

Gymnázium Christiana Dopplera, Zborovská 45, Praha 5

ROČNÍKOVÁ PRÁCE
NodeShot - RTS hra v Javě

Vypracoval: Dan Rakušan
Třída: 8.J
Školní rok: 2018/2019
Seminář: Seminář z programování

Prohlašuji, že jsem svou ročníkovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím s využíváním práce na Gymnáziu Christiana Dopplera pro studijní účely.

V Praze dne 22. prosince 2019

Dan Rakušan

Obsah

1	Úvod	3
1.1	Základní informace	3
1.2	Koncept hry	3
2	Ovládání a herní mechaniky	4
2.1	Herní elementy	4
2.2	Průběh hry	4
2.3	Uživatelské rozhraní v průběhu hry	5
2.4	Ovládání a klávesové zkratky	5
3	Technická stránka	6
3.1	Terén a simulace	6
3.1.1	Terén	6
3.1.2	Hlavní simulační vlákno	8
3.1.3	Korupce/Creeper	8
3.1.4	Vývoj a pokusy s algoritmem šíření korupce	9
3.2	Herní entity	10
3.2.1	Budovy	10
3.2.2	Projektily	11
3.2.3	Efekty	11
3.3	Game art design	12
3.3.1	Modely budov	12
3.3.2	Exploze	13
3.4	Průběh vývoje	14
3.5	UI a hlášení výjimek	14
4	Závěr	16
Přílohy		17

1. Úvod

1.1 Základní informace

Předmětem této práce je hra žánru RTS (Realtimová strategie), kterou momentálně vyvíjím. Pracovní název aplikace je také „NodeEngine“. Jak už je řečeno v názvu práce, hra je napsaná víceméně od základu v jazyce Java. Hra využívá především herní knihovnu s názvem LibGDX, která umožňuje používat hardwarově akcelerované renderování skrz OpenGL a která mimo jiné poskytuje početné množství herně orientovaných pomocných tříd a i několik externích programů. LibGDX samotné používá další populární knihovnu jménem LWJGL (Lightweight Java Game Library), která zajišťuje všechny elementární volání OpenGL.

Hra jako taková (tzn. především z pohledu herních mechanik) je v relativně nedokončeném stavu a stále se na ní pracuje. Tudíž některé mechaniky v této práci zmíněné nemusí být plně funkční.

1.2 Koncept hry

Hra je hrána z takzvaného top-down ortografického pohledu a probíhá v náhodně generovaném světě tvořeném mřížkou políček. Tento terén se skládá z vrstev různě vysokých ploch, na které lze stavět budovy. Koncept hry je silně inspirován hrami Creeper World a Factorio. Ze hry Creeper World od vývojáře Knuckle Cracker je i vypůjčen hlavní nepřítel hráče: korupce (jinak nazývaný „creeper“). Korupce je systém políček s hodnotou reprezentující úroveň jakési viskózní tekutiny, proti které hráč bojuje. Tato tekutina se postupně šíří z jednoho políčka do políček sousedících a dynamicky reaguje na výšky terénu a možné útoky od hráče. Hráč útočí zejména útočnými budovami, které se musí postavit a připojit do širokého systému dodávek střeliva. To nás přivádí k systému „Packagu“, předmětu, které s pomocí konektorů a dopravníků putují mezi jednotlivými hráčem postavenými budovami. Celý systém začíná u zdroje rud, které je možné vytěžit a napojit na ostatní budovy zpracovávající dané rudy na použitelné materiály, které lze posléze přeměnit na střelivo nebo jiné produkty. Střelivo pak může putovat buď do skladů, nebo přímo k útočným budovám („turretům“).

Ekonomika základny je založena na využití měny, tzn. „bitů“, za které je možno kupovat jednotlivé budovy. Bity lze získat prodejem zpracovaných materiálů nebo ničením generátorů korupce. Dalším strategickým elementem je využití energie, kterou budovy spotřebují a její výroba energetickými generátory.

2. Ovládání a herní mechaniky

V současném stavu je hra pojata spíše jako technické demo či sandbox. Herní mechaniky jsou v dosud rané fázi a cíl hry víceméně postrádá význam.

2.1 Herní elementy

Zde je krátký souhrn všech základních herních elementů a systémů:

- **Terén a herní svět**

Ohraničený herní svět tvořený políčky. Terén má různé výšky a okraje ploch jsou specificky zakončeny. Políčka mají mnoho vlastností jako například výskyt těžitelných rud. Mnoho dalších herních systémů na nich závisí a jejich výšky hrají velkou roli ve spojení s korupcí.

