Stredná priemyselná škola elektrotechnická  
Hálova 16, 851 01 Bratislava

**<Názov práce>**

KOMPLEXNÁ ODBORNÁ MATURITNÁ PRÁCA

Bratislava, 2024 4.D <Meno Priezvisko>

Stredná priemyselná škola elektrotechnická  
Hálova 16, 851 01 Bratislava

**<Názov práce>**

KOMPLEXNÁ ODBORNÁ MATURITNÁ PRÁCA

**Študijný odbor**: 2573M programovanie digitálnych technológií  
**Konzultant**:

Bratislava, 2024 4.D <Meno Priezvisko>

**Čestné vyhlásenie**

Vyhlasujem, že prácu stredoškolskej odbornej činnosti na tému <autor napíše názov svojej práce>, som vypracoval samostatne, s použitím uvedených literárnych zdrojov. Prácu som neprihlásil a ani neprezentoval v žiadnej inej súťaži, ktorá je pod gestorstvom MŠVVaM SR. Som si vedomý dôsledkov, ak uvedené údaje nie sú pravdivé.

.........................................................

V Bratislave, <dd. mm. rrrr> <Meno a priezvisko autora/autorov>

**Poďakovanie**

Rád by som sa touto cestou poďakoval svojmu <školiteľovi> za prístup a odborné rady. Tiež by som sa rád poďakoval <spoločnosti> za finančnú podporu pri realizácii praktickej časti mojej práce.

**Obsah**

[0 ÚVOD I](#_Toc189913240)

[1 Postup tvorby počítačovej hry II](#_Toc189913241)

[1.1 Definovanie základných parametrov II](#_Toc189913242)

[1.2 Vývojové metody II](#_Toc189913243)

[1.3 Produkcia a vývoj III](#_Toc189913244)

[2 Analýza a porovnanie technológií na tvorbu počítačových hier III](#_Toc189913245)

[3 Návrh dizajnu hry VI](#_Toc189913246)

[3.1 Základné informácie VI](#_Toc189913247)

[3.2 Herná slučka (Game Loop) VII](#_Toc189913248)

[3.3 Herný svet a vizuálny štýl VII](#_Toc189913249)

[4 Technický návrh (Technical Design Document - TTD) VII](#_Toc189913250)

[5 Art Design Document (ADD) X](#_Toc189913251)

[5.1 Art Design Document X](#_Toc189913252)

[6 Pripravte technólógie na projekt v unity XI](#_Toc189913253)

[6.1 Github XI](#_Toc189913254)

[6.2 Sourcetree XII](#_Toc189913255)

[7 Naprogramujte hernu mechaniku XIII](#_Toc189913256)

[7.1 Pohyb XIII](#_Toc189913257)

[7.2 Lezenie po stenách XIV](#_Toc189913258)

[7.3 Zlepšenia a návrhy na optimalizáciu XV](#_Toc189913259)

[7.4 Tlačenie objektov XVI](#_Toc189913260)

[8 Integrujte mechanizmus prepínania stavov rozhodovacej mechaniky hry XVII](#_Toc189913261)

[8.1 Rozdelenie tried a ich funkcie XVII](#_Toc189913262)

[9 Vytvorte systém na vykresľovanie grafických detailov XIX](#_Toc189913263)

[10 Navrhnite systém renderovania objektov XX](#_Toc189913264)

[11 Pripravte analýzu trhu XXII](#_Toc189913265)

**Zoznam skratiek, značiek a symbolov**

<skratky zoradené v abecednom poradí>

**Zoznam tabuliek, grafov a ilustrácií**

<Zoznam skratiek, značiek a symbolov>

1. ÚVOD

Rozhodol som sa zvoliť túto tému, pretože som chcel prekročiť svoju komfortnú zónu a venovať sa niečomu novému. Mojím cieľom bolo naučiť sa nové zručnosti a posunúť sa ďalej v oblastiach, ktoré som doteraz nepoznal. Táto výzva mi pomohla rozvíjať nielen technické schopnosti, ale aj kreatívne myslenie a riešenie problémov.

V tomto projekte som sa venoval vývoju vlastného pohybového systému, práci s animáciami a prepojeniu postavy s kostrou (rigging). To znamená, že som sa musel naučiť, ako vytvoriť postavu, animovať jej pohyby a zabezpečiť, aby sa tieto animácie správne implementovali v hernom prostredí.

Okrem toho, že som pracoval v programe Unity, som sa oboznámil aj s ďalšími nástrojmi, ako sú **Sourcetree**, **Blender** a **GitHub**. Každý z týchto programov zohral dôležitú úlohu vo vývoji mojej hry:

* **Unity**: Slúžilo ako hlavná platforma na vytvorenie herného prostredia a implementáciu pohybov a animácií.
* **Blender**: Použil som na modelovanie postáv a ich prípravu pre animácie (rigging).
* **Sourcetree a GitHub**: Tieto nástroje mi pomohli pri riadení verzií projektu a spolupráci na kóde. Naučil som sa, ako efektívne spravovať svoje súbory, zaznamenávať zmeny a uchovávať bezpečné zálohy projektu.

Vďaka kombinácii týchto programov a nástrojov som bol schopný vytvoriť vlastnú hru od základov. Tento projekt mi nielen rozšíril obzory, ale tiež mi ukázal, že vyjsť z komfortnej zóny môže byť veľmi obohacujúce.

# Postup tvorby počítačovej hry

Tvorba počítačovej hry je komplexný proces, ktorý si vyžaduje detailné plánovanie a koordináciu medzi rôznymi odborníkmi. Tento proces môžeme rozdeliť do niekoľkých základných krokov, pričom každý z nich hrá kľúčovú úlohu pri vytváraní kvalitného produktu.

## Definovanie základných parametrov

Na začiatku projektu sa určuje základná koncepcia hry. Tu sa zvažujú aspekty ako:

* Cieľová veková kategória: Je dôležité určiť, pre koho je hra určená. Inak sa navrhuje hra pre deti, tínedžerov alebo dospelých
* Žáner hry: Napríklad First Person Shooter (FPS), RPG, stratégia alebo simulácia
* Herná estetika: Štýl grafiky, farebné palety (napr. RGB) a celkový vizuálny dizajn

Na tejto úrovni sa tiež skúma trh a konkurencia. Analýza konkurenčných titulov pomáha identifikovať, v čom by sa hra mohla odlíšiť a priniesť hráčom nový zážitok.

Stanovenie herných mechaník: Herné mechaniky určujú, čo hráč v hre robí. Sem patrí pohyb, bojové systémy, interakcia s prostredím alebo riešenie hádaniek. Je dôležité presne definovať: Aké budú hlavné činnosti hráča? Aký bude cieľ hry a spôsob jeho dosiahnutia? Aké výzvy a prekážky sa objavia počas hry?

