VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HERNÍ DEMO VYUŽÍVAJÍCÍ NETRIVIÁLNÍ VYKRESLOVÁNÍ SVĚTA

GAME DEMO USING NON-TRIVIAL WORLD RENDERING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ALEXANDER SILA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILET, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

153378

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Sila Alexander**
Program:	Informační technologie
Název:	**Herní demo využívající netriviální vykreslování světa**
Kategorie:	Počítačová grafika
Akademický rok:	2023/24

Zadání:

1. Nastudujte techniky herního vývoje a struktury herních enginů. Seznamte se s herním enginem Unity. Prostudujte netriviální techniky počítačové grafiky. Seznamte se s portálovými algoritmy a algoritmy založenými na stencilovém bufferu.
2. Navrhněte hru využívající nestandardní vykreslování světa. Soustřeďte se na využití vybraného zobrazení v herních mechanikách.
3. Implementujte navržené herní demo.
4. Změřte výslednou realizaci a uživatelsky ji otestujte.
5. Sepište závěry, budoucí rozšíření a aplikaci zveřejněte. Vytvořte demonstrační video.

Literatura:

- Gregory, Jason. *Game engine architecture*. crc Press, 2018. ISBN 1351974289, 9781351974288
- Bishop, Lars, et al. "Designing a PC game engine." IEEE Computer Graphics and Applications 18.1 (1998): 46-53.
- Adams, Ernest, and Joris Dormans. Game mechanics: advanced game design. New Riders, 2012. ISBN 0321820274, 9780321820273

Při obhajobě semestrální části projektu je požadováno:
Funkční prototyp s netriviálním zobrazováním a alespoň tři herní mechaniky využívající dané zobrazování. Třicet stran technické dokumentace.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Milet Tomáš, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	9.11.2023

Abstrakt

Cílem této práce je využít metody netriviálního vykreslování k vytvoření nových herních mechanik, které využívají netradiční zobrazení portálů. Řešení tohoto problému je především založeno na stencil bufferu. Ten slouží pro vytváření 2D efektů, proto je rozšířený při vykreslování plochých portálu. V této práci je efekt využit pro kulovité portály v 3D prostoru. Tento netradiční typ portálu navíc prolíná dimenze světů v herním prostředí. Herní objekty, které se vyskytují na pomezí těchto dimenzí jsou částečně vykreslovány včetně modifikovaných stínů. Koncept této hry je zasazen do procedurálně generovaného světa, který využívá Unity job systém k paralelnímu generování. Umělá inteligence nepřátel využívá virtuálního navigačního prostoru k pohybu v tomto komplexním prostředí. Hra je navíc rozšířena o správu inventáře, pokročilý animační systém, ukládání světa a mnoho dalších funkcionalit. Výsledkem této práce je plně hratelné herní demo, které přináší inovativní přístup k hernímu designu a rozšiřuje možnosti tohoto žánru.

Abstract

The aim of this work is to utilize non-trivial rendering methods to create new gameplay mechanics that utilize unusual portal displays. The solution to this problem is primarily based on the utilization of the stencil buffer, which is used for creating 2D effects, thus it is commonly employed in rendering flat portals. However, in this work, this effect is employed for spherical portals in a 3D space. Additionally, this unconventional type of portal intersects dimensions of worlds within the gaming environment. Game objects that appear at the boundary of these dimensions are partially rendered, including modified shadows. The concept of this game is set in a procedurally generated world, utilizing the Unity job system for parallel generation. Enemy artificial intelligence utilizes a virtual navigation space to move within this complex environment. Furthermore, the game is enhanced with inventory management, advanced animation system, world saving, and many other functionalities. The outcome of this work is a fully playable game demo that brings an innovative approach to game design and expands the possibilities of this genre.

Klíčová slova

Stencilový buffer, stencilová maska, shader, vykreslování, Unity, herní engine, portál, dimenze, procedurální generování, Unity job systém, paralelní výpočty, správa inventáře, virtuální navigační prostor

Keywords

Stencil buffer, stencil mask, shader, rendering, Unity, game engine, portal, dimensions, procedural generation, Unity job system, parallel computations, inventory management, NavMesh

Citace

SILA, Alexander. *Herní demo využívající netriviální vykreslování světa.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet, Ph.D.

Herní demo využívající netriviální vykreslování světa

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Alexander Sila
28. dubna 2024

Poděkování

Především bych chtěl upřímně poděkovat panu doktorovi Tomáši Miletovi za jeho profesionální přístup a skvělé rady, které mi pomohly dosáhnout úspěchu. Dále bych chtěl vyjádřit vděk všem účastníkům testování, kteří poskytli konstruktivní zpětnou vazbu.

Obsah

Kapitola 1

Úvod

Videohra je forma interaktivní digitální zábavy, kde hráč ovládá postavu nebo situaci v rámci virtuálního prostředí. Tato forma zábavy kombinuje prvky grafiky, zvuku a interaktivity s cílem poskytnout hráčům různorodé zážitky.

Herní průmysl nabízí širokou škálu žánrů, které splňují různé požadavky a preference hráčů, od akčních her, strategií, sportovních her po adventury a mnoho dalšího. Tvorba video her je kreativní činnost, která nejen přináší hráčům rekreační požitek z virtuálního světa, ale také rozvíjí nové technologie od pokročilé grafiky po virtuální realitu. Herní průmysl je jeden z nejrychleji rostoucích průmyslových odvětví na světě, generující několik miliard korun ročně. S tímto trhem souvisí i prodej herního hardwaru a softwaru.

Na světě existuje mnoho her, které se řadí do určitých žánrů, tudíž je logické, že některé spadají do stejné kategorie, proto se každá nová hra snaží být v něčem odlišná, aby zaujala co nejvíce lidí a poskytla hráčům nové možnosti a zážitky. To platí i pro hry, které se rozhodly jít cestou netriviálním vykreslováním světa. V takových hrách se často vyskytují prvky jako jsou například portály, prostoupení do jiných světů, ohýbání prostoru nebo podobných modifikací prolínání světů. Takový design světa dává široký prostor pro herní mechaniky, které fungují na bázi těchto jevů.

Hry s netriviálním vykreslováním světa přinášejí do světa her několik významných prvků a zážitků, které mohou poskytnout hráčům poutavější a bohatší herní prostředí. To vytváří vizuálně atraktivní zážitek a umožňuje hráčům ponořit se do bohatých a živých herních světů, což přidává do hry dynamiku a rozmanitost. Tvorba takového prostředí vzbuzuje v hráči touhu po prozkoumávání a nutí ho k zamyšlení, jak se v takovém nepřirozeném prostředí pohybovat a jakým způsobem s ním interagovat.

Proto cílem této práce je vytvořit nevídaný svět, ve kterém hráč bude prozkoumávat různé podoby světa, setkávat se s tamějším životem a interagovat s prostředím prostřednictvím unikátních herních mechanik, které umožní prolínání různých světů mezi sebou. A obohatit tento žánr o novou hru, která přinese inovativní nápady a nový pohled na řešení netriviálního vykreslování.

Představení herního dema

Díky této práci vznikla unikátní hra **Realmweaver**, která využívá herní mechaniky založené na netriviálním vykreslování. Pomocí těchto herních mechanik hráč prozkoumává nekonečný svět a řeší v něm logické hádanky. Interakci se světem zajišťují netradiční portály, které prolínají dimenze mezi třemi světy. Každý svět tvoří unikátní ekosystém plný surovin a tajemných struktur. Hráč postupem času sbírá materiály, které využívá k výrobě předmětů a součástek. Cílem hry je opravit vesmírnou loď a odletět z této podivuhodné planety.

Kapitola 2

Seznámení s Problematikou

Při vykreslování netriviálního světa vývojáři naráží na mnoho zásadních problémů, které jsou v rozporu s normálním chápáním fyzického světa. Proto tato kapitola přibližuje konkrétní problémy a vysvětluje důvody komplikací, které vznikají při vývoji her s prvkem netriviálního vykreslování. Dále se zabývá průzkumem herního trhu, jehož cílem je identifikovat důležité aspekty jednotlivých her, pochopit realizaci herních mechanik a jejich zasazení do herního prostředí.

2.1 Základní problémy

Netriviální vykreslování sebou přináší spoustu softwarových a hardwarových výzev, které je nutné korektně řešit. Tato sekce se zabývá analýzou různých problémů a jejich příčin či důsledků. Hlavním cílem je nastínit problematické aspekty netriviálního vykreslování, které jsou klíčové pro řešení této práce.

Problém více světů

Pod pojmem „prolínání světů“ člověka napadne mnoho abstraktních představ. Hlavním problémem při takovém představování je projekce a rozmístění různých světů. Portál by mohl sloužit jako brána mezi dvěma velmi vzdálenými světy. Jiný portál může reflektovat změnu světa v čase nebo další dimenzi, která kopíruje terén původního světa, ale existují v něm jiné či modifikované objekty. Abstraktní myšlení nemá hranice, ovšem některé nápady a myšlenky jsou obtížně realizovatelné v herním prostředí.

Realizace těchto konceptů je také ovlivněna herními mechanismy, které určují úroveň hratelnosti a konečnou implementaci. Dalším omezením jsou prostředky a nástroje, které jsou k dispozici. Vzhledem k omezenému počtu her, ve kterých je netriviální vykreslování klíčovým prvkem, je nutné vytvářet unikátní nástroje a řešení.

Vizualizace

Vizualizace je naprosto fundamentální pro netriviální vykreslování. Objekty musí být prezentovány hráči tak, aby bylo jasné, do jakého světa patří. Mohou nastat případy, kdy je potřeba vykreslit pouze určitou část objektu s různými vlastnostmi. Další základní problém nastává v situaci, kdy je objekt na pomezí dvou světů a nabývá vícero podob. Je proto nezbytné správně zobrazit objekt v každém možném scénáři.

Při vykreslování je velice důležité v jakém pořadí se objekty vykreslují. Nejen kvůli zabránění nesmyslnému překrytí objektů, ale také kvůli využití určitých technik, které závisí na tomto pořadí.

Kolizní systém

Problém s kolizemi vzniká u objektů, které sice nejsou aktuálně viditelné, ale stále interagují se světem. Je důležité si uvědomit, že nevykreslení objektu neznamená jeho absence v prostoru nebo jeho ztrátu interaktivity. Tento problém může být jednoduše řešitelný, ovšem i výpočetně náročný. Pokud existuje v herní úrovni mnoho kolizí, tak fyzikální výpočty mohou značně zpomalit plynulost hry. Je proto nutné nalézt kompromis mezi kvalitou kolizí a výpočetním výkonem.

Jednoduchým řešením je porovnávat vzdálenosti mezi pozicí entity a rádiem portálu, aby se rozhodlo, zda má být kolize aktivována či deaktivována. Avšak toto řešení není zcela přesné a může vést k neúplně přesným kolizím. Především u velkých kolizních objektů.

Pro úplně přesné kolize je nutné za běhu aplikace vypočítat, zda jsou vrcholy objektu viditelné skrze portál z pohledu kamery, a podle této informace vypočítat a aktualizovat kolize každý snímek. Toto řešení může být optimalizováno selektivním výběrem entit, které jsou viditelné skrze portál nebo jsou v blízkosti hráče, což může šetřit výpočetní zdroje. Nicméně pro velké scény může být takový přístup nepraktický, pokud má aplikace běžet plynule.

Stíny

U každého objektu lze nastavit, zda má vrhání stínů povoleno nebo zda má umožnit ostatním objektům vrhat na něj stíny. Avšak vzniká problém, když hráč vidí pouze část objektu. Je přirozené, že neviditelná část by neměla vrhat stín, a proto je nezbytné správně vypočítat ořezaný stín pro každý objekt zvlášť. Při vykreslování stínů vznikají i další problémy, které jsou důsledkem omezení vývojového prostředí.

Interakce a chování v odlišných dimenzích

U objektů, které se projevují odlišně v různých světech, může dojít i ke změně interakce s okolním prostředím nebo k modifikaci chování. Například v jiných světech může na objekt působit odlišná gravitace nebo se organismy přizpůsobují specifickým podmínkám dané dimenze. Proto je klíčové identifikovat okamžik změny a zajistit plynulý přechod v chování herních objektů.

Procedurální generovaní světa

Procedurální generování představuje vytváření obsahu (dat, grafiky, zvuků, terénu atd.) pomocí algoritmů nebo procedur. Díky tomu nejsou vývojáři nuceni ručně umísťovat nebo vytvářet všechny objekty, což přidává dynamiku do světa a šetří drahocenný čas. Navíc tento způsob generování může teoreticky tvořit nekonečně velké světy. Nicméně s sebou přináší i určité nevýhody, které vyžadují komplexnější řešení. Cílem procedurálního generování je vytvořit svět, který buď předčí nebo se co nejvíce přiblíží k tvořivým schopnostem člověka. Realizace této myšlenky však není jednoduchá.

Jednou z výzev procedurálního generování je řešení správy světa a výkonu. Generování světa podle složitějších pravidel zatěžuje procesorový čas. Je nutné konzistentně generovat nebo načítat různé části světa, zejména při opakovaném navštěvování určitých oblastí.

Výkon a optimalizace

Vysoké nároky na výpočetní výkon a zdroje jsou způsobeny velkým množstvím entit ve světě, které je třeba vykreslit. S přibývajícím počtem světů, které se mohou vzájemně prolínat, roste i počet entit, které je nutné spravovat. I když hráč nevidí objekty v jiných dimenzích, procesy na těchto objektech stále probíhají a je nezbytné provádět výpočty pro správné vykreslování. Ve většině her je zpravidla zohledněn pouze jeden svět, a i ten může být náročný na správu. Pracovat s více světy současně proto není o nic jednodušší záležitost.

Více aktivních světů přináší mnohonásobně větší zatížení pro dostupné prostředky. Proto je zásadní provádět pouze nezbytně nutné výpočty, optimalizovat modely, udržovat přiměřený počet skriptů, které provádějí akce každý snímek, efektivně načítat světy za běhu aplikace a provádět další optimalizace.

2.2 Analýza podobných her

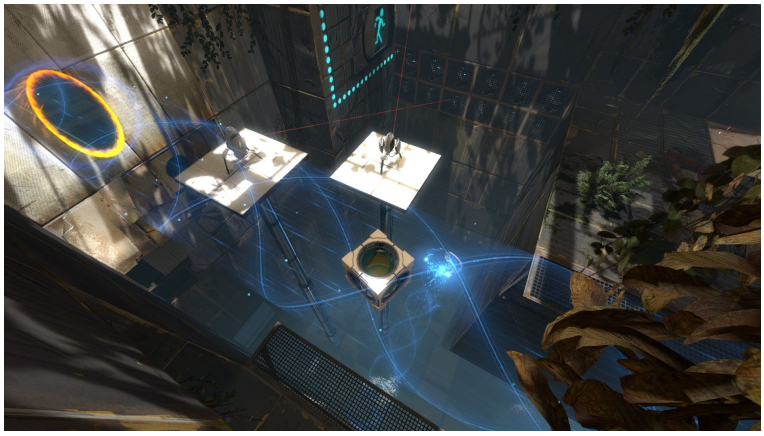
Pro lepší pochopení a představu se tato část zaměřuje na konkrétní problémy a jejich řešení v reálných hrách. Jelikož jedním z mnoha cílů této práce je obohatit tento žánr o nové inovativní herní mechaniky a design, tak je nezbytné provést průzkum trhu. Tento průzkum poskytuje vývojářům a designérům nové myšlenky, které vychází z úspěšných i nezdařených nápadů, jež byly představeny ve světě her. To umožňuje využít příležitosti k poučení se z minulých neúspěchů, které lze buď zcela opustit, nebo zdokonalit.

Portal 2

Jedná se o logickou hru vydanou společností *Valve Corporation* v roce 2011[1]. Hra je strukturována do několika úrovní, ve kterých hráči řeší hádanky za pomocí portálů. Hráč je omezen pouze na dva portály, každý jiné barvy, které může vystřelit v libovolném pořadí na plochy k tomu určené. Skrze portály může hráč nahlédnout do světa z úhlu pohledu protějšího portálu nebo mezi nimi procházet a transportovat objekty. Portály mají oválný tvar a jsou vždy umístěny na rovné ploše, což z nich činí 2D portály, které promítají 3D svět.

Portály fungují na principu *stencil bufferu*, který maskuje pixely tvořící daný portál. Tento princip je později popsán v sekci 3.4. Průchod mezi portály je složitý, neboť musí být zachována rychlost a směrový vektor objektu, který proniká do portálu. Kromě toho se objevuje problém, kdy je objekt částečně vně nebo vyčnívá z portálů. Tato obtíž se řeší duplikací daného objektu na druhé straně portálu. Jakmile objekt úplně projde prvním portálem, zanikne a zůstane zachována pouze instance objektu, která byla duplikována a transportována na druhý konec.

[1] Dostupné na: https://store.steampowered.com/app/620/Portal_2/.



Obrázek 2.1: Ukázka ze hry Portal 2. Na obrázku lze pozorovat jak hráč využil oranžový portál k přesunu krychle.

Quantum Conundrum

Jedná se o logickou hru z pohledu první osoby, kterou vydala společnost *Airtight Games* v roce 2012[2]. Hra je rozdělena do několika úrovní, ve kterých hráč řeší hádanky pomocí přepínání dimenzí. Při změně dimenze zůstává svět a pozice všech objektů stejná, avšak se aplikují nové fyzikální zákony odpovídající dané dimenzi. V každé dimenzi platí jiné fyzikální zákony, které hráč musí využít k řešení hádanek a postupu do další úrovně.

V Prvním světě jsou všechny objekty desetkrát lehčí, což umožňuje hráči přesouvat i těžší předměty. Ve druhém světě jsou objekty naopak několikrát těžší, ale jejich nová hustota jim brání v poškození. Ve třetím světě se čas výrazně zpomalí, ale hráč si zachovává původní rychlost, což mu umožňuje rychle se pohybovat ve zpomaleném světě. V poslední dimenzi je gravitace obrácená, což však nemá vliv na pohyb hráče. Současně může být ovlivněn pouze jeden aspekt světa.

Při změně dimenze se na obrazovce hráče aplikuje grafický filtr a některé materiály objektů se mohou změnit podle příslušné dimenze. Ve hře se sice netriviální vykreslování světa vyskytuje zřídka, ovšem lze si povšimnout využití herních mechanik a důsledků přepínání dimenzí, které lze aplikovat ve světech s netriviálním vykreslováním.

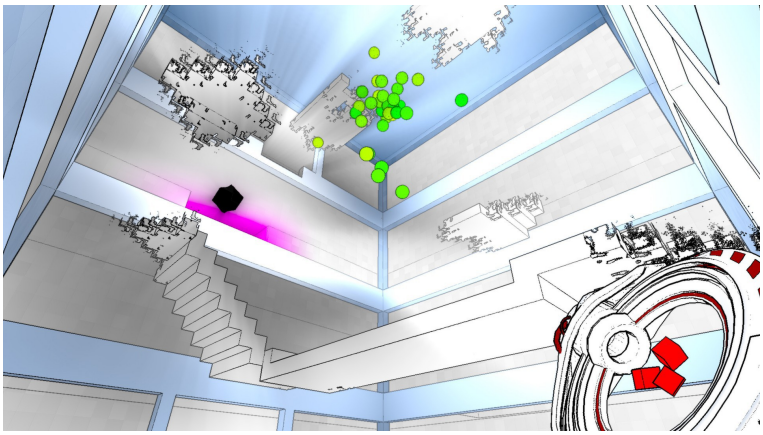
Obrázek 2.2: Ukázka ze hry Quantum Conundrum

[2]Dostupné na: https://store.steampowered.com/app/200010/Quantum_Conundrum/.

Antichamber

Jedná se o logickou hru z pohledu první osoby vydanou společností *Demruth* v roce 2014[3]. Hra je strukturována do několika úrovní, ve kterých hráč řeší hádanky využívající neeuklidovský prostor. Hráč se musí naučit orientovat se v tomto netradičním prostoru, aby dosáhl cíle. K řešení hádanek také využívá pozici a orientaci kamery, která ovlivňuje chování dalších objektů a relativně zkresluje prostor. Například, když hráč vejde do zdánlivě prázdné místnosti, musí se otočit a projít stejným vchodem, aby se dostal na úplně jiné místo. Hra využívá psychologické triky, jako jsou iluze, nejistoty a rozporuplné informace, aby hráče zmátla a způsobila, že bude váhat nad tím, co je skutečné a co není.

Některé prvky neeuklidovského prostoru lze řešit *stencil bufferem*, ovšem pro plný efekt neeuklidovského prostoru je nutné modifikovat geometrii v prostoru.



Obrázek 2.3: Ukázka ze hry Antichamber

2.3 Specifika cílů

Cílem této práce je využít netriviální vykreslování k vytvoření nových herních mechanik a přetvořit klasické portály do netradiční podoby. Tradiční portály slouží k teleportaci, ale nové využití portálů spočívá v prolínání světů. Všechny dimenze jsou součástí jednoho celku, což umožňuje hráči procházet různými světy na stejném místě. Kulovité portály mohou vzniknout kdekoliv ve světě, což dává hráči volnost v jejich umístění. Hráč má k dispozici i druhý portál, který mění okolní svět kolem hráče. Tyto portály ovlivňují chování objektů a hráčův pohled na svět. Hráč může interagovat s prolínajícími se světy a využívat změny prostředí k řešení hádanek.

Celý tento koncept netriviálního vykreslování se tato práce snaží integrovat do procedurálně generovaného světa. Procedurální generování musí zajistit řešitelnost všech hádanek a umožnit hráči používat všechny herní mechaniky bez zbytečných komplikací.

Optimalizace je v tomto případě klíčová. Komplexní vykreslování a velký procedurálně generovaný svět představuje výzvu pro výpočetní výkon a dostupné zdroje. Proto je důležité využít metody paralelního zpracování na více vláknech.

Výsledná aplikace je optimalizována pro běh na stolním počítači s operačním systémem Windows.

[3]Dostupné na: https://store.steampowered.com/app/219890/Antichamber/.

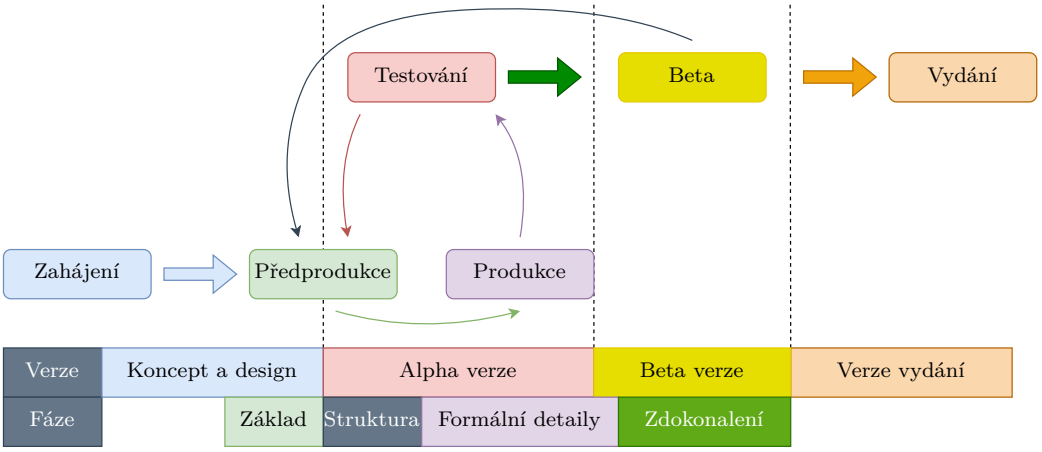
Kapitola 3

Teorie

V této kapitole jsou teoreticky popsány všechny důležité aspekty, které tvoří výslednou realizaci aplikace. V první části je detailně popsán iterativní proces životního cyklu vývoje her [19] a vysvětleny jednotlivé fáze [3]. Poté následuje sekce herních enginů 3.2, která přibližuje vývojové prostředí Unity, použitého pro implementaci této práce. Následně je vysvětlen pojem *Shader* v sekci 3.3 a jeho využití, které je klíčové k dosažení netriviálního vykreslování. V neposlední části kapitoly je zmíněn princip procedurálního generování a jeho využití k vytváření rozsáhlých světů. Na závěr je vysvětleno chování umělé inteligence a její orientace v prostoru.

3.1 Vývoj her

Vývoj her představuje komplexní a časově náročný proces, vzhledem k široké škále požadavků, které musí hra splňovat. Náročnější projekty vyvíjí desítky či stovky lidí, kteří vyžadují efektivní řízení. Každý člen týmu přispívá k realizaci projektu a jeho práce musí být pečlivě začleněna do celkového rámce projektu. Existuje mnoho způsobů, jak organizovat vývoj, ale ne každý je vhodný pro všechny druhy her. Jednou z nejčastěji používaných a osvědčených metod vývoje je tzv. „*GDLC*“[1]. Tato iterativní metoda rozděluje vývoj do několika fází, jak je možné vidět na obrázku 3.1.



Obrázek 3.1: Znázornění vývoje her podle modelu **GDLC**.

[1]GDLC (game development life cycle) reprezentuje iterační cyklus vývoje her.

Zahájení

Na začátku je nutné vytvořit hrubý koncept hry a definovat základní prvky, které budou součástí hry. Toto vyplývá z průzkumu trhu a analýzy cílového publika. Odborníci stanoví rozpočet pro vývoj a odhadnou potřebnou velikost týmu.

Předprodukce

V první fázi *GDLC* vývoje je nutné vytvořit herní design a prototyp. Designeři definují základní koncepty celé hry, jako jsou herní mechaniky, příběh, žánr, charakteristika postav, design světa, hratelnost a tak podobně. Tyto rozhodnutí a nápady jsou pečlivě zdokumentovány v tzv. „*GDD*“[2]. Zhotovení tohoto dokumentu započne týmovou práci na prototypu, který se snaží naplnit hlavní představu designerů. Během počátečního vývoje tým zkoumá, jak efektivně lze herní nápady převést do souvislého celku. Při testování prototypu je kladen důraz především na zábavnost hry, což je klíčové pro zachování správného směru vývoje.

Produkce

Druhá fáze představuje nejnáročnější a nejdelší část procesu, která může trvat i několik let. Tým se převážně zaměřuje na vytváření tzv. „*assetů*“[3], designování úrovní a jejich následnou integraci. Tým se rozděluje do následujících rolí:

- **Manažer produkce** v herním vývoji je klíčovým členem týmu, který se zaměřuje na řízení a koordinaci všech aspektů produkce hry. Jeho úkolem je zajistit, aby vývojový tým pracoval efektivně, dodržoval stanovené termíny a dosáhl stanovených cílů.
- **Programátoři** jsou odpovědní za implementaci herních mechanik, vývoj nástrojů pro ostatní členy týmu, grafické rozhraní a další technické prvky hry.
- **Grafici** mají významnou roli při vytváření vizuálního obsahu a estetiky, která tvoří zásadní část celkového herního zážitku. Jejich práce zahrnuje návrh postav, prostředí a tvorbu tzv. „*Game ready assetů*“, což jsou optimalizované herní objekty, které zaručují plynulost a efektivitu hry.
- **Animátoři** jsou zodpovědní za vytváření animací postav a pohyblivých objektů.
- **Level designeři** se specializují na vytváření herních úrovní, které hráči procházejí a interagují s nimi, včetně návrhu prostředí a strukturování úrovní.
- **Zvukaři** vytvářejí zvukové prostředí hry, včetně zvuků pro postavy, objekty, prostředí a hudby, která podporuje atmosféru hry.
- **Testeři** zkoumají herní mechaniky, identifikují chyby, testují hratelnost a výkon hry s cílem zajistit, že je připravena k další fázi vývoje nebo vydání.

[2] GDD (game design document) je označení pro dokument, který obsahuje designový koncept hry.

[3] Asset je digitální objekt, zdroj nebo soubor, který může reprezentovat například grafiku, zvuky, animace, 3D modely, textury, skripty, atp.

Testování

Testování, známé také jako *QA*[4] představuje poslední fázi jednoho iterativního cyklu vývoje, během které je nezbytné odhalit veškeré nedostatky a zhodnotit zábavný element hry. Tuto fázi provádí skupina testerů, kteří prověřují dosavadní práci pomocí různých metod. Jejich hlavním cílem je zajistit kvalitu hratelnosti a ověřit specifikace nových implementací. Každou nalezenou chybu detailně dokumentují a hlásí v podobě tzv. „*bug reportu*“, který obsahuje popis daného problému. Proto je důležité, aby tester jednoznačně popsal chybu a poskytl smysluplný návod k její replikaci.

Testovací fázi zakončuje tzv. „*Smoke test*“, který relativně za krátkou dobu obecně otestuje všechny základní aspekty hry a ověří, zda je možné přejít do další fáze vývoje.

Beta

Po několika iteracích alfa testování přichází fáze beta testování, která si klade za cíl získat zpětnou vazbu od hráčů, kteří reprezentují konečné uživatele. Tato fáze simuluje vydání hry, aby se předešlo případným problémům, které by mohly nastat, až si hráči skutečně zakoupí finální produkt. Beta verze hry je obvykle dostupná omezenému počtu uživatelů, kteří mohou získat předčasný přístup ke hře a pomoci tak vývojářům odhalit a vyladit nedostatky.

Beta testování se obvykle dělí do dvou kategorií. První je tzv. „*Uzavřená beta*“, kdy je obsah hry dostupný pouze vybraným hráčům. Druhou variantou je tzv. „*Volně přístupná beta*“, kdy je hra přístupná všem, kteří se zaregistrovali pro beta testování.

Na základě zpětné vazby od beta testerů může vývoj buď znovu zamířit do fáze předprodukce kvůli negativním ohlasům, problémům či jiným plánům, nebo může přejít do fáze vydání, ve které je hra zpřístupněna veřejnosti.

Vydání

V poslední fázi je hra kompletní a připravená k vydání. V tomto okamžiku je klíčové rozhodnout, zda má vývoj pokračovat ve formě přidání dalšího obsahu a opravování chyb, nebo je čas začít plánovat práci na novém projektu. V praxi se tento rozhodovací proces obvykle řeší tím, že jsou vývojáři rozděleni do dvou týmů. Jedna část týmu se nadále věnuje podpoře a vývoji již vydané hry, zatímco zbývající část pracuje na nových projektech.

3.2 Vývojové herní prostředí

Herní engine je základním nástrojem v herním vývoji, který poskytuje klíčové funkcionality a prostředky pro tvorbu počítačových her. Tento softwarový systém zjednodušuje proces vývoje her a umožňuje vývojářům zaměřit se na konkrétní implementaci aplikace, místo toho aby museli vytvářet vlastní fyzikální engine, editor či komplexní proces vykreslování.

Na trhu existuje mnoho herních enginů, přičemž mezi nejznámější patří **Unity**, **Unreal Engine**, **CryEngine**, **Godot Engine** a další. Každý z těchto enginů má své vlastní charakteristiky a výhody, což umožňuje vývojářům vybrat ten, který nejlépe vyhovuje jejich potřebám a preferencím.

Tato práce se zaměřuje na využití vývojového prostředí Unity. Bohatým zdrojem informací o tomto enginu je kniha [10] od *Will Goldstone*, který pracuje ve společnosti *Unity*

[4] QA (quality assurance) zkratka pro zajištění kvality výsledného produktu.