- **Korupce / Creeper**

Hlavní nepřítel hráče, mřížkový systém rozšiřující se entity dynamicky reagující na terén a interakce s hráčem. Vytváří se v generátorech korupce a může být zničena útočnými věžemi.

- **Budovy a package systém**

Modulární systém budov, které spolu mohou interagovat pomocí systému balíčků („package systém“). Budovy zajišťují těžbu a produkci materiálů, obranu před korupcí, zdroj energie i bitů.

2.2 Průběh hry

Hra se při zapnutí musí nejprve načíst. V pozadí se otevře černé okno a v popředí se objeví menší okénko oznamující průběh načítání. Po načtení se objeví velice základní hlavní menu, kde je možné hru pouze zapnout či vypnout. Po zapnutí hry se hráč objeví přímo ve hře s náhodně generovanou mapou o velikosti 256x256 políček. Ve středu mapy se objeví jedna budova s názvem „Headquarters“. Tento objekt je hlavní budova hráče a jejím zničením hráč prohrává. Hráč může stavět ostatní budovy a rozširovat svoji základnu. Při rozširování základny se mohou aktivovat náhodně rozprostřené generátory nepřátelské korupce, které začnou generovat šířící se korupci. Pokud se dostane korupce do kontaktu s políčkem, na kterém stojí budova, tato budova bude zničena. Cíl hráče je postupně zničit všechny generátory korupce na mapě prostřednictvím stavby útočných věží a produkcí munice.

Budovy lze propojit pomocí dopravních pásů. Aby se propojil například důl („mine“) s továrnou („factory“), nejdříve se musí postavit „exporter“ vedle dolu, který lze posléze sítí dopravních pásů propojit k příslušnému „importeru“ postaveného vedle továrny.

2.3 Uživatelské rozhraní v průběhu hry

Uživatelské rozhraní hry je v tuhle chvíli velice základní. Ve spodní části obrazovky se nachází hlavní herní panel. V tomto panelu jsou zobrazeny budovy, které může hráč postavit. Také se zde zobrazují informace o herních objektech a stavě herních zdrojů.

Při stisknutí tlačítka Escape se hra pozastaví a objeví se nabídka pozastavení. Z této nabídky může hráč jít zpět do hlavního menu (hra zůstane běžet), může otevřít panel nastavení nebo regenerovat mapu. Panel nastavení obsahuje především nastavení ladění, ale také pár grafických nastavení.

2.4 Ovládání a klávesové zkratky

Shrnutí klávesových zkratek:

„**LMB**“ - Levé tlačítko myši

„**RMB**“ - Pravé tlačítko myši

„**MMB**“ - Stisknutí kolečka myši

Klávesa nebo kombinace stisknutí	Akce
LMB	Výběr budovy / Postavení budovy / Pohyb kamery
LMB + Levý SHIFT	Postavení budovy a pokračování ve stavbě stejné budovy
RMB	Výběr budovy / Zrušení stavby budovy / Rozšíření dopravního pásu
MMB	Rotace budov (které lze otáčet) / Pohyb kamery
LMB + C (při posouvání myši)	Vytvoření korupce kolem kurzoru o průměru štětce (brush)
LMB + V (při posouvání myši)	Odstranění korupce kolem kurzoru o průměru štětce (brush)
LMB + T (při posouvání myši)	Nastavení výšky terénu kolem kurzoru o průměru štětce (brush). Výška bude nastavena na hodnotu výšky vrstvy (layer)
NUM 0 - 9	Nastavení výšky vrstvy (0 až 9)
Hvězdička (*)	Nastaví výšku vrstvy na 10
Šipka doprava	Zvýší rychlosť simulace
Šipka dolů	Sníží rychlosť simulace
Šipka doleva	Výchozí rychlosť simulace
Kolečko myši dolů / Page DOWN	Zmenší přiblížení
Kolečko myši nahoru / Page UP	Zvětší přiblížení
Q	Rotace budovy doleva
E	Rotace budovy doprava

