**Obsah**

[Osnova 2](#_Toc98782327)

[Úvod 2](#_Toc98782328)

[Typy Renderingů 2](#_Toc98782329)

[Ray Casting 2](#_Toc98782330)

[Studium Problematiky 2](#_Toc98782331)

[Herní Enginy 2](#_Toc98782332)

[GUI a Obsluha událostí 3](#_Toc98782333)

[Vybrané Java API 3](#_Toc98782334)

[Analýza 3](#_Toc98782335)

[Cíl hry 3](#_Toc98782336)

[Příběh 3](#_Toc98782337)

[Scénář 3](#_Toc98782338)

[Tvorba Projektu 4](#_Toc98782339)

[Příklady tříd 4](#_Toc98782340)

# Úvod

Tématem mé maturitní práce je 3D hra v Javě. Hra je vytvářená na způsob staré střílečky Wolfenstein 3D. Důvody, proč jsem si vybral tohle téma jsou celkem tři. Chci se zabývat programováním a vývojem video her. Hry typu DOOM, Warcraft I a hlavně The Elder Scrolls: Daggerfall jsem si opravdu oblíbil. A zároveň algoritmus, na základě kterém Wolfenstein funguje je v zásadě dost jednoduchý, tudíž je to dle mého názoru úctyhodný start pro nadšeného programátora a matematika.

Cílem práce je naprogramovat jednoduchou hru s funkčním systémem renderingu.

# Studium Problematiky

## Typy Renderingů

### Ray Casting (Wolfenstein 3D)

Ray casting ve hře Wolfenstein 3D je způsob renderingu, ve kterém posíláme paprsky ve 2D prostoru a zjišťujeme kolizi/protnutí s objekty. Tento proces je poměrně rychlý a ne tolik náročný, protože vysíláme paprsek pouze za každý pixelový sloupec na obrazovce z kde každý paprsek počítá pouze jednu jednoduchou matematickou rovnici, díky čemuž byl velice populární v dobách prvních 3D her. Je však omezený a kvůli geometrickým úpravám, které se v průběhu dělají, aby obraz vypadal realistický ztrácí schopnost vykreslovat všechny tvary a tak je limitován na víceméně jen jednodušší prostředí.

V klasickém Ray Castingu se paprsky vysílají za každý pixel na obrazovce, a tudíž se zde neobjeví problém se zobrazenou geometrií. Tímto způsobem ale opakujeme ještě složitější algoritmus mnohonásobně vícekrát a k plynulému chodu programu by nejspíše byla potřeba lepšího výkonu.

### Ray Tracing

Ray Tracing je velmi podobný Ray Castingu, ale jak název napovídá, jedná se o algoritmus, který se zabývá sledováním světla, resp. jeho paprsků. V této metodě se sleduje nejen cesta světelného paprsku, ale také to, co se s ním stane když dopadne na nějaký povrch, kde se může odrazit nebo vstřebat. Stejně jako je tomu i u Ray Castingu zde vystřelujeme paprsek za každý pixel a sledujeme, zda narazí na nějaký povrch. Při kolizi se pak podle vlastnosti daného povrchu, např. zda se jedná o průhledný nebo o odrazivý povrch, spustí spoustu dalších výpočtů, které vedou k vytvoření osvětlení a stínů.

## Herní Enginy

Co se vývoje 3D hry týče existují zde enginy, ve kterých se hry tohoto typu dají vytvořit bez řešení jakýchkoliv problémů, týkajících se matematiky renderingu. Mezi takové dva největší enginy bych vybral Unity Engine a Unreal Engine. Tyto programy už v sobě mají zabudované tyto systémy a vývojář se může prakticky soustředit jen na ty „jednodušší“ úkoly a nemusí se zaobírat vytvářením už několikrát použitých algoritmů, které se jen opakují. Na svůj projekt jsem však žádný herní engine nepoužil, protože mým hlavním cílem bylo naprogramovat právě algoritmus na renderování prostředí.

## GUI a Obsluha událostí

Grafické rozhraní je ve hře zpracováno pomocí Java knihovny Swing, která má v sobě zabudované funkce pro vykreslení textu do okna. Celá hra se ovládá pouze klávesnicí, ale je možné použít myš při volení možnosti u vyskakujících okna.

## Vybrané Java API

K tvorbě hry jsem použiDl knihovny Java Swing a Java AWT, které jsou v projektu využity k projekci a vykreslení jak pohledu hráče tak i samotného grafického rozhraní.

# Analýza

## Cíl hry

Dostat se na konec a zničit zdroj hrozby, kterým je květ v nejvyšším patře budovy.