Ak sa projekt riadi princípmi agilného vývoja, herné mechaniky a ďalšie aspekty môžu byť postupne dopĺňané a upravované podľa potreby.

## Vývojové metody

Pri tvorbe hier sa často využívajú dva hlavné systémy riadenia projektov: Agile a Waterfall. Každý z nich má svoje výhody a nevýhody.

Agile systém je najčastejšie používaný systém, pretože umožňuje dynamický a flexibilný prístup. Práca sa delí na dvojtýždňové cykly nazývané šprinty. Po každom šprinte sa tím stretáva a hodnotí: Čo bolo dokončené. Čo je potrebné zmeniť, pridať alebo odstrániť

Agile poskytuje výhodu v prípade, že sa počas vývoja objavia nečakané problémy alebo potreba zmeny. Je ideálny na projekty, kde sa produkt postupne vyvíja a zlepšuje na základe spätnej väzby.

Waterfall systém je tradičný systém, kde sa všetky kroky plánujú a určujú vopred. Tento systém sa najčastejšie používa pri projektoch, kde zmeny počas realizácie nie sú možné alebo sú veľmi komplikované, napríklad pri stavbe koncertných hál.

Nevýhodou Waterfall systému je jeho nízka flexibilita. Ak sa počas vývoja objavia komplikácie, môže to spôsobiť veľké problémy, ktoré narušia celý harmonogram.

## Produkcia a vývoj

Po stanovení parametrov a výberu metodiky nasleduje samotný vývoj. Tento proces zahŕňa: Programovanie: Implementácia herných mechaník, umelej inteligencie a fyzikálnych systémov. Grafický dizajn: Tvorba herných modelov, textúr, animácií a používateľského rozhrania. Zvukový dizajn: Výber hudby, tvorba zvukových efektov a dabingu. Testovanie: Neustále testovanie hry na odhaľovanie chýb a zabezpečenie kvalitného herného zážitku. Po vytvorení prototypu sa hra zvyčajne sprístupní na testovanie menšej skupine hráčov (tzv. beta testovanie). Na základe ich spätnej väzby sa môžu upravovať mechaniky, odstraňovať chyby a dolaďovať detaily. Poslednou fázou je vydanie hry na trh. Tu sa kladie dôraz na marketing, distribúciu a podporu hráčov po vydaní. Niekedy sa po vydaní vydávajú aj aktualizácie, ktoré hru ďalej zlepšujú alebo pridávajú nový obsah.

# Analýza a porovnanie technológií na tvorbu počítačových hier

Tvorba počítačových hier je komplexný proces, ktorý vyžaduje kombináciu kreativity, technických znalostí a vhodných nástrojov. Na trhu existuje široké spektrum technológií a platforiem, ktoré môžu byť použité na vývoj hier. Medzi najpoužívanejšie patria herné engine ako Unity, Unreal Engine, Godot a CryEngine. Tento dokument analyzuje a porovnáva ich výhody, nevýhody a vhodnosť pre rôzne typy projektov.

**Unity** je jednou z najpopulárnejších platforiem na tvorbu hier. Podporuje vývoj pre rôzne platformy, vrátane PC, mobilných zariadení, konzol a virtuálnej reality.

*Výhody:*

* Jednoduché užívateľské rozhranie, vhodné pre začiatočníkov
* Veľká knižnica nástrojov a pluginov
* Aktívna komunita a široká dostupnosť tutoriálov
* Podpora skriptovania v jazykoch C# a JavaScript

*Nevýhody:*

* Môže byť menej efektívne pri ťažkých 3D projektoch v porovnaní s inými enginmi
* Licenčné poplatky pri vyšších príjmoch z hry.

**Unreal Engine** je využívaný najmä na vývoj graficky náročných hier. Poskytuje vysoko kvalitné vykresľovanie a je preferovaný pri AAA projektoch.

*Výhody:*

* Realistická grafika vďaka technológiám ako Lumen a Nanite
* Prístup k zdrojovému kódu
* Podpora vizuálneho skriptovania cez Blueprint system
* Vhodné pre veľké a ťažké projekty

*Nevýhody:*

* Vyššia náročnosť na hardvér.
* //treba upravit// Prudko stúpajúca krivka učenia pre začiatočníkov.

**Godot** je open-source herný engine, ktorý je často preferovaný pre indie projekty a začiatočníkov.

*Výhody:*

* Bezplatný a open-source model
* Flexibilné nástroje na tvorbu 2D hier
* Jednoduchý skriptovací jazyk GDScript, podobný Pythonu
* Možnosť vývoja pre viaceré platformy

*Nevýhody:*

* Obmedzené možnosti pri ťažkých 3D projektoch.
* Menšia komunita a menej dostupných zdrojov v porovnaní s Unity alebo Unreal Engine.

**CryEngine** je známy pre svoje schopnosti realistického vykresľovania, najmä pri otvorených svetoch.

*Výhody:*

* Pokročilé vykresľovanie a fotorealistická grafika.
* Nástroje na tvorbu otvorených svetov
* Výborný fyzikálny engine.

*Nevýhody:*

* Komplexný a náročný na učenie
* Nižšia podpora komunity a menej dokumentácie
* Licenčné poplatky

**Porovnanie a odporúčania:** Pre začiatočníkov a menšie projekty je najlepšou voľbou Unity alebo Godot vďaka ich jednoduchej používateľskému rozhraniu a dostupnosti zdrojov. Unity je univerzálnejšie, zatiaľ čo Godot exceluje v 2D projektoch.

Pre veľké a graficky náročné hry je Unreal Engine jasnou voľbou, vďaka svojim schopnostiam realistického vykresľovania. CryEngine je však vhodný pre špecifické potreby otvorených svetov.

Pri výbere technológie je potrebné zvážiť typ hry, rozpočet, tím a cieľové platformy. Každý engine má svoje špecifiká a poskytuje možnosti na realizáciu kreatívnych nápadov.

# Návrh dizajnu hry

Dizajn hry(GDD, Game designe dokument): sú v ňom základne informácie ako je napr.(Názov hry, veková kategória, cieľová platforma) ale aj veci hlbšie do hry ako sú mechaniky (Game mechanics), herná slučka(Game loop) o čom vlastne hre ide a o čom je tá hra.

**Game Design Document (GDD)**

## Základné informácie

* **Názov hry: "Project-N"**
* **Žáner:** Akčná adventúra s prvkami hádaniek a platformingu
* **Cieľová platforma:** PC
* **Veková kategória:**12+ (obsahuje mierne násilie a zložitejšie hádanky)
* **Cieľová skupina:** Hráči, ktorí majú radi príbehové hry, dynamický pohyb a kreatívne riešenie problémov.