Technologies. V této knize je zmíněn vznik Unity enginu a obsahuje detailní popis jeho klíčových elementů, z nichž tato práce čerpá.

Unity Engine

Toto vývojové prostředí bylo vytvořeno s úmyslem usnadnit herní vývoj a vytvořit uživatelsky přívětivější prostředí pro herní vývojáře. Původě sloužil jako herní engine pro *Mac OS*, avšak díky pozitivní zpětné vazbě od vývojářů a motivací k vytvoření definitivního nástroje pro tvorbu her, začal Unity engine podporovat i další platformy.

V dnešní době se jedná o multiplatformní herní engine vlastněný společností *Unity Technologies*. Podporuje vývoj her pro mobilní platformy, konzole, stolní počítače, virtuální realitu i webové aplikace *WebGL* [13]. Umožňuje tvorbu 2D i 3D her a dalších simulací, které se využívají nejen ve filmech, ale i v průmyslovém odvětví, architektuře a armádě. Engine je napsán v jazyce `C++` a převážně používá jazyk `C#` jako své skriptovací *API*[5].

Editor

Editor Unity je rozdělen do více částí, představujících okna, která lze libovolně přesouvat a měnit jejich velikost. Poskytuje uživatelsky přívětivé prostředí pro vývoj her a aplikací. Jeho flexibilita umožňuje vývojářům přizpůsobit editor jejich potřebám. Editor se skládá z následujících částí [8], které jsou představeny na obrázku 3.2:

- **Průzkumník projektu (Project Browser)** obsahuje všechny importované nebo vytvořené assety. Tvoří ho adresářová struktura, kde jsou umístěny textury, modely, skripty, shadery, prefaby a další soubory. Umožňuje jednoduše vytvářet *assety* přímo v okně průzkumníku a následně je přetáhnout do herního prostředí.
- **Inspektor (Inspector)** zobrazuje všechny komponenty, které charakterizují daný objekt, ve kterých lze modifikovat jejich parametry pro dodatečné ladění i během spuštění.
- **Pohled na scénu (Scene View)** vizualizuje náhled na editované herní prostředí. Funguje na principu *WYSIWYG*[6]. Nabízí různé nástroje pro editaci a tvorbu světa, včetně posouvání, selekce, rotace a změny měřítka objektů. Tento pohled také poskytuje možnosti filtrování, stínování a dalších vizualizací. Navíc disponuje užitečným nástrojem pro modelování terénu.
- **Herní pohled (Game View)** umožňuje vývojáři spustit aktivní scénu v reálném čase a interagovat s ní bez nutnosti zdlouhavé kompilace. Tento pohled také zobrazuje statistky, jako je grafické využití nebo nastavení zvuků.
- **Hierarchie (Hierarchy)** zobrazuje všechny herní objekty ve scéně, strukturované do nadřízených a podřízených objektů. Každý objekt může mít více podřízených objektů, a každý z nich je závislý na svém nadřízeném objektu, což ovlivňuje jeho transformaci a existenci ve scéně.

[5] Zkratka API (Application Programming Interface) neboli rozhraní pro programování aplikací. Jedná se o sadu definicí, nástrojů a protokolů, které umožňují různým softwarovým aplikacím komunikovat mezi sebou.

[6] WYSIWYG (What you see is what you get) je anglická zkratka pro „Co vidíte, to dostanete“.

Obrázek 3.2: Představení Unity editoru. Barevně jsou zvýrazněny a popsány již zmíněné elementy editoru.

Komponenty

Unity je postaveno na modulárních komponentech, které definují vlastnosti a chování herních objektů [8]. Tyto komponenty disponují předem definovanými proměnnými, ke kterým lze skrze danou komponentu přistoupit. Mezi základní komponenty patří:

- **Transformace (Transform)** definuje základní vlastnosti objektu v prostoru, jako je pozice, rotace a měřítko ve světě. Pokud objekt v hierarchii nemá nadřízený objekt, tak se jedná o pozici ve světových souřadnicích. V opačném případě se jedná o lokální pozici, která je relativní vůči nadřízeným objektům.

- **Geometrie modelu (Mesh Filter)** drží referenci pro konkrétní geometrii 3D modelu.

- **Komponenta vykreslení (Mesh Renderer)** umožňuje objektům ve scéně získat vizuální podobu. Hlavním účelem této komponenty je přiřazování materiálů a textur k geometrii objektu, což definuje jeho vzhled a způsob, jakým je zobrazen ve hře. Dále poskytuje možnost nastavení osvětlení a stínů, což ovlivňuje, jak bude objekt reagovat na světelné podmínky ve scéně.

- **Kolize (Collider)** vytváří model kolize v prostoru, který slouží pro fyzikální výpočty a detekci průniku s ostatními kolizními objekty. Unity poskytuje základní tvary kolizí v podobě primitivních tvarů nebo tzv. „*Mesh Collider*“, který vytváří kolizní model na základě geometrie objektu.

- **Fyzikální komponenta (RigidBody)** modifikuje transformaci objektu na principu fyzikální simulace, kterou ovlivňuje například gravitace, síla zrychlení, síly jiných entit, které působí na objekt a tak podobně.

- **Skript** definuje chování herního objektu a lze ho přidat pouze k objektům, které jsou rozšířeny o rozhraní *MonoBehaviour*[7]. Pokud je to možné, tak by programátor měl psát skripty modulárně, pro jejich znovu použití u jiných objektů.
- A mnoho dalších ...

Prefab

Unity využívá tzv. systém „*Prefabů*“[8], který slouží k specifické konfiguraci a sestavení herního objektu [10, 8]. Každá nová instance prefabu dědí jeho výchozí vlastnosti a dokonce umožňuje jejich modifikaci pro konkrétní instance. Proto jsou užitečné pro opakované použití stejných prvků na různých místech, kde nastavení parametrů může být identické či upravené podle potřeb. Souhrn základních výhod tohoto systému je následující:

- **Znovupoužitelnost** – Prefaby umožňují opakované použití stejných prvků, což zjednodušuje vývoj, údržbu a tvorbu instancí za běhu aplikace.
- **Aktualizace** – Jakmile je prefab upraven, tak se všechny jeho instance implicitně aktualizují ve všech scénách.
- **Snadná správa** – Prefaby mohou zahrnovat složitější struktury, které obsahují více objektů, komponent a dalších nastavení, což usnadňuje správu a organizaci objektů.

Scény

Scéna v Unity reprezentuje virtuální prostor, který uchovává všechny informace o přítomných objektech a dalších nastavení. Každá scéna je označená příslušným indexem, kde nejnižší index reprezentuje výchozí scénu. Pro statické objekty ve scéně je možné předem vypočítat světla a uložit je v podobě *assetu*. Existují dva konkrétní režimy pro načítání scén, konkrétně *Single* (Jednotný režim) a *Additive* (Aditivní režim). Jednotný režim načte pouze jednu scénu a tím nahradí tu aktuální. V druhém případě aditivní režim umožňuje přidávat další scény k té stávající, což lze například využít k načítání herních úrovní, které se člení do více částí. V nastavení projektu je nutné specifikovat, které scény jsou zahrnuty do finální kompilace.

Licence

Unity podporuje menší projekty a za určitých podmínek nevyžaduje po vývojářích žádné poplatky, ovšem se značnými omezeními. Pro potřeby větších projektů nebo při dosažení určitých tržeb Unity nabízí licence ve formě předplatného obchodního modelu[9].

- **Osobní verze (Personal)** Tato licence je vhodná pro jednotlivce nebo malé týmy s ročními tržbami nebo rozpočtem nižšími než 100 000 $. Vývoj aplikací je omezen na určité platformy bez možnosti vývoje pro herní konzole. Každý projekt vytvořen s touto licencí musí obsahovat úvodní logo Unity.

[7] *MonoBehaviour* nabízí základní funkce pro definování životního cyklu objektů ve scéně.
[8] Prefab je zkratka pro „prefabricated“ (předem vyrobený) a označuje předem vytvořený objekt nebo skupinu objektů, které lze opětovně využít.
[9] Další informace o Unity licencích na https://unity.com/pricing/compare-plans.

- **Profesionální verze (Professional)** Pokud tržby překročí stanovený limit, je nutné zakoupit licenci Unity Professional, která nabízí nové funkcionality. Tato verze umožňuje vývoj pro herní konzole, odstranění nebo modifikaci úvodního loga, vylepšení fyzikálního enginu a poskytuje nástroje pro vývoj *augmentované* reality.
- **Podniková verze (Enterprise)** Tento licenční model je vhodný pro velké společnosti a projekty s vyšší komplexitou. Poskytuje možnost uzavření individuálních smluv, speciálních výhod a dalších možností přizpůsobení. Navíc umožňuje vývojářům přístup k zdrojovým kódům a za poplatek umožňuje vydání aplikace s upraveným zdrojovým kódem.
- **Průmyslová verze (Industry)** Jedná se o nejkomplexnější a nejrozšířenější licenci, kterou Unity nabízí za € 414 měsíčně nebo € 4554 ročně. Obsahuje všechny služby a nástroje, které Unity poskytuje, včetně školení a profesionální podpory.

3.3 Shadery

V počítačové grafice je shader zvláštní programový kód, který běží na grafickém procesoru *GPU* a je odpovědný za vykreslování a manipulaci s grafickými objekty. Shadery jsou klíčovým prvkem v moderních grafických procesech vykreslování a umožňují vytvářet různé vizuální efekty a optimalizace. Především se využívají k modifikaci geometrie modelů a zbarvení jednotlivých pixelů. Výsledkem práce shaderů je obraz, který uživatel vidí na obrazovce. Tato práce především čerpá z knihy [14], která vysvětluje funkcionalitu shaderů v Unity.

Rendrovací pipeline

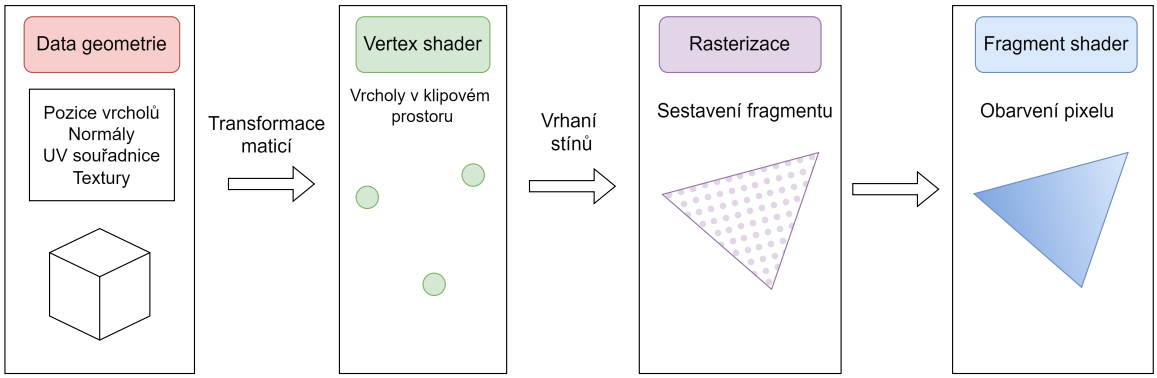
Render pipeline tvoří sériový proces, kterým prochází grafická data při vykreslování scény. Unity jako jeden z mála enginů nabízí více druhů těchto procesů. Jako výchozí je nastaven *Built–in Render Pipeline (BRP)*, který poskytuje základní funkcionalitu a průměrnou náročnost na hardware. Na mobilní platformy a slabší zařízení se specializuje *Universal Render Pipeline (URP)*. Nejmodernější herní tituly využívají *High Definition Render Pipeline (HDRP)* k simulaci realistického a fyzikálního vykreslování. Výchozí render pipeline tvoří několik fází, kde každá z nich má svoji specifickou roli a přispívá k vytváření finálního obrazu [12]. Typické tři základní fáze jsou následující[10]:

- **Culling** rozhoduje o tom, zda je smysluplné některé objekty vykreslovat. Pokud se 3D modely nacházejí mimo komolý pohled kamery nebo jsou plně zahaleny jinou geometrií, tak je nevykresluje a tím šetří výpočetní čas [7].
- **Vykreslování (Rendering)** je nejnáročnější proces, při kterém se vykreslují pixely podle jejich vlastností, vlivu světla ve scéně a dalších výpočtů.
- **Post-processing** je operace, která aplikuje dodatečné efekty pro výsledné fragmenty obrazu.

Fáze vykreslování je velmi komplexní proces. Dělí se do několika dalších částí, které jsou zobrazeny na obrázku 3.3. Na začátku vykreslování se inicializují všechny objekty a jsou

[10] Více o fázích vykreslovaní a render pipeline na https://docs.unity3d.com/Manual/render-pipelines.html.

načteny jejich vrcholy, normály, *UV* souřadnice a další příslušné informace o 3D modelu. Vrcholy objektů jsou nejprve transformovány pomocí modelových transformací, což jsou posuny, rotace a škálování aplikované na vrcholy vzhledem k jejich lokálním souřadnicím. Poté jsou transformované vrcholy převedeny do souřadnicového systému kamery pomocí pohledové transformace. Tato transformace simuluje pohled kamery na scénu a umožňuje vykreslení scény z jejího pohledu. Nakonec jsou vrcholy v pohledovém prostoru převedeny do klipového prostoru *(clip space)* pomocí perspektivní projekce. Tato projekce simuluje perspektivu a vytváří efekt hloubky ve scéně. Vrcholy, které jsou mimo oblast komolého pohledu, jsou odstraněny, což optimalizuje vykreslování a zlepšuje výkon. Následně po transformaci vrcholů do klipového prostoru nastane fáze **rasterizace**, což je proces při kterém se 3D geometrie převede na 2D obrazovku. Tyto fragmenty jsou pak zpracovány ve *fragmet shaderu*, který určuje jejich konečnou barvu. Poté co data projdou *vertex shaderem*, nastává výpočet stínů podle modifikovaných vrcholů.



Obrázek 3.3: Shrnutí procesu vykreslování.

Vertex shader

Tento shader je naprosto fundamentální pro zobrazení geometrie. Na vstup postupně přijímá jednotlivé vrcholy každého objektu a pomocí transformačních matic je transformuje do 2D souřadnic, podle kterých se pixel vykresluje na obrazovce. Dokáže manipulovat s pozicemi vrcholů a *UV souřadnicemi* textur. Používá se převážně k deformacím, či jiným změnám tvaru daného 3D modelu. Nedokáže ovšem vytvořit nebo přidat k modelu žádné nové vrcholy. Využívá se například pro animování vody, deformaci terénu či simulaci větru.

Fragment shader

Plně definuje výsledný vzhled jednotlivých fragmentů. Barvu pixelu modifikuje přidaným světlem, stíny, lesklostí povrchu, průhledností, normálními mapami, které tvoří iluzi dodatečné hloubky a mnoho dalších. Dokáže měnit hodnoty v Z-bufferu a tím upravovat jejich hloubku. Při vykreslování fragmentů, je shader omezen pouze na data, které reprezentují fragmenty, tudíž nedokáže modifikovat geometrii modelů. Slouží hlavně k dodatečnému zpracování obrazu. Využívá se například pro rozmazání obrazu, dobarvování, zvýraznění hran, zářivé efekty, změnu kontrastu, jasu či sytosti a tak podobně. Především se používá k stínování a osvětlení nebo animování časových efektů.

Geometry shader

Přijímá vstupní geometrii ve formě 3D primitiv [11] a na bázi těchto základních modelů dokáže generovat novou geometrii či měnit tu stávající. Pracuje přímo s jednotlivými vrcholy geometrie. Je využíván pro různé efekty, například rozpad objektu po explozi, procedurální geometrie nebo simulace částic.

Tesselation shader

Tento typ shaderu umožňuje dynamicky měnit rozlišení geometrie v reálném čase. Používá se především k vytváření složitých povrchů nebo terénů s vysokou úrovní detailu. Pracuje s tzv. „*tesselací*“, což je proces, který rozšiřuje jednoduchou geometrii na složitější tvary. Tím umožňuje vytvářet hladké křivky a zakřivené povrchy. Proces tesselace funguje tak, že vstupní geometrii rozdělí na menší části, které jsou následně vyplněny novými body, čímž vytvářejí vyšší úroveň detailu. Oproti geometrickému shaderu nepracuje pouze s vrcholy ale přímo s trojúhelníky.

Compute shader

Slouží k paralelním výpočtům na *GPU*, které jsou nezávislé na grafickém kontextu, tudíž bez potřeby spouštět celý *rendering pipeline*. Převážně se používá pro *GPGPU* algoritmy a obecné zrychlení procesu vykreslování. Využívá se například pro simulaci kapalin, částic, vegetace a tak podobně.

Stíny

V Unity je pro vytváření stínů používána technika *Shadow mapping*[11], která počítá stíny v reálném čase. Při tomto výpočtu se nejprve provede vykreslení scény z pohledu světla do hloubkové mapy (textury). Tato mapa obsahuje hloubkové informace o všech viditelných objektech z pohledu světla. Poté, co je získána hloubková mapa, se každý pixel ve scéně transformuje do prostoru kamery z pohledu hráče [11]. Pro každý pixel se poté zjistí, zda je zastíněn od světla podle informací uložených v hloubkové mapě. Nakonec se stínované pixely vykreslí na obrazovku pomocí shaderu, který interpoluje stínování mezi pixely podle vzdálenosti od světla, což vytváří dojem stínu na objektech ve scéně.

3.4 Metody vykreslování

Tato sekce obecně popisuje metody a jejich principy, které lze uplatnit v netriviálním vykreslování a vykreslování portálů. Na začátku je vysvětlen klíčový pojem stencil buffer a jeho využití v grafickém zpracovávání. Poté je zmíněno k čemu se používá metoda ořezávání a její využití pro netriviální vykreslování. Na konci je vysvětlena metoda, která využívá více kamer k vytvoření portálové iluze.

Stencil buffer

Stencil buffer má své vyhrazené místo v paměti na grafické kartě, které původně reprezentoval pouze 1 bit. S pokrokem grafického hardwaru se tato hodnota zvýšila až na jeden bajt[12].

[11] Více o této metodě na https://docs.unity3d.com/Manual/shadow-mapping.html.
[12] Více o stencil architektuře na https://learnopengl.com/Advanced-OpenGL/Stencil-testing.

Realizace jednobitového bufferu sloužila k označení pixelů hodnotou 1 nebo 0 pro vytvoření jednoduché masky, podle které se vyhodnocuje zda daný pixel vykreslit nebo zahodit. S technologickým pokrokem se velikost *stencilového bufferu* rozšířila a umožnila vytvářet pokročilejší efekty. Mezi tyto efekty patří například iluze konstruktivní geometrie *(CSG)*, zvýraznění obrysů, pokročilé maskování nebo může sloužit k zjednodušenému výpočtu pro vytvoření stínů [5].

Konfigurace

Unity nabízí spoustu možností pro specifické nastavení *stencily*. Základním klíčovým slovem je `Ref` stanovující hodnotu, která se porovnává při *stencilovém* testu s aktuální hodnotou ve *stencilovém* bufferu pro daný pixel. Typ porovnání stanovuje slovo `Comp`. Klíčová slova `ReadMask` a `WriteMask` fungují na bázi bitové masky. O výběr operace při úspěšném provedení *stencilového* testu rozhoduje `Pass`, v opačném případě rozhoduje `Fail`. Při neúspěchu hloubkového testu lze změnit referenční hodnotu za pomocí `Zfail`. Výchozí hodnota každého pixelu ve *stencil bufferu* je nastavena na 0. Obecný příklad nastavení stencil bufferu viz tabulka 3.1, která je vytvořena podle dokumentace[13].

Nastavení stencil v shaderu		
Klíčové slovo	Datový typ	Funkcionalita
Ref	Integer <0,255>	Referenční hodnota.
ReadMask	Integer <0,255>	Binární maskování vstupů.
WriteMask	Integer <0,255>	Binární maskování výsledku.
Comp	Porovnávací operace	Srovnání s referenční hodnotou.
Pass	Stencil operace	Změna hodnoty ve stencil bufferu.
Fail	Stencil operace	Změna hodnoty ve stencil bufferu.
ZFail	Stencil operace	Změna hodnoty ve stencil bufferu.

Tabulka 3.1: Nastavení a vysvětlení stencil parametrů.

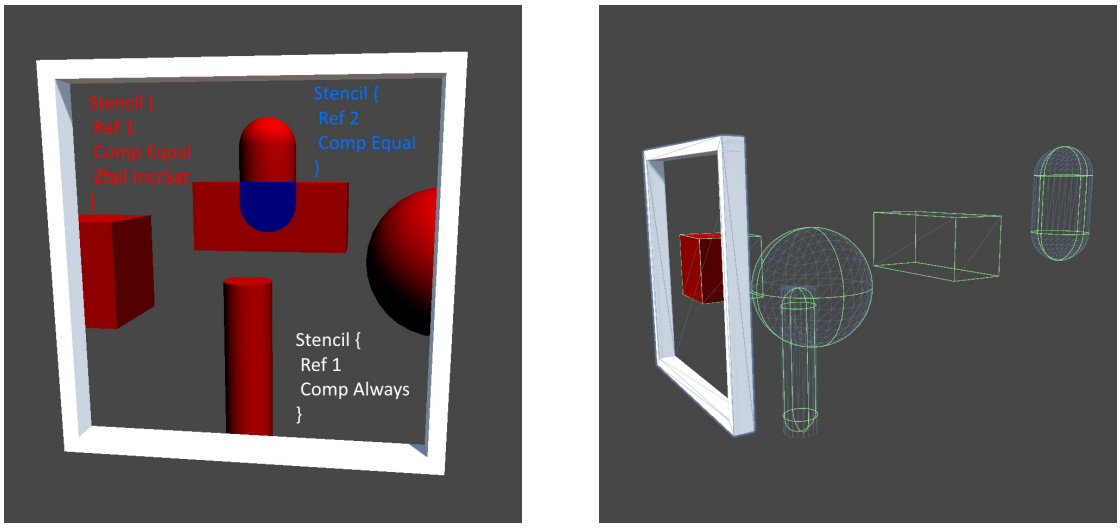
Stencil test

Na základě `Comp` porovnávací funkce, se porovná referenční hodnota s aktuální hodnotou pro daný pixel, která se nachází v stencil bufferu. Podle úspěšnosti testu se rozhodne, zda se daný pixel modifikuje ve fragment shaderu [16]. Stencilový test se vždy provádí před tzv. „hloubkovým testem“, který vykresluje objekty podle hloubky ve scéně. Hloubkový test určí viditelnost geometrie z pohledu kamery a proto zbytečně nevykresluje pixely objektů, které jsou zahaleny další geometrií. Podle úspěšnosti testů a konfigurace se pixelům změní jejich referenční hodnota a modifikují se jejich vlastnosti. V tabulce 3.2 jsou znázorněny jednotlivé operace *stencilového* testu. Základní použití této metody zobrazuje obrázek 3.4.

[13] Více o konfiguraci stencil bufferu na `https://docs.unity3d.com/Manual/shadow-mapping.html`.

Stencil operace		Porovnávací operace	
Parametr	Ref(x) vs Stencil buffer(y)	Parametr	Ref(x) vs Stencil buffer(y)
Keep	$x = x$	Never	0
Zero	$x = 0$	Less	$x < y$
Replace	$x = y$	Equal	$x = y$
IncrSat	$x + 1$	LEqual	$x \leq y$
DecrSat	$x - 1$	Greater	$x \geq y$
Invert	$\neg x$	NotEqual	$x \neq y$
IncrWrap	$x = 255 \rightarrow 0 : x + 1$	GEqual	$x \geq y$
DecrWrap	$x = 0 \rightarrow 255 : x - 1$	Always	1

Tabulka 3.2: Stencil testování referenční hodnoty s hodnotou ve stencil bufferu.



(a) Červeně jsou znázorněny pixely s referenční hodnotou 1. Modře jsou zvýrazněny pixely s referenční hodnotou 2.

(b) Pohled z boku mimo masku rámu

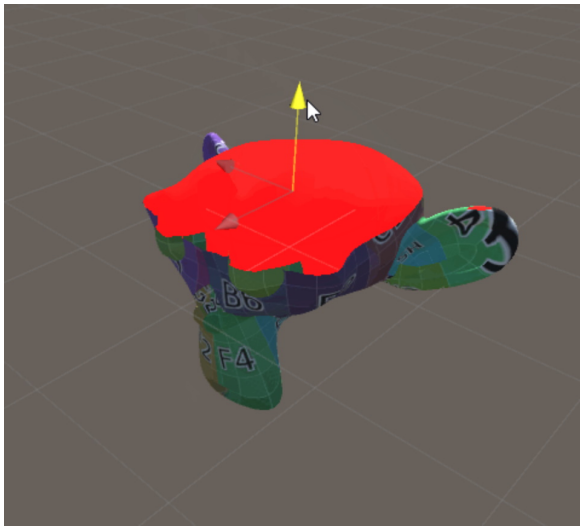
Obrázek 3.4: Ukázka použití stencil bufferu v praxi. Vnitřek bílého rámu představuje masku s referenční hodnotou 1.

Metoda ořezávání

Metoda ořezávání je základní technikou ve výpočetní grafice, která se používá k určení viditelných částí objektů, v závislosti na pozici pozorovatele a rozsahu viditelného pole. Tato metoda se používá například pro optimalizaci vykreslování, jelikož nevykresluje 3D objekty, které se nachází mimo komolý pohled kamery, což značně sníží výpočetní nároky na grafické kartě [11]. V případě netriviálního vykreslování se tato metoda používá při maskování objektů nebo k dynamickému řezání objektů [7], jak je zobrazeno na obrázku 3.5. Tuto metodu lze využít zejména v následujících fázích:

- Ve **vertex shaderu** lze odstranit jednotlivé vrcholy vstupní geometrie, tudíž nedojde ani k rasterizaci.

- Ve **fragment shaderu** lze odstranit jednotlivé pixely, ovšem stíny se stále počítají podle geometrie modelu.



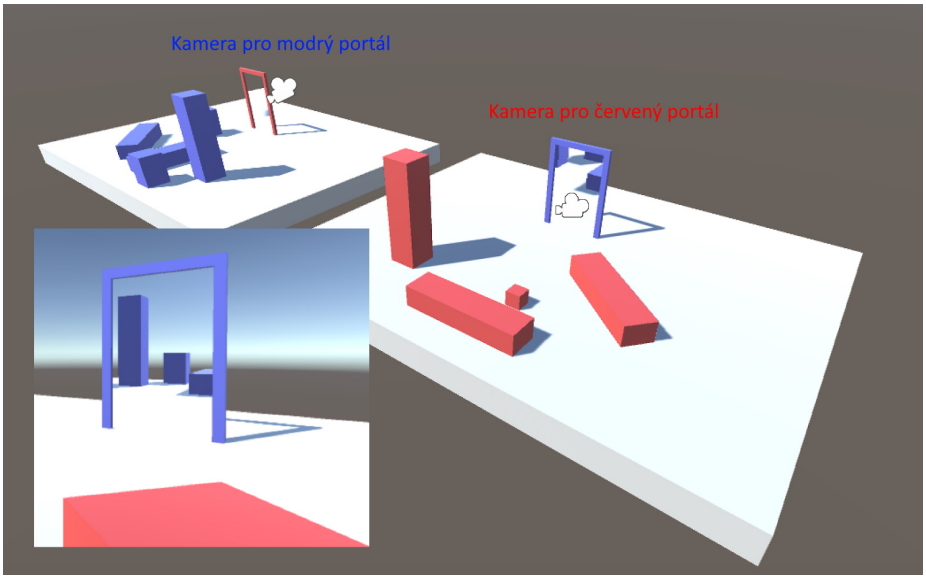
Obrázek 3.5: Ukázka metody ořezávaní. Obrázek je převzatý z https://www.ronja-tutorials.com/post/021-plane-clipping/.

Vykreslování do textury

Tato metoda využívá alternativní kameru, která je umístěna takovým způsobem, aby vykreslovala určitou část scény relativně k hráči. Získaný obraz z této kamery se uloží do textury, která je následně použita pro materiál portálu. Tento způsob vykreslování lze jednoduše aplikovat například ve hře, kde existuje kamerový systém, který přenáší obraz na displej ve scéně.

Původní portály ve hře Portal 2 fungovaly na tomto principu jak zmínil jeden z hlavních vývojářů[14]. Nicméně během vývoje se vývojáři rozhodli přejít na použití *stencil bufferu* kvůli určitým komplikacím. Prvním velkým problémem bylo zvládnutí *antialiasingu* textury. Pokud se hráčova kamera přiblížila k portálu na určitou vzdálenost, *antialiasing* způsoboval nežádoucí vizuální artefakty. Další závažný problém se týkal alokace paměti, protože textury zabíraly velkou část tohoto prostoru [1]. Proto při nešťastném rozmístění portálů může rekurzivní vykreslování výrazně zatížit výpočetní výkon, protože s každou rekurzí vznikají nové textury.

[14]Prezentační video od vývojářů hry Portal 2 dostupné na https://youtu.be/riijspB9DIQ.



Obrázek 3.6: Ukázka portálu na principu vykreslování do textury. Předloha obrázku dostupná z: https://tomhulton.blogspot.com/2015/08/portal-rendering-with-offscreen-render.html.

3.5 Procedurální generování

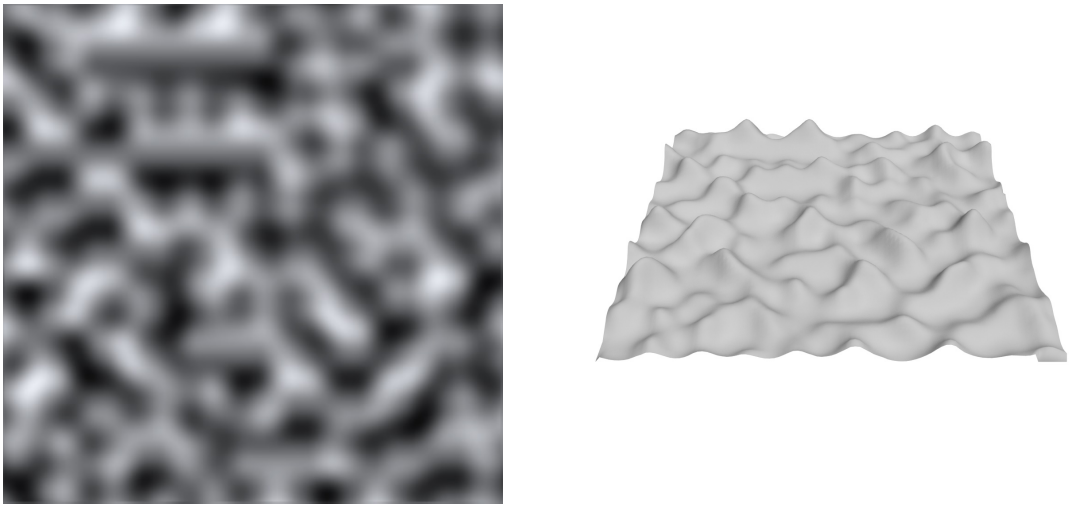
V tomto kontextu se procedurální generování využívá pro tvoření téměř nekonečných světů, přičemž každý z nich vypadá víceméně odlišně od těch ostatních. Především se liší v podobě prostředí, jelikož jsou všechny entity rozmístěny jiným způsobem na novém terénu.

