Ekonomická fakulta
Faculty of Economics

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované ekonomie a ekonomiky

Bakalářská práce

Vývoj 2D herní plošinové hry v Unity

Vypracoval: Jiří Kořínek
Vedoucí práce: Mgr. Radim Remeš, Ph.D.

České Budějovice 2023

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jiří KOŘÍNEK**
Osobní číslo:	**E20058**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Ekonomická informatika**
Téma práce:	**Vývoj 2D herní plošinové hry v Unity**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Cílem bakalářské práce je vytvoření herní aplikace pomocí vývojového systému Unity.
Hra se bude skládat z jedné velké mapy s několika oblastmi s různými obtížnostmi. Bude se jednat o 2D plošinovou adventure hru. Na hrací mapě se budou nalézat překážky v podobě pastí, cílem hry bude dostat se přes celou mapu na vítězný konec. Během vývoje hry bude využito programovacího jazyku C# a pro grafiku 2D spritů.

Metodický postup:

1. Studium odborné literatury.
2. Popis použitých technologií pro vývoj aplikace.
3. Návrh, popis vývoje a implementace aplikace.
4. Zhodnocení použitelnosti aplikace pro nasazení v reálném prostředí.
5. Závěr a doporučení.

Rozsah pracovní zprávy:	**40 – 50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. Albahari, J. (2021). *C# 9.0 in a Nutshell.* O'Reilly.
2. Bond, J. G. (2022). *Introduction to game design, prototyping, and development.* Addison Wesley.
3. Jackson, S. (2014). *Mastering unity 2D game development.* Packt Publishing.
4. Halpern, J. (2018). *Developing 2D games with unity: Independent game programming with C#.* Apress.
5. Hocking, J. (2015). *Unity in Action.* Manning Publications.

Vedoucí bakalářské práce: **Mgr. Radim Remeš, Ph.D.**
Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: **11. ledna 2022**
Termín odevzdání bakalářské práce: **14. dubna 2023**

doc. Dr. Ing. Dagmar Škodová Parmová
děkanka

doc. RNDr. Tomáš Mrkvička, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 16. března 2022

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury. Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ..... Jiří Kořínek .....

Abstrakt

Tato práce se zaměřuje na kompletní vývoj 2D hry vytvořené v herním enginu Unity pomocí programovacího jazyka C#. Vývoj zahrnuje programování herních funkcí pomocí jazyka C# a vizualizaci modelu hráče, světa a objektů pomocí různých grafických spritů. Cílem práce je napodobit jiné známé plošinové hry a začlenit do nich herní mechanismy, které hráči umožňují procházet mapou pomocí různých typů pohybu. Součástí projektu je naprogramování různých nepřátelských AI a hra se drží stylu plošinovek s rozsáhlými úrovněmi, kde se obtížnost zvyšuje s postupem hráče. Některé sprity jsou vytvořeny pomocí generativní umělé inteligence a programu Photoshop. Výsledným produktem je zábavná hra, která si klade za cíl zaujmout.

Klíčová slova: Unity, C#, vývoj plošinové hry

Abstract

This thesis focuses on the complete development of a 2D game created in the Unity game engine using the C# programming language. The development includes programming game functions with the C# language and visualizing the player model, world, and objects with different graphic sprites. The work aims to emulate other well-known platformer games, incorporating gameplay mechanics that enable the player to traverse the map faster through various types of movement. The project involves programming different hostile AIs, and the game follows the platformer style with a large level where the difficulty increases as the player progresses. Some sprites are custom created using generative AI and Photoshop. The final product is an entertaining game that aims to captivate its audience.

Keywords: Unity, C#, platform game development

Poděkování

Děkuji panu Mgr. Radimu Remešovi, Ph.D., za konzultace, vstřícnost a odborné vedení při zpracování této bakalářské práce. Dále chci poděkovat svým přátelům, rodině a blízkým za podporu v době studia.

Jiří Kořínek

Obsah

1 Úvod

Cílem této práce je vytvoření 2D plošinové hry pomocí Unity editoru. V této hře je za úkol uživatele dostat se na konec mapy za pomocí pohybu, skoků a speciálních pohybů. V tomto cíli mu zabraňují pokrok různé platformové hádanky, nástrahy a nepřátelé, přes které musí uživatel projít za pomocí svých reflexů a načasování. U této hry je proto důležité, aby nabídla dostatečné časové okno k naučení ovládání postavy pro překonání těžších úseků a aby chování nepřátel bylo dostatečně férové vůči uživatelovi.

V rešeršní části jsou popsány hry, které jsou přímou inspirací pro tuto práci po grafické a hratelnostní stránce a jejich historie. Jsou zde diskutovány typy počítačové grafiky a rozdíly v nich. Jako další jsou zde přiblíženy také typy softwaru, které byly použity při vývoji této aplikace a jejich výhody. Dále je vysvětlena základní terminologie při vývoji her.

V praktické části je popsán proces vývoje hry. V této části je vysvětleno, jak funguje pohyb hlavní postavy včetně speciálních pohybů, vytváření speciálních objektů ve světě, jako jsou bodáky či projektily, vytváření nepřátel, tvorba spritů s pomocí AI a design hlavní mapy se speciálními efekty.

2 Videohry

Videohry jako takové slouží hlavně k zábavě a simulacím. Jedná se o interaktivní software, který se hraje převážně na počítačích se systémem Windows, mobilních zařízeních a herních konzolích, jako je PlayStation, Xbox, nebo Nintendo Switch. Kvůli vysoce narůstající rychlosti nejen procesoru, ale i grafických karet a pamětí dosahují hry větší reálnosti jak po grafické stránce, tak v animacích a námětech. Zároveň jsou nejnovější hry čím dál tím větší, co se týče velikosti světa a detailů v něm. (PCMag, b.r.)

Mezi videohry se řadí jakákoliv hra, která se hraje na jakémkoliv typu zobrazovacího zařízení. Mezi první velké videohry se dají považovat arkádové hry. Tyto hry byly na specializovaných zařízeních, které se převážně vyskytovaly v budovách pod názvem „Arcade“. Tyto hry víceméně propadly v pozapomnění, avšak v lokacích jako např. Japonsko je stále velká „Arcade“ komunita. Mezi nynějšími platformami jsou stále rozdíly, avšak při hraní videoher se hlavně jedná o rozdíly ve výkonu, nabízených hrách, designu a ekosystému vybrané platformy. (“Video Game - New World Encyclopedia,” b.r.)

Pro hraní videoher lze používat několik možných herních periferií. V případě videoherních konzolích jako je PlayStation se využívají specializované ovladače, jiné přenosné konzole mají v sobě přímo zabudované ovládaní, jako je například Steam Deck. U počítačových her se převážně využívá k hraní kombinace myši a klávesnice, avšak stále se zde dá využít ovladač, či jiné typy periferií, jako např. volant k hraní závodních her, nebo joystick při hraní her s letadly. V posledních letech přišlo na trh mnoho zařízení virtuální reality. Tyto zařízení se ovládají pomocí brýlí na virtuální realitu, které se nasadí na hlavu a ovladačů, které se nasadí na ruce. Dokoupit se zároveň dají ovladače pro zbytek těla, ale tyto ovladače je většinou třeba koupit od prodejců třetí strany. Tento typ zařízení má poskytovat lepší vtáhnutí do světa, ve kterém hrajete. U těchto zařízení je problém cenové přístupnosti a toho, že většina her vyžaduje vetší fyzickou námahu než hry na ostatních platformách, a proto se ještě neujali u větší skupiny lidí. (“Video Game - New World Encyclopedia,” b.r.)

Téměř každý hrál někdy v životě nějakou videohru, ať se jedná o minihru na mobilu nebo o hru na konzoli od velké společnosti. Proto diskuse o videohrách zaujímá zvláštní místo. Navzdory tomu, že videohry jsou všepřítomné, málokrát se o nich mluví ve stejné úctě jako o jiných mediích, jako jsou knihy nebo filmy. S dobou se však tato diskuse pomalu, ale jistě mění, diskuse o hudbě či o grafické stránce her nabírá nových

rozměrů a lidé začínají tuto uměleckou formu brát vážně. Pro většinu lidí jsou ale stále hry jako provinilé potěšení, které radši potají. Herní designér Sid Meier poznamenal, že „hra je série zajímavých voleb“. Hraní těchto her a volby v nich utváří průběh hry a odhaluje kdo jsme a jak přemýšlíme. Hraní her je tedy nabytý i o akt odhalení. (Von Ehren, 2020)

2.1 Plošinové hry

Plošinová hra, anglicky „Platformer game“, je žánr videoher, ve kterých se uživatel pohybuje hlavně pomocí skákání a při tom plní úkoly, které vedou v pokroku hrou. Tento žánr má své místo jak ve 2D, tak 3D. Jediným požadavkem, aby se hra dala označit za plošinovou hru je, aby hlavní zaměření této hry bylo skákání po platformách a aby se tím pádem nehodila do jiného žánru. (Worral, 2021)

Název "plošinová hra" tedy odkazuje na řadu plošin, po kterých smí uživatel v mnoha těchto hrách běhat, skákat anebo používat různé schopnosti, pomocí kterých je cíle dosaženo rychleji. V plošinové hře jsou zároveň většinou implementovány hádanky, nepřátelé, překážky, sběratelské položky, a různé vedlejší cíle, které lze plnit při řešení hlavního cíle. Plošinové hry se dají zařadit do mnoho žánrů, jako jsou dobrodružné hry, strategie, či hlavolamy. Nejvíce se tento typ her ale považuje za podžánr akčních her. (MasterClass, 2021)

Mezi první plošinové hry se řadí hry, které se hráli pouze na statické herní ploše a místo skákání se hlavně zaměřovali na lezení po žebřících. První hrou v tomto žánru se považuje „Space Panic“, která vyšla roku 1980. Mezi více známou hru, která již podporovala skákání, se řadí „Donkey Kong“, tato hra vyšla roku 1981 a poprvé představila postavu Maria pod jménem „Jumpman“. V této hře Mario cestuje po ocelových plošinách směrem nahoru a vyhýbá se a skáče přes sudy, které na něj Donkey Kong hází. (Klappenbach, 2021)