O čom je hra? **" Project-N "** vo svete kde tvár je všetko vám zlá organizácia zobrala tvár tak že vám ju dali do železnej helmy ktorá nejde dať dole. Jedného dna za rozhodnete že chcete zistiť pravdu prečo to urobili a vlastne kvôli čomu je tvár tak dôležitá. A však nebude to ľahké vaša cesta bude náročná budú vás chcieť umlčať navždy ale vy chcete zisti pravdu o tomto svete bez tváre. Cieľom hry je si dať dole vašu železnú helmu a zistí pravdu o tom to svete no avšak je na vás či ju poviete všetkým alebo ju navždy zničíte a bude to pokračovať naďalej.

**Herné mechaniky (Game Mechanics)**

**Manipulácia s objektamy:** **Zdvihanie a puživanie objektov:** Hráč môže interagovať z objektami takže ich vie presunúť alebo použiť na to aby sa dostal do ďalšej časti. **Strkanie objektov:** Niektoré objekty sú moc ťažké aby ich hráš zdvihol a preto ich môže len posúvať po X a Z kordinátov. **Interagovanie s okolím:** Hráč môže interagovať aj s vecami ako je napríklad poklop šachty aby sa dostal do ďalšej časti a však s týmito objektami môže len raz interagovať

**Pohybové mechaniky:** Parkour systém: behanie ,skoky, visenie z kraju objektu, pohybovanie na kraji objektu.

**Hádanky:** Logické hádanky založené na udalosti príbehu

**Zberateľské predmety:** Hráč môže zbierať papieriky ktoré ho budú sprevádzať počas hry. Tieto papieriky sa môžu nachádzať aj v skrytých miestach a preto hráč musí byt pozorný a vynaliezaví aby ich našiel

## Herná slučka (Game Loop)

1. **Prieskum:** Hráč objavuje prostredie, zbiera predmety, hľadá tajomstvá ktoré sú skryté po mape.
2. **Riešenie problémov:**  Hráč bude musieť pristupovať ku problémom s rozumom aby ho nechytili
3. **Boj:**  Hráč nemá žiaden prístup ku útokom je čisto odkázaní na to aby ho nechytili. Budem musieť byt rýchlejší ako nepriatelia a v tom mu môže pomôcť prostredie v ktorom sa nachádza
4. **Progresia:** Čím ďalej v príbehu tak sa hra stane ťažšia budú nové hlavolamy a noví nepriatelia ktorým sa bude ťažšie vyhýbať ako napríklad na začiatku hry

## Herný svet a vizuálny štýl

* **Prostredie:** Tmavý svet s rozpadnutými budovami v ktorých kedysi bývali ľudia.
* **Vizuálny štýl:** Kombinácia realistických textúr s nadprirodzenými silami (napr. lesk vody, Manekýni ktoré sa hýbu).
* **Nepriatelia:** Robotickí strážcovia času, príšery vytvorené nestabilitou času a bossovia strážiaci fragmenty artefaktu.

# Technický návrh (Technical Design Document - TTD)

**Úvod do technického návrhu**

Tento dokument popisuje technickú architektúru hry "Project-N". Zahŕňa popis herných systémov, spôsob ich implementácie, interakcie medzi nimi a riešenia technických problémov, ktoré sa objavili počas vývoja.

1. **Engine: Unity – využívaný pre jeho flexibilitu, podporu pre multiplatformové hry a robustné nástroje na spracovanie fyziky, animácií a osvetlenia.**
2. **Programovací jazyk: C# – hlavný jazyk pre herné skripty a spracovanie logiky.**
3. **Verzovacia kontrola: GitHub na správu kódu a Sourcetree ako GUI pre Git.**
4. **Grafické nástroje: Blender na tvorbu modelov, Substance Painter na textúrovanie a Photoshop na 2D prvky ako UI a textúry.**

Architektúra hry je modulárna, aby bola umožnená jednoduchá údržba a rozširovanie hry.

**Rozdelená je na tieto hlavné moduly:**

**Pohybový systém**

1. **PlayerController.cs: Zodpovedá za pohyb hráča. Obsahuje logiku pre behanie, skoky, parkour (zavesenie na kraje objektov), pohyb po hrane a prekonávanie prekážok.Jedna z výzievbola detekcia okrajov objektov. Riešenie: Použitie Raycastov na zisťovanie polohy a vzdialenosti okrajov. Ak hráč interaguje s krajinou, kontroluje sa, či sa nachádza v dosahu zavesenia.**
2. **Animácie: Unity Animator Controller spravuje prechody medzi stavmi ako „Idle“, „Run“, „Jump“, „Climb“. Blend Tree: Používa sa na plynulý prechod medzi animáciami v závislosti od rýchlosti pohybu.**

**Systém manipulácie s objektmi**

1. **ObjectInteraction.cs: Detekcia objektov v dosahu pomocou Colliderov a interakcia na základe vstupu hráča (napr. zdvihnutie alebo strkanie objektov). Objekty sú označené pomocou vlastnosti „InteractiveObject“, ktorá obsahuje informácie o type interakcie (zdvihnúť, strčiť, otvoriť).**
2. **Fyzika objektov: Pre manipuláciu s objektmi sa využíva Rigidbody komponent. Väčšie objekty zaberú 2 ruky namiesto jednej**

**D. Umelá inteligencia nepriateľov (AI)**

**Využíva sa presnejšie Finate state machine ktorý bude rozpísani neskôr**

1. **EnemyAI.cs: Nepriatelia majú tri hlavné stavy: Patrol, Chase, Search. Patrol: Nepriatelia sledujú preddefinované cesty. Chase: Ak hráča spozorujú, začnú ho prenasledovať. Search: Ak hráč zmizne z dohľadu, hľadajú ho v okolí.**
2. **Detekčný systém: Používa kombináciu Raycastov na zisťovanie polohy hráča.**
3. **Jenda z výziev bola optimalizácia pre viac nepriateľov v jednej scéne.**

**E. Herná slučka a progresia**

1. **GameManager.cs: Spravuje globálny stav hry (napr. stav misií, herný progres). Obsahuje logiku na zvyšovanie obtiažnosti (napr. pridanie nových nepriateľov, komplexnejšie hádanky).**
2. **LevelDesign: Prostredie je rozdelené na menšie úseky (scény), ktoré sa načítavajú dynamicky podľa postupu hráča. Výzva: Hladké načítavanie scén bez prerušovania hry. Riešenie: Použitie „additive loading“ v Unity na načítavanie častí scény na pozadí.**

**Optimalizácia a výkon**

1. **Level of Detail (LOD): Použitie rôznych úrovní detailov pre objekty podľa vzdialenosti od kamery.**
2. **Profily osvetlenia: Použitie predpočítaných svetelných máp (lightmaps) na statické objekty.**
3. **Finnate state machine: Funguje na báze animátoru takže bežia veci len ktoré majú. Lepšie sa debuguje a hladjú chyby. Lepši preformence**

# Art Design Document (ADD)

## Art Design Document

Art Design Document slúži ako vizuálny základ pre hru **"Project-N"**. Popisuje vizuálny štýl, estetiku a umelecké smerovanie hry. Tieto prvky vytvárajú pohlcujúci herný svet a zároveň podporujú naratívny tón príbehu. Primárna inšpirácia bola dystopické a post-apokalyptické prostredie so zreteľnou surrealistickou atmosférou.Kombinácia realistických prvkov (základy architektúry a textúry) s abstraktnými a nadprirodzenými efektmi (napr. svetelné javy, anomálie). Farebná paleta tmavé, studené odtiene: sivá, oceľová modrá, čierna. Kontrastné akcenty: krvavo červená, neónová žltá a jemne tyrkysová pre zvýraznenie dôležitých prvkov.