Procedurální generování přináší spoustu výhod. Jelikož je svět generován za běhu aplikace algoritmicky a ze stejných *assetů*, tak ho není nutné celý úkládat a tím šetří místo na disku. Usnadňuje práci designérům úrovní, protože nemusí světy upravovat ručně.

Této práci na toto téma výrazně přispěly knihy [20, 18], které vysvětlují základní principy a metody procedurálního generování a práci s 3D geometrií.

Perlinův šum

Perlinův šum je technika generování pseudonáhodných hodnot, které jsou spojeny tak, aby vytvářely plynulé a přirozené vizuální efekty. Tuto metodu vytvořil *Ken Perlin* v roce 1983 [21]. Perlinův šum je často používán v počítačové grafice a v různých aplikacích, kde je potřeba generovat realistické náhodné vzory, jako je simulace terénu, textur nebo animací. Běžně se tento šum používá ve 2D, ovšem existují i další metody, které fungují na stejném principu ve více rozměrech [4]. Generuje se pomocí interpolace hodnot z náhodně vygenerovaného pole vektorů, které je označováno jako *gradientové* [6]. Výsledné interpolační hodnoty tvoří mřížku (2D pole) s určitým rozptylem, v praxi to jsou většinou hodnoty od 0 až 1. Tyto hodnoty lze využít jako gradient, kde nejnižší či nejvyšší hodnota stupnice představuje černou nebo bílou barvu, tudíž hodnoty mezi reprezentují stupně šedi. Reprezentace šumu je zobrazena na obrázcích 3.7. Další zásadní vlastnost Perlinova šumu je konzistentnost, jelikož pro stejný vstup, funkce vrací stejnou hodnotu.



(a) Zobrazení šumu v podobě textury.

(b) Zobrazení šumu v geometrické podobě.

Obrázek 3.7: Ukázka základního Perlinova šumu poskytnutý Unity.

Modifikace Perlinova šumu

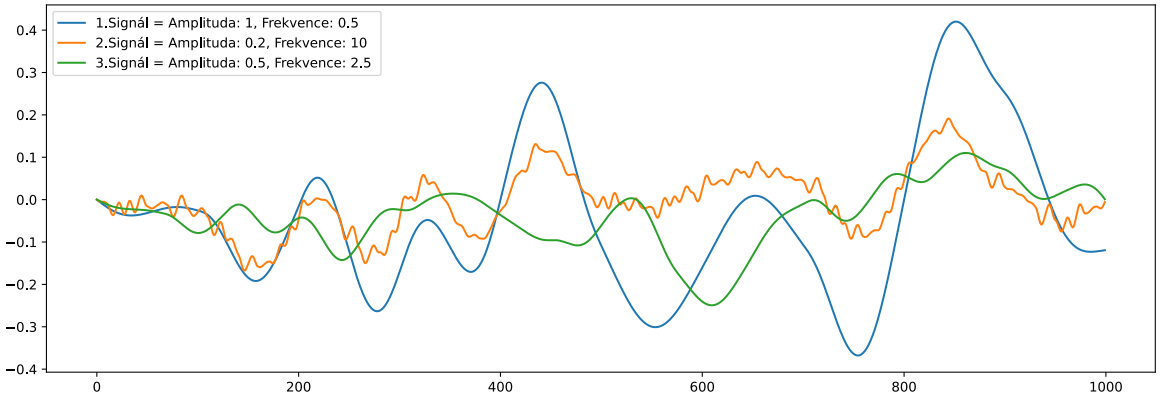
Tvorba detailnějšího terénu vyžaduje dodatečné úpravy Perlinova šumu. Proto existují jisté metody, které využívají tento šum pro dosažení specifických efektů [9]. Následující parametry značně ovlivňují podobu šumu:

- **Frekvence** určuje, jak rychle se změny v šumu opakují. Vyšší frekvence zapříčiní rapidní změny v detailech, zatímco nižší frekvence tvoří větší a hladší detail.
- **Amplituda** ovlivňuje výšku (intenzitu) vrcholů a nížin v šumu. Vyšší amplituda znamená, že vrcholy jsou výraznější a doliny jsou hlubší, což vytváří kontrastnější obraz.
- **Oktávy** reprezentují různé vrstvy nebo–li úrovně šumu, které jsou mezi sebou kombinovány. Každá oktáva obsahuje šum s jemnějšími nebo hrubšími detaily. Když se oktávy kombinují, dochází k vytvoření komplexnějšího šumu s různými úrovněmi detailů.
- **Posun** slouží k odsazení hodnot. Perlinův šum si lze představit jako enormní mapu hodnot, tudíž ji lze rozdělit na části podle souřadnic, na kterých je možné se pohybovat. Tato vlastnost je využitelná například při navazování terénu.

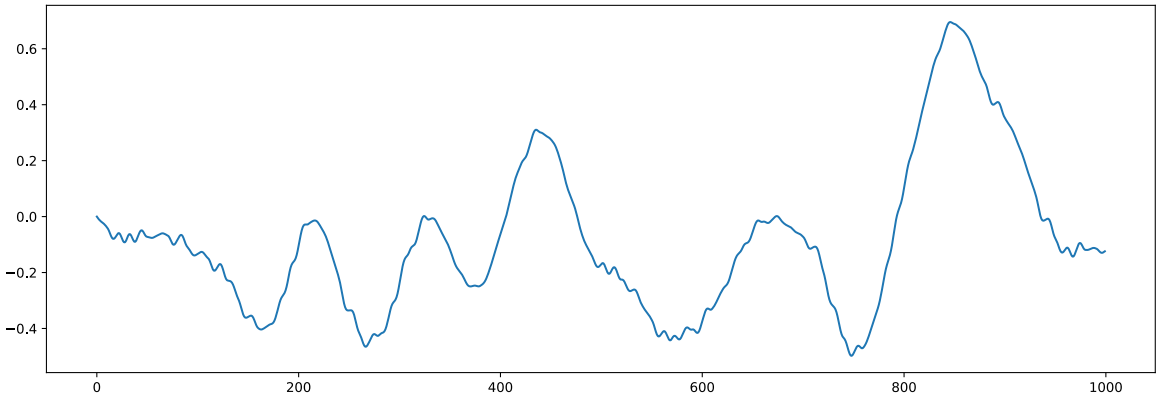
Na demonstračním obrázku 3.8a jsou znázorněny průběhy signálů, které využívají původní hodnoty Perlinova šumu k modifikacím podle parametrů (viz legenda grafu). V grafu 3.8b lze pozorovat finální modifikovaný signál, který vznikl kombinací těch předchozích. Následující obrázky demonstrují využití parametrů a Perlinova šumu v podobě textury. Každý šum byl vygenerován s těmito parametry v tabulce 3.3, kromě jeho změny, která je uvedena u jeho obrázku 3.9.

Základní parametry		
Oktávy	Amplituda	Frekvence
4	0.6	5

Tabulka 3.3: Základní nastavení parametrů pro modifikaci šumu.



(a) Perlinův šum s dodatečnými parametry

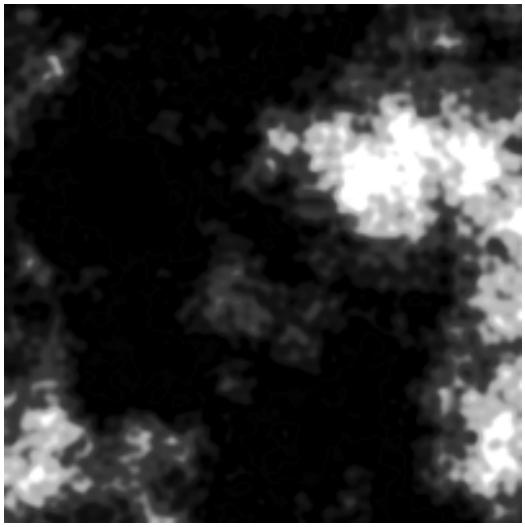


(b) Kombinace signálů s různými parametry

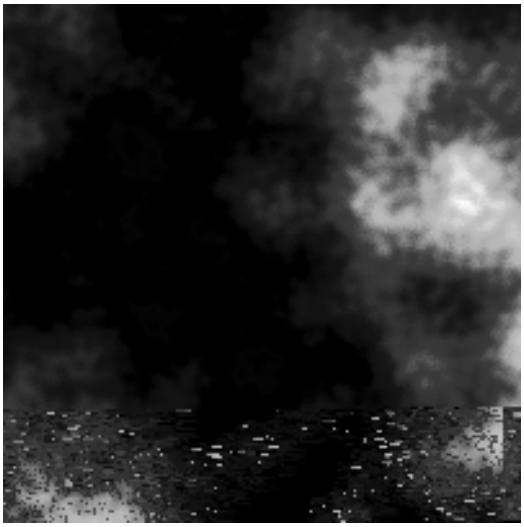
Obrázek 3.8: Znázornění modifikace Perlinova šumu

(a) Původní nastavení bez změn

(b) Změna: Oktávy = 2



(c) Změna: Amplituda = 0.6



(d) Změna: Frekvence = 5

Obrázek 3.9: Příklady modifikovaného Perlinova šumu v podobě 2D textury.

Tvorba geometrie

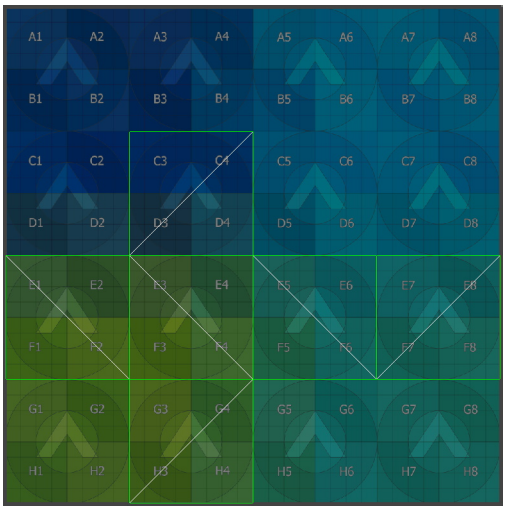
Unity umožňuje generovat *mesh* (geometrii) za běhu aplikace. Mesh se skládá z vrcholů a hran, které dohromady tvoří *polygony* (mnohoúhelníky), které se skládají z trojúhelníků. Pro vytvoření nové geometrie jsou zapotřebí následující čtyři jednorozměrné pole:

- **Pole vrcholů**, kde každý element obsahuje vektor o třech složkách (x, y, z), které reprezentují pozici v prostoru.
- **Pole trojúhelníků**, kde každý element indexuje jeden vrchol, který v trojicích určuje hrany spojující vrcholy a vytvářejí daný trojúhelník.
- **Pole normál**, kde každý element obsahuje vektor o třech složkách (x, y, z), který určuje směr povrchu ve 3D prostoru u každého vrcholu. Zpravidla jsou normály normalizované vektory a kvůli tomu jsou nezávislé na měřítku. Jsou důležité pro různé

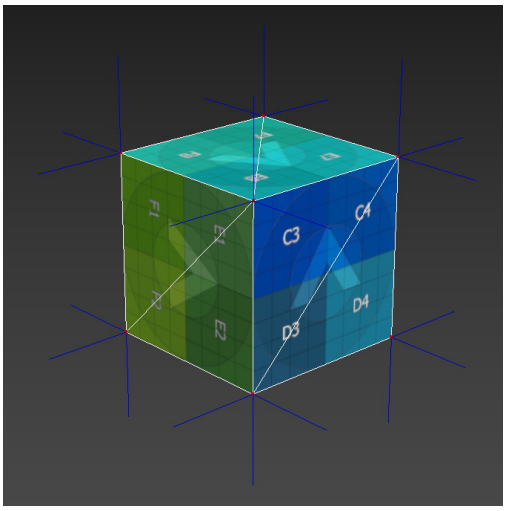
grafické a fyzikální výpočty, jako je osvětlování a stínování. Každý trojúhelník má dvě strany a normály pomáhají určit, zda je daný trojúhelník natočen směrem k pozorovateli (přední strana) nebo od něj pryč (zadní strana). Normály využívá tzv. proces *face culling*, který zbytečně nevykresluje trojúhelníky odvrácené od kamery. To vede k efektivnějšímu vykreslování scény a optimalizaci výkonu grafické karty.

- **Pole UV souřadnic**, kde každý element obsahuje vektor o dvou složkách (x, y), které mapují jednotlivé vrcholy geometrie na síť ve 2D prostoru jak je představeno na obrázku 3.10a.

  Všechny prvky v každém poli jsou závislé na pořadí, jelikož společně postupně definují tvar a vzhled výsledné geometrie jak je zobrazeno na obrázku 3.10b.



(a) UV mapa.



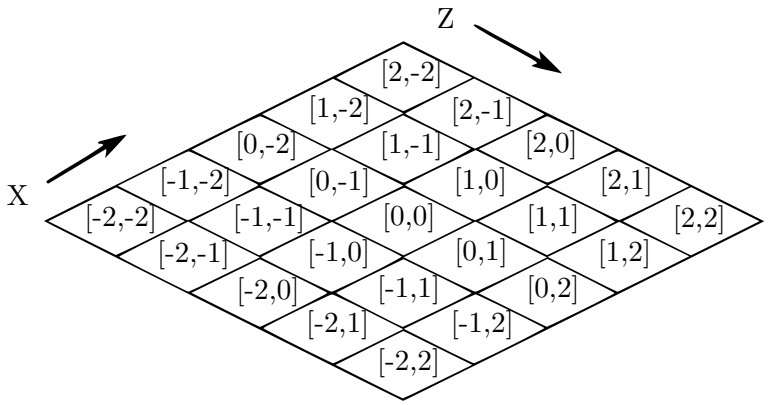
(b) Geometrie s normály (modré čáry) a namapovanou texturou.

Obrázek 3.10: Vlevo je příklad mapování UV mapy na geometrii krychle a na druhé straně znázornění jednotlivých elementů potřebných ke generování.

Rozdělení terénu do částí

Terén se skládá z mnoha vrcholů, tudíž není efektivní ani možné ho vykreslovat jako jeden velký objekt. Každý objekt v Unity se může skládat z maximálně (65 536) vrcholů[15], jelikož jsou jeho vrcholy indexovaný do 16 bitů. Proto je nutné terén rozdělit do více *chunků*. Chunk je část terénu s určitou šířkou, délkou a souřadnicemi. Obsahuje příslušné entity, jako jsou například stromy, kameny, vegetace a tak podobně. Podle vlastních rozměrů a souřadnic reprezentuje částečnou plochu šumu, který tvoří terén celého světa. Každá pozice chunku je definována ve vlastních 2D souřadnicích, ze kterých se počítá transformace do světových souřadnic.

[15] https://docs.unity3d.com/ScriptReference/Mesh-indexFormat.html.



Obrázek 3.11: Reprezentace rozložení chunků podle vlastních souřadnic.

3.6 Navigační systém

Každý inteligentní život založený na umělé inteligenci, který je schopen pohybu v prostoru, musí být řádně definován. Existuje několik přístupů k řešení tohoto problému, ale pro potřeby této práce je nejvýhodnější řešení orientace podle navigační geometrie. Umělá inteligence není schopna libovolně se pohybovat v herním prostoru jako člověk, proto potřebuje informace, které reprezentují virtuální prostor.

Komponenta Navmesh

Navmesh[16] je fundamentálním elementem navigačního systému, který umožňuje umělé inteligenci pohyb v herním prostředí. Jedná se o síť polygonů, která reprezentuje plochu virtuálního prostoru. Tato síť je vytvořena tak, aby umožňovala plynulý pohyb a navigaci herních postav bez toho, aby se dostaly do nepřekonatelných překážek.

Slouží k optimalizaci a zjednodušení procesu rozhodování herních postav o tom, kam se mají pohybovat. Když herní postava potřebuje přejít z jednoho místa na druhé, systém navigace na základě *navmeshe* určí nejlepší cestu, která je volná od překážek a umožní plynulý pohyb postavy. Tím se minimalizuje potřeba složitých výpočtů a zvyšuje efektivitu pohybu.

Agent

V rámci navigačního systému v počítačových hrách se termín „agent“ používá pro virtuální entitu nebo postavu, která se pohybuje ve virtuálním prostoru. Tento agent[17] může být ovládán umělou inteligencí nebo hráčem a má schopnost interagovat s prostředím a pohybovat se v něm.

Agenti se orientují a pohybují na základě navigační geometrie. Když agent potřebuje přejít z jednoho místa na druhé, navigační systém vypočítá optimální trasu, aby se vyhnul překážkám a dosáhl cíle.

V praxi může být agentem postava hráče nebo nepřátelský *NPC*[18]. Každý z těchto agentů má své vlastní cíle a úkoly, které realizuje ve virtuálním navigačním prostoru. Sys-

[16] Navmesh (navigation mesh) je zkratka pro navigační geometrii.

[17] Více informací o funkcionalitě agenta je k dispozici na https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html.

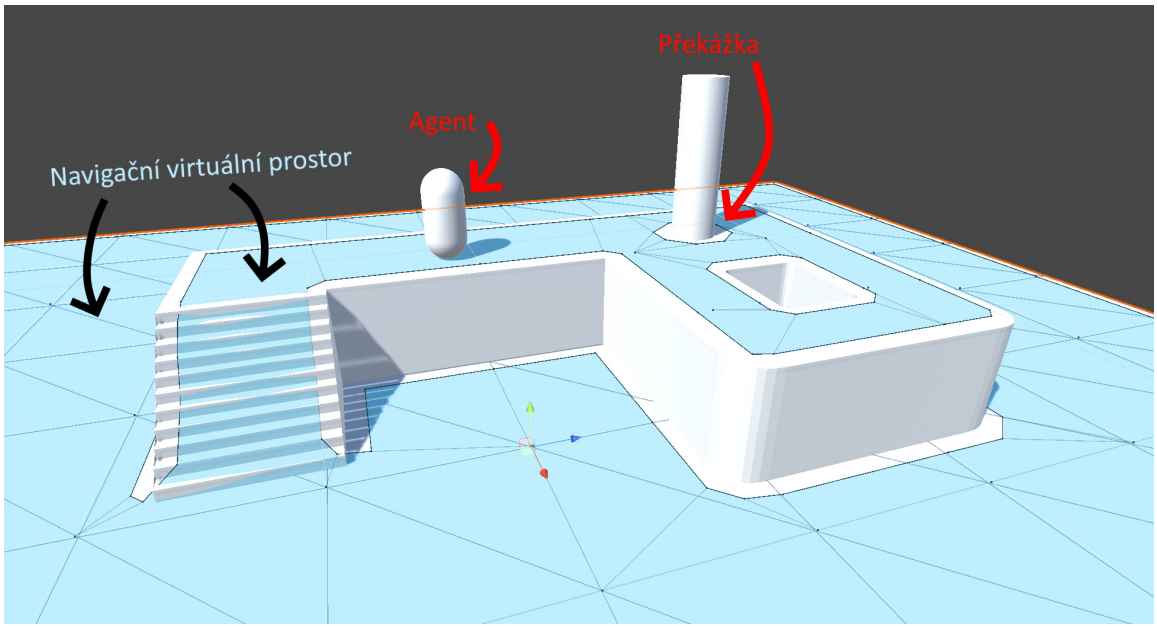
[18] Anglická zkratka (Non-Player Character) reprezentuje postavu, která je ovládána umělou inteligencí.

témy navigace a virtuálního prostoru jsou klíčové při vývoji her, kde je žádoucí, aby se postavy pohybovaly inteligentně a reagovaly na změny v prostředí.

Překážky

Pro korektní pohyb agenta je nutné identifikovat překážky *NavMeshObstacle*[19], které by mohly potencionálně blokovat jeho cestu. Za překážky se nepovažují jen statické objekty, ale i další agenti či jiné pohyblivé objekty, kterým se musí agent vyhnout. Náhlý výskyt dynamické překážky na vypočítané cestě může být často problematický, proto je nutné, aby agent reagoval dostatečně včas a našel novou trasu.

Při vytváření *NavMeshe* se berou statické překážky v potaz a ovlivňují jeho výslednou geometrii jak je vidět na obrázku 3.12. Dynamické překážky vyřezávají oblasti navigačního prostoru za běhu aplikace podle kolizní geometrie. Agent se pohybuje pouze na jeho přiděleném prostoru a nikdy mimo něj.



Obrázek 3.12: Ukázka navigačního prostoru

[19]Více o nastavení této komponenty je k dispozici na https://docs.unity3d.com/540/Documentation/Manual/class-NavMeshObstacle.html.

Kapitola 4

Návrh hry

Cílem návrhu je vytvořit uživatelsky přívětivé prostředí, ve kterém se uživatel snadno orientuje. Uživatelské rozhraní, grafické zpracovávání a zpětná zvuková vazba ve hře hraje důležitou roli, ovšem zábavnou hru tvoří především herní mechaniky [2]. Pokud hra disponuje nejlepší moderní grafikou či stovkami unikátních zvuků, ale nenabízí hráči žádný zábavný element, tak ho pravděpodobně velice rychle omrzí.

Problém nastává při vymýšlení herních mechanik a jejich následné implementování do herního světa. Interakce se světem musí být zábavná a do určité míry i trochu náročná. Pokud hráč dosáhne postupu příliš rychle nebo mu stačí vynaložit minimální úsilí k dosažení cíle, tak o tyto mechaniky ztratí zájem. Pokud nějaká herní mechanika vypadá skvěle na papíře, tak to implicitně neznamená, že se jedná o úžasnou herní mechaniku. Velmi důležitá je následná realizace v herním prostředí, která musí být naprosto bezchybná a plynulá.

Při navrhování herních mechanik pro tuto práci byl nejprve vytvořen jednoduchý herní prototyp, který testoval hratelnost a složitost realizace.

4.1 Koncept hry

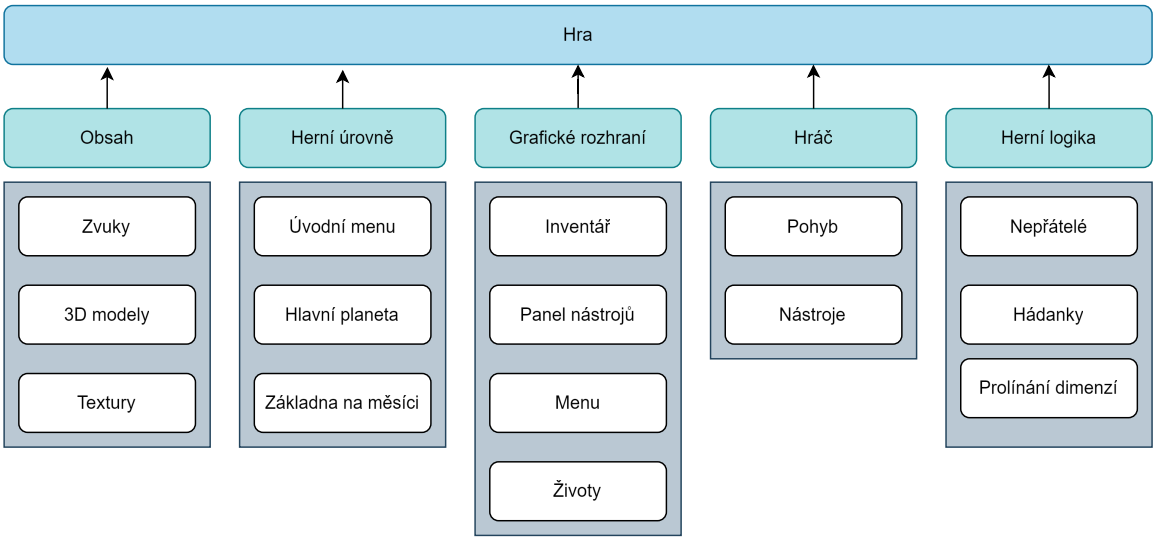
Hra začíná na měsíci, na kterém ztroskotala hráčova vesmírná loď a dopadem se poškodila. Bez toho aniž by hráč opravil problémový modul motoru nemůže pokračovat ve svojí cestě napříč vesmírem a tím dohrát hru. Potřebné materiály pro opravu lodi hráč nalezne na podivuhodné planetě, kde existuje život ve více dimenzích současně. Každá podoba světa představuje jiný ekosystém plný příslušných surovin.

Tato hra obsahuje prvky z her o přežití v podobě průzkumu světa, získávání surovin a jejich následné zpracování pro výrobu nových předmětů. Druhá stránka hry se soustředí na logické uvažování, řešení hádanek a orientaci v prostředí. Proto se spíše přibližuje k hernímu žánru adventur. Ovládá se z třetího pohledu ve 3D světě, ve kterém se může volně pohybovat. Jedná se o hru pro jednoho hráče.

Hlavní herní mechanika spočívá v interakci se světem skrze portály. Ty lze libovolně rozmístit v herní úrovni nebo změnit jejich aktuálně promítanou dimenzi. S portály lze nahlédnout do dalších dimenzí nebo přeměnit specifickou část světa.

Herní mechanismy nevyžadují po hráči komplexní ovládání, reflexní zdatnost nebo hlubokou znalost světa, což umožňuje hráčům se rychle zapojit do hry. Díky její jednoduchosti může být zábavná pro širokou škálu hráčů, bez ohledu na jejich herní preference. Víceméně si tuto hru může v klidu zahrát i hráč, který je nový ve světě videoher.

Shrnutí a návrh jednotlivých částí hry je zobrazeno na obrázku 4.1.



Obrázek 4.1: Návrh herních částí hry

Návrh herního prostředí

Herní svět je procedurálně generovaný a rozdělen do části (chunků), které se postupně generují ve směru pohybu hráče. Velikost mapy je omezena dohledem hráče, ovšem kolem něho se svět neustále generuje, tudíž nikdy nenarazí na jeho konec. Pro každou část světa jsou vygenerovány tři podoby terénu a příslušné entity, které jsou viditelné pouze v určitých dimenzích.

Na této netradiční planetě se navíc vyskytují struktury a pozůstatky původní civilizace, které jsou uchovány v truhlách. Hráč není na této planetě sám, tudíž může narazit na tamější nepřátelský život.

Vlastnosti světů

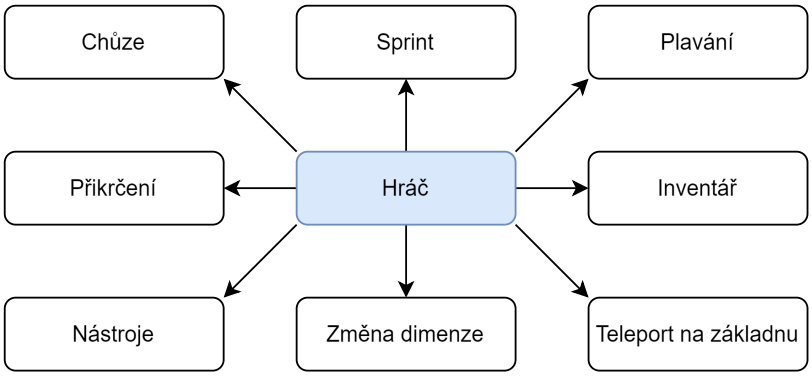
Světy se od sebe výrazně liší nejen vzhledově, ale i charakteristicky. Každá dimenze reprezentuje určitý ekosystém, který se skládá z odlišných rostlin a nerostných surovin. Tyto zdroje hráč potřebuje sbírat k vytváření předmětů.

Dalším klíčovým prvkem prostředí je voda, která mění své vlastnosti podle příslušných dimenzí. V určitých případech může voda výrazně pomoci hráči. Například při úniku proti nepřátelům nebo překonání hlubokých vod. Ovšem v dalších situacích může hráči ublížit či znepříjemnit pohyb.

- **První svět** nabývá podoby běžného lesa plného stromů a rostlin. V čisté vodě hráč může plavat a sbírat barely, které plavou na hladině.
- **Druhý svět** je vyprahlý a plný písku. V tomto pouštním světě není voda, což umožňuje hráči prozkoumat prostor pod vodní hladinou. Navíc je v tomto světě snížená gravitace, kterou lze využít k dosažení vyšších míst.
- **Třetí svět** je plný mlhy a toxické vody, která uděluje hráči poškození a ztěžuje přechod mezi jezery.

Návrh hráče

Hlavní akcí hráče je pohyb, který ovlivňuje jeho průzkum v dynamicky generovaném světě, jak je zobrazeno na diagramu 4.2. Vzhledem k hojnému výskytu vodních ploch je hráči umožněno přeplavat dlouhé úseky jezer. Interakce hráče se světem probíhá prostřednictvím nástrojů a změnou dimenzí. Kdykoliv hráč uzná za vhodné, tak se může teleportovat zpět na základnu pomocí předmětu v inventáři.



Obrázek 4.2: Znázornění hráčových akcí.

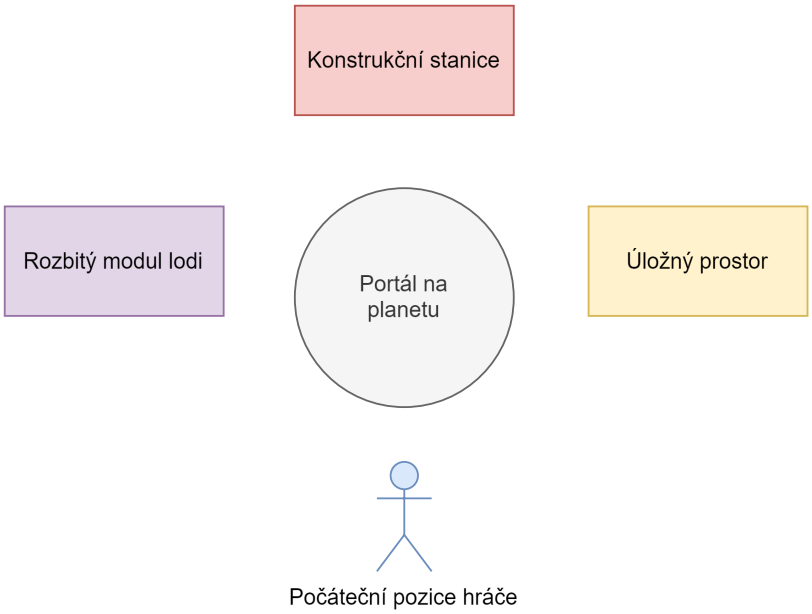
Hráčovi nástroje

Hráč využívá nástroje pro řešení hádanek, získávání surovin a obranu. K dispozici jsou hráči následující nástroje:

- **Sekera** a **krumpáč** slouží k těžení či získávání surovin. Sekera je především určena k těžbě dřeva a sběru rostlin, ovšem lze ji také použít k útoku na nepřátele. Krumpáč dokáže vytěžit jakýkoliv druh nerostné suroviny.
- **Kladivo** umožní hráči vytvořit portál kdekoliv ve světě. Pomocí tohoto nástroje lze portál vystřelit, zničit či měnit jeho dimenzi. Symbol vedle ikony značí, zda je možné vystřelit nový portál, jelikož ve světě může být pouze jeden.
- **Laser** slouží k aktivaci či destrukci energetických materiálů. Laser mění paprsek energie podle světa, ve kterém se hráč nachází. Podle energie ovlivňuje nepřátele. Například modrý paprsek omráčí nepřítele, zelený ho zpomalí a naopak oranžový dodá nepřátelům rychlost a sílu.

