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

SURVIVAL HRA V PROCEDURÁLNÍM SVĚTĚ

SURVIVAL GAME IN PROCEDURALLY GENERATED WORLD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LUBOŠ MACHÁČEK**
AUTHOR

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

BRNO 2017

Abstrakt

Projekt je zaměřený na vývoj 3D počítačové hry využívající procedurální generování pro tvorbu herního světa. Součástí je i herní engine, který danou hru pohání. Při vývoji jsou využívané různé experimentální přístupy a techniky, a cílem je odzkoušet jejich použitelnost v praxi.

Abstract

Project is about developing a 3D computer game using procedural generation for game world creation. The development includes a game engine which powers the game. Various experimental approaches and mechanics are used during the development to find out their usability in practice.

Klíčová slova

hra, přežití, procedurální generování, semínko, herní engine, 3d, odložené stínování, fyzikální systém, grafika, experiment, prostorová paměť, prostorová orientace

Keywords

game, survival, procedural generation, seed, game engine, 3d, deferred shading, physics engine, graphics, experiment, spatial memory, spatial orientation

Citace

MACHÁČEK, Luboš. *Survival hra v procedurálním světě*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Milet Tomáš.

Survival hra v procedurálním světě

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.

.....
Luboš Macháček
15. května 2017

Poděkování

Rád bych vyslovil poděkování vedoucímu práce Ing. Tomáši Miletovi za jeho vstřícný přístup při vedení práce.

Obsah

Kapitola 1

Úvod

Tento projekt slouží jako vstupní brána do vývoje nezávislých her (indie games) bez ohledu na míru úspěšnosti.

1.1 Vize

Původním cílem bylo vytvořit plnohodnotnou fantasy hru, ve které se hráč vžije do role bojovníka či kouzelníka a bude prozkoumávat temná podzemí, nalézat poklady, vylepšovat svoje vybavení, zdokonalovat své schopnosti a snažit se přežít v nehostinném prostředí, které by každou hru vypadalo jinak. Toto všechno se však v rámci bakalářského projektu ukázalo být jako nadlidský výkon, zvláště pro aktivně pracujícího člověka, který ještě ke všemu s *OpenGL* teprve začíná. Naučit se principy a techniky *OpenGL*, vytvořit vlastní herní engine a sestavit herní mechanismy společně s vlastní grafikou zabralo nemálo času, a tak bylo nutné cíle během vývoje trochu pozměnit...

Novým jasnějším cílem bylo vytvořit jednoduchou fantasy hru, ve které hráč navštěvuje neznámá podzemí a sbírá magické krystaly. Toulku po podzemí mu znepříjemňují určité nástrahy uvedené dále.

1.2 Realizace

Projekt je rozdělen do tří základních částí: herní engine, procedurální generátor a výsledná hra. Na zmíněné části se zaměřují jednotlivé kapitoly této práce.

Celý projekt je implementován pomocí objektově orientovaného programovacího jazyka *C++* verze 14 a jsou využívány některé open-source knihovny, které řeší určitou problematiku nad rámec tohoto projektu. Díky multiplatformnosti těchto knihoven lze projekt přeložit na všech běžných operačních systémech: *GNU\Linux*, *Microsoft Windows* a *Mac OS*. Pro úspěšný překlad musí být splněny další požadavky těchto externích knihoven.

Grafický kontext je realizován pomocí *OpenGL* verze 3.3. Výslednou aplikaci lze tedy úspěšně spustit pouze na zařízení, jehož grafická karta podporuje danou verzi *OpenGL* a obsahuje potřebné ovladače.

Pro přípravu překladových zdrojových souborů slouží aplikace *CMake* využívající upravený konfigurační soubor vypůjčený ze školního projektu *FitGL*.

Kapitola 2

Herní engine

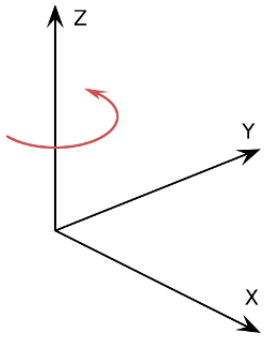
Místo počítačové či konzolové hry je možné si představit kupříkladu auto, kde herní engine představuje motor daného auta a fyzikální zákony okolního světa. Jedná se o jádro aplikace, které vytváří pro uživatele (programátora/vývojáře) určitou úroveň abstrakce a zároveň představuje nástroj šetřící čas. Uživatel se nemusí zaobírat implementováním běžné funkčnosti, ale akorát využívá nástroje a možnosti daného enginu. Profesionálnější herní enginy většinou umožňují uživateli jednoduché a rychlé rozšíření sady nástrojů vlastním kódem, protože někteří vývojáři mohou mít i své specifické požadavky.

2.1 Základní informace

Hojně se upřednostňuje vyšší výkon před pohodlností pro uživatele, což znamená, že většinu optimalizačních nástrojů a pravidel si uživatel nastavuje resp. bude nastavovat sám ručně – nebudou existovat žádné automatické nástroje pro inteligentní optimalizaci paměťové náročnosti a výkonu. Dále se upřednostňuje větší výpočetní výkon na úkor volné operační paměti.

Problematika enginu je rozdělena mezi různé moduly tvořící jeho komponenty. Každý modul obstarává funkcionalitu něčeho jiného a vzájemně spolupracuje s některými ostatními moduly. Dále se využívají tzv. prvky, které slouží k realizaci samotné *OpenGL* scény a herní logiky v rámci hlavního modulu *Aplikace*.

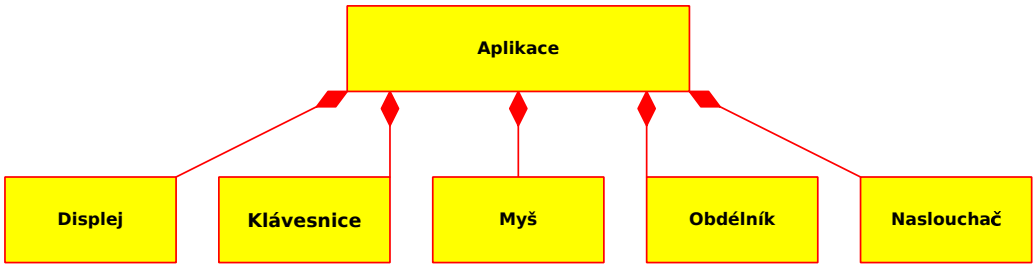
Engine využívá souřadnicový systém pravé ruky otočený tak, že osa Z směřuje vzhůru (viz obrázek 2.1).



Obrázek 2.1: souřadnicový systém enginu

2.2 Moduly

Moduly jsou implementovány jako třídy programovacího jazyka a slouží k rozdělení složitosti enginu do menších celků. Vztahy mezi moduly jsou zobrazeny na obrázku 2.2. Programátor pracuje přímo pouze s hlavním modulem *Aplikace*, ostatní lze akorát nastavovat nepřímo prostřednictvím operací hlavního modulu a konfiguračních dat. Vzhledem k tomu, že je tento engine prozatím použit pouze v tomto projektu, tak jsou konfigurační možnosti omezené jen na nastavení parametrů okna a pohybových vstupních zařízení.



Obrázek 2.2: hierarchie modulů

2.2.1 Aplikace (Application)

Jedná se o modul nejvyšší úrovně, který všechno zastřešuje a umožňuje uživateli propojit engine s jeho vlastní aplikací. Propojení je realizováno děděním třídy daného modulu.

Samotný engine má dvě základní počáteční fáze:

1. inicializační – zahrnuje uživatelskou konfiguraci, inicializaci všech ostatních modulů a vytvoření požadovaných shaderů;

2. spouštěcí – zahrnuje vykonání počátečního uživatelského kódu a spuštění hlavní smyčky aplikace.

Uživatelská konfigurace a počáteční kód se vkládá pomocí přetěžení specifických funkcí volaných na počátku jednotlivých fází. Další uživatelský kód se implementuje již pouze pomocí procedur.

Procedura představuje členskou funkci vlastní definované třídy a její obsah není nijak omezen. Pro zjednodušení vytváření procedur (objekt pro danou funkci) se využívá pomocných maker programovacího jazyka. Modul uchovává frontu uživatelem vložených vytvořených procedur, které se postupně zpracovávají. Myšlenka je taková, že v počátečním kódu se vloží prvotní procedura a další jsou vkládány prostřednictvím zpracovávaných procedur. Každým cyklem aplikace se zpracuje jedna procedura a nepokračuje se dokud není plně vykonána. To znamená, že jakoukoliv složitou problematiku, která se zpracovává delší dobu (např. načítání), je vhodné rozdělit do více procedur, aby bylo možné na obrazovku vykreslovat nějaký pokrok.

Pouze tento modul nabízí funkce pro úpravu konfiguračních dat, které jsou sdíleny napříč všemi moduly. Konfigurační data mají univerzální podobu ve formě dvojice řetězců: klíč a hodnota. Pokud je na nějakou hodnotu konfiguračních dat odkazováno v každém cyklu, tak se ukládá do soukromé pomocné proměnné, aby se zamezilo zbytečnému opakovanému převádění na jiný datový typ.

Další důležitou vlastností tohoto modulu je vytváření a aplikování/přepínání kontextů, což představuje prvek nejvyšší úrovně a nemá nic společného s *OpenGL* kontextem. Více podrobností o prvcích lze zjistit v kapitole 2.3. Samozřejmostí je i přístup k vytvořeným kontextům a uchovávání odkazu na právě aktivní kontext, který může být ve stejném čase vždy jen jeden.

Pomocí tohoto modulu je možné propojovat události vstupních zařízení s akcemi (veřejné členské funkce) objektu představující další typ dostupného prvku. Objekty jsou podrobně popsány v kapitole 2.3.6.

2.2.2 Displej (Display)

Tento modul obstarává vytváření okna aplikace včetně *OpenGL* kontextu a nastavování základních *OpenGL* parametrů. Taktéž propojuje obslužné funkce s událostmi, které mohou být vyvolány daným oknem, např. změna velikosti... Další důležitou vlastností je přepínání vykreslovacích bufferů, což je nutné provádět na konci vykreslovací fáze každého cyklu aplikace – souvisí s technikou *Double Buffering* a samotnou funkčnost zajišťuje knihovna *GLFW*.

Vykreslovací buffery jsou k dispozici dva:

- přední (front) – obsah tohoto bufferu se zobrazuje na obrazovce,
- zadní (back) – do tohoto bufferu se na pozadí v průběhu cyklu vykresluje.

Při každém přepnutí dojde k záměně zmíněných bufferů – zadní se stane předním a obráceně. Monitor tedy obnoví obraz až je veškeré vykreslování daného cyklu dokončené. Neznamená to však, že monitor stihne vykreslit celou obrazovku než přijde požadavek na další vykreslení obrazu – to řeší vertikální synchronizace (V-SYNC) [2].

2.2.3 Klávesnice (Keyboard)

Modul primárně propojuje obslužné funkce s událostmi vyvolanými klávesnicí, což představuje stisk a uvolnění klávesy. Samozřejmostí je možnost zapnutí/vypnutí obsluhy těchto událostí. Dále uchovává mapování uživatelských funkcí (vlastní definovaná funkcionalita) na stisk či uvolnění specifické klávesy. Pro jednu klávesu a každý její stav může existovat pouze jedna uživatelská funkce.

2.2.4 Myš (Pointer)

Modul primárně propojuje obslužné funkce s událostmi vyvolanými myší, což představuje pohyb zařízení a stisk/uvolnění tlačítka. Dále uchovává mapování uživatelských funkcí na stisk či uvolnění specifického tlačítka nebo pohyb zařízení. Pro jedno tlačítko a každý její stav může existovat pouze jedna uživatelská funkce. Stejné pravidlo platí i pro pohyb.

Jsou k dispozici tři volitelné režimy:

- zapnutý – kurzor myši je normálně viditelný;
- skrytý – kurzor myši je skrytý, přesto však stále ovladatelný;
- vypnutý – kurzor myši je skrytý a automaticky centrován do středu obrazovky.

2.2.5 Obdélník (Quad)

Tento modul úzce souvisí s technikou *Deferred Shading*, kde je vykreslování rozděleno do několika fází, ve kterých se nevykresluje na obrazovku, ale do připravených textur, u kterých se nastavují nové parametry při každé změně rozlišení obrazu či velikosti okna. Se zmíněnou technikou se využívá *Multiple Render Targets (MRT)*, což umožňuje vykreslování do více textur zároveň – do každé textury jinou potřebnou informaci [3].

Využívané *MRT* se skládá z textury pro (v závorkách je počet přiřazených bitů pro každý bod):

- hloubku (24b) – depth,
- difůzní barvu (24b) a intenzitu odlesku (8b) – albedo + specular intensity,
- normálu (48b) – normal,
- světlo (48b) – emissive,
- odlesk (48b) – specular.

