Digitalizace textové hry (Gamebooku) pomocí nástroje Unity

Hana Chrenčíková

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Hana Chrenčíková**
Osobní číslo:	**A20490**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Digitalizace textové hry (Gamebooku) pomocí nástroje Unity**
Téma práce anglicky:	**Digitization of a Gamebook using the Unity Tool**

Zásady pro vypracování

1. Vypracujte literární rešerši na téma prohledávání mapy a hledání řešení v prostoru.
2. Popište použité pracovní prostředí.
3. Implementujte problém pomocí vhodné modelové reprezentace a popište algoritmy umělé inteligence, které mohou problém vyřešit – optimalizovat.
4. Problém nasimulujte pomocí vhodného herního enginu.
5. Porovnejte skóre dosaženého pomocí algoritmů umělé inteligence se skóre dosaženého lidskými subjekty.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Markov Decision Processes in Practice BOUCHERIE, Richard J. a Nico M. VAN DIJK, ed. Markov Decision Processes in Practice [online]. Cham: Springer, 2017 [cit. 2022-11-29]. ISBN 978-3-319-47766-4. Dostupné z: https://link.springer.com/book/10.1007/978-3-319-47766-4
2. UNITY – První seznámení s tvorbou počítačových her, Tomáš Holan HOLAN, Tomáš. Unity: první seznámení s tvorbou počítačových her. Praha: CZ.NIC, z.s.p.o., 2020. CZ.NIC. ISBN 978-80-88168-57-7.
3. Deep learning on graphs, Yao Ma Jiliang Tang MA, Yao a Jiliang TANG. Deep Learning on Graphs. Cambridge: Cambridge University Press, 2021. ISBN 978-1108831741.
4. Unity Documentation [online]. San Francisco, 2022 [cit. 2022-11-29]. Dostupné z: https://docs.unity.cn/2022.1/Documentation/Manual/ManualVersions.html
5. Artificial Intelligence engines: A tutorial introduction to the mathematics of deep learning STONE, James V. Artificial intelligence engines: a tutorial introduction to the mathematics of deep learning. USA: Sebtel Press, 2019. ISBN 978--0-9563728-1-9.

Vedoucí bakalářské práce: **Ing. Adam Ulrich**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Cílem této práce je přiblížit koncept textové hry dětem a mládeži ve věku 6 až 15 let, poskytnout jim motivaci jít do knihovny, kde si mohou hru zahrát a ukázat jim tak, že i čtení může být zábavné. Druhá část práce se zabývá vývojem algoritmu umělé inteligence a porovnáním jeho výkonu s lidskými hráči.

Toto srovnání umožní optimalizovat obtížnost hry a poskytne poznatky o preferencích a schopnostech mladších hráčů. Mezi očekávané výsledky patří lepší vyvážení obtížnosti hry, cenná doporučení pro navrhování textových her pro tuto věkovou skupinu a poznatky o potenciálních přínosech interaktivního čtení.

Klíčová slova: Unity, C#, Umělá inteligence, Textové hry

ABSTRACT

The aim of this thesis is to introduce the concept of a gamebook to children and young people aged 6 to 15, motivate them to go to the library to play the game and to show them that reading can be fun. The second part of the thesis deals with the development of an artificial intelligence algorithm and comparing its performance with human players.

This comparison will allow to optimize the difficulty of the game and provide insights into the preferences and abilities of younger players. Expected results include a better balance of game difficulty, valuable recommendations for designing gamebook-based games for this age group, and insights into the potential benefits of interactive reading.

Keywords: Unity, C#, Artificial Intelligence, Gamebook

Ráda bych poděkovala vedoucímu mé bakalářské práce panu Ing. Adamu Ulrichovi za odborné vedení a cenné rady, které mi poskytl.

Dále bych chtěla poděkovat panu Ing. Tomáši Vogeltanzovi, Ph.D. jehož hodiny Vývoje počítačích her mi značně usnadnily tvorbu počítačové hry v Unity.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Inspirací a hlavní motivací pro tuto práci byla knihovnice z knihovny v Halenkovicích, která se na mě obrátila s prosbou o pomoc. Dělalo ji starost, že děti a mládež projevují malý zájem o čtení. Přestože se snažila pořizovat různé druhy knih, žádná nedokázala zaujmout jejich pozornost. Knihovna vlastnila poutavou textovou hru, která bohužel zůstala od svého pořízení nedotčena. Kdykoli ji knihovnice nabídla k vypůjčení, děti často uváděly jako odrazující faktor její tloušťku. Tento argument zřejmě převládal i u většiny dalších knih. Po vyslechnutí této myšlenky se zrodil nápad převést koncept textových her do digitální podoby.

Převedením knihy na digitální hru děti již nebudou mít možnost posuzovat délku textové hry z hlediska počtu stránek. Hry jsou mezi dětmi a dospívajícími nesmírně oblíbené a mají tendenci se do jejich hraní ochotně zapojovat. Při vývoji hry jsem brala v úvahu také skutečnost, že předložení ohromného množství textu na začátku by nemuselo vzhledem k okolnostem přinést výrazný úspěch. Proto jsem se snažila najít rovnováhu. Následně jsem se ujala psaní příběhu a herního textu. Dalším zásadním aspektem bylo grafické zpracování hry, které jsem se snažila udělat barevné, a především co nejméně strašidelné. Výběr vhodné hudby byl posledním krokem, který byl nutný k tomu, aby hra ožila.

Pro algoritmus umělé inteligence jsem zvolila algoritmus Monte Carlo Tree Search. Při implementaci pro řešení ostatních her vykazoval dobré výsledky a počítá s nejistými proměnnými, takže se pro mou hru dobře hodí. Zároveň není příliš sofistikovaný, když se používá samostatně, což jej činí vhodným pro srovnání s cílovou skupinou.

I. TEORETICKÁ ČÁST

1 HERNÍ ENGINE

Něž se začne vyvíjet jakákoli hra, je nutné vybrat vhodný software. V případě her se nejčastěji používá speciální software zvaný herní engine [1]. Tento software kombinuje všechny aspekty hry do jednoho celku a umožňuje tak urychlit vývoj hry. Herní engine zajišťuje načítání herních objektů, vykreslování grafiky, zprostředkovává zvuk a herní fyziku, osvětlení a vše ostatní potřebné k vytvoření počítačové hry.

Vývoj hry bez tohoto speciálního softwaru je samozřejmě možný, ovšem je také náročnější a zdlouhavější. V současnosti jich existuje hned několik. Mezi nejznámější se řadí Unity Game Engine a Unreal Engine. Pro potřeby této práce byl vybrán Unity Game Engine.

1.1 Unity Herní Engine

Unity game Engine, známý též jako Unity, vyšel prvně v roce 2005. Vytvořila jej firma Unity Technologies, která byla založeno o rok dříve, tedy 2004. Původně byl určen pro vývoj na operační systém Mac OS X, ale později byla přidána i podpora pro vývoj na Windows a Linux. V současnosti je možné spolu s desktopovými aplikacemi vyvíjet v Unity i pro mobilní, konzolové a webové aplikace.

Unity nabízí několik licencí [2]. Základní licence je licence pro osobní použití. Tato licence umožňuje používat a vyvíjet na platformě Unity bezplatně, pokud jednotlivec nebo firma nepřekročí výdělek více než 100 000 dolarů za rok. Podobná je licence pro studenty, která je dostupná pro všechny studenty starší 16 let. Unity dále nabízí i některé placené licence, ale při tvorbě této práce je využíváno pouze licence pro osobní použití.

1.2 Základy Unity

1.2.1 Prostředí

Unity má dedikovaný software s názvem Unity3D Editor, který slouží jako vývojářské prostředí pro práci s Unity Game Engine. Prostředí se skládá z několika oken. Ty lze libovolně přesouvat, zavírat a otevírat. Jedním z nejdůležitějších je okno Scene, kde se zobrazují aktivní objekty scény.

1.2.1.1 Scene

Okno Scene umožňuje uživateli provádět změny ve scéně v reálném čase. Jde objekty přesouvat, zvětšovat či zmenšovat, různě otáčet a podobně. Poskytuje takový náhled, jak bude scéna ve hře vypadat.

1.2.1.2 Hierarchie

Dalším důležitým oknem je Hierarchie. Zde se zobrazuje struktura scény, tedy všechny objekty, které daná scéna obsahuje a vazby mezi nimi. Jeden herní objekt může obsahovat více objektů, je tedy jejich rodičem. Díky tomu se dá přesouvat a upravovat více objektů najednou. Rodič může mít neomezené množství potomků.

1.2.1.3 Inspector

Při kliknutí na některý objekt v okně Hierarchie se zobrazí jeho vlastnosti a všechny jeho komponenty v okně Inspector. Inspector je velice důležité okno, protože nám říká vše o objektu, jako jeho jméno, pozici, jaké komponenty obsahuje, v jaké vrstvě se nachází a další. Taky lze díky oknu Inspector objekty zapínat a vypínat, přidávat, odebírat a nastavovat komponenty, upravovat vzhled objektů, měnit jejich polohu a další. V případě skriptů zobrazuje Inspector obsah skriptu, tedy kód, aniž by potřeboval vývojové prostředí. Pro úpravu kódu je ale už vývojové prostředí zapotřebí.

1.2.1.4 Project

V okně Project můžeme najít všechny složky a soubory, které náš projekt obsahuje. Umožňuje jednoduše přesouvat, přidávat a mazat soubory uložené v projektu. Lze tak provádět veškeré akce se soubory přímo v Unity editoru, bez nutnosti přepínat do průzkumníku souborů. Okno Project je také velmi užitečné díky funkci „drag and drop“ pomocí které může uživatel jednoduše přiřadit některý soubor k objektu nebo komponentě.

1.2.1.5 Game

Asi nejzajímavější okno je okno Game. Toto okno zobrazuje pohled kamery. V Unity editoru lze pomocí tlačítka Play na horní liště spustit předběžnou verzi hry (hra ještě nemá všechny funkcionality a vlastnosti hry). Hra se spustí právě v okně Game, kde lze vidět, jak všechno funguje v pohybu a jestli je to v pořádku. Hru lze pozastavit kliknutím na tlačítko Stop, které je vedle tlačítka Play na horní liště. K úplnému ukončení se musí potom znovu kliknout na tlačítko Play. Během doby, kdy hra běží, lze různě upravovat herní objekty a

vidět změny přímo ve hře, ale po ukončení hry se všechny uživatelem provedené změny vrátí zpět.

1.2.2 Scéna

V Unity nám jako základ slouží tzv. scéna [3]. Ta obsahuje herní objekty, které se na ní pohybují a opakovaně vykreslují. To, jak se budou objekty vykreslovat, určuje kamera, která představuje pohled hráče. Další nutnou součástí scény je světlo. Bez něj by nebylo v kameře nic vidět. Scén samozřejmě může být ve hře více než jen jedna.

1.2.3 Vykreslování scén

K renderování grafiky lze využít několik „Render Pipelines“ [2]. Ty se starají o to, aby se objekty ve scéně zobrazily na obrazovce. Unity má ve výchozím nastavení Built-in Render Pipeline, která je vhodná pro širokou škálu zařízení. Standartně nabízí další dvě, a to URP (Universal Render Pipeline) a HDRP (High Definition Render Pipeline). URP (Universal Render Pipeline) je optimalizovaná tak, aby mohla běžet na mobilních zařízeních i na počítačích a herních konzolích. Je možné si ji přizpůsobit a rozšířit dle vlastních potřeb. HDRP (High Definition Render Pipeline) nabízí podstatně kvalitnější vykreslování, ale je omezena na hardware s vysokým výkonem, tedy počítače s dedikovanou grafickou kartou a výkonnější herní konzole. Samozřejmě i HDRP je přizpůsobitelná, ovšem jak URP, tak i HDRP mají svá omezení co se týče možnosti přizpůsobení. Z tohoto důvodu Unity nabízí i možnost použít kompletně vlastní vykreslování.

1.2.4 Fyzika

Unity má už v sobě systém simulace fyziky, který zajišťuje, že objekty kolidují s jinými objekty, zrychlují, zpomalují a obecně simulují chování reálných objektů ve skutečném světě [2]. Usnadňuje se tím programování a zvyšuje pocit reálnosti hry. Unity nabízí několik základních fyzikálních modulů, které si může uživatel zvolit podle typu hry. Jsou to 3D, 2D, objektově-orientované moduly a datově-orientované moduly. Datově-orientované systémy nejsou na rozdíl objektově orientovaných 3D a 2D modulů, součástí základního balíčku Unity, ale dají se dodatečně stáhnout přes Unity Asset Store.

1.2.5 Scripty

K vytvářením herní logiky a přecházení mezi scénami slouží skripty. Ty se sepisují v jazyce C#. Ke tvorbě skriptů je ve výchozím nastavení pro Windows využíváno vývojového

prostředí Microsoft Visual Studio [2], ale pokud nejsme s tímto vývojovým prostředím spokojeni, Unity nabízí alternativy v podobě Microsoft Visual Studio Code nebo JetBrains Rider.

1.2.6 C#

C# je moderní objektově orientovaný programovací jazyk a staticky typovaný jazyk, to znamená, že každá proměnná musí být explicitně deklarována s určitým datovým typem. C# patří mezi C jazyky a je tudíž velmi podobný jazykům C, C++, Java a JavaScriptu [4]. Tento jazyk má široké uplatnění ve vývoji desktopových aplikací, webových aplikací, her a mobilních aplikací pro systémy Windows, iOS, Android a další platformy.

1.2.7 Prefabs

V případě, že uživatel potřebuje znovupoužít některý ze svých objektů, je možné jej uložit jako tzv. prefab. Prefaby si udrží stejnou strukturu, včetně všech komponent, kterou měly ve scéně. Mohou tak sloužit jako vzor, podle kterého můžeme vytvářet instance. Výhoda prefabu oproti kopii je v tom, že pokud změníme prefab, změní se všechny instance. Další výhodou je, že instance můžeme vytvářet skriptem. To se hodí třeba na vytváření velkého množství stejných objektů, například střel ve hrách nebo čehokoliv, co nebude ve scéně od začátku [3].

Prefab taky může sloužit jako základ pro vytváření podobných objektů. Například máme prefab ve tvaru krychle. Naše krychle má modrou barvu a váhu půl kila a máme nastavené, že na ni působí gravitace. Jelikož je to prefab, můžeme jich jednoduše do scén vložit několik. Jakmile ji máme ve scéně, můžeme měnit vlastnosti individuálních krychlí, aniž bychom ovlivnili ostatní. Můžeme tak mít tak jednu červenou krychli, jednu zelenou, další třeba kilovou krychli a podobně.

Pokud chceme ale mít deset modrých a deset červených krychlí, je rozumnější vytvořit tzv. prefab variant, která umožňuje vytvořit varianty již hotových prefabs. Prefaby lze i zanořovat do sebe. Příkladem by byla postava nepřítele. Máme ji vytvořenou jako prefab, protože nepřátel bude více, a máme prefab různých zbraní. Ve scéně potom můžeme postavě nepřítele přiřadit libovolnou zbraň. Jinému nepříteli můžeme dát zase jinou a tak dál.

1.2.8 Assets a AssetsBundles

Asset označuje jakoukoli položku použitou v projektu pro vytvoření hry [2]. Může se jednat o vizuální či zvukové prvky, jako jsou 3D modely, textury, sprite, zvukové efekty, skladby nebo i více abstraktní prvky jako jsou barevné gradienty, animace, texty nebo data. Assety mohou pocházet jak ze samotného Unity, například scény použité ve hře, prefaby a scripty, tak i ze souborů vytvořených mimo Unity, nejčastějším příkladem jsou 3D modely, obrázky a hudba.

AssetsBundles může označovat dvě různé věci. V prvním případě označuje AssetBundle jakýsi archiv, ve kterém jsou uloženy všechny Assety použité ve hře. Ve druhém případě označuje AssetBundle objekt, pomocí kterého lze skrze kód načítat Asset z konkrétního AssetBundle.

1.2.9 Unity Asset Store

Unity Asset Store je obchod s různými assety. Jsou tam jak placené assety, tak i zcela zdarma. Tyto assety jsou poskytovány jak od Unity Technologies, tak i od uživatelů. Dají se tam najít postavy, animace, textury, zvuky, 3D modely a spousta dalších doplňků do her. V současnosti je asset store dostupný pouze na webu [5], ale stahovat a vkládat do hry je lze přímo v Unity Editoru přes Package Manager [2].

1.3 Alternativy k Unity

Kromě Unity Game Engine se nabízí celá řada komerčně dostupných variant. Každý z těchto herních enginů má řadu výhod, ale i nevýhod oproti Unity Game Engine. Za zmínku zcela jistě stojí Unreal Engine [6], ale i řada dalších, méně známých herních enginů.