**Prostredie (Environment Art)**

**Rozpadnuté mestské oblasti:** Budovy sú opustené, popraskané, zarastené machom a vegetáciou. Použitie opotrebovaných textúr s detailmi, ako sú grafity a rozbité okná. **Podzemné komplexy:** Mechanické a industriálne oblasti s temnými chodbami, blikajúcimi svetlami a ruchom strojov. **Detaily:** Prostredie je navrhnuté tak, aby odrážalo príbeh: absencia tvárí je metaforicky vyjadrená cez rozmazané a neúplné obrazy na stenách, zničené sochy alebo zrkadlá, ktoré neodrážajú obraz.

**Postavy (Character Art)**

Hlavná postavanosí železnú helmu bez výrazných rysov tváre, ktorá je hrubá a robustná, no zároveň nesie jemné detaily ako škrabance a stopy po boji. Oblečenie je jednoduché, praktické, v tmavých tónoch. Animácie pohybu sú plynulé, no zdôrazňujú váhu helmy a boj o prežitie. Strašiak je postavený z dreva, s ručne vyrezávanými detailmi, ktoré pôsobia starodávne a strašidelne. Má hlavu bez tváre, ktorá pripomína prázdnu masku, čo zdôrazňuje tematiku hry. Na tele má zavesené roztrhané látky, ktoré sa pri pohybe vlajú, čím pridávajú dynamiku. Kolieska sú staré a pískajú pri každom pohybe, čo vytvára napätú atmosféru.

**Technická realizácia art dizajnu**

**Modelovanie:** Použité nástroje: Blender na tvorbu 3D modelov. **Optimalizácia:** Nízky počet polygónov pre bežné objekty s použitím normal máp pre detaily. High-poly modely len pre bossov a kľúčové objekty. **Textúrovanie:** Substance Painter pre vytvorenie realistických, opotrebovaných povrchov (napr. hrdza na kovových povrchoch, praskliny na betóne). **Osvetlenie:** **Statické osvetlenie:** Pre pevné prostredie na zvýšenie výkonu. **Dynamické osvetlenie:** Použité pre pohyblivé objekty a anomálne zóny, kde svetlo hrá dôležitú úlohu vo vizuálnej prezentácii. **Efekty:** **Post-processing:** Použitie efektov ako Bloom (žiar svetiel), Depth of Field (rozmazanie vzdialených objektov) a Chromatic Aberration (farebný posun) na vytvorenie dramatickej atmosféry. **Častice:** Systém častíc pre jemné efekty ako padajúci prach, iskry alebo drobné anomálie.

# Pripravte technólógie na projekt v unity

## Github

Repozitár názov: Na to aby sme vedeli projekt dat na github potrebujem repozitár. Repozitár je miesto kde sa náš projekt ukladá. Môžeme ho nazvať ako chceme ale všetky znaky musia byť spolu to znamená že nemôže byť Projekt N ale musí byť Projekt\_N alebo Projekt-N.

Repository Description: Tuto sa môžu ale nemusia uviesť údaje o projekte ako je napr: o čom bude ten projekt

Repository Security: Reposidory môže byť verejné alebo súkromné. Verjené znamená že si ho vie každý naklonovať a pozerať jeho obsah inak povedáné je to open source.Private je presný opak projekt môžu vidieť len ľudia ktorý autor pozve aby na ňom robily.Nikto iný si tento projekt naklonovať iba ľudia ktorý boli pozvaný na tento projekt.

README: Je to súbor ktorý môže mať podobu txt,pdf,... .Tento súbor sa použiva na to aby oboznámil ľudí pred tím ako ho spustia čo a ako funguje. Nie je to povinný súbor ale pomocou tohto súboru vieme o čo sa jedná v tomto projekte.

.gitignore: Súbor .gitignore je dôležitý pri práci s Gitom, pretože určuje, ktoré súbory a adresáre Git nemá sledovať ani zahrnúť do verzionovania. To je obzvlášť užitočné pri projektoch v Unity, kde sa generuje množstvo súborov, ktoré nie sú potrebné (alebo sú dokonca nežiaduce) na verzionovanie, ako sú dočasné súbory, cache, a buildy.

License: V kontexte projektov, najmä softvérových, licencia určuje, aké práva a obmedzenia sa vzťahujú na používanie, distribúciu, modifikáciu alebo komerčné využitie vášho kódu. Pri vývoji projektu v Unity (alebo všeobecne v softvéri) je výber licencie dôležitý, aby bolo jasné, ako môžu ostatní používatelia alebo vývojári nakladať s vaším projektom.

Licencia definuje: Používanie či môže niekto váš kód používať, a ak áno, za akých podmienok. Distribúcia: Či môže byť kód zdieľaný alebo predávaný. Modifikácia: Či môže byť váš kód upravený alebo rozšírený. Komerčné využitie: Či môže byť váš kód použitý v komerčných projektoch. Autorské práva: Kto si zachováva práva na projekt a ako je potrebné autora uvádzať.

## Sourcetree

Sourcetree je vizuálny nástroj od spoločnosti Atlassian, určený na správu verzionovania pomocou Git. Ide o grafické používateľské rozhranie GUI , ktoré zjednodušuje používanie príkazov Git pre vývojárov, ktorí preferujú vizuálny prístup namiesto príkazového riadku.

**Prečo používať Sourcetree?**

1. Prehľadné zobrazenie verzií:Ponúka stromové alebo grafové zobrazenie commitov, vetiev a tagov. Umožňuje vizualizovať históriu zmien a ich vzťahy (merge, rebase atď.).
2. Jednoduché používanie:Užívateľsky prívetivé rozhranie na vykonávanie bežných operácií Git, ako sú commit, push, pull, merge a rebase.Odstraňuje potrebu manuálneho zadávania príkazov v termináli.
3. Podpora viacerých repozitárov:Umožňuje spravovať viacero projektov a repozitárov súčasne.
4. Pokročilé funkcie:Práca so submodulmi, stashing, interaktívny rebase a sledovanie zmien súborov.
5. Bezplatné použitie:K dispozícii zdarma pre komerčné aj nekomerčné účely.