Není využívána textura pro uložení pozice 3D bodu, protože pozice je vypočítána z hloubkové textury uvnitř využívaného shaderu.

Každá vykreslovací fáze pracuje se svým vykreslovacím bufferem a příslušnými shadery. Pokud nepočítáme základní vykreslovací buffer *OpenGL* využívaný v poslední třetí vykreslovací fázi, tak jsou dohromady k dispozici dva, kde první se využívá při vykreslování geometrie (1. fáze) a druhý při vykreslování osvětlení (2. fáze).

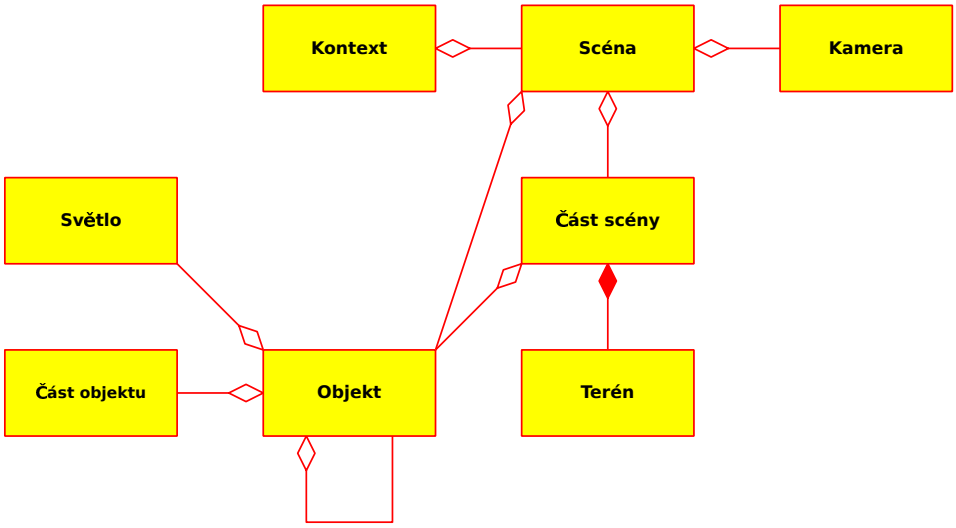
Při vykreslování geometrie se pro každý bod na obrazovce ukládá do příslušných textur: hloubka, barva s intenzitou odlesku a normála. V následující fázi se využívají již naplněné hloubkové a normálové textury, a pro každý bod na obrazovce se ukládá do příslušných textur: světlo a odlesk. Nyní jsou všechny potřebné textury naplněné a může se přejít k poslední fázi, ve které se využívá již standardní *OpenGL* vykreslovací buffer. Teď se již akorát na obrazovku vykreslí 2D obdélník s výslednými texturami a nastavenou hodnotou osvětlení obrazu.

2.2.6 Naslouchač (Listener)

Už samotný název napovídá, že se jedná o modul zastřešující všechny obslužné funkce, které jsou v ostatních modulech propojovány s požadovanými událostmi – zmíněné ostatní moduly totiž využívané obslužné funkce nevlastní. Uvnitř obslužných funkcí se potom zpracovávají vzniklé události a mohou se dále volat konkrétní namapované uživatelské funkce.

2.3 Prvky

Prvky jsou implementovány jako třídy programovacího jazyka a celkově slouží k realizaci obsahu *OpenGL* scény a herní logiky. Jsou uspořádány do hierarchické struktury a některé jsou i vzájemně propojeny – viz obrázek 2.3. Uživatel primárně pracuje s prvkem *Objekt*, který prostřednictvím dědičnosti a možnosti přetěžování funkcí nabízí širokou implementaci vlastního kódu. S ostatními prvky uživatel pracuje zpravidla nepřímo prostřednictvím operací nadřazených prvků nebo hlavního modulu.



Obrázek 2.3: hierarchie prvků

2.3.1 Kontext (Context)

Kontext představuje kontext aplikace, nikoliv *OpenGL* kontext, a je identifikován jednoznačným řetězcem. Tento prvek se v hierarchické struktuře nachází u samotného kořene. Slouží k tomu, aby bylo možné mít připraveno více *OpenGL* scén, mezi kterými půjde jednoduše přepínat bez nutnosti dalšího nahrávání do operační paměti.

Kontext se dělí na jednotlivé scény (nikoliv *OpenGL*) a nabízí funkce pro jejich vytváření či přepínání/aktivování. Samozřejmostí je přístup k vytvořených scénám a odkaz na právě aktivní scénu, která zatím může být ve stejném čase vždy jen jedna, tudíž zatím neexistuje možnost zobrazovat více scén zároveň, což by sloužilo pro realizaci plynulého přechodu.

Každý kontext dále uchovává vlastní sadu assetů, o kterých se lze více dozvědět v kapitole 2.4. Zmíněná sada se nahrává do paměti grafické karty při aktivování daného kontextu. Toto řešení není úplně optimální, protože se na aktivní scéně nemusí nacházet všechny assety daného kontextu, tudíž se značně redukuje dostupná paměť grafické karty. Samozřejmostí jsou operace pro přípravu assetů a nahrávání do operační paměti.

Aktivování kontextu je řetězová operace. Pokud v daném kontextu existuje nějaká aktivní scéna, tak se provede reaktivace dané scény, což způsobí další nahrávání potřebných dat do paměti grafické karty.

2.3.2 Scéna (Scene)

Scéna je součástí nějakého existujícího kontextu a dělí jej na menší samostatné celky. Je identifikovaná jednoznačným řetězcem. Z důvodu hlubšího optimalizování výkonu je scéna rozdělena ještě do menších částí, se kterými se již reálně pracuje v rámci vykreslování. Všechny části se musí připravit pomocí k tomu určených operací ve fázi vytváření scény, protože se jedná o vzájemně propojené statické prvky, které musí být definovány před vložením prvního objektu na scénu.

Každá scéna uchovává své existující kamery, které jsou k dané scéně vázané, a nabízí operace pro jejich vytváření či přepínání/aktivování. Samozřejmostí je přístup k vytvořeným kamerám a odkaz na právě aktivní kameru, která může být ve stejném čase vždy jen jedna.

OpenGL scéna se vykresluje v rámci aktivní kamery, na jejíž základě se vytváří pohledová matice.

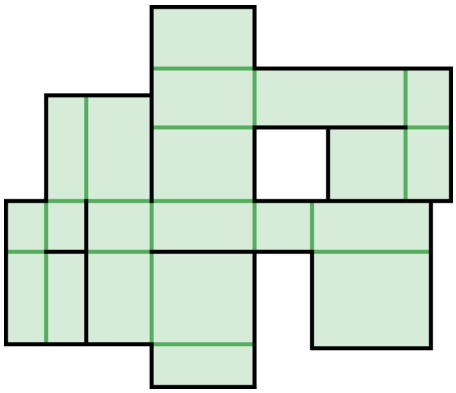
Scéna dále slouží pro uchovávání některých aktivních objektů, které pravidelně každý cyklus mění nebo mohou měnit svůj stav a nejsou dostupné skrz jiné aktivní objekty.

Aktivování scény (z kontextu) je řetězová operace. Pokud na dané scéně existuje nějaká aktivní kamera, tak se provede reaktivace dané kamery, což způsobí další nahrávání potřebných dat do paměti grafické karty.

2.3.3 Část scény (Chunk)

Tento prvek představuje nejmenší jednotku scény se kterou reálně pracují všechny objekty a kamery. Je ohraničený minimálním a maximálním 3D bodem, který tvoří neorientovaný pomyslný kvádr (AABB). Střed zmíněných dvou bodů taktéž definuje pozici v rámci scény, což se využívá pouze při vytváření terénu. Části scény se nesmí překrývat vyjma hran pomyslného ohraničujícího kvádru.

Překrývání hran je nutné pro přechod mezi propojenými částmi scény. Definování propojených částí zajišťuje uživatel při vytváření scény. Ohraničující kvádr má 6 stran, tudíž každým směrem může existovat pouze jedno vzájemné propojení s jinou částí scény. Není možné jedním směrem přejít do dvou různých částí a stejné platí obráceně. Obrázek 2.4 slouží pro lepší představu této problematiky. Nemá-li strana žádné propojení, tak přes ní nelze část scény opustit.



Obrázek 2.4: ukázka korektních částí scény ve 2D zobrazení (zelená čára představuje propojení)

Každá část scény uchovává prvek terénu a sadu neviditelných kolizních tvarů, které ve výsledku tvoří přístupný resp. nepřístupný prostor v rámci dané části. Dále vlastní svoji sadu všech potencionálně viditelných částí scény (visibility sets), která se využívá pro rychlejší a efektivnější vykreslování. Tato sada dříve zároveň definovala i to, které všechny terény se nahrály do paměti grafické karty při aktivování kamery. Z důvodu problematiky popsané níže byla tato funkcionalita vypnuta.

Další klíčovou vlastností rozdělení scény na části bylo to, že v paměti grafické karty se nacházela pouze 3D data terénů těch částí scény, které bylo možné potencionálně vidět z části, ve které se nacházela kamera. Při přechodu kamery do jiné části se do paměti grafické karty nahrála data terénů nově potencionálně viditelných částí a vymazala data těch terénů, jejichž části již nešlo potencionálně vidět. K tomu sloužily další dvě sady částí scény, které

už předem definovaly, které nové terény nahrát a které vymazat. Vzhledem k tomu, že se tato realizace v jednovláknové aplikaci ukázala jako neefektivní, protože při větším objemu nahrávaných dat docházelo ke chvilkovému viditelnému snížení snímkové frekvence, tak bylo od tohoto přístupu dočasně upuštěno a bylo to nahrazeno jiným přístupem, který nahrává data terénů všech existujících částí scény při aktivování příslušné scény. Nový přístup však může být velice problematický při rozsáhlých scénách, ve smyslu zaplnění paměti grafické karty.

Každá část scény dále obsahuje odkazy na objekty, které se v dané části nachází (dle středu objektu) a nebo do ní jakkoliv zasahují svým pomyslným ohraničujícím kvádrem (AABB). Pro tento účel jsou k dispozici pomocné operace, které vkládají/vyjímají objekty do/ze specifikovaných částí. Je důležité zmínit, že objekty si zase uchovávají zpětné odkazy na části scény.

2.3.4 Terén (Terrain)

Terén je vždy svázaný s nějakou částí scény, ve které se nachází, a má pouze definovanou pozici v rámci scény. Skládá se ze svých unikátní meshů, které jsou vytvářeny na základě nějakých modelů. Jedná se pouze o vizuální prvek, který pro prvky *Objekt* jednoduše neexistuje a nemohou s ním nijak kolidovat.

Při rozšiřování terénu se zkopírují meshe z nějakého modelu tak, že meshe mající stejný materiál se zkombinují do jednoho, aby se redukoval počet vykreslovacích volání. Díky tomu již sice není možné z rozšířeného terénu něco odebírat, ale jedná se o cílenou funkcionalitu, protože jako terén se bere všechno, co se po celou dobu existence scény již nemění. Samozřejmostí je možnost nastavit všem bodům kopírovaného meshe pozici a rotaci v rámci scény.

Výše zmíněná realizace je pouze experimentální, protože v ní dochází k nadměrné redundanci dat, ale na druhou stranu zase umožňuje, aby každá část scény obsahovala jinak vypadají terén.

2.3.5 Kamera (Camera)

Kamera je identifikovaná jednoznačným řetězcem a definovaná svojí rotací a pozicí v rámci scény, ve které se nachází. Při každém otočení kamery se přepočítávají směrové vektory. Dále má nastavené své specifické parametry, které upravují perspektivu a minimální/maximální vykreslovací vzdálenost.

Platná kamera se musí vždy nacházet v nějaké části scény, na jejíž základě se vykresluje okolí. K tomuto slouží uložený odkaz na danou část scény, který však nelze nastavovat ručně. Kamera tedy musí být připevněná na nějaký objekt, přesněji objekt rozšířený o možnost držet kameru. Objekt jako jediný má možnost dynamického přesunu mezi částmi scény a přeposílá kameře odkaz na aktuální část scény, ve které se nachází.

2.3.6 Objekt (Object)

Objekt je základní prvek scény a jako jediný se může v rámci ní pohybovat a přecházet mezi jednotlivými částmi. Je identifikovaný jednoznačným kladným číslem, které je možné buď nastavit ručně a nebo automaticky skrytým snižujícím se počítadlem. Není nijak kontrolováno, zda-li identifikátor není duplicitní.