## Příběh

Na Zemi hned vedle gymnázia J. K. Tyla spadl meteorit z černého kamene. Toto spadlé vesmírné těleso zpočátku nevykazovalo žádné známky nebezpečí, ale po několika týdnech se začali dít opravdu podivuhodné věci. Zvěř v okolí byla výraznějí agresivnější a plody rostlin byly absolutně nepoživatelné. Zároveň se začaly rostliny zvedat ze země a útočily na civilisty. Hlavní postava je vycvičený voják, který se ujímá životně nebezpečného úkolu a to prosekání si cesty skrze napadenou a polorozpadlou budovu gymnázia a zničit “srdce rostlin“, čímž ukončí dosavadní teror.

## Scénář

Hráč je v prvním podlaží budovy gymnázia J. K. Tyla. Velká část podlaží je zaterasená sutinami vyšších pater. Prostřílí si cestu skrze lidožravé rostliny a najde cestu do dalšího patra.

Ve druhém patře hráč opět prochází polozavalené prostředí budovy školy. Na konci opět nalezne cestu do dalšího podlaží.

Třetí podlaží je již znatelně propletené rostlinnými pletivy a cesta do posleního patra je celá v pletivech zahalená.

Čtvrté a tedy poslední podlaží je již celé zabalené v rostlinném pletivu a hráč je zde vystaven pouze těm nejsilnějším nepřátelům. Na konci podlaží se nachází “srdce rostlin“ jehož zničením hráč ukončí hru.

# Tvorba Projektu

## Příklady tříd

### Třída Level Object

Obsah obrázku text

Popis byl vytvořen automatickyLevel Object je třída, definující jakýkoli objekt ve hře, který má svou danou pozici ve světě a vizuální podobu, obrázek, kterou je prezentován hráči.

Mezi její malé množství atributů patří desetinné hodnoty x a y, které reprezentují souřadnice pozice tohoto objektu ve scéně. Dalšími desetinnými hodnotami jsou screenX a screenY. Tyto vlastnosti objektu říkají programu, kde na obrazovce se daný objekt nachází. Hitbox každého objektu je na výšku postavený sloup a hodnota hitbox udává poloměr tohoto hitboxu. Hodnota size je zakomponována do algoritmu, který zajišťuje vykreslení bojektu na obrazovku a ovlivňuje ji tím způsobem, že prakticky určuje jak velký či malý objekt bude. Posledním atributem Level Objektu je myImage. Ten drží takový obrázek, který se má zrovna ukazovat hráči, protože objekty mohou mít více obrázků, které střídají na základě toho, zda například útočí, nebo jsou zraněni. Atribut myImage je poté proměnná, do které je uložen obrázek prezentující právě probíhající stav objektu.

Obsah obrázku text

Popis byl vytvořen automatickyKonstruktor třídy Level Objekt pouze najde pomocí třídy ResFileReader soubor ve zdrojové složce a získá si z toho informace, které potřebuje pro vytvoření nového objektu, což je například počáteční pozice a poloměr hitboxu.

Jedinými dvěma metodami této třídy jsou distToPlayer a distToPlayerTan. Obě metody vracejí hodnotu, která se rovná vzdálenosti objektu od hráče, ale ne tu stejnou vzdálenost a ne stejným způsobem. Metoda distToPlayer vrací jednoduše způsobem Pythagorova trojúhelníku vzdálenost objektu hráče. Hodnota je ale vrácená v nezmenšeném tvaru a při jejím použití je třeba nejprve vložit získanou hodnotu pod odmocninu. Metoda distToPlayerTan je je lehce odlišná. Tato metoda nemá sloužit k tomu, abychom zjistili jak daleko je tento objekt od hráče ve scéně, ale potřebujeme vědět jak daleko od něj je z pohledu kamery, abychom mohli poté příkladným matematickým způsobem vykreslit na obrazovku objekt správné velikosti. Jedná se vlasntě o metodu rozšiřující k metodě distToPlayer, akorát s tou výjimkou, že kdybych vytvořil rovinu na hráčském pohledu tak, aby směr pohybu byl normálovým vektorem pro tuto rovinu, poté metoda distToPlayerTan určuje vzdálenost od této roviny. Závislost na této funkci je zapříčiněna způsobem jakým vykresluji zdi, protože i u zdí používám vzdálenost od této roviny a ne vzdálenost od samotného bodu pohledu proto, aby při vykreslení obrazovky nevznikal tzv. efekt rybího oka.