3. Technická stránka

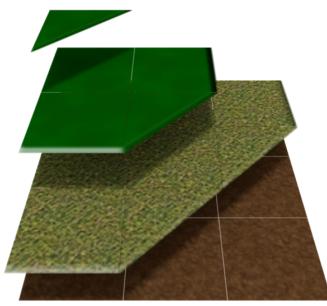
3.1 Terén a simulace

3.1.1 Terén

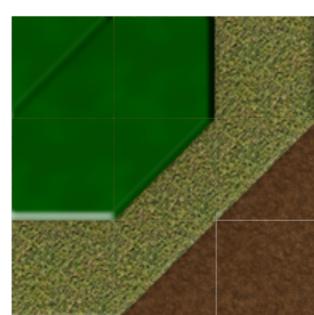
Terén hry je náhodně generovaná síť políček (v kódu takzvaných „Chunků“) rozdělených do 32x32 sekcí. Sekce slouží jako prostředek optimalizace a umožňují kdykoli získat oblast kolem daného políčka. Kamera se pohybuje v jistém rozmezí kolem tohoto terénu tvořící herní mapu. Celou mapu můžeme rozdělit na 3 komponenty:

- **Terén a korupce**

Políčka mají pevně dané výškové úrovně, na které reaguje korupce a budovy. Okraje těchto výškových zón jsou systematicky „zaobleny“. Reálně je terén složen ze čtvercových polí a tyto okraje mohou vypadat jako například trojúhelníky nebo jakýkoli jiný tvar. Toto vyžaduje vrstvení jednotlivých textur pomocí speciálního shaderu, který v jednom OpenGL volání renderuje kompozit až 4 na sobě složených textur. Podobnou variaci shaderu používá i vykreslování korupce. Ta je renderována odděleně, ale stejně jako terén jsou její jednotlivé úrovně spojeny do jedné textury. Systém generace a vykreslování korupce navíc řeší odstín / barvení jednotlivých úrovní korupce a její průhlednost.



(a) 3D render ze strany



(b) 2D render ze shora

Obrázek 3.1: Vysvětlení principu renderování terénu a korupce. Na tento terén je poté ještě nanesen fog of war

- **Fog of war**

Fog of war je název systému viditelnosti polí. Budovy poskytují vizi v jistém okruhu. Ve chvíli, kdy je vize jednou políčkem získána, už se jí nevzdá. Tato vize slouží spíše jako oblast aktivní nebo neaktivní korupce. Korupce se obnovuje a šíří pouze v oblastech s aktivní vizí. Neaktivní pole navíc fungují jako bariéra a představují nepřekonatelnou zed'. Fog of war se ve hře projevuje jako temný odstín překrývající neaktivní pole. Cílem tohoto systému je usměrňovat šíření korupce a představovat problematiku rozšiřování základny.

```

varying vec4 v_color;
uniform sampler2D u_texture;
varying vec2 v_texCoords0;
varying vec2 v_texCoords1;
varying vec2 v_texCoords2;
varying vec2 v_texCoords3;

void main()
{
    vec4 tex1 = texture2D(u_texture, v_texCoords0);
    vec4 tex2 = texture2D(u_texture, v_texCoords1);
    vec4 tex3 = texture2D(u_texture, v_texCoords2);
    vec4 tex4 = texture2D(u_texture, v_texCoords3);

    vec4 finalColor = vec4(0,0,0,0);

    if (tex4.a > 0.15) {
        if (tex4.a == 1.0) {
            finalColor = tex4;
        } else {
            finalColor = vec4(mix(tex4.rgb, finalColor, 0.8), 1.0);
        }
    }
    if (tex3.a > 0.15) {
        if (tex3.a == 1.0) {
            finalColor = tex3;
        } else {
            finalColor = vec4(mix(tex3.rgb, finalColor, 0.8), 1.0);
        }
    }
    if (tex2.a > 0.15) {
        if (tex2.a == 1.0) {
            finalColor = tex2;
        } else {
            finalColor = vec4(mix(tex2.rgb, finalColor, 0.8), 1.0);
        }
    }
    if (tex1.a > 0.15) {
        if (tex1.a == 1.0) {
            finalColor = tex1;
        } else {
            finalColor = vec4(mix(tex1.rgb, finalColor, 0.8), 1.0);
        }
    }
    gl_FragColor = finalColor;
}

```

(a) Příklad terrain shaderu



(b) Okraj viditelnosti

Obrázek 3.2: Zdrojový kód terrain shaderu v jazyce GLSL a obrázek systému „fog of war“.