Další jedinečnou vlastností je možnost kolidovat s jiným objektem či nepřístupným prostorem části scény. S tím souvisí zařazení do určité kolizní třídy (možno více zároveň), která

definuje, co s čím bude kolidovat. Kolizních tříd je dohromady k dispozici 32 a v případě nutnosti není problém rozšířit na 64. Kolizní třídy jsou pro daný objekt uloženy v jediné celočíselné proměnné reprezentující bitové pole – při porovnávání se tedy využívají rychlé bitové operace.

Objekt dále obsahuje klasifikátor (číslo), na základě něhož je uživatel potom schopen přetypovat objekt na nějakou svoji definovanou třídu, která dědí ze třídy objektu, a přistupovat ke svým členským funkcím a proměnným. Více se o této problematice lze dozvědět dále.

Každý objekt má definovanou pozici, rotaci (v eulerových úhlech) a měřítko v rámci scény. Výsledné rotační úhly se vždy normalizují do intervalu od -360°do 360°. Dále uživatel může nastavit tzv. translátor a rotátor, které určují, jaké dimenze (X, Y, Z) pozičního či rotačního vektoru není možné měnit. U rotátoru lze specifikovat i minimální či maximální hodnoty daných dimenzí.

Změna stavu v rámci scény je realizovatelná dvěma způsoby:

- okamžitým posunem, rotací či změnou měřítka;
- úpravou rychlostí posunu, rotace či změny měřítka v daném směru (zbytek obstarává fyzikální systém).

V případě využití okamžitých změn musí uživatel obstarat prostřednictvím speciálních operací dvě základní věci: uložení stavu objektu před změnami a vyhodnocení následků změn. Fyzikální systém tyto věci obstarává sám. Při vyhodnocování následků změn se dle potřeby vykonají některé následující automatické operace:

- kontrola a případný přechod mezi částmi scény,
- přepočítání ohraničujících tvarů a pomyslného ohraničujícího kvádru (AABB),
- kontrola a vyhodnocení kolizí.

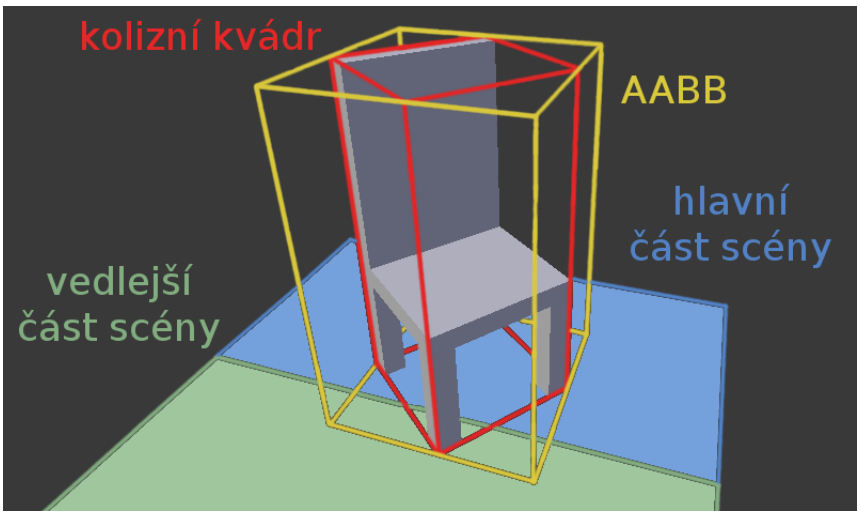
Pro potřeby fyzikálního systému jsou k dispozici následující základní operace:

- zrychlení – definuje se směr, zrychlení za sekundu a maximální rychlost;
- zpomalení – definuje se směr, zpomalení za sekundu a minimální rychlost;
- odstrčení – definuje se směr a okamžitá rychlost;
- zastavení.

Dále je možné nastavit tzv. uklidňovací vektory, které definují míru automatického zpomalování rychlosti posunu, rotace či změny měřítka. Pokud není nastaveno žádné uklidňování, tak si objekty uchovávají svoji stálou rychlost.

Platný objekt se vždy musí nacházet v nějaké části scény. Stále si uchovává odkaz na hlavní část scény, ve které se nachází jeho střed. Vedle toho si uchovává části scény, do kterých zasahuje svým pomyslným ohraničujícím kvádrem (AABB). Uživatel musí akorát definovat počáteční část scény a zbytek již obstarává engine. K dispozici je i možnost přiřazení počáteční části scény na základě pozice objektu, ale z důvodu optimalizace výkonu se toto neděje automaticky. Pro vkládání/vyjímaní do/z části scény jsou k dispozici pomocné operace v rámci objektu, scény a nebo její části.

Objekt může mít definované dva druhy ohraničujícího tvaru: viditelností tvar a kolizní tvar. Oba druhy mají stejné fyzické vlastnosti, ale odlišnou sémantiku. Kolizní tvar se využívá ve fyzikálním systému. Viditelnostní tvar se definuje u objektů jejichž vizuální část přesahuje kolizní tvar a nebo nemají kolizní tvar definovaný vůbec, a využívá se pouze při vykreslování. Pokud by objekt neměl definovaný ani jeden ohraničující tvar, tak by se vykresloval pouze v rámci části scény, ve které se nachází jeho střed, přestože by svojí vizuální podobou zasahoval i do jiných částí. Z důvodu optimalizování výkonu se viditelností tvar nepočítá automaticky na základě vizuálních meshů. Vybírat lze z následujících tvarů: koule, stojící válec, kvádr rotující okolo osy Z a orientovaná úsečka. Prozatím je implementována kontrola průniku kolizních tvarů jen pro některé kombinace.



Obrázek 2.5: ukázka objektu na scéně

Na základě všech ohraničujících tvarů je spravován minimální a maximální 3D bod, který tvoří pomyslný neorientovaný kvádr (AABB), který ohraničuje vše ostatní uvnitř. Tento pomyslný kvádr využívá i fyzikální systém při kontrolování kolizí. Předtím než se přejde ke kontrole průniku kolizních tvarů, tak se v první řadě nejdříve provede kontrola průniku pomyslných kvádrů, což je velice rychlá výpočetní operace. Pokud dojde k úspěšnému průniku pomyslných kvádrů, tak se pokračuje kolizními tvary, což už v závislosti na použitých tvarech může být náročnější výpočetní operace [1].

Objekt se může skládat z několika částí, které slouží k připevňování vizuálních meshů, a různých připevněných světel. K tomuto slouží univerzální připevňovací operace. Dále si může uchovávat odkazy na objekty, se kterými nějakým způsobem pracuje. Důležité je zmínit to, že scéna nemusí vědět o všech vložených objektech, tudíž existenci objektu primárně ovlivňují odkazy na objekty v rámci nějaké části scény či jiného objektu. Pokud se objekt nenachází v žádné části scény, tak nelze ani aktivovat.

Pro potřeby upevňování kamery existuje speciální varianta objektu, ve které jsou kameře automaticky přeposílány jakékoliv změny pozice, rotace a odkazu na hlavní část scény.

Uživatel si definuje vlastní typy objektů pomocí dědičnosti a přetěžení funkcí, které se volají při úspěšné kolizi nebo na konci vyhodnocení následků změn. Dalším nástrojem jsou tzv. činnosti, které představují vlastní definované funkce a lze je přidávat či odebírat ze seznamu aktivních činností. Aktivní činnosti jsou automaticky zpracovávány v každém cyklu

aplikace, pokud se jedná o aktivní objekt, pro který si scéna uchovává odkaz. Samozřejmostí je i možnost aktivace/deaktivace objektu.

2.3.7 Část objektu (Part)

Každá část je definovaná pozicí, rotací (v eulerových úhlech) a měřítkem v rámci daného objektu. Jedná se pouze o vizuální prvek. Nemá žádný jednoznačný identifikátor. Pro přístup k jednotlivým částem si tedy musí uživatel pamatovat jejich pořadí při vkládání. Další nevýhodou jest, že díky eulerovým úhlům není možné optimálně dynamicky měnit natočení, protože se vždy provádí postupné natáčení přes osy Z, Y a X.

Část objektu si uchovává sadu meshů, které odkazují na meshe modelů ze sdílné sady assetů v rámci kontextu. Nabízí vykreslovací operaci, která akorát volá vykreslovací operaci každého meshe.

2.3.8 Světlo (Light)

Každé světlo je definováno pozicí, rotací (v eulerových úhlech) a měřítkem v rámci daného objektu, bez kterého nemůže existovat. V současnosti je k dispozici akorát bodové světlo, u kterého se dále nastavuje jen barva a síla osvětlení.

Na základě typu světla je přiřazený odpovídající tvar (volume) ze sdílené sady assetů – momentálně musí obstarávat sám uživatel. Bodové světlo je tedy reprezentováno koulí. Fyzické tvary světel jsou využívány při vykreslování osvětlení pomocí *Deferred Shading* [5].

K dispozici je vykreslovací operace, která se stará o nastavení některých parametrů potřebného shaderu a volá vykreslovací operaci přiřazeného tvaru.

2.4 Assets

Sada assetů je vždy sdílená v rámci nějakého aplikačního kontextu. Uživatel se stará akorát o nahrávání modelů a tvarů, které si již samo obstarává vytváření textur a materiálů.

2.4.1 Textura (Texture)

Textura si uchovává v operační paměti data nahraného obrázku, včetně jeho rozměrů a počtu potřebných bytů pro pixel. Dále je specifikováno, o jaký typ textury se jedná: difúzní či odleskovou (specular). Při nahrávání je ošetřeno to, aby nevznikaly duplicitní textury – ve výsledku tedy různé materiály mohou odkazovat na stejné textury.

Jsou k dispozici potřebné operace pro nahrávání dat do paměti grafické karty (včetně uvolňování). Toto však uživatel sám nekoordinuje, ale obstarává to samotný engine.

U textur je vždy využíván *Trilinear filtering* – názorná ukázka na obrázku 2.6.

2.4.2 Materiál (Material)

Materiál si uchovává difúzní a odleskovou barvu, a odkaz na difúzní a odleskovou texturu. Při vytváření materiálů se dává pozor, aby nevznikaly zbytečné duplicity – ve výsledku tedy různé meshe mohou odkazovat na stejné materiály.

Obrázek 2.6: žádné filtrování (vlevo) vs Trilinear filtering (vpravo)

2.4.3 Mesh

Mesh si uchovává 3D body, které jsou definovány pozicí, normálou a 2D polohou v rámci textury. Dále si uchovává indexy, sloužící pro redukování počtu duplicitních 3D bodů v rámci jednoho meshe, a využívaný materiál.

Každý mesh si obstarává minimální a maximální 3D bod, který tvoří pomyslný ohraničující kvádr (AABB).

Pro úpravy bodů meshů slouží transformační operace, které nabízí: posun, rotace a převrácení (dle osy). Po poslední úpravě je potřeba zavolat ukončovací operaci, která se postará o přepočítání pomyslného ohraničujícího kvádru.

Nabízí potřebné operace pro nahrávání dat do paměti grafické karty (včetně uvolňování). Toto však uživatel sám nekoordinuje, ale obstarává to samotný engine.

K dispozici je vykreslovací operace obstarávající i nastavování potřebných parametrů ve využívaném shaderu a aktivování použitých textur.

2.4.4 Model

Model slouží pouze jako obal pro meshe a ulehčuje jejich předávání prvkům. Dále se využívá například při úpravách, aby se nemusel každý mesh upravovat jednotlivě. Kromě transformačních operací nabízí možnost svého duplikování.

Při nahrávání z externích souborů se vždy obsah ukládá do modelu.

2.4.5 Tvar (Volume)

Tvar má stejné vlastnosti jako mesh, s tím rozdílem, že si u každého 3D bodu uchovává pouze jeho pozici. Dále u něho nelze definovat žádný materiál. Stejně jako mesh nabízí vykreslovací operaci.

Každému světlu je vždy přiřazen nějaký tvar, který představuje jeho fyzický vzhled ve scéně.

2.5 Smyčka aplikace

Na konci počáteční spouštěcí fáze enginu se začne provádět neustále se opakující smyčka aplikace, která lze ukončit pouze pomocí speciální ukončovací operace v rámci hlavního modulu. Samotná smyčka se skládá z následující posloupnosti fází:

1. příprava,
2. aktualizace,
3. vykreslování,
4. odpočinek.

Každá fáze se skládá z několika kroků, které jsou podrobně popsány v následujících podkapitolách.

2.5.1 Příprava (Prepare)

Tato fáze slouží pouze pro výpočet počtu snímku za sekundu (FPS) a časového rozdílu zvaného *Delta Time*. Jelikož engine běží nezávisle na počtu snímků za sekundu, tak je potřeba zajistit, aby alespoň zdánlivě běžel stále stejnou rychlostí. K tomu slouží zmíněný *Delta Time*, který představuje uplynulý čas od počátku zpracování předchozího cyklu v sekundách. Při menší hodnotě jsou veškeré změny stavů prvků na scéně (např. pohyb či rotace) plynulé. Při větší hodnotě, což mimo jiné znamená i menší počet snímků za sekundu, jsou změny skokové, protože prvky musí dohnat svůj aktuální stav [4].