1.3.1 Unreal Engine

Unreal Engine byl vytvořen společností Epic Games Inc. v roce 1998. Unreal Engine je přímým konkurentem Unity [6]. Oba jsou si velmi podobné, ale mají i své odlišnosti.

1.3.1.1 Skriptovací jazyk

Na rozdíl od Unity, který používá pro psaní skriptů jazyk C#, skripty do Unreal Enginu jsou napsány v jazyce C++ a samotný kód herního enginu kombinuje tzv. Blueprinty [6], to je vizuální skriptovací jazyk, který používají hry a produkty firmy Epic Games, a jazyk C++.

1.3.1.2 Grafika

Když přijde na realistické hry, má Unreal Engine nad Unity výhodu. Oproti Unity totiž nabízí vysoce kvalitní grafiku [6]. To neznamená, že v Unity nejde dosáhnout realistické grafiky, pouze je to složitější, časově náročnější, a ne vždy lze dosáhnout stejně kvalitní grafiky jako v Unreal Engine. I co se týká vykreslování, tak Unreal Engine nabízí rychlejší vykreslování než Unity. Unreal Engine také nabízí více efektů a posouvá tak hru ještě dál.

1.3.1.3 AI

Unreal Engine má již v sobě obsažený systém pro tvorbu herních AI, jedná se konkrétně o Behavior Tree [7], který umožňuje vytvářet poměrně komplexního chování. Unity má podporu pro vytváření AI [2], ale rozhodně není tak daleko jako Unreal Engine a uživatel musí většinu práce odvést sám.

1.3.1.4 Podpora pro 2D

Unity je považováno za lepší volbu v případě, že chceme vytvářet 2D hru. Unreal Engine nabízí některé pluginy, která mají tvorbu 2D her usnadnit, například Paper2D [7], ale obecně se spíše zaměřuje na tvorbu 3D her.

1.3.1.5 Unreal Marketplace

Unreal Marketplace [8] je mladší než Unity Asset Store a nedisponuje tak velkou kvantitou assetů jako Unity. Ovšem nedostatek kvantity Unreal Marketplace kompenzuje jejich kvalitou, obzvlášť když se jedná o přírodní materiály a modely, jako tráva, kameny a podobně. Dalším rozdílem je poměrně vysoké zastoupení zdarma dostupných assetů na Unreal Marketplace.

1.3.1.6 Cena a licence

Stejně jako Unity, tak i Unreal Engine je zdarma. Unreal dále nabízí dvě licence k zakoupení. Jednu s fixní cenou a druhou bez předem stanovené ceny, která je dělána na míru kupujícímu [9]. Hlavní rozdíl mezi licencemi v Unity a Unreal Engine je, že Unity vyžaduje zakoupení licence v případě, že projekt vydělá více než 100 000 dolarů za rok, kdežto Unreal Engine žádnou licenci nevyžaduje.

Vyžaduje pouze 5% podíl ze zisků v případě, že projekt, který ke spuštění přímo vyžaduje technologii Unreal Engine, přesáhne výdělek 1 000 000 dolarů. Tedy prvních milion dolarů je zdarma, potom se teprve ze zisků odvádí společnosti Epic Games Inc. 5 %.

1.3.1.7 Přístupnost

Tady měl Unreal Engine výhodu díky Blueprintům, které umožňují programovat bez znalosti programovacího jazyku. Jde pomocí nich vytvořit prakticky celou hru [9]. Značnou nevýhodou je, že jsou oproti C++ pomalejší a nejsou příliš vhodné pro velké a komplexní hry. V Unity dlouho nic takového nebylo, ale v roce 2018 přidali technologii Bolt, která se podobá Blueprintům. Oba mají své výhody i nevýhody, ale oba vyžadují základní znalosti programování, jako například co jsou to proměnné, for-loop a tak dále.

Když přijde na jazyk samotný, C# je daleko snadnější se naučit oproti C++, který používá Unreal Engine. Unity má také větší komunitu a množství návodů na internetu, takže je považován za přívětivější pro nováčky.

1.3.1.8 Zdrojový kód

Unreal Engine je „Source available“ [9], umožňuje tak do jisté míry upravit základní kód frameworku. Naproti tomu Unity toto umožňuje pouze v případě zakoupení licence a dalšího dodatečného příplatku [10].

1.3.1.9 Známé projekty

Jednou u nejznámějších her za posledních pár let vyvinutá v Unreal Engine se stala hra Fortnite vydaná přímo společností Epic Games Inc. Dobrou ukázkou skvělých grafických dovedností Unreal Engine je potom hra Hellblade: Senua’s Sacrifice vyvinuta studiem Ninja Theory, která nabízí naprosto nádhernou grafiku s neuvěřitelnými detaily.

1.3.1.10 Shrnutí

Unity je zpravidla lepší volbou pro začínající vývojáře [6], kteří mají zájem vyvíjet v jednoduchém prostředí a rozsáhlou komunitou, která nabízí spoustu nástrojů, modelů a dalších assetů, jež lze stáhnout nebo zakoupit na Unity Asset Store. Unreal nabízí propracovanější a kvalitnější grafické zpracování, které i přesto, že je náročnější na výpočetní výkon, vypadá zpravidla lépe než podobně zpracovaná hra v Unity. Unreal je tudíž zpravidla lepší volbou pro zkušenější vývojáře [6], pracující na velkých, fotorealistických 3D hrách.

2 UMĚLÁ INTELIGENCE

Umělá inteligence, známá převážně pod anglickou zkratkou AI (Artificial Intelligence), je poměrně široký pojem. Zatím za mě nejlepší definici podal John McCarthy v jednom svém článku [11], kde popsal umělou inteligenci jako vědu a techniku vytváření inteligentních strojů a zejména inteligentních počítačových programů, přičemž se umělá inteligence nemusí omezovat na biologicky pozorovatelné metody čili na metody typicky používané lidmi.

2.1 Turingův test

Alan Turing ve své práci z roku 1950 vyslovil otázku, zda dokážou stroje myslet [12]. Aby tuto otázku zodpověděl, přišel s testem, dnes známým jako Turingův test [13], jak inteligenci stroje prokázat či vyvrátit. Test se skládá se dvou lidí a jednoho počítače, kde má člověk rozhodnout, kdo je počítač a kdo je člověk. Ač je test v provedení jednoduchý, ještě žádná umělá inteligence jím neprošla.

Z Turingova testu vyplývá, že umělá inteligence se snaží replikovat lidské myšlenkové procesy uvnitř stroje. Ještě v minulém století toto byl běžný přístup, ale v posledních letech se od toho ustoupilo a byl přijat více praktický přístup, kdy jde spíše o to maximalizovat výhody stroje oproti člověku, především schopnost stroje zpracovat velké objemy dat v relativně krátkém čase a racionálně uvažovat i jednat.

2.1.1 Argument čínského pokoje

Na nedostatky Turingova testu poukazuje tzv. Argument čínského pokoje, který byl představen Johnem Searlem v článku z roku 1980 [14]. Jedná se pouze o myšlenkový experiment, který předkládá otázku, zda stroj, který dokáže následovat instrukce a vyhledávat v datech je skutečně inteligentní nebo se jenom jeví jako inteligentní.

2.2 Uplatnění umělé inteligence

Díky své schopnosti analyzovat velké objemy dat našla umělá inteligence široké uplatnění v moderním světě. Příkladem jsou samořídící auta, spamové filtry, cílená reklama, různí asistenti jako třeba Siri od společnosti Apple a další. V posledních letech můžeme sledovat její stále se rozšiřující uplatnění i v medicíně [15], kde pomáhá s diagnostikou pacientů. Nesmíme opomenout její využití v počítačových hrách, které je dnes již běžné.

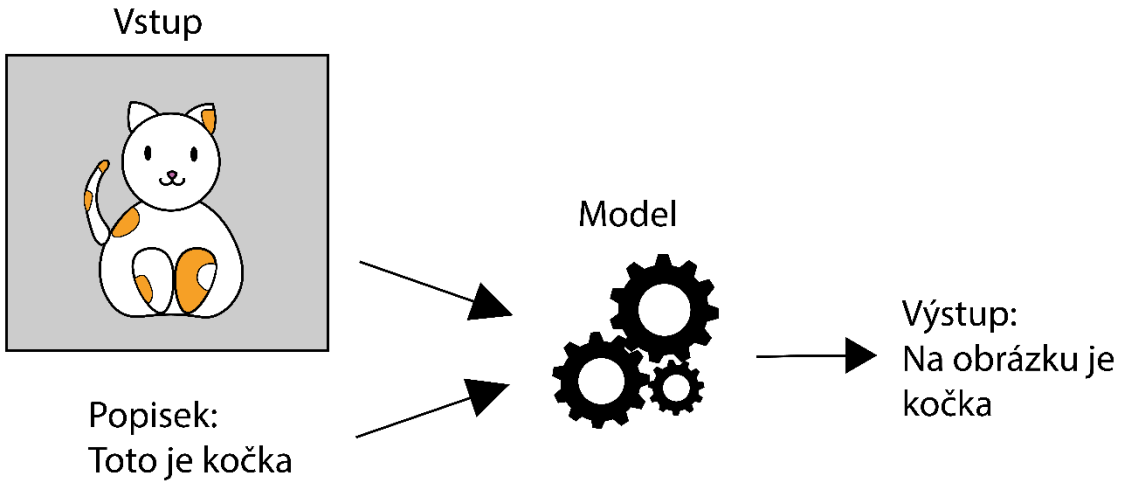
2.3 Učení

Schopnost se učit je jedním s klíčových požadavků pro inteligentní chování [16]. Proto se obor umělé inteligence zabývá technikami učení. Existuje více různých přístupů, jak učit umělou inteligenci. Přístup k učení se vybírá podle toho, co od umělé inteligence požadujeme.

Strojové učení je podkategorie umělé inteligence, která využívá algoritmy, statistické modely a předešlé zkušenosti k vylepšení sebe sama a svého výkonu v určitých úkolech [17]. Systém je trénován na velkém objemu dat, dokud jeho přesnost nedosáhne požadované hodnoty. Strojové učení se dá rozdělit do třech základních typů: učení s učitelem, učení bez učitele a posilované učení.

2.3.1 Učení s učitelem

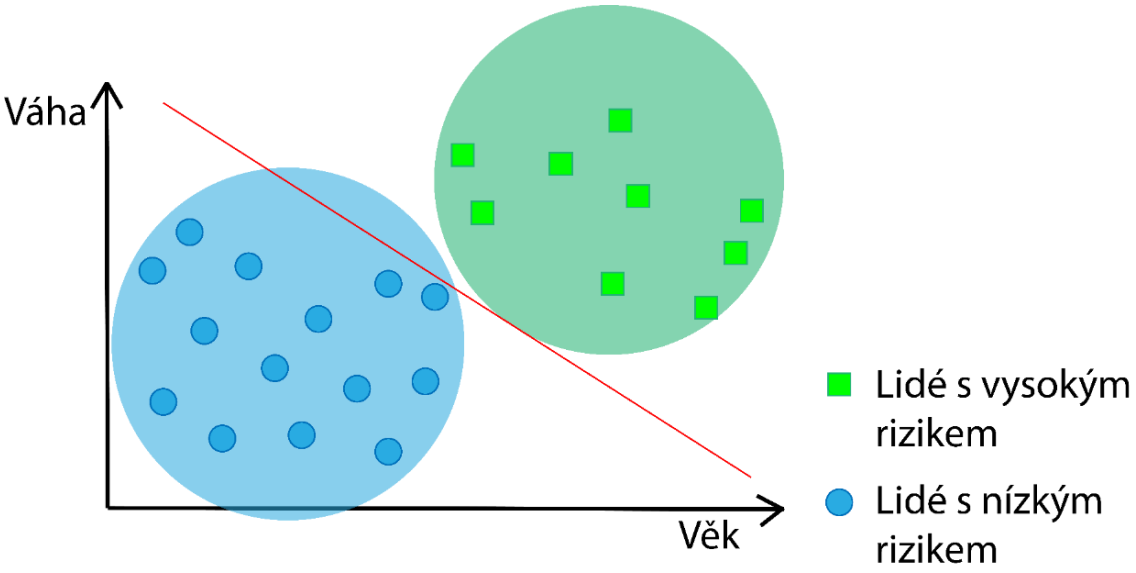
Metoda učení s učitelem využívá tréninkové sety, kde jsou data označena správným výstupem. Například obrázek s kočkou je označen jako ‚kočka' a obrázek s psem jako ‚pes'. Jakmile se model všechny obrázky naučí, je mu předložen nový obrázek, který nebyl součástí tréninkového setu. V této fázi se měří přesnost modelu. Pokud bylo dosaženo požadované přesnosti je trénink ukončen.



Obrázek 1. Učení s učitelem

2.3.2 Učení bez učitele

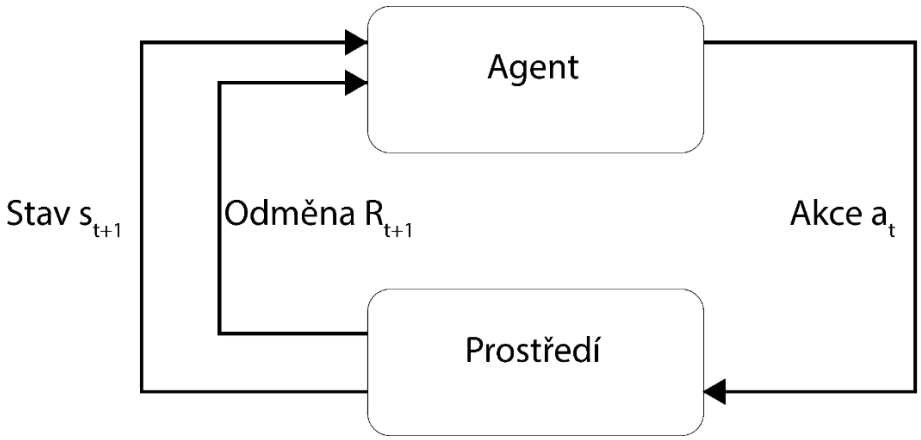
Metoda učení bez učitele je typ strojového učení, kde jsou modelu poskytnuta neoznačená data. Model se potom snaží hledat vzorce, struktury a závislosti mezi daty [17]. Tento typ učení je vhodný, pokud nemáme specifický atribut, který v datech hledáme. Skvělým příkladem je stanovení pravděpodobnosti cukrovky u lidí v knize od Johna D. Kellehera a Brendan Tierney s názvem Data Science [18]. Cukrovku způsobuje celá řada faktorů a nelze přesně určit kdo ji dostane a kdo ne. Model se potom dívá na data, která jsou blízko sebe a vytváří shluky dat, tzv. clusters. Je zde nutno podotknout, že je velice důležité vybrat vhodná data, které modelu poskytneme. Například můžeme zahrnout do dat o lidech s cukrovku, že 60 % z nich vlastní televizi značky Samsung. S cukrovkou to nijak nesouvisí, ale to umělá inteligence nemůže vědět, a tudíž to použije při tvoření svých závěrů a predikcí.



Obrázek 2. Shluky dat

2.3.4 Posilované učení

Posilované učení funguje na principu odměn. Model se snaží provádět takové akce, které mu přinesou co největší zisk a vyhýbat se akcím, které by mu mohly uškodit. Model se pomocí akcí učí, které jsou nejvýhodnější, a které naopak výhodné nejsou, a to nejen okamžitě, ale i v průběhu času. Takže akce, která se v daném okamžiku zdá nevýhodná, se může později ukázat jako výhodná. Stejně to platí i naopak. Posilované učení toto do jisté míry bere v potaz. Rozhodnout o tom, která akce nejvíce přispěla k dobrému, či špatnému výsledku bývá často obtížné. Příkladem toho je třeba když se vám ráno udělá špatně z jídla, které jste měli včera. Určitě jste den před tím měli více než jen jedno jídlo, je tudíž obtížné říct, které přesně bylo příčinou vaší nevolnosti. Tento problém se nazývá problém přiřazení dočasného kreditu [19], známý spíše pod anglickým termínem „temporal credit assignment problem“.



Obrázek 3. Posilované učení. Agent používá současný stav prostředí pro generování své další akce a_t. Ta následně změní stav prostředí na s_{t+1} a vyprodukuje odměnu R_{t+1}.

2.5 Prohledávání mapy a hledání řešení v prostoru

Prohledávání mapy a hledání řešení v prostoru jsou klíčové úlohy v oblasti umělé inteligence a robotiky. Analýza existujících přístupů poskytne ucelený pohled na problematiku prohledávání mapy a hledání řešení.

2.5.1 Historie

První výzkumy prohledávání mapy a hledání řešení sahají až do 50. let 20. století. Předchůdcem moderního výzkumu byla práce Johna McCarthyho, který vytvořil pojem umělé inteligence a zavedl myšlenku použití algoritmů pro řešení složitých problémů v oblasti mapování a prohledávání prostoru. Motivací pro tyto studie byla snaha vytvořit autonomní agenty, schopné efektivně prohledávat neznámá prostředí a nalézt optimální řešení.