**Hlavné funkcie Sourcetree**

Grafická reprezentácia verzií: Zobrazuje vetvy, commity, tagy a merge operácie v prehľadnom grafe. Pomáha lepšie pochopiť štruktúru projektu a jeho históriu. Práca s vetvami: Vytváranie nových vetiev, prepínanie medzi nimi a ich spájanie (merge) je jednoduché cez GUI. Možnosť sledovať konflikty a ich riešenie. Zmeny v súboroch: Prehľad zmien v jednotlivých súboroch pred a po commite. Zobrazenie rozdielov (diff) v grafickej podobe. Stash a pop: Dočasné odloženie zmien (stashing) a ich neskoršie obnovenie (pop). Umožňuje experimentovať bez ovplyvnenia aktuálnej vetvy. Remote operácie: Možnosť klonovania, pushovania a pullovania zmien do/z vzdialeného repozitára (napr. GitHub, Bitbucket).Zjednodušené spravovanie remote vetiev. Konflikty a ich riešenie:Intuitívne rozhranie na zobrazenie a manuálne riešenie konfliktov pri merge alebo rebase.

**Ako používať Sourcetree – Základný pracovný postup**

Naklonovanie repozitára: V Sourcetree vyberte Clone Repository a vložte URL vášho Git repozitára. Zvoľte cieľový priečinok na vašom lokálnom disku. Zmeny v súboroch: Pridajte nové súbory alebo upravte existujúce. Sourcetree automaticky identifikuje zmeny a zobrazí ich v sekcii Working Copy. Commit: Vyberte zmenené súbory, pridajte popis commitu a kliknite na Commit. Push: Po commite môžete svoje zmeny odoslať na vzdialený server pomocou Push. Pull: Na stiahnutie zmien z remote repozitára použite Pull. Riešenie konfliktov: Ak nastane konflikt, Sourcetree vám ho vizuálne zobrazí. Použite zabudovaný editor alebo externý nástroj na ich vyriešenie.

**Výhody a nevýhody Sourcetree**

**Výhody:**

* Intuitívne rozhranie pre začiatočníkov.
* Vizualizácia histórie a vetiev.
* Podpora pokročilých operácií (rebase, stash, submoduly).
* Bezplatné použitie.

**Nevýhody:**

* Môže byť pomalé pri veľkých repozitároch.
* Pre pokročilých používateľov terminál ponúka viac flexibility.

# Naprogramujte hernu mechaniku

## Pohyb

Táto trieda Movemnt implementuje komplexnú hernú mechaniku pohybu hráča s mnohými funkciami, ako sú chôdza, beh, skákanie, plazenie, interakcia s objektmi, a prepojenie s animáciami. Mechanika je modulárna, čo umožňuje jednoduchú integráciu do existujúceho systému.

Pohyb hráča: Pohyb je riadený pomocou CharacterController v závislosti od vstupov hráča. Rozlišuje medzi chôdzou, behom a plazením. Obsahuje vyhladzovanie rýchlosti pomocou Mathf.Lerp, čo vytvára plynulý pohyb. Skákanie a gravitácia: Umožňuje hráčovi skákať, ak je na zemi. Ak hráč nie je na zemi, aplikuje sa gravitácia, čo simuluje realistický pád. Interakcia s objektmi: S využitím SphereCastNonAlloc kontroluje objekty, ktoré sú v dosahu interakcie. Podporuje použitie rozhrania IInteractable, čo umožňuje ľahké rozšírenie o ďalšie typy interakcií. Animácie: Prepojené s Animator pre synchronizáciu pohybu s animáciami. Animácie zahŕňajú: skákanie, pád, plazenie, pristátie, a bežné pohybové animácie. Crouch Mechanika: Plazenie sa prepína pomocou klávesy Left Control. Pri plazení môže byť implementovaná ďalšia logika, ako napríklad zmena výšky hráča. Prechod medzi stavmi: Táto trieda dedí z PlayerState, čo naznačuje použitie stavového automatu na správu rôznych stavov hráča. Metódy CanEnter, CanExit, OnEnter, a OnExit definujú správanie pri vstupe a výstupe zo stavu.

**Možné vylepšenia a optimalizácie**

Optimalizácia Raycastov: Raycasty sú intenzívne na výpočty, najmä ak sú volané vo vysokej frekvencii. Uistite sa, že sú používané iba v prípade potreby (napr. pri stlačení klávesy na interakciu). Zjednodušenie interakcie: Momentálne používa SphereCastNonAlloc s pevnou veľkosťou poľa výsledkov (5). Ak je potrebné pracovať s viacerými objektmi, zvážte dynamické zväčšovanie. Pohybová fyzika: Aplikujte fyziku len v prípade, keď je hráč vo vzduchu (napr. kontrolou characterController.isGrounded). Animácie: Prepojte viac animácií s parametrami, ako je rýchlosť alebo smer pohybu, pre plynulejšie prechody. Metóda Update je veľká a ťažko čitateľná. Rozdeľte ju na menšie metódy, napríklad: HandleMovement(), HandleJump(),HandleInteraction() Parameterizácia pohybu: Namiesto pevne zadaných hodnôt (napr. gravity, jumpSpeed) ich presuňte do konfiguračného súboru alebo editoru Unity ako vlastnosti. Crouch funkcia: Pri plazení pridajte zmenu výšky CharacterController pre realistickú mechaniku (napr. zmena height a center).

## Lezenie po stenách

Trieda Climbing implementuje komplexnú hernú mechaniku pre lezenie, vrátane interakcií s rukami, animácií, detekcie prostredia a pohybu. Tento systém je modulárny, umožňuje ľahkú integráciu do existujúceho systému a poskytuje flexibilitu v prechode medzi rôznymi stavmi hráča.

**Prepis a popis triedy Climbing**

Trieda Climbing je zodpovedná za implementáciu pohybu hráča počas lezenia. Implementuje funkcie ako interakcia s objektmi (napr. rukami), animácie, detekcia prostredia, pohyb medzi pozíciami a prechod medzi rôznymi stavmi.

1. Pohyb rukami

Systém detekuje pozíciu ruky (pravá aj ľavá) pomocou Raycast a nastavuje ich pozíciu v 3D priestore. Ak hráč drží ruku nad určitou vzdialenosťou od cieľového bodu, ruka sa pohybuje k tomuto bodu. Ruky sú animované pomocou archového pohybu, ktorý simuluje prirodzený pohyb počas lezenia.