Jako další přišly na řadu plošinové hry pod názvem „Side-scroller“. U těchto her je možné se pohybovat po mapě do všech směrů. První hrou, která tento systém pohybování představila, je hra pod názvem „Jump Bug“, která vyšla roku 1981, jen pár měsíců po hře „Donkey Kong“. Mezi nejznámější hru tohoto typu se dá řadit „Super Mario Bros“. Tato hra vyšla roku 1985 a stala se archetypem pro mnoho dalších plošinových her. V této hře uživatel kontroluje postavu Maria a snaží se dostat k vlajkám na konci úrovní. Cestou

k vlajce se musí vyhýbat různým nástrahám a nepřátelům a může sbírat mince, či další vylepšení postavy. (Klappenbach, 2021)

Nástup 3D her v plošinovém žánru odstartovala hra pod názvem „Super Mario 64“ roku 1996 pro konzoli „Nintendo 64“. Tato hra nastavila standard pro všechny pozdější 3D plošinové hry. Úrovně se vyskytovaly na otevřené mapě a podporovaly volbu vlastní cesty. Mnoho vývojářů uvádí tuto hru, jako inspiraci při tvoření nových her. Zároveň se v této hře vyskytuje dynamický kamerový systém a 360stupňové analogové ovládání, které bylo velmi lehké použít, a i proto je často tato hra označována za nejlepší v historii. (Gaming, 2022)

Hra „Super Mario 64“ si získala kultovní oblibu a dodnes je velká převážně ve „speedrunning“ komunitě. Lidé z této komunity se snaží nastavovat rekordy v tom, jak rychle tuto hru dokážou dokončit. (Gaming, 2022)

Jednou z největších záhad při stanovení rekordu, se stala uživatelovi se jménem DOTA_Teabag. Na úrovni s názvem „Tick Tock Clock“ se uživatelovi povedlo teleportovat na římsu v lokaci, kde to jinak bylo považováno za nemožné a tím si zkrátil čas o několik sekund. Ihned po nahrání videa byla uživatelem pannenkoek12 vydána 1000$ odměna pro toho, kdo dokáže tento trik zopakovat. Video bylo mnohokrát zanalyzováno a mnoho uživatelů se pokusilo tento trik zopakovat. I po přesné replikaci veškerých kláves, které DOTA_Teabag stiskl, tak nemohl tento trik být zopakován. Došlo se tedy k neuvěřitelnému zjištění, při hraní nastal fenomén s názvem „single-event upset“. Při tomto fenoménu se změní hodnota binárního stavu v bitu, buďto z 0 na 1, nebo opačně. Stane se tomu tak při tom, když ionizující částice zasáhne citlivé elektronické zařízení. K tomu dochází v důsledku výboje v paměťových bitech způsobeného ionizací částice v blízkosti uzlu. Když kosmické částice proniknou do zemské atmosféry, narazí na atomy atmosféry a vznikne déšť protonů a neutronů, který může poškodit elektronická zařízení. Většinou jsou následky sotva patrné, a přestože zasažená oblast nemá velký význam, tak tento příklad byl velmi nápadný. Ionizující částice z vesmíru narazila během závodu do počítače N64 uživatele DOTA_Teabag a invertovala osmý bit prvního bajtu Mariovy výšky. Přepnula bajt z 11000101 na 11000100, neboli z "C5" na "C4". To způsobilo změnu výšky z C5837800 na C4837800, což bylo shodou okolností přesně tolik, kolik bylo v té době potřeba k tomu, aby se Mario dostal do vyššího patra. Po zjištění, jak tento fenomén mohl nastat, tak to bylo otestováno pomocí skriptu, který manuálně přepnul bit v tom přesném okamžiku a výsledky tohoto testu to tím potvrdily. (Burtt, 2020)

2.1.1 Hry určené pro inspiraci

Tato práce je inspirována převážně hrou Hollow Knight a hrou Ori and the Will of the Wisps, které se svými prvky řadí mezi akční plošinové hry.

Hollow Knight je 2D akční plošinová hra ve stylu „Metroidvania“. Styl „Metroidvania“ je složením z názvů herních sérií „Metroid“ a „Castlevania“. Hry tohoto typu využívají herní mechanismy, podobné v hrám z těchto sérií a soustředí se na prozkoumávání otevřené mapy, hledání tajemství a hledání schopností, která umožní se dostat do dříve nepřístupných oblastí. Tato hra vyniká především svým rozmanitým příběhem a vyprávěním tohoto příběhu pomocí detailní mapy podzemního království „Hallownest“. Oblasti na sebe navazují tematicky s logikou, která se odhaluje až při průzkumu. Příběh je postaven na mizející historii tohoto království, jejíž detaily nejsou ze začátku nikde vysvětleny, místo toho se nachází pouze pár lokací o málo lidech, kteří zde ještě žijí. Průběhem hry a sbíráním sběratelských předmětů se mlha této historie rozpouští a tím se odemyká více detailů o pádu, tohoto dříve velkolepého království. Hra se zároveň velmi soustředí na střety s tzv. „bossy“, neboli nepřáteli, kteří jsou obzvlášť nároční na poražení a kteří většinou blokují postup hrou. Tito „bossové“ dále doplňují detaily o životě v tomto království a o lidech v něm. (“Hollow Knight Review,” 2017)

Ori and the Will of the Wisps je esteticky působivá a emocionálně zajímavá akční plošinovka. Hra je přímým pokračováním hry "Ori and the Blind Forest" a pokračuje v dojemném příběhu Oriho, strážného ducha, který se vydává na cestu, aby uchránil svůj lesní domov před řadou blížících se pohrom. Sprity této hry jsou ručně kreslené a obsahují spoustu detailů o objektech a příbězích. Uživatelé při vedení hlavní postavy zažijí hádanky, plošinové části a souboje s "bossy". Hru po celou dobu doplňuje bohatá symfonická hudba. Základní herní prvky zahrnují plynulé a dynamické herní mechanismy a pohyb hlavní postavy je svižný a plynule reaguje na vstupy z klávesnice, což uživatelům umožňuje procházet různorodou krajinou a zdolávat obtížné skoky na platformy. Hra také disponuje řadou schopností, které hlavní postava získává během svého putování hrou. Hlavním rysem této hry je její děj, v němž vystupují fascinující postavy, které umocňují zážitek z příběhu. (“Ori And the Will of the Wisps Game Review,” 2020)

3. Počítačová grafika

Grafika má obrovský vliv a podstatu, ať se jedná o ručně namalované sprity, či o modely, které se snaží co nejvíce přiblížit reálnému životu. Dnešní hry jsou založeny především na tématech, to znamená že hra nemusí mít nejlepší grafiku ve stylu toho, že efekty a modely se přibližují reálnému životu, ale i hra s jednoduchými sprity může dosáhnout velké komunity. Grafika má obrovský vliv a podstatu, ať se jedná o ručně namalované sprity, či o modely, které se snaží co nejvíce přiblížit reálnému životu. Zároveň ale grafiku nelze zanedbat, a i u her, které jsou založeny na „pixel art“ je nutné svět obohatit o detaily a o speciální efekty. (Sayers, 2022)

3.1 2D počítačová grafika

2D grafika má dlouhou historii v herním světě, ačkoliv byla časem částečně vytlačena 3D grafikou, stále si udržuje své místo v srdcích hráčů. V počátcích videoher byly 2D hry s bočním posouváním nejpopulárnější formou zábavy. Avšak s příchodem 3D her se grafický průmysl obrátil k novým dimenzím a 2D grafika ustoupila do pozadí. Přesto i v současnosti jsou 2D herní aplikace stále významné a mají stále velkou oblibu. Tento trend je dobře patrný i u herních konzolí, jako je Xbox od společnosti Microsoft, PlayStation od společnosti Sony a Switch od společnosti Nintendo, které přinášejí zpět do herního světa staré arkádové hry a konzolové tituly v podobě her ke stažení. V současné době lze na těchto platformách nalézt i nové a inovativní 2D hry, které nabízejí zábavu i pro novou generaci hráčů. 2D grafika tak stále zůstává nezbytnou součástí herního průmyslu a svým charakteristickým stylem oslovuje hráče po celém světě. (Küçükkarakurt, 2021)

V tvorbě 2D počítačové grafiky umělec kombinuje tvary, barvy a čáry do obrazu určeného k zobrazení na obrazovce počítače. Tento typ obrázku je vnímán pouze z jednoho úhlu, ať už zezadu, ze strany, zepředu nebo z výšky postavy, avšak postava se stále jeví jako plochá. Tímto úhlem a vhodným stínováním kolem postavy může být vytvořena jemná iluze trojrozměrnosti, i když obraz zůstává stále plochým. Přestože 2D počítačová grafika omezuje prostorovou hloubku, vyniká svou jednoduchostí a estetickým ztvárněním. Umožňuje umělcům vyjádřit svou kreativitu a vytvářet nádherné vizuální umění, které je ideální pro animace, ilustrace, a další grafické projekty. Ačkoliv 3D grafika nabízí realističtější zážitek, 2D počítačová grafika si stále udržuje své místo v digitálním světě díky svému unikátnímu charakteru a schopnosti přenášet výraz a emoce skrze jednoduchost linií a barev. (“What Is a 2D Computer Graphic?,” 2023)

3.1.1 Rastrová grafika

Rastrová grafika, známá také jako bitová mapa, představuje způsob ukládání obrazových dat jako pevných pixelů. Každý pixel reprezentuje jednu konkrétní barvu a je umístěn v mřížce, která se nazývá "rastr", odtud pochází i název. Tento přístup umožňuje vytvářet přehledné obrazy, podobně jako mozaiku, když se krokem zpět shromáždí všechny tyto barevné pixely. Pomocí rastrové grafiky lze detailně zachytit vzhled a strukturu obrázku, a to včetně jemných barevných odstínů a nuancí. Každý pixel přispívá k celkovému vizuálnímu dojmu a umožňuje vytvořit poutavé, realistické, nebo i umělecky stylizované obrázky. Díky své jednoduchosti a efektivitě se rastrová grafika stala populárním formátem pro digitální obrázky a využívá se ve webových designech, videohrách, animacích, a mnoha dalších oblastech. Přestože má své omezení v ohledu škálování a zachování kvality při zvětšení, rastrová grafika stále zůstává jednou z nejčastěji používaných metod pro tvorbu a ukládání digitálních obrázků. (Weinreb, b.r.)