3.1.2 Hlavní simulační vlákno

Ve hře běží 2 oddělené herní cykly. Jeden LWJGL cyklus vykreslování a druhý cyklus ovládaný simulačním vláknem. Tento cyklus běží na pomalejších 30 obnoveních za sekundu a podle něho se aktualizují a časují všechny logické operace ve hře. To jsou například aktualizace budov, prvků package systému, korupce a podobně. Tento přístup vyžaduje částečnou synchronizaci herních zdrojů, ale umožňuje kontrolu nad rychlostmi a průběhu mnoha dějů.

3.1.3 Korupce/Creeper

Korupce se přechovává jako hodnota v políčku. Podle různých parametrů se šíří do sousedících políček. Korupci generují speciální budovy, které udržují stálou vysokou hodnotu korupce v jednom políčku, ze kterého se posléze šíří do okolní oblasti. Korupce reaguje na terén a šíří se na políčka s vyšší úrovní, pouze když je úroveň korupce dost vysoká. Políčko s korupcí šíří jisté procento své korupce při volání simulačního vlákna a úroveň korupce musí být trochu vyšší než výška cílového souseda.



Obrázek 3.3: Šíření korupce a její spawner

Korupce je v současné verzi ovládána odděleným vláknem běžícím vedle hlavního simulačního vlákna. Toto je z toho důvodu, že korupce je obnovována ještě pomaleji než ostatní logika, která běží na frekvenci simulačního vlákna (to je 30 snímků za vteřinu). Simulační vlácko každých přibližně 10 logických snímků volá a obnovuje ze spánku vlácko korupční, které poté nezávisle na běhu simulačního vlákna zavolá update metody všech aktivních světových sekcí. Volány jsou pouze sekce, ve kterých jsou políčka s korupcí. Ve chvíli kdy korupční vlácko dokončí svojí operaci, uvrhne samo sebe do spánku a čeká na volání hlavního simulačního vlákna.

Stejně jako terén jsou jednotlivé úrovně korupce ohraničené hranami. Tyto hrany jsou předurčený set textur. Tyto textury jsou pouze pro jeden odstín a nejsou to všechny možné kombinace. Skládání těchto textur na sebe, jejich průhlednost a jejich speciální atribut odstínu (tmavost jednotlivých úrovní, která je odvozena z množství korupce nad výškovou úrovní terénu) řeší speciální shader,

který pracuje podobně jako zmíněný shader terénu. Hlavní rozdíl je využití atributu odstínu, který je specifický každému políčku. Korupce i terén využívá při vykreslování OpenGL koncept objektů VBO (Vertex Buffer Object) a IBO (Index Buffer Object). Každému vrcholu políčka je přiřazeno celkem 13 atributů. 8 z nich jsou souřadnice textur a mezi nimi je i jedna hodnota odstínu, která umožnuje do 1 políčka vykreslit na sebe hrany s rozdílným odstínem. Nevýhoda tohoto přístupu je fakt, že se obarví/zatemní i pixely tvořící okraj. To znamená, že čím tmavší je odstín, tím je okraj méně výrazný. Možné řešení do budoucna by bylo vykreslovat okraje nezávisle na plochách korupce.

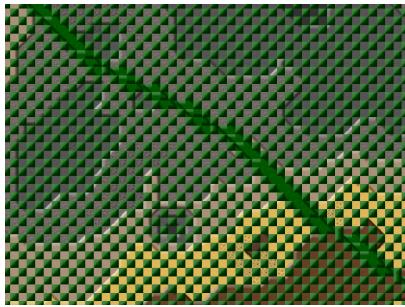
3.1.4 Vývoj a pokusy s algoritmem šíření korupce

Algoritmus, který udává, jak se políčka s korupcí chovají, se během vývoje mnohokrát změnil. Cílem těchto algoritmů bylo rozdělit na základě jistých parametrů hladinu korupce jednoho políčka na políčka sousední, a to co nejrychlejším a nejednodušším způsobem. To se zprvu může zdát jako triviální úkol, ale podle mých zkušeností tomu tak není.

První verze tohoto algoritmu se jednoduše podívala na hodnoty korupce okolních políček a na základě těchto rozdílů rozdělila část své vlastní korupce ostatním polím. Dřív však než takový systém mohl fungovat, bylo důležité zajistit, aby samotné rozdělení úrovní korupce proběhlo rovnoměrně a ve stejném okamžiku. Kvůli tomu si každé políčko drží svoji hodnotu „creeperChange“, která reprezentuje změnu hladiny vyvolanou minulým simulačním krokem. Po simulačním kroku a aktualizaci těchto hladin proběhne ještě jedna aktualizace, která plošně aplikuje tyto změny hladin. Pokud by k tomuto nedošlo, tak by se projevoval výrazný pohyb hladin korupce ve směru ovlivněném způsobem, jakým byla políčka cyklována při jejich aktualizaci.