2.5.2 Prohledávání grafu

Prohledávání grafu je jedním z klíčových přístupů k prohledávání mapy. Algoritmy jako prohledávání do šířky (Breadth-first search), prohledávání do hloubky (Depth-first search) a algoritmus A* jsou široce používané metody při hledání nejkratších cest, optimálních tras nebo plánování pohybu v prostoru [20]. Tyto algoritmy využívají různé strategie pro prohledávání grafu a jsou užitečnými nástroji pro hledání řešení v mapě.

2.5.3 Heuristické metody

Pro zlepšení efektivity prohledávání a hledání řešení se používají heuristické metody. Tyto metody využívají odhadů a aproximací pro rychlejší určení směru pro další kroky. Příkladem je algoritmus A*, který kombinuje informace o vzdálenostech a heuristiky k nalezení nejkratší cesty [20]. Další pokročilé heuristické techniky zahrnují genetické algoritmy, simulované žíhání a optimalizační algoritmy založené na mravenčích koloniích.

2.5.4 Stochastické metody

Prohledávání mapy a hledání řešení často vyžadují zohlednění nejistoty a neznámých faktorů. Stochastické metody, jako je Markovův rozhodovací proces a Monte Carlo Tree Search, umožňují modelování a rozhodování v prostředí s nedeterministickým chováním [21]. Tyto metody se uplatňují při navigaci robotů, strategickém plánování a hledání řešení v dynamických prostředích.

2.5.5 Uplatnění

Prohledávání mapy a hledání řešení v prostoru mají široké uplatnění v různých oblastech. V robotice se používají pro autonomní pohyb robotů, mapování neznámých prostředí a plánování cest. V herních technologiích se využívají pro tvorbu umělé inteligence pro strategické hry a hledání nejlepších tahů. Další uplatnění zahrnují logistiku, optimalizaci trasy, navigaci v GPS systémech a analýzu dat.

2.6 Umělá inteligence pro rozhodovací procesy

Umělá inteligence přinesla revoluci do mnoha oblastí a ani oblast rozhodování to neminulo. Díky schopnostem umělé inteligence zpracovávat velké objemy dat je schopná vytvářet přesnější předpovědi a cílenější doporučení. To dělá umělou inteligenci žádaným nástrojem pro rozhodování v různých odvětvích. Rozhodovací systémy založené na umělé inteligenci mohou organizacím pomoci přijímat rychlejší a přesnější rozhodnutí, zlepšit alokaci zdrojů a optimalizovat výsledky. Kromě využití v podnikání, zdravotnictví a státní správě si umělá inteligence našla cestu i do herního průmyslu.

Rozhodovací systémy založené na umělé inteligenci se staly důležitou součástí moderních videoher a umožňují vývojářům vytvářet poutavější a náročnější herní zážitky. Herní roboti pohánění umělou inteligencí mohou v reálném čase analyzovat herní data a podle nich upravovat své chování, aby hráčům poskytli dynamičtější a poutavější zážitek. Systémy umělé inteligence lze například využít k vytváření realistických a nepředvídatelných protivníků ve hrách pro více hráčů, což zlepšuje celkový herní zážitek. Algoritmy AI navíc mohou analyzovat chování a preference uživatelů, což umožňuje vývojářům her přizpůsobit herní zážitek každému hráči. S rostoucí popularitou online her se očekává, že rozhodovací systémy založené na umělé inteligenci budou hrát stále důležitější roli při utváření budoucnosti herního průmyslu [22].

2.6.1 Markovovy rozhodovací procesy

Markovovy rozhodovací procesy jsou matematickým rámcem, který se používá k modelování rozhodovacích problémů v situacích s nejistým výsledkem. Jsou široce používány v různých oblastech, včetně ekonomie, robotiky a umělé inteligence.

2.6.1.1 Princip

Markovovy rozhodovací procesy sestávají z množiny stavů, akcí, ceny přechodů mezi stavy a pravděpodobností pro každý přechod z jednoho stavu do druhého [23]. Právě element pravděpodobnosti dělá z Markovových rozhodovacích procesů stochastickou metodu. Proto je potřeba mít nějakou rozhodovací politiku. Většinou se jedná o tabulku, která algoritmu říká, jakou akci vybrat na základě stavu, ve kterém se právě nachází [21]. Při modelování problému pro průchod grafem se nejčastěji pro rozhodování používá politika, která upřednostňuje volby s nejmenší průměrnou ztrátou. Politiku lze rozšířit i o podmínky, které když jsou splněny, tak algoritmus vybere jinou akci.

2.6.1.2 Řešení

Řešení Markovových rozhodovacích procesů zahrnuje nalezení optimální politiky nebo hodnotové funkce. Hodnotová funkce přiřazuje každému stavu nebo dvojici stav-akce hodnotu, která představuje očekávanou kumulativní odměnu, kterou lze z daného stavu nebo dvojice stav-akce získat při dané politice. Optimální politika je taková, která maximalizuje hodnotovou funkci nebo minimalizuje ztrátu.

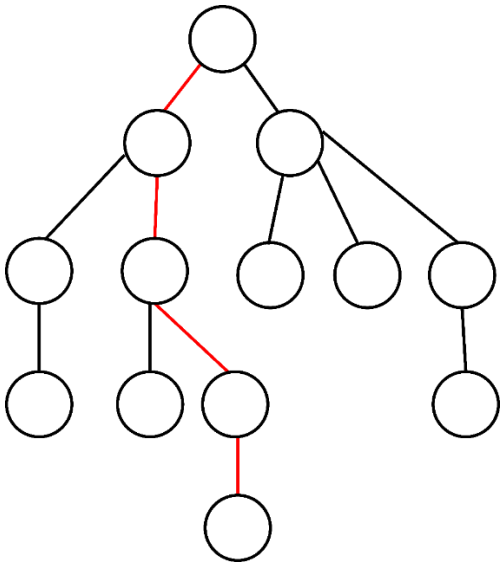
K jejich řešení lze použít různé algoritmy, například dynamické programování, metody Monte Carlo a techniky učení s posilováním. Tyto metody umožňují agentovi iterativně se učit a zlepšovat svou politiku na základě interakcí s prostředím.

2.6.2 Monte Carlo Tree Search

Monte Carlo Tree Search je algoritmus, který používá kombinaci hledání, simulací a statické analýzy k prohledávání stromových struktur v komplexním a nestálém prostředí. Jeho hlavním účelem je učinit co nejvhodnější rozhodnutí v situacích, kdy existuje spousta možných rozhodnutí a výsledky každého z nich jsou nejisté.

Monte Carlo Tree Search vytváří stromovou strukturu tak, že si opakovaně vybírá jeden uzel, od kterého potom provádí simulace náhodných her. Toto provádí, dokud nedosáhne požadovaného počtu simulací. Potom aktualizuje hodnoty daného uzlu na základě předchozích simulací. To samé provede pro další uzly, dokud je všechny neprojde. Jakmile jsou všechny simulace dokončeny, algoritmus se rozhodne, kterou akci provede, respektive, do kterého uzlu se vydá. Rozhodnutí závisí na rozhodovací politice. Ve hře to může být uzel s největším skoré, při plánování přepravy třeba nejméně strávených hodin na cestě. V dalším uzlu znovu

projde všechny z něj vedoucí uzly a takhle pokračuje, dokud mu nedojdou uzly nebo dokud nedojde na konec hry [24].



Obrázek 4. Ukázka fungování Monte Carlo Tree Search algoritmu

Díky všem těmto svým vlastnostem je schopná hrát i komplexní hry jako je třeba Go, šachy nebo 2048. Hry však nejsou jeho jediným využitím. Mezi další využití patří například právě plánování v reálném světě či optimalizace.

Monte Carlo Tree Search má čtyři základní fáze: Selection, Expansion, Play-out, Backpropagation [25].

2.6.2.1 Selection

V této fázi algoritmus sestupuje po dosud sestaveném stromu, dokud nedojde k uzlu, který vyžaduje rozšíření. Sestup probíhá podle předem určené strategie.

2.6.2.2 Expansion

V této fázi algoritmus přidá k dosavadnímu stromu jeden nebo více uzlů. Běžný přístup je přidání jednoho uzlu v každé iteraci.

2.6.2.3 Play-out

Tato fáze začíná v nově přidaném uzlu. Algoritmus zde začíná hrát hru až do jejího konce nebo do předem určeného počtu tahů. Strategie pro vybírání tahů je více. Mezi nejčastější patří Monte Carlo, kdy si algoritmus vybírá tahy náhodně.

2.6.2.4 Backpropagation

Výsledek simulace se zpětně šíří přes vybraný uzel a uzly jemu předcházejícími, aby se aktualizovaly jejich statistiky, např. počet návštěv a celková odměna. Podle toho se zhodnotí jejich potenciál jako dobrých tahů v budoucnosti. Algoritmus tyto kroky opakuje, dokud není splněna podmínka pro zastavení, například dosažení časového limitu nebo maximálního počtu iterací.

2.6.3 Monte Carlo

Monte Carlo jsou statistické metody, které využívají náhodné vzorkování za účelem odhadnout výsledek. Nezřídka kdy se využívají v situacích, kde jsou analytické metody příliš složité nebo nejsou dostupné. Mezi nejznámější použití patří fyzikální simulace. Například simulace pohybu částic v plynu [26], interakce mezi částicemi a další. Z tohoto důvody se dá také využít i při tvorbě více realistických her a herních prostředí. Další využití našly tyto metody v oblasti financí pro vytváření ekonomických předpovědí, dokonce se dají využít i pro předpověď počasí.

V rámci umělé inteligence se tyto metody často kombinují s algoritmy strojového učení. Monte Carlo Tree Search je právě jednou z metod, která využívá Monte Carlo.

2.7 Hluboké učení na grafech

Hluboké učení na grafech označuje aplikaci technik hlubokého učení na graficky strukturovaná data. Grafy jsou matematické struktury sestávající z uzlů, někdy nazývaných také jako vrcholy, uzly jsou spojené hranami, které mohou představovat vztahy nebo interakce mezi entitami.

Hluboké učení na grafech zahrnuje vývoj specializovaných architektur neuronových sítí, známých jako grafové neuronové sítě, které mohou efektivně zpracovávat a analyzovat data s grafovou strukturou. Grafové neuronové sítě fungují na základě šíření informací napříč uzly a hranami grafu, což umožňuje každému uzlu shromažďovat a aktualizovat informace od svých sousedů [27].

Hluboké učení na grafech si získalo značnou pozornost a bylo úspěšně aplikováno v různých oblastech, mimo jiné v analýze sociálních sítí, doporučovacích systémech, molekulární chemii, sítích interakcí mezi proteiny, předpovídání dopravy a hraní her.

2.7.1 Použití ve hrách

Hluboké učení na grafech může být cenným přístupem k nalezení nejlepší cesty ve hře z několika důvodů.

2.7.1.1 Reprezentace grafů

Hry často zahrnují prostorové vztahy a interakce mezi různými entitami, které lze efektivně reprezentovat jako grafy. Hluboké učení na grafech umožňuje modelovat a analyzovat komplexní vztahy mezi uzly a hranami v grafu [27], které tak pomáhají lépe zachytit dynamiku herního prostředí.

2.7.1.2 Nelineární vztahy

Modely hlubokého učení, jako jsou neuronové sítě na grafech, mohou zachytit nelineární vztahy mezi entitami ve hře. To je užitečné zejména v případech, kdy určení nejlepší cesty zahrnuje zohlednění složitých interakcí a závislostí mezi různými herními prvky.

2.7.1.3 Implicitní vztahy

Hluboké učení na grafech dokáže extrahovat smysluplné rysy ze struktury grafu a atributů spojených s uzly a hranami. Tyto rysy mohou zachytit důležité vlastnosti herního prostředí, jako je vzdálenost, propojení, překážky nebo odměny, které jsou zásadní pro určení nejlepší cesty.

2.7.1.4 Posilované učení

Hluboké učení na grafech se dá použít v kombinaci s posilovaným učením a pomoci ke zlepšení umělé inteligence.

Využitím hlubokého učení na grafech mohou vývojáři her vytvářet inteligentní agenty schopné najít nejlepší cestu v dynamickém a komplexním herním prostředí. Tito agenti se mohou učit ze zkušeností, přizpůsobovat se různým scénářům a rozhodovat se na základě základní struktury grafů a vztahů.

3 TEXTOVÉ HRY (GAMEBOOKY)

3.1 Co je to Gamebook

Gamebook je kniha, ve které čtenář má možnost ovlivňovat děj na základě svých rozhodnutí. Často se stává, že čtenář sám je hlavním hrdinou. Jedná se tedy v podstatě více o textovou hru nežli o knihu. Jakmile čtenář dojde na stránku, kde si má vybrat odpověď, je vždy u všech možností označena stránka, na které příběh pokračuje pro danou volbu. Některé gamebooky mají i tzv. adventure sheet. Adventure sheet typicky obsahuje pole pro tři základní statusy hráče: schopnost, výdrž a štěstí. Dále mívají něco jako inventář, kde si čtenář může zaznamenávat, co všechno jeho postava má k dispozici. Statusy hráče se většinou rozhodují na začátku hry, a to pomocí hodu kostkami. Většina gamebooků v takovém případě má vlastní instrukce na to, jak tyto staty přesně spočítat.

3.2 Historie

Za úplný základ a vůbec první jakousi předběžnou verzi gamebooku se považuje kniha Examen de la obra de Herbert Quain od Jorge Luise Borgese vydaná roku 1941 [28]. Kniha má tři části a celkem devět možných konců. Zde ještě nebyl použit koncept gamebooku jako takový. Autorovi nešlo o interaktivity čtenáře a knihy, ale spíše o to vyprovokovat čtenáře k zamyšlení a zpochybnit jeho předpoklady o literatuře, jelikož poslední část knihy mění kontext první části.

Od té doby proběhly nějaké pokusy o gamebooky, ale bylo to až doby kolem roku 1980, kdy se gamebooky staly populárními. Prvním velkým průkopníkem byla série Fighting Fantasy [29] vydána v roce 1982. Tato série nejenže přímo vyžadovala po hráči aktivně činit rozhodnutí, ale i hody kostkami pro určení výsledků soubojů, a dokonce mít i přehled o věcech, které si čtenář zakoupil nebo získal během četby. Tato podoba gamebooků se stala asi nejznámější.

Před touto sérií vznikly i jiné známé tituly, jedním z velmi populárních byla i série Choose your own adventure, která vyšla v roce 1979.

Postupem času a s rostoucí popularitou a dostupností počítačů obliba gamebooků klesla a namísto nich přišli na scénu počítačové hry, které nabízely daleko větší interaktivitu. Gamebooky se tak těšily slávě pouze zhruba 20 let.

3.3 Současnost

V současnosti se gamebooky moc velké oblibě netěší, přesto občas někdo nějaký vydá. Také existují online komunity, které udržují staré tituly a přispívají svými vlastními gamebooky. Ovšem na znovu vzestup gamebooků to nevypadá.

3.4 Existující digitalizované Gamebooky

3.4.1 Choice of Games

Choice of Games [30] je firma, která se zaměřuje na vytváření textových her. Jde o jednoduchý design, kdy se na obrazovce zobrazuje vypsaný text a hráč si může vybrat z několik možností pod textem. Nejsou zde přítomny žádné obrázky ani interakce se hrou kromě klikání odpovědí. Jejich hry jsou spíše příběhy s různými konci. V jejich placených hrách jsou občas přítomny i obrázky, ale není jich moc.

3.4.2 Fighting Fantasy Classics

Fighting Fanatsy Classics je hra založená na několika gameboocích z řady Fighting Fantasy. Vydalo ji studio Tin Man Games v roce 2018. Tato hra je už daleko více grafická a velice se podobá klasickým gamebookům. Design má jako opravdová kniha, s očíslovanými stránkami, občasnými obrázky a má i adventure sheet.

3.4.3 The Warlock of Firetop Mountain

The Warlock of Firetop Mountain je další z adaptací gamebooku z řady Fighting Fantasy, konkrétně se jedná o titul stejného jména, jaké má hra, tedy o The Warlock of Firetop Mountain. Tato hra byla vydaná studiem Tin Man Games v roce 2016. Jedná se o 3D hru, kde si hráč na začátku vybere, za koho bude hrát a potom prochází skrze dungeon, dělá rozhodnutí a sleduje jak se odehrávají souboje a vyvíjí se děj. Stále se drží klasického pojetí gamebooku, tedy spousta textu ke čtení, s tím, že přidává element 3D grafiky a hudby.