2.5.2 Aktualizace (Update)

Na počátku této fáze dochází ke zpracování vstupních událostí, které se během cyklu ukládají do fronty, a volání příslušných obslužných funkcí. Dále se vyjme a zpracuje jedna procedura pokud fronta nezpracovaných procedur není prázdná.

Pakliže není aktivovaný žádný kontext a scéna, tak se v této fázi již nic dalšího neděje. V opačném případě se projdou všechny aktivní objekty odkazované v rámci scény, zpracují se jejich aktivní činnosti a fyzikální systém provede potřebné změny a vyhodnotí následky změn.

2.5.3 Vykreslování (Render)

Vykreslování je možné pouze pokud existuje nějaký aktivní kontext, scéna a kamera. Nejprve se na základě kamery nastaví perspektivní a pohledová matice.

Jako první je na řadě vykreslování geometrie scény, pro které se aplikuje příslušný vykreslovací buffer a shader. Vezme se část scény ve které se nachází kamera a v rámci dané části se projdou všechny další potencionálně viditelné části scény. V každé části se projdou jednotlivé terény a části všech objektů, pro které jsou postupně nastavovány modelové matice a volány vykreslovací operace. Všechny nastavované matice jsou postupně předávány shaderu. Výsledek vykreslování je nyní uložený v hloubkové, normálové a barevné textuře.

Dále přichází na řadu vykreslování osvětlení, které má zase vlastní vykreslovací buffer a shader. Kromě požadovaných matic a pozice kamery, je nutné shaderu ještě předat velikost bodu na obrazovce, což se využívá při výpočtu pozice 3D bodu ve scéně. Nyní se projdou

jednotlivá světla všech potencionálně viditelných objektů. Pro každé světlo je zase nastavována potřebná matice a volána vykreslovací operace. Na základě pozice kamery se nastavuje vykreslování zadní nebo přední části fyzického tvaru osvětlení – jedná se o jediný případ, kdy se kamera může platně nacházet i uvnitř meshe. Při tomto vykreslování jsou využívány zmíněné dříve naplněné textury. Výsledek je uložen do světelné a odleskové textury.

V posledním kroku vykreslování se pouze aplikuje základní vykreslovací buffer *OpenGL* a s pomocí dalšího shaderu se vykreslí 2D obdélník, který má na sobě aplikovanou kombinaci barevné, světelné a odleskové textury, což ve výsledku tvoří požadovaný obraz. Při tomto vykreslování se využívá již ortogonální perspektivy a je potřeba pouze perspektivní matice.

Na úplném konci této fáze dojde k přepnutí vykreslovacích bufferů obrazovky, díky kterému se projeví změna na zobrazovacím zařízení.

2.5.4 Odpočinek (Rest)

Jelikož je vertikální synchronizace vypnuta, tak je potřeba ručně zajistit, aby aplikace zbytečně nepřetěžovala grafickou kartu. K tomu se využívá tzv. uzamykání počtu snímků za sekundu (FPS Lock), což je technika zaručující nějakou určitou minimální dobu zpracování každého cyklu. Ve výsledku tedy máme aplikaci, která nepřesáhne námi požadovanou hodnotu FPS. Momentálně není možné měnit uzamykání uživatelem a je v základu nastaveno na 120 FPS, i přestože většina běžných obrazovek není schopná zobrazit více jak 60 FPS.

Uzamykání FPS je implementováno pomocí stále se opakujícího cyklu, který běží tak dlouho, dokud čas zpracování cyklu nepřesáhne určitou hodnotu.

2.6 Externí knihovny

Engine využívá některé open-source multiplatformní knihovny, které se starají o funkcionalitu, která je specifická pro cílový operační systém nebo je nad rámec tohoto projektu. Zároveň není nutné vytvářet něco, co už dávno existuje a má to ulehčovat práci vývojářům.

Jsou využívány následující externí knihovny:

- *GLEW* (*OpenGL Extension Wrangler Library*) – načítání *OpenGL* rozšíření dostupných na cílové platformě;
- *GLFW* – API pro vytváření oken *OpenGL* aplikací a jejich kontextů, zachytávání a zpracování vstupních událostí či událostí operačního systému;
- *GLM* (*OpenGL Mathematics*) – funkce pro práci s vektory a maticemi;
- *ASSIMP* (*Open Asset Import Library*) – importování 3D modelů různých formátů.
- *stb_ image* – nahrávání obrázků ve formátu *PNG*.

2.6.1 GLFW

Ze začátku vývoje se *GLFW* tvářila jako ideální volba: umožňovala jednoduše vytvářet okna a zachytávat vstupní události; nic více nebylo pro tento engine potřeba. V pozdější fázi vývoje se však ukázalo, že pod některou distribucí *GNU\Linux* nefunguje zachytávání a zpracování vstupních událostí tak zcela správně. Během vývoje se narazilo na dva zásadní problémy:

- Pokud je klávesa klávesnice uvolněna po dlouhém stisku, tak *GLFW* nezaznamená pouze jednu událost: uvolnění klávesy; ale rovnou tři: uvolnění klávesy + stisknutí klávesy + uvolnění klávesy.
- Pokud nejsou zaznamenané vstupní události z *GLFW* přijaty a zpracovány po určitou dobu, např. při načítání hry, tak události poté přichází v nespecifikovaném pořadí a občas je místo stisku klávesy chybně posíláno uvolnění klávesy.

První problém šlo chytře eliminovat kódem, který si zaznamenává, která poslední klávesa je stisknuta po delší dobu a v případě uvolnění dané klávesy se po přijmutí první události zahodí veškeré další události od stejné klávesy, které přijdou pod 49 ms. Málokdo opakovaně stiskne stejnou klávesu v daném časovém intervalu. Rozhodně to není nemožné, ale při běžném hraní se hráč k této skutečnosti ani nedostane.

Druhý problém již nijak obejít nešlo a z důvodu nedostatku času již nebylo možné vyměnit knihovnu za jinou.

Kapitola 3

Procedurální generátor

Procedurální generování je technika, která umožňuje na základě definovaných pravidel a postupů vytvářet pseudonáhodná data. Vstupem většinou bývá tzv. semínko, což představuje nějakou hodnotu či skupinu hodnot. Stejné semínko zaručuje vždy stejné výsledky, jedná se tedy o deterministické chování. Výstupem může být prakticky cokoliv, ať už nějaký náhodně vytvořený herní svět nebo třeba jen různé textury...

Správně implementované procedurální generování značně ulehčuje čas vývojářům. Samotná implementace však zabere taktéž nemálo času a tudíž je potřeba si rozmyslet, zda-li se vůbec při určitém vývoji vyplatí. V některých případech se ale zase jedná o jediné řešení, pokud je požadavkem například vždy jiný herní svět, což není možné realizovat ručně. Důležité je taktéž zmínit to, že procedurální generování většinou nedosahuje kvalit ruční práce, kde se využívá i cit samotného vývojáře a ne pouze předem definovaná pravidla.

3.1 Základní informace

Procedurální generátor je vytvářen pouze pro potřeby tohoto projektu, tudíž není univerzální, a je úzce svázán se zmíněným herním enginem. Z časových důvodů nebyla prováděna hlubší optimalizace rychlosti generování. Na procesoru *AMD Phenom II X4 965* (pracovní frekvence 3,4 GHz) zabere kompletní generování v rámci tohoto projektu přibližně 2 sekundy. Generátor je implementován jako třída programovacího jazyka a jakákoliv funkčnost je realizována uvnitř dané třídy.

Tento generátor se stará o vytváření herní mapy a naplnění kontextu aplikace. Mapa se skládá ze stejně velkých buněk (čtverce), které jsou zarovnány do mřížky. Tento přístup nabízí jednodušší práci s mapou, protože se s ní dá pracovat jako s dvourozměrným polem, kde Y určuje pozici v rámci výšky mapy a X pozici v rámci její šířky. Šířka a výška tudíž představují počet buněk v dané dimenzi.

Na vstup generátoru jsou předány požadované rozměry mapy, kontext aplikace (s vytvořenou scénou) se kterou bude pracovat a vytvořené semínko. Není žádný výstup, který by se předával zpět aplikaci, protože veškeré změny jsou realizovány přímo v předaném kontextu.

Generování je rozděleno do následujících základních fází:

1. tvorba šablony (půdorys mapy),
2. tvorba mapy,
3. plnění kontextu.

Jednotlivé fáze se dělí ještě na další kroky, které jsou podrobně popsány v dalších kapitolách.

3.2 Semínko

Semínko je implementované jako třída programovacího jazyka a skládá se z kruhové posloupnosti čísel mající hodnotu od 1 do 100, což reprezentuje procenta udávající míru řádu a chaosu (větší hodnota = větší chaos). Posloupnost lze nastavit buď ručně a nebo automaticky pomocí generátoru pseudonáhodných čísel ze standardní *C++* knihovny (od verze 11).

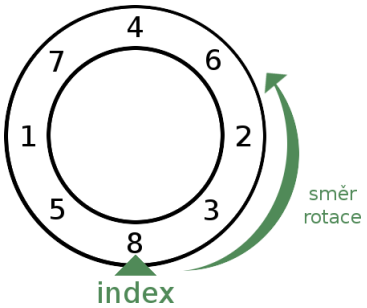
V případě ručního nastavování je semínku při vytváření jeho objektu předán nějaký řetězec. Z řetězce je pomocí střídajících se matematických funkcí pro sinus a cosinus, a následného převodu do intervalu od 1 do 100, vytvořená požadovaná posloupnost čísel. Čím je delší řetězec, tím je různorodější výsledná posloupnost. Dvě matematické funkce jsou použity proto, aby řetězec obsahující stejné znaky vytvářel proměnlivou posloupnost.

Při automatickém nastavování se využívá *Mersenne Twister* s rovnoměrným rozložením hodnot. Takto vytvořená posloupnost se skládá z 256 čísel. Z počátku byl základem pro počáteční stav *Mersenne Twisteru* nedeterministický rovnoměrný generátor náhodných čísel *random_device* (ze stejné knihovny), který správně funguje pro *GNU\Linux*, avšak pro *Microsoft Windows* je implementovaný deterministicky, což ve výsledku způsobovalo pořád stejné semínko při každém spuštění aplikace. Z toho důvodu je nově pro počáteční stav využíván uplynulý čas v sekundách od roku 1970 (začátek epochy).

Semínko nabízí jedinou operaci, která vrací první hodnotu z posloupnosti. Při každém získání hodnoty se posloupnost pootočí. Vzhledem k tomu, že je posloupnost kruhová, tak nikdy nedojde konce.

3.3 MSystém (MSystem) – realizace náhodnosti

Kdykoliv je při generování zmíněna náhodnost, tak se jedná o pseudonáhodnou hodnotu získanou pomocí nějakého tzv. MSystému implementovaného šablonovou třídou programovacího jazyka. Tyto MSystémy pracují úzce s vytvořeným semínkem, které je sdílené napříč celým generátorem.



Obrázek 3.1: vizualizace MSystému

Každý MSystém při jeho vytváření dostane na vstup nějakou posloupnost možných hodnot, která je realizována jako běžné dynamické pole. Datový typ hodnot je definován při vytváření objektu. MSystém si uchovává index na právě aktuální prvek pole. Při získávání hodnoty MSystému, což je jediná možná operace, se nejdříve provede kruhový posun indexu

na základě aktuální hodnoty semínka – dojde-li index na konec, tak se vrátí zase na začátek pole. Čím větší je hodnota semínka, tím delší posun se provede. Maximální hodnota semínka představuje posun o celou délku dostupného pole, což znamená, že index se nezmění. Po úspěšném posunu je vrácena hodnota, na kterou index právě ukazuje.

3.4 Tvorba šablony (půdorys mapy)

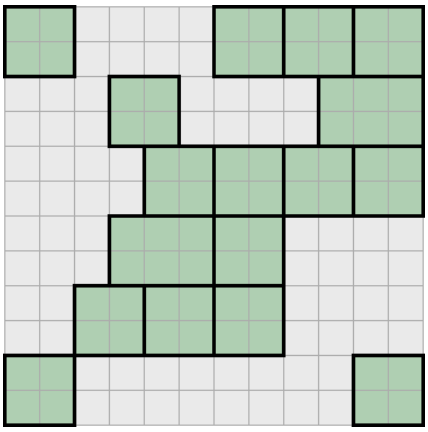
Tato fáze je rozdělena na vytváření dvou vrstev sektorů stejných rozměrů a jejich následné kombinování. Sektor představuje nějakou část mapy a je realizovaný obdélníkem, který má definovaný počátek (X + Y) a rozměry (šířka + výška). Sektor tedy určuje ohraničení, do kterého spadají některé buňky mapy.