Základna na měsíci

Na měsíci si hráč vytvořil základnu do které se lze zpětně teleportovat z hlavní planety. Jedná se o naprosto bezpečné prostředí, ve kterém si hráč může odložit získané předměty a připravit se na návrat na planetu. Měsíc slouží jako hráčova základna, na které si může odložit věci do úložného prostoru nebo vytvářet součástky pro opravu lodi pomocí konstrukční stanice. Uprostřed základny se nachází mechanický teleport, který hráče přesune na planetu měsíce. Detailnější popis základny je na obrázku 4.3.

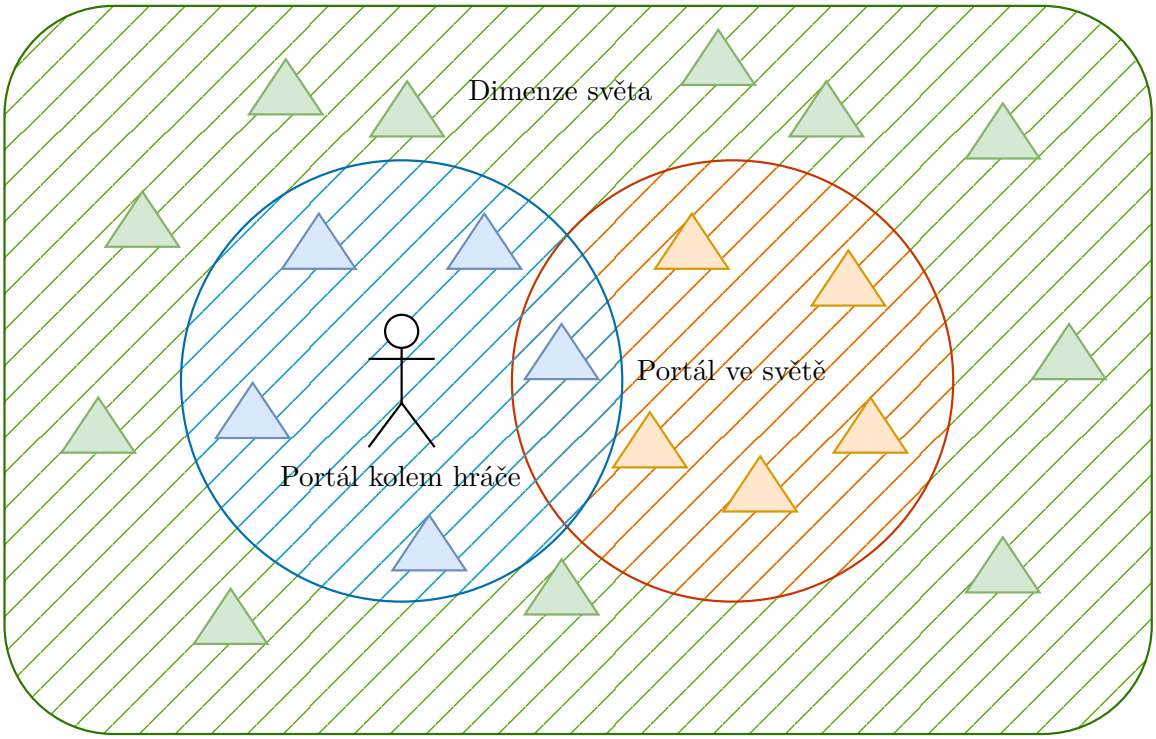


Obrázek 4.3: Vlevo se nachází hráčova loď, kterou musí opravit. Napravo je umístěna hráčova bedna. Uprostřed na zemi leží teleport a nad ním je umístěna konstrukční stanice.

Prolínání světů

Nad celým světem lze uvažovat jako nad třemi sektory, ve kterých hráč může libovolně měnit jejich dimenze či je vzájemně kombinovat. Portály jsou hlavní hráčovy nástroje, které zprostředkovávají interakci se všemi dimenzemi a umožňují jejich vzájemné prolínání. Každý sektor ovlivňuje prostředí s určitou prioritou jak je vidno na obrázku 4.4.

- **Portál kolem hráče:** Hráč má kolem sebe v určitém rádii možnost změnit prostředí na jednu ze tří dostupných dimenzí. V prolínání světů má portál kolem hráče vždy nejvyšší prioritu.
- **Portál ve světě:** Do světa si hráč může umístit jeden kulovitý portál a následně měnit jeho aktuální dimenzi, kterou promítá. Prostředí ovlivňuje s druhou nejvyšší prioritou.
- **Svět mimo portály:** Tento sektor vyobrazuje zbytek světa, který nebyl modifikován hráčovými portály. Prolíná prostředí s nejnižší prioritou.



Obrázek 4.4: Ukázka vzájemného prolínání dimenzí. Trojúhelníky představují herní objekty, které jsou závislé na dimenzích. Modře je znázorněn portál kolem hráče. Oranžově je znázorněn portál ve světě a zbytek prostředí, které není ovlivněno portály je znázorněno zelenou barvou.

Logické hádanky

Hráč využívá přepínání dimenzí k získávání užitečných surovin, které se vykytují pouze v určitých podobách světa.

Na mapě je také možné nalézt truhly s lukrativními odměnami. K jejich odemčení je nutné vyřešit příslušnou hádanku, která po splnění danou bednu hráči zpřístupní. Části hádanek se ukrývají v různých dimenzích nebo reagují na specifický druh energie z určitého světa. Jelikož se světy mezi sebou prolínají, tak k vyřešení logických hádanek je nutné vykonat určité akce v různých dimenzích. Druhy hádanek jsou následující:

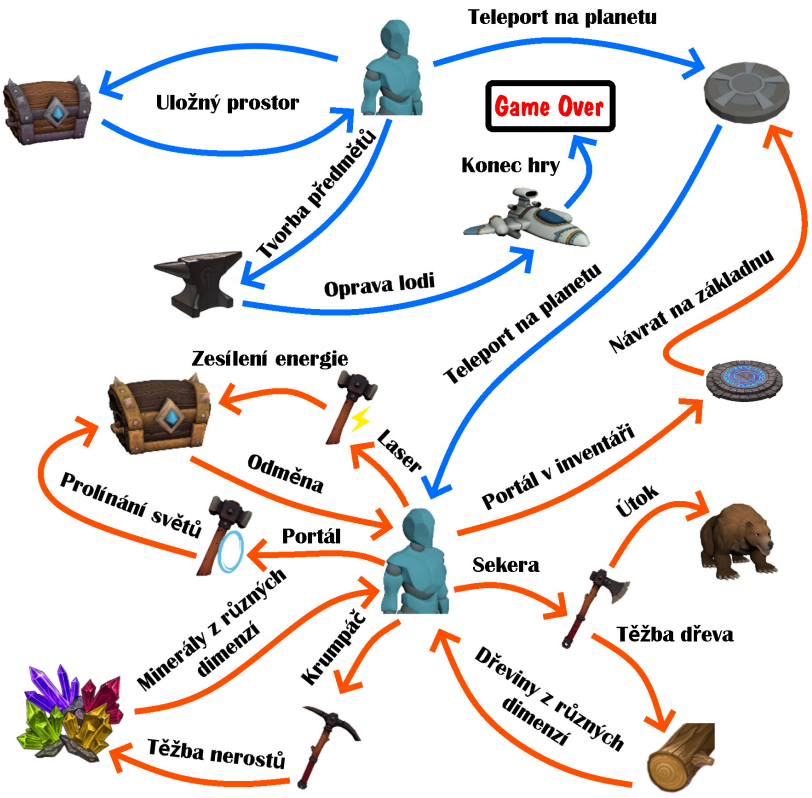
- **Létající krystaly** se vznáší nad truhlou a svojí přítomností znemožňují její odemčení. Každý krystal nabývá jiné zářivé barvy, která reprezentuje energii jedné z dimenzí. Zbarvení krystalu určuje jaký druh energie je schopen krystal zničit. Po zničení všech krystalů se truhla odemkne.
- **Energetické pilíře** reagují na změnu světa v jejich blízkosti. Pokud se pilíř vyskytuje ve správné dimenzi, tak vyzdvihne svůj energetický článek do vzduchu a tím se zaktivuje. Každý pilíř se aktivuje v jiné dimenzi, takže hráč musí využít všechny dostupné portály. Po aktivaci všech pilířů prolne energetický paprsek všechny energetické články a truhla se odemkne.
- **Aktivátor** se vyskytuje na náhodné pozici blízko truhly. Pokud se kolem aktivátoru změní svět na určitou dimenzi, kterou reprezentuje barva vyzařovacích částic, tak aktivátor vyšle paprsek světla na krystal, který odemkne příslušnou truhlu.

- **Energetické štíty** blokují hráči průchod k truhle. Podle zbarvení reagují na určitý typ energie, který je dokáže zničit.
- **Magické koule** se vznášejí nad truhlou a jsou ukryty před hráčovým zrakem. Tyto koule lze spatřit a zničit pouze skrze portál, který si hráč vystřelil do světa.
- **Pod vodou** se nacházejí pouze určité druhy rud a rostlin, které se nevyskytují nikde jinde na povrchu. K těmto surovinám lze přistoupit pouze v pouštním světě, jelikož v něm není voda.

Ve světě se vyskytují dva druhy truhel podle jejich vzácnosti. Dřevěná bedna obsahuje základní suroviny, zatímco zlatá truhla uchovává vzácnější předměty.

Hlavní herní cyklus

Zjednodušený diagram 4.5 popisuje hlavní interakci mezi hráčem a světem. Definuje sadu po sobě jdoucích akcí, které je nutné vykonat k dosažení různých cílů. Nejvíce času stráví hráč na hlavní planetě, ovšem také se bude muset často vracet na základnu, kvůli omezenému inventáři. Každým přechodem mezi herními úrovněmi se implicitně uloží dosavadní postup hráče.



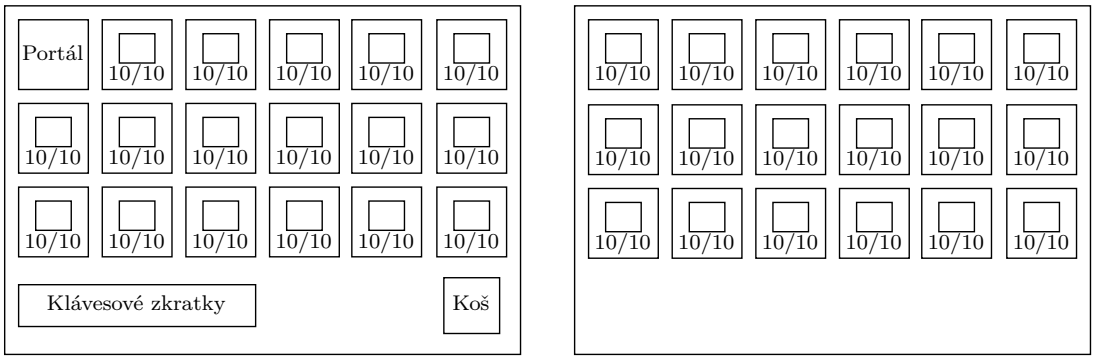
Obrázek 4.5: Oranžově jsou znázorněny akce, které hráč provádí na hlavní planetě. Modré akce jsou proveditelné pouze na základně.

4.2 Grafický design a rozhraní

Grafické zpracování této hry je stylizované do podoby *cartoonu* s nízkým polygonovým počtem. Tento design přináší elegantní vizuální estetiku, která si zakládá na jednoduchosti tvarů s výraznými barvami a kontrasty. Díky použití *lowpoly* grafiky jsou objekty a postavy ve hře zjednodušeny do základních geometrických tvarů, což vytváří charakteristický, čistý vzhled s minimálními detaily.

Inventář a truhly

Hráčův inventář plně disponuje několika základními funkcemi, které jsou zobrazeny v návrhu 4.6a. Předměty v inventáři se po stisknutí levého tlačítka myši přesunou po jednom do aktuálně otevřené bedny. Návrh rozhraní pro truhlu je zobrazen na obrázku 4.6b. Při držení tlačítka `shift` se přesunou všechny předměty z daného slotu. Inventář lze také kdykoliv přeskládat přetažením myši. Nechtěné předměty lze zahodit do světa či permanentně zničit. Pod ikonou předmětu se zobrazuje aktuální množství z celkového možného počtu v jednom slotu. Spotřební předměty a teleport lze použít pravým tlačítkem myši.



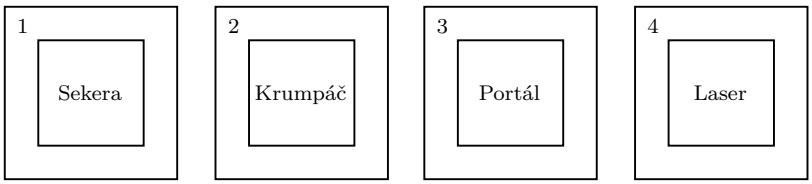
(a) Hráčův inventář.

(b) Úložný prostor truhly.

Obrázek 4.6: Návrh grafického rozhraní inventáře a truhly.

Výběr nástrojů

Hráč má na výběr ze čtyř nástrojů, které jsou v podobě tzv. „*quickslotů*“[1] jak je vidět na obrázku 4.7. Při výběru nástroje daný slot ztmavne a hráči se objeví v ruce. Každý slot je očíslován podle pořadí, které reprezentuje i přiřazenou klávesu. Opakovaným výběrem stejného slotu si hráč daný nástroj schová.



Obrázek 4.7: Návrh grafického rozhraní pro výběr nástrojů.

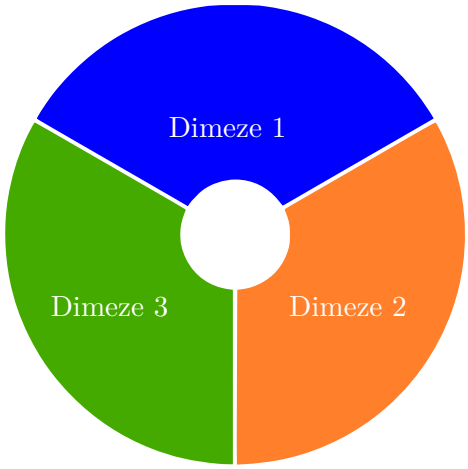
[1] Quickslot je vyznačen políčkem s ikonou v uživatelském rozhraní a umožní hráči rychle zvolit nástroj či předmět v daném slotu.

Využití nástrojů pro získávání surovin je vskutku intuitivní, ovšem to u ostatních nástrojů neplatí. Proto v pravém dolním rohu každý nástroj zobrazuje pro hráče nápovědu, která představuje jednu z možností své funkcionality a k ní přiřazenou klávesu jak zobrazeno na obrázku 4.8.

Obrázek 4.8: Ukázka nápovědy pro nástroj, který střílí portál.

Volba dimenzí

Pomocí tohoto rozhraní hráč může měnit dimenzi kolem sebe i celého světa. Podržením prostředního tlačítka zobrazí tři dostupné dimenze. Kurzorem si hráč vybere dimenzi a při puštění prostředního tlačítka myši zvolí příslušnou dimenzi. Pokud zvolí dvakrát tu stejnou dimenzi, tak se změní i podoba celého světa. Na obrázku 4.9 je návrh grafického rozhraní pro výběr dimenzí.

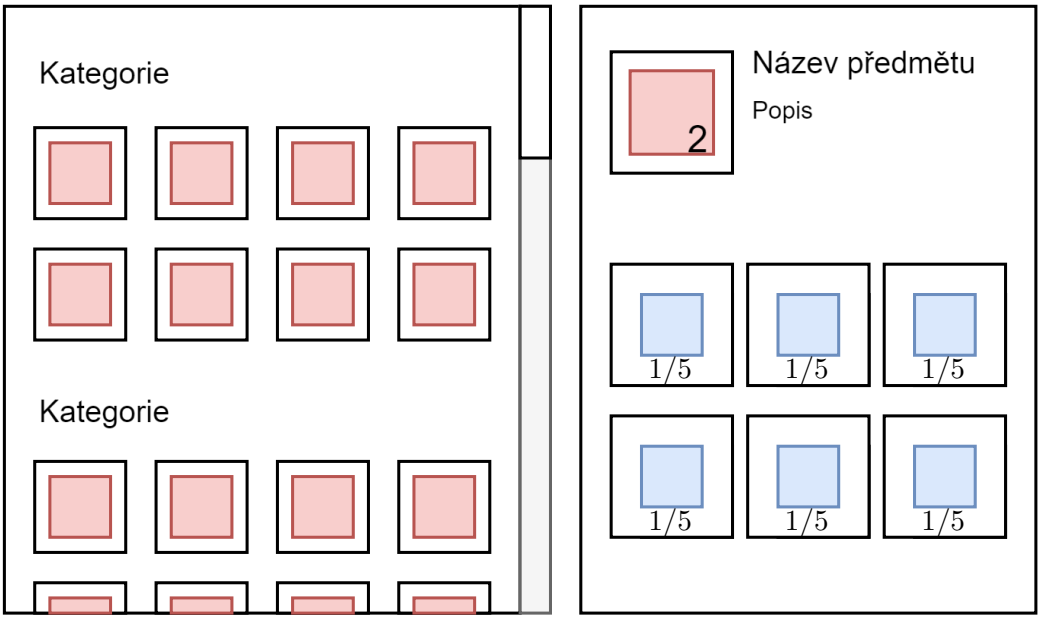


Obrázek 4.9: Návrh přepínače dimenzí.

Konstrukční stanice

Konstrukční stanice umožňuje hráči vytvářet různé druhy předmětů, které jsou rozděleny do specifických kategorií jak je zobrazeno na obrázku 4.10. Z rostlin lze vytvořit léčivé lektvary a z ostatních surovin se vyrábějí součástky do rozbitého modulu. Při kliknutí na zvolený recept se v dodatečném okně zobrazí název předmětu, popisek a potřebné ingredience pro jeho sestavení. V pravém dolním rohu u ikony předmětu se zobrazuje množství, které hráč získá při vytvoření. U každé ingredience se zobrazuje potřebné množství a aktuální množství, které má hráč v inventáři.

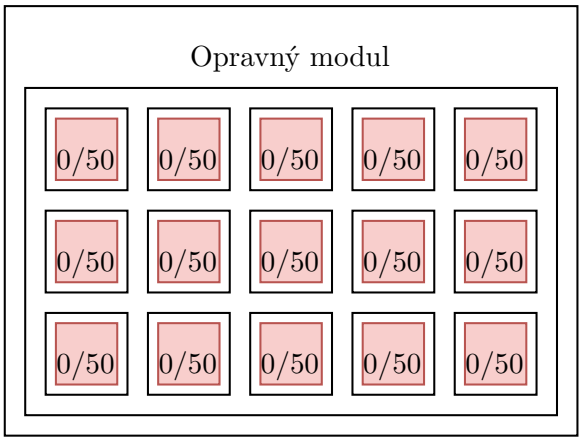
Pokud hráči některá ingredience chybí nebo nemá místo v inventáři, tak daný předmět nevytvoří. Toto upozornění je sděleno hráči ve formě zprávy v dodatečném okně.



Obrázek 4.10: Návrh rozhraní konstrukční stanice. Červeně jsou znázorněny ikony vytvářených předmětů. Modře jsou znázorněny ikony ingrediencí.

Porouchaný modul

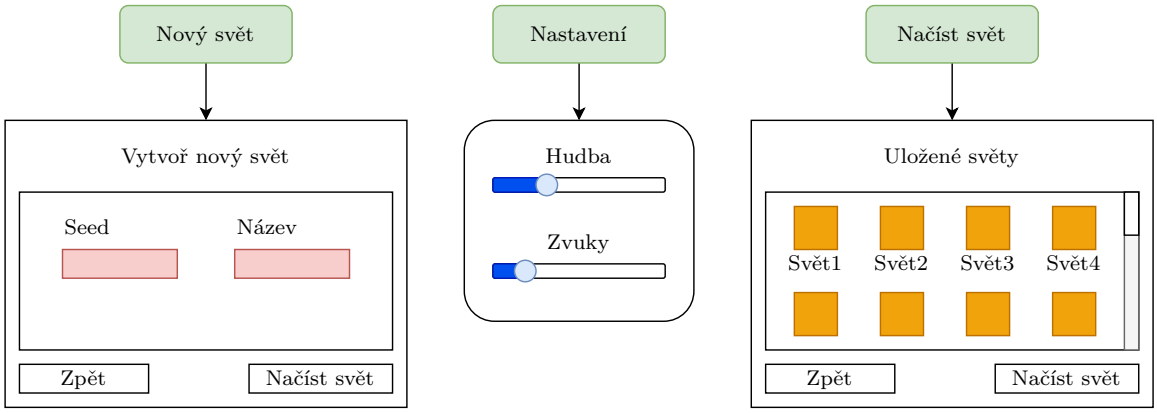
Tento grafický prvek se svým rozložením velmi podobá hráčskému inventáři. Zobrazuje aktuální a potřebné množství součástek k opravení lodi 4.11. Při získání všech potřebných surovin si může hráč opravit svoji loď a opustit měsíc, čímž dohraje hru.



Obrázek 4.11: Návrh rozhraní porouchaného modulu

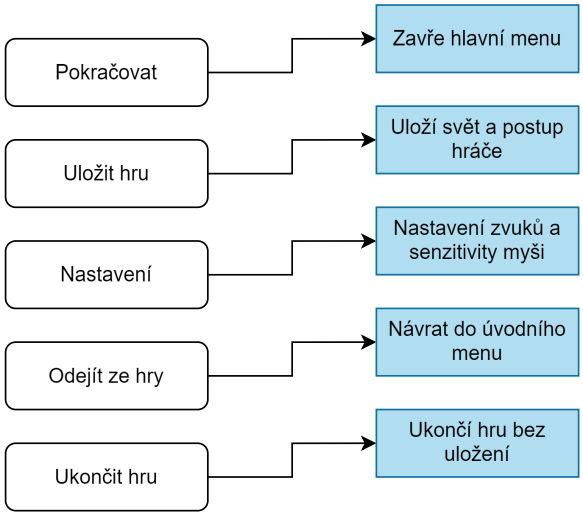
Úvodní a herní menu

V úvodní scéně si hráč vytváří světy a spravuje je. V nastavení je možné nastavit hlasitost zvuků. Návrh uživatelského rozhraní v této scéně je možné vidět na obrázku 4.12.



Obrázek 4.12: Návrh rozhraní pro úvodní scénu

Během hraní je hráči k dispozici rozhraní hlavního menu, jehož návrh včetně funkcionality je znázorněn na obrázku 4.13.



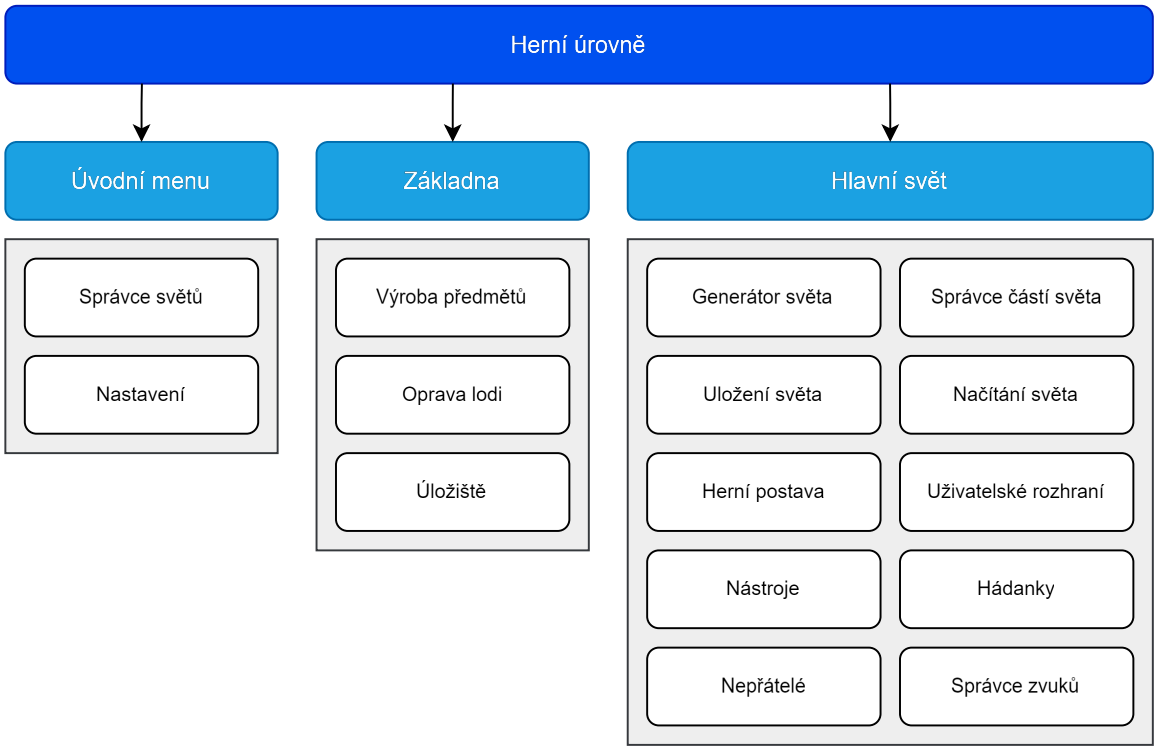
Obrázek 4.13: Návrh hlavního menu v hlavní scéně.

4.3 Rozvržení implementačních částí

Hra je rozdělena na tři scény, které hráči nabízí různé aspekty hry 4.14. V úvodním menu hráč vytváří nové světy nebo načítá již uložené hry. Navíc je umožněno vybrané světy i vymazat. V kategorii nastavení lze změnit hlasitost hudby či senzitivitu myši.

Na základně si hráč uschovává získané předměty z hlavního světa, které později využívá pro výrobu součástek k opravě lodi.

Hlavní svět víceméně tvoří podstatu celé hry. V první řadě je nutné vygenerovat svět a vyplnit ho herními objekty, které tvoří ekosystém světa. Tento svět je rozdělen do částí, které se dynamicky mění. Herní postup hráče a stav světa musí být průběžně ukládán. Z pohledu hratelnosti je nutné implementovat hráčův pohyb a herní nástroje, které využívá k interakci se světem. K této interakci patří především řešení hádanek a boj s nepřáteli. Každá herní akce musí být řádně ozvučena.



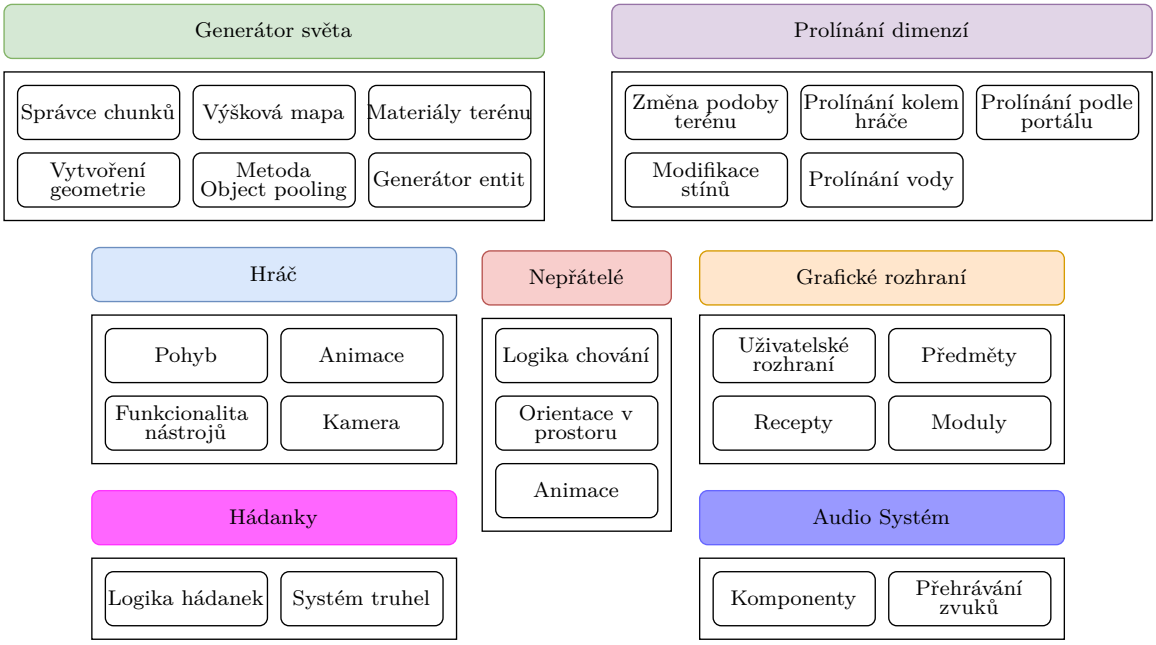
Obrázek 4.14: Návrh implementačních komponent.

Kapitola 5

Implementace

Tato kapitola se zaměřuje na praktickou realizaci konceptů a funkcionalit definovaných v předchozích částech této zprávy, jak je znázorněno na diagramu 5.1. Unity disponuje mnoha funkcionality, které jsou přímo zabudované do herního *enginu*. Kniha [17] vysvětluje jak tyto funkce využít v praxi a jejich principy. Dalším důležitým zdrojem této práce je kniha [15], která vysvětluje použití matematických metod a jejich využití v *shaderech* a fyzikálním světě.

Nejdříve je popsán základní životní cyklus herních objektů. Následně jsou popsány procesy a techniky použité k paralelnímu generování světa. Poté následuje sekce o prolínání dimenzí, která vysvětluje princip vykreslování herních objektů podle příslušných dimenzí světa. Následně je popsána implementace hráče, včetně jeho pohybu, animací a nástrojů, které umožňují interakci s herním prostředím. Další významnou sekcí této kapitoly je implementace grafického rozhraní a hádanek. V neposlední řadě je popsána realizace chování umělé inteligence a zvukového systému, který spravuje audio hry.



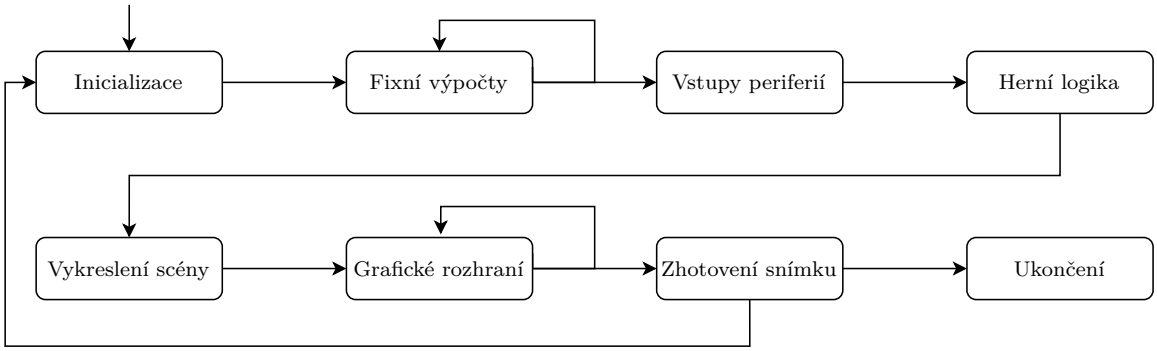
Obrázek 5.1: Znázornění jednotlivých částí implementace.