3.4.4 Jsou moderní RPG hry digitalizovanými gamebooky?

Částečně. Sem tam se objeví hra, která adaptuje myšlenku několika konců v závislosti na rozhodnutích hráče. Některé hry na to jdou jednoduše podle toho, kterou odpověď hráč v dialogu zvolí, za příklad zde může sloužit Zaklínač 3. Jiné jsou méně nápadné a sledují hráčovi činy a styl jeho hry, tento styl adaptovala například hra Dishonered. Existují i takové, které tyto dva komponenty míchají dohromady. Tady může posloužit za příklad hra Deus Ex:

Human Revolution se čtyřmi konci, které přímo závisí na hráčově volbě a třemi variantami těchto čtyř konců, které jsou závislé na tom, jak hráč hrál hru. Celkem tak existuje dvanáct různých konců.

3.4.5 Shrnutí

Ač existují různé variace digitalizovaných gamebooků, jedno mají společné. Většina z nich cílí na hráče kolem 15 let a výš. Staré klasické zpracování gamebooků jaké nabízí Fighting Fantasy Classics nebo The Warlock of Firetop Mountain mladší hráče nepřiláká a novodobé hry jsou natolik odlišné od gamebooků, že ani ti, kdo gamebooky znají, si to nutně nemusí spojit. Termín gamebook je navíc pro valnou většinu lidí mladších 15 let naprosto neznámý. Další důležitá věc, kterou je nutno zvážit, je fakt, že většina těchto her, i samotných gamebooků, existuje pouze v angličtině a jen málokteré plně podporují češtinu. Většina hráčů, kteří neumí plynule anglicky, v takovém případě ani text nečte, pouze se dívá na značky, a plní úkoly podle toho, kam je zavedou. Pokud se zaměříme pouze na české hráče, je jednoduché vidět, proč nejsou gamebooky příliš v oblibě u mladších generací.

II. PRAKTICKÁ ČÁST

4 POŽADAVKY NA HRU

Před zahájením programování bývá zvykem sepsat si požadavky na finální produkt, v tomto případě na hru.

4.1 Funkční požadavky

Funkční požadavky představují funkcionality, které by hra měla nabídnout uživateli.

4.1.1 Menu

Hra by měla disponovat základní navigací skrze hru, aby hráč mohl hru ukončit nebo začít novou hru. Ovládání by mělo být jednoduše pochopitelné.

4.1.2 Pohyb hráče po mapě

Chceme, aby se hráč mohl po mapě pohybovat a odhalovat tak dialogy schované na mapě. Zároveň ovšem nechceme, aby mohl odejít z mapy. Je tedy nutné jeho pohyb nějak omezit, takže se hráči jeho postava neztratí z dohledu. Ovládání musí být dostatečně jednoduché, aby ho pochopily i mladší děti a lidé, kteří moc často hry nehrají anebo ani nikdy předtím žádnou hru nehráli.

4.1.3 Počítání skoré

Chceme, aby hra hlídala pro hráče jeho dosavadní postup. V případě této hry, kolik má zlata a stav jeho meče. Taky chceme, aby se hráči tyto informace zobrazovali kdykoliv, když se jejich hodnoty změní.

4.1.4 Dialog

Další mechanikou, kterou musíme implementovat je zobrazování dialogových oken. Budeme chtít, aby se zobrazily, jakmile hráč dojde k nějaké postavě. Tam dostane na výběr ze dvou možností, hráč si vybere jednu z nich a bude moct pokračovat dál. V některých scénách na výběr mít nebude, v takovém případě to však neovlivní jeho skóre.

4.2 Nefunkční požadavky

Nefunkční požadavky představují vlastnosti hry, které jsou důležité pro její správný chod a pro jednoduché využívání uživateli.

4.2.1 Náročnost na hardware

Hra by měla běžet i na méně výkonných zařízeních. Zároveň však chceme, aby vypadla dobře i na výkonnějších strojích. Proto bude existovat ve dvou verzích, které berou ohled na rozlišení monitoru.

4.2.2 Intuitivnost

Hra se zaměřuje především na mladších hráče, takže musí být jednoduchá na pochopení i ovládání. Příběh zároveň nesmí být příliš složitý ani moc dlouhý, aby je zdlouhavé čtení textu neodradilo.

4.2.3 Zábavnost

Jde nám hlavně o to, aby byla hra zábavná a vtáhla svým příběhem a zpracováním hráče do světa fantazie. Jedná se především o snahu nalákat hráče na ideu gamebooků a zbavit je představ, že všechny knihy jsou nutně nudné knihy. Hra proto musí upoutat.

4.2.4 Dostupnost

Hra by měla být zdarma, aby přilákala co nejvíce nových hráčů. Hra bude dlouhodobě dostupná především v knihovně na Obecním úřadu Halenkovice, jako jeden ze zdrojů zábavy pro tamější čtenáře a školní družinu.

5 ASSETY

Většina assetů byla stáhnuta z internetu nebo ze stránek Unity Asset Store. Některé byly dále upraveny, jiné jsou zachovány v jejich původní podobě.

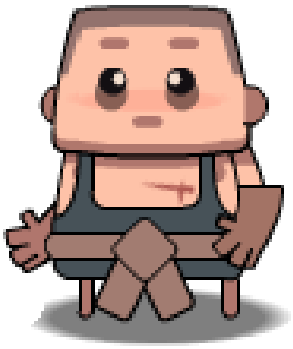
5.1 Grafika

Vzhledem k tomu že se jedná o 2D hru, nebylo potřeba žádných 3D modelů a na většinu grafiky postačily obyčejné obrázky.

5.1.1 Hlavní hrdina

Postava hlavního hrdiny pochází z balíčku Tasty Characters – Castle Pack od Severin Baclet. Balíček obsahuje jak hotové postavy, tak i jejich jednotlivé části, ze kterých je možné vytvořit si vlastní postavu. Pro účely hry postačila již předem vytvořená postava rytíře. Tento balíček byl vybrán z toho důvody, že postavy vypadají roztomile a neškodně a jsou proto vhodné i pro děti.

Postava hrdiny na konci hry při splnění určitých podmínek změní vzhled. Pro tento model byl využit balíček Tasty Characters – Village Pack od stejného tvůrce. Tato postava byla vytvořena spojením několika částí postav dohromady.



Obrázek 5. Modifikovaná verze hlavního hrdiny

5.1.2 Strážce dungeonu

Většina úrovní obsahuje postavu Strážce dungeonu. Tato postava je pouze obyčejný 2D Sprite s obrázkem sochy. Obrázek byl převzat od tvůrce, který vystupuje pod jménem Upklyak [https://www.freepik.com/author/upklyak]. Socha byla barevně upravena, aby lépe zapadla do prostředí hry.

5.1.3 Vězeň

Postava vězně pochází z balíčku Tasty Characters – Village Pack od Severin Baclet. Tato sada obsahuje postavu vězně, takže žádné další úpravy provedeny nebyly.

5.1.4 Obchodník

Na jedné úrovni se vyskytuje postava obchodníka, u kterého si mohou hráči zakoupit jeden předmět. Jedná se o postavu kočky. Pochází od stejného umělce jako postava Strážce dungeonu, s tím rozdílem, že tato postava nebyla nijak upravena.

5.1.5 Golem

Golem se nachází na dvou úrovních hry. U této postavy došlo k několika úpravám v programu Adobe Illustrator. Tento program není k dispozici zadarmo, ale nabízí 7denní zkušební licenci, potom se musí k jeho využití platit měsíční poplatky. Původní postava Golema pochází od Upklyak.

Obrázek 6. Golem

5.1.6 Pozadí

Většina pozadí pro jednotlivé úrovně pochází od tvůrce Upklyak. Pokud byly upraveny, jednalo jsem jen o barevné schéma, případně nejsou viditelné celá. Pouze úroveň s vězením byla rozebrána na menší díly, aby se tam dala vložit postava vězně.

5.1.7 Dialogová okna

Pro dialogová okna byl využit balíček Fanatsy Wooden GUI: Free od Black Hammer. Sestává z několika různých designů oken a variant tlačítek. Nejčastěji je ve hře použit design dialogových oken, který připomíná pergamen. Změny byly provedeny pouze u tlačítek, aby měnily barvu při najetí myší na ně.

5.1.8 Text

Font byl stáhnut z internetové stránky, která nabízí fonty zdarma ke stažení. Ve hře byl použit font s názvem Immortal od tvůrce Apostrophic Labs. Autor fontu uvádí, že font je zcela zdarma.

5.2 Hudba a zvukové efekty

Všechny zvukové efekty pochází z internetové stránky, kde jsou zdarma k dispozici. Ve hře hraje obvykle skladba s názvem Borgar od umělce Alexander Nakarada. V případě, že hráč draka zabije, začne místo ní hrát skladba Skaga od stejného umělce. Většina úprav hudby se týkala pouze stříhání skladeb tak, aby se mohly spouštět ve smyčce a nebylo to pro hráče zpozorovatelné.

5.3 Import do Unity

Balíčky z Unity Asset Storu umožňuje Unity stáhnout a importovat přímo v Editoru, takže to jsem také udělala. Když přijde na obrázky a hudbu většinou není problém. Unity podporuje většinu formátů, pouze u audia doporučuje použít nezkomprimované typy jako je WAV a AIFF, protože Unity Editor provádí při importu vlastní kompresi. U obrázků je zajímavé, že podporuje dokonce typ PSD, což je nativní formát souboru používaný aplikací Adobe Photoshop. To se však nevztahuje na Adobe Illustrator, který k ukládání používá formát EPS. Tento formát už Unity Editor neotevře, takže se obrázky musí nejprve exportovat do jiného formátu, já jsem vybrala formát PNG.

6 PREFABS

Pro usnadnění práce jsem si jako první vytvořila objekty, které budu ve hře potřebovat nejčastěji a uložila jsem je jako prefab, takže je potom nemusím kopírovat ani vytvářet znovu pro každou scénu.

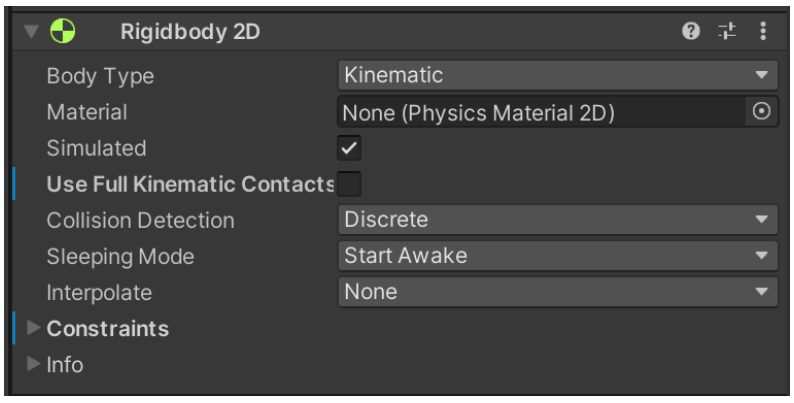
6.1 Hlavní hrdina

Začala jsem tím, že jsem do scény přidala postavu hlavního hrdiny. Nejprve musím přidat nový herní objekt typu sprite a tvaru čtverec. Objekt bude mít tak automaticky přidanou komponentu Sprite Renderer, která zajišťuje renderování objektu. Jednoduše potom objektu přidám do kolonky Sprite obrázek hlavního hrdiny a místo bílého čtverce se zobrazí vybraný obrázek.

Protože chci, aby se postava po mapě pohybovala, musí se jí přidat skript a důležitá komponenta RigidBody2D.

6.1.1 RigidBody2D

RigidBody2D je komponenta, která zajišťuje fyziku pro daný objekt. V Unity existují dvě varianty: RigidBody pro 3D objekty a RigidBody2D pro 2D objekty. Pokud bychom chtěli pro postavu hrdiny 3D vlastnosti, tak bychom mohli namísto RigidBody2D použít RigidBody. Moc velký rozdíl v nich není, ale použitím 2D varianty šetříme místo pro jiné výpočty a objekt se může pohybovat pouze po ose X a Y. V této hře 3D vlastnosti nepotřebuji, takže není důvod nepoužít RigidBody2D.



Obrázek 7. Komponenta RigidBody2D

6.1.1.1 BodyType

U Body Type pro RigidBody2D je na výběr ze tří možností: Dynamic, Kinematic a Static. Body Type má vliv na pohyb, rotaci a kolize objektu, objekt bude kolidovat ale jen v případě, že má komponentu Collider.

6.1.1.2 BodyType – Dynamic

Dynamický typ znamená, že na objekt bude působit gravitace a další síly. Je to nejčastější nastavení objektu, u kterého chceme, aby se hýbal. Protože interaguje se vším kolem, je to taky výkonově nejnáročnější typ.

6.1.1.3 BodyType – Kinematic

Tento typ, na rozdíl od dynamického typu, není ovlivněn gravitací a dalšími silami. Je nastaven tak, aby se pohyboval pouze při velmi specifických podmínkách, například když uživatel stiskne tlačítko. Díky tomu není tak náročný na výkon. Při kolizi se objekt chová, jako kdyby měl nekonečnou hmotnost. Dalším rozdílem je, že koliduje pouze s objekty, které mají dynamický BodyType. Jde však povolit možnost „Use Full Kinematic Contacts“, která umožňuje kolize se všemi typy.

6.1.1.4 BodyType – Static

Statický typ koliduje pouze s dynamickým a na rozdíl od kinematického objektu není určen k tomu, aby se pohyboval. Kolize dvou statických objektů tudíž nejsou podporovány.

V této hře není zapotřebí gravitace, ale zároveň chci, aby se hlavní hrdina pohyboval, takže jsem nastavila BodyType na Kinematic.

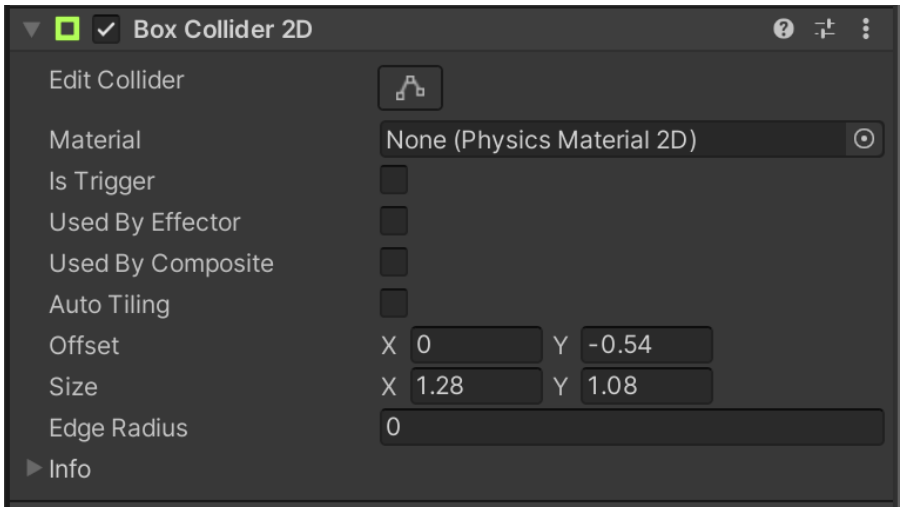
Protože potřebuji zobrazovat dialogy, jakmile postava hrdiny dojde k jiné postavě nebo na určité místo na mapě, přidala jsem komponentu Collider.

6.1.1.5 Simulated

Dalším důležitým nastavením u RigidBody2D je možnost Simulated. Ta určuje, zda bude objekt interagovat s 2D fyzikální simulací, kde simulace pohybuje komponentou RigidBody2D a ta zase pohybuje postavu hrdiny. Bez povolení možnosti Simulated by se náš hrdina nemohl pohybovat po mapě.

6.1.2 Collider2D

Collider2D definuje tvar objektu, který bude kolidovat s ostatními objekty. Collider není ve scéně viditelný a nemusí nutně mít přesně stejný tvar jaký má objekt. Collider je vázaný na RigidBody, proto nelze kombinovat Collider2D s RigidBody a naopak. Oba musejí být 2D nebo 3D. Já jsem zvolila RigidBody2D, takže i Collider musí být 2D. Typy pro Collider2D, které mohou být použity s RigidBody2D jsou: Circle, Box, Polygon, Edge, Capsule, Compoite Collider2D.



Obrázek 8. Komponenta BoxCollider2D

6.1.2.1 Box Collider2D

Protože potřebuji pouze detekovat kolize, postačí úplně obyčejný BoxCollider2D. Unity automaticky při přidání Collideru nastaví i jeho velikost vzhledem k velikost spritu, takže nejsou úpravy většinou nutné, ale kdykoliv se dají provést kliknutím v komponentě BoxCollider2D na tlačítko Edit Collider.

6.1.3 Skripty

Budu potřebovat skript na ovládání postavy a druhý na to, abych zabránila postavě odejít ze scény, respektive mimo kameru.

