ZÁPADOČESKÁ UNIVERZITRA V PLZNI

Fakulta aplikovaných věd

Předmět KIV/ZPG

semestrální práce na téma:

3D Maze

Autor: Michal Malík

Email: malikm@gapps.zcu.cz

Datum: 26. května 2025

1 Základní implementace

V této sekci se podíváme, jak fungují základní mechanismy naší hry.

1.1 Pohyb

Uživatel k pohybu využívá standardní kombinace WASD nebo ↑←↓→. Směr pohybu hráče je také závislý na směru kamery a na rychlosti, kterou se kamera může pohybovat.

1.2 Vytvoření scény

Základní rozložení se získává z textového souboru, kde různé znaky znamenají různé objekty.

- mezera, 'a' až 'n' - volný prostor
- 'o' až 'z' - zeď (obecně neprůchodné pole)
- '@' - startovní pozice pozorovatele. Vždy právě jeden na celé mapě.
- '*', '!' - světla
- 'A' až 'G' - dveře a vstupy tajných chodeb
- 'R' - rampa
- 'T' až 'Z' - předměty určené ke sběru
- '1' až '9' - teleport
- Prázdný řádek - odděluje od nastavení nového podlaží

Při inicializaci kamery se nastaví její pozice Y na hodnotu $1,70$ m, a její průměrná rychlost je nastavena na $1,4$ m/s.

Průběžně při vytváření scény se naplňuje i 2D pole booleanů, které uchovává informace o možných kolizích, a kde na ně můžeme narazit. Celá mapa je indexována od nuly a výš.

1.2.1 Základní stavební pilíře

Nejzákladnějšími prvky na scéně jsou objekty sloupů, které dohromady tvoří zeď a celkovou strukturu bludiště. Ty nabývají rozměru 2x2x3.
Druhým důležitým stavebním prvkem je podlaha, která se rozpíná po celé délce herního pole.

1.2.2 Dveře

Dveře mají zatím stejné rozměry jako sloupy. Jediné čím se od sloupů liší, je jejich možnost je otevírat klávesou E.

1.2.3 Sběratelské předměty

Sběratelské předměty zatím slouží pouze jako dekorativní doplněk. Na konci semestrální práce by jeho funkcionalita měla být taková, že když přes něj hráč projde, předmět zmizí a bude připsán uživateli do skóre.

1.3 Kolize

Ověřování kolize probíhá v pohybové funkci kamery, která jako parametr dostává také, už dříve zmíněné, 2D pole kolizí. Zároveň si dává i pozor na neviditelné stěny, v případě, že by chyběl objekt stěny.

1.3.1 Implementace

Na začátku kontroly si spočítáme, na jakou novou pozici bychom se dostali. Novou pozici nejdříve porovnáme s rozlohou mapy, aby se nám nestalo, že projdeme skrze neviditelnou stěnu.
Poté začneme nové pozice X a Z ověřovat zvlášť (abychom v případě kolize mohli sklouzávat po stěně). Obě pozice se porovnávají pomocí sousedních indexů a radiusu, který má za účel držet hráče dál od stěny, aby na ní nebyl nalepen a neviděl skrz ni. Pro lepší optimalizaci je dán break, který se provede, při srážce alespoň s jedním ze sousedů, aby nemusel kontrolovat další, jelikož by to bylo zbytečné.

1.4 Baterka

Implementace baterky je z větší části převzatá z OpenTK dokumentace. Směr a pozice baterky je ovlivňován pohybem kamery, jediný rozdíl je přidání deprese při úhlu o $2°$. Další rozdíl je v pozici baterky, kde je pozice na ose Y přesunuta, aby odpovídala $2,05$ metrům.

1.5 FPS Počítadlo

V hlavičce aplikace může uživatel vidět počet FPS, z důvodu testování správnosti výpočtu je zaplý V-sync.

1.5.1 Implementace

Máme vytvořenou frontu, ve které jsou uložené časové hodnoty jednotlivých snímků. Pokud je v nich hodnota větší než 1 sekunda, vyndaváme jej z fronty ven. Jako finální počet FPS slouží hodnota délky této fronty.