Životní cyklus Unity

Unity nabízí základní třídu, která umožňuje vytvářet skripty a ovládat chování herních objektů. Každý skript, který se nachází ve scéně na herním objektu musí dědit z rozhraní *MonoBehaviour*. Toto rozhraní umožní objektům začlenit se do hlavního životního cyklu aplikace a tím zprostředkuje mnoho užitečných funkcí. Životní cyklus se dělí do několika částí, které se vykonávají v určité posloupnosti jak je zobrazeno na obráku 5.2. V odstavcích níže jsou vysvětleny jednotlivé fáze ve vykonávaném pořadí:

- Aplikace v Unity vždy začínají inicializací všech skriptů. Každý skript při spuštění či změně aktivního stavu provádí určité operace. Inicializace probíhá ve dvou krocích.
- Fixní výpočty především využívá fyzikální engine a animační systém Unity. Provádí se ve fixních časových intervalech, které nejsou závislé na aktuálním počtu snímků za vteřinu. Existují dva typy fyzikálních kolizí. Přesněji tzv. „*trigger*“, který vyloženě neblokuje ostatní kolizní objekty, ale pouze registruje jejich průnik a volá příslušné funkce. Tento fyzikální prostor registruje prvotní protnutí s jakýmkoliv typem kolize. Následně vykonává akce každý fixní snímek, dokud kolize neopustí jeho prostor. Pod druhou variantou si lze představit normální kolizní geometrii, která fyzicky ovlivňuje další fyzikální objekty. Víceméně disponuje stejnou funkcionalitou jako fyzikální *trigger*, ovšem tyto kolize jsou statické, tudíž není možné, aby nastal jejich průnik. Proto si v kontextu fyzických kolizí lze volání předešlých funkcí interpretovat jako při dotyku, nikoliv při průniku.
- V této fázi hra registruje hráčovy vstupy z periferií a uloží si je pro další zpracování.
- Herní logika je jádrem celého životního cyklu. Tato fáze se vykonává pouze jednou za každý snímek na všech aktivních skriptech ve scéně. Řízení probíhá pouze na hlavním vlákně, proto jakýkoliv časově náročný výpočet zpomalí celou hru. Každá plynulá hra by měla generovat několik snímků za vteřinu, proto je naprosto zásadní tuto část kódu důkladně optimalizovat. Tento problém do určité míry řeší tzv. proces „*Coroutine*“, který asynchronně vyčkává na dokončení jiných akcí.

  Tato fáze je naprosto univerzální a dovoluje za běhu aplikace spravovat téměř všechny aspekty herního *enginu*. Definuje chování, pravidla, interakci mezi objekty a stará se o správné fungování aplikace. Zajišťuje správu herních systémů jako jsou například animace, zvuky, fyzika, grafické rozhraní a tak podobně.
- Tato práce bere v potaz pouze zabudovaný *render pipeline* poskytnutý od Unity, který byl již detailněji popsán v sekci 3.3. V této fázi komunikuje procesor s grafickou kartou. Mezi základní funkcionalitu patří například detekce vykreslených objektů podle kamery. Po vykreslení scény je signalizováno její dokončení, což lze využít například pro zahájení *post–processingu*.
- Grafické rozhraní provádí výpočty i vícekrát za jeden snímek. Pokud více skriptů přistupuje k různých elementům grafického rozhraní ve stejný snímek, tak se s každou změnou daný prvek vždy překreslí.
- Přímá dealokace instance volá funkci zničení na příslušném skriptu. O ukončení aplikace se stará funkce, která se nevolá pro každý objekt zvlášť, ale pouze jednou při ukončení aplikace.



Obrázek 5.2: Diagram životního cyklu aplikace v Unity.

5.1 Generování světa

Generování dynamického herního světa je náročný a složitý proces, který spotřebuje spoustu výpočetního času. Proto je nutné rozdělit tento proces na několik dalších paralelních procesů a ty korektně synchronizovat. Navíc tento svět musí být navržen tak, aby podporoval i ostatní herní mechaniky.

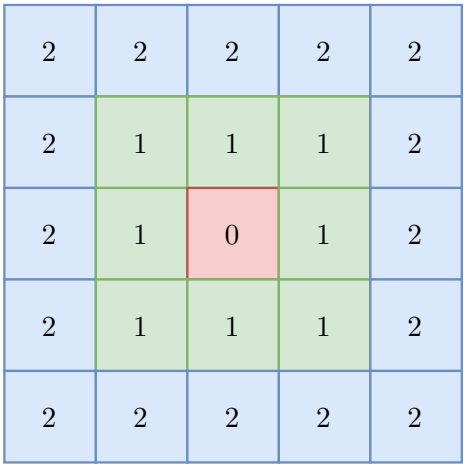
V této sekci jsou detailně popsány strategie pro správu chunků v závislosti na dohledu hráče, techniky paralelního generování geometrie terénu, textur, šumu a entit, stejně jako mechanismus pro efektivní ukládání stavu světa.

Správa generovaných chunků

Svět se generuje podle dohledu hráče jak je zobrazeno na obrázku 5.3. Manažer *ChunkQueueManager* neustále transformuje hráčovu pozici do souřadnic chunků a kontroluje, zda kolem hráče existuje nevyplněná část světa. Tato třída drží reference na následující dva listy:

- List *activeList*, který obsahuje aktivní vygenerované chunky.
- List *queueList*, který obsahuje nevygenerované chunky v dohledu hráče.

Pokud určitý chunk neexistuje v žádném listu, tak se přidá do listu *queueList*. Když se právě negeneruje žádný chunk, tak se spustí paralelní generování chunku na nultém indexu v *queueList*.



Obrázek 5.3: Ukázka rozložení světa při dohledu s hodnotou 2. Jak lze pozorovat na obrázku, tak při tomto nastavení dohledu, vznikne mřížka 25 chunků.

Paralelní generování

Unity disponuje tzv. „*Jobs system*“[1], který je navržen pro paralelní zpracovávání úloh. Efektivně využívá vícevláknové zpracování a zaručuje korektní synchronizaci. Navíc poskytuje kompilační nástroj *Burst Compiler*, který optimalizuje kompilaci kódu s využitím vektorizace a paralelizace instrukcí. Jeho efektivita závisí na konkrétní implementaci a typu prováděných operací v kódu, proto neumožňuje využití specifických funkcí a struktur. K využití tohoto systému nabízí Unity následující rozhraní:

- **IJob** reprezentuje samostatnou úlohu na jednom vlákně, která může být prováděna paralelně s jinými úlohami.
- **IJobParallelFor** reprezentuje více úloh pro zpracování dat ve formě pole. Tato metoda umožňuje efektivní paralelní zpracování iterací nad polem dat, kde je práce pro každý index přidělena jednotlivému vláknu.

Plánování úloh zprostředkovává struktura `JobHandle`, která řídí spouštění, zastavení a synchronizaci jednotlivých úloh. Navíc umožňuje vytvářet vzájemné závislosti mezi úlohami. Při implementaci se využívá speciální datový typ pole *NativeArray*, který slouží k efektivní manipulaci s daty na úrovni nízkoúrovňových paměťových operací. Na dokončení procesů se asynchronně čeká v *coroutine*, která korektně ukončí dokončený proces a zpřístupní data dalším procesům.

Paralelní generování šumu

Výšková mapa se generuje podle velikosti chunku a je uložena v jednorozměrném poli desetinných čísel, kde každý index reprezentuje výšku jednoho vrcholu geometrie. Pro výpočet výšky se využívá matematická funkce *Mathf.PerlinNoise*, která vrátí hodnotu šumu podle vstupních souřadnic. K těmto souřadnicím je připočten posun podle pozice chunku v prostoru. Podle počtu oktáv se v iteracích modifikuje výsledná hodnota šumu určitou frekvencí a amplitudou.

[1] Více o tomto systému na https://docs.unity3d.com/Manual/JobSystem.html.

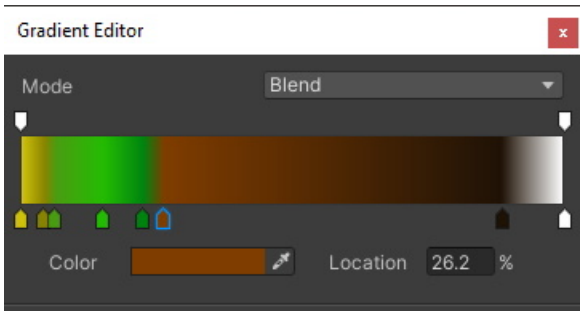
V tomto případě je velmi výhodné použít rozhraní **IJobParallelFor**, jelikož lze rozprostřít výpočet vrcholů mezi více vláken.

Paralelní generování textur

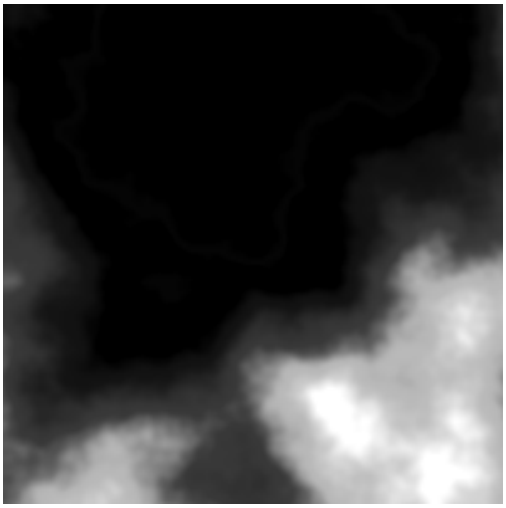
Pro každou část terénu je nutné vygenerovat tři textury, kde každá z nich je uložena v jednorozměrném poli struktur `Color32`. Tato struktura udává barevnost v podobě *RGBA*. Skládá ze 4 bajtů a obsahuje následující prvky, které tvoří 8 bitů (256 hodnot):

- **Byte R** – Udává hodnotu červené složky barvy.
- **Byte G** – Udává hodnotu zelené složky barvy.
- **Byte B** – Udává hodnotu modré složky barvy.
- **Byte A** – Udává průhlednost barvy.

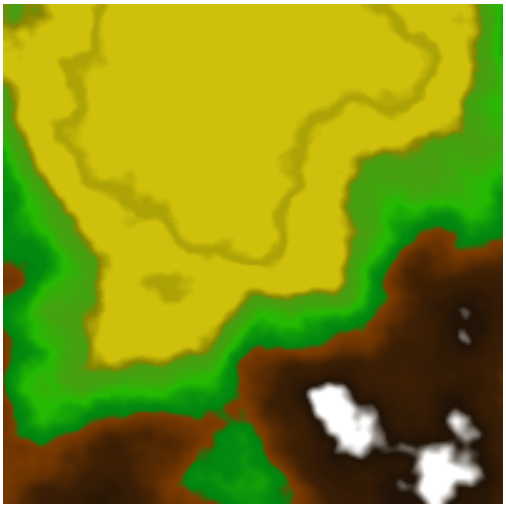
Tento paralelní proces je opět rozšířen o rozhraní **IJobParallelFor**, kde každý index reprezentuje zbarvení jednotlivých vrcholů. Zbarvení funguje na principu lineární interpolace normalizované výšky vrcholu mezi hodnoty 0–255 podle gradientu barev příslušné dimenze, který je zobrazen na obrázku 5.4. Na obrázcích 5.5a a 5.5b je znázorněn převod šumu na výslednou texturu materiálu.



Obrázek 5.4: Ukázka gradientu, který procentuálně reprezentuje zbarvení terénu podle jeho výšky.



(a) Ukázka modifikovaného Perlinova šumu

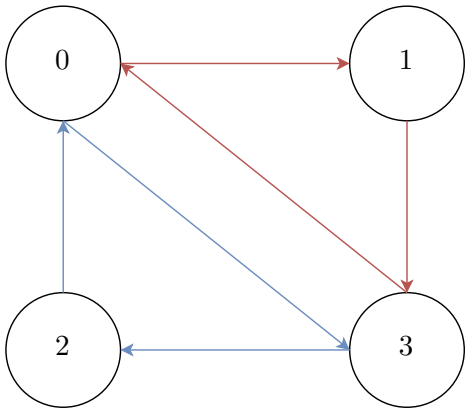


(b) Ukázka zbarveného šumu podle gradientu

Obrázek 5.5: Ukázka výsledné textury terénu

Paralelní generování geometrie

Tento proces je rozšířen rozhraním **IJob**, tudíž vždy zaměstnává pouze jedno vlákno. Úkolem je sestavit pole vrcholů v prostoru, spojit je trojúhelníky a vytvořit **UV mapu**. Generování probíhá ve dvou zanořených iteracích, které cyklují přes vrcholy výškové mapy. Geometrie se tvoří z levého horního rohu po řádcích s určitým odstupem, který je definován úrovní detailu *LOD*[2]. Vrcholy jsou rovnoměrně mapovány s normalizovaným odstupem. Pořadí vrcholů trojúhelníku definuje jeho orientaci. Proto jsou indexy trojúhelníků v tomto případě generovány ve směru hodinových ručiček [21]. Prvně se vytvoří trojúhelník z bodů $[0, 3, 2]$ a poté $[0, 1, 3]$, čímž vznikne polygon ve tvaru čtverce jak je zobrazeno na obrázku 5.6.



Obrázek 5.6: Ukázka sjednocení vrcholů pomoci trojúhelníků. Modrý trojúhelník vznikne jako první, poté následuje červený. Čísla v buňkách reprezentují indexy vrcholů.

Jelikož geometrie chunků na sebe navazuje, tak je nezbytné, aby normály vrcholů na okrajích geometrie měly mezi sebou plynulý přechod. Jedno z možných řešení zahrnuje

[2]LOD (level of detail) je zkratka pro úroveň detailu.

aproximaci pozic těchto vedlejších vrcholů příslušného chunku, podle kterých se upraví normály. Jednoduší řeší, které vyžívá tento generátor, spočívá ve vygenerování větší části térénu než je potřeba. Původně je šířka a výška chunku zvýšena o jednu světovou jednotku. Po vygenerování se krajní vrcholy odstraní, ovšem normály se zachovají. Po této úpravě normál je geometrie terénu ve finálním stavu.

Instanciace terénu

Po zhotovení výsledné geometrie se paralelně vytvoří fyzikální kolize terénu, aby se ušetřila práce na hlavním vláknu. Výpočet a přidělení kolize je v tomto bodě důležité pro rozmístění generování entit, které jsou závislé na fyzice.

Po dokončení se vytvoří herní objekt chunku a přidělí se mu skript *ChunkManager*, který spravuje daný chunk. Následně se k tomuto chunku přiřadí herní objekt vody. V tomto bodě se ručně dealokují nepotřebné proměnné, které vznikly při paralelním generování. Zachovají se pouze data výškové mapy pro generování entit.

Generování virtuálního navigačního prostoru

K vytvoření virtuálního navigačního prostoru byl využit experimentální balíček *NavMeshComponents*[3], který poskytuje přímo Unity. Po instanciaci terénu je možné vytvořit virtuální navigační prostor pro agenty podle jeho fyzikální geometrie. Bohužel v tomto kontextu nelze vytvořit pro každou část terénu vlastní navigační prostor, jelikož by agenti nemohli přecházet mezi chunky. Proto s každým nově vygenerovaným chunkem je nutné přidat jeho fyzikální kolizi do *NavMeshBuildSource*, což je struktura, která obsahuje geometrickou předlohu pro sestavení jednoho virtuálního prostoru. Tato struktura podle zahashované hodnoty zjistí, které části předlohy virtuálního prostoru jsou nové či změněné a podle toho je asynchronně sestaví. Díky této funkcionalitě je možné přidávat či odebírat menší části světa a během běhu aplikace plynule modifikovat virtuální prostor.

Paralelní generování entit

Tento proces je rozšířen rozhraním **IJob**. Správu generování entit řídí skript *ChunkPropSpawner*. Ten obsahuje následující listy herních objektů:

- První list obsahuje objekty vegetace a dalších nerostných surovin.
- Druhý list obsahuje objekty, které se vyskytují pod vodou.
- Třetí list obsahuje objekty, které tvoří hádanky.

Tyto objekty jsou rozděleny do více listů, kvůli přehlednosti. Při inicializaci se tyto listy spojí do jednoho. Úkolem je vygenerovat typy entit a ty rozmístit podle příslušných pravidel. Během generování entit se iteruje přeš výškovou mapu s určitým rozestupem. V každé iteraci se podle výšky terénu s určitou šancí náhodně zvolí index v listu a ten se uloží do pole vygenerovaných entit. Pro každý tento index je v poli pozic uložena výška terénu. Na konci tento procesu vrátí pole entit a jejich pozic.

[3]Tento balíček je dostupný na https://github.com/Unity-Technologies/NavMeshComponents.

Instanciace entit

Vytváření herních objektů spravuje skript *ChunkPropSpawner*, který nejen generuje entity, ale také přemísťuje či vytváří nové instance. Každý herní objekt musí být při vytvoření alokován, což je ve velkém množství objektů vcelku časově náročná operace. Proto existuje tzv. „*Object pooling*“ metoda, která řeší tento problém. Spočívá v tom, že při inicializaci světa vytvoří nové instance objektů během načítání, které poté recykluje. Takže před vytvořením nové instance se zkontroluje zda v listu *pooledObjects* již existuje připravená instance, kterou lze využít. Každý zničený chunk přidá do tohoto listu všechny jeho instance herních objektů. Pokud nastane situace, kdy není připravená žádná instance daného typu, tak se každý herní snímek alokuje pouze pár nových. Důležité je rozprostřít alokování do co nejvíce snímků a při tom zachovat plynulost hry.

Ukládání a načítání světa

Správu dat jednotlivých chunků zajišťuje skript *SaveChunk*. Data jsou uložena ve struktuře `ChunkSave`, která obsahuje souřadnici chunku a list jeho entit. Při inicializaci se vytvoří následující dva listy:

- List *saveData* obsahuje načtené data ze souboru `ChunkSave`.
- List *activeChunks* obsahuje data právě aktivních vygenerovaných chunků.

Při běhu aplikace se modifikují data v listu *activeChunks*. V případě, kdy je nutné uložit data opět do souboru, tak se přepíší data v listu *saveData* novými daty z *activeChunks*. Pokud ukládaný chunk neexistuje v uloženém souboru, tak se jednoduše přidá do listu. Tato data jsou rozděleny do dvou listů, z optimalizačních důvodů. Pokud hráč v určitém chunku vytěží jeden strom a poté odejde, tak je nesmyslné prohledávat a modifikovat celou databázi chunků. Proto se změny provádí v tomto krátkém listu *activeChunks*. Uložení dat se provádí ve dvou situacích. První případ nastává, když hráč explicitně uloží celý svět pomocí tlačítka *Save* v hlavním menu. Druhý případ nastane, když se chunk vzdálí mimo hráčův dosah, což způsobí uložení jeho stavu a následné zničení příslušné instance. Ukládání a načítání ostatních dat řídí skript *SaveManager*, který operuje s následujícími daty:

- V souboru **WorldData** jsou uloženy dimenze portálů, pozice portálu, pozice hráče a jeho životy. Především obsahuje seed světa.
- Soubor **SafeZone** obsahuje data o postupu hráče na základně. Přesněji se jedná o obsah úložného prostoru a stav porouchaného modulu.
- V souboru **PlayerInventory** je uložen hráčův obsah inventáře.

5.2 Prolínání dimenzí

Prostředí světa se mění v závislosti na umístění a dimenzí portálů. Tyto portály ovlivňují vzhled terénu, herních objektů a oblohy. Nejprve je v této sekci popsán proces vykreslování terénu. Následně je rozebrán proces, který nastane před vykreslením objektů. Poté je detailněji rozebrán vliv portálů na vykreslování herních objektů a oblohy.

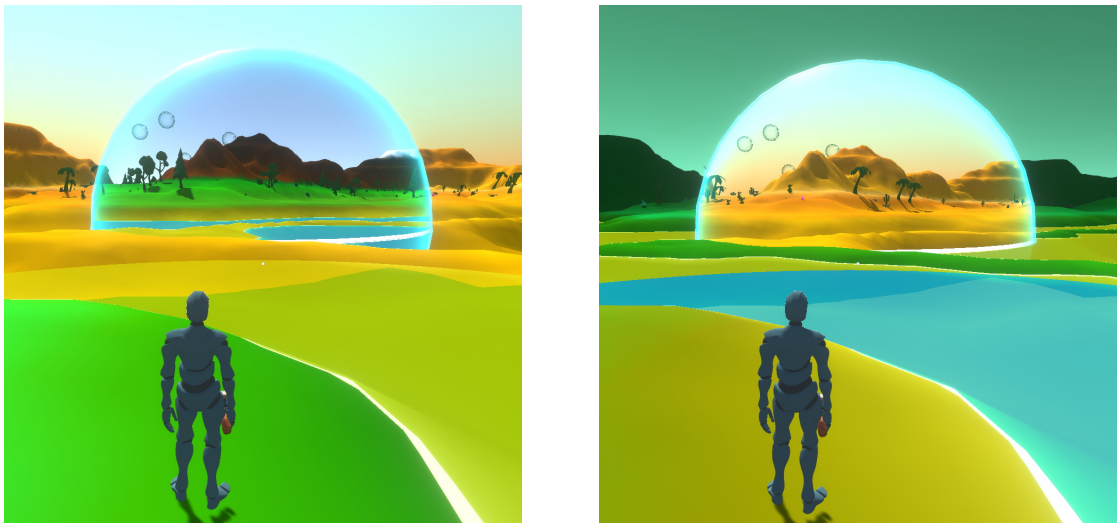
Změna podoby terénu

Vykreslování terénu spravuje shader *GroundMaskShader*, který mění jeho vzhled podle herních portálů. Pro každou dimenzi obsahuje jednu texturu. Vykreslování probíhá v následujících fázích:

- První fáze se zaměřuje na úpravu terénu v okolí hráče. Pro každý vrchol terénu se vypočítá jeho vzdálenost od pozice hráče. Pokud je tato vzdálenost menší než poloměr portálu kolem hráče, pixely jsou obarveny podle příslušné dimenze.
- Ve druhé fázi se kontroluje stencil maska s referenční hodnotou pixelu. Pokud se tato hodnota shoduje s hodnotou portálu ve světě, příslušné pixely jsou vykresleny podle dimenze tohoto portálu. Tato maska ovšem ovlivní i pixely před portálem. Tento problém lze vyřešit výpočtem skalárního součinu následujících vektorů:
  - První vektor udává směr od vrcholu terénu ke kameře.
  - Druhý vektor udává směr od vrcholu terénu ke středu portálu.

  Shader pak na základě produktu skalárního součinu těchto vektorů rozhoduje, zda přiřadit daným pixelům příslušnou texturu či ne. Pixely, které se nacházejí v rádiu portálu a jsou umístěny mezi hráčem a středem portálu, zůstávají zbarveny podle dimenze portálu.

Pixely terénu, které nebyly ovlivněny herními portály, jsou vykresleny podle dimenze světa. Tento shader nepoužívá dynamické osvětlení scény, avšak simuluje vlastní osvětlení pomocí vestavěné funkce *ShadeSH9*, která vytváří dojem osvětlení na základě normál vrcholů. Kromě toho je při prolínání s toxickou dimenzí navíc vykreslována mlha.



(a) Ukázka lesního portálu v pouštním světě

(b) Ukázka pouštního portálu v toxickém světě

Obrázek 5.7: Prolínání terénu podle dimenze a pozice portálu.

Nastavení objektů před vykreslením

Prolínání objektů spravuje skript *PropDimensionManager*, který každému objektu přiřadí číslo sektoru, ve kterém je viditelný. Vykreslení stínů a objektů řídí shader *DimensionClip*. Sektory jsou očíslovány následovně:

- **Sektor -1** značí, že se tento objekt nenachází v žádné aktivní dimenzi.
- **Sektor 0** reprezentuje objekty, které jsou viditelné mimo herní portály.
- **Sektor 1** značí, že je tento objekt viditelný ve všech částech světa.
- **Sektor 2** reprezentuje prostor pro objekty kolem hráče.
- **Sektor 3** reprezentuje objekty, které jsou viditelné skrze portál.

Pokud se objekt nachází v sektoru 3, tak se jeho referenční hodnota nastaví podle *stencilové* masky portálu. Ostatní sektory pracují s výchozí referenční hodnotou *stencil bufferu*. Většina těchto objektů sdílí stejný materiál, který využívá paletu barev v podobě textury. Proto je možné využít optimalizační metodu *batching*, která vykresluje v jeden okamžik objekty se stejným materiálem.

Podle těchto sektorů se zaktivují či deaktivují fyzikální kolize herních objektů.

Vykreslení objektů kolem hráče

Kolem hráče jsou vykresleny pouze objekty, které spadají do sektoru číslo 2. Ke každému vrcholu se vypočítá vzdálenost od středu hráče. Tato vzdálenost se poté porovná s poloměrem portálu a podle sektoru se rozhodne, zda se daný objekt vykreslí. Při ořezávání jsou navíc části objektů zbarveny zářivou barvou typu *HDR*[4] podle hráčovi dimenze.

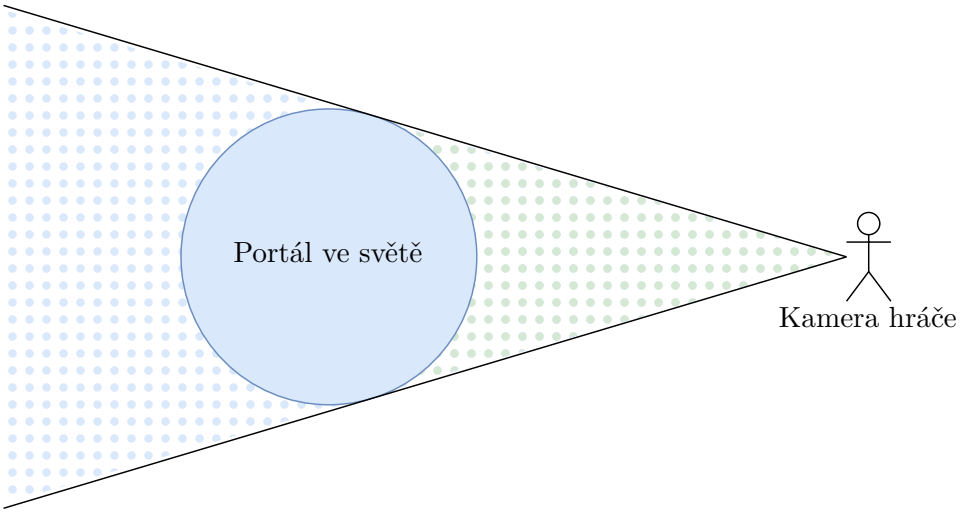
Obrázek 5.8: Ukázka vykreslených objektů kolem hráče.

Vykreslení objektů podle portálu ve světě

Pro tento portál je nastavena *stencilová* maska, která binárně reprezentuje číslo 3. Portál se vykresluje před všemi herními objekty, aby s předstihem označil pixely na obrazovce,

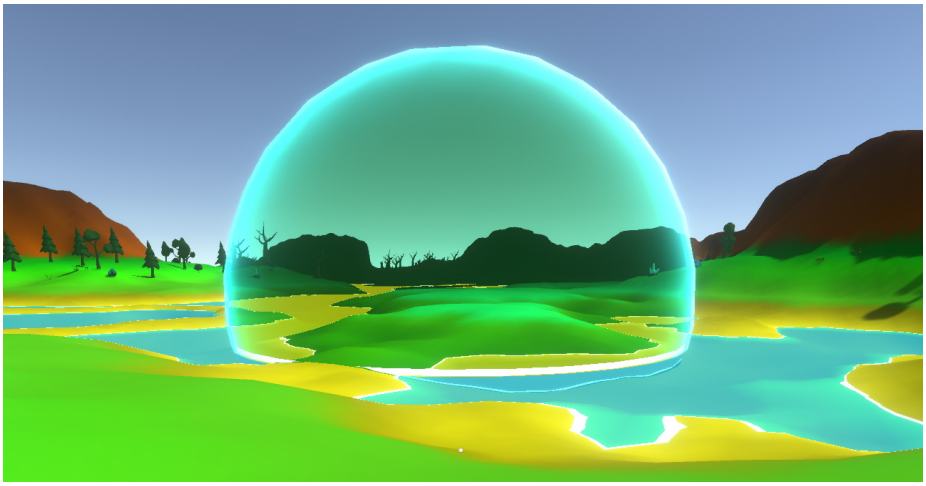
[4]Zkratka HDR (High dynamic range) definuje barvy na rozšířené škále, na které lze nastavit svítivost barvy.

které reprezentují prostor portálu. V této fázi nastává zásadní problém a omezení *stencil bufferu*, jelikož nedrží informace o hloubce ve scéně. Proto se kvůli označeným pixelům vykreslují objekty nejen za portálem, ale i před ním. Tento problém je vyřešen ořezáváním nežádoucích objektů na základě produktu skalárního součinu vektorů, jak již bylo zmíněno v sekci 5.2. Pro lepší představu je záměr vykreslování zobrazen na obrázku 5.9.



Obrázek 5.9: Zelenými tečkami je zvýrazněn prostor, ve kterém se nesmí vykreslit objekty, které promítá portál. Naopak v modré části se vykreslují pouze objekty dané dimenze.

K změně oblohy za portálem byl modifikován zdrojový kód *skybox shaderu*, který poskytuje Unity. Tento shader je obohacen o *stencilový* test, podle kterého se obloha zbarvuje s určitým odstínem, expozicí a dalšími parametry.



Obrázek 5.10: Ukázka vykresleného portálu, který mění podobu světa.

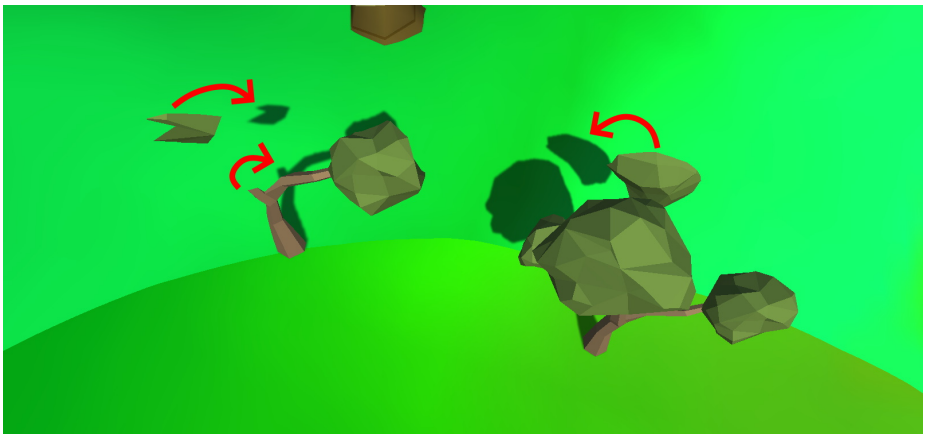
Vykreslení stínů

V první fázi vykreslovaní se vykreslí geometrie objektů včetně jejich materiálů. Po vykreslení všech objektů nastane druhá fáze, ve které se vypočítají stíny, které vrhají herní objekty. V poslední fázi se stíny vykreslí na základě informací z předchozí fáze.

Stencil buffer je v podstatě 2D efekt, který pracuje s fragmenty pixelů. Proto informace o pouhých pixelech není možné využít při výpočtu osvětlení. Z tohoto důvodu je nutné vypočítat vlastní stíny a ty následně modifikovat podle představ.

Pro modifikaci stínů je využit tzv. „*ShadowCaster*“, který vykresluje informace o stínech do stínové či hloubkové mapy. Pokud se v této fázi odstraní specifické vrcholy geometrie objektů, tak se pro ně v poslední fázi žádný stín nevykreslí. Tento princip se využívá k skrytí stínů neviditelných objektů. V případě, kdy je objekt ořezán jinou dimenzí, tak se vykreslí stíny pouze pro neořezanou část objektu jak je znázorněno na obrázku 5.11.