2. Interakcia s prostredím:

Využíva sa Raycast na detekciu rôznych bodov (napr. na rohu alebo na zemi), ktoré ovplyvňujú pohyb hráča. Hráč sa môže priblížiť k zisteným bodom a vykonať plynulé prechody medzi týmito bodmi, čím sa simulujú realistické pohyby počas lezenia (napr. chytanie sa za hrany).

3. Animácie a Rigging:

Animácie sú synchronizované s pohybom postavy pomocou Animator. Napríklad pri držaní rúk sa spustí animácia, ktorá vizualizuje držanie a pohyb. Systém využíva Rig na riadenie váhy rúk počas pohybu a interakcie s objektmi.

4. Prechod medzi stavmi (Stavový automat):

Tento systém využíva dedičnosť z triedy PlayerState, čo naznačuje použitie stavového automatu na správu rôznych stavov hráča (napr. lezenie, pohyb po zemi, atď.). Metódy CanEnter, CanExit, OnEnter, a OnExit definujú správanie pri vstupe a výstupe zo stavu.

## Zlepšenia a návrhy na optimalizáciu

**Optimalizácia Raycastov**

Problém: Použitie Raycast vo vysokých frekvenciách môže byť výpočtovo náročné, najmä pri interakciách s rukami. Riešenie: Pred použitím Raycast overte, či je potrebné vykonať detekciu (napr. na základe vstupov od hráča). Vylepšenie: Použiť Physics.RaycastNonAlloc na zníženie alokácie pamäte pri detekcii.

**Zjednodušenie interakcie**

Problém: Kód používa pevné hodnoty pre vzdialenosti pri interakcii s objektmi (napr. 3m pre pravú ruku). Riešenie: Parameterizujte tieto hodnoty do konfiguračných premenných, ktoré môžu byť nastavené v Unity editore.

**Prechod medzi pozíciami pomocou Lerp a Bézierovej krivky**

Problém: Použitie Bézierovej krivky pre pohyb je skvelé, ale môže byť ťažké udržať plynulosť pohybu pri rôznych výškach a uhloch. Riešenie: Implementujte metódy na interpoláciu (napr. Vector3.Lerp pre horizontálny pohyb, a vertikálny pohyb pomocou výpočtu, ktorý zahŕňa výšku). Vylepšenie: Upravte kód tak, aby podporoval plynulé a prirodzené prechody medzi rôznymi polohami.

**Detekcia a animácia pohybu rúk**

Problém: Pohyb rúk je riadený pomocou arc motion, čo môže byť náročné na správne vykreslenie animácie. Riešenie: Použiť analytický prístup pri pohybe rúk a zabezpečiť, aby boli animácie pre rôzne pozície rúk správne synchronizované. Vylepšenie: Použiť animátora s viac parametrami, ktoré umožnia prepojenie animácií s aktuálnym stavom (napr. prechody medzi animáciami podľa aktuálnej polohy ruky).

**Zjednodušenie metódy Update**

Problém: Metóda Update je veľmi dlhá a zložito čitateľná. Riešenie: Rozdeľte metódu na menšie, zrozumiteľnejšie metódy, ktoré sa budú starať o jednotlivé časti mechaniky pohybu (napr. HandleHandMovement(), HandleClimbing(), HandleAnimations()).

**Parametrizácia hodnôt**

Problém: Niektoré hodnoty sú pevne stanovené v kóde (napr. rýchlosť pohybu rúk, vzdialenosti). Riešenie: Presuňte tieto hodnoty do Unity editora ako verejné premenné alebo použite externý konfiguračný súbor pre lepšiu flexibilitu pri ladení a úpravách.

## Tlačenie objektov

Trieda PushObject implementuje jednoduchú hernú mechaniku tlačenia objektov. Táto mechanika umožňuje hráčovi interagovať s objektmi, ktoré majú Rigidbody komponent, a aplikovať na ne silu pri kolízii. Prijímanie kolízií: Metóda OnControllerColliderHit sa spustí vždy, keď sa hráč (s CharacterController) dotkne iného objektu. Kontrola objektov: Zisťuje sa, či objekt, s ktorým sa hráč dotýka, má Rigidbody. Ak objekt nemá fyzikálne vlastnosti (napr. je isKinematic), tlačenie sa neaplikuje. Výpočet smeru tlačenia: Smer tlačenia je odvodený od smeru pohybu hráča (hit.moveDirection), ale len na rovine (os X a Z). Aplikácia sily: Sila tlačenia je vypočítaná ako smer násobený pevnou hodnotou pushPower a aplikovaná pomocou AddForce.

**Vylepšenia a optimalizácie**

Ohraničenie smeru tlačenia:Obmedzte tlačenie na konkrétne osi, ak potrebujete, aby objekty reagovali len v určitých smeroch (napr. len dopredu). Pridanie vizuálnej spätnej väzby: Pri tlačení môžete spustiť animácie alebo efekty, ktoré zvýraznia interakciu. Pridanie kontroly váhy: Skontrolujte hmotnosť objektu a obmedzte tlačenie pre veľmi ťažké objekty.

# Integrujte mechanizmus prepínania stavov rozhodovacej mechaniky hry

**Celkový koncept prepínania stavov v hre**

Mechanizmus prepínania stavov umožňuje nepriateľovi dynamicky reagovať na situácie v hre (napríklad prenasledovanie hráča alebo návrat na pôvodnú pozíciu). Tento koncept je známy ako **State Machine** (stavový automat). Každý stav definuje konkrétne správanie, pričom nepriateľ dokáže medzi stavmi plynule prepínať podľa vopred definovaných pravidiel.

## Rozdelenie tried a ich funkcie

**Abstraktná trieda EnemyBase**

Táto trieda slúži ako základ pre všetky stavy nepriateľa. Obsahuje tri základné metódy: OnEnter(): Táto metóda sa volá vždy, keď nepriateľ prechádza do daného stavu. Update(): Táto metóda obsahuje logiku, ktorá sa vykonáva počas trvania daného stavu. OnExit(): Táto metóda sa volá pri opustení daného stavu. Abstraktná trieda zabezpečuje, že všetky stavy majú rovnakú štruktúru a je jednoduché pridávať nové stavy.

**Konkrétne stavy**

Každý stav je odvodený od EnemyBase a má svoju vlastnú implementáciu logiky.

**StacionaryState**

Čo robí: Po vstupe do stavu sa nastaví cieľ na prvý waypoint. V Update() sleduje, či nepriateľ dosiahol cieľový bod. Ak áno, prechádza na ďalší waypoint. Optimalizácie: Môžete pridať náhodný výber waypointu namiesto sekvenčného prechodu. Možno implementovať zastavenie nepriateľa na určitý čas pri každom waypointu.