3.4.1 První vrstva sektorů

První vrstva představuje sektory využívané při pozdějším vytváření místností.

Na počátku vytváření první vrstvy sektorů se vytvoří jeden velký sektor mající rozměry celé mapy. Postupným dělením poté dochází k rozdělování sektorů na menší sektory. Jestliže je výška větší než šířka, tak se dělí horizontálně, v opačném případě vertikálně. Pokud je šířka a výška stejná, tak je orientace dělící čáry zvolena náhodně (50:50). Pozice dělící čáry v rámci sektoru je taktéž vždy zvolena náhodně a může být i případně upravena, aby nevznikl žádný sektor mající některou stranu menší než je požadovaná minimální možná šířka/výška sektorů. Sektor je možné dále dělit pouze pokud některá jeho strana přesahuje požadovanou maximální možnou šířku/výšku sektorů. Pokud již neexistují žádné sektory, které lze dále dělit, tak se přechází k další části kroku.

V tomto projektu je definovaný minimální rozměr sektoru 2 a maximální 3.



Obrázek 3.2: příklad první vrstvy sektorů

V navazující části dochází k odstraňování některých vytvořených sektorů. Odstraňování je realizováno pomocí tzv. červíčků, kteří se pohybují po mapě a odstraňují sektory. Náhodně na mapě, kde existuje nějaký sektor, se vytvoří červíček mající náhodnou životnost a počáteční směr pohledu. Životnost představuje kolik pohybů dokáže udělat než dojde k jeho odstranění. Červíček se různě náhodně otáčí a skokově postupuje ve směru svého pohledu. Jakmile narazí na nějaký sektor, tak jej kompletně celý odstraní. Červíček nemůže „vypadnout“ z mapy, při každém pokusu o posun za okraj mapy se otočí do protisměru. Jakmile

se aktuální červíček odstraní, tak se na jiné pozici vytvoří nový. Všechno tohle se provádí tak dlouho, dokud není původní počet sektorů redukován na polovinu.

Při odstraňování sektorů bylo potřeba zajistit, aby se červíčci nevytvářeli stále jen v místech, kde se žádný sektor nenachází – způsobilo by nekonečné zacyklení. Po 100 neúspěšných pokusech se provede ruční vytvoření červíčka na pozici, kde se nachází první existující sektor (sektory jsou uložené v dynamickém poli).

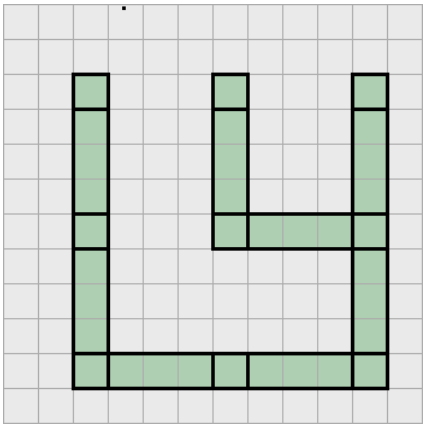
3.4.2 Druhá vrstva sektorů

Druhá vrstva představuje sektory využívané při pozdějším vytváření chodeb.

V první řadě se vytvoří mřížka cest, kde spojnice představují křižovatky. Z každé neokrajové křižovatky tedy vedou čtyři cesty různým směrem. Rozměr mřížky je závislý na rozměru požadované mapy a nemůže ji přesahovat. Cesty musí vždy začínat křižovatkou a taktéž jí musí být i zakončeny. Vzdálenost mezi křižovatkami, nebo-li délka cesty, je vždy minimální rozměr sektoru zvětšený o jedničku – v tomto projektu je vzdálenost 3.

Dále se postupně prochází každá čtveřice sousedních křižovatek, což představuje čtverec složený z cest mezi danými křižovatkami. Pokud se zmíněný čtverec stále skládá ze čtyř cest, tak je možné jednu náhodně odstranit. Tímto způsobem máme garantováno, že nezůstane nějaká cesta nepřístupná a přitom vzniknou náhodně propojené cesty.

Z existujících cest a křižovatek se poté již jen vytvoří sektory. Druhá vrstva sektorů je tedy připravena.

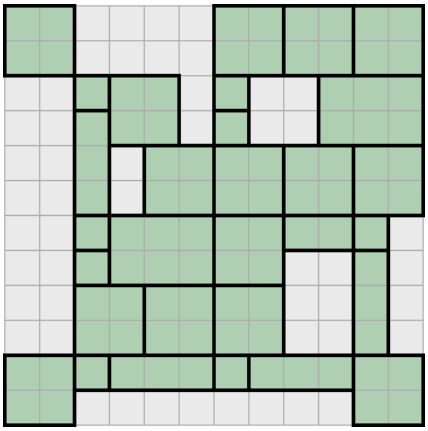


Obrázek 3.3: příklad druhé vrstvy sektorů

3.4.3 Kombinace vrstev

Zde se postupně prochází všechny sektory z druhé vrstvy a přepisují se do první vrstvy tak, že existující sektory první vrstvy zůstanou zachovány beze změny. Ve výsledku se tedy některé sektory vůbec nepřepíší a nebo se přepíší jen zmenšené, protože není možné, aby některé sektory sdílely stejnou buňku.

Zkombinované vrstvy představují samotnou šablonu pro tvorbu mapy. Kromě zvláštního případu, kdy je nějaká část mapy přístupná pouze přes sdílený roh, tak máme zaručeno, že se mapa neskládá z žádných nepřístupných částí. Zmíněný zvláštní případ je ošetřený až při tvorbě mapy.



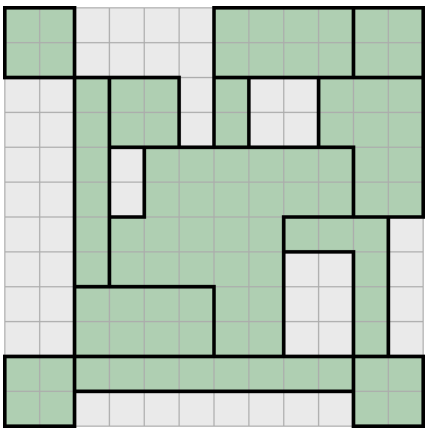
Obrázek 3.4: příklad zkombinované vrstvy sektorů

3.5 Tvorba mapy

Tato fáze pracuje s připravenou šablonou a je rozdělena na vytváření mega sektorů, tvorbu průchodů a následné vytvoření nových sektorů – to vše tvoří výslednou mapu. Mega sektor představuje nějakou ucelenější část mapy. Reprezentuje místnosti či chodby. Skládá se ze sektorů, které tvoří jeho celkový obsah. Neprůchodný okraj mega sektoru tvoří nesdílené hrany (či jejich části) jeho sektorů. Dále si uchovává sadu průchodů sloužící pro propojování mega sektorů.

3.5.1 Mega sektory

Postupně se zpracovávají všechny existující sektory z připravené šablony. Sousední sektory mající rozměr některé strany 1 se vždy spojí do jednoho velkého mega sektoru reprezentující chodbu. Sousední sektory ostatních rozměrů se náhodně spojují do mega sektorů reprezentující místnosti. Čím delší hrany jednotlivé sektory sdílí, tím je větší šance, že se propojí. Pokud sektory sdílí jen roh, tak se nepovažují za sousední.



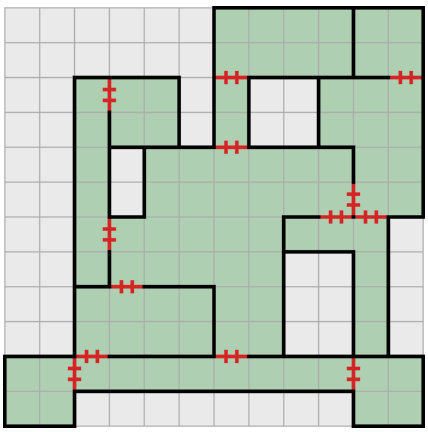
Obrázek 3.5: příklad vytvořených mega sektorů

3.5.2 Průchody

Postupně se prochází všechny vytvořené mega sektory a na základě určitých pravidel se některé propojují pomocí průchodů. Průchod je definován pozicí a směrem. Ke každému průchodu vždy existuje opačný průchod ze sousedního mega sektoru. Dva mega sektory jsou považovány za sousední, pokud v prvním existuje nějaký sektor sdílející nějakou část hrany se sektorem z druhého.

U každého mega sektoru se kontroluje, zda-li jsou všichni jeho sousedi přístupní buď přímo a nebo přes ostatní mega sektory, se kterými je daný mega sektor již postupně propojen. V praxi to znamená, že sousedi mohou být vzájemně přístupní, ale nemusí sdílet žádný průchod. Pokud kontrolované mega sektory nejsou vzájemně přístupné, tak je ihned vytvořen průchod, který je náhodně umístěn na některé sdílené hraně. Pokud již vzájemně přístupné jsou, tak se na základě náhodnosti mohou vytvořit i další dodatečné průchody, aby neexistovala pouze jen jedna možná cesta.

Mega sektory reprezentující chodby jsou zpracovány jako první, aby přes ně vedly hlavní cesty a chodba tak plnila svůj účel.



Obrázek 3.6: příklad vytvořených průchodů (červené čáry)

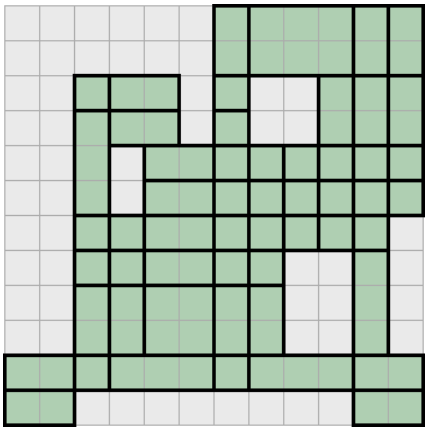
Pokud zůstane nějaký mega sektor bez žádného průchodu, tak je ihned odstraněn. Takový mega sektor se totiž pouze skládá z jednoho sektoru, který s ostatními maximálně sdílí jen roh – jedná se o zvláštní případ zmíněný dříve.

3.5.3 Sektory

Jakmile jsou mega sektory a průchody připravené, tak se může přistoupit na poslední část tvorby mapy, což představuje vytvoření zcela nových sektorů. Sektory budou později sloužit jako části scény, avšak v současném stavu zatím nesplňují požadavky herního enginu, tudíž je potřeba je vytvořit znova dle jiných pravidel.

Pro každý mega sektor se do vlastní pomocné mřížky zaznamenají všechny příslušné buňky, které jsou ohraničeny jeho sektory, a vyznačí se celkový okraj. Průchody na určitých buňkách způsobí mezeru ve vyznačeném okraji, tím se vytvoří dva rohy. Nyní se postupně po buňkách prochází pomocná mřížka a z každého rohu se vede do všech možných směrů vertikální či horizontální pomyslná dělící čára, dokud se nenarazí na okraj. Na úplném konci daného procházení jsou v mřížce pomocí existujících dělících čar vyznačené nové sektory, které již stačí akorát vytvořit a přiřadit příslušnému mega sektoru.

Nově vytvořené sektory budou zároveň sloužit i pro vytváření sad potencionálně viditelných částí scény, protože každý roh znamená změnu potencionální viditelnosti.



Obrázek 3.7: příklad nově vytvořených sektorů

3.6 Plnění kontextu

Jedná se o poslední a nejrozsáhlejší fázi procedurálního generování, při kterém se plní předaný kontext a jeho scéna. K tomu využíváme již připravenou mapu. Fáze sestává z následujících částí: příprava assetů, příprava pomocných mřížek, vytváření sad potencionální viditelnosti, plnění scény a konečné čištění assetů.

Podrobný popis a vysvětlení vkládaných objektů se bude nacházet až v další kapitole zaměřující se na výslednou hru.

3.6.1 Příprava assetů

Nejdříve je nutné nahrát všechny potřebné modely. Pro modely využívané při vytváření terénu je dále potřeba připravit dočasné modely reprezentující všechny možné variace natočení, aby je nebylo nutné opakovaně rotačně deformovat u každé buňky mapy – jediná deformace, která se u každé buňky provádí, je tedy pouze posun. Každý terén si totiž uchovává vlastní kopie meshů jejichž 3D body se již nachází v konečné požadované pozici v rámci scény – tento přístup byl podrobně popsán v kapitole 2.3.4.

3.6.2 Příprava pomocných mřížek

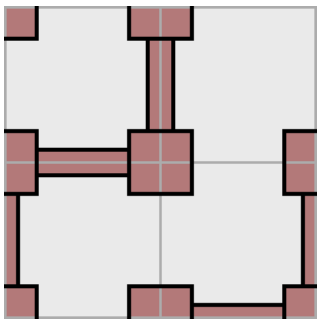
Pro účely plnění scény je připraveno několik pomocných mřížek (dvojrozměrná pole), do kterých se pro každou buňku mapy ukládají různé potřebné informace.