Při vykreslování objektů lze jednoduše označit pixely na obrazovce a ty vykreslit, ovšem stíny se nepočítají podle viditelných pixelů, ale podle geometrie objektu. Proto je nutné zjistit, které vrcholy vidí kamera a ty zachovat. Tuto situaci značně ztěžuje herní portál, který lze vytvořit kdekoliv ve světě. Viditelnost herních objektů skrze portál lze zjistit pomocí průsečíku přímky s kulovitým portálem. Tato přímka značí vektor od hráče k vrcholu geometrie. Pokud přímka proniká prostorem kulovitého portálu, tak je daná část objektu skrze něho vidět. V opačném případě je vrchol z stínových dat odstraněn.

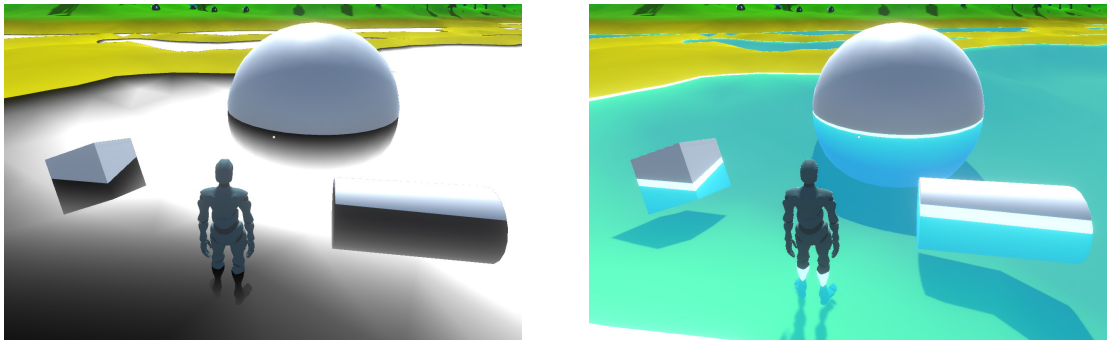


Obrázek 5.11: Červené šipky znázorňují stíny ořezaných objektů.

Vykreslení vody

Materiál vody spravuje shader *StylizedWater*, který simuluje vlny a zbarvuje vodu podle její hloubky. Vlny vznikají rozpohybováním geometrie podle goniometrických funkcí, které mění v čase výšku vrcholu na ose Y. Mělkost vody určuje její zbarvení. Tohoto efektu je dosaženo za pomocí hloubkové textury, která zbarvuje pixely podle vzdálenosti mezi objekty ve stupni šedi. Úroveň mělkosti v tomto případě udává černá barva, jak je reprezentováno na obrázku 5.12a. Tento shader využívá dvě barvy, které udávají zbarvení mělké a hluboké vody. Podle stupňů šedi se interpoluje mezi těmito barvami. Pokud je v určitém místě dostatečně černá barva, tak se na této pozici vykreslí pěna, jak je vidno na obrázku 5.12b. K pochopení těchto technik vykreslovaní, tato práce čerpala z článku, který napsal Erik Roystan[5].

[5]Návod k vykreslení vody podle hloubkové textury je dostupný na https://roystan.net/articles/toon-water/.

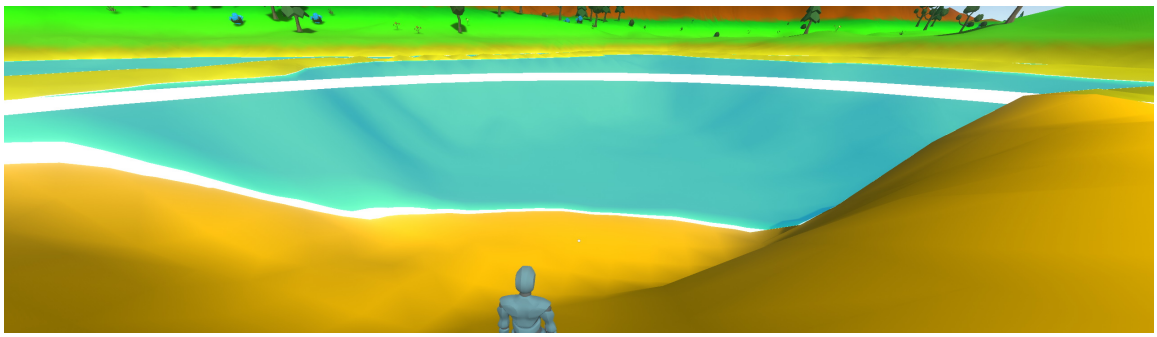


(a) Ukázka hloubkové textury na hladině vody

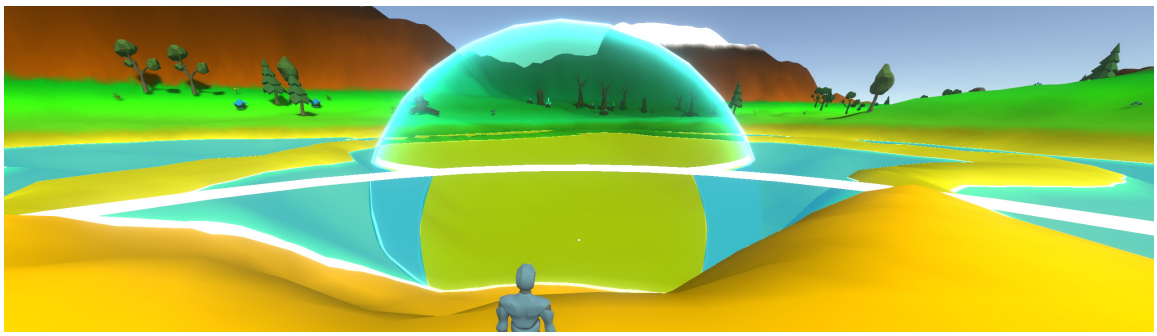
(b) Výsledné zbarvení hladiny vody

Obrázek 5.12: Ukázka zbarvení vody podle hloubky a vykreslení pěny kolem objektů.

Voda mění podobu hladiny podle příslušné dimenze kolem hráče. Proto se při vykreslování dynamicky volí parametry, které reprezentují zbarvení v jednotlivých dimenzích. Kolem hráče je herní objekt ve tvaru jeho portálu, který vykresluje vodu pod hladinou jak ukazuje obrázek 5.13a. Materiál tohoto objektu funguje na stejném principu jako vykreslování hladiny, ovšem ořezává všechny pixely, které jsou nad hladinou vody. Jelikož se hráč vždy nachází uvnitř tohoto objektu, tak jsou při vykreslování převráceny jeho normály směrem k němu. Při prolínání dimenzí má portál kolem hráče vždy prioritu. Na obrázku 5.13b hráčův portál ořezává a vykresluje vodu pod hladinou podle portálu ve světě.



(a) Ořezávání vody kolem hráče.

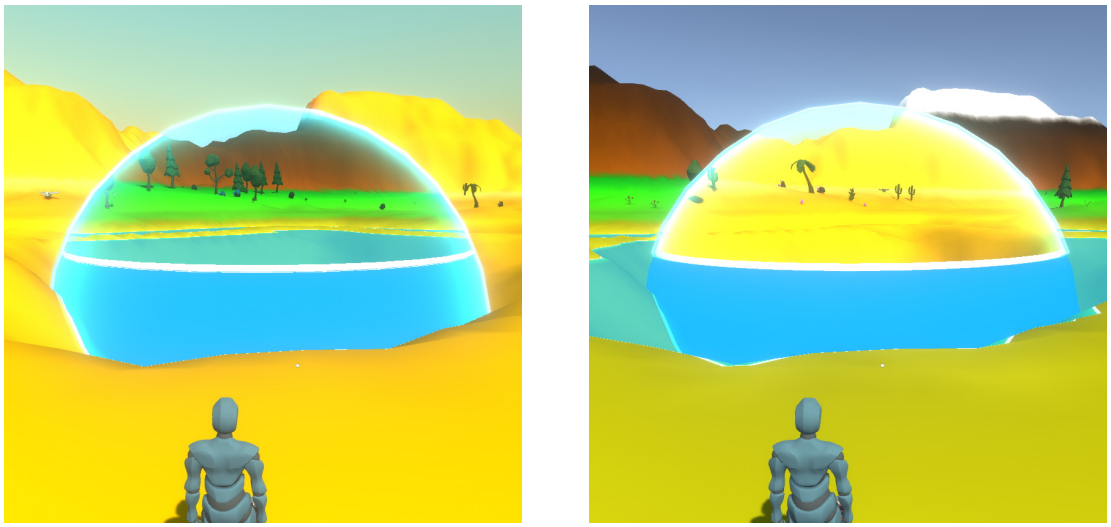


(b) Prolínání vody mezi hráčem a portálem.

Obrázek 5.13: Ukázka ořezávání a prolínání vody.

Portál je také schopný ořezávat vodu či měnit její podobu. V tomto případě je opět využit *stencilový* test, který porovnává referenční hodnotu pixelu vody s maskou portálu.

Pokud se tyto hodnoty rovnají, tak se tyto pixely vykreslí v podobě příslušné dimenze. Kolem portálu se také nachází pomocný objekt, který představuje vodu pod hladinou. V tomto případě jsou normály tohoto objektu natočeny směr ven.



(a) Voda v portálu.

(b) Voda mimo portál.

Obrázek 5.14: Ukázka prolínání vody.

5.3 Realizace hráče

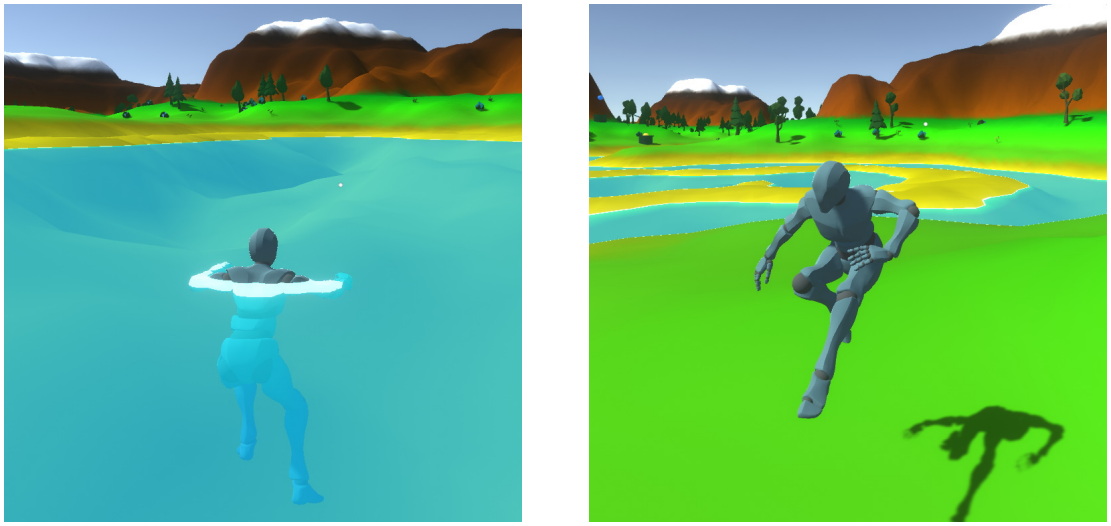
Tato sekce se zaměřuje na implementaci hráče v herním prostředí Unity. Klíčovým prvkem této implementace je hráčův animační systém, který zajišťuje plynulý pohyb a interakci hráče se světem. Prvně jsou popsány různé techniky implementace animačního systému, které poskytuje přímo Unity. Kromě samotného animačního systému jsou také rozebrány jednotlivé nástroje, které ovlivňují chování herních objektů.

Pohyb

K rozpohybování hráče je využit tzv. „*Character Controller*“, což je komponenta v Unity, která zjednodušuje pohyb postav. Tato komponenta simuluje fyzikální pohyb ve všech směrech v rámci herního světa, ovšem nebere v potaz gravitační sílu. Interaguje s okolními kolizními objekty a zajišťuje jejich interakci podle nastavených parametrů. Další užitečnou vlastností této komponenty je schopnost pohybu na nerovném povrchu. Parametr *slopeLimit* určuje maximální úhel sklonu, pod kterým se může postava pohybovat, což se například využívá při chůzi do schodů.

Character controller navíc poskytuje informaci, zda se postava právě nachází ve vzduchu či je nohama na zemi. Pokud je hráč uzemněn a stiskne klávesu skoku, tak se k jeho vektoru pohybu přičte gravitační zrychlení směrem nahoru, které každý fixní snímek postupně klesá na původní hodnotu.

Pokud se hráč nachází pod vodou, tak automaticky vyplave na povrch. Když hráč plave na hladině, tak na něj nepůsobí gravitace.



(a) Plavání ve vodě.

(b) Hráč ve skoku

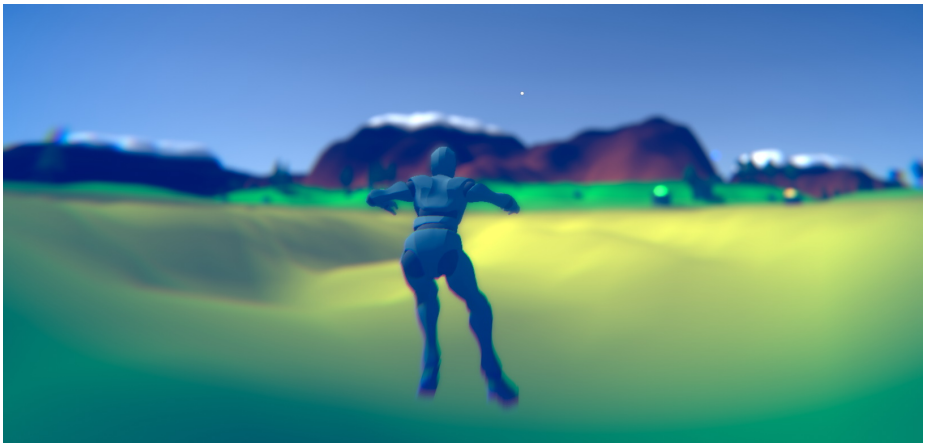
Obrázek 5.15: Ukázka akcí pohybu

Ovládání kamery

Dalším klíčovým aspektem implementace hráče je ovládání a nastavení kamery. Unity nabízí externí balíček nástrojů zvaný *Cinemachine*, který rozšiřuje funkcionalitu základní kamery. K využití tohoto balíčku je nutné přidat na hlavní kameru komponentu *CinemachineBrain*, která využívá nastavení z herního objektu *CM virtual camera*.

Jelikož se hra hraje z třetího pohledu, tak hlavní herní kamera orbituje kolem hráče. Kamera neustále sleduje pohyb hráče a rotuje takovým způsobem vůči jeho pozici, tak aby na něj vždy mířila. Přirozeně s pohybem hráče se mění i pozice kamery. Na kameře je nastaven posun, který simuluje pohled přes rameno. Vstupy k pohybu hráče jsou transformovány podle směru kamery. Pokud je kamera přesně čelem k hráči a ten zmáčkne klávesu pro pohyb vpřed, tak se otočí o 180 stupňů a pokračuje v pohybu podle směru kamery.

Pokud se kamera ocitne pod hladinou vody, tak se na ní aplikuje dodatečný efekt, jak je zobrazeno na obrázku 5.16.

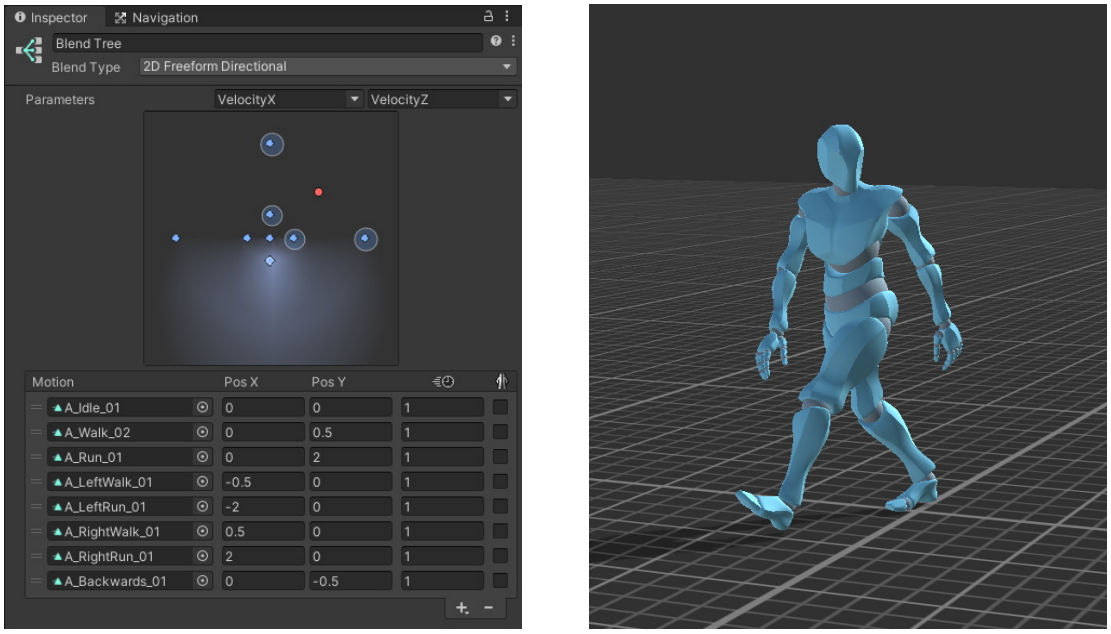


Obrázek 5.16: Filtr na kameře, který simuluje vizuální efekt pod vodou.

Animace

Postava hráče využívá základní animační automat 5.18 a tzv. „Animation blend tree“[6], který umožňuje komplexnější a plynulejší přechody mezi animacemi. Na obrázku 5.17a je vidět souřadnicový systém plný modrých teček (animací). Tyto animace jsou rozmístěny mezi sebou v určitých vzdálenostech a reprezentují pohyb hráče do čtyř směrů. Vzdálenější tečky od středu tvoří animace běhu.

Červená tečka se skládá z parametrů *VelocityX* a *VelocityZ*, které udávají směr a intenzitu pohybu. Podle pozice tohoto bodu, se mezi sebou procentuálně prolínají animace, čímž vzniknou animace i pro pohyb do 8 a více směrů. Navíc postupné zrychlení hráče tvoří plynulý přechod mezi chůzí a sprintem.

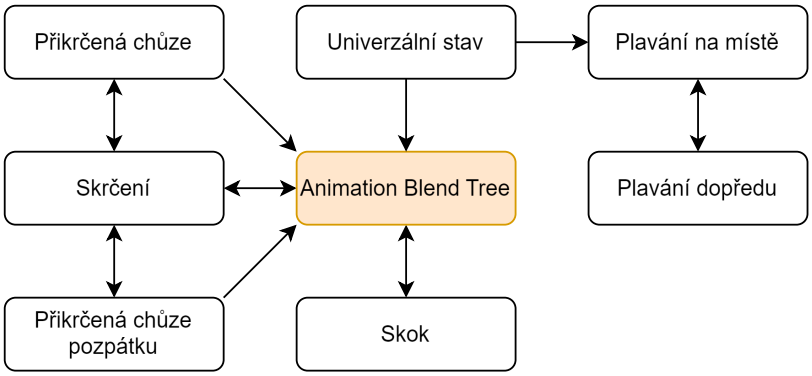


(a) Ukázka rozhraní *Blend Tree*

(b) Výsledek smíšených animací hráče

Obrázek 5.17: Vlevo na obrázku červená tečka reprezentuje intenzitu a směr pohybu hráče. Každá modrá tečka reprezentuje jednu animaci. Parametry *VelocityX* a *VelocityZ* se na obrázku rovnají jedné.

[6] Více o tomto nástroji na https://docs.unity3d.com/Manual/class-BlendTree.html.



Obrázek 5.18: Animační automat hráče. Oranžově je vyznačen počáteční stav.

Těžba a útok

Animace sekání se skládá z jednotlivých snímků, ke kterým lze přidat volání jedné funkce příslušného skriptu. Při rozmachování zbraní se čeká na určitý animační snímek, kdy hráč vyzdvihne nástroj nad hlavu. V ten moment se zaktivuje kolizní kvádr, který pokrývá celý objem nástroje. Následujících pár snímků se ukládají do listu všechny herní objekty, které s daným nástrojem kolidují. Jakmile hráč dosekl a vrací ruku do původní pozice, tak se již zmíněná kolize opět vypne a vyhodnotí se všechny zasažené objekty. Začátek animace také hráči znemožní opět seknout, dokud se nepřehraje poslední animační snímek.

Krumpáč a sekera slouží k těžení surovin, ovšem každý nástroj lze použít pouze na určitý druh suroviny. Proto každý vytěžitelný objekt disponuje skriptem *Harvestable*, který drží následující tři informace:

- **Typ nástroje**, kterým lze surovinu vytěžit.
- **Životnost** před rozbitím.
- **List předmětů** s množstvím, které lze získat z dané suroviny.

Pokud jedním ze zasažených objektů je nepřátel, tak ten utrpí poškození a přehraje se jeho animace zásahu.

Laser

Laser mění svůj zdroj energie podle aktuální dimenze kolem hráče. Proto se jeho energie dělí na tři typy. Podle typu zbarví svůj paprsek a umožní ovlivňovat konkrétní objekty, které reagují na příslušný typ energie.

Interakci mezi energií laseru a objekty zajistí skript *LaserHitManager*. Ten definuje chování zasažených objektů a akce, které nastanou při kontaktu s laserem.

Paprsek vykresluje komponenta *LineRenderer*, která vytvoří geometrii mezi pozicemi v prostoru. Tyto pozice uchovává v poli trojrozměrných vektorů. Pochopitelně k vykreslení potřebuje minimálně dva body, ovšem umožňuje vytvořit i komplexnější křivku s více body. Výslednou geometrii lze modifikovat pomocí grafů či míry zaoblení mezi rohy.

Střelba portálu

Při výstřelu portálu se vyšle z kamery tzv. „*Raycast*“[7]. Pokud úspěšně zasáhne terén, tak v tom bodě vytvoří instanci portálu. Před výstřelem probíhá kontrola počtu portálů, jelikož ve scéně může existovat právě jeden. Kdykoliv je možné změnit typ dimenze klávesou `X` nebo portál pravým tlačítkem myši ze světa odebrat. Každá tato akce invokuje delegát, který oznámí všem závislým objektům změnu stavu. Aktuální stav dimenze portálu drží statická třída *WorldState*.

5.4 Funkcionalita grafického rozhraní

Tato sekce se zaměřuje na implementaci klíčových prvků grafického uživatelského rozhraní. Hlavním cílem této implementace je vytvoření intuitivního a efektivního prostředí pro hráče, které usnadní navigaci a interakci s herním světem.

Poté nasleduje implementace inventáře, který slouží k uchovávání a správě předmětů, které hráč získává během hraní. Inventář je důležitou součástí této hry, proto je rozepsán detailněji. Dalším klíčovým prvkem je uživatelské rozhraní, které popisuje stav hráče, upravuje hratelnost, ukládá svět a tak podobně. Nakonec je představena implementace interaktivních objektů, jako je konstrukční stanice a opravný modul. Tyto objekty umožňují hráči vytvářet a zpracovávat předměty.

Manažer životů

O správu životů hráče se stará skript *PlayerHealthManager*, který drží aktuální hodnotu zdraví. Pokud hráč utrpí poškození nebo vypije léčivý lektvar, tak změní aktuální hodnotu zdraví, kterou promítne do uživatelského rozhraní. Navíc zaručuje, že hráčovy životy nepřesáhnou maximální hranici životů.

Obrázek 5.19: Ukázka rozhraní životů

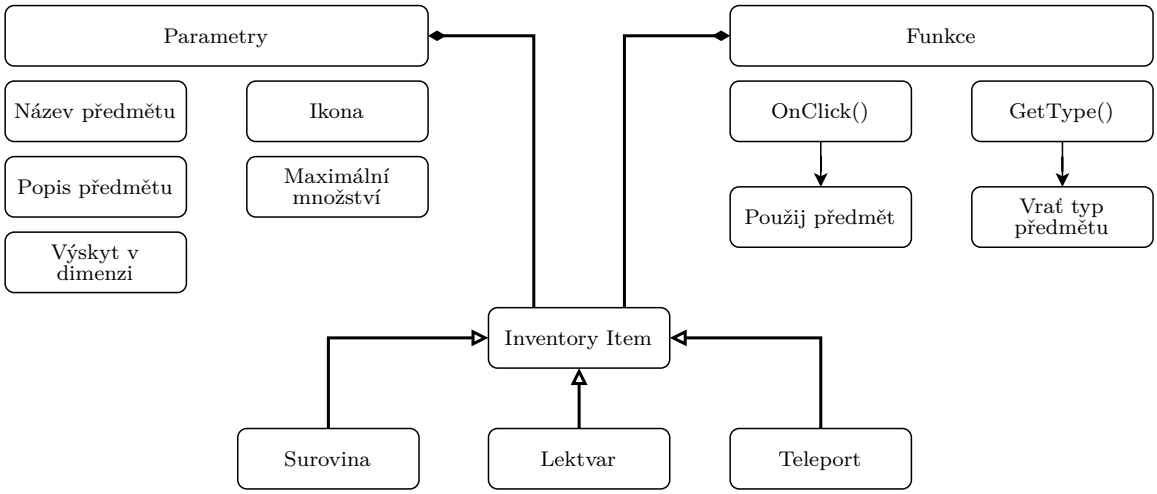
Smrt hráče

Pokud hráči klesne hodnota zdraví na nulu, tak zemře a přijde o všechny jeho předměty, které nese v inventáři. Tato skutečnost je hráči sdělena v podobě okna, které překrývá celou obrazovku. V tomto rozhraní se nachází tlačítko pro znovuzrození, které uloží dosavadní postup ve světě a hráče teleportuje zpět na základnu.

Předměty

Předmětů je ve hře spousta, proto pro ně byla vytvořena univerzální struktura `InventoryItem`, která drží jejich data a rozděluje je do typů podle funkcionality. Každý předmět dědí z této abstraktní třídy a definuje si vlastní chování. Pro přehlednost jsou tyto předměty rozšířeny o rozhraní *ScriptableObject*, které zjednodušuje vytváření a načítání herních objektů. Charakteristika této třídy a typy předmětů jsou znázorněny na diagramu 5.20.

[7] Virtuální paprsek, který detekuje kolize s herními objekty ve specifických vrstvách.



Obrázek 5.20: Charakteristika třídy předmětu

Nastavení kurzoru

Při interakci s menu nebo inventářem je nutné hráči odemknout a zobrazit kurzor. Tuto akci řeší statická třída *UIData*. Stavy interakcí jsou definovány v podobě enumerační struktury, která je reprezentována binární maskou. Příslušné metody této třídy operují s bity podle vstupních parametrů. Jakákoliv změna nesmí narušit stav ostatních bitů. Proto aktivace či deaktivace stavu probíhá následovně:

- Pro aktivaci je využit logický operátor **OR**, který nastaví příslušný bit na jedničku.
- Pro deaktivaci je využit logický operátor **AND**, který nastaví příslušný bit na nulu podle invertovaného čísla stavu.

Během hraní je možné uvést herní kurzor do následujících tří stavů:

- Stav *CursorLockMode.None* zobrazí a odemkne myš.
- Stav *CursorLockMode.Locked* schová myš a zamkne ji na střed obrazovky.
- Stav *CursorLockMode.Confined* zobrazí myš a omezí její pohyb pouze v rozmezí obrazovky.

Inventář

Funkcionalitu inventáře spravuje skript *InventoryManager*, který především drží hráčův obsah inventáře ve struktuře *InventoryData*. Na začátku hry se dynamicky vytvoří sloty inventáře podle velikosti batohu. Každému slotu se přiřadí index a jeho obsah. Při inicializaci inventáře jsou všechny sloty prázdné a struktura předmětu je nastavená na *null*. Po inicializaci se načte z uložené hry obsah inventáře, který drží data jednotlivých slotů. Pokud načtený obsah není prázdný, tak se rozbalí do inventáře na příslušné indexy. Struktura inventáře se skládá z následujících informací:

- Struktura *InventoryItem*
- Množství v daném slotu

- Reference na herní slot
- Index slotu

Funkcionalita

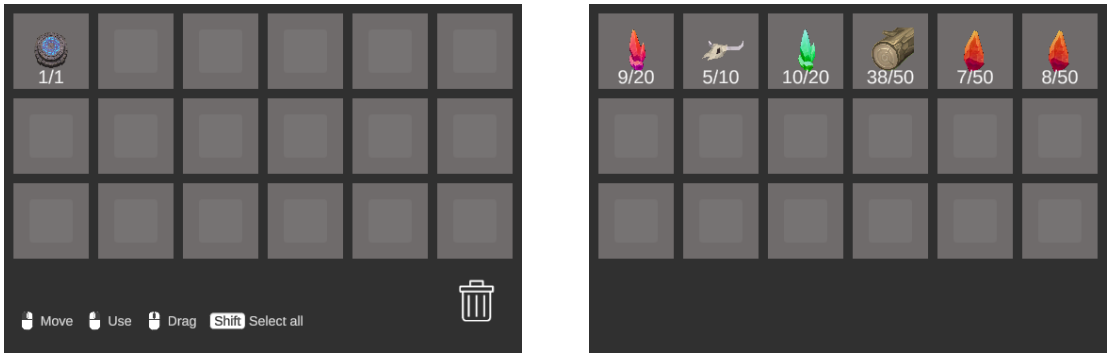
K přidání nové položky do inventáře je zapotřebí vědět o jaký předmět se jedná a jeho množství. Přidané množství se postupně doplňuje do počtu ve slotech, které obsahují stejný předmět s volným místem. S každým doplněním slotu se zmenší hodnota dočasné proměnné, která reprezentuje zbytek přidávaného množství. V případě, kdy jsou všechny sloty stejného předmětu plné, tak se nalezne první prázdný slot a tomu se zbytek přidělí. Pokud je inventář plný, tak příslušná funkce vrátí zbytek množství, které nebylo možné přidat do inventáře. Tato skutečnost může nastat v jakékoliv iteraci doplňování, proto je řádně ošetřena. Pro odebrání předmětu se na příslušném indexu sníží množství slotu o danou hodnotu. Pokud je množství ve slotu rovno nule, tak se automaticky tento slot vyčistí. V případě kdy jakýkoliv slot změní svoji hodnotu, tak je překreslen v grafickém rozhraní s novými daty. Ostatní funkce slouží jako výpomocné a fungují na podobném principu. Iterují skrze list předmětů, dokud nedokončí svůj úkol. Jedná se například o následující funkcionalitu:

- Mazání předmětů v pořadí podle indexů, dokud se nevymaže požadované množství.
- Zjištění celkového množství specifického předmětu v inventáři.
- Zjištění dostupného místa pro konkrétní předmět.
- Otevírání a zavírání inventáře.
- Zničení všech předmětů kromě teleportu.