Zpět k samotným aktualizacím. Již zmíněný systém do jisté míry fungoval, ale vytvořil velice chaotický pohyb hladin mezi políčky. Takové chování nebylo zcela žádoucí, a proto sem musel zkoušet jiné způsoby kalkulace a převodu korupce. Současný systém se dívá pouze na 4 sousedy (ve 4 hlavních směrech) s nižší úrovní korupce a těmto sousedům přidá nezávisle na ostatních procentuální díl rozdílu mezi jejich úrovněmi (v kódu je toto procento označeno „flowRate“). U tohoto modelu je problém, že může nastat situace, kdy pole celkově vydá více korupce než samo má. Z toho důvodu nesmí „flowRate“ přesáhnout 25%, což zabraňuje tomuto modelu rychle měnit hladiny na větších plochách (tak by se chovala řídká voda).

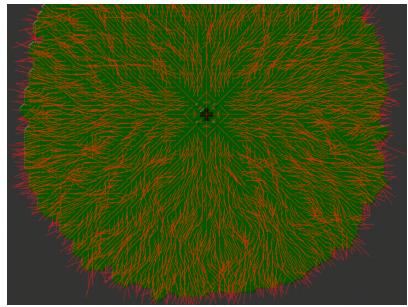
Mimo tyto dva modely jsem také zkoušel napsat model, který by pracoval s vektorovým směrem proudu a jeho silou. Takový pokus nepřinesl uspokojivé výsledky a byl mnohonásobně výkonostně náročnější. Na začátku ledna jsem se také pokoušel vytvořit model, který by se choval jako již zmíněná voda. To znamená model, který by se snažil dosáhnout co nejrychleji jedné hladiny. Bohužel všechny mé testovací modely se chovaly moc chaoticky a pomalu.



Obrázek 3.4: Když „flowRate“ přesáhne 25%



Obrázek 3.5: Neúspěšné pokusy s modelem vody



Obrázek 3.6: Pokus se směrovými vektory a sílou proudu

3.2 Herní entity

Všechny herní objekty nutně implementují interface „Entity“. Tento interface zajišťuje základní metody pro představení daného objektu v herním světě a určením jeho typu.

Entity ve hře vždy mají vlastní metodu vykreslování a metodu aktualizace. Ve všech případech se entity přidávají do specifických tříd manažerů. Tyto třídy jsou velice důležité pro organizaci kódu a jsou zodpovědné za volání metod vykreslování a logických aktualizací jejich přidruženého typu entit.

3.2.1 Budovy

Mimo třídy reprezentující každou herní budovu, kterou lze postavit je v kódu několik abstraktních tříd, které představují různé vlastnosti budov. Tyto základní implementace mohou být například „AbstractStorage“, „AbstractMine“, nebo „AbstractIOPort“. Na těchto třídách jsou postaveny všechny ostatní budovy. Cílem tohoto systému je jistá modulárnost budov. V kódu například není abstraktní třída útočné věže, protože útočná věž ve skutečnosti pouze potřebuje nějaký počet uskladněné munice a také musí být napojitelná na síť předmětů. Proto stačí, aby dědila po třídě „AbstractStorage“.

Pro zasazení budov v systémů polí má každé políčko referenci na budovu, která se na poli nachází. Toto například umožňuje detekci kolizí budov, když se mají postavit přes sebe.

Package systém

Package systém pracuje s konceptem bodů (v kódu „Node“) navzájem propojených konektory (v kódu „Connector“, ve hře dopravní pásy), po kterých mohou putovat různé typy balíčků (v kódu „Package“) představujících herní předměty. Tyto balíčky posléze komunikují s budovami pomocí speciálních vstup/výstup terminálů (v kódu „AbstractIOPort“). Konce dopravních pásů (objekty „Node“) ani pásy samotné nemají ve hře reprezentaci jako budovy. Proto si každý z těchto objektů spravuje vlastní reprezentaci v herním světě jako budovy, čímž získávají vlastnosti budov.

Balíčky samotné se po síti pásů pohybují vždy mezi dvěma body. Jejich pozice se pak určuje jako procentuální násobek vektoru přímého pásu, po kterém zrovna putuje. Když dosáhne sta procent

a dostane se na konec pásu, upozorní svůj takzvaný „PathHandler“, který určí, na jaký pás může balíček pokračovat a resetuje balíček na následujícím pásu.