Pokud například zvětšíme rastrový obrázek, aniž bychom změnili jeho rozlišení, kvalita se ztratí a obrázek se jeví jako rozmazaný nebo rozpixelovaný. K tomu dochází proto, že pixely jsou roztaženy na větší oblast, což vede k rozmazání a snížení ostrosti. To je typický problém při úpravě rastrových obrázků, i když se mu lze vyhnout použitím specializovaných programů pro úpravu rastrových obrázků, jako je například Photoshop. Tyto programy umožňují upravit rozlišení obrázku a správně jej škálovat, čímž se zachová čistota a ostrost obrázku i při zvětšení. Díky nim lze zachovat vysokou kvalitu rastrových obrázků a zároveň je přizpůsobit více velikostem a formátům. (“Research Guides: All About Images: Raster Vs. Vector Images,” b.r.)

Výhodou obrázků s vysokým rozlišením je vyšší hustota pixelů, protože se na jeden palec používá méně pixelů, označovaných jako DPI (dots per inch) nebo PPI (pixels per inch). To jim umožňuje zachytit více detailů a dosáhnout vynikající kvality obrazu, zejména v případě fotografických snímků. Rastrové obrázky s vysokým DPI využívají méně pixelů, což umožňuje umístit na stejnou plochu širší škálu barev. Vysoce kvalitní obrázky mají například rozlišení 300 DPI nebo vyšší, které poskytuje ostré a složité vizuály, zatímco obrázky se standardním rozlišením mají obvykle rozlišení kolem 72 DPI. Vyšší DPI je nezbytné pro profesionální grafiku, tisk a jakékoli aplikace, které upřednostňují zachování maximální kvality obrazu. (Weinreb, b.r.)

3.1.2 Vektorová grafika

Vektorová grafika je vytvářena pomocí bodů, čar a tvarů, a matematické rovnice následně říkají počítači, jak nakreslit výsledný obrázek. I když se technicky jedná o 2D grafiku, je možné dodat obrázku 3D vzhled prostřednictvím vrstvení různých prvků a použitím textur. Nejběžněji používaným formátem vektorové grafiky je .SVG, ale lze se také setkat s formáty .eps, .cgm, .odg a .xml. (iStock Blog, 2023)

Grafičtí designéři vytvářejí vektorové umění přidáním bodů, čar a tvarů na plátno ve svém preferovaném programu pro vektorový design. Při otevření vektorového souboru vektorovým softwarem jsou tato uložená data zpětně převedena na obrazová data a obrázek je přesně vykreslen podle uložených souřadnic. Tento proces zajišťuje, že vektorový obrázek zůstává ostrý a kvalitní, bez ohledu na jeho velikost, což je klíčová výhoda vektorové grafiky. To umožňuje využívat vektorová umění v různých projektech, od malých ikon a log po velké tiskoviny, aniž by došlo ke ztrátě kvality. Vektorová grafika se stále stává nedílnou součástí profesionálního grafického designu a nabízí designérům mnoho možností pro kreativní tvorbu. (“How Do Vector Graphics Work? | CorelDRAW,” b.r.)

Při zvětšení kruhu, stačí jednoduše zvětšit hodnotu poloměru r pomocí geometrické rovnice $(x-h)^2+(y-k)^2=r^2$. Namísto sledování hromady dalších pixelů, počítač pouze upraví toto číslo, což zabere jen minimální místo v souboru. Tímto způsobem se zachovává dokonalá ostrost a kvalita obrázku bez ohledu na jeho velikost. (“Research Guides: All About Images: Raster Vs. Vector Images,” b.r.)

4. Software použitý pro vývoj

4.1 JetBrains Rider

IntelliJ Rider od společnosti JetBrains je integrované vývojové prostředí (IDE) určené pro vývojáře aplikací.NET a ASP.NET s multiplatformní podporou pro systémy Windows, macOS a Linux. Rider zvyšuje produktivitu a kvalitu kódu díky rozsáhlé analýze kódu, označování chyb a automatickému refaktoringu. Rider běží jako samostatný proces, přičemž front-end IntelliJ komunikuje s back-endem ReSharperu pomocí protokolu Rider, což poskytuje izolovaný paměťový prostor a lepší výkon. Rider zároveň podporuje vizuální čistotu, všechna menu obsahují méně položek, které již jinak nejsou důležité. Toto IDE vyniká při tvorbě her Unity, podporuje Xamarin a spolupracuje se systémy pro správu verzí, jako je Git. Podporuje webové technologie, jako jsou HTML, CSS a JavaScript, pro vývoj celého balíku aplikací. Jetbrains zároveň podporuje licenci zdarma pro studenty, přičemž tato licence byla použita pro práci na této hře. (CODE Magazine, EPS Software Corp., Chris Woodruff and Maarten Balliauw, b.r.)

4.2 Herní engine Unity

Unity je známý herní engine a integrované vývojové prostředí (IDE), které se hojně využívá v herním průmyslu. Tento je na trhu od roku 2005 a od té doby nabral velikou komunitu a širokou škálu velmi známých her, které v tomto enginu byly vytvořeny. Díky vysoké flexibilitě a jednoduchému použití se tento engine hodí jak pro začínající, tak i zkušené vývojáře. Odlišností Unity od ostatních herních enginů je dynamický "Asset Store", kde mohou tvůrci přistupovat k rozmanitému výběru assetů vytvořených komunitou, od vizuálních efektů po ovládací skripty, což dále usnadňuje proces tvorby. Tyto assety lze od tvůrců koupit pod různými licencemi, nebo je také k výběru vysoké množství assetů zdarma, u kterých se ale licence také liší. Díky veliké komunitě lze proto najít mnoho online tutoriálů na různé problémy, na které lze při vývoji hry narazit. Tyto tutoriály lze najít jak na oficiálních fórech Unity, tak na Youtube a třeba i na stránce Reddit. (Sinicki, 2021)

Unity editor disponuje lehce srozumitelné architektuře. Každá úroveň ve hře se dá rozložit do scén v Unity editoru, každá tato scéna pak obsahuje všechny potřebné herní objekty. Zároveň tento engine poskytuje rodičovský vztah v hierarchii mezi jednotlivými objekty a lze tedy lehce přidat několik objektů pod jeden hlavní objekt. Unity editor zároveň obsahuje inspektor, pomocí kterého lze vlastnosti objektů měnit za běhu hry

a pomocí toho zkoušet jaké hodnoty fungují nejlépe bez nutnosti restartování . Další důležitou součástí je Unity Scripting API. Tento engine obsahuje obsáhlé aplikační programovací rozhraní pro skriptování, které poskytuje rychlý přístup ke všem nejvíce používaným funkcím. Umožňuje tak s lehkostí vytvářet chování postav a herní mechanismy. K tomuto API je také přiřazena obsáhlá dokumentace, kde lze všechny tyto informace dohledat. (Zenva, 2023)

Unity poskytuje vývoj pro mnoho platform jako je Android, Windows či iOS a zároveň podporuje vývoj pro až 27 unikátních platform. V posledních letech s novými technologiemi se to stal preferovaný nástroj při vývoji aplikací pro virtuální realitu a augmentovanou realitu. Na vývoj aplikací v tomto enginu spoléhají i společnosti jako je Google nebo NASA. (freeCodeCamp.org, 2021)

4.3 Jazyk C#

Jazyk C# je moderní, objektově orientovaný a typově bezpečný programovací jazyk, který vývojářům umožňuje vytvářet bezpečné a robustní aplikace.NET. Vychází z rodiny jazyků C, takže jej budou znát programátoři v jazycích C, C++, Java a JavaScript. Jazyk C# podporuje hodnotové a referenční typy, nulovatelné typy, zpracování výjimek, lambda výrazy a Language Integrated Query (LINQ). Klade důraz na verzování pro zajištění dlouhodobé kompatibility a pracuje v prostředí .NET, přičemž využívá běhové prostředí CLR (Common Language Runtime) a sadu knihoven tříd. Prostory názvů, typy a členy slouží k organizaci programů v jazyce C#, které jsou sdruženy do sestav pro spuštění. (BillWagner, 2023)

4.4 Leonardo AI

Leonardo AI je generativní AI pro vytváření obrázků, s hlavním cílem vytvářet textury, assety a koncepční umění do her. Leonardo AI podporuje mnoho typů modelů, včetně modelů od uživatelů pro tvoření různých typů obrázků na základě stylu a detailů. Stránka obsahuje intuitivní a přívětivé rozhraní, pomocí kterého lze rychle začat tvořit vlastní obrázky. S komunitními modely je toto AI perfektní pro unikátní assety do her. Pro přístup do této aplikace je potřeba se nejdříve dostat do listu osob, které mají přístup přes registraci na hlavní stránce. V posledních měsících tato aplikace rozšiřuje svoje možnosti a kapacitu, a tak je tento přístup udělován celkem rychle. Leonardo dále podporuje mnoho nástrojů, jako je například trénovaní vlastních modelů, tvoření textur pro 3D objekty a AI

canvas, pomocí kterého lze rozšiřovat již vytvořené obrázky. Tato aplikace je stále v aktivním vývoji a přidává mnoho nových nástrojů každým měsícem. (Ramlochan, 2023)

4.5 Photoshop

Adobe Photoshop je všestranný softwarový nástroj, který se hojně používá k úpravám obrázků a retušování fotografií na platformách Windows i MacOS. Jeho rozsáhlá nabídka nástrojů umožňuje designérům, fotografům, tvůrcům webových stránek a kreativním profesionálům snadno vytvářet, vylepšovat a upravovat obrázky. Mezi rozmanitou řadu produktů vyniká Photoshop CC jako verze pro profesionály, která je určena pokročilým uživatelům, zatímco Photoshop Elements se zaměřuje na nováčky v oboru. Na druhou stranu se Photoshop Lightroom ukazuje jako neocenitelný nástroj pro efektivní zpracování velkých dávek snímků prostřednictvím nedestruktivních úprav. Rozsah funkcí, které Photoshop nabízí, zahrnuje úpravy obrázků, manipulaci s barvami, kreslení, integraci textu, organizaci obrázků a všestranné možnosti exportu pro různé formáty souborů. Od svého vzniku v roce 1987 se Photoshop neustále vyvíjí a každá nová verze přináší inovativní funkce a vylepšení. Zejména přechod společnosti Adobe na model předplatného Creative Cloud vedl k přechodu z verze Photoshop CS na pokročilejší verzi Photoshop CC, což dále upevnilo jeho postavení předního softwaru v oboru. (Smith, b.r.)