Nejdříve se postupně prochází existující mega sektory a všechny jejich sektory. Pro každý sektor je vytvořena odpovídající část scény. K buňkám v pomocných mřížkách jsou uloženy následující informace:

- identifikátor mega sektoru – aby bylo možné odlišit různé místnosti či chodby;
- odkaz na příslušnou část scény;

- vizuální tématika – použitá sada 3D modelů;
- typ buňky – zda-li je součástí místnosti nebo chodby;
- struktura – podlaha, strop, zdi a sloupy.

Zdi se nachází na okraji mega sektorů a sloupy na vrcholech mezi buňkami propojující minimálně dvě zdi. Každá buňka tedy teoreticky může mít až 4 strany zdí (co hrana to zeď) a až 4 čtvrtky sloupů (v každém rohu jedna) – viz obrázek 3.8. Existuje-li v buňce podlaha, tak vždy automaticky existuje i strop.



Obrázek 3.8: příklad zdí a sloupů

Procházení sektorů se děje ve dvou cyklech, protože v jednom cyklu není možné v použitém algoritmu správně vyplnit všechny viditelné čtvrtky sloupů. Druhým cyklem se doplní potřebné čtvrtky v buňkách nemající žádné zdi, kde některý z jejich rohů je součástí sloupu propojující zdi sousedních buněk.

Následně se projdou všechny přidružené průchody, pro které se v odpovídajících buňkách odstraní zdi a náhodně nastaví, zda-li bude mít průchod dveře či nikoliv. U mega sektorů reprezentující místnosti a mající pouze jediný průchod se vždy nastaví dveře a navíc ještě uzamčené – tímto způsobem se nikdy nemůže stát, že by uzamčená místnost oddělovala jiné místnosti. Dále se využívá pravidlo, že klíče se nesmí nacházet v uzamčené místnosti a tudíž není nutné kontrolovat jejich přístupnost.

V posledním kroku se dle určitých pravidel ještě náhodně nastaví rozmístění pokladů, jejichž počet a umístění v určité buňce je již nutné znát při plnění scény. Poklady jsou různé objekty určené ke sbírání.

3.6.3 Vytváření sad potencionální viditelnosti

Části scény jsou již vytvořené, ale pro každou zatím chybí sada potencionálně viditelných částí scény. Sady se vytváří opět na základě mřížky a tudíž není výsledek vždy úplně přesný, protože se využívá rasterizace pomyslné úsečky na relativně velké buňky. Nedokonalost použitého algoritmu taktéž způsobuje, že se občas chybně nezobrazuje nějaká část za rohem.

Postupně se prochází existující sektory a kontroluje se viditelnost se všemi ostatními sektory. Sektory jsou společně uložené v jedné pomocné sadě, aby nebylo nutné již znova procházet mega sektory. Kontrola probíhá tak, že se z prvního sektoru vezmou buňky nacházející se v jeho rozích a vede se pomyslná rovná čára k nějaké rohové buňce z druhého sektoru. Duplicitní buňky v rámci sektoru nejsou zbytečně opakovaně kontrolovány. Pomyslnou čáru tvořící úsečku je poté nutno rasterizovat. Pokud rasterizovaná úsečka neprochází

přes zeď, tak jsou dané sektory označeny jako vzájemně viditelné a tím pádem jsou rozšířeny potencionální viditelnostní sady příslušných částí scény.

Vzhledem k tomu, že při takovéto rasterizaci dochází k velké ztrátě přesnosti, tak se pro jednu úsečku provádí třikrát, vždy s různým zaokrouhlením aktuální pozice kroku. Krok představuje posun na rasterizované úsečce jakýmkoliv směrem (i diagonálním) o jednu buňku. Maximální počet kroků je vždy největší strana pomyslného obdélníku tvořeným dvěma kontrolovanými buňkami. Využívají se následující zaokrouhlovací techniky:

- běžné matematické zaokrouhlení,
- zaokrouhlení dolů,
- zaokrouhlení nahoru.

Stále to však bohužel nestačí k dokonalému výsledku.

Rasterizovaná úsečka může mít jednotlivé buňky spojené pouze rohem. V tomto případě stačí když jsou buňky přístupné přes alespoň jednu ze dvou možných cest.

3.6.4 Plnění scény

Nyní se může přistoupit k samotnému plnění scény. Pomocné mřížky mají stejné rozměry a tudíž se buňky ze všech mřížek prochází zároveň. V rámci každé neprázdné buňky se pro příslušnou část scény vytváří objekty, rozšiřuje terén a vytváří neprůchodné kolizní tvary. Zároveň zde i dochází k propojování částí scény.

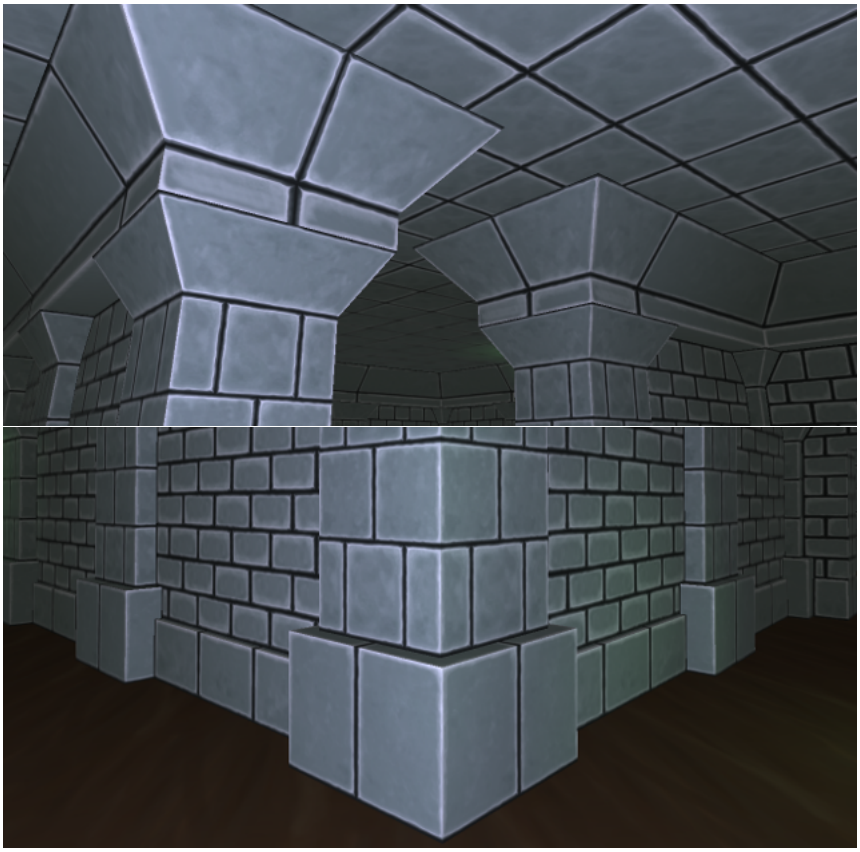
V závislosti na typu buňky (místnost či chodba) se náhodně vytváří a rozmísťují objekty. Taktéž se při vytváření bere v potaz, zda-li se v buňce nachází nějaká zeď či nikoliv. Některé objekty se totiž nevkládají na zem ale přímo na zeď. Dále existují objekty, které má smysl vkládat pouze ke zdím. Pro každý objekt je nastaven kolizní nebo viditelnostní tvar. Dále jsou připojeny potřebné modely či světla. Pokud se má v buňce nacházet i nějaký poklad, tak je většinou náhodně umístěn na právě vloženém objektu – možnosti umístění jsou ručně nastavené pro každý druh objektu.

Při plnění se nijak nekontroluje, zda-li vkládané objekty s něčím nekolidují, ani jestli existuje dostatek prostoru pro pohyb mezi objekty. Tím pádem se vše vkládá tak, aby byl bez kontroly u všeho zaručený bezkolizní stav a nebyla narušena průchodnost. Po vložení objektu se v generátoru ani neuchovává odkaz na daný objekt, akorát se v příslušné buňce nastaví příznak, že zde již není možné vkládat nějaké jiné objekty. Při pohledu na scénu je tedy patrné, že rozmístění objektů dodržuje mřížku.

Na základě struktury a vizuální tématiky buňky se z připravených modelů a jejich natočených variací sestavuje terén pro příslušnou část scény – ukázku sestaveného terénu lze vidět na obrázku 3.9. Chodby a místnosti mohou využívat odlišné modely. Při rozšiřování terénu se tedy kromě použitého modelu nastavuje akorát pozice v rámci scény. Vzhledem k tomu, že terén je akorát vizuální prvek, tak je nutné část scény vyplnit ještě nějakými kolizními tvary, aby nebylo možné procházet skrz zdi a sloupy. Zdi a sloupy jsou ohraničeny kolizními kvádry, které se zpravidla vzájemně překrývají, aby byla zaručena dokonalá neprůchodnost.

Pokud se mezi sousedními částmi scény nenachází žádná zeď, tak jsou vzájemně propojeny – tudíž jsou průchozí.

V poslední řadě se vytvoří potřebné objekty spojené s hráčem a umístí na nějakou předem vybranou počáteční lokaci.



Obrázek 3.9: ukázka sestaveného terénu

3.6.5 Čištění assetů

Po ukončení plnění scény již akorát nastává odstranění všech dočasných variací modelů, které byly použity při sestavování terénu.

Kapitola 4

Výsledná hra

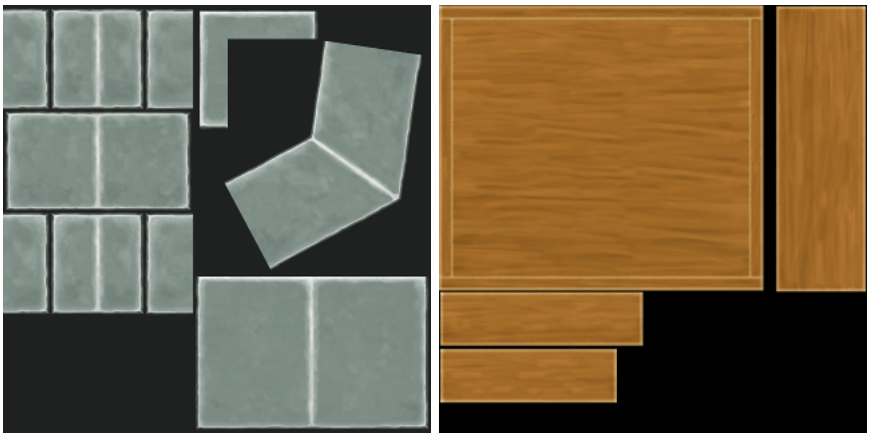
Samotná výsledná hra představuje integraci vlastní aplikace s herním enginem společně s naplněním kontextů pomocí procedurálního generátoru nebo vlastních aplikačních procedur. Dále je potřeba implementovat herní logiku a mechaniky, což se děje taktéž ve zmíněných procedurách nebo vlastních definovaných objektech. V neposlední řadě je nutné vytvořit požadované 3D modely a nakreslit potřebné textury. Na konci všeho přichází testování hry a dolaďování nedostatků.

4.1 Grafika

Veškeré kreslení textur se provádělo v open-sourcovém grafickém editoru *GIMP*. Modely se vytvářely pomocí open-sourcového 3D editoru *Blender*.

4.1.1 Textury

Textury jsou ukládány ve formátu *PNG*, který nabízí bezeztrátové uchovávání obrázků. Další vhodnou vlastností je podpora alfa kanálu (průhlednost), který však v tomto projektu nebyl u žádné textury využit. Tento formát již dlouhou dobu patří mezi rozšířené standardy, tudíž nebylo problém najít vhodnou knihovnu, která umožňuje jednoduché nahrávání obrázků v daném formátu pro potřeby *OpenGL*.



Obrázek 4.1: textura pro sloup (vlevo) a pro knihovnu (vpravo)

Maximální využívané rozměry textur činí 1024x1024 bodů, což se aplikovává na větší modely. U těch menších modelů se využívá 512x512 nebo 256x256 bodů. Přestože engine podporuje i odleskové (specular) textury, tak se využívají jen difůzní (diffuse).