Standartně používá Unity pro otevírání skriptů Visual Studio, já použila Rider od JetBrains.

6.1.3.1 Skript pro pohyb postavy

Hned na začátku skriptu je atribut [RequireComponent(typeof(Rigidbody2D))]. To znamená, že objekt, kterému je skript přiřazen, musí mít komponentu RigidBody2D. Samotná třída dědí ze třídy MonoBehaviour, což je základní třída pro všechny Unity skripty, které interagují s herními objekty.

Členské proměnné s přísnějším modifikátorem přístupu než public, jako jsou private a protected, nelze spravovat prostřednictvím Unity Editoru, z tohoto důvodu dekorujeme členské proměnné atributem [SerializeField]. Tato serializace se provádí pomocí vnitřního serializačního systému Unity, nikoliv pomocí serializační funkce rozhraní .NET.

Dále musíme deklarovat proměnné použité ve skriptu. Tady je kromě standartních typů v C# přidaný typ RigidBody2D, což je odkaz na tu komponentu v herním objektu.

Unity volá funkci Start pouze jednou, volá ji při inicializaci skriptu a před jakoukoliv Update metodou, je tak ideálním místem pro všechny inicializace. V tomto skripty přiřazuje proměnné _rigidBody2D referenci na komponentu RigidBody2D pomocí funkce GetComponent. Je zde zřetelné, že inicializace objektu se neprovádí pomocí funkce konstruktoru. Samotnou konstrukci objektu obstarává Editor a neinicializuje se hned na začátku hry, ale až jakmile je vytvořena instance herního objektu, ke kterému je skript připojený.

Funkce Update je volána každý snímek a zodpovídá za aktualizace herního objektu ve hře. Nejčastěji se zde nacházejí příkazy pro pohyb, spouštění akcí a odpovědi na vstupy od uživatele, prostě všechno, co se musí zpracovávat v průběhu hraní. V tomto kódu se jedná o aktualizaci proměnné pro polohu postavy na ose X pomocí funkce

GetAxisRaw a volání funkce Flip, která otáčí obrázek hrdiny podle směru, kterým se pohybuje, takže bude vždy chodit dopředu a nikoliv pozpátku. Funkce GetAxisRaw může vrátit jednu ze tří hodnot: -1, 0 nebo 1. Hodnota -1 znamená, že se postava pohybuje směrem doleva, 1 znamená pohyb doprava a 0 indikuje, že se postava nepohybuje.

Funkce FixedUpdate je volána v pravidelných intervalech a je určena pro fyzikální výpočty. Ve výchozím nastavení je volána každých 0.02 sekundy. Díky této funkci můžeme nastavit rychlost, či jinou sílu, pro RigidBody2D, která se bude aplikovat v každém snímku. V tomto kódu aplikujeme na RigidBody2D rychlost tak, že osa Y zůstává neměnná a osa X se mění na základě horizontální polohy herního objektu a proměnné speed, kterou jsme nastavili na začátku. Pokud se nám bude zdát pohyb herního objektu ve hře příliš pomalý nebo rychlý, můžeme ho kdykoliv změnit v Editoru, protože jsme ho dekorovali atributem [Serialize-Field].

Funkce Flip zajišťuje, že obrázek postavy bude otočený směrem, kterým se pohybuje. Pokud je postava otočená doprava a hodnota proměnné _horizontal je záporná nebo je postava otočená doleva, ale hodnota proměnné _horizontal je kladná čili postava se pohybuje doprava, tak se provede otočení obrázku na ose X.

Obrázek 9. Hlavní hrdina ve scéně

```
lic class char_controller : MonoBehaviour

 [SerializeField] private float speed = 5;  "10"

private Rigidbody2D _rigidbody2D;
private float _horizontal;
private bool _isFacingRight = true;

 Event function
void Start()
{
    _rigidbody2D = GetComponent<Rigidbody2D>();
}


 Event function
void Update()
{
    _horizontal = Input.GetAxisRaw("Horizontal");
    Flip();
}


 Event function
private void FixedUpdate()
{
    _rigidbody2D.velocity = new Vector2( x: _horizontal * speed, _rigidbody2D.velocity.y)
}

 Frequently called  1 usage
private void Flip()
{
    if (_isFacingRight && _horizontal < 0f || !_isFacingRight && _horizontal > 0f)
    {
        _isFacingRight = !_isFacingRight;
        Vector3 localScale = transform.localScale;
        localScale.x *= -1f;
        transform.localScale = localScale;
    }
}
```

Obrázek 10. Skript pro pohyb postavy

6.1.3.2 Skript pro omezení pohybu postavy

Protože nechci, aby se hráči postava ztratila z dohledu musím nastavit nějaké hranice. Díky tomu, že má postava Kinematic Body Type, nepadá dolů a nemusela jsem tak řešit žádnou zem, která by ji zastavila. Ale postava stále může opustit zorné pole kamery. Na ošetření postačí jednoduchý skript, kde jako zjistím pomocí funkce WorldToViewportPoint pozici objektu vůči kameře. Potom zjistím šířku a výšku objektu tak, že se podívám do komponenty Sprite Renderer. Jako další spočítám velikost objektu vůči souřadnicím kamery. Hodnota na ose Z je nastavena na -10, aby zajistila, že objekt bude zobrazen nad všemi ostatními objekty ve scéně, takže bude pořád viditelný. Funkce Mathf.Clamp omezí hodnotu pozice na určitý interval, ve kterém se může postava pohybovat aniž by se ztratila z dohledu kamery.

```
public class boundaries : MonoBehaviour
{
    public Camera mainCamera;  Changed in 1+ assets

    private float objectWidth;
    private float objectHeight;

    Event function
    private void Update()
    {
        Vector3 pos = mainCamera.WorldToViewportPoint(transform.position);

        float objectWidth = transform.GetComponent<SpriteRenderer>().bounds.size.x*20;
        float objectHeight = transform.GetComponent<SpriteRenderer>().bounds.size.y*20;

        Vector3 objectViewportSize = new Vector3( x: objectWidth / mainCamera.pixelWidth,
            y: objectHeight / mainCamera.pixelHeight, z: -10);

        pos.x = Mathf.Clamp( value: pos.x, min: objectViewportSize.x, max: 1 - objectViewportSize.x);
        pos.y = Mathf.Clamp( value: pos.y, min: objectViewportSize.y, max: 1 - objectViewportSize.y);

        transform.position = mainCamera.ViewportToWorldPoint(pos);

    }
}
```

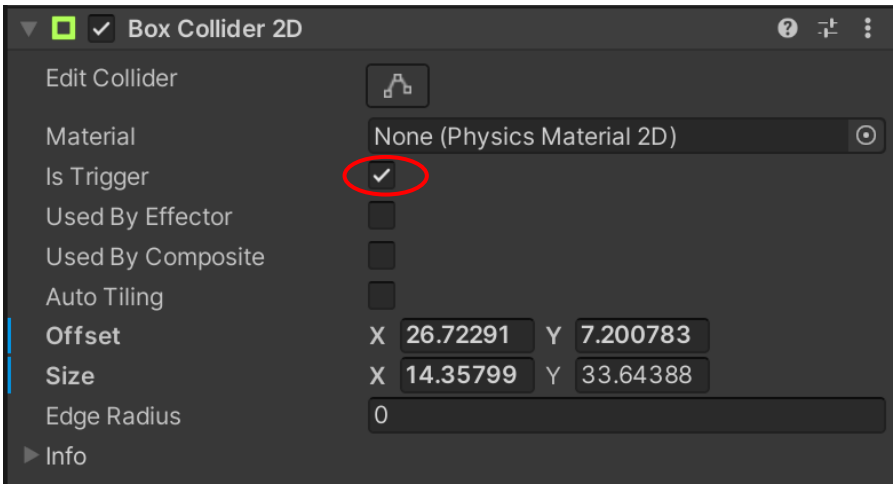
Obrázek 11. Skript pro vytvoření hranice

Potom už jen stačilo přidat skripty postavě hlavního hrdiny, vytvořit z něj prefab a vložit do scény. Pohyb doleva a doprava se ovládá na klávesnici šipkou doleva a doprava nebo klávesami A a D. Toto je výchozí nastavení Unity, jde změnit přímo v Editoru bez psaní skriptu. Je dostupný pod záložkou Edit-> Project Settings-> Input Manager. Samozřejmě jde přepsat i skriptem.

6.2 Strážce

Hlavní hrdina bude po cestě potkávat další postavy, takže budou potřeba další postavy. Znovu jsem vytvořila Sprite a dala mu obrázek pro strážce. Protože chci, aby interagoval s postavou hlavního hrdiny bude potřebovat collider. Stejně jako u hlavního hrdiny postačí BoxCollider2D.

Kinematic Body Type neumožňuje detekovat ani odpovídat na kolize, ale dokáže detekovat, zda se protnul se spouštěčem, tzv. trigger. Spouštěče zahrnují pouze detekci kolize, ale neposkytují žádné další informace jako je například rychlost kolize. Lze pouze získat informaci o vstupu do kolize, respektive spouštěče, zda je objekt uvnitř spouštěče nebo jestli objekt spouštěč opustil. Proto je potřeba u tohoto collideru povolit možnost Is Trigger. Povolením se zajistí, že se collider bude chovat jako spouštěč pro nějakou událost. V této hře to bude zobrazení dialogového okna. Postava strážce se hýbat nebude, takže chybí napsat už jen skript, který bude reagovat na kolizi postavy strážce a hlavního hrdiny.



Obrázek 12. BoxCollidet2D jako Trigger

6.2.1 Skript

Funkce OnTriggerEnter2D se spustí, jakmile komponenta Collider2D přidaná k hernímu objektu, v tomto případě k postavě hlavního hrdiny, se potká s komponentou Collider2D, kterou má přidanou postava strážce. Skript bude fungovat, díky tomu, že u postavy strážce je collider nastaven tak, aby se choval jako spoušť. Uvnitř funkce volám funkci ShowDialog, která je definovaná v jiném skriptu.

```
public class task_detector_lvl3 : MonoBehaviour
{
    Event function
    private void OnTriggerEnter2D(Collider2D collision)
    {
        dialog_logic.Instance.ShowDialog("Stanul jsi před starou truhlicí. Můžeš ji vykrást a dostat 10 zlaťáků, " +
                                         "ale riskovat Prokletí dungeonu, nebo zabít strážce a získat zlaťáků pět." +
                                         "Co uděláš?\n" +
                                         "A) Vykradeš truhlici\n" +
                                         "B) Zabiješ strážce", aAction: () =>
        {
            char_behaviour.PridejZlato( choice: 2, char_behaviour.ValidniVolba( choice: 0));
        }, bAction: () =>
        {
            char_behaviour.PridejZlato( choice: 1, char_behaviour.ValidniVolba( choice: 1));
        });

    }
}
```

Obrázek 13. Skript pro detekci kolize objektů

Postava strážce je hotová, znovu jsem z ní vytvořila prefab a vložila so scény. Později ve hře existují variace této postavy, jako je obchodník, golem a vězeň, kde vypadá jinak nebo má jiný skript, případně obojí, ale princip je stále stejný.

Obrázek 14. Varianta strážce jako obchodníka

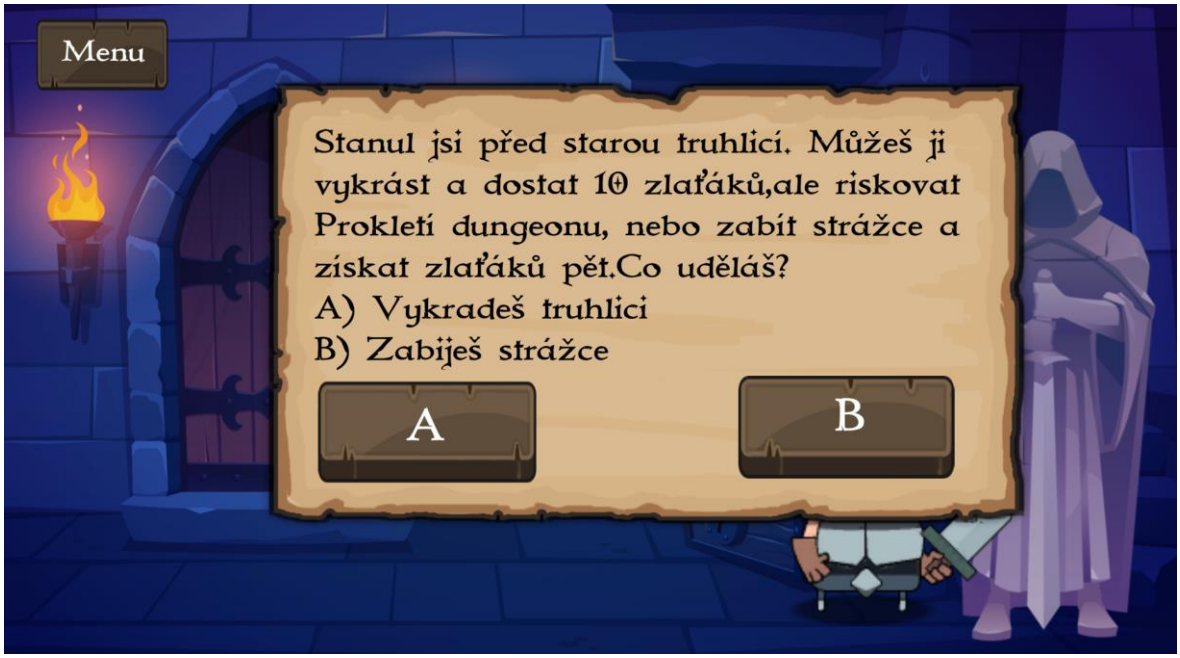
Obrázek 15. Varianta strážce jako golema

6.3 Hlavní dialog

Velkou součástí hry jsou dialogová okna, skrze která je vyprávěn příběh hry a umožňují hráči činit volby a posouvat se dál hrou až na konec. Aby se jakýkoli text vykreslil musí se do scény přidat tzv. canvas. Potom jsem vytvořila prázdný herní objekt jako potomka objektu canvas a v tomto novém objektu jsem vytvořila objekt image, který bude sloužit jako pozadí, objekt text pro textové pole a jako poslední dvě tlačítka. Stejně tak i tlačítkům jsem přidala objekty Image a Text. Tlačítko má komponentu Button, díky kterému lze přímo v Editoru přiřazovat při kliknutí na něj různé akce, umožňuje přidání různých efektů, jako je například změna barvy tlačítka, když uživatel přes něj přejede myší a podobně. Při vytváření textového pole Unity vyžaduje instalaci balíčku TextMeshPro.

TextMeshPro je knihovna speciálně vytvořena pro Unity a nabízí lepší kvalitu textu ve hře a obsahuje užitečné nástroje pro úpravu textu.

Unity standartně obsahuje pouze jeden font. Pro použití fontu v Unity je potřeba z něj vytvořit asset. To se dá jednoduše udělat přímo v Editou.



Obrázek 16. Ukázka dialogového okna

6.3.1 Skript

Na začátku hry nechci, aby byl dialog viditelný, takže bude zapotřebí skript, který bude dialog schovávat a znovu ukazovat. Hned na začátku deklaruji statickou propertu Instance, která umožňuje přístup k instanci třídy dialog_logic z ostatních skriptů. Je zde použit Singleton, takže bude vždy existovat pouze jedna instance třídy.

Funkce Awake se v Unity volá jako úplně první funkce, ještě před funkcí Start. Volá se hned potom, co je herní objekt instanciovaný. Pokud objekt není za začátku aktivní, tak se funkce Awake nezavolá hned, ale až potom, co se objekt aktivuje. V této funkci je instance třídy přiřazena propertě Instance.

Proměnným TextMeshPro, yesBtn a noBtn jsou přiřazeny reference na odpovídající komponenty pomocí funkce GetComponent. Nakonec je zavolána funkce Hide, která celý herní objekt schová. Díky tomu, že se funkce Awake volá ještě před funkcí Start se objekt schová dřív, než se vykreslí na obrazovku.

Funkce ShowDialog umožňuje zobrazovat dialog.

Tlačítkům je pomocí funkce onClick.AddListener přidán tzv. event handler. Díky tomu se po kliknutí na tlačítko provedou všechny akce s ním spojené. Pro obě tlačítka se po kliknutí přehraje zvukový efekt, dialogové okno se schová a vykonají se akce specifikované v delegátu.