4.6 Github

GitHub je cloudová služba a webová stránka, která poskytuje vývojářům platformu pro ukládání, správu a sledování změn jejich kódu. Ve své podstatě se GitHub opírá o dva základní principy: správu verzí a systém Git. Správa verzí je klíčová pro sledování a správu změn kódu softwarového projektu v průběhu jeho růstu. Místo přímých úprav hlavního zdrojového kódu mohou vývojáři bezpečně pracovat na konkrétních částech projektu vytvářením větví a jejich slučováním zpět do hlavní kódové základny, jakmile jsou změny úspěšné. Cloudová infrastruktura zajišťuje snadný přístup k úložištím z jakéhokoli zařízení, což vývojářům na cestách poskytuje jedinečné pohodlí. Kromě toho umožňuje GitHub vývojářům prezentovat své projekty a příspěvky, čímž se jejich profily mění v působivá portfolia. (Juviler, 2022)

Základem GitHubu je Git, open-source systém pro správu verzí, který v roce 2005 vytvořil Linus Torvalds. Jedná se o distribuovaný systém správy verzí, který zajišťuje, že každý vývojář má přístup k celé kódové základně a historii na svém počítači, což

usnadňuje větvení a slučování. Podle průzkumu Stack Overflow používá systém Git více než 87 % vývojářů. (Kinsta, 2022)

Samotný GitHub je ziskovou společností nabízející cloudové služby hostování úložišť Git, což jednotlivcům i týmům umožňuje pohodlně využívat systém Git pro správu verzí a spolupráci. Jeho uživatelsky přívětivé rozhraní jej zpřístupňuje i začínajícím programátorům a odstraňuje potřebu rozsáhlých technických znalostí a používání příkazového řádku, které jsou spojeny pouze se systémem Git. Jednou z nejatraktivnějších vlastností služby GitHub je, že se do ní může zaregistrovat kdokoli a hostovat veřejný repozitář kódu zdarma, díky čemuž je nesmírně populární pro projekty s otevřeným zdrojovým kódem. (Kinsta, 2022)

5 Herní komponenty

5.1 Skript

Skript označuje soubor obsahující vlastní kód napsaný ve skriptovacím jazyce. Skript je v podstatě část textu, který udává instrukce objektům ve hře. Těmito instrukcemi může být např. přesouvání herních objektů a změny ve velikosti objektů, ale i také složitější pohyby hlavní postavy a chování nepřátel. Při psaní skriptu se musí také dbát na správné jmenování proměnných a dalších názvů, aby se při budoucí návštěvě skriptu dalo vyvodit, co ten skript dělá. Při psaní skriptů v Unity v C# je nejlepší praktikou použít jmenovací konvekce „PascaleCase“ a „camelCase“. (Christensson, 2023)

5.2 Sprite

Sprity jsou 2D bitmapové grafické obrázky, které se dají použít pro různé objekty ve hře, jako jsou postavy, projektily a další elementy. Tyto sprity se ale také dají použít ve 3D hrách, pro zlehčení nutného grafického výkonu počítače, použití lze nalézt např. na trávě či objektech v dálce, kam uživatel dohlédne, ale nemá se tam jak dostat. Sprity pak lze také shromáždit do tzv. „sprite atlasů“, jedná se o jeden obrázek se všemi tyto sprity dohromady, program jej pak dokáže automaticky, nebo za pomoci uživatele oddělit zpět na jednotlivé sprity. (“Sprite (Concept) - Giant Bomb,” b.r.)

5.3 Prefab

Pokud je potřeba použít herní objekt vícekrát na mapě, tak je užitečné z tohoto objektu vytvořit tzv. „prefab“. Tento objekt je užitečný právě když je unikátně nakonfigurován nebo když je složen z více spritů. Může se tak jednat například o různé postavy nebo části terénu. Pomocí těchto prefabů je pak možné vytvářet změny v několika objektech najednou, i když jsou roztroušené na různých místech mapy. Pokud je ale potřeba, aby se nějaká instance prefabu lišila od ostatních, tak jej lze přepsat. Používání prefabů zjednodušuje vytváření částí mapy a tím zkracuje i čas potřebný k jejímu vytvoření. (Unity Technologies, 2018)

6 Vývoj aplikace

Implementace hry, která je popisována v této práci, byla dosažena pomocí herního enginu Unity, přesněji pomocí verze 2021.3.20f1. Tento engine obsahuje velmi intuitivní prostředí, dostává velmi obsáhlou podporu od komunity vývojářů, k enginu ke dodávána podrobná dokumentace, která obsahuje všechny potřebné informace k problémům, na které lze narazit a zároveň je s Unity enginem propojen Unity Asset Store, který obsahuje mnoho placených a neplacených assetů, které se mohou stáhnout a využít v aplikaci. Díky všem těmto vlastnostem byl pro tuto práci zvolen právě Unity engine.

Při vytváření nového projektu je nejdříve potřeba si zvolit šablonu, lze si zvolit z několika šablon, pro tuto práci je zvolena šablona 2D, která je nakonfigurovaná pro vývoj 2D hry, ale neobsahuje žádné věci navíc. Při otevření projektu si lze všimnout 4 oken, se kterými se pracuje. Mezi tyto okna se řadí hierarchie, v tomto okně se zobrazuje struktura objektů. Je možné také vytvářet prázdné objekty pod které jdou vložit další objekty, tento způsob řazení může sloužit k lepší orientaci v projektu. U nového projektu si lze všimnout, že je automaticky vytvořena nová scéna pod názvem „SampleScene“ a v této scéně je obsažena hlavní kamera pod názvem „MainCamera,“ která zobrazuje vše, co je obsaženo v projektu. Po označení objektu se v okně inspektoru zobrazí všechny vlastnosti, které lze měnit, zároveň se zde může pracovat s komponenty, které mohou být přiřazeny objektu. Tyto objekty mohou být individuálně upravovány, ale také je možné označit několik objektů stejné struktury a měnit je zároveň. Jako další okno je okno projektu, zde je možné si zobrazit všechny assety, které jsou v projektu a také balíčky, které se v projektu využívají. Zároveň je zde položka „oblíbené“ kde si lze zobrazit všechny nejvíce používané materiály, modely a prefaby. Jako poslední okno je hlavní okno scény, zde se zobrazuje vše, co je obsažené v aplikaci, lze zde posouvat s objekty a přímo vidět změny provedené v inspektoru.

6.1 Pohyb hlavní postavy

6.1.1 Běhání

Ve hře je potřeba aby se hlavní postava dokázala pohybovat do všech směrů, tj. nahoru, dolů, doprava, doleva jejich kombinaci. Zároveň je potřeba upřesnit rychlost postavy, skok a další různé schopnosti. Kvůli těmto nárokům je potřeba vytvořit novou část kódu ve formě skriptu. Pro pohyb postavy se nejdříve potřebuje zjistit, jestli se uživatel chce pohybovat nebo ne. Existuje několik způsobů, jak tohoto dosáhnout, avšak v Unity se dá použít funkce „Input.GetAxisRaw()“, do závorek se poté jako parametr dá zvolit námi požadována osa ve formátu string. Tato metoda poté vrací tři možné hodnoty: -1 označuje pohyb doleva, 0 označuje žádný pohyb a 1 označuje pohyb doprava. V projektové nastavení v položce „Input manager“ lze specifikovat pomocí jakých kláves se může postava posouvat doprava, či doleva. „Input.GetAxis()“ existuje jako alternativní řešení k „Input.GetAxisRaw()“, přičemž místo toho, aby tato hodnota vracela pouze tři hodnoty, tak vrací hodnotu float od velikosti -1 po 1, tato hodnota se s postupem času vyhlazuje a je tedy vhodná pro plynulý a nepřetržitý pohyb.

K samotnému pohybu postavy je již použito několik proměnných, jako první se musí přidat objektu komponent „RigidBody2D“, v tomto komponentu se dá měnit několik proměnných, o které se již stará samotný engine Unity, mezi tyto parametry patří např. váha objektu, lineární odpor a síla gravitace. Tento komponent je poté v kódu zastoupen pomocí proměnné „_rb“. Další důležitá proměnná pro pohyb je „moveVelocity“, tato proměnná udává, jestli se postava má pohybovat doleva, či doprava a to tím, že se tato hodnota mění, přičemž Vector3.left udává hodnotu 1 a znamená doprava a Vector3.right udává hodnotu -1 a znamená doleva. Další proměnná, pomocí které přímo kontrolujeme rychlost postavy se nazývá „movePower“, tato proměnná je public a lze tedy zároveň měnit rychlost postavy přímo v inspektoru v Unity editoru. Tyto proměnné se poté vloží do rovnice a postava se tímto začne pohybovat.

Výpis kódu 1: Pohyb postavy

```
private void Run()
{
    Vector3 moveVelocity = Vector3.zero;
    _anim.SetBool( id: IsRun,  value: false);


    if (Input.GetAxisRaw("Horizontal") < 0)
    {
        _direction = (float) -0.5;
        moveVelocity = Vector3.left;

        transform.localScale = new Vector3( x: _direction,  y: (float) 0.5,  z: 1);
        if (!_anim.GetBool( id: IsJump))
            _anim.SetBool( id: IsRun,  value: true);
    }

    if (Input.GetAxisRaw("Horizontal") > 0)
    {
        _direction = (float) 0.5;
        moveVelocity = Vector3.right;

        transform.localScale = new Vector3( x: _direction,  y: (float) 0.5,  z: 1);
        if (!_anim.GetBool( id: IsJump))
            _anim.SetBool( id: IsRun,  value: true);
    }

    transform.position += moveVelocity * (movePower * Time.deltaTime);
}
```

Zdroj: Autor (2023)

6.1.2 Skákání

Jako další možnost pohybu je skok, pro tuto funkci se potřebuje zjistit, zda uživatel zmáčkl specifické klávesy a tím se dosáhne pomocí funkce „Input.GetButtonDown()“ a názvu příslušného tlačítka ve formě string. Tato funkce kontroluje, jestli uživatel zmáčkne klávesy, které je možné v „Input Manager“ definovat. Postava má zároveň dovoleno několik skoků pomocí proměnné „_extraJumps“ a „extraJumpsValue“, přičemž druhá proměnná má atribut „[SerializeField]“ pomocí kterého lze měnit počet skoků přímo v editoru. Proměnná „_extraJumps“ převezme počet skoků z „extraJumpsValue“ při zapnutí programu, přičemž pokaždé, když uživatel provede skok a počet skoků je zároveň vyšší jak 0, tak se od této hodnoty odečte 1, toto je dosaženo pomocí „_extrajumps--“. Jakmile se postava dotkne země, tak je toto „počítadlo“ vyresetováno a je mu zpět přiřazen počet skoků dán v „extraJumpsValue“. Pro samotný skok je potřeba definovat sílu skoku, toho lze dosáhnout proměnnou „jumpPower“. Ke skoku existuje podmínka,

kterou lze vidět ve výpisu kódu 2, pokud kód přes tuto podmínku projde, pak se „Rigid Body 2D“ které je v kódu nahrazeno „_rb“ rozpohybuje tím, že „Vector2.up“, který nám udává směr nahoru od 0 do 1, se vynásobí s „jumpPower“ a následně je proveden skok.