Balíčky mohou také vytvářet kolony. Každý balíček na pásu zastupuje nějaký procentuální interval. Každý pás kontroluje, zda se nějaký balíček nepokusil pohnout do intervalu jiného. V tom případě bude jeden z balíčků pozastaven. Tento mechanismus poté vytváří kolony například v případě, že jeden balíček nemá kam pokračovat na konci pásu.

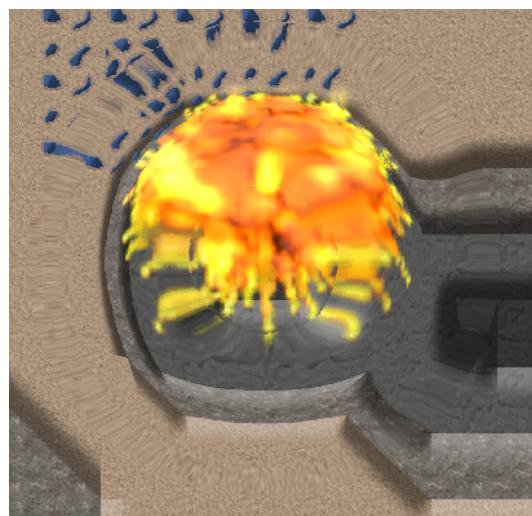
3.2.2 Projektily

Projektily jsou jednoduché entity, které danou rychlosť putují mezi dvěma body (tedy útočnou věží a cílem) a při dosažení cíle provedou na souřadnicích cíle potřebnou akci (např. efekt exploze a narušení terénu).

3.2.3 Efekty

Efekty jsou další jednoduché entity, které reprezentují vizuální efekty v herním světě. Ve hře to je exploze.

Exploze se skládá ze tří samostatných efektů: animované exploze, kouřových částic a efektu tlakové vlny. Poslední z těchto efektů je poměrně zajímavý, protože se jedná o speciální shader, který na základě červeného a zeleného kanálu kruhovité textury vykreslené mimo obrazovku (do tzv. „offscreen bufferu“) posouvá pixely na obrazovce podél obou os. Toto vytváří efekt tlakové vlny deformující prostor.



Obrázek 3.7: Efekt exploze v akci

3.3 Game art design

3.3.1 Modely budov

Budovy jsou renderovány jako obrázky pokrývající polička, na kterých stojí. Tyto obrázky jsou plnohodnotné rendery texturovaných modelů vytvořených v 3D editoru Blender. Tento přístup mi dává velkou kontrolu na vzhledem daných budov, jelikož mám v Blenderu mnoholeté zkušenosti. Budovy jsou vyobrazeny ortografickou kamerou pod úhlem 60°, což celkově dodává světu jistý náznak třetí dimenze v porovnání s plně 2D terénem.



Obrázek 3.8: Perspektivní rendery různých budov.

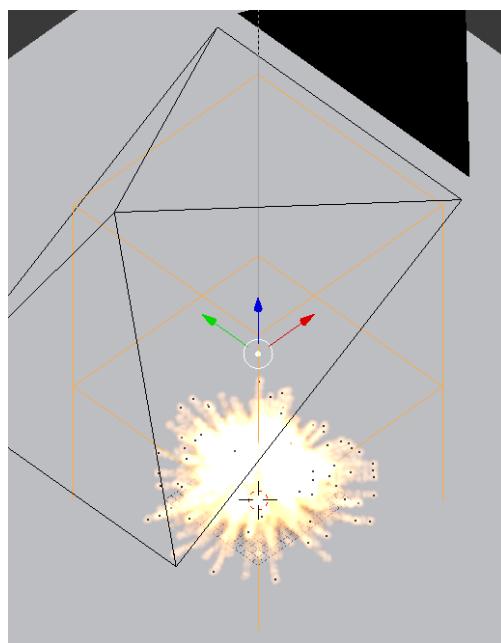
Budovy také mohou mít různou grafiku pro různé stavby nebo mohou být animované. Všechny snímky animací i obrázky budov jsou spojené do několika stránkového atlasu. Tento atlas slouží jako databáze jednotlivých snímků a hra si z těchto stránek vybírá jednotlivé obrázky podle souřadnic textur, které atlas poskytuje. Tento princip je velice důležitý kvůli výkonu vykreslování. Do grafické karty se nahrávají tyto celé stránky a jsou využity pro vykreslování více obrázků. To předchází načítání každé textury zvlášť a je to důležitá optimalizace. Do takového atlasu jsou naškládány i jednotlivé grafiky terénu, korupce i všech ostatních entit. Animace fungují jednoduše jako rychlé výměny obrázků. Snímky animací jsou v atlase očíslovány a hra je pak podle času mění. Speciální případ je rotace útočných věží. Turrety nepoužívají jeden samotný rotující obrázek věže. Místo toho používají sérii již vykreslených snímků jakési animace. Každý snímek reprezentuje jistý úhel rotace hlavy věže. Logika věže poté vybere správný snímek pro její rotaci.