Příklad volání funkce ShowDialog je demonstrován na obrázku 13.

```
public class dialog_logic : MonoBehaviour
{
    11 usages
    public static dialog_logic Instance { get; private set; }
    private TextMeshProUGUI textMeshPro;
    private Button yesBtn;
    private Button noBtn;
    Event function
    private void Awake()
    {
        Instance = this;
        textMeshPro = transform.Find("Text").GetComponent<TextMeshProUGUI>();
        yesBtn = transform.Find("YesBtn").GetComponent<Button>();
        noBtn = transform.Find("NoBtn").GetComponent<Button>();

        Hide();
    }

    10 usages
    public void ShowDialog(string dialogText, Action aAction, Action bAction)
    {
        FindObjectOfType<AudioManager>().Play( name: "dialog");
        gameObject.SetActive(true);

        textMeshPro.text = dialogText;
        yesBtn.onClick.AddListener( call: () =>
        {
            FindObjectOfType<AudioManager>().Play( name: "confirm");
            Hide();
            aAction();
        });
        noBtn.onClick.AddListener( call: () =>
        {
            FindObjectOfType<AudioManager>().Play( name: "confirm");
            Hide();
            bAction();
        });
    }

    3 usages
    private void Hide()
    {
        gameObject.SetActive(false);
    }
}
```

Obrázek 17. Skript pro dialog

6.4 Informační dialog

Podobně jako hlavní dialog jsem vytvořila i další dialog, s tím rozdílem, že tento bude mít pouze jedno tlačítko. Tvorba dialogu i skriptu je velice podobná hlavnímu dialogu. Potom, co byl dialog vytvořen, jsem přidala tlačítku script, který umožní přechod z jedné scény do druhé.

6.4.1 Skript pro přechod mezi scénami

Na začátku je deklarovaná proměnná typu Animator. Představuje komponentu animator, která zodpovídá za animaci. Animace byla vytvořena v Editoru a jedná se o jednoduchou manipulaci alfa kanálu v čase. Vytváří tak dojem plynulého přechodu mezi scénami.

Funkce LoadNextLevel volá tzv. coroutine. Ta umožňuje rozprostřít úlohy přes několik snímků. Obyčejně když je zavolána funkce, tak se dokončí během jediného snímku, to by pro hru znamenalo, že objekt zmizí instantně. Proto je zapotřebí použít coroutine. V C# se deklaruje jako IEnumerator a volá se pomocí StartCoroutine.

V komponentě animator je nastavena podmínka pro přechod z jedné animace do druhé. Na začátku při načtení scény se automaticky spustí první animace, kdy dochází k přechodu z tmavé obrazovky do scény. Potom animator čeká, dokud není splněna podmínka pro přechod do druhé animace. Jakmile je podmínka splněna, animace se spustí a obrazovka postupně zčerná. Tato podmínka se spouští ve funkci LoadLevel funkcí SetTrigger.

Příkaz yield return new WaitForSecods(2) pozastaví coroutine, v tomto kódu na dvě sekundy, a dovolí tak nerušeně přehrát animaci. Po uplynutí dvou sekund je zavolána funkce LoadScene, která načte scénu s odpovídajícím indexem.

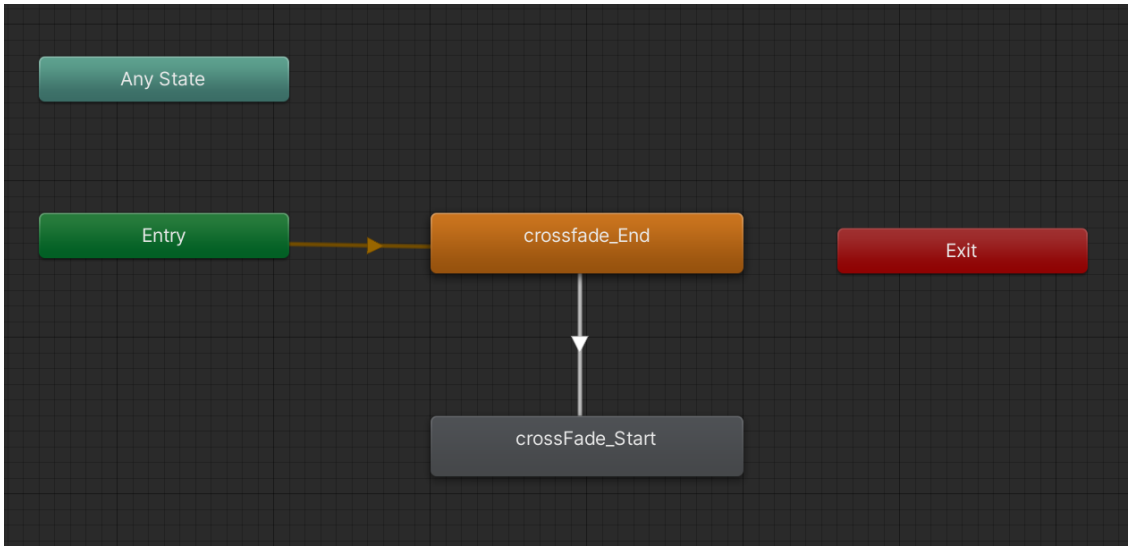
V Unity je třída Scene Manager součástí Unity API a používá se k ovládání scén ve hře. Umožňuje jednoduše zjistit, ve které scéně se hráč nachází, její index nebo jméno scény, načítat scény podle jejich indexu nebo jména, přesouvat herní objekty z aktivní scény do jiné scény a další.

```
public class LevelLoader : MonoBehaviour
{
    public Animator trannsition;  Changed in 14+ assets

    0+ asset usages
    public void LoadNextLevel()
    {
        StartCoroutine( routine: LoadLevel(SceneManager.GetActiveScene().buildIndex + 1));
    }

    0+ asset usages
    public void MenuLoad()
    {
        StartCoroutine( routine: LoadLevel(2));
    }

    Frequently called  2 usages
    IEnumerator LoadLevel(int levelIndex)
    {
        trannsition.SetTrigger( name: "StartTransition");

        yield return new WaitForSeconds(2);

        SceneManager.LoadScene(levelIndex);
    }
}
```

Obrázek 18. Skript pro přechod mezi scénami

6.4.2 Animace přechodu

Abych se mohl animovat objekt, musí se nejdříve vytvořit. Pro tuto animaci jsem vytvořila prázdný objekt, do něj jsem umístila canvas a do canvasu jsem přidala obrázek tmavého čtverce, kterým jsem vyplnila celou kameru. Obrázek sloužil pro vytvoření dvou animací přechodu. Jeden pro začátek scény, jeden pro přehrání při opuštění scény. Při ukládání animace vytváří Unity i ovladač pro danou animaci. V ovladači je definováno, které animace se mají pro daný herní objekt přehrát, kdy se mají přehrát a jak se mají přehrát. Potom stačilo přidat komponentu animator pro objekt canvas a nastavit Controller v animatoru na ovladač pro tuto animaci. Z celého objektu je vytvořen prefab a je přítomen v každé scéně.



Obrázek 19. Ovladač pro animaci přechodu

Teď když už mám hotový objekt, můžu přidat tlačítku u informačního dialogu funkci pro načtení další scény. V komponentě button je možnost přidat funkci, která se spustí při kliknutí na tlačítko. Musí se jí přiřadit objekt a vybrat funkce ze skriptu, který má k sobě přiřazený, jak daný objekt, tak i samotné tlačítko, jinak se funkce definovaná ve skriptu nezobrazí. Pro tuto hru to je standartně funkce LoadNextLevel z obrázku 18.

6.5 Hlavní menu

Hlavní menu sestává ze dvou tlačítek, jedno pro zahájení hry a jedno pro její ukončení. Obě tlačítka mají přiřazený skript, který přehraje zvuk, pokud jsou stisknuta a komponentu animator, která ovládá jejich animace. Tlačítko pro zahájení hry má potom ještě skript pro načítání další scény a funguje podobně jako u informačního dialogu. Tlačítko pro ukončení hry zobrazuje při kliknutí dialog pro ukončení hry.

Obrázek 20. Hlavní menu

6.6 Ukončit hru

Dialog pro ukončení hry je skoro totožný s hlavním dialogem. Vyvstává tady otázkou, proč rovnou nepoužít hlavní dialog. Je to proto, že se tento dialog používá ve hře nejen pro ukončení hry, ale i během samotného hraní, kde slouží jako možnost návratu do hlavního menu. A protože nelze mít ve hře dvě instance hlavního dialogu, bylo potřeba vytvořit další dialog.

Během hry je v levém horním rohu umístěno tlačítko Menu pro návrat do scény s hlavním menu. Oba tyto objekty, dialog pro ukončení hry a tlačítko menu, spravuje skript, který byl přidán na hlavní kameru. I když to s její funkcionalitou nijak nesouvisí, je to jediný objekt, který je dostupný na všech scénách, a nemá přidané žádné jiné skripty, takže bylo nejlepší přidat tento skript právě na kameru.

6.6.1 Skript pro ukončení hry a návrat do menu

Ve funkci Update se nejprve zjistí, které tlačítko je stisknuto a podle toho se zobrazí buď dialog pro ukončení hry, ten je dostupný pouze z hlavního menu, nebo dialog pro návrat do hlavního menu. Při dialogu pro návrat do hlavního menu je volána funkce NewGame, která vymaže hráčův dosavadní postup.

```
public class Esc : MonoBehaviour
{
    Event function
    void Update()
    {
        string BtnName = EventSystem.current.currentSelectedGameObject.name;

        if (BtnName == "QuitBtn")
        {
            Quit_logic.Instance.ShowInfo( dialogText: "Opravdu chceš ukončit hru?", aAction: () =>
            {
                Application.Quit();
            }, bAction: () => { });
        }

        if (BtnName == "MenuBtn")
        {
            Quit_logic.Instance.ShowInfo( dialogText: "Opravdu se chceš vrátit do menu? " +
                                          "Ztratíš tím dosavadní postup.", aAction: () =>
            {
                char_behaviour.NewGame();
            }, bAction: () => {});
        }

    }
}
```

Obrázek 21. Script pro ukončení hry

6.7 Audio Manager

Hudba je důležitou součástí hry. Protože chci ovládat kdy se bude přehrávat daný zvukový efekt a jaká hudba bude hrát, je potřeba vytvořit skript, který umožní ovládat zvuk. Nejzákladnější je schopnost ovládat nastavení hlasitosti hudby a výšky tónu. Důležité je nastavení, zda se bude skladba přehrávat ve smyčce nebo se přehraje pouze jednou. Mezi další užitečné proměnné patří název skladby, skladba samotná a komponenta audio source, tedy zdroj zvuku. Pro všechny tyto proměnné jsem vytvořila třídu Sound. Celá třída je dekorovaná atributem [System.Serializable], takže všechny členské proměnné uvnitř třídy budou viditelné a také změnitelné uvnitř Editoru. Poslední proměnná source představuje komponentu, která zajišťuje přehrávání zvuku. V tomto skriptu je dekorovaná atributem [HideInInspektor], který zajistí, že se v Editoru jako jediná nezobrazí. Je dobré se ujistit, že kamera má komponentu Audio Listener, protože jinak žádný zvuk slyšet nebude. Případně tuto komponentu může obsahovat i jiný objekt ve scéně, ale při vytváření projektu ji Unity běžně umisťuje právě na kameru.

Třída Sound zapouzdřuje vlastnosti a atributy zvukové stopy, která se používá v systému pro správu zvuku. Lze vytvořit více instancí této třídy, které reprezentují různé zvukové efekty, hudební stopy nebo jiné zvukové prostředky používané ve hře.

```
[System.Serializable]
4 usages
public class Sound
{
    public string name;  Serializable

    public AudioClip clip;  Serializable

    [Range(0f,1f)]
    public float volume;  Serializable
    [Range(.1f,3f)]
    public float pitch;  Serializable

    public bool loop;  Serializable

    [HideInInspector]
    public AudioSource source;  Serializable
}
```

Obrázek 22. Třída Zvuk

Další skript bude pro samotný manager audia. Nejprve jsem vytvořila pole, které bude obsahovat objekty představující kolekci audio nahrávek spravované audio managerem a statickou referenci na instanci audio managera, díky které na něj můžou přistoupit ostatní skripty bez nutnosti přímé reference.

Ve funkci Awake je použit vzor singleton a kontroluje, zda ve scéně existuje instance objektu AudioManager. Pokud ne, přiřadí scéně instanci objektu, pokud ano, zničí současnou instanci. Díky tomu se nemusí objekt AudioManager přidávat do všech scén, ale pouze do první scény. Ve hře chceme docílit kontinuálního přehrávání zvuku, proto je zde funkce DontDestroyOnLoad, který zajistí, že objekt AudioManager je přenesen ze současné scény do další, tudíž přehrávání zvuku nerušeně pokračuje dál. Nebýt této funkce zvuk by se pokaždé při načítání další scény zastavil a v další scéně by začal hrát znovu od začátku. Poslední částí funkce Awake je smyčka, která přidává objektu ve scéně všechny nezbytné proměnné, které jsou definované ve třídě Sound.

Po funkci Awake se volá funkce Start. Protože instance objektu audio manager je zajištěna ve funkci Awake, můžeme ve funkci Start už s objektem pracovat a spustit zvukovou stopu. Další dvě funkce, Play a Stop, hledají v objektu zvukové stopy podle jejich názvu a umožňují je spouštět a zastavovat z jakéhokoliv skriptu.

```
public class AudioManager : MonoBehaviour
{
    public Sound[] sounds;  Serializable  More...

    public static AudioManager instance;

    Event function
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
                Destroy(gameObject);
                return;
        }

        DontDestroyOnLoad(gameObject);

        foreach (Sound s in sounds)
        {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;

            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
        }
    }
```

Obrázek 23. Skript pro řízení zvuku, část 1

```
private void Start()
{
    Play( name: "theme");
}

Frequently called  28 usages
public void Play(string name)
{
    Sound s = Array.Find(sounds,  match: sound => sound.name == name);
    s.source.Play();
}

Frequently called  8 usages
public void Stop(string name)
{
    Sound s = Array.Find(sounds,  match: sound => sound.name == name);
    s.source.Stop();
}
```

Obrázek 24. Skript pro řízení zvuku, část 2

7 SCÉNY HRY A BODOVÁNÍ

7.1 Mapa hry

Každá dobrá hra potřebuje úvodní logo a název. Protože v neplacené verzi Unity si nelze plně přizpůsobit úvodní snímek, vytvořila jsem samostatnou scénu, která bude sloužit jako úvodní snímek. Na začátku hry se kvůli tomu jako první objeví snímek s logem Unity, to lze změnit až v placené verzi, a potom moje vlastní logo s názvem hry.

Obrázek 25. Úvodní scéna

Následuje úvod do hry, za koho hráč hraje, kam se vydává a proč a co je cílem hry. Pro další snímek jsem se podívala na to, jak je strukturovaná většina současných her a rozhodla se tam umístit hlavní menu a až potom následovat se scénou, která vysvětluje ovládání hry. Po něm už následuje hra samotná.

Jádro hry tvoří mapa s celkem devíti hlavními uzly, kde si hráč může zvolit ze dvou možností. Na jedno místě se mapa větví a vede k různým uzlům, ale hned se zase znovu spojuje. Každé rozhodnutí se počítá a rozhoduje o tom, který konec hráč dostane. Hra má celkem sedm konců a dvě konečné scény. Kterou scénu dostane na konci hry dostane, závisí na počtu konečného množství zlaťáků. O tom, který konec dostane rozhoduje kombinace dvou faktorů, a to počet konečného počtu zlaťáků a zda hráč skolil draka či nikoliv.

Po konečné scéně následuje scéna s logem hry a závěrečné titulky s informacemi o tvůrcích hry a použitých assetů. Hra potom navazuje na scénu s úvodem do hry. Hra má celkem 18 scén.