Výpis kódu 2: Metoda umožňující skok

```
private void Jump()
{
    if (Input.GetButtonDown("Jump") && _extraJumps > 0)
    {
        _rb.velocity = Vector2.up * jumpPower;
        _extraJumps--;
        _anim.SetBool( id: IsJump,  value: true);
```

Zdroj: Autor (2023)

Mezi první problém, na který šlo narazit, se týkal počtů skoků za sebou, jelikož toto nebylo nijak hlídáno, tak uživatel mohl skákat kolikrát chtěl pouze pomocí toho, že stiskl tlačítko pro skok několikrát. Pro vyřešení tohoto problému je potřeba zjistit, jestli se postava dotýká vrstvy, od které se její počítadlo se skoky vyresetuje, ale k tomu je nejdříve potřeba definovat, jaká tato vrstva je. Tohoto lze dosáhnout tím, že se v inspektoru klikne na „Add Layer“ a vytvoří se nová vrstvu s názvem „Ground“. Poté je potřeba tuto vrstvu jednotlivě přidělit každému objektu který má být definován jako země. Jako další je potřeba těmto objektům přiřadit buďto „Box Collider 2D“ nebo „Edge Collider 2D“. Pro úseky, které jsou ploché stačí použít „Box Collider 2D“ jelikož toto vytvoří obdélníkový tvar, pro více složité tvary je potřeba použít „Edge Collider 2D“. Tyto collidery je potřeba použít kvůli přesnému ohraničení masky „Ground“. U postavy je dále potřeba určit, jestli se dotýká této masky, toho lze dosáhnout pomocí metody _isGrounded(), tato metoda je typu bool a vrací proto buď true, nebo false. K určení, jestli metoda _isGrounded() má vracet true nebo false je použito „Physics2D.OverlapCircle()“. Tato metoda vytváří neviditelný kruh, který je umístěn pod postavou a kontroluje, zda se dotýká určité vrstvy. Této metodě se předávají parametry: „groundCheck.position“ udává lokaci kruhu, „checkRadius“ udává poloměr tohoto kruhu a „whatIsGround“ definuje vrstvu, u které probíhá kontrola na kolizi.

Výpis kódu 3: Kontrola dotyku masky

```
private void FixedUpdate()
{
    _isGrounded = (bool)Physics2D.OverlapCircle((Vector2)groundCheck.position, checkRadius, (int)whatIsGround);
```

Zdroj: Autor (2023)

6.1.3 Speciální pohyb „Dash“

Mezi speciální pohyb se dá zařadit „dash“. Na mapě jsou úseky, přes které je velmi těžké se dostat, nebo přes které se nedá dostat pouhými skoky, proto je mezi speciálními pohyby i takzvaný „dash“. Jakmile uživatel zmáčkne klávesu levý shift a má možnost provést tento pohyb, tak se postava rychle posune o určitou vzdálenost v krátkém časovém úseku. Toto je kontrolováno pomocí funkce „Input.GetKeyDown(KeyCode.LeftShift)“, proměnné „canDash“ a proměnné „isDashing“. Postava se může pohybovat na jakýkoliv směr, toto je kontrolováno pomocí metody „dashDirection“ a dvou proměnných, které jsou k tomu potřeba, tím je „horizontalInput“ a „verticalInput“. Do těchto proměnných se zjistí hodnota chtěné osy pomocí metody „Input.GetAxis()“, přičemž jako parametr se zvolí název osy ve formátu string. Při provedení tohoto pohybu se zapne časovač s proměnnou „dashCooldown“, po uběhnutí času, který lze nastavit přímo v inspektoru editoru, proběhne metoda „ResetDash“, kde se proměnná „canDash“ přepne na true. Na výpisu kódu 4 lze vidět výpočet pohybu postavy a rychlost tohoto pohybu. Při provedení tohoto pohybu je zároveň gravitace postavy přepsaná na 0, aby se při delších pohybech postava pohybovala pouze v jednom směru, toto je zařízeno pomocí „rb.gravityScale“. Aby došlo k iluzi plynulejšího pohybu je také změněna vlastnost „rb.drag“, tato vlastnost udává odpor vzduchu, který působí na objekt a brzdí jeho pohyb. Zvýšením této hodnoty začne působit síla proti směru objektu a začne ho brzdit. Po ukončení pohybu se zavolá metoda „EndDash“, čas, po kterém se má zavolat je kontrolována pomocí „dashDuration“. Tato metoda zajistí, že vlastnosti gravitace a odporu vzduchu se vrátí zpět na hodnoty, které byly přítomny před provedením tohoto pohybu.

Výpis kódu 4: Speciální pohyb „Dash“

```
private void DashMove()
{
    if (!isDashing)
    {
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");
        dashDirection = new Vector2( x: horizontalInput,  y: verticalInput).normalized;

        isDashing = true;
        canDash = false;
        Invoke(nameof(ResetDash), dashCooldown);

        rb.velocity = dashDirection * dashDistance / dashDuration;
        rb.gravityScale = 0;
        rb.drag = 10;

        Invoke(nameof(EndDash), dashDuration);
```

Zdroj: Autor (2023)

6.1.5 Speciální pohyb „WallSlide“ a „WallJump“

Jako další speciální pohyb je „WallSlide“. Na mapě jsou také úseky, kde je potřeba, aby uživatel provedl velmi přesné skoky, nebo úseky na kterých je nutné, aby uživatel tento pohyb využil. Pomocí tohoto pohybu se hlavní postava „přilepí“ ke zdi poté co k ní dojde v kontakt, dokud se hlavní postava dotýká zdi buďto levou, či pravou stranu, tak dochází ke zpomalení postavy a tím pádem postava padá pomaleji a dochází ke „skluzu“ po zdi. Pro dosažení tohoto chování je nejdříve potřeba vytvořit novou vrstvu s názvem „Wall“, všem objektům, které slouží jako zeď je potřeba tuto vrstvu přiřadit. Poté je potřeba kontrolovat, zda postava přijde do styku se zdí, tohoto je dosaženo pomocí podmínky, která je ukázána ve výpisu kódu 5. Pokud kód přes podmínku projde, pak proběhne metoda pomocí které se postava zpomalí, to je zde vypsáno pomocí „rb.velocity“. Ke zpomalení proběhne tím, že se hodnoty rychlosti osy x a osy y, které právě probíhají v „Rigid Body 2D“ dosadí do nového vektoru, ale vertikální hodnota se vynásobí se „slideMultiplier“, tato hodnota se také může měnit přímo v editoru. Výsledkem je nová vertikální rychlost hlavní postavy.

Na některých úsecích je potřeba se dostat na bod, který je však příliš vysoko a na cestě nejsou žádné plošiny s vrstvou „Ground“ a není teda možné, jak se tam dostat pomocí skoků a ani pomocí speciálního pohybu „Dash“. V těchto úsecích je potřeba využít nový pohyb s názvem „WallJump“. Tento pohyb je zakomponován ve skriptu kde se nachází pohyb „WallSlide“ a tyto dva pohyby fungují ruku v ruce. Pokud se hlavní postava nachází v kolizi s objektem s vrstvou „Wall“, pak se počet skoků přepíše na „extraJumps“, tato hodnota se dá přepsat v Unity editoru. Tímto způsobem může uživatel vyskákat i části mapy, přes které by to jinak nebylo možné.

Aby tyto pohyby vypadali realisticky a zároveň dávali smysl v kontextu této herní aplikace, tak i zde se provádí animace obou speciálních pohybů. Jelikož uživatel může narazit na zdi, které jsou jak vlevo, tak vpravo od hlavní hráčské postavy, tak je nutno kontrolovat, kde je jak hráč, tak i stěna a jakou stranou k sobě oba objekty čelí. Svět v editoru je rozdělen na tři osy: osu x, osu y a osu z. Jelikož hra probíhá pouze ve dvourozměrné rovině, tak se může ignorovat osa z bez jakýchkoliv problémů. Ve výpisu kódu 5, lze vidět, že při pohybu „WallSlide“ se začne přehrávat animace pomocí funkce „_anim.SetBool()“, dále jsou zde 2 podmínky a to pro případ toho, pokud hráč má čelit vpravo, nebo vlevo. Proměnná „wallPositionX“ nám udává hodnotu osy x objektu, který je v kolizi s postavou hlavního hráče, proměnná „playerPositionX“ nám poté udává

hodnotu osy x hlavního hráče. Pokud tedy je „wallPositionX“ menší jak „playerPositionX“, tak to znamená, že zeď je od hráčské postavy nalevo a tím pádem má postava čelit vpravo. To se provede tím, že proměnné „_direction“ nastavíme hodnotu 0.5f, tato hodnota nám udává velikost postavy na ose x a jelikož je hodnota pozitivní, tak sprite této postavy čelí vpravo. Proměnná „_direction“ se poté předá metodě „transform.localScale“, tato metoda transformuje hlavní postavu s parametry velikosti na osách x, y a z, přičemž osa x je proměnná „_direction“. To samé se provede pro případ, kdy zeď je od postavy vpravo, v tomto případě „_direction“ nabude mínusové hodnoty -0.5 a otočí se tím pádem doleva.

