně na jiné úrovni a svět vypadá opravdu nádherně. V průběhu generování se mění jednotlivé biomy, takže ve hře můžeme narazit na kopcovitou krajinu plnou stromů, zasněžené kopce, písečné pouště takřka bez kopců, různé vodopády anebo taky moře.

Výroba předmětů
: Výroba předmětů je další obrovský rozdíl. V Minecraftu je tento systém daleko propracovanější. Zatímco v mé verzi si jen vybíráte ze seznamu možných předmětů a pak už stačí mít jen potřebné materiály a předmět si jednoduše vyrobíte, v Minecraftu má každý předmět svůj předepsaný tvar, do kterého musíte potřebné suroviny v příslušném okénku poskládat. Zároveň v samotné hře Minecraft se nikde nedozvíte, jaké předměty můžete vyrábět a jak je vyrobit. Musíte na to buď přijít sám, nebo se podívat do návodu na internetu.

Rozmanitost předmětů

Viditelný rozdíl je také samozřejmě v rozmanitosti předmětů. Tento rozdíl jde také ruku v ruce s rozmanitosti materiálů, které lze ve světě těžit. Tento rozdíl ale není nijak závažný, protože má verze hry je připravena na případné rozšíření jak o nové materiály, tak o nové předměty.

Bojový systém

Bojové systémy obou her jsou velice podobné. Jediným velkým rozdílem je to, že v Minecraftu máte možnost vyrobit si luk, a tak není boj omezen jen na boj z blízka.

Střídání dne a noci

Tato mechanika, která oproti Minecraftu v mé verzi chybí, je zajímavý prvek, který samotný boj o přežití posouvá ještě na další úroveň. S příchodem noci v Minecraftu se totiž všude na světě začnou objevovat různí nepřátelé, kteří na hráče budou neustále útočit. Zároveň je ve hře tma zpracována tak, že bez louče nebo jiného zdroje světla nevidíte ani na krok.

Hlad

Další mechanika, kterou jsem v implementaci mé verze hry vynechal.
V Minecraftu hráč musí průběžně konzumovat nějaké jídlo, aby udržoval hladinu hladu na únosné úrovni. Pokud hlad přesáhne určitou úroveň, začnou se hráči ztrácet životy a může zemřít. Pokud se hráč nají a úroveň hladu sníží na dostatečný počet, tak životy začnou postupně přibývat. Já jsem tuto mechaniku upravil tak, že konzumace jídla doplňuje životy přímo. Jediná věc, která hráči může životy snížit jsou útočící jednotky.

5 Jak hrát

V hlavním menu, které se zobrazí po spuštění hry, si vyberete již rozehraný svět nebo napíšete jméno nového světa a následně kliknete na tlačítko Play. Pokud načítání světa zabere více než pár sekund, zobrazí se načítací obrazovka, kde můžete vidět, jak se generuje část světa.

Po načtení všech potřebných částí světa se objeví objekt hráče a vy můžete začít hrát.

Ovládání hráče je jednoduché, po světě se pohybujete pomocí šipek a kamera se otáčí pomocí pohybu myši.

Levým tlačítkem myši můžete těžit bloky nebo útočit na jednotky. Musíte však být v dostatečné blízkosti.

Pravým tlačítkem myši umísťujete bloky z inventáře zpět do světa a konzumujete jídlo.

Písmenem *E* na klávesnici můžete otevřít inventář s okénkem pro tvorbu předmětů.

V inventáři pak můžete pomocí kliknutí levého tlačítka myši a táhnutí přesouvat předměty. Jednoduchým kliknutím na předmět pak zvolíte předmět jako aktivní.

Poslední aktivní tlačítko je klávesa *ESC* na klávesnici, kterou vyvoláte nabídku, ve které se můžete přesunout do menu nebo hru úplně ukončit. Při přesunu do menu nebo při ukončení hry, se hra automaticky uloží a vy se můžete k rozehranému světu vrátit později.

Obrázek 8: Hlavní menu

Závěr

Náplní diplomové práce bylo vytvořit odlehčenou verzi hry Minecraft v Unity3D. Hra měla obsahovat modely vytvořené pomocí programu Blender, herní mechaniky jako procedurální generování světa, těžbu, vytváření předmětů, boj o přežití a bojový systém. Architektura hry měla umožňovat případné rozšíření hry. Součástí práce mělo být srovnání s originální hrou Minecraft. Všechny tyto požadavky byly splněny. Generovaný svět měl být původně více rozmanitější a obsahovat různé biomy, ale při implementaci jsem zjistil, že tato problematika je až moc náročná a musel jsem od toho opustit abych zvládl dokončit celou hru.