Interakce mezi inventáři

Hráč může interagovat nejen s vlastním inventářem, ale i s ostatními objekty, které zprostředkovávají úložný prostor. Existují dva způsoby jak lze předmět mezi inventáři přesouvat. První možnost je pouhé kliknutí levého tlačítka myši, která přesune do aktuálně otevřeného inventáře jednu jednotku daného předmětu. Alternativní rozšíření první možnosti je použití klávesy `Shift`, která přesune všechny předměty z daného slotu. Druhou možností je tzv. *Drag–and–drop*, která umožňuje hráči uchopit myší předmět a ten přetáhnout na jiný slot, kde uživatel pustí tlačítko myši a tím se daný předmět přesune.

O provedení těchto akcí rozhoduje skript *ItemAction*, který se vyskytuje na každém slotu v inventáři. Provedení první akce zajistí příslušný skript inventáře daného slotu. Při provedení druhé akce se předají reference na oba inventáře skriptu *InventoryDragManager*, který provede odstranění předmětu v prvním inventáři a přidělí ho tomu druhému.



(a) Hráčův inventář.

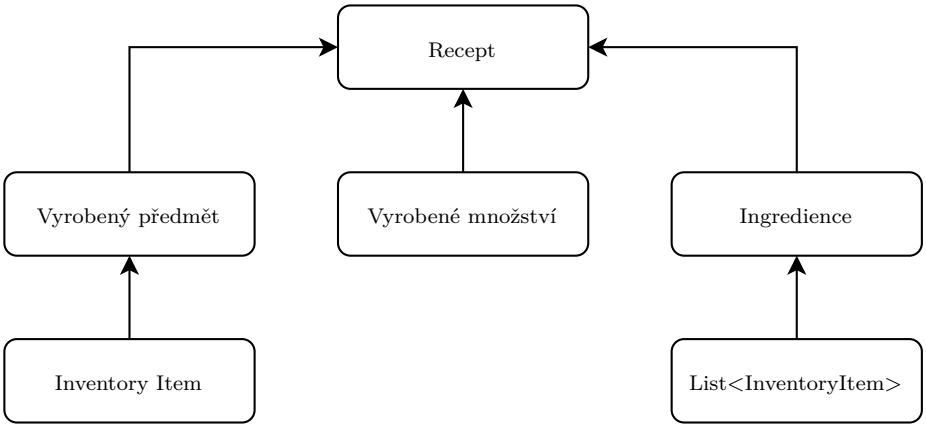
(b) Předměty v truhle.

Obrázek 5.21: Ukázka grafického rozhraní inventáře a truhly

Konstrukční stanice

Rozhraní umožňuje hráči vytvářet nové předměty, které potřebuje k opravení lodi. Předměty jsou ve stanici rozděleny do kategorií. Každý recept je definován strukturou `CraftItem` 5.22, který se skládá z následujících dat:

- **Vyrobené množství** určuje získané množství při výrobě předmětu.
- **Výrobní předmět** drží referenci na předmět, který hráč získá po sestavení.
- **Ingredience** tvoří list předmětů s jejich potřebným množstvím k zhotovení předmětu.



Obrázek 5.22: Struktura receptu.

Správu této stanice řídí skript *CraftDataManager*, který zobrazuje potřebné data a zajišťuje korektní výrobu předmětů. Kategorie předmětů 5.23a a jejich obsah je předem definován v podobě *prefabů*. Každý slot v kategoriích ovládá skript *CraftItemAction*, který drží referenci na příslušný recept. Informace o receptu zobrazuje pomocné okno 5.23b, které se naplní daty po jeho otevření.

Po stisknutí tlačítka „*Craft*“, se zkontroluje zda hráčův inventář obsahuje dostatečné množství potřebných ingrediencí k vytvoření předmětu. Pokud obsahuje všechny položky a

má místo v inventáři, tak se ingredience z inventáře odstraní a hráč získá nový předmět. Úspěšnou výrobu předmětu doprovází sekvence separátních audio stop.

(a) Kategorie výrobních předmětů (b) Pomocné výrobní okno

Obrázek 5.23: Ukázka rozhraní konstrukční stanice

Oprava lodi

Pokrok oprav rozbitého module řídí skript *RepairModuleManager*, který zobrazuje potřebné předměty k opravě 5.24 a ovládá příslušné sloty. Při inicializaci se dynamicky vytvoří instance slotů podle počtu předmětů v listu *repairItems*, což usnadňuje rozšíření tohoto modulu o nové předměty. Obsah slotů tvoří inventář, který se při uložení hry serializuje a uloží do souboru **SafeZoneData**.

Obrázek 5.24: Ukázka rozhraní porouchaného modulu

Přepínač dimenzí

K změně dimenzí je využita struktura *PointerEventData*, kterou poskytuje přímo Unity. Tato struktura se používá při práci s vstupními událostmi v rámci uživatelského rozhraní. Obsahuje informace o události související s kurzorem jako je kliknutí myší nebo dotyk

obrazovky. Potom co uživatel pustí tlačítko myši, tak nastane změna prostředí podle toho, na kterém objektu zůstal kurzor jako poslední.

Úvodní menu

Svět v úvodním menu byl vygenerován v hlavní úrovní a exportován volně dostupným nástrojem *FBX exporter*[8]. Následně byl svět upraven v modelovacím programu *3ds Max*[9] a importován zpět do Unity. Vegetace a další objekty jsou rozmístěny ručně jak je zobrazeno na obrázku 5.25.

Obrázek 5.25: Ukázka úvodního menu

Vytvoření světa

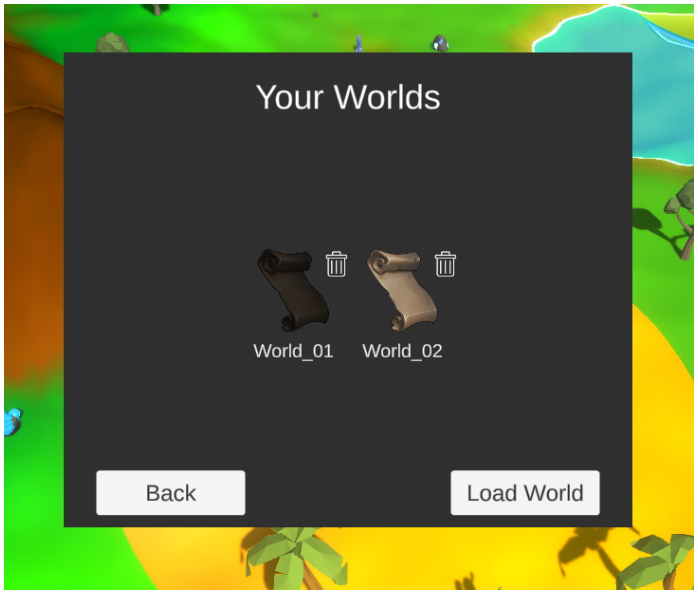
Pro vytvoření nového světa je nutné určit jeho jméno a parametr *seed*, podle kterého bude celý svět generován. Při inicializaci světa se vytvoří složka s odpovídajícím jménem a naplní se relevantními daty. Tato data drží informace o stavu světa a postupu hráče. Poté se načte nová scéna, která vygeneruje svět a hráč se umístí na náhodnou pozici.

Správa uložených světů

Všechny světy jsou uloženy ve složce `BP_Save`. Při načítání se načtou všechny složky, jejichž názvy odpovídají uloženým hrám jak je možné vidět na obrázku 5.26. Po kliknutí na tlačítko *Load World* se načte vybraný svět. Pokud hráč klikne na ikonu koše u daného světa, budou všechna jeho data včetně složky rekurzivně odstraněna.

[8]Více o nástroji FBX exporter na https://docs.unity3d.com/Packages/com.unity.formats.fbx@2.0/manual/index.html.

[9]Více o nástroji 3ds Max na https://www.autodesk.com/products/3ds-max/.



Obrázek 5.26: Ukázka správy světů v úvodním menu.

5.5 Implementace hádanek

Tato sekce popisuje systémy, které spravují jednotlivé hádanky. Ke každé hádance je přidělená truhla, která plní mnoho funkcionalit. Proto je tomuto systému věnována separátní sekce. V neposlední řadě jsou popsány techniky a systémy pro správu jednotlivých hádanek.

Systém truhel

Odemykání truhel řídí skript *OpenChestManager*, který také ovládá animátor bedny. Kvůli velkému počtu truhel ve scéně je po dokončení animace optimalizačně výhodné komponentu animátoru deaktivovat.

Truhly se dělí na dva typy podle jejich vzácnosti. O jejich typu rozhodne náhodné číslo, které se vygeneruje na základně zahešované pozice. Proto během generování světa není nutné ukládat typ beden, jelikož bude vždy stejný. Každý typ truhly generuje svůj obsah podle příslušných parametrů ve vlastní tabulce. Obě tabulky sdílí stejnou strukturu 5.1, ovšem každá obsahuje jiné předměty.

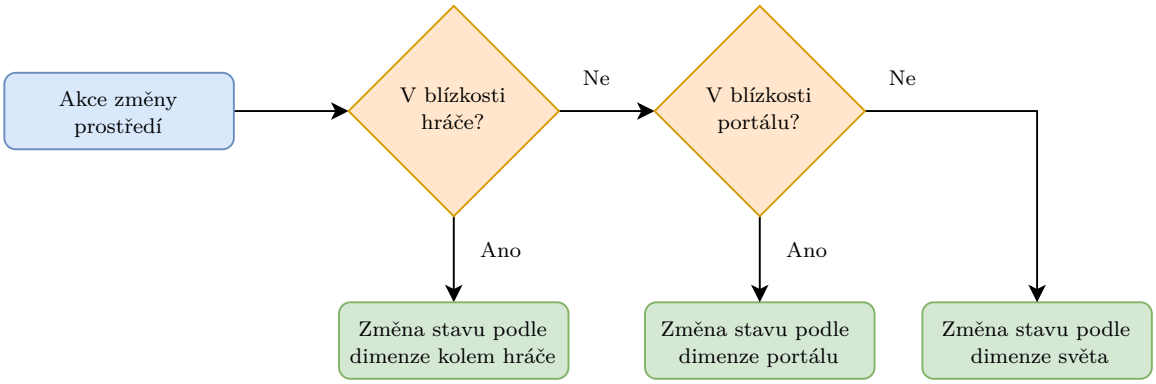
Předmět	**Min. množství**	**Max. množství**	**Pravd. výskytu**
Jakýkoliv předmět	Přirozené číslo	Přirozené číslo	0–100%

Tabulka 5.1: Reprezentativní ukázka struktury generovaných předmětů v podobě tabulky.

Poprvé když hráč otevře truhlu, tak se pro ni implicitně vygeneruje obsah inventáře. Generátor předmětů bere v potaz pravděpodobnost výskytu a množství v určitém rozsahu. Na začátku se zvolí náhodné číslo, které udává maximální možný počet předmětů v bedně. Poté se v určitém počtu iterací náhodně vybere index v příslušné tabulce a podle pravděpodobnosti výskytu daného předmětu se do truhly přidá nebo ne.

Systém hádanek

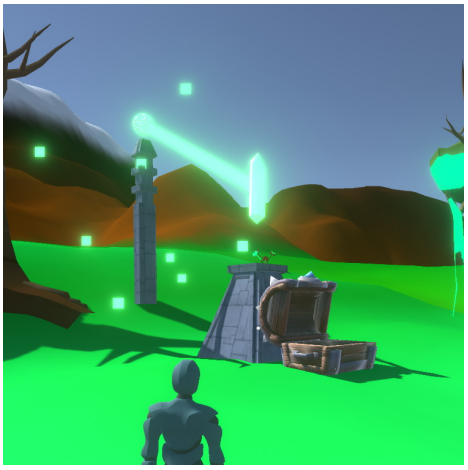
Jednotlivé hádanky lze rozdělit do dvou kategorií. Každou hádanku řídí příslušný skript, který kontroluje stavy svých podobjektů. Následný diagram 5.27 demonstruje aktivaci jednotlivých elementů hádanek tohoto typu.



Obrázek 5.27: Aktivování prvků hádanek podle změny dimenzí.

V první kategorii jsou prvky závislé na změně prostředí, která je ovlivněna portály a jejich prioritou. Aktivace kolem hráče funguje na principu kolizních *triggerů*. V případě přemístění portálu či změny jeho dimenze je proveden kontrolní výpočet vzdálenosti od portálu k danému hernímu objektu.

- **Energetický pilíř** je aktivní pouze v jedné dimenzi. Tuto dimenzi reprezentuje zbarvení vyzařovaných částic jak je vidno na obrázku 5.28a.
- **Aktivátory** se vyskytují ve trojicích a každý je aktivní v jiné dimenzi. Proto k jejich aktivaci je nutné využít všechny herní portály a změnit jejich dimenze podle barvy energetické koule. Ukázka této hádanky je zobrazena na obrázku 5.28b.



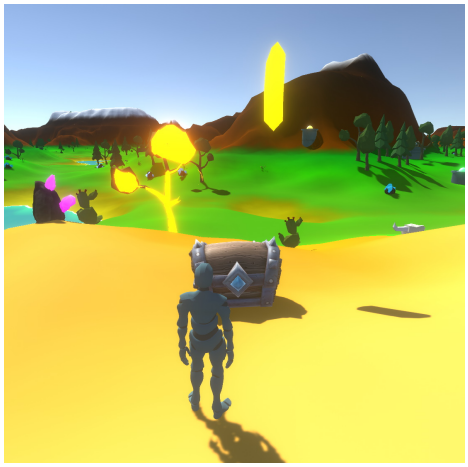
(a) Ukázka energetického pilíře

(b) Ukázka aktivátorů

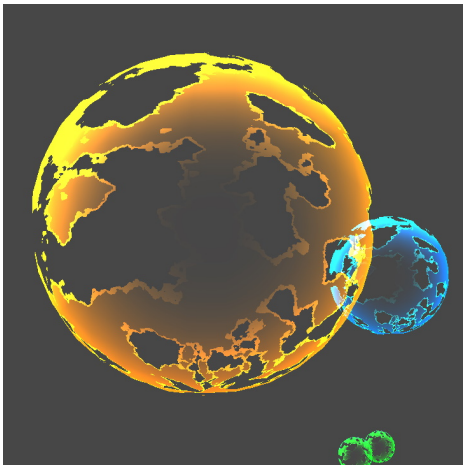
Obrázek 5.28: Znázornění hádanek prvního typu.

Klíčové prvky hádanek v druhé kategorii reagují na určitý typ energie. Při inicializaci se energeticky závislým objektům náhodně přidělí druh energie, který je reprezentován strukturou *EnergyType*. V případech, kdy se hádanky skládají ze tří částí, tak každé z nich je přidělen jiný druh energie, což nutí hráče aktivně měnit zobrazení světa. Ovšem pokud obsahuje více částí, tak je přidělení energie čistě náhodné. K tomuto typu se váží následující hádanky:

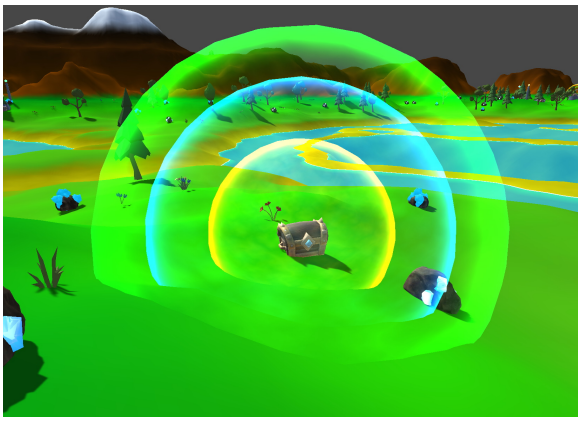
- Kolem **Bedny s krystaly** poletují tři energetické krystaly, které jsou viditelné pouze v dimenzích, kde je lze zničit. Po roztříštění všech krystalů se truhla odemkne. Na obrázku 5.29a je vidět pouze jeden vykreslený krystal příslušné dimenze.
- **Energetický štít** blokuje hráčovi přístup k bedně. Okraje tohoto štítu využívají *Fresnelův efekt*, k zvýraznění hran. Během toho, co hráč ničí laserem štít, tak se mění jeho viditelnost podle zbývajících životů. Ukázka této hádanky je na obrázku 5.29c.
- **Energetické koule** se nachází na obloze v blízkosti truhly. K vykreslení této koule je využit černobílý šum, který s klesajícími životy koule postupně zvyšuje práh rozpadu jak je vidno na obrázku 5.29b.



(a) Vznášející se energetický krystal kolem truhly.



(b) Ukázka energetické koule



(c) Ukázka energetického štítu

Obrázek 5.29: Znázornění hádanek druhého typu.

5.6 Logika umělé inteligence

Orientaci ve virtuální prostoru řeší komponenta *NavMeshAgent*. Tato komponenta poskytuje zabudovanou funkci *SetDestinantion()*, která podle vstupní pozice nalezne optimální cestu k danému bodu. Dále zohledňuje šířku a výšku agenta, jeho rychlost a schopnost navigace po šikmých površích. Co se týče animací, ovládání, přemýšlení a chování agenta, tak tyto vlastností je nutné definovat pomocí skriptů, které jsou blíže popsány v této sekci.

Chování agenta

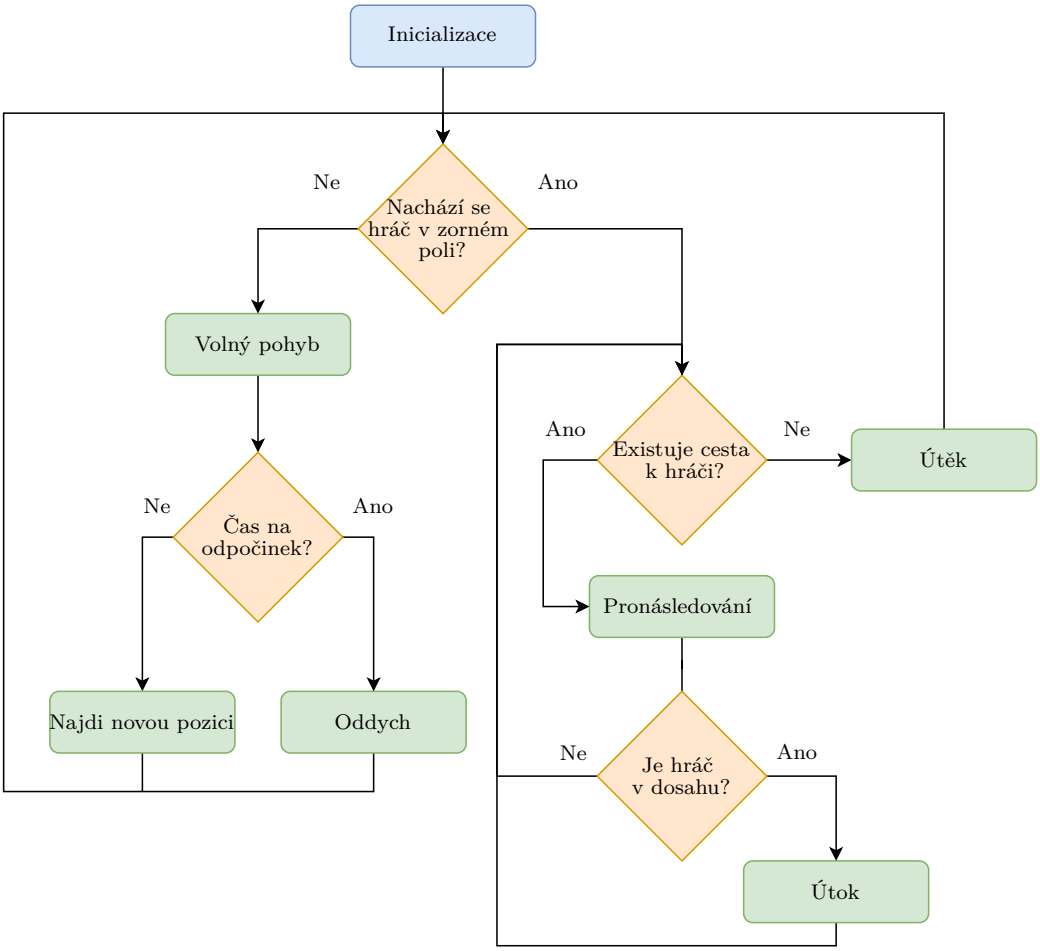
Skript *BearBehaviour* zodpovídá za chování agenta. Tento skript s určitou pravděpodobností uvede agenta do stavu bloudění nebo oddechu. Poté asynchronně čeká v náhodném časovém intervalu na vyhodnocení nového stavu.

Dohled agenta řídí skript *AgentVision*, který pomocí kolizního *triggeru* detekuje hráče. Pokud se vyskytne hráč v jeho dohledu, tak se přeruší všechny jeho aktuální akce a nastane stav souboje. Během tohoto stavu agent pronásleduje hráče do určité vzdálenosti a pokud se

dostatečně přiblíží, tak na něho zaútočí. V určitých intervalech v *coroutine* kontroluje, zda existuje bod na navigačním prostoru, který je v dostatečné blízkosti hráče. Pokud taková pozice neexistuje, tak se agent přepne do stavu útěku a běží na náhodnou pozici, která je protilehlá vůči hráči.

Agent nabývá následujících stavů, které definují jeho chování:

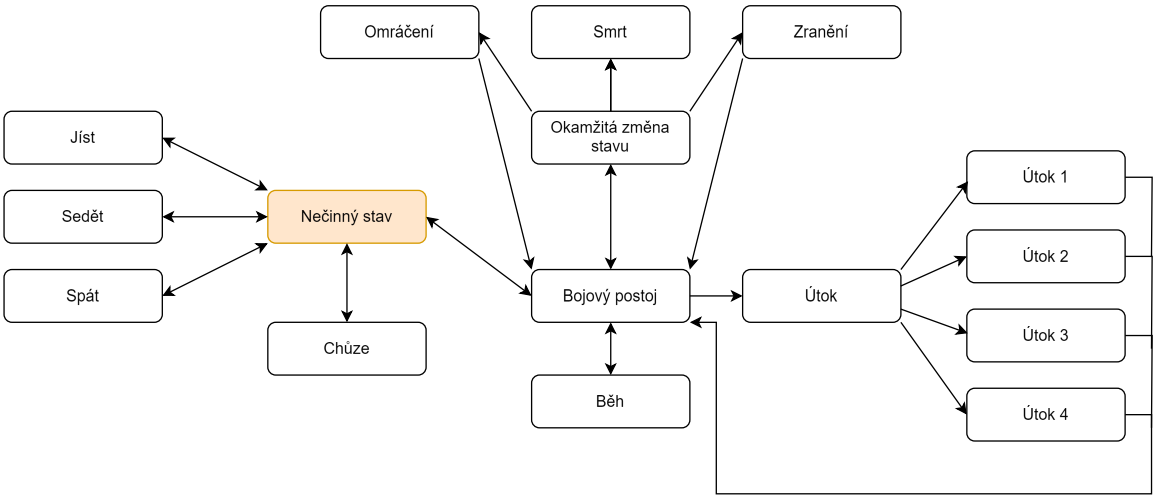
- **Nečinný stav** je pouze inicializační.
- **Stav bloudění** představuje volný pohyb agenta ve světě.
- **Bojový stav** uvede agenta do bojového režimu, kdy pronásleduje hráče do určité blízkosti a snaží se na něj zaútočit.
- **Stav oddychu** přeruší aktuální činnost a uvede agenta do nečinnosti.
- **Pauza** vypne chování agenta, pokud se vzdálí na určitou vzdálenost od hráče, z optimalizačních důvodů.
- **Smrt** představuje konečný stav.



Obrázek 5.30: Algoritmus pro změnu stavu agenta

Animační automat

Každá akce agenta je reprezentována sadou animací. Jelikož se stavy agenta během hry mění, tak tomu musí odpovídat i jeho animace. Správu animací řídí skript *BearAnimationController*, který pomocí pravdivostních parametrů a triggerů mění stavy agenta. Každý přechod mezi stavy je podmíněn parametry. V tomto kontextu existuje tzv. „*Any State*“, což je univerzální stav, který implicitně zajistí přechod do okolních stavů nezávisle na tom aktuálním. Přechod z toho speciálního stavu je podmíněn právě vyvolaným *triggerem*. Na diagramu 5.31 jsou znázorněny přechody mezi animacemi.

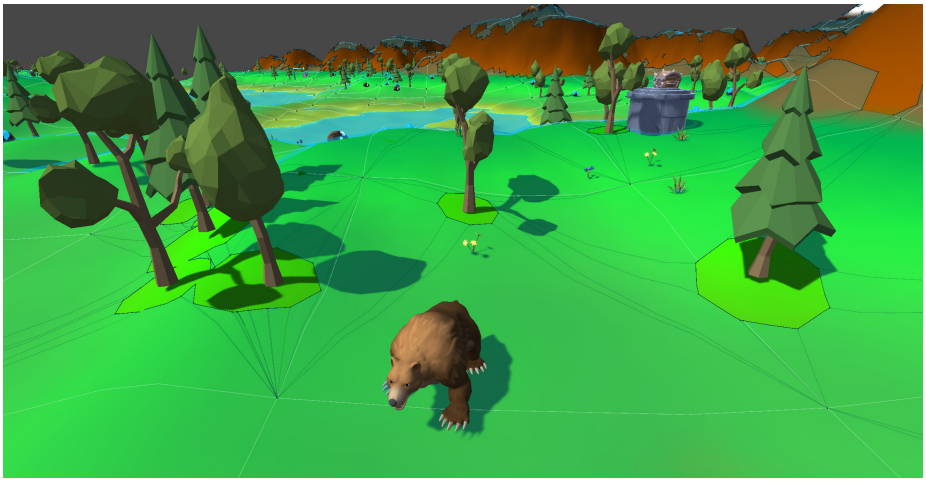


Obrázek 5.31: Ukázka animačního automatu agenta. Oranžově je znázorněn počáteční stav.

Reakce na změnu prostředí

Při pohybu je nutné kolem agenta neustále sledovat změnu dimenzí, které mění překážky v jeho okolí, kterým se snaží vyhnout. Proto je kolem agenta fyzikální trigger, který zajišťuje u vybraných objektů vyřezávání virtuálního prostoru, jak je zobrazeno na obrázku 5.32. Vyřezávání geometrie virtuálního prostoru umožňuje parametr *carving*[10], který poskytuje komponenta *NavMeshObstacle*.

[10]Více o vlastnostech vyřezávání na https://docs.unity3d.com/ScriptReference/AI.NavMeshObstacle-carving.html.



Obrázek 5.32: Vyřezávání geometrie virtuálního prostoru podle dimenze kolem agenta.

5.7 Zvukový systém

Unity disponuje základním audio systémem, který nabízí jednoduché nástroje v podobě komponent k přehrávání a modifikaci zvuků. Tento systém přehrává čtyři zvukové formáty (**AIFF**, **WAV**, **MP3**, **Ogg**). Zvukové stopy dokáže nejen přehrávat, ale i nahrávat z jednotlivých mikrofonů, které jsou připojeny k zařízení. Nejzásadnější nástroje jsou následující:

- **Zvukový posluchač** určuje orientaci a pozici posluchače ve scéně, proto je implicitně připnutý k hlavní kameře. V herní relaci vždy existuje právě jedna instance tohoto typu.
- **Zvukový zdroj** reprezentuje zvukový objekt v herním světě. Jeden zdroj přehrává pouze jednu zvukovou stopu, proto ve scéně existuje více těchto instancí, které vydávají různé zvuky. Ostatním komponentám umožňuje audio nahrávku modifikovat v podobě zvukových efektů. Mezi základní nastavení této komponenty patří například hlasitost, audio stopa, modifikace výšek, aktivace zvukových efektů, priorita, smyčka a mnoho dalších[11].
- **Zvukový mixér** je nástroj, který upravuje více druhů zvuků najednou. Umožňuje modifikovat určité skupiny zvuků zvlášť, což vytváří dynamičnost prostředí, které lze kdykoliv přizpůsobit herní situaci.
- **Zvukový filtr** přidává dodatečné efekty k celému zvukovému výstupu nebo jednotlivým zvukovým objektům. Mezi tyto efekty patří například ozvěna, reverb, zkreslení a dolní či horní propust.
- **Dozvuková zóna** zkresluje zvuky podle pozice posluchače relativně k prostoru dané zóny.

Přehrávání zvuků

O správu zvuků se stará skript *SoundManager*, který tvoří potřebné instance audio kanálů a drží odkaz na databázi audio stop v podobě listu. Jeho hlavním úkolem je najít volný zvu-

[11] Více o nastavení zvukových zdrojů na https://docs.unity3d.com/Manual/class-AudioSource.html.

kový kanál na kterém spustí danou audio stopu, kterou vyhledá pomocí názvu v databázi. Ovládá hlasitost herní muziky a všech ostatních kanálů dle nastavení v hlavním menu.

Při požadavku zastavení konkrétního zvuku zjistí, které kanály přehrávají daný zvuk a následně je v časovém intervalu odezní. Tuto funkcionalitu například využívá kamera, která se vynoří z vody. V neposlední řadě dovede například ztlumit všechny zvuky nebo přehrát náhodný výběr muziky.

Kapitola 6

Testování

V této kapitole jsou popsány metody a techniky k testování aplikace. V první části jsou popsány nástroje, které byly využity při průběžném testování během vývoje. Poté následuje důležitá sekce věnovaná testování s uživateli. V této sekci je prezentován souhrn zpětné vazby od uživatelů a hodnocení výsledného herního dema. Tato část je důležitá pro zjištění uživatelských preferencí, identifikaci potenciálních nedostatků a stanovení oblastí k dalšímu vylepšení.

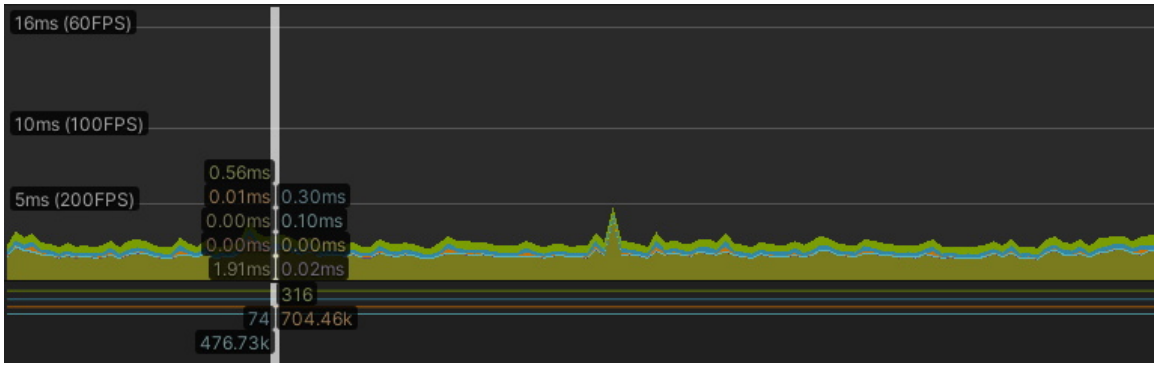
6.1 Nástroje k odladění aplikace

Během průběžného testování byl kladen důraz na ověření herních mechanismů a správného vykreslování herního světa. V pozdějších fázích vývoje bylo nezbytné zaměřit se více na optimalizaci procedurálního generování a prolínání dimenzí. Pro optimalizování Unity nabízí efektivní nástroje, které se soustřeďují na konkrétních části hry.