Výpis kódu 5: Speciální pohyb WallSlide a WallJump

```
private void OnCollisionStay2D(Collision2D collision)
{
    if (collision.gameObject.layer == LayerMask.NameToLayer(wallLayerName))
    {
        rb.velocity = new Vector2(rb.velocity.x, y: rb.velocity.y * slideMultiplier);
        _playerController._extraJumps = extraJumps;

        float wallPositionX = collision.transform.position.x;
        float playerPositionX = transform.position.x;
        _anim.SetBool(id: IsJump, value: true);

        if (wallPositionX < playerPositionX)
        {
            _direction = 0.5f;
        }
        else if (wallPositionX > playerPositionX)
        {
            _direction = -0.5f;
        }
        transform.localScale = new Vector3(x: _direction, y: (float) 0.5, z: 1);
```

Zdroj: Autor (2023)

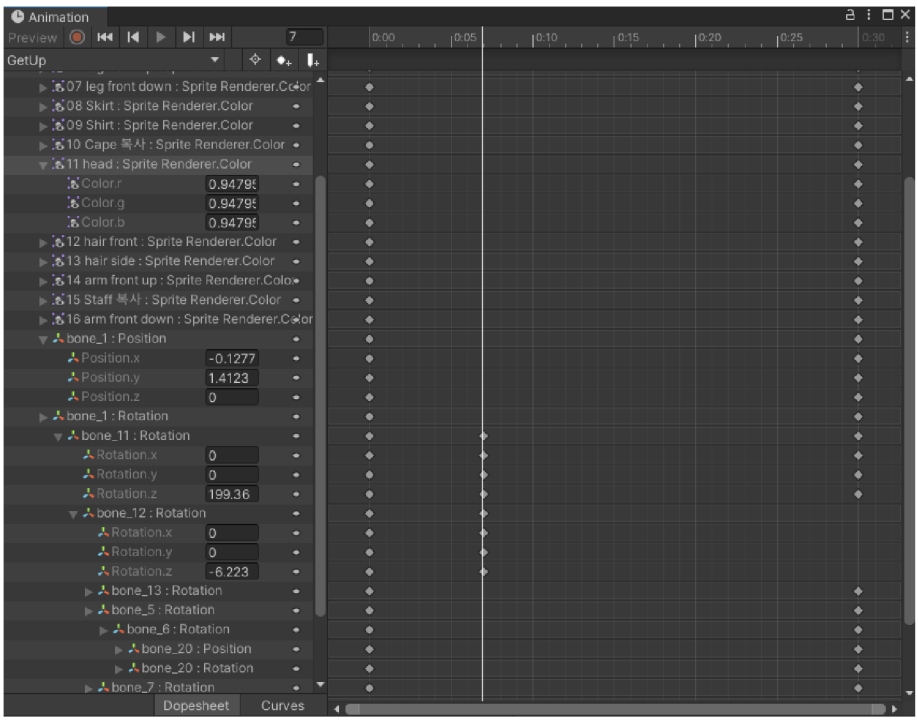
6.1.6 Animace

Pro lepší reprezentování pohybu postavy jsou v kódu volány dvě funkce, první z nich je „transform.localScale“. Pomocí této metody se objekt postavy transformuje tím, že využije proměnnou „_direction“. Pomocí „Input.GetAxisRaw“ lze zjistit na jakou stranu se postava pohybuje a tím měnit „_direction“. Na výpisu kódu 1 lze vidět jaké hodnoty se do této funkce doplňují, pomocí této funkce buďto objekt postavy čelí vlevo, či vpravo. Pro druhou a hlavní funkci pro animaci je použito „_anim.SetBool()“. Tato funkce odkazuje na komponent „Animator“ u postavy pomocí proměnné „_anim“, pro správné

přehrání animace jsou potřeba do funkce dosadit 2 parametry: název animace a true nebo false. Animator poté zavolá chtěnou animaci a postava jí provede.

Existuje několik způsobů pro vytvoření animace, pro tuto práci je vybráno použití „Animation Clip“ a v něm změna rotace spritů které tvoří postavu, nebo například změna jejich barvy pro chtěný efekt běhu, skákání, pádu, ublížení, smrti a útoku. Aby celý proces fungoval správně a animace vypadala plynule, tak je potřeba použít v animačním klipu klíčové snímky, které je potřeba manuálně nastavit a přidat jim chtěné hodnoty, jako je například barva nebo rotace, tento proces lze vidět v obrázku 2. Pomocí interpolace se pak těchto hodnot plynule dosáhne v daném časovém úseku. Pro plynulý přechod je vždy potřeba použít minimálně dvou klíčových snímků. Pro každou animaci je vytvořen vlastní animační klip, který mění rotaci spritů postavy a v některých případech jejich barvu.

Obrázek 1: Animační klip a klíčové snímky

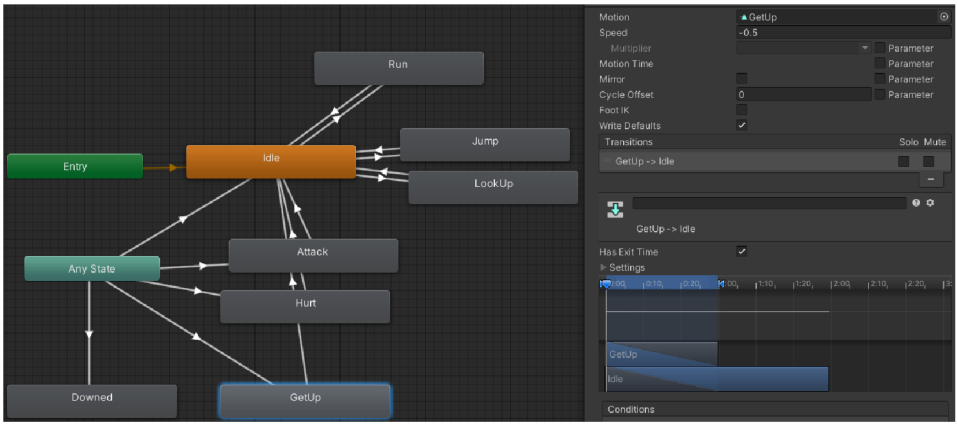


Zdroj: Autor (2023)

Jak lze na obrázku 2 vidět, tak při animaci pádu se mění jak rotace spritů, tak barva postavy, v tomto případě si postava pomalu „lehne“ a její celková barva nabere temnější odstín šedi. Na obrázku 2 lze vidět veškerá návaznost animací. V levé polovině je ukázáno, že v tomto případě se animace „GetUp“ může přehrávat z jakékoliv jiné animace a dále navazuje na animaci „Idle“, toto je provedené kvůli návaznosti animací a díky tomu vypadají animace plynuleji. V pravé polovině si lze všimnout, že animace se

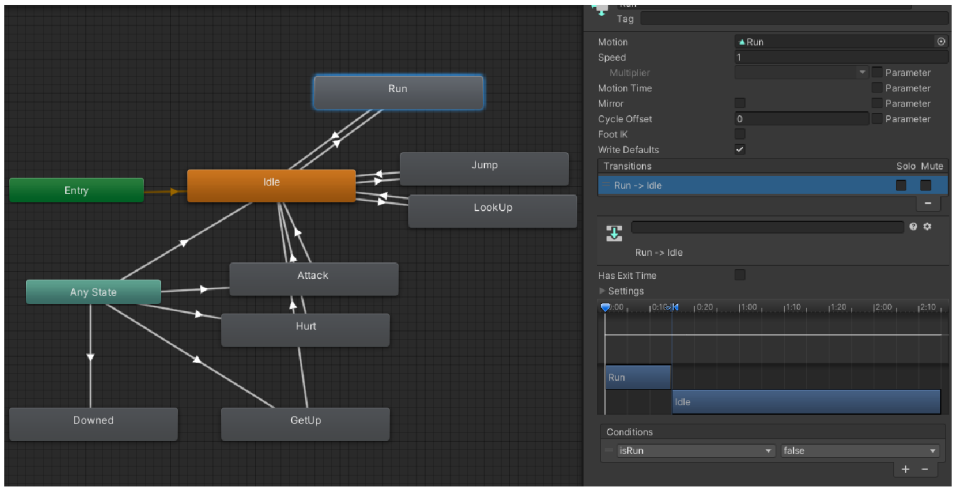
přehrává pozpátku, to lze měnit pomocí „speed“ parametru, který je nastaven na hodnotu -0.5. Pomocí znaménka mínus se animace přehrává opačně, to znamená že místo toho, aby se animace přehrála od 1. do 30. snímku, tak se přehraje od 30. do 1. snímku, hodnota 0.5 nám dále značí rychlost animace, v tomto případě animace nabere poloviční rychlost. Dále je zde možné vidět přechod animace „GetUp“ na animaci „Idle“ a rychlost prolínání mezi těmito animacemi. Jelikož tato animace je přehrávána v moment, kdy uživatel nemá kontrolu nad postavou, tak je možné použít plynulejší a pomalejší přechod mezi animacemi bez toho, aniž by to narušilo jakýkoliv pohyb hlavní postavy.

Obrázek 2: Návaznost animace „GetUp“ a „Idle“



Zdroj: Autor (2023)

Obrázek 3: Návaznost animací „Run“ a „Idle“



Zdroj: Autor (2023)

6.2 Objekty ve světě

6.2.1 Bodáky

Ve světě se nacházejí různé objekty, jako jsou bodáky, které jsou strategicky umístěné pozicích, přes které se uživatel pomocí hlavní postavy a precizních pohybů a reflexů musí dostat. Tyto bodáky by měly sloužit jako zvýšení obtížnosti pro uživatele a vytváření napětí během hry. Bodáky jsou reprezentovány pomocí různých spritů a velikostí a mohou být umístěny ve strategických pozicích, které vyžadují precizní pohyby a reflexy od uživatele. Když se uživatel pokusí projít bodákem, dojde k interakci mezi postavou a bodákem. Pro tuto interakci jsou nastavené takzvané „killboxy“. K zajištění kolize je potřeba přidat objektu „killbox“ komponent s názvem „Box Collider 2D“, v tomto komponentu je potřeba v inspektoru potvrdit „isTrigger“, to znamená že objekt funguje jako spouštěč a ne jako kolizní těleso, to znamená že pokud dojde ke kolizi mezi hráčskou postavou a touto oblastí, tak se spustí skript s názvem „TeleportToPlace“. Ve výpisu kódu 6 lze vidět, že nejdříve je potřeba definovat jaký objekt je hlavní postava, kterou uživatel ovládá. Hráčská postava se nalezne pomocí metody „GameObject.FindWithTag()“ a přidáním parametru ve formátu string s názvem štítku, který má hlavní postava přiřazená a tato postava je poté označena jako „_player“. Pokud kolize proběhne s postavou hráče, tak je tato postava teleportována na předem danou osu x a osu y na herní mapě. Tyto osy se dají definovat přímo v Unity editoru.