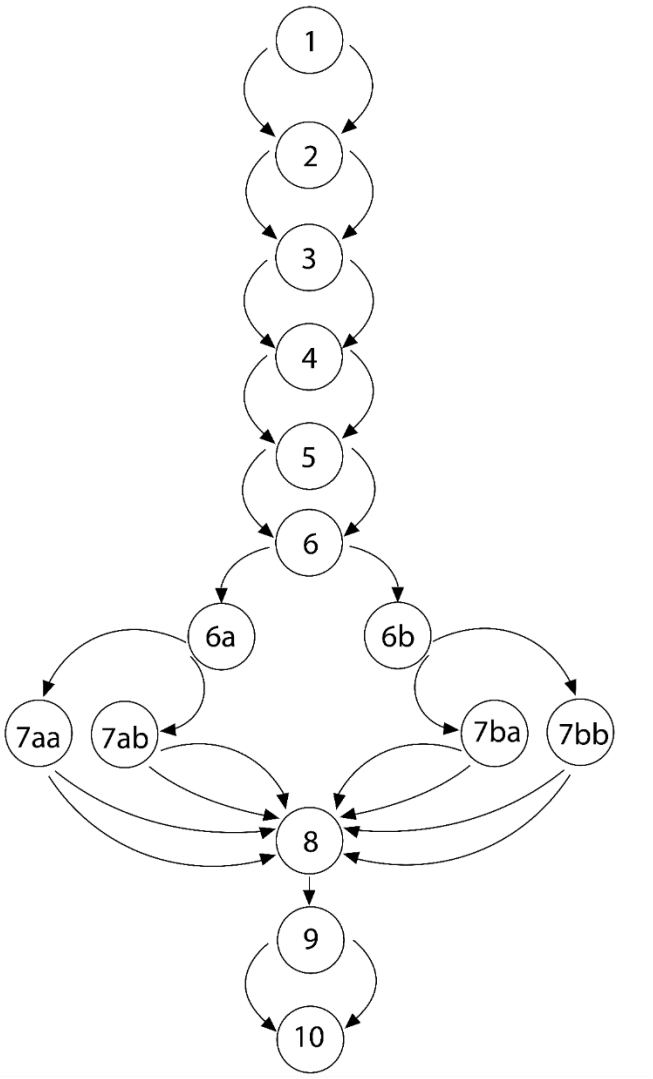
7.2 Bodování

Hra je bodována na základě voleb, které hráč učiní. Podle toho dostává skrze hru zlaťáky, ale taky je může ztratit. Přijít o zlaťáky jde dvěma způsoby. Hráč buď může svými činy aktivovat Hněv Dungeonu a bude tak mít každý další uzel, včetně uzlu, kdy jej aktivoval, procentuální šanci ztratit 3 zlaťáky. Druhý způsob je v uzlu 6, viz obrázek 14, kde se nachází obchodník. Hráč si od něj může koupit jeden předmět, a to i v případě, že stojí víc zlaťáků, než má hráč v současnosti k dispozici. Pokud bude hráč na konci hry v mínusu se zlaťáky, zadluží se vůči dungeonu a vyústí to v jednu ze dvou konečných scén. Pokud je na konci hry na nule se zlaťáky nebo v plusu, dostane druhou konečnou scénu.

Ve hře je jeden skrytý mechanismus, a to Požehnání bohyně. Jedná se o procentuální možnost pro hráče získat při každém rozhodnutí jeden zlaťák navíc. Požehnání bohyně se ale neaktivuje, pokud hráč aktivoval Hněv dungeonu.

Hráč taky musí sledovat stav svého meče, protože při některých rozhodnutích, většinou pokud útočí, se meč poškodí. Pokud se stav meče dostane na nulu, zlomí se a hráč jej nebude moct použít.

Hra je unikátní oproti většině současných her, že nechá dělat hráče i rozhodnutí, která nedávají smysl. Například se může rozhodnout zaútočit, aniž by měl meč. V takovém případě přijde o možnost jakéhokoliv výdělku v daném uzlu, nic se mu nepřičte a pokračuje dál. Hráč tady platí za svou hloupost.



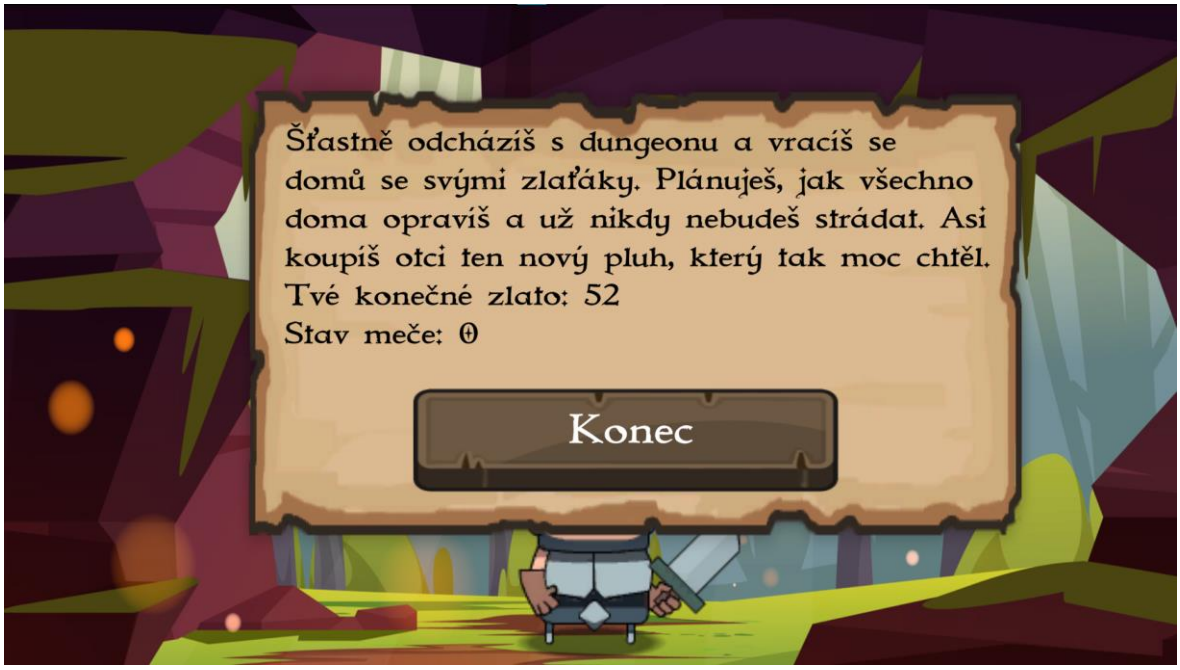
Obrázek 26. Mapa hry

7.2.1 Nejnižší možný počet zlaťáků

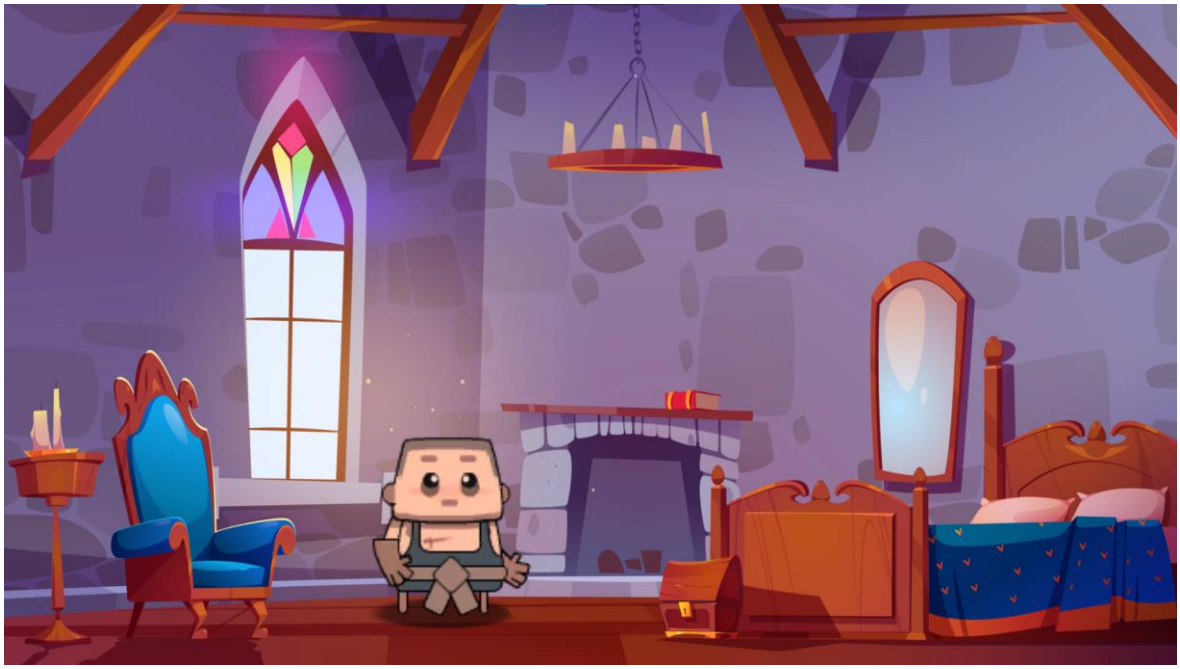
Nejnižší možný počet zlaťáků je -16 zlaťáků. Aby hráč tohoto čísla docílil, musel by mít opravdu smůlu. Aby se hráč dostal do mínusu musel by probudit Hněv dungeonu. Bez toho se do mínusu jde dostat pouze jednou velmi specifickou sekvencí rozhodování, a ještě k tomu by za celou hru hráč nesměl ani jednou dostat Požehnání bohyně.

7.2.2 Nejvyšší možný počet zlaťáků

Teoretický nejvyšší počet zlaťáků je 67. To je ale pouze teoretický počet, protože hráč by musel v každém uzlu dostat jeden zlaťák navíc z Požehnání bohyně a jeho meč by musel pokaždé dostat nejmenší možné poškození. Průměrný počet zlaťáků se bude při optimálním rozhodování pohybovat mezi 42 a 55 zlaťáky.



Obrázek 27. Jeden z možných konců



Obrázek 28. Alternativní konec

8 PROCES VYTVÁŘENÍ UMĚLÉ INTELIGENCE

Umělá inteligence je napsána v jazyce python. Jako první jsem přepsala hru do jazyka pyhon a potom jsou zkoušela různé přístupy řešení pomocí umělé inteligence.

8.1 Tvorba hry

Začala jsem vytvořením funkce pro inicializaci všech proměnných a mapy hry. V tomto případě se jedná o dva listy, každý z nich obsahuje deset prvků. Protože jsem potřebovala vědět celkový postup mapou a přesně určit, která volba byla učiněna, každý prvek v obou listech dostal unikátní označení. Pomocí slovníku jsem potom každému prvku v obou listech přiřadila číselnou hodnotu. Zapotřebí byl ještě jeden slovník, který vrátí informaci, zda dané rozhodnutí, v tomto případě prvek v listu, znamená, že hráč útočí. Tato verze hry je pouze konzolová, takže veškerý text hry se vypisuje na konzoli a volby se činí vepsáním „a“, „A“ nebo „b“, „B“ do konzole.

8.2 Náhodný výběr

Pro generování čísel jsem použila klasický pseudo-random generátor, který python standartně obsahuje. Jedná se o obyčejný random. Jako druhý generátor náhodných čísel jsem použila funkci random.SystemRandom. Ani jeden není skutečně náhodný, ale random.SystemRandom je méně předvídatelný. Modul random v jazyce python poskytuje funkce pro generování pseudo-náhodných čísel. Pseudonáhodná čísla nejsou skutečně náhodná, místo toho jsou generována pomocí deterministických algoritmů. Pro mnoho aplikací však tato pseudo-náhodná čísla stačí. Modul random používá pro generování těchto čísel algoritmus Mersenne Twister. Poskytuje funkce jako random(), randint() a choice() pro generování náhodných čísel a voleb.

Naproti tomu třída random.SystemRandom používá jako zdroj náhodnosti generátor náhodných čísel operačního systému. Dokáže tak přistoupit ke kryptografickému generátoru náhodných čísel operačního systému, který je obvykle považován za bezpečnější a nepředvídatelnější než výchozí generátor pseudonáhodných čísel.

Skutečně náhodný výběr v jazyku python není, ale našla jsem aplikaci třetí strany pro generování skutečně náhodných čísel. Jedná se o The ANU Quantum Random Number Generator. Python poskytuje knihovnu quantumrandom, která s ní umožňuje interagovat. Pro hraní hry se však příliš nehodí, jelikož generování čísel skrze tuto knihovnu trvá poměrně dlouho.

8.3 Monte Carlo Tree Search

Implementovat algoritmus Monte Carlo Tree Search jsem se rozhodla z několika důvodů.

8.3.1 Nároky na algoritmus

V první řadě má hra mapu, kterou je nutné prohledat. Dále hra obsahuje více neurčitých proměnných a dělá ji tak poměrně komplexní. To také znamená, že další tah ve hře nemusí dát očekávaný výsledek, takže algoritmus by se měl být schopný tomu přizpůsobit. A jako poslední neexistuje jedna přesná cesta, která vždy nutně povede k nejlepšímu výsledku.

8.3.2 Prozkoumávání prostoru

Monte Carlo Tree Search je vyhledávací algoritmus založený na simulaci, díku tomu je schopný efektivně prozkoumat velký vyhledávací prostor vzorkováním a vyhodnocováním různých cest. Ve hře se dvěma možnostmi volby s nejistými proměnnými může být na základě provedených voleb více možných výsledků. Monte Carlo Tree Search umožňuje zkoumat různé kombinace voleb a vyhodnocovat jejich potenciální výsledky.

8.3.3 Zpracování neurčitosti

Algoritmus Monte Carlo Tree Search se dobře hodí pro práci s nejistými proměnnými. Dokáže zpracovávat pravděpodobnostní výsledky a rozhodovat se na základě statistické analýzy. Simulací více her a nashromážděním jejich výsledků může Monte Carlo Tree Search poskytnout odhad očekávaného konečného výsledku nebo podat informaci o užitečnosti různých voleb za přítomnosti nejistoty.

8.3.4 Adaptivní rozhodování

Jak algoritmus Monte Carlo Tree Search pokračuje v simulaci a vyhodnocování různých cest, přizpůsobuje své rozhodování na základě nashromážděných statistik. Tato adaptivní povaha umožňuje algoritmu docílit lepších výsledků.

8.3.5 Škálovatelnost

Monte Carlo Tree Search je známý svou škálovatelností, díky níž je vhodný pro komplexní hry s velkými prohledávacími prostory. Zvládne hry s vysokým faktorem větvení a velkým počtem možných stavů a zvládne efektivně prozkoumat a vyhodnotit různé kombinace.

Celkově Monte Carlo Tree Search poskytuje robustní a flexibilní rámec pro rozhodování ve hrách s nejistotou. Simulací a vyhodnocováním různých cest si dokáže poradit se složitostí a neurčitostí hry, přizpůsobit své volby a poskytnout pravděpodobnostní odhady očekávaných výsledků.

8.3.6 Implementace

Algoritmus má několik parametrů. Pro začátek potřebuje od hry vědět, kolik rozhodnutí má učinit, aby se dostal na konec hry a jaký je současný stav hry a jejích proměnných. V algoritmu je deklarován počet simulací, které mají proběhnout pro každou z možných voleb.

Algoritmus vejde do smyčky, kterou iteruje přes všechny dostupné možnosti. Uvnitř této smyčky je další smyčka, uvnitř které dochází k samotné simulaci hry. Při simulované hře algoritmus činí rozhodování náhodně. Po dokončení simulací algoritmus smyčky opustí, spočítá průměrné skóre pro všechny možnosti a vrátí možnost s nejvyšším průměrným skóre.

Celý tento proces rozhodování se opakuje pokaždé, když hra vejde do nového bodu na mapě, kde je potřeba učinit další rozhodnutí. Simulace probíhá od současného uzlu až po konečný a spočítá průměrné skoré pro každou možnost.

Celkově tento kód představuje základní implementaci Monte Carlo Tree Search pro herní scénář s neurčitými proměnnými. Simuluje více her, sčítá skóre a vybírá možnost s vyšším průměrným skóre.

8.4 Možnosti vylepšení

Jedna z možností, jak implementovaný algoritmus vylepšit je zkombinovat jej s Markovovými rozhodovacími procesy. To nám umožní využít silné stránky obou přístupů. Monte Carlo Tree Search lze použít k efektivnímu prozkoumání prostoru možných akcí a sestavení stromu možných budoucích stavů, zatímco model Markovova rozhodovacího procesu může poskytnout přesnější odhad pravděpodobnosti přechodů mezi stavy. To může vést k lepšímu rozhodování a efektivnějšímu hraní her.

9 VÝSLEDKY LIDSKÝCH HRÁČŮ A UMĚLÉ INTELIGENCE

V posledních letech slýcháme o umělé inteligenci čím dál více. Její rozvoj přinesl mnohé změny a měl značný vliv i v oblasti her. Algoritmy a systémy umělé inteligence ukázaly pozoruhodné schopnosti při hraní komplexních her a často překonávají schopnosti lidských hráčů. Navzdory těmto působivým výsledkům a schopnostem, disponují lidští hráči určitými jedinečnými schopnostmi, jako jsou intuice, emoční inteligence a kreativita. Tyto schopnosti a aspekty člověka hrají důležitou roli v jejich herním stylu. Lidští hráči vychází i z vlastních zkušeností, jak z reálného světa, tak i z těch herních. Předešlé zkušenosti z her ovlivňují i veškeré jejich budoucí hry. Všechny tyto aspekty tak mají vliv i na samotný vývoj nových her.

Srovnání lidských hráčů s algoritmem umělé inteligence nemá jen ukázat rozdíl mezi lidským hráčem a umělou inteligencí, ale taky má sloužit jako zpětná vazba vůči obtížnosti hry.

Srovnávání lidských hráčů a umělé inteligence proběhne především na základě konečného počtu zlaťáků.