Obsah obrázku text

Popis byl vytvořen automaticky

## Zpracování pohybu

Obsah obrázku text

Popis byl vytvořen automatickyPohyb hráče je velice jednoduchý. Program si zjistí s využitím hráčovy nastavené rychlosti a jeho úhlu pohledu, kam by se měl v příštím snímku hráč pohnout. Hráč se na tuto pozici pohne a proběhne kontrola prostředí, zda hráč nestojí ve zdi nebo se za procesu nedostal dovnitř cizího hitboxu. Pokud ano, je vypočítan nejbližší bod, na kterém může hráč stát. V případě tzv. sebratelných předmětů to hráče neposune, ale při kolizi je daný objekt ze současné hry odstraněn, a tudíž není třeba hráče nikam posouvat. Metoda performMovement má jeden parametr a tím je celočíselná hodnota a, která reprezentuje směr pohybu vůči otočení hráče ve stupních. Pokud se například chceme pohnout doleva, a nese hodnotu 90.

## Ovládání

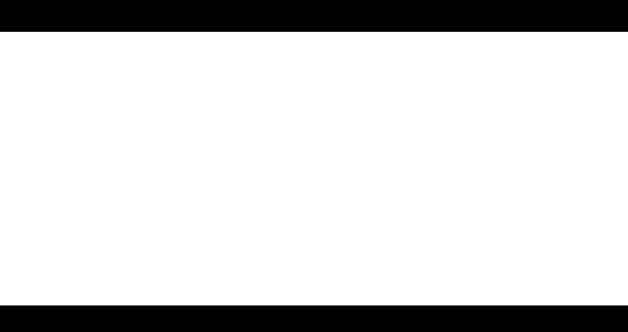
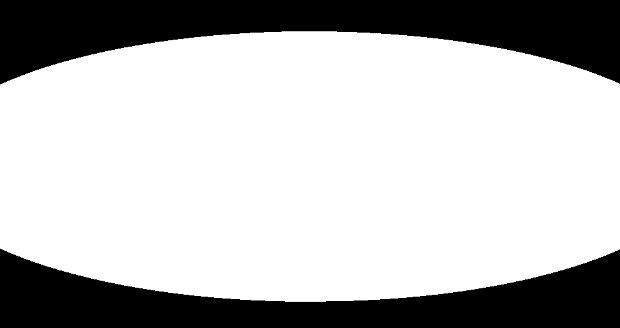
Jediné, co hráč ke hraní této hry potřebuje, je klávesnice. Ať už se jedná o hlavní menu, kde hráč jednoduše jenom dá jedním tlačítkem pokyn ke startu hry, nebo o samotnou hru, kde se hráč pomocí šipek a tlačítek WSAD pohybuje a pomocí klávesy Space střílí nepřátele, hra se ovládá pouze klávesnicí.

## Testování

-V procesu-

## Řešení problémových částí

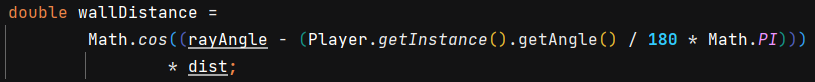
Největším problémem mé práce bylo odstranit tzv. rybí efekt, který se objevoval v důsledku způsobu použití renderovacího algoritmu.



Pohled na zeď s rybím efektem

Pohled na zeď bez rybího efektu

Ve hře jsem použil renderovací algoritmus na způsob hry Wolfenstein 3D, což znamená, že jsem za každý pixelový sloupec na obrazovce vyslal jeden paprsek do 2D prostoru a sledoval jeho trasu a zda se nestřetl se zdí. V případě, že nastala kolize se zdí, se nezíská barva jako u klasického Ray Castingu, ale hra si uloží vzdálenost hráče a bodu střetu a během vykreslovacího procesu, kde v každém pixelovém sloupci vybarví určitý počet pixelů, podle toho jak daleko daný paprsek doletěl. Čím větší vzdálenost, tím méně pixelů je vybarveno. Problém s rybím efektem nastával práve v procesu zpracovávání vzdálenosti, kterou paprsek urazil. Jak je patrné z levého obrázku, krajní paprsky urazili větší vzdálenost než došlo ke střetu se zdí, než paprsky ležící uprostřed. To dává smysl, neboť jestliže náš pohled směřuje kolmo na rovnou zeď, tak potom náš pohled vytváří jakýsi trojúhelník, kde krajní paprsky opravdu urazí větší vzdálenost. Takové prostředí ale působí docela nepřírozeně a tak bylo třeba se tohoto efektu zbavit tak, aby byli všechny paprsky jakoby sjednocené, reps. aby měli všechny paprsky stejnou vzdálenost, pokud je hráč otočen přímo proti rovné zdi, tak aby pohled vypadal stejně jako na obrázky vpravo. Toho jsem docílil jednoduše pomocí krátkého výpočtu, kde každý paprsek násobím funkcí cosinus na úhel, o který se liší od úhlu otočení hráče.



**Sources**

https://www.techradar.com/news/ray-tracing