Výpis kódu 6: Teleportace hlavní postavy

```
    public void Start()
    {
        _player = GameObject.FindWithTag("Player");
    }

    Event function
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            _player.transform.position = (Vector3)new Vector2(XAxis,YAxis);
        }
    }
}
```

Zdroj: Autor (2023)

6.2.2 Projektily a samonaváděcí projektily

Ve hře byly nadále rozmístěny objekty, pomocí kterých svět nabývá obtížnosti. Tyto objekty jsou projektily, které jsou reprezentovány pomocí spojením několika spritů do prefabu. Tyto projektily se objevují hlavně na místech, na kterých je uživatelovi povoleno využít plně speciální pohyb „Dash“. Při správném načasování a použití „Dash“ pohybu se hráč může vyhnout projektilům a překoná náročné pasáže. Tento prvek přidává do hry dynamiku a vyžaduje od hráče rychlé reakce a precizní provedení pohybu. Pokud se uživatel dotkne projektilu, tak to jeho hlavní postavu přemístí na předem danou lokaci. Tento proces zařizuje skript s názvem „TeleportToPlace“.

Před tím, než projektil začne fungovat, je potřeba prefabu projektilu přiřadit 4 komponenty. Prvním je „Rigid Body 2D“, tento komponent může zůstat beze změny. Dalším je „Circle Collider 2D“, v tomto komponentu je potřeba změnit kolizní hranice tohoto objektu a dále je velmi důležité povrdit „isTrigger“. Poté je třeba přidat samotný skript „TeleportToPlace“, v inspektoru lze nastavit koordinace x a y, tyto koordinace nám mají určit, kam se hlavní postava přemístí při kolizi. Jako poslední je komponent „Animator“, pro lepší vizuální zážitek je pro projektil vytvořená animace z individuálních spritů, které tvoří celý prefab. Ve výpisu kódu 7 lze vidět celý proces teleportace. Při spuštění hry se jako první najde herní objekt pod se štítkem „Player“ a zastoupí ho proměnná „_player“, poté se při každé kolizi s jakýmkoliv kolizním objektem kontroluje, jestli tento objekt má štítek „Player“, pokud ano, tak se hlavní postava přemístí na pozici, která je nastavená v inspektoru pomocí osy x a y.

Výpis kódu 7: Teleportace hráče při kolizi s projektilem

```
    public void Start()
    {
        _player = GameObject.FindWithTag("Player");
    }

    Event function  N0rki
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            _player.transform.position = (Vector3)new Vector2(XAxis,YAxis);
        }
    }
}
```

Zdroj: Autor (2023)

Kromě projektilů, které cestují pouze v jednom směru, se v herním světě nachází i tzv. „samonavádějící projektily“. Tyto projektily putují za hráčem ihned poté co se vytvoří. Za hlavní postavou putují „jemně“, to znamená že nedělají žádné ostré pohyby a místo toho postupně mění svoji dráhu a tvoří plynulý pohyb. Aby hráče tento projektil nepronásledoval až do konce mapy, tak je nastaven časovač, po kterém se navádění vypne a projektil již pouze putuje po směru, který mu byl naposled dán. Výpis kódu 8 nám ukazuje, že tento skript využívá metodu „FixedUpdate()“, tato metoda se spouští ve stejné rychlosti jako fyzikální systém, tj. 50 snímků za vteřinu, neboli každých 20ms. Pokud kód projde přes podmínku, tak se jako první uloží vektor pro směr, tento směr se vypočítá pomocí odečtení pozice projektilu od pozice hlavní postavy, tato hodnota je poté normalizovaná, to znamená že se ukládá v celých jednotkách, to nám poté dává iluzi plynulého pohybu. Jako poslední je potřeba nastavit gravitační sílu na nulu, pokud by byla gravitační síla příliš silná, tak to zabraňuje plynulému pohybu.

Výpis kódu 8: Samonavádějící projektil

```
private void FixedUpdate()
{
    if (isHoming && target != null)
    {
        Vector2 direction = (Vector2)(target.position - transform.position).normalized;
        rb.velocity = direction * speed;
        rb.gravityScale = 0f;

        timer += Time.deltaTime;
        if (timer >= homingDuration)
        {
            isHoming = false;
        }
    }
}
```

Zdroj: Autor (2023)

K projektilům je také potřeba nastavit objekty, které tyto projektily budou vytvářet. Tyto objekty se nazývají "generátory projektilů". Jejich úkolem je vytvářet instance projektilů a umísťovat je do herního světa ve správných pozicích a směrech. Nejdříve se pro tuto funkci vytvoří objekt se správným spritem pro generátor projektilů, tomuto objektu jsou poté přidány 2 komponenty: „Animator“ a skript generování projektilů pod názvem „Projectile Spawner“. Do komponentu „Animator“ se přidá správná animace spritu. Komponent „Projectile Spawner“ poté umožňuje několik možností nastavení projektilu. Do „Projectile Prefab“ se přesune prefab projektilu, který se chce použít. „Spawn Interval“ udává hodnotu v sekundách a určuje, jak rychle se projektily mají generovat. „Spawn Force“ hodnota udává jakou silou se tento projektil posune pryč ihned po generaci. „Projectile Lifetime“ je velmi důležitá hodnota která udává, jak dlouho má tento projektil zůstat ve světě před tím, než se zničí. Pomocí této proměnné hra využije méně zdrojů, pokud by část kódu, která tuto hodnota používá neexistovala, tak by se hra stala více náročnou na hardware, čím déle by běžela. Ve výpisu kódu 9 lze vidět, že ihned po spuštění hry začne „StartCoroutine(SpawnProjectiles())“. Metoda „StartCoroutine()“ je velmi užitečná tím, že se na konci dá pozastavit pomocí „WaitForSeconds()“, po uběhnutí stanovené doby se funkce znovu spustí. Pomocí metody „Instantiate() se vytvoří nová instance projektilu, který je zvolen v inspektoru, druhý parametr udává pozici a třetí rotaci objektu. Komponentu „Rigid Body 2D“ projektilu se poté aplikuje síla směrem dolů s intenzitou, která je dána v inspektoru. Tato síla je aplikována jako impuls, což způsobí okamžitý nárůst rychlosti projektilu. Následně je projektil zničen po určitém čase pomocí Destroy(), „projectileLifetime“ určuje hodnotu, po jaké době je projektil zničen, tato hodnota je také dána v inspektoru.

Výpis kódu 9: Generátor projektilů

```
private void Start()
{
    StartCoroutine( routine: SpawnProjectiles());
}

Frequently called  1 usage  N0rki *
private IEnumerator SpawnProjectiles()
{
    while (true)
    {
        GameObject projectile = Instantiate(projectilePrefab, transform.position, Quaternion.identity);

        Rigidbody2D projectileRigidbody = projectile.GetComponent<Rigidbody2D>();
        projectileRigidbody.AddForce(Vector2.down * spawnForce, ForceMode2D.Impulse);

        Destroy(projectile, projectileLifetime);

        yield return new WaitForSeconds(spawnInterval);
```

Zdroj: Autor (2023)

6.2.3 Zobrazení a odstranění textu

Při hraní hry se uživatel setká s pasážemi, ve kterých je nutné použít některý ze speciálních pohybů, avšak kdyby mu tato informace nebyla nijak předána, tak neví jak pokračovat. K tomuto je vytvořen skript, pomocí kterého se text objeví při překročení předen daného bodu ve hře, tento text zůstane po určitý časový interval viditelný a poté pomalu začne mizet.

Pro zobrazení textu je potřeba jako první stáhnout nový balíček pod názvem „TextMeshPro“. Jako další je potřeba vytvořit prázdný objekt pomocí „Create Empty“ a tomuto objektu je potřeba přiřadit 3 komponenty. „Rect Transform“ který nám umožní ukotvit text do středu rodičovského kontejneru, ve kterém je tento text zabalen. Dalším komponentem je „Mesh Renderer“ pomocí kterého se přiřadí vizuální vlastnosti objektu. Poslední je třeba přiřadit „TextMeshPro – Text“, tento komponent se stará už o samotný text, v inspektoru se poté nastaví velikost textu, font, barva a další vlastnosti. Tento objekt se poté přidá pod objekt spouštěče. Objektu spouštěč je nutné přidat komponent „Box Collider 2D“, v tomto komponentu se může upravit kolizní hranice a potvrdí se možnost „isTrigger“. Jako další komponent je nutné přidat samotný skript pro zobrazení a odstranění textu, tento komponent se buďto přenese do inspektoru, nebo se v inspektoru může vyhledat. V inspektoru se tomuto komponentu přiřadí objekt s textem, který byl vytvořen. V obrázku 4 lze vidět průběh mizení textu pomocí lineární interpolace změnou alfa hodnoty barvy po předem dané časové periodě.

Obrázek 4: Plynulé mizení textu

Zdroj: Autor (2023)

Ve výpisu kódu 10 se zjišťuje, jestli objekt, který kolidoval s kolizní hrací byl skutečně objekt se štítkem „Player“, pokud tomu tak je, tak proběhne funkce na zobrazení textu. Metoda která volá metodu pro zobrazení textu je nazvaná „InvokeRepeating()“. Tato metoda zajišťuje plynulé zobrazení kvůli parametrům, které ji lze přiřadit. První parametr je název metody, kterou je potřeba volat. Druhým parametrem je pak časová hodnota, která udává za jak dlouho se tato metoda má volat, v našem případě je tento parametr nastaven na 0f, což znamená že se bude volat okamžitě. Posledním parametr je, jak často se tato metoda bude volat, tento parametr je nastaven na 0.01f, to znamená, že se bude volat 100 snímků za vteřinu neboli každých 10ms. Metoda „Invoke()“ poté zajišťuje že parametr „isFading“ se přepne na false a může tedy běžet část kódu ve výpisu kódu 11. Jako první parametr v této metodě je název metody která se má vyvolat a jako druhý parametr je časová hodnota, za jak dlouho se tato metoda má vyvolat.