9.1 Výsledky umělé inteligence

Jak lze vidět v tabulce 1, pseudo-náhodný výběr možností si vedl nejhůře. O něco sofistikovanější metoda náhodného výběru si vedla lépe, ale pořád nebyl ani jeden způsob lepší než algoritmus umělé inteligence Monte Carlo Tree Search. Samotná umělá inteligence byla lepší už pouze s jednou simulací než náhodný výběr. S rostoucím počtem simulací se dále zlepšovala, doku nedosáhla stropu kolem pětiset simulací. Při zvýšení na tisíc simulací bylo průměrné skóre podobné jako u pětiset simulací nebo dokonce horší. To může být způsobena několika důvody. Vzhledem ke komplexitě hry a počtu proměnných, které hru ovlivňují se může stát, že jejich akumulací vznikne šum a ten negativně ovlivní simulace. Celkový výsledek bude tak horší.

Dalším zajímavým ukazatel je poslední sloupec tabulky. Skoré nad 41 zlaťáků je dolní hranice, které lze dosáhnout, pokud jsou ve hře učiněna správná klíčová rozhodnutí. Vyšší skóre lze dosáhnout už jen pokud má hráč štěstí. Algoritmus Monte Carlo Tree Search tohoto skóre či vyššího dosahuje při 500 simulací ve více jak polovině odehraných her.

Tabulka 1. Výsledky umělé inteligence a náhodných rozhodnutí

Algoritmus	Počet simulací	Počet her	Průměrné skóre	Nejlepší skoré	Počet her se skoré nad 41
Pseudo-Random	0	100	10.94	50	4
SystemRandom	0	100	15.09	51	8
MCTS	1	100	18.77	54	11
MCTS	10	100	25.97	52	35
MCTS	50	100	26.53	54	38
MCTS	100	100	30.29	58	46
MCTS	100	100	29.40	57	45
MCTS	500	100	32.87	56	54
MCTS	500	100	33.71	59	58
MCTS	500	100	30.91	57	52
MCTS	1000	100	28.30	57	41
MCTS	1000	100	31.22	57	50

9.2 Výsledky lidských hráčů a jejich připomínky ke hře

U lidských hráčů jsem sledovala počet zlaťáků pro srovnání s umělou inteligencí, jejich věk a jestli měli předešlé zkušenosti s hraním her. Každý hráč měl možnost zanechat i komentář ke hře a pomoct ji tak vylepšit. Díky některým včasnějším komentářům jsem měla možnost i hru trochu předělat, aby byla příznivější pro budoucí hráče. Celkem se testování zúčastnilo 30 hráčů z různých věkových kategorií, ale převládali především děti do 15 let, jelikož těm byla hra určena. Jeden hráč z testovací skupiny zanechal zajímavý komentář, že hra by se hodila i pro důchodce jako způsob, jak jim přiblížit nové technologie. Je to zajímavý nápad, bohužel ve zbylém času nebyl dostatek zájemců abych tuto teorii otestovala. Mezi dětmi, které hráli hru byli jak chlapci, tak i dívky. Chlapci byli trochu více kritičtí, to je například vidět u jednoho komentáře, který zmiňuje chybějící funkcionalitu pro skákání, což se odrazilo v jeho hodnocení hry. Většina komentářů byla pozitivní, některé byli validní připomínky, ale obecně měla hra u dětí úspěch. Naopak většina hráčů nad 40 let řekla, že by si hru znovu nezahrála. Podařilo se mi i získat pár hráčů, kteří měli zkušenost s textovou knihou, tedy gamebookem, a ti byli hrou přímo nadšení. Jedna z jejich nejčastějších poznámek byla, že má málo příběhu a že by příběh chtělo rozšířit a přidat více textu. To je v přímém

kontrastu s komentářem od jednoho dítěte, které žádalo o méně textu. Protože je hra určena primárně dětem, tak jsem text navíc nepřidala.

9.2.1 O designu hry

Průměrné skoré u lidských hráčů bylo 27.7 a skoré nad 41 dosáhlo z celkem 30 hráčů osm. To implikuje, že hra je dostatečně komplexní, aby představovala výzvu. Z tabulky 2 je vidět, že šest z osmi hráčů se skoré nad 41 pravidelně hraje hry. Z toho vyplývá, že předešlá zkušenost s hraním her může přispět k dosáhnutí lepšího skoré. Hra je poměrně komplexní, a ne všechny herní mechaniky jsou na první pohled zřetelné. Zkušení hráči mají výhodu, protože jsou více obeznámeni s hraním her a můžou tak věnovat pozornost i méně nápadným detailům, které mohou nezkušeným či nepozorným hráčům uniknout. Samotný fakt, že vysoké skoré dosáhlo pouze osm hráčů a dva z nich hry nehrají, je způsoben i faktem, že design hry nespoléhá pouze na zkušenostech a schopnostech hráčů, ale i na čirém štěstí.

9.2.2 Herní styl

Spousta hráčů, jak zkušených, tak i nezkušených, volila strategii nejmenšího odporu. To znamená, že si nechtěli nikoho znepřátelit, všem pomoct, nikomu neublížit a příliš neriskovat. To nebyla úplně nejlepší strategie. Jeden z důvodů, proč měli pravidelní hráči převahu je, že se tolik nebáli udělat potenciálně riziková rozhodnutí, která v konečném důsledku přinesla větší zisk. Na druhou stranu, pokud příliš spoléhali na předchozí zkušenosti z her, mohlo je to vést k ne až tak dobrému konci. Nejlepší výsledek dostal hráč, pokud strategie kombinoval.

Tabulka 2. Výsledky lidských hráčů

Aktivně hraje hry	Věk	Konečné skoré	drak byl zabit	hodnocení 1 až 5	komentář *nepovinné
Ne	47	15	Ano	5	
Ne	48	10	Ne	3	
Ano	9	4	Ano	4	dobré
Ano	7	39	Ano	5	výborné
Ano	7	50	Ano	5	byla dobrá
Ne	7	49	Ano	5	hra se povedla
Ano	7	45	Ano	5	hra se hodně líbila
Ne	6	45	Ano	5	hra se hodně líbila
Ano	7	44	Ano	5	nejlepší hra, moc mě bavila
Ano	7	39	Ano	5	hra je v pohodě
Ne	49	8	Ano	5	moje první hra
Ne	19	32	Ano	3	dá se
Ano	12	20	Ne	5	výborné
Ano	13	11	Ano	4	nedalo se skákat
Ano	10	48	Ano	3	dobrý
Ano	9	36	Ano	4	dobrý
Ano	16	22	Ne	5	dobré
Ne	7	2	Ne	5	líbilo
Ano	7	15	Ne	5	
Ano	8	14	Ano	5	hrálo se mi dobře
Ano	7	21	Ne	5	
Ano	9	61	Ano	5	dobré -10 z 10
Ano	7	6	Ano	5	dobrá
Ne	8	41	Ano	5	dobrý
Ne	8	35	Ano	5	dobrý
Ano	12	39	Ano	4	zlepšit instruktáž
Ne	12	14	Ne	5	méně textů
Ne	47	12	Ne	5	pěkné i pro důchodce
Ano	23	56	Ano	5	
Ano	15	-2	Ne	2	

9.3 Umělá inteligence vs lidé

Lidští hráči měli průměrné skoré 27.7, což je lepší než umělá inteligence s 1, 10 a 50 simulacemi. Umělá inteligence je předčila až když měla 100 a 500 simulací, přičemž dosáhla nejlepšího výsledku při 500stech simulacích. Nejlepšího skoré dosáhli lidští hráči, a to skoré 61. Pokud se podíváme ale na počet her, které přesáhly skoré 41, vedli si lidští hráči poměrně špatně. Protože lidé hráli pouze 30 her, převedu údaje do procent. Pouhých 26.67 % odehraných her lidskými hráči přesáhlo tento limit, kdežto umělá inteligence už při deseti simulacích přesáhla limit ve 35 % her. Pro jistotu jsem provedla i měření výsledků umělé inteligence pouze pro 30 her, ale výsledek se od původního příliš nelišil. Skoré nad 41 měl ve 11 hrách ze 30, což je procentuálně 36.67 a stále tak převyšuje lidské hráče, viz tabulka 3. Výsledky měření pro 30 her pro algoritmus umělé inteligence se procentuálně příliš nemění od měření, kdy her bylo 100.

Tabulka 3. Výsledky pro 30 her, Lidé vs UI

Algoritmus	Počet simulací	Počet her	Průměrné skóre	Nejlepší skoré	Počet her se skoré nad 41
MCTS	1	30	18.70	50	1
MCTS	10	30	25.93	60	11
MCTS	50	30	26.96	55	11
Lidé	0	30	27.7	61	8
MCTS	100	30	30.46	53	13
MCTS	500	30	32.26	57	16

9.3.1 Shrnutí

Lidští hráči neměli tolik pokusů projít hrou jako měla umělá inteligence. I přesto si vedli obstojně a umělá inteligence je předčila až potom, co posbírala před každou volbou data ze sta simulací. To není ale neúspěch umělé inteligence. Lidé mají mnoho výhod, jako jsou předchozí zkušenosti, porozumění psanému textu, ve hře je někdy i napsané kolik která volba vynese. Mají intuici, představivost a pocity. Algoritmus umělé inteligence, který zde byl použit, oproti tomu má schopnost provádět spoustu simulací v krátkém čase. Ať už lidský hráč nebo umělá inteligence, oba mají své vlastní výhody a každý se je snaží uplatnit.

ZÁVĚR

V teoretické části byly porovnány typy herních enginů a jako nejvhodnější byl zvolen herní Engine Unity, především pro příjemné uživatelské prostředí, dostatek funkcionalit, jazyku C# a pro dostatek dostupných assetů.

Jako další je probrána problematika umělé inteligence s důrazem na algoritmy umělé inteligence zabývající se problematikou prohledávání mapy a hledání řešení v prostoru s důrazem na použitý algoritmus Monte Carlo Tree Search.

Poslední sekce teoretické části byl popsán vznik textových her, jejich definice a byly provedeno porovnání existujících digitalizovaných textových her, které jsou v současnosti k dispozici. Všechny se ukázaly být spíše zaměřené na dospělejší audienci, než na jakou jsem mířila já se svou hrou.

Prvním krokem tvorby samotné hry bylo najít vhodné materiály pro hru, upravit je a vytvořit z nich herní objekty. Pokud nebyl žádný vhodný materiál pro některý zamýšlený objekt dostupný, byl mnou vytvořen.

Kromě herní logiky bylo zapotřebí vytvořit i přehlednou navigaci ve hře, zakomponovat do hry instrukce, jak hru ovládat a zajistit správný přechod mezi scénami a zajistit tak nerušený tok příběhu.

Po dokončení hry v Unity byla její konzolová verze vytvořena v jazyce python, aby mohla být hrána algoritmem umělé inteligence. Jako hlavní algoritmus umělé inteligence byl použit Monte Carlo Tree Search.

Jako poslední byla posbírána data od lidských hráčů, kteří hráli hru vytvořenou v Unity a porovnána s výsledky umělé inteligence, která hrála konzolou hru vytvořenou v jazyce python.

Konečná hra se mezi dětmi osvědčila, a ještě dlouho bude sloužit jako zdroj zábavy pro všechny, kteří přijdou do Halenkovické knihovny. V plánu je pozměnit hru podle zanechaných komentářů a rozšít hru o více scén a rozhodnutí. Na základě výsledků umělé inteligence je v plánu hru, alespoň pro mladší hráče, zjednodušit.

SEZNAM POUŽITÉ LITERATURY

[1] What is a Game Engine? [online] [cit. 20.05.2023]. Dostupné z: https://usv.edu/blog/what-is-a-game-engine/

[2] Unity Documentation [online] [cit. 20.05.2023]. Dostupné z: https://docs.unity.cn/2022.1/Documentation/Manual/UnityManual.html

[3] Holan, Tomáš. Unity: první seznámení s tvorbou počítačových her. Praha: CZ.NIC, z.s.p.o., 2020. CZ.NIC. ISBN 978-80-88168-57-7

[4] A tour of the C# language [online] [cit. 20.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[5] Unity Asset Store [online] [cit. 20.05.2023]. Dosupné z: https://assetstore.unity.com/

[6] Arora, Simran Kaur. Unity vs Unreal Engine: Which Game Engine Should You Choose? [online] [cit. 20.05.2023]. Dostupné z: https://hackr.io/blog/unity-vs-unreal-engine

[7] Unreal Engine Documentation [online] [cit. 20.05.2023]. Dostupné z: https://docs.unrealengine.com/5.1/en-US/

[8] Marketplace [online] [cit. 20.05.2023]. Dostupné z: https://www.unrealengine.com/marketplace/en-US/store?sessionInvalidated=true

[9] Unreal Engine [online] [cit. 20.05.2023]. Dostupné z: https://www.unrealengine.com/en-US

[10] Unity [online] [cit.20.05.2023]. Dostupné z: https://unity.com/

[11] McCarthy, John. What is Artificial Intelligence? [online] [cit. 20.05.2023]. Dostupné z: http://jmc.stanford.edu/articles/whatisai.html

[12] Turing, Alan M. COMPUTING MACHINERY AND INTELLIGENCE [online] [cit. 20.05.2023]. Dostupné z: https://academic.oup.com/mind/article/LIX/236/433/986238

[13] Larson, Erik J. The myth of artificial intelligence: why computers can't think the way we do. The Belknap Press of Harvard University Press, 2022. ISBN 978-0-674-27866-0

[14] The Chinese Room Argument [online] [cit. 20.05.2023]. Dostupné z: https://plato.stanford.edu/entries/chinese-room/

[15] How is artificial intelligence used in medicine? [online] [cit. 20.05.2023]. Dostupné z: https://www.ibm.com/topics/artificial-intelligence-medicine

[16] Taylor, Peter. The Birth of Project Intelligence [online] [cit. 20.05.2023]. Dostupné z: https://pmi.org.sg/symposium-2017-overview/120-pmquest/archive-pmquest/222-the-birth-of-project-intelligence

[17] What is machine learning? [online] [cit. 20.05.2023]. Dostupné z: https://www.ibm.com/topics/machine-learning

[18] Kelleher, John D. a Tierney, Brendan. Data science. The MIT Press, 2018. ISBN 978-0-262-53543-4

[19] Stone, James V. Artificial intelligence engines: a tutorial introduction to the mathematics of deep learning. Sebtel Press, 2019. ISBN 978-0-9563728-1-9.

[20] John Song, 2019, A Comparison of Pathfinding Algorithms, YouTube video [cit. 20.05.2023]. Dostupné z: https://www.youtube.com/watch?v=GC-nBgi9r0U

[21] Computerphile, 2022, Markov Decision Processes. YouTube video [cit. 20.05.2023]. Dostupné z: https://youtu.be/2iF9PRriA7w

[22] Simonov, Andrey a Zagarskikh, Aleksandr a Fedorov, Victor. Applying Behavior characteristics to decision-making process to create believable game AI [online] [cit. 20.05.2023] Dostupné z: https://www.sciencedirect.com/science/article/pii/S187705091931141X

[23] Boucherie, Richard J. a van Dijk, Nico M., Markov Decision Processes in Practice. Springer, 2017. ISBN 978-3-319-47766-4

[24] Browne, Cameron B., Powley, E., Whitehouse, D., Lucas, Simon M., Cowling, Peter I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. a Colton, S. A Survey of Monte Carlo Tree Search Methods [online] [cit. 20.05.2023]. Dostupné z: http://www.incompleteideas.net/609%20dropbox/other%20readings%20and%20resources/MCTS-survey.pdf

[25] Sironi, Chiara Federica. Monte Carlo Tree Search for Artificial Intelligence in Games [online] Maastricht, 2019. Disertační práce. Maastricht University. Prof. dr. ir. R.L.M. Peeters. Dostupné z: https://dke.maastrichtuniversity.nl/c.sironi/wp-content/uploads/2020/01/Thesis-MCTSforAGIinGames.pdf

[26] Allen, Michael P., a Tildesley, Dominic J. Computer simulation of liquids. Oxford University Press, 2017. ISB 978-0-198-80319-5

[27] Ma, Yao a Jiliang Tang. Deep Learning on Graphs. Cambridge University Press, 2021. ISBN 978-1-108-83174-1

[28] Gamebook Examen de la obra de Herbert Quain [online] [cit. 20.05.2023]. Dostupné z: https://gamebooks.org/Item/7380/Show

[29] The Life and Death of Gamebooks [online] [cit.20.05.2023]. Dotupné z: https://awesomeliesblog.wordpress.com/2021/08/27/the-life-and-death-of-gamebooks/

[30] Choice of Games [online] [cit. 20.05.2023]. Dostupné z: https://www.choiceofgames.com/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GPS Global Positioning Systém

UI Umělá Inteligence

AI Artificial Intelligence

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Příloha P1: Přiložený Flash disk

PŘÍLOHA P I: PŘILOŽENÝ FLASH DISK

Bakalářská práce ve formátu PDF

Build hry

Projekt hry v Unity

Zdrojový kód algoritmu umělé inteligence