Obrázek 3.9: Stránka atlasu s obrázky budov, je zde i několik snímků fází rotace útočných věží.

3.3.2 Explosions

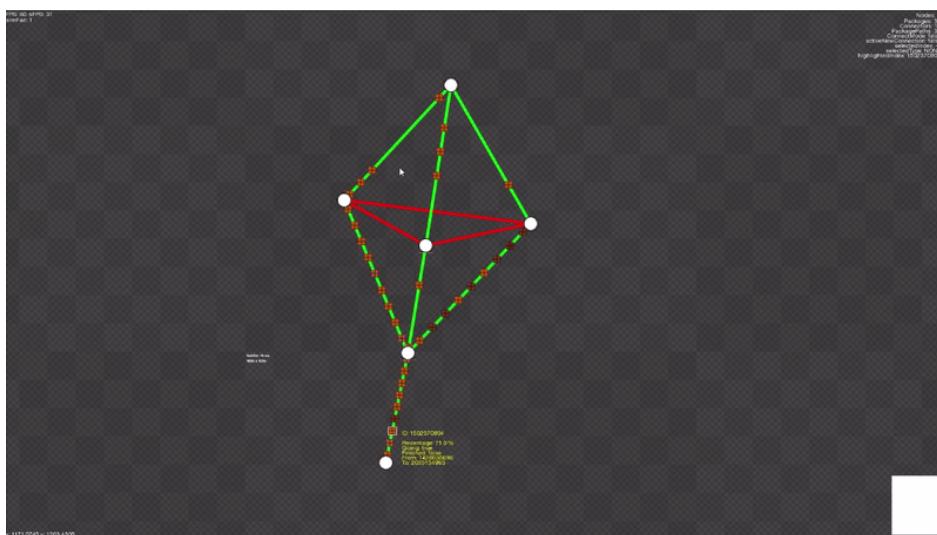
Dále jsem také mohl využít kouřové simulace Blenderu pro vytvoření animace exploze a popřípadě v budoucnu dalších efektů. Simulační engine Blenderu je velice flexibilní a dokáže vytvořit širokou škálu efektů se specifickými vlastnostmi, které jsou v daném případě zapotřebí. Při vykreslování těchto efektů a jejich zanesení do herního světa se mohou objevit problémy s například průhledností nebo se zkreslením barev. Obecně je vykreslování volumetrických objektů problematictější a výkonně náročnější než vykreslování standardních objektů.



Obrázek 3.10: Simulace kouře v Blenderu

3.4 Průběh vývoje

Projekt začal relativně skromně v lednu 2018 jako malá testovací aplikace napsaná bez pomoci externích knihoven a využívající softwarového renderování Java2D. Tato aplikace se postupně přeměnila na současnou verzi hry. Tato aplikace položila základy současného systému konektorů a dopravníků. Začátkem léta 2018 jsem hledal způsob jak zlepšit výkon vykreslování a narazil jsem na Swing knihovnu GLG2D, která využívá právě OpenGL. To mě přivedlo k myšlence využití přímo OpenGL a rozhodl jsem se přepsat projekt a adaptovat ho pro využití s LibGDX. Postupně se tato aplikace přeměnila na současnou verzi hry, která využívá mnoho konceptů OpenGL jako například využití shaderů a celkově efektivnějšího vykreslování OpenGL. LibGDX také nabízí třídu kamery, která poskytuje metody přepočítávání projekčních maticí a tudíž možnost jednoduchého využití sekundárního systému herních souřadnic a pohyb uživatelského pohledu v herním světě.