**ChaseState**

Čo robí?: Nastavuje cieľ na pozíciu hráča pomocou meshAgent.SetDestination(). Prenasledovanie pokračuje, pokiaľ je hráč v zornom poli. Možné rozšírenia: Pridať kontrolu vzdialenosti – ak je hráč príliš ďaleko, nepriateľ môže prejsť do stavu „LostState“. Implementovať animácie, napríklad zrýchlenie nepriateľa.

**LostState**

Čo robí?:Po vstupe do stavu sa začne odpočítavať čas, počas ktorého nepriateľ hľadá hráča. Po troch sekundách prejde späť do stavu „StacionaryState“. Optimalizácie: Použiť náhodné pohyby počas hľadania hráča. Pridať zvukové efekty alebo vizuálne signály, aby hráč vedel, že nepriateľ hľadá.

**Trieda Enemy**

Táto trieda riadi celkový chod nepriateľa a obsahuje logiku na prepínanie stavov. Prepínanie stavov (metóda ChenckUpdate)

Overuje, či sa aktuálny stav môže zmeniť na iný.Ak je podmienka CanGo splnená, vykoná sa: OnExit() na opustenie aktuálneho stavu. Zmena stavu na nový (m\_EnemyBase = To). OnEnter() na inicializáciu nového stavu.

Zorné pole nepriateľa (metóda SeePlayer)Táto metóda zisťuje, či je hráč v zornom poli nepriateľa. Uhlu medzi smerom nepriateľa a hráča. Maximálnej vzdialenosti viditeľnosti. Raycast kontroluje, či hráča nezakrývajú prekážky. Optimalizácia: Táto metóda je volaná viackrát za cyklus, čo môže byť náročné na výkon. Možno ju zefektívniť pomocou intervalového časovača.

**Príklady rozšírenia mechanizmu**

Pridanie nového stavu AttackState,Nepriateľ útočí na hráča, ak je blízko.

**Možné vylepšenia**

Použitie ScriptableObject pre stavy umožní oddeliť logiku stavu od nepriateľa a zjednoduší prácu s viacerými nepriateľmi. Výkonnostná optimalizácia: Implementácia pooling systému na recykláciu nepriateľov. Zníženie volaní SeePlayer pomocou intervalových kontrol. Vylepšenie animácií: Prepojenie stavov s animáciami pomocou Animator Controller.

# Vytvorte systém na vykresľovanie grafických detailov

**Čo je LOD (Level of Detail)?**

LOD (Level of Detail) je technika optimalizácie výkonu, ktorá znižuje počet polygonov objektu v závislosti od jeho vzdialenosti od kamery. Keď je objekt blízko, zobrazuje sa vysoká kvalita. Keď je ďalej, Unity prepína na jednoduchšiu verziu modelu, aby šetrilo výkon. Ak je veľmi ďaleko, objekt sa môže úplne skryť (Culled). Cieľ LOD je znížiť počet vykreslených polygonov a tým zvýšiť FPS.

**Ako Unity implementuje LOD?**

Unity používa komponent "LOD Group", ktorý umožňuje nastaviť rôzne verzie modelu podľa vzdialenosti od kamery.

LOD úrovne v Unity: LOD 0 – Plný detail (najviac polygonov). LOD 1 – Zjednodušený model. LOD 2 – Ešte jednoduchší model. LOD 3+ – Môže byť ešte viac úrovní. Culled – Objekt sa prestane vykresľovať. Unity automaticky prepína medzi týmito modelmi podľa vzdialenosti od kamery.

**Ako nastaviť LOD v Unity?**

Manuálne nastavenie cez "LOD Group".Vyber objekt v Hierarchy. Pridaj komponent "LOD Group". Klikni na objekt → "Add Component" → "LOD Group"  
Pridaj rôzne verzie modelu. Drag & Drop rôzne modely do jednotlivých LOD slotov. Každý model by mal mať menej polygonov ako predošlý.  
Nastav percentá prepínania.V LOD Group komponente sú posuvníky (sliders). Posuň ich tak, aby určili kedy sa má ktorá verzia modelu zobraziť.  
Otestuj v Scene View. V Scene View klikni na LOD režim (pravý horný roh). Pohybuj kamerou a sleduj, ako sa modely menia.

Príklad vizuálnej reprezentácie LOD úrovní:

1. Blízko kamery: Plný detail (100 000 polygonov)
2. Stredná vzdialenosť: Menej detailov (50 000 polygonov)
3. Ďaleká vzdialenosť: Veľmi jednoduchý model (5 000 polygonov)
4. Veľmi ďaleko: Model sa už nezobrazuje (Culled)

**Ako generovať LOD automaticky?**

Ak nemáš ručne vytvorené modely s nižším počtom polygonov, môžeš použiť automatické generovanie LOD. Automatická tvorba LOD v Unity umožňuje generovanie LOD modelov priamo v editore: Klikni na 3D model v Project okne.Otvor Inspector a klikni na "Generate LODs". Unity vytvorí automatické LOD modely s nižším počtom polygonov. Nevýhoda: Automatická metóda nemusí vždy vytvoriť ideálne optimalizované modely.

# Navrhnite systém renderovania objektov

**Úvod**

V 3D grafike sa často vykresľujú objekty, ktoré nie sú viditeľné pre hráča, čo zbytočne zaťažuje grafickú kartu a procesor. Occlusion Culling v Unity je technika, ktorá umožňuje optimalizovať renderovanie tým, že skryje objekty, ktoré nie sú viditeľné, pretože ich zakrývajú iné objekty.

Occlusion Culling sa líši od Frustum Culling, ktoré deaktivuje len objekty mimo zorného poľa kamery. Occlusion Culling navyše skrýva objekty, ktoré sú zakryté inými objektmi, čím znižuje počet vykreslených objektov a zlepšuje výkon.

**Ako funguje Occlusion Culling?**

Unity pri bake procesu rozdelí scénu na tzv. occlusion bunky (cells) a vypočíta viditeľnosť objektov. Počas hrania potom porovnáva polohu kamery a renderuje len viditeľné objekty.

Hlavné komponenty Occlusion Culling

1. Occluders – Objekty, ktoré blokujú pohľad (napr. steny, budovy).
2. Occludees – Objekty, ktoré môžu byť zakryté a teda skryté z renderovania.
3. View Cells – Oblasti, v ktorých sa pohybuje kamera a na základe nich sa určuje viditeľnosť.

Target Cells – Oblasti obsahujúce objekty, ktoré môžu byť skryté.

Výber správnych occluders a occludees je kľúčový pre efektívne využitie Occlusion Culling.Nastavenie Occlusion Culling v Unity. Otvorenie Occlusion Culling okna.Prejdite do Window → Rendering → Occlusion Culling. Označenie objektov ako statické. Vyberte objekty, ktoré budú slúžiť ako Occluders (napr. steny, budovy). V Inspector aktivujte Static → Occluder Static. Objekty, ktoré môžu byť skryté, označte ako Occludee Static.