V rámci diplomové práce jsem se naučil základní práci s modelovacím softwarem Blender a tvorbu textur v Gimpu. Dál jsem si prohloubil znalosti při práci s Unity3D a osvojil jsem si praktiky procedurálního generování.

Text diplomové práce se převážně zaměřuje na implementaci jednotlivých herních mechanizmů, popis použitých technologií a rámcové srovnání s originální hrou Minecraft.

Conclusions

The goal of this thesis was to create light version of Minecraft game using Unity3D. The game was supposed to have 3D models created using Blender, implemented game mechanics like procedural generated world, mining, crafting, survival system and combat system. The game was supposed to be ready for future extension. Thesis was supposed to contain comparison with original game Minecraft. All of these requirements were met. World generation was supposed to be more diverse, but during implementation I found this problem is too difficult to solve, so I decided to not to implement it so I can finish rest of the game.

I have learned basics of 3D modeling in Blender and texture painting using Gimp. I have improved my skills working with Unity3D and I have also learned how procedural generation works.

Text of this thesis describe implementation process of all game mechanics, text also contains list of used technologies and comparison with original Minecraft game.

A Obsah přiloženého CD

Na přiloženém CD se nachází dvě složky a dva soubory.

`MyCraft.exe`
: Zkompilovaný soubor, který slouží pro spuštění hry

`UnityHubSetup.exe`
: Instalační soubor nástroje Unity Hub. Pomocí tohoto nástroje je potřeba nainstalovat Unity 3D ve verzi 2019.1.0f2, se kterou pak lze zkompilovat hru ze zdrojových souborů.

`mycraft/`
: Tato složka obsahuje zdrojové soubory vytvořené hry.

`doc/`
: Složka obsahuje text diplomové práce ve formátu PDF. Ve složce se také nachází všechny soubory, které jsou potřebné pro vytvoření tohoto souboru.

Literatura

[1] UNITY TECHNOLOGIES. Unity for all. Unity [online]. [cit. 2019-04-22]. Dostupné z: https://unity3d.com/

[2] MINECRAFT. IT'S A GAME ABOUT PLACING BLOCKS AND GOING ON ADVENTURES [online]. [cit. 2019-04-22]. Dostupné z: https://www.minecraft.net/en-us/what-is-minecraft/

[3] MINECRAFT WIKI. Official Minecraft Wiki, [online]. [cit. 2019-04-22]. Dostupné z: https://minecraft.gamepedia.com/Item

[4] MOJANG. YES, WE'RE BEING BOUGHT BY MICROSOFT [online]. [cit. 2019-04-22]. Dostupné z: https://mojang.com/2014/09/yes-were-being-bought-by-microsoft/

[5] A* PATHFINDING PROJECT. Lightning fast pathfinding for Unity3D. [online]. [cit. 2019-04-22]. Dostupné z: https://www.arongranberg.com/astar/front

[6] BLENDER. Blender - Free & Open Source 3D creation software. Blender [online]. [cit. 2019-04-22]. Dostupné z: https://www.blender.org/

[7] GIMP. The Free & Open Source Image Editor [online]. [cit. 2019-04-22]. Dostupné z: https://www.gimp.org/

[8] AURACITY. Audacity - free, open source, cross-platform audio software. Audacity team [online]. [cit. 2019-04-22]. Dostupné z: https://www.audacityteam.org/

[9] KHAN ACADEMY. Perlin noise [online]. [cit. 2019-04-22]. Dostupné z: https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise

[10] UNITY SCRIPTING REFERENCE. This section of the documentation contains details of the scripting API that Unity provides. [online]. [cit. 2019-04-22]. Dostupné z: https://docs.unity3d.com/ScriptReference/

[11] 25 'MINECRAFT' CREATIONS THAT WILL BLOW YOUR FLIPPIN' MIND. [online]. [cit. 2019-04-22]. Dostupné z: https://mashable.com/2013/02/13/amazing-minecraft-creations/?europe=true

[12] 16-BIT CALCULATOR + DOWNLOAD - MINECRAFT. [online]. [cit. 2019-04-22]. Dostupné z: https://www.youtube.com/watch?v=ygwQJPPO7-E