Obrázek 3.11: První verze běžící na OpenGL

3.5 UI a hlášení výjimek

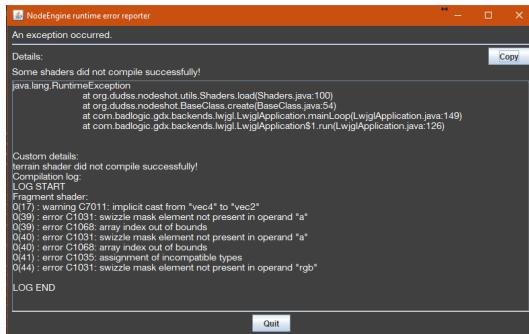
Všechno uživatelské rozhraní uvnitř hlavního okna, které běží na hlavním vlákně OpenGL využívá knihovny LibGDX. To nabízí knihovnu plně přizpůsobitelného grafického uživatelského rozhraní, které nabízí podobné funkce jako ostatní klasické grafické rozhraní (jako např. Java Swing nebo C# WinForms). Není sice zdaleka podobně komplexní, ale funguje dobře a je dostačující. Je využité jako herní menu a okna uvnitř hry.

Ve dvou případech nejsou využity tyto knihovny LibGDX a to z pádných důvodů. Problém s těmito LibGDX okny je ten, že fungují pouze na vlákně OpenGL (které je pouze jedno) a navíc se nechovají jako pravé externí okna, ale naopak jsou vždy uvnitř hlavního okna hry.

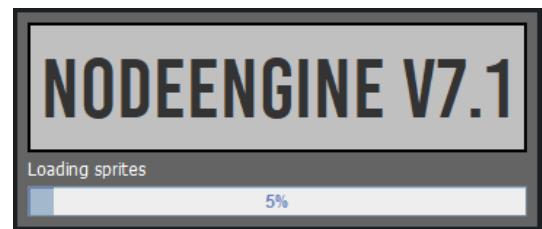
Tyto dvě vlastnosti nejsou žádoucí při fatálních chybách. Proto jsem vyvinul relativně jednoduché

hlášení neočekávaných výjimek pomocí grafického rozhraní Swing bežícího na odděleném vlákně, které se hodí na diagnostiku problémů, když hra běží mimo vývojářské prostředí

Druhý případ, kdy je využití oken LibGDX nevhodné je načítání hry. Když se načítají herní zdroje tak není možné zároveň provádět vykreslování. LibGDX pro to ve standardních knihovnách má svoje vlastní řešení, ve kterém se střídá načítání zdrojů a aktualizace obsahu okna, ale vyžaduje použití dedikované LibGDX třídy na načítání zdrojů. Protože jsem implementoval svoji vlastní třídu na načítání zdrojů tak pro mne nebylo toto řešení vhodné a vyžadovalo by velké úpravy ve způsobu nejen načítání, ale i přistupování k načteným datům. Proto jsem využil relativně nešetrné, ale jednoduché řešení a to použití Swingového okna bez rámu, které je úplně oddělené od hlavního okna a funguje jako takzvaný „splash screen“. Toto řešení tedy umožňuje souběžně načítat data a dynamicky aktualizovat průběh v odděleném okně.



Obrázek 3.12: Hlášení o chybné komplikaci shaderu



Obrázek 3.13: Úvodní „splash screen“

4. Závěr

V tuhle chvíli je hra ve zvláštní fázi. Tento projekt nezačal jako hra a ani dlouhou dobu po jeho začátku nebylo vytvoření hry cílem. Dlouhou dobu byla tato práce jen produkt mého testování, experimentování a učení se nových věcí. Zejména tedy v pozdější fázi OpenGL, jelikož byl toto první projekt, ve kterém jsem se s OpenGL setkal. Z této absence jasného cíle se vytvořil současný hlavní problém hry jako takové, a to její koncept, který nebyl nikdy přímo vymyšlen, ale spíše se přetvořil v to, co bylo zrovna potřeba.

Celkově mi projekt určitě rozšířil obzory a to nejen v rámci počítačové grafiky, ale i samotného programovaní v jazyce Java i programování větších projektů, které potřebují průběžnou dokumentaci a čistý kód. Určitě si myslím, že pokoušet se vyvíjet RTS hru tohoto měřítka jako první projekt nebyla dobrá volba a tento projekt ještě dokončený zcela není. Ale jak už jsem zmínil, v tomto případě byl průběh vývoje trochu odlišný.

Přílohy

Link na BitBucket repozitář práce:

<https://bitbucket.org/xDUDSSx/nodeshotengine-opengl/src/master/>

Pokyny k spuštění aplikace se nachází v souboru README.md