Profiler

Pro ověření výkonu byl použit integrovaný nástroj *Profiler*, který poskytuje informace o zatížení různých částí aplikace. Sleduje a zaznamenává důležité data o provedení každé akce, jak je popsáno na obrázku 6.1b. Na obrázku 6.1a je zobrazen graf, který zaznamenává v čase výkyvy výkonu, které se promítají do výsledného počtu snímků za sekundu. Barevné zvýraznění prvků v grafu má následující význam:

- **Světle hnědá barva** reprezentuje výkon spotřebovaný přímo editorem Unity. Tím pomáhá identifikovat poklesy snímků, které nejsou přímo způsobeny hrou. Jelikož tento nástroj také spotřebovává výkon k monitorování všech operací, může se stát, že způsobí výkyvy. Na základě toho lze odhadnout výsledný počet snímků ve zkompilované verzi hry.
- **Modrá barva** reprezentuje náročnost výpočtů prováděných herními skripty.
- **Zelená barva** značí spotřebovaný výpočetní čas při vykreslování scény.
- **Oranžová barva** reprezentuje fyzikální výpočty prováděné herním enginem.
- A mnoho dalších ...



(a) Ukázka vytížení v podobě grafů

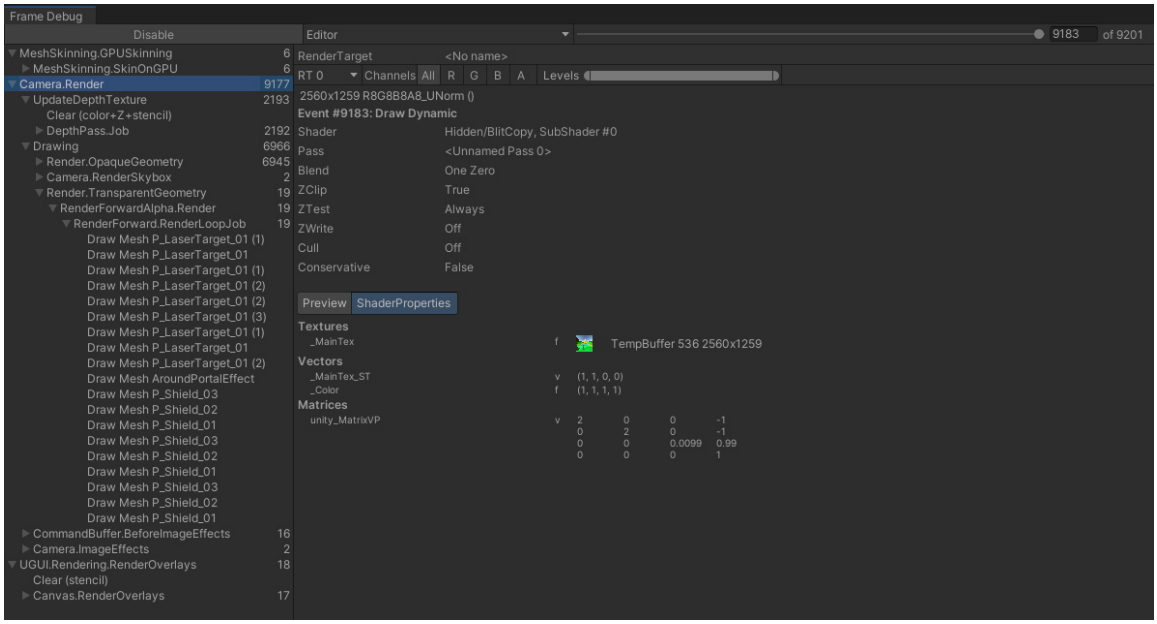
	Total	Self	Calls	GC Alloc	Time ms	Self ms
PlayerLoop	71.4%	1.1%	3	400 B	2.07	0.03
PreLateUpdate.AIUpdatePostScript	27.1%	0.0%	1	0 B	0.78	0.00
NavMeshManager	27.1%	0.0%	1	0 B	0.78	0.00
Components.NavMeshObstacle.Transform	27.0%	27.0%	1	0 B	0.78	0.78
Carving.Prepare	0.0%	0.0%	1	0 B	0.00	0.00
Camera.Render	20.7%	0.5%	1	0 B	0.60	0.01
PreUpdate.AIUpdate	6.8%	0.0%	1	0 B	0.19	0.00
Update.ScriptRunBehaviourUpdate	3.6%	0.0%	1	32 B	0.10	0.00
PreLateUpdate.DirectorUpdateAnimationEnd	1.8%	0.0%	1	0 B	0.05	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	1.7%	0.0%	1	0 B	0.05	0.00
PostLateUpdate.UpdateAllRenderers	1.2%	0.0%	1	0 B	0.03	0.00
PreLateUpdate.DirectorUpdateAnimationBegin	1.1%	0.0%	1	0 B	0.03	0.00
UGUI.Rendering.RenderOverlays	0.8%	0.0%	1	0 B	0.02	0.00
PostLateUpdate.UpdateAudio	0.6%	0.0%	1	0 B	0.01	0.00
PostLateUpdate.PlayerUpdateCanvases	0.6%	0.0%	1	0 B	0.01	0.00
PreUpdate.SendMouseEvents	0.5%	0.0%	1	0 B	0.01	0.00
GUI.Repaint	0.5%	0.2%	1	368 B	0.01	0.00
PostLateUpdate.UpdateAllSkinnedMeshes	0.4%	0.0%	1	0 B	0.01	0.00

(b) Ukázka funkcí a jejich statistik

Obrázek 6.1: Ukázka rozhraní *profileru*. Na řádcích se vyskytují názvy volaných funkcí. Ve sloupcích jsou například data o spotřebovaném výpočetním času, počet volaní a velikost příslušných alokací v bajtech.

Frame debugger

Tento nástroj je jedním z nejdůležitějších nástrojů pro optimalizaci výkonu a ladění grafiky při vývoji her v Unity. Každý vykreslený snímek je zde zachycen a poskytuje detailní analýzu každého vykresleného objektu, jak je patrné na obrázku 6.2. Jednotlivé snímky jsou uspořádány za sebou podle jejich vykresleného pořadí. Tento nástroj zejména zobrazuje informace o prováděných operacích a odpovídajících parametrech, které jsou klíčové pro daný snímek. Jeho hlavní výhodou je možnost jednoduchého krokování procesu vykreslování.



Obrázek 6.2: Ukázka rozhraní frame debuggeru

6.2 Uživatelské testování

První část této sekce zdůvodňuje výhody uživatelského testování. Následující sekce pak podrobně popisuje průběh testování s uživateli. Získané výsledky jsou následně prezentovány formou grafů a shrnuty v příslušných sekcích. Testování vyhovělo české herní studio *Gamanuj Games*, které poskytlo této práci profesionální zpětnou vazbu.

Výhody uživatelského testování

Pokud vývojář testuje svoji vlastní hru, tak má to své výhody i nevýhody. Jednou z výhod je například schopnost programátora myslet nejen jako hráč, který hru hraje, ale také jako ten, kdo zná logiku, která stojí za danými herními mechanismy. Tím pádem může snadno ověřit, zda všechny funkce fungují správně. Nicméně nevýhodou je, že často provádí akce ve správném pořadí a způsobem, jaký byl zamýšlen, což může vést k přehlédnutí potenciálních problémů. Proto je důležité provádět testování hry na uživatelích, kteří nemají představu o tom, jak herní mechaniky a prostředí fungují. To vytváří situace, které by vývojáři mohli přehlédnout. Hra je nakonec určena pro uživatele, kteří ji budou skutečně hrát. Z toho důvodu je klíčové získat jejich zpětnou vazbu a na jejím základě opravit případné nedostatky.

Realizace testování

Pro účely testování bylo vytvořeno herní demo, které mohli testeři vyzkoušet. K tomuto demu byl také přiložen manuál, který obsahoval popis základního konceptu hry a cílů, kterých měli testeři dosáhnout. Po splnění všech úkolů vyplnili dotazník, poskytli zpětnou vazbu a jejich připomínky byly diskutovány. S cílem zajistit možnost účasti uživatelů, kteří nemluví česky, byl tento manuál a dotazník psán v angličtině.

Hlavní úkoly

Pro usnadnění testování byl uživatelům předán seznam úkolů, který měl obecně otestovat celou funkcionalitu hry. K dokončení těchto úkolů bylo nutné využít všechny dostupné nástroje, orientovat se v uživatelském rozhraní, prolínat dimenze a prozkoumávat herní svět. Provedení těchto akcí mělo odhalit, jak uživatelé reagují a využívají všechny herní prvky dohromady. Seznam klíčových úkolů byl následující:

- Vytvoř svět, ulož hru, vrať se do hlavního menu a načti svět.
- Vyřeš alespoň tři typy hádanek.
- Vytěž pár surovin v každé dimenzi.
- Teleportuj se na základnu a využij výrobní stanici.
- Přidej součástky do opravného modulu lodi.
- Teleportuj se zpět na planetu.
- Zjisti podobu vody v každé dimenzi.
- Nalezni a zabij medvěda.

Forma dotazníků

Pro získání zpětné vazby od uživatelů byl vytvořen dotazník využívající **Likertovu škálu**, ve kterém testeři vyjadřují své názory pomocí stupnice od „silně souhlasím“ až po „silně nesouhlasím“. Při hodnocení složitosti hádanek je dotazník navíc obohacen o jejich obrázky, pro zjednodušení. Na konci dotazníku byl prostor sepsat své poznatky a názory v textové formě.

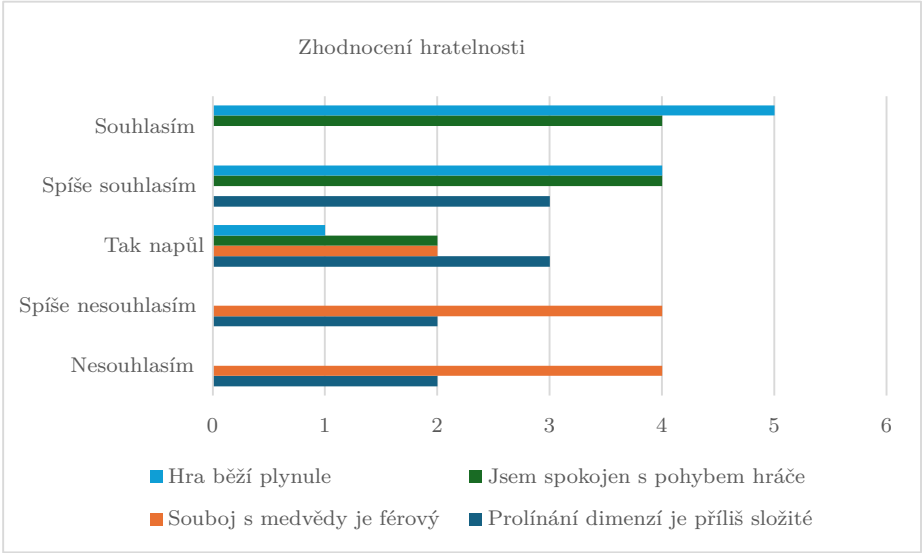
Hratelnost

Zhodnocení hratelnosti dopadlo víceméně pozitivně jak zobrazuje tento graf 6.3. Nejzásadnější poznatkem je, že hráči pochopili smysl a princip přepínání dimenzí. Ovšem většina testerů by ocenila, kdyby bylo možné změnit dimenzi celého světa jednou akcí, aby kvůli tomu nemuseli měnit prostředí kolem hráče.

Souboj s medvědy testeři zhodnotili vcelku negativně. Hlavním problémem je, že hráč nemá schopnost se útokům vyhýbat a medvědi jsou celkem houževnatí.

Pohyb hráče působí přirozeně, ovšem při neustálém skákání se hráč na chvíli zasekne v jedné animaci.

S plynulostí aplikace u většiny testerů nebyl žádný problém, ovšem naskytly se případy, kdy docházelo k menšímu poklesu snímků za vteřinu. Tento problém pravděpodobně způsobilo ukládání světa, které neprobíhá asynchronně.



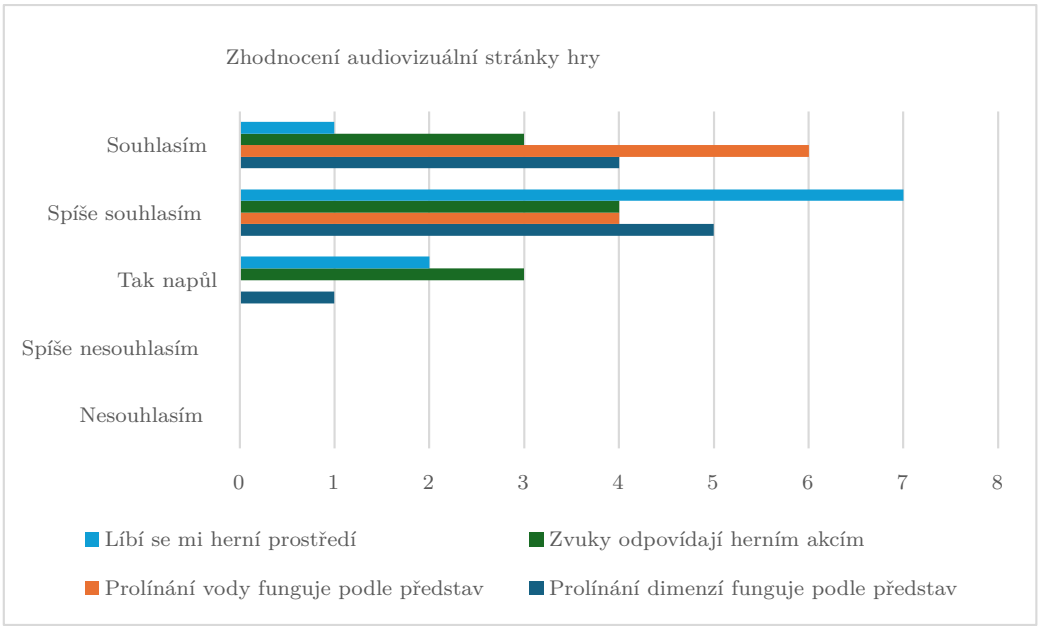
Obrázek 6.3: Výsledné zhodnocení hratelnosti

Zhodnocení audiovizuálních prvků

Celkové zhodnocení audiovizuálních prvků je zobrazeno ve grafu 6.4. Hlavním poznatkem je, že testerům přišlo prolínání dimenzí vizuálně intuitivní a působivé. V krajních případech, kdy je kamera umístěna na pomezí hranice portálu, vznikají ve světě nechtěné stíny.

Muziku a zvukové efekty testeři zhodnotili jako příliš hlasité. Většina zvuků byla označena za vhodné až na ozvučení vody. Vcelku chválili doprovázející zvuky kroků a prostředí při změně dimenzí.

Herní prostředí na testery zapůsobilo pozitivně, především kvůli stylizovanému designu a prolínání dimenzí.



Obrázek 6.4: Výsledné zhodnocení audiovizuálních prvků

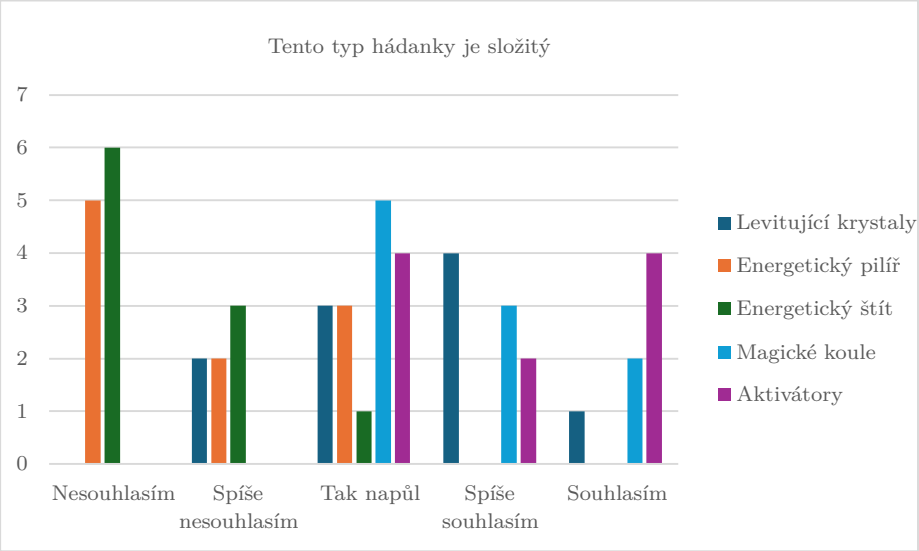
Složitost hádanek

Jednotlivé zhodnocení logických hádanek lze pozorovat na grafu 6.5. Největší problém způsobila hádanka, která zahrnuje tři aktivátory. Ta každému testerovi zabrala nejvíce času, protože kombinuje použití všech dostupných portálů a dimenzí.

Druhá nesložitější hádanka zahrnuje magické koule, které je nutné zneškodnit skrze portál pomocí laseru. Většina testerů si všimla poletujících koulí na obloze čirou náhodou během hraní. Ovšem ti, kteří objevili tento typ hádanky bez využití portálu, nevěděli co dělat. Po chvilce hraní na to přišli stejně jako ostatní.

Hádanka zahrnující létající krystaly je unikátní v tom, že se její jednotlivé krystaly zobrazují v odlišných dimenzích. Proto ji testeři zhodnotili jako středně složitou.

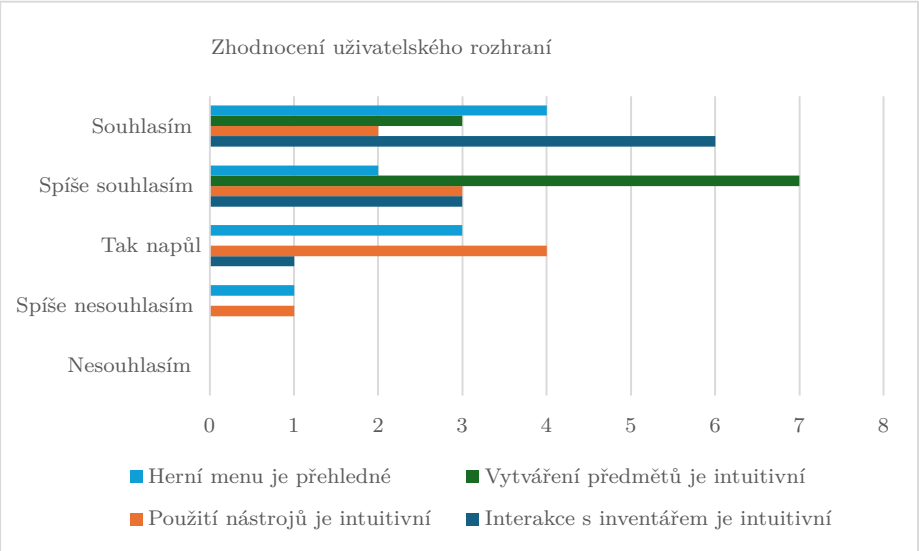
Ostatní typy hádanek působily víceméně primitivně a při řešení se nevyskytli žádné problémy.



Obrázek 6.5: Výsledné zhodnocení hádanek

Zhodnocení uživatelského rozhraní

Zhodnocení intuitivnosti uživatelského rozhraní je zobrazeno na grafu 6.6. Uživatelské rozhraní označili testeři za velmi přívětivé a intuitivní. Ocenili poskytnuté rady v pravém dolním rohu při změně nástrojů. Trochu zavádějící bylo použití laseru, jelikož ho testeři používali i na neenergetické objekty. Jeden z testerů měl problém pochopit rozdíl funkcionality tlačítek *Quit* a *Exit*.



Obrázek 6.6: Výsledné zhodnocení grafického rozhraní

Celkový dojem ze hry

Celkový dojem ze hry testeři označili za víceméně pozitivní, jak ukazuje graf 6.7. Je však důležité poznamenat, že se jedná pouze o herní demo, proto je snadné objevit všechny herní prvky poměrně rychle. Hádanky a herní objekty se ve světě opakují, proto je zásadní přidat více obsahu a obohatit tento svět o více zábavných prvků. I přes snahu o vyvážení časově náročného procesu tvorby předmětů a oprav lodi, byla tato úprava označena za přehnanou, zejména kvůli vysokým nákladům na výrobu některých předmětů a požadavku na množství součástek k opravě lodi.

V poslední části dotazníku měli účastníci možnost sdělit své poznámky a nápady pro budoucí rozšíření hry. Jak bylo předtím naznačeno, testeři by ocenili více herního obsahu. Rozšířili by hru o další hádanky, nepřátele, suroviny a vylepšili by dynamičnost terénu. Dále by přidali možnost vytvářet více spotřebních předmětů.

Koncept hry ohodnotili jako mimořádně zajímavý, protože nikdy dříve nehráli či neviděli podobnou hru. Herní mechaniky spojené s netriviálním vykreslováním je zaujali a vidí v nich velký potenciál.

Obrázek 6.7: Výsledné zhodnocení hratelnosti

Kapitola 7

Závěr

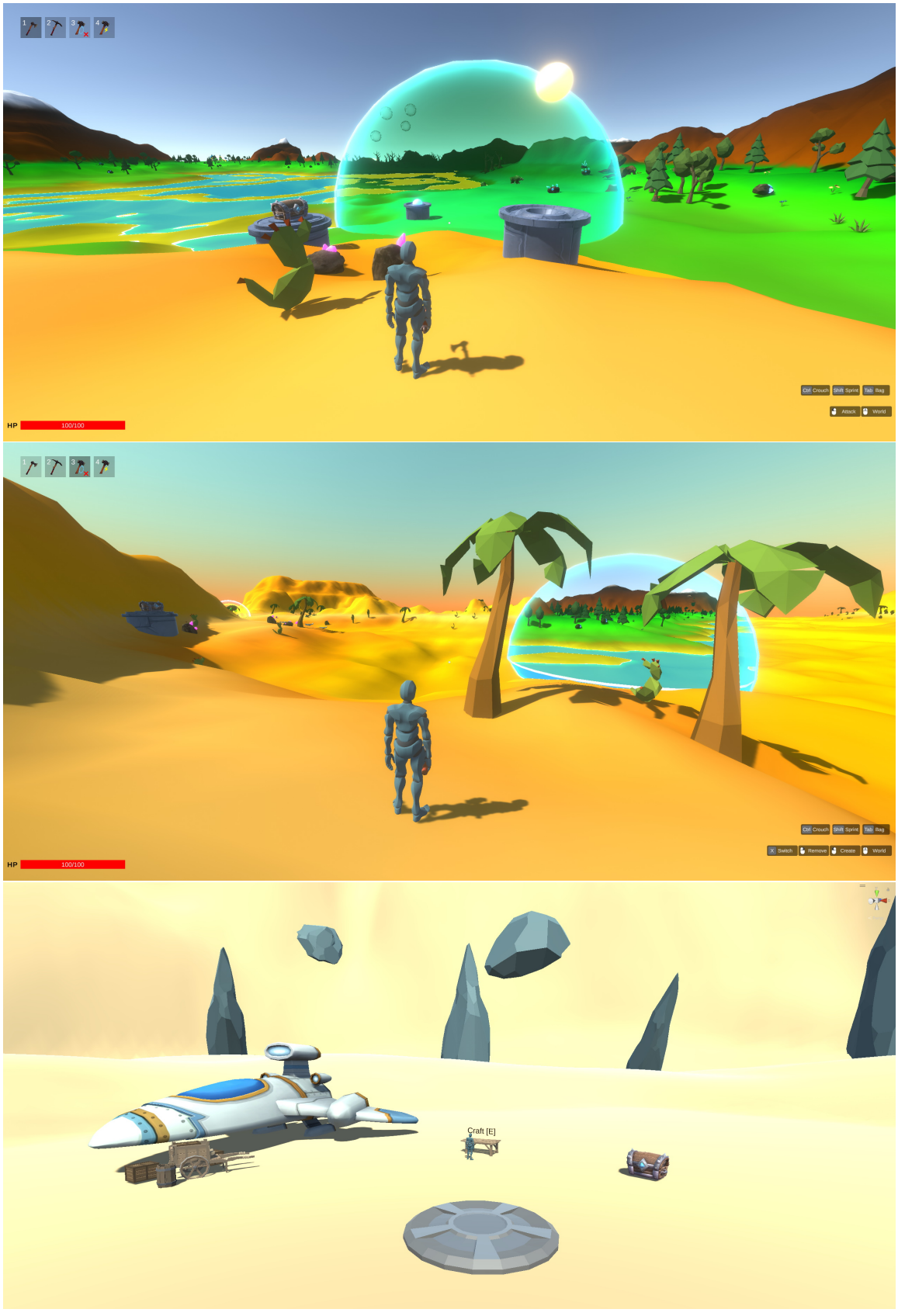
Hlavním cílem této práce bylo vytvořit plně funkční herní demo, které využívá herní mechaniky založené na netriviálním vykreslováním. Tento cíl byl úspěšně splněn.

Celý koncept hry je zasazen do procedurálně generovaného světa, který by nebylo možné vytvořit bez paralelních procesů. Podle průzkumu trhu byla vytvořena zcela unikátní hra, která využívá portály netradičním způsobem. Celkem může hráč prolínat dimenze mezi třemi světy. Kromě prolínání a vykreslování dimenzí hra disponuje mnoha dalšími prvky, jako je ukládání světa, správa herního inventáře, výroba předmětů, kompletní implementace hráče včetně pokročilého animačního systému a integrace umělé inteligence do tohoto neobvyklého světa. V této fázi herního dema je hra plně hratelná a hráč má možnost ji dohrát.

Pro úspěšné dokončení tohoto projektu bylo nezbytné nutné nastudovat strukturu herního vývojového prostředí Unity a proces grafického vykreslování. Herní mechanika prolínání světů je postavena na využití stencil bufferu, který je kombinován s dalšími technikami vykreslování. Procedurální generování světa využívá modifikaci Perlinova šumu. Realizaci pohybu nepřátelských postav umožnil experimentální balíček NavMesh poskytnutý přímo Unity technologies. Výsledné herní demo prošlo uživatelským testováním, ze kterého byla provedena analýza, zdokumentovány poznatky a sepsán závěr.

Celkově považuji tuto práci jako významný přínos pro mé vzdělání. Do určité míry jsem porozuměl principům vykreslování a paralelního zpracování, což mi otevřelo nové obzory ve světě herního vývoje. Zlepšil jsem své dovednosti v programování her a pochopil důležitost optimalizace výsledné aplikace.

V práci bych chtěl pokračovat tak, že bych ji rozšířil o další herní obsah, který by obohatil herní svět a rozšířil ho o další herní mechaniky. Před zveřejněním této práce bych chtěl zapracovat na dalších elementech hry, které nebyly hlavním cílem této práce jako je například tvorba vlastních 3D modelů, rozšíření hratelnosti, vizuální efekty a tak podobně. V druhé řadě bych chtěl umožnit hráčům vytvářet více instancí portálů. Ovšem k tomu by bylo zapotřebí zdokonalit techniku ořezávání, která by vykreslovala stíny podle více rozmístěných portálů v prostoru současně. Myslím si, že další velkou výzvou by mohla být realizace této hry pro více hráčů. Synchronizovat tento herní svět s více uživateli a umožnit jim vzájemně prolínat dimenze mezi sebou.


HP
100/100
Ctrl Crouch
Shift Sprint
Tab Bag
Attack
World
HP
100/100
Ctrl Crouch
Shift Sprint
Tab Bag
X Switch
Remove
Create
World
Craft [E]

Literatura

[1] Ablett, D., Cunningham, A., Lee, G. A. a Thomas, B. H. Portal Rendering and Creation Interactions in Virtual Reality. In: *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2022, s. 160–168. DOI: 10.1109/ISMAR55827.2022.00030.

[2] Adams, E. a Dormans, J. *Game Mechanics Advanced Game Design*. 1. vyd. New Riders, 2012. ISBN 0321820274.

[3] Aleem, S., Capretz, L. F. a Ahmed, F. Game development software engineering process life cycle: a systematic review. *Journal of Software Engineering Research and Development*. 2016, sv. 4, č. 1, s. 6. DOI: 10.1186/s40411-016-0032-7. ISSN 2195-1721. Dostupné z: https://jserd.springeropen.com/articles/10.1186/s40411-016-0032-7.

[4] Archer, T. Procedurally Generating Terrain. *MICS Symposium Proceedings*. Sioux City, Iowa 51106: [b.n.]. 2011. Dostupné z: https://micsymposium.org/mics_2011_proceedings/mics2011_submission_30.pdf.

[5] Batagelo, H. a Costa, I. Real-time shadow generation using BSP trees and stencil buffers. In: *XII Brazilian Symposium on Computer Graphics and Image Processing (Cat. No.PR00481)*. 1999, s. 93–102. DOI: 10.1109/SIBGRA.1999.805714.

[6] Biagioli, A. *Understanding Perlin Noise*. 2014. Dostupné na https://adrianb.io/2014/08/09/perlinnoise.html.

[7] Bishop, L., Eberly, D., Whitted, T., Finch, M. a Shantz, M. Designing a PC game engine. *IEEE Computer Graphics and Applications*. 1998, sv. 18, č. 1, s. 46–53. DOI: 10.1109/38.637270.

[8] Buttfield, A., Manning, J. a Nugent, T. *Unity Game Development Cookbook*. 1. vyd. Gravenstein Highway North: OReilly Media, 2019. ISBN 1491999152.

[9] Games, R. B. *Making maps with noise functions*. 2022. Dostupné na https://www.redblobgames.com/maps/terrain-from-noise/.

[10] Goldstone, W. *Unity 3.x Game Development Essentials*. 2. vyd. Birmingham, UK: Packt Publishing Ltd, 2011. ISBN 978-1-84969-144-4.

[11] Gregory, J. *Game Engine Architecture*. 3. vyd. A K Peters/CRC Press, 2018. ISBN 1138035459.

[12] Hasu, J. *Fundamentals of shaders with modern game engines*. 2018. Diplomová práce. Lappeenranta University of Technology.

[13] Hocking, J. *Unity in Action*. 1. vyd. Shelter Island, NY: Manning Publications, 2015. ISBN 161729232X.

[14] Ilett, D. *Building Quality Shaders for Unity*. 1. vyd. Coventry, UK: Apress, 2022. ISBN 1484286510.

[15] Lengyel, E. *Mathematics for 3D Game Programming and Computer Graphics*. 1. vyd. Charles River Media, 2001. 382 s. ISBN 1-58450-037-9.

[16] Lowe, N. a Datta, A. A technique for rendering complex portals. *IEEE Transactions on Visualization and Computer Graphics*. 2005, sv. 11, č. 1, s. 81–90. DOI: 10.1109/TVCG.2005.1.

[17] Norton, T. *Learning C# by Developing Games With Unity 3D Beginner's Guide*. 1. vyd. Birmingham, UK: Packt Publishing, 2013. ISBN 1849696586.

[18] Polack, T. *Focus On 3D Terrain Programming (Game Development)*. 1. vyd. Premier Press, 2003. ISBN 1592000282.

[19] Ramadan, R. a Widyani, Y. Game development life cycle guidelines. In: *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE, 2013. ISBN 978-979-1421-19-5.

[20] Shaker, N., Togelius, J. a Nelson, M. J. *Procedural Content Generation in Games*. 1. vyd. Springer Cham, 2016. ISBN 978-3-319-42716-4.

[21] Thanh, L. *Procedural terrain generation using Perlin noise*. 2023. Diplomová práce. Faculty of California State Polytechnic University, Pomona.