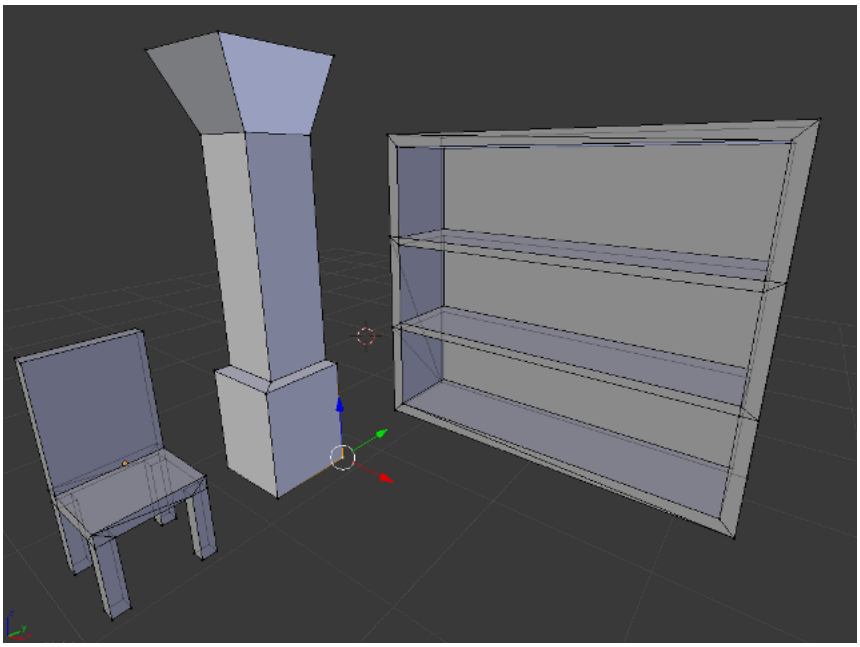
Styl textur se snaží co nejvíce přiblížit texturám z fantasy her jako je například *Torchlight* nebo *World of Warcraft*. Výsledek je samozřejmě od inspirace hodně daleko, avšak postupem času bude docházet k viditelnému zlepšování. Nedokonalost textur má na svědomí nejen nedostatečné zkušenosti s 2D grafikou, ale i nevhodné vybavení – vše bylo kresleno pomocí myši. Kvůli tomu nebyly do textur kresleny detaily jako jsou například rýhy či úlomky, protože ve výsledku to poté vypadalo hůře než textury bez detailů.

V případě nějakého velkého projektu by samozřejmě bylo vhodné, aby se o textury postaral zkušený a umělecky založený grafik.

4.1.2 Modely

Modely jsou ukládány ve formátu *OBJ*, který sice nenabízí moc vlastností, ale pro potřeby tohoto projektu bohatě postačuje. Soubory tohoto formátu jsou krásně čitelné v jakémkoliv textovém editoru, tudíž není problém provádět případné rychlé ruční úpravy.

Rozměry všech modelů jsou definovány tak, aby je již nebylo nutné při zobrazování zvětšovat či zmenšovat. Dále se při modelování odstraňují ty polygony, které nelze při všech možných natočeních kamery vidět – např. záda objektů připevněných ke zdi či spodní strany stojících objektů.



Obrázek 4.2: některé použité modely

Při vytváření modelů se využívá přístup low-poly modelování (meshe se skládají z malého počtu polygonů), což ve výsledku tvoří velice jednoduché modely působící na pohled hodně hranatě. Pokud by se u modelů vyvarovalo pravých úhlů a rovnoběžných hran, tak by byl výsledek mnohem lepší.

V případě nějakého velkého projektu by samozřejmě bylo vhodné, aby se o modely postaral zkušený a umělecky založený 3D grafik.

4.2 Ovládání a nastavení

Hra se ovládá pomocí klávesnice a myši. Nejsou žádné speciální nároky na vstupní zařízení, tudíž postačí ty úplně nejběžnější/nejlevnější.

Vzhledem k tomu, že hra nenabízí žádné nastavení vlastních kláves (zahrnuje i tlačítka myši), tak jsou pro vyvolání určité akce k dispozici i alternativní klávesy. Ovládat postavu lze následovně:

- W A S D / šipky – pohyb,
- myš – otáčení,
- E / PCtrl / levé tlačítko myši – interakce/aktivace/sebrání,
- LShift / PShift / pravé tlačítko myši – běh.

Běh je možné zapnout pouze při pohybu směrem vpřed. Je k dispozici možnost nastavení vlastní citlivosti pohybu myši, protože každé vstupní zařízení může mít odlišnou citlivost.

Jelikož nejsou dostupné možnosti nastavení rozlišení, tak hra automaticky běží v režimu zobrazení přes celou obrazovku, což bez obtíží funguje na všech běžných zobrazovacích zařízeních. Ve hře je možné si nastavit vlastní hodnotu jasu, aby si hráč atmosféru podzemí mohl přizpůsobit podle toho, zda-li hraje ve dne či v noci.

4.3 Herní logika a mechaniky

Samotná herní logika se dělí do dvou hlavních sekcí: počáteční místnost zvanou hub a podzemí. Hub je vytvořený ručně a vypadá pořád stejné. Slouží jako náhrada běžného hlavního menu. Podzemí je vytvořeno procedurálním generátorem a vypadá vždy jinak. Vstup do podzemí představuje samotný start hry. Každá sekce je realizována vlastním kontextem.

Obrázek 4.3: načítací obrazovka (vlevo) a konec hry (vpravo)

Existují ještě dodatečné kontexty, které se využívají při zobrazování informací o načítání (loading) nebo konci hry (game over). Vzhledem k tomu, že engine prozatím nemá žádné možnosti pro zobrazování 2D objektů, tak jsou veškeré informační texty realizovány texturou aplikovanou na 3D stojící obdélník osvětlený 3D světlem.

4.3.1 Hub

Hra záměrně nemá žádné běžné hlavní menu. Cílem je vyzkoušet, do jaké míry lze zažité menu u menších her eliminovat a nahradit ho in-game realizací.

Ihned po spuštění se hráč objeví v malé počáteční místnosti a před očima mu visí dřevěná tabule znázorňující, které klávesy se používají k ovládání hry. Není však vysvětleno, co která klávesa dělá, a tak si musí hráč vše vyzkoušet sám. Když si zmíněnou místnost více prohlédne, tak narazí na další tabule, na kterých se nachází: informace o autorovi, znázorněné poslední a nejvyšší dosažené skóre, nastavování citlivosti myši, nastavování jasu obrazu a krátký obrázkový návod vysvětlující základní princip hry.

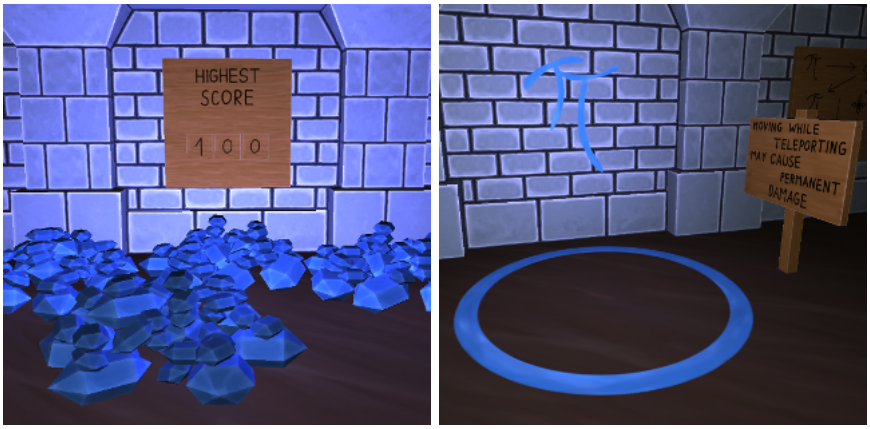
Obrázek 4.4: ovládání (vlevo nahoře), cíl hry (vpravo nahoře), nastavení citlivosti myši (vlevo dole) a nastavení jasu obrazu (vpravo dole)

Novým hráčům nemusí být z obrázkového návodu ihned patrný princip hry, avšak jakmile se více porozhlédnou a vyzkouší pár věcí, tak postupně na všechno přijdou. Tento přístup nutí hráče samostatně objevovat neznámé věci a zvyšuje potřebu znovuhratelnosti. Ne každému však bude tento starodávný přístup vyhovovat, protože je moderní generace

hráčů zvyklá na tzv. „vodění za ručičku“. Taktéž i občasní hráči digitálních her mohou okamžitě ztratit zájem. Jedná se tedy o experimentální přístup, který však bude v případné lepší podobě uplatňován i ve všech dalších hrách.

Pod tabulí ukazující nejvyšší dosažené skóre se nachází pevně umístěné modré zářící krystaly, které hráči mají napovědět, na základě čeho se asi bude počítat výsledné skóre. Krystaly v podzemí představují poklad určený ke sbírání. Vzhledem k tomu, že každé podzemí má odlišný počet krystalů, tak je výsledné skóre počítáno procentuálně.

Dále se v místnosti nachází přenosový svítící portál znázorněný modrou runou na zdi a kruhem na podlaze. Jak se portál používá si taktéž hráč musí zjistit sám. Nejedná se však o nic složitého, stačí přijít k portálu a dívajíc se na runu na zdi použít aktivační klávesu. Při aktivování portálu dojde k teleportaci hráče do neznámého opuštěného temného podzemí.



Obrázek 4.5: nejvyšší skóre (vlevo) a přenosový portál (vpravo)

4.3.2 Podzemí

Po vstupu do podzemí se hráč objeví opět na stejně vypadajícím přenosovém portále a před očima se mu zjeví rotující krychle se šesti jehlany znázorňující abstraktní časovač. Brzy zjistí, že jehlany v určitém časovém intervalu postupně mizí. Jakmile zmizí poslední jehlan, tak se portál uzavře a okamžitě nastává konec hry s nulovým skóre. Pro úspěšné zakončení hry je nutné se vždy vrátit k přenosovému portálu a stihnout se přenést zpět do hubu, kde je poté vypočítáno dosažené skóre.

Hráč tedy musí stále myslet na dostatečnou časovou rezervu. Zároveň není nikde psáno kolik času je dohromady k dispozici. Po pár průchodech podzemí však hráč brzy začne odhadovat kolik přibližně času představuje jeden jehlan na časovači. Jedinci toužící po co nejlepším skóre jistě sáhnout po stopkách a dostupný čas si změří, avšak se tím ochuzují o možnost rozvíjení schopnosti podvědomého odhadování času.

Uvnitř podzemí hráč prozkoumává jednotlivé místnosti a chodby, a snaží se posbírat co nejvíce modrých krystalů. Občas narazí na nějaké zamčené dveře, ke kterým nejdříve musí najít klíč – dveře bez klíčové dírky lze otevírat běžně. Za zamčenými dveřmi lze najít větší krystaly, které dávají dvojnásobné skóre.

Počet náhodně rozmístěných menších krystalů je vždy poloviční počet vytvořených nezamčených místností, do kterých se zmíněné krystaly vkládají. Větší krystaly se vždy náhodně umísťují do zamčených místností – jejich počet tedy přesně odpovídá počtu zamče-

ných místností. V jedné místnosti se nikdy nenachází dva krystaly. Hráč nemá k dispozici informaci o celkovém počtu umístěných krystalů.

Obrázek 4.6: rotující časovač (vlevo nahoře), běžné dveře (vpravo nahoře), menší krystal (vlevo dole) a větší krystal (vpravo dole)

Existují dohromady až tři různé klíče sloužící k odemykání zamčených dveří. Každý klíč má svůj specifický tvar, který odpovídá specifické klíčové dírce. Počet zamčených dveří se stejnou klíčovou dírkou je vždy v závislosti na vygenerované mapě a tudíž náhodný. Může se tedy i stát, že nebudou k dispozici všechny tvary klíčů. Nikde se neukazuje, které klíče již hráč vlastní. Před sebráním klíče je tedy vhodné si jej prohlédnou a následně zapamatovat. Tento přístup podporuje rozvíjení krátkodobé paměti.

Každý klíč se vždy náhodně umístí do nějaké chodby. Pokud existuje dostatečný počet chodeb, tak se nikdy nenachází dva klíče ve stejné chodbě.