2 Textury

Práce s texturami je převzatá ze cvičení. Naši třídu Vertex jsme doplnili o atribut *Vector2 texCoords*, který v sobě uchovává souřadnice textury. Zároveň jsme si vytvořili nové shadery pro textury. Kde vertex shader získává z layoutu textové koordinace. Tu předává do fragmentu shaderu, který je ještě doplněn o uniformu difuzní složky textury.

2.1 Třída Texture

Třída `Texture` slouží k načtení obrázku ze souboru a vytvoření 2D textury v prostředí OpenGL. Při inicializaci:

1. Načte obrázková data ze souboru pomocí knihovny `StbImageSharp`.
2. Vytvoří texturu v OpenGL – vygeneruje ID, nahraje data do GPU a vygeneruje mipmapy.
3. Nastaví parametry textury (např. filtrování nebo opakování) podle zadaných nebo výchozích hodnot.
4. Umožňuje navázání textury na shader pomocí metody `Bind`, kde se specifikuje texturová jednotka a umístění uniformy.

Pro správu parametrů textury využívá pomocnou třídu **TextureSetting**, která obsahuje sadu běžných OpenGL nastavení jako výchozí hodnotu.

2.2 BetterObjLoader

Speciálně pro soubory *.obj* je vytvořená třída **BetterObjLoader**, která zpracuje standardní objektový soubor. Jako parametr přebírá cestu k danému souboru a pozici k objektu.

3 Drátěný model

Pro vizualizaci scény v drátěném režimu (tzv. *Wireframe*) se používá tzv. **LightMode verze** rendru, která má na starosti vykreslení objektů v podobě pouze jejich hran.
Renderování probíhá ve třech krocích:

1. Nejprve se všechny objekty vyrenderují standardním způsobem (např. s plným osvětlením a texturami).
2. Následně se vyčistí `colorBuffer`, aby se odstranil barevný výstup, ale zachovala se hloubková informace (`depth buffer`).
3. Poté se objekty vykreslí znovu, tentokrát s nastaveným režimem `PolygonMode.Line`, což způsobí, že se zobrazí pouze jejich hrany (drátěný model).

Díky tomuto postupu zůstává zachováno správné respektování viditelnosti jednotlivých ploch — části objektů, které nejsou z pohledu kamery viditelné, zůstávají skryté i v drátěném režimu.

4 Minimapa

Třída minimapy dědí svou funkcionalitu od rodičovské třídy **Camera**. Nastavíme ji menší **Viewport**, aby se nacházela v levém horním rohu obrazovky. Dále ji nastavíme při inicializaci tak, aby se dívala shora dolů.

4.1 Render scény

Pro minimapu vykreslíme jen podlaží, ve kterém se právě nacházíme. Na závěr zavoláme metodu minimapy. Která ještě rendruje ikonu uživatele.

4.2 Rotace minimapy

Pokud nastavíme atribut minimapy *up* na hodnotu *front* naší hlavní kamery, znamená to, že „nahoru“ z pohledu minimapové kamery je to, kam se dívá hlavní kamera.

5 Sběratelské předměty

Sběratelské předměty jsou reprezentovány objektem sáčku peněz. Ke každému předmětu můžeme dojít a sebrat. V názvu okna vidíme informace o stavu skóre, v podobě: <`pocet_sesbiranych_predmetu`> /<`pocet_vsech_predmetu`>.

Pokud uživatel sesbírá všechny `collectibles`, aplikace se vypne.

6 Teleport

Objekt teleportu zakreslíme do *.md* souboru mapy jako charaktery 1 až 9. Tato čísla se používají zároveň jako klíč do slovníku, který ukládá vždy prvního člena dvojice teleportů. To znamená, aby člověk zprovoznil teleport ve hře, musí mít dvojici dvou stejných čísel na mapě. (Pzn. teleport nefunguje mezi patrově).

6.1 Funkcionalita

Program se podívá přes všechny teleporty v patře, ve kterém se hráč nachází. Pokud se hráč nachází v jeho prostoru aktivace, začne probíhat 5 sekundová akce, kdy hráč musí zůstat stále v daném radiusu. Postup je také značen postupným zbělováním obrazovky.

To se vytváří tak, že si uchováváme hodnotu progresu teleportace. Tu předáme jako uniformu do fragment shaderu. Kde se před nastavením finální barvy ještě vypočítá přechod finální barvy a jedničkového vektoru (čistá bíla) na bázi předaného progresu.

7 K-Patro

Na závěr jsem se rozhodl implementovat úlohy pro podporu více podlaží. Při řešení tohoto problému jsem myslel na 3 podproblémy.

1. Jak se uživatel dostane do jiného patra
2. Vytvoření díry, aby uživatel viděl do dalšího patra
3. Optimalizace při renderování

7.1 Rampa

Podpůrný objekt rampy pomáhá hráči při přesunu mezi patry. Má vlastní detekci kolize, když uživatel přijde z jiné strany než má (ze stran, a kontrola správné strany v závislosti na patře, kde se hráč nachází).

Pro změnu pozice Y využívám hodnotu progrese, získanou z aktuální pozice na rampě. Pokud dojdu na konec rampy, změním hráčovo hodnotu k, která zachovává informaci o aktuálním patře.

Rampa se nevykreslí, pokud se nachází v posledním patře. Jelikož je zde zbytečná.

7.2 Podlaží

Dále mě čekala změny třídy **Plane**, kterou jsem přetvořil z jedné velké plochy na sbor dlaždiček o rozměrech 2x2. Při vytvoření rampy vytváříme zároveň i podlaží (z toho vyplývá jedno podlaží, jedna rampa). Zde pošleme souřadnice rampy, aby se zde dlaždička nevykreslila.

7.3 Optimalizace při renderování

Abych programu ušetřil při velkém množství pater. Vykresluji patra pouze 3. Aktuální patro, kde se nacházím, a patra pode mnou a nade mnou. To zařídí abych přes díry mezi podlažím mohl vidět objekty co se nachází v jiném patře.

8 Animace

Ve hře se nachází dva druhy animace (pro dveře a teleport se sběratelskými předměty). Tyto animace jsou vytvořené pouze pomocí transformací.

8.1 Dveře

Pokud uživatel otevírá nebo zavírá dveře, provede se animace posunutí, jako ve starých Wolfenstein hrách. Při načtení mapy se program podívá kolem inicializovaných dveří, kde se kolem něj vyskytuje zeď. Podle toho si nastaví polohu, do které se chce otevírat.

Při spuštění akce otevírání dveří se program podívá na pozici, do které má vklouznout a pomocí deltaTime zjistí progres posunutí. Pomocí hodnoty progresu spočítáme aktuální pozicový vektor pro dveře.

8.2 Teleport a předměty

Zde probíhá animace rotace, pro teleport a předmět dvě různé rychlosti otáčení. Zároveň pomocí sinus funkce a deltaTime hodnoty provádíme animaci vznášení.

9 Klávesové akce

9.1 WASD

Základní klávesy pro pohyb postavy. Zároveň řeší detekci se zdmi a zavřenými dveřmi.

9.2 Escape

Ukončí běžící proces videohry.

9.3 F

Má na starost vypínání a zapínání baterky. Když je baterka vypnutá, posíláme do fragment shaderu nulový vektor jako barvu světla.

9.4 E

Akční tlačítko pro sbírání sběratelských předmětů a otevírání/zavírání dveří. Po stisknutí klávesy zkontrolujeme všechny dveře a předměty v patře, ve kterém se právě nacházíme (optimálnější, než procházet úplně všechny po celé hře)

9.5 L

Zapínání LightMode verze, která má na starost renderovat mapu v drátěném modelu. Ten se vykresluje způsobem, že nejdříve se vyrendrují objekty normálním způsobem. Poté vyčistí colorBuffer a znova renderuje objekty, pouze s nastavením **PolygomMode.Line**.
Díky tomu zachováme respektování viditelnosti jednotlivých stran.

10 Závěr

V rámci práce na projektu bylo přínosné se seznámit se základními principy počítačové grafiky, především v kontextu moderního programování s využitím knihovny OpenGL a její .NET obálky OpenTK.

Díky praktické implementaci bylo možné pochopit nejen samotné techniky renderování (jako například práce s texturami, buffery, shader programy nebo drátěný model), ale také problémy běžné ve vývoji videoher. Mezi ty patří například detekce kolizí, správa aktivních objektů ve scéně nebo obecná logika interakcí hráče s prostředím.

Tato zkušenost přispěla k hlubšímu porozumění jak technické stránce grafického výstupu, tak i praktickým aspektům návrhu a implementace herních systémů.