Výpis kódu 10: Metoda OnTriggerEnter2D pro zobrazení textu

```
private void OnTriggerEnter2D(Collider2D col)
{
    if (hasTriggered)
    {
        return;
    }
    if (col.CompareTag("Player"))
    {
        appearTimer = 0f;
        InvokeRepeating( methodName: "AppearTextGradually",  time: 0f,  repeatRate: 0.01f);
        Invoke( methodName: "StartFading", fadeDuration);
    }
    hasTriggered = true;
```

Zdroj: Autor (2023)

Výpis kódu 11 ukazuje metodu „Update()“, tato metoda se volá každý snímek ve hře. Proměnná „fadeTimer“ a „fadeDuration“ zajišťuje postupné zmizení textu místo toho, aby text zmizel ihned. Výpočet proměnné „fadeTimer“ umožňuje sledovat uplynulý čas od poslední aktualizace metody Update(). Pomocí tohoto výpočtu lze měřit časové intervaly nezávisle na rychlosti snímání obrazovky a je možné použít proměnnou na základě uplynulého času. Hodnota alfa je vypočítaná pomocí „Mathf.Lerp(), tato funkce zajistí lineární interpolaci mezi viditelným textem a textem bez viditelnosti. Po výpočtu hodnoty alfa se textu obnoví hodnota alfa pomocí metody „SetTextAlpha()“ a díky tomu postupně zvyšuje svoji průhlednost. Po doběhnutí „fadeTimer“ se nastaví „isFading“ na false a kód tedy nemůže dále běžet.

Výpis kódu 11: Metoda Update pro odstranění textu

```
private void Update()
{
    if (!isFading)
    {
        return;
    }
    fadeTimer += Time.deltaTime;

    float alpha = Mathf.Lerp(a: 1f, b: 0f, t: fadeTimer / fadeDuration);

    SetTextAlpha(alpha);

    if (fadeTimer >= fadeDuration)
    {
        isFading = false;
        fadeTimer = 0f;
```

Zdroj: Autor (2023)

Výpis kódu 12 ukazuje, jak probíhá plynulé zobrazení textu. Tato metoda je volána každých 10ms kvůli parametrům v metodě „InvokeRepeating()“, přesněji kvůli časové hodnotě opakování 0.01f. Proměnná „appearTimer“ udává kolik času uběhlo od začátku zobrazování textu, to funguje právě kvůli tomu jak se tato metoda volá každých 10ms, jelikož této proměnné se přičte 10ms pokaždé co je volána. Metoda „Mathf.Lerp“ poté zajišťuje plynulý přechod pomocí hodnoty alfa z úplné neviditelnosti do plné viditelnosti pomocí lineární interpolace. Textu poté přiřazena tato hodnota pomocí metody „SetTextAlpha()“. Pokud se metoda „appearTimer“ vyrovná časovému parametru, který udává, jak dlouho má tato metoda běžet, tak se spustí metoda „CancelInvoke()“ a tato metoda přestane být volána.

Výpis kódu 12: Plynulé zobrazení textu

```
private void AppearTextGradually()
{
    appearTimer += 0.01f;
    float alpha = Mathf.Lerp(a: 0f, b: 1f, t: appearTimer / appearDuration);
    SetTextAlpha(alpha);

    if (appearTimer >= appearDuration)
    {
        CancelInvoke(methodName: "AppearTextGradually");
    }
```

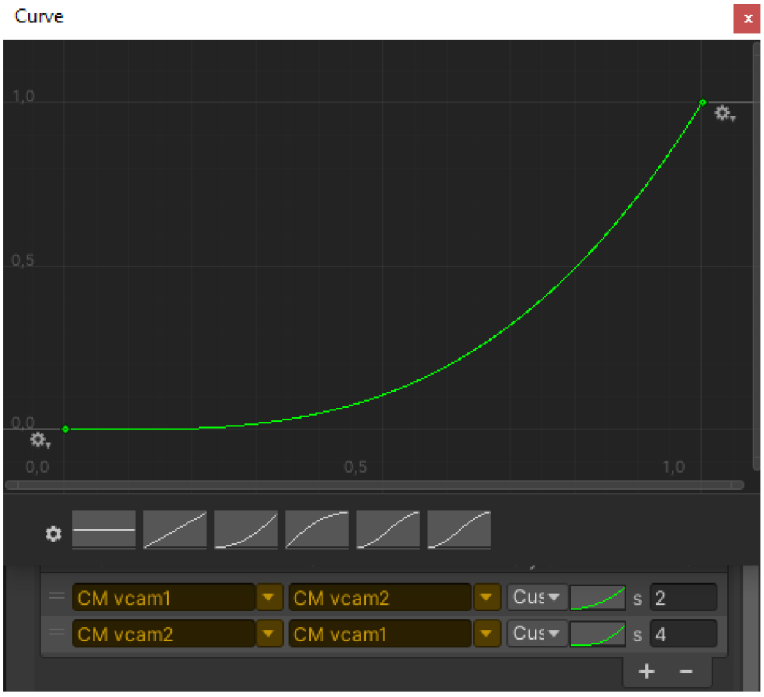
Zdroj: Autor (2023)

6.2.4 Přepínání virtuálních kamer

Ve hře se nacházejí úseky, ve kterých je výhodnější, pokud se kamera oddálí, aby uživatel mohl vidět různé nástrahy přes které se musí dostat, nebo úseky ve kterých je právě lepší, když je kamera přiblížená. Pro lepší funkčnost kamer se jako první musí nainstalovat balíček „Cinemachine“, tento balíček poskytuje pokročilé nástroje pro kontrolu kamery ve hře. „Cinemachine“ dále umožňuje vytvářet dynamické kamery, které se automaticky přizpůsobují ději a interakci hráče.

Po instalaci „Cinemachine“ se na hlavní kameru automaticky přidá komponent „CinemachineBrain“. V tomto komponentu se dají měnit různé vlastnosti jak hlavní kamery, tak možnosti prolínaní virtuálních kamer. V inspektoru po označení hlavní kamery se zvolí komponent „CinemachineBrain“ a v něm možnost „Custom Blends“, tato možnost zaručuje vytvoření plynulého prolínání kamer. Po kliknutí na vybrání prolínání kamer editor vyzve pro vytvoření nového prolínání, jelikož žádné neexistuje. Po výběru jména se toto prolínání uloží. V „Custom Blends“ je nyní vybráno toto prolínání, nyní je potřeba vybrat mezi jakými kamery toto prolínání bude fungovat, po výběru je nutné vytvořit a uzpůsobit křivku přechodu mezi těmito kamerami. Po vybraní či vytvoření této křivky je pouze potřeba přidat, jak dlouho tento časový přechod má trvat. Tento proces vytvoření křivky a výběru kamer lze vidět na obrázku 5.

Obrázek 5: Vytvoření vlastního přechodu kamer



Zdroj: Autor (2023)

Dále je potřeba vytvořit virtuální kamery, které budou ve světě používány. Po pravém klinutí myši v hierarchii na znaménko plus vyjede nabídka pro vytvoření různých objektů. Kvůli nainstalování balíčku „Cinemachine“ přibyla nová nabídka s názvem „Cinemachine“. Zde je pro výběr několik druhů kamer, mezi těmito kamerami je „Virtual Camera“, tyto kamery se vytvoří čtyři. V těchto kamerách se nastaví „Ortho Size“, tato vlastnost je zkratka pro „Orthographic size“, což určuje velikost zobrazovaného prostoru. Tato hodnota je měřena od středu kamery k hornímu či dolnímu okraji zobrazované oblasti. Pro kameru, která má být více oddálená se do vlastnosti „Ortho Size“ napíše větší hodnota než do ostatních, opak poté slouží pro kameru, která má být více přiblížená. Jako další se nastaví vlastnost priority těchto kamer, tato vlastnost určuje, která kamera má přednost při řízení zobrazení ve scéně. Hlavní kameře, která bude řídit zobrazení ihned po zapnutí hry, se přiřadí hodnota 1. Všem ostatním kamerám poté se přiřadí hodnota 0.

Dalším krokem je určení, co vlastně mají tyto kamery sledovat, to se udělá tak, že objekt hlavní postavy se přesune na vlastnost „Follow“. Všem kamerám se poté ve vlastnosti „Body“ přiřadí „Framing Transposer“. Vlastnost „Body“ se poté rozbalí a zde se změní vlastnost „Soft Zone“, tato vlastnost reguluje, jak rychle a plynule kamera reaguje na pohyb objektu. Pokud se objekt nachází uvnitř „Soft Zone“, tak je pohyb kamery jemnější a plynulejší, což umožňuje přirozené sledování objektu. Pohyb kamery může být rychlejší a reaktivnější, když se sledovaný objekt pohybuje směrem k „Soft Zone“ nebo od ní, což umožňuje kameře rychle reagovat na změny polohy objektu.

Přepínání kamer je pak pouze o změnění jejich priority. Ve skriptu se tedy nastaví, že pokud nastane kolize s objektem a tento objekt má štítek „Player“, tak se přepne mezi kamerami, které se přidají skriptu v inspektoru. Tento skript se poté přidá neviditelným objektům s vlastností „isTrigger“ a tyto objekty se rozmístí na úseky, kde se tyto kamerami mají přepínat.

Výpis kódu 13: Přepínání virtuálních kamer

```
private void OnTriggerEnter2D(Collider2D col)
{
    if (!col.CompareTag("Player")) return;
    _mainVirtualCamera.Priority = 0;
    _secondVirtualCamera.Priority = 1;
}
```

Zdroj: Autor (2023)

6.2.5 Zatřesení kamery

Pro vytvoření intenzivnější akce a přidání dynamického prvku do hry je implementován efekt třesení kamery, který se spustí, když postava spadne do určitých částí mapy. Tento prvek zintenzivňuje hráčův zážitek, a ještě více ho vtáhne do herního světa.

Před tím, než je skript schopný provádět zatřesení, je nutno přiřadit objektu kamery novou vlastnost. Ve virtuální kameře v komponentu „CinemachineVirtualCamera“ se ve spodní části vybere vlastnost „Noise“, tato vlastnost se rozklikne a vybere se „Basic Multi Channel Perlin“. Základní vícekanálový Perlinův šum označuje variantu Perlinova šumu, která místo jednoho kanálu generuje více šumových kanálů. Výpis kódu 14 ukazuje první část skriptu, který tato chování provádí