**Bake procesu**

Kliknite na Bake a nechajte Unity vypočítať viditeľnosť objektov.Výsledky si môžete skontrolovať v záložke Visualization.

**Optimalizácia Occlusion Culling**

Veľké objekty je vhodné rozdeliť na menšie časti, aby ich bolo možné individuálne skrývať. Malé objekty, ktoré nie sú occluders, je možné zlúčiť do väčších skupín, aby sa znížil počet draw calls.

Nastavenia Occlusion Culling Bake

V okne Bake môžeme upraviť nasledujúce parametre:

1. Smallest Occluder – Najmenšia veľkosť objektu, ktorý môže blokovať pohľad.
2. Smallest Hole – Minimálna veľkosť otvoru, cez ktorý sa môže vidieť.
3. Backface Threshold – Určuje, ktoré povrchy sa majú považovať za neviditeľné.

Použitie Occlusion Areas

Ak potrebujete manuálne definovať oblasti occlusion culling, môžete použiť Occlusion Area: Vytvorte prázdny GameObject a pridajte komponent Occlusion Area. Nastavte Size a Center podľa požadovanej oblasti. Aktivujte Is View Volume, ak chcete skryť statické objekty vo vnútri oblasti. Testovanie Occlusion Culling. Použitie Overdraw módu.V Scene View môžete zapnúť Overdraw mode, ktorý ukáže, koľko objektov sa zbytočne vykresľuje.

**Vizualizácia Occlusion Culling**

V Occlusion Culling okne prejdite na záložku Visualization a pohybujte kamerou. Skontrolujte, či objekty správne miznú, keď sú zakryté inými objektmi. Ladenie problémov: Skontrolujte, či sú správne označené ako Occluder Static alebo Occludee Static. Upravte hodnoty Smallest Occluder a Smallest Hole. Rozdeľte príliš veľké objekty na menšie časti.

**Výhody a nevýhody Occlusion Culling**

Výhody:

1. Zlepšuje výkon hry znížením počtu vykreslených objektov.
2. Vhodné pre interiérové scény a levely s množstvom prekážok.
3. Funguje automaticky po nastavení a bake procese.

Nevýhody:

1. Vyžaduje manuálne nastavenie a bake proces, čo môže byť časovo náročné.
2. Nie je efektívne v otvorených svetoch s veľkou viditeľnosťou.
3. Niekedy môže nesprávne skryť objekty, ak nie sú správne nastavené parametre.

# Pripravte analýzu trhu

Analýza trhu pre hru závisí od viacerých faktorov, ako je žáner hry, cieľová skupina, aktuálny stav herného priemyslu, konkurencia a ďalšie aspekty. Na základe toho by sa analýza mohla skladať z nasledujúcich krokov:

**Definovanie žánru hry**

1. Aký je žáner vašej hry? (napr. akčná, adventúra, RPG, simulátor, športová, stratégia, horor atď.)
2. Ako sa vaša hra líši od existujúcich hier v tomto žánri? (inovácie, mechaniky, príbeh)

**Cieľová skupina**

1. Kto sú hráči, ktorým je vaša hra určená? (vek, pohlavie, záujmy, nákupné správanie)
2. Kde sú vaši hráči geograficky umiestnení? (lokálne alebo globálne trhy)
3. Aké sú preferencie hráčov, ktorí preferujú tento žáner? (napr. preferencia na konzolách alebo PC, hranie mobilných hier)

**Trendy v hernom priemysle**

1. Aktuálne trendy v hernom priemysle: Je záujem o konkrétny žáner alebo nový herný mechanizmus?
2. Rastúce trhy: Napríklad trhy v Ázii, Latinskej Amerike alebo ďalšie emerging markets.
3. Technologické trendy: Pokrok v oblasti virtuálnej reality (VR), rozšírenej reality (AR), cloudového hrania, blockchainu v hrách alebo NFT.

**Analýza konkurencie**

1. Hlavní konkurenti: Kto sú najväčší konkurenti na trhu s podobnými hrami?
2. Výkonnosť konkurencie: Ako sú na tom konkurenti v oblasti predajov, používateľských recenzií a ziskovosti?
3. Ceny konkurencie: Aké sú cenové stratégie konkurencie, ako sa líši vaše cenové nastavenie?

**Distribúcia a platformy**

1. Na akých platformách bude vaša hra dostupná? (PC, konzoly, mobilné zariadenia, VR/AR, cloudové služby)
2. Obchodné modely: Predaj cez Steam, Epic Games Store, Google Play, App Store, alebo vlastná distribúcia cez webové stránky.
3. Plán na DLC, mikrotransakcie, sezónne pasy atď.

**Marketingová stratégia**

1. Spôsoby propagácie hry: Aké marketingové kanály plánujete využiť? (influenceri, reklama na sociálnych sieťach, videorecenzie, streamovanie)
2. PR a médiá: Spolupráca s hernými webmi, recenzentmi, a blogermi.
3. Komunitné aktivity: Súťaže, beta testovanie, komunita okolo hry.

**Finančné aspekty**

1. Počiatočné náklady: Vývoj, dizajn, marketing, distribúcia.
2. Očakávané výnosy: Projekcia predajov podľa trhových analýz.
3. ROI (návratnosť investícií): Odhad doby, kedy sa investícia vráti.

**Riziká a výzvy**

1. Konkurencia: Vysoká konkurencia v populárnych žánroch môže ovplyvniť predaj.
2. Technické výzvy: Problémy s vývojom hry, kompatibilitou s rôznymi zariadeniami alebo technické nedostatky.
3. Zákonné a regulačné riziká: Napríklad predpisy týkajúce sa ochrany osobných údajov alebo herného obsahu v rôznych regiónoch.

**SWOT analýza (Silné stránky, Slabé stránky, Príležitosti, Hrozby)**

1. Silné stránky: Inovatívne mechaniky, silný príbeh, výnimočný dizajn alebo technológie.
2. Slabé stránky: Nedostatok skúseností v určitých oblastiach, rozpočet, technické problémy.
3. Príležitosti: Rastúci trh, technológie (napr. VR, AR), partnerstvá.
4. Hrozby: Konkurencia, zmena trendov, problémy s distribúciou alebo financovaním.

**Predikcia vývoja trhu**

1. Predpokladaný rast trhu: Aký je predpokladaný rast trhu s hrami v nasledujúcich rokoch?
2. Potenciálne novinky a inovácia: Aké technológie môžu ovplyvniť herný trh? (napr. AI, blockchain, nové herné modely)

**príloha B - Fotodokumentácia**