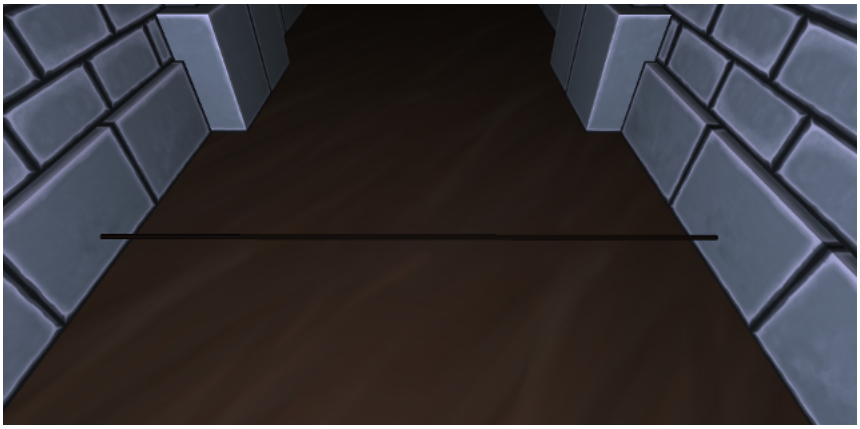
Hlavní výzva při sbírání krystalů spočítá v tom, že se hráč nesmí ztratit. Nemá k dispozici žádnou mapu ani prvotní ukázku vygenerovaného podzemí. Jako orientační body mu můžou sloužit různé dekorační objekty (nábytek, svítící houby, . . .) či otevřené/zavřené dveře, ale při troše nepozornosti není obtížné se i tak ztratit. Podzemí tedy svým způsobem představuje takové bludiště, což napomáhá rozvíjení prostorové paměti a orientace. Aby to ještě nebylo tak jednoduché, tak se otevřené dveře v náhodném časovém intervalu zase zavírají.

Pokud hráč nebude využívat běh, tak nemá šanci dosáhnout vyššího skóre. Běžet však nelze pořád a je to závislé na dostupné energii, která se při běhu rychle vyčerpává. Při

Obrázek 4.7: klíč (vlevo) a zamčené dveře (vpravo)

běžné chůzi či stání se energie zase pomalu dobíjí. Záměrně není k dispozici ukazatel zbývající energie, aby si hráč na funkčnost této mechaniky musel přijít sám – zvyšuje nutnost znovuhratelnosti. Ve výsledku pravděpodobně skončí u stylu nazývaného „spartanský běh“. Jakýkoliv náraz do zdi nebo objektu přeruší běh.

Toulku po podzemí hráči ještě stěžují pasti ve formě náhodně u země natažených provazů nacházejících se v chodbách. Pokud hráč při běhu narazí na nějaký takový provaz, tak o něj zakopne a zvedáním ztrácí svůj drahocenný čas. Stejně tak vyčerpá svoji veškerou energii a tudíž není schopen ihned pokračovat v běhu dále. Při nepozornosti není obtížné provaz přehlédnout, protože do určité vzdálenosti není moc vidět – za předpokladu nastaveného optimálního jasu obrazovky. Nic nebrání hráči, aby si nastavil vysoký jas a provazy tak viděl na míle daleko, avšak ochuzuje se tím o pravou atmosféru temného podzemí. Provaz lze bez obtíží přejít běžnou chůzí.



Obrázek 4.8: provaz natažený mezi zdmi

Nábytek v podzemí má akorát dekorační a orientační význam. Nevážou se na něj žádné speciální akce. Svítící houby slouží k vylepšování atmosféry temného podzemí a mají budit pocit toho, že v podzemí opravdu dlouho nikdo nepůsobil – příroda si to začíná zabírat.

Obrázek 4.9: svítící houba (vlevo nahoře), stůl (vpravo nahoře), prázdná knihovna (vlevo dole) a židle (vpravo dole)

4.4 Uživatelské rozhraní

Nepočítáme-li rotující časovač, tak hra nemá žádné grafické uživatelské rozhraní. Cílem je, aby hráč viděl přesně to, co vidí jeho postava ve hře. Tímto způsobem se hráč lépe s danou postavou sžije. Dosáhnout toho v tomto projektu nebylo ani moc obtížné, protože není nutné hráči předávat žádné informace o stavu postavy.

Někteří hráči mohou mít problém se správným mířením, protože není zobrazený žádný vizuální zaměřovač, a tak musí přesný směr pohledu nacházející se ve středu obrazovky odhadovat. Z toho důvodu mají všechny interaktivní objekty širší interaktivní oblast, aby nebylo nutné mířit přesně na daný objekt. Dále má postava poměrně dlouhé „ruce“, tudíž hráč nemusí odhadovat, zda-li na nějaký objekt již dosáhne či nikoliv, protože si podvědomě stoupne do optimální vzdálenosti. Případně si to párkrát vyzkouší a poté bude již správně odhadovat.

Na první pohled není patrné se kterými objekty lze interagovat, protože nejsou nijak zvýrazněny – ani při namíření kamery. Mnozí mohou tento přístup negativně kritizovat, avšak myšlenka je taková, že je hráč nucen vyzkoušet vše na co narazí – objevuje pro něho zatím neznáme věci. Hra tedy nepodporuje přístup: „všímej si jen tohoto,“ ale naopak: „všímej si všeho“. Na druhou stranu je ale pravda, že v této hře není dostatek různých objektů a nečekaných událostí, aby se dal využít plný potenciál zmíněného přístupu.

4.5 Testování a doladďování

Už od začátku vývoje se počítalo s tím, že samotná hra bude trvat jenom pár minut. Není tedy nutné generovat moc velkou mapu, ve které by se hráč s největší pravděpodobností stejně brzy ztratil. Mapa však nesmí být ani moc malá, aby hra nebyla zase moc jednoduchá. Je tedy zvolen optimální střed. Výsledná mapa má rozměry 48x48 buněk.

Dále bylo nutné zvolit takový odpovídající dostupný čas, u kterého je i při používání běhu obtížné dosáhnout vyššího skóre, aby byl hráč nucen k znovuhratelnosti a sebezdokonalování. Různě se testovalo s časem v intervalu od 3 do 5 minut. Nakonec je časovač nastaven na 3 a půl minuty, a získat maximální skóre je téměř nemožné.

Taktéž i rychlost vyčerpávání/dobíjení energie bylo potřeba vyzkoušet při různých hodnotách, protože samotný běh je klíčem k úspěchu. Nejrozumněji se tvářily následující hodnoty: 6,25 s pro kompletní vyčerpání energie a 12,5 s pro plné dobití energie. Z toho jednoznačně vyplývá, že energie se vyčerpává dvakrát rychleji. Pro dosažení vysokého skóre je tedy důležité, aby hráč nezakopával o provazy, protože tím ztrácí veškerou nabitou energii. Na začátku každé hry je energie plně nabita.

Kapitola 5

Závěr

5.1 Herní engine

Herní engine je stále ve vývoji a bude se na něm dále pracovat. Většina funkčnosti je implementována experimentálně a je stále předmětem změny. Optimalizace, což představuje zrychlení výkonu a zmenšení paměťové náročnosti, se bude ve větší míře řešit až v další fázi vývoje. Zatím nebylo rozhodnuto, které části enginu budou uživateli přístupné přímo, proto je naprostá většina členských proměnných a funkcí veřejná (vlastnosti tříd jazyka *C++*). Tento způsob realizace prozatím zvyšuje riziko nechtěného zásahu do některé části programového kódu, což může způsobit zvláštní chování nebo pád aplikace. Engine tedy není zatím připraven pro využití širší veřejností a už vůbec ne v komerční sféře.

Prozatím nejsou implementovány techniky *Frustrum Culling* a *Occlusion Culling*, tudíž se momentálně vykreslují věci jak za kamerou, tak i schovány za zdí. To vše ale pouze v rámci potencionálně viditelných částí scény, takže určitá optimalizace vykreslovacích nároků již vyřešena je.

Velkou změnou budou muset projít prvky pro scénu, aby bylo možné jednotlivé scény propojovat, což umožní i vzájemný přechod jejich kamer a vložených objektů. Dále je nutné implementovat nahrávání assetů do paměti grafické karty na základě aktivní scény určitého aktivního kontextu – momentálně se nahrávají assety pro všechny scény daného kontextu. Pravidla pro nahrávání se budou nastavovat ručně pro jednotlivé scény, aby se zbytečně neomezoval výkon nějakou inteligentní správou assetů.

Přístup nahrávání 3D dat terénů příslušných částí scény do paměti grafické karty se taktéž musí předělat. Pravděpodobně dojde k navrácení k původnímu přístupu (viz 2.3.3), avšak přidá se další vlákno aplikace, které bude samotné nahrávání obstarávat, tudíž vykreslování obrazu v hlavním vlákně aplikace nebude nijak omezováno.

Zde se nachází seznam dalších plánovaných změn a rozšíření:

- zapínání/vypínání V-SYNC,
- zvukový systém,
- podpora pro gamepady,
- nastavování vlastností textur,
- více tvarů světel,
- další ohraničující tvar pro objekt – zásahový tvar,

- nastavování síly a směru gravitace objektu,
- deformace meshů pomocí kostí,
- *Anti-Aliasing* pomocí *FXAA*,
- *Ambient Occlusion* a další.

5.2 Procedurální generátor

Přestože generátor využívá relativně jednoduché techniky, tak výsledek je dostatečně kvalitní. Budoucí herní projekty budou určitě procedurální generování opět využívat, tudíž se bude pokračovat ve zdokonalování zde použitého generátoru.

Ve výsledné hře se nepromítlo to, že již nyní generátor nabízí možnost odlišného vzhledu každé místnosti a chodby – stačí jen provést menší změny v programovém kódu a mít potřebné na sebe navazující modely, ze kterých lze poskládat výsledný terén.

Primárně se plánuje upravit generátor tak, aby výsledná mapa již nevyužívala mřížku, resp. mřížka se bude využívat pouze v počátečních fází generování a v některé pozdější fázi dojde k její zahození.

5.3 Výsledná hra

Kvůli nedostatku času hra bohužel nemá žádné zvuky ani hudbu. Taktéž i atmosféra prostředí není tak úplně ideální. Samotný princip hry v dnešní době ani moc hráčů nezaujme. Jednalo se však o jedinou možnou cestu, jak splnit potřebné požadavky zadání projektu v dostupném čase a při pracovních povinnostech.

Přestože výsledek nemá s původním cílem moc společného a splňuje jen nový realističtější cíl, tak se jedná o docela zajímavou hru, jejíž vývoj poskytl důležité poznatky a zkušenosti. Nyní je mnohem jasnější představa o tom, jak dlouho a kolik práce přibližně zabere vývoj jedné plnohodnotné hry. Zároveň však byla potvrzena i očekávaná smutná skutečnost, že požadavky hráčů není v dnešní době vůbec jednoduché uspokojit. Zájem o testování této hry byl velice nízký a zájem o dlouhodobé hraní ještě menší, ne-li žádný. Vzhledem k tomu, že se jedná akorát o takovou prvotní zkoušku, tak není vůbec nutné věšet hlavu. Další hra bude již mnohem více propracovanější a zajímavější, protože již existují nějaké nabité zkušenosti a zároveň nebude působit žádný velký časový pres.

Na základě reakcí od lidí, kteří měli zájem vyzkoušet tuto hru, je možné vyslovit očekávaný konečný verdikt: neúspěch. Samotný neúspěch této hry spočívá v tom, že pouhé sbírání krystalů zaujme spíše pouze občasné hráče, avšak na druhou stranu je zase ihned odradí to, že na vše musí přijít sami postupným zkoušením a znovuhratelností. Pravidelní hráči nemají problém se zkoumáním herních mechanik, avšak sbírání krystalů je zase nijak nezaujme. Hra tedy nesplňuje základní potřebné požadavky ani jedné ze zmíněných skupin. S úspěchem této zkušební hry se však ani nepočítalo.

Ve vývoji této konkrétní hry se nebude již dále pokračovat, ale veškeré úsilí bude přesunuto do vývoje další hry, která bude využívat již vylepšený daný herní engine. Tato hra však neupadne v úplném zapomnění, protože se jedná o předchůdce (prequel) plánované hry, která již splní prvotní vizi tohoto projektu. Kvůli vysokým nárokům na herní engine, se však na ní bude muset nějakou dobu počkat...

Literatura

[1] Albeza, B.; Mills, C.; Bloomer, H.: *3D collision detection*. Mozilla Developer Network, Říjen 2015, [Online; navštíveno 28.02.2017].
URL https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection

[2] Geelnard, M.; Berglund, C.: *GLFW Window guide*. Srpen 2016, [Online; navštíveno 10.02.2017].
URL http://www.glfw.org/docs/latest/window_guide.html

[3] Policarpo, F.; Fonseca, F.: *Deferred shading tutorial*. Pontifical Catholic University of Rio de Janeiro, Březen 2009, [Online; navštíveno 15.03.2017].
URL
https://web.archive.org/web/20090306111435/http://www710.univ-lyon1.fr/~jciehl/Public/educ/GAMA/2007/Deferred_Shading_Tutorial_SBGAMES2005.pdf

[4] Technologies, U.: *Unity - Delta Time*. 2016, [Online; navštíveno 10.02.2017].
URL https://unity3d.com/learn/tutorials/topics/scripting/delta-time

[5] University, N.: *Tutorial 15: Deferred Rendering*. Newcastle University, Říjen 2012, [Online; navštíveno 16.03.2017].
URL https://research.ncl.ac.uk/game/mastersdegree/graphicsforgames/deferredrendering/Tutorial%2015%20-%20Deferred%20Rendering.pdf

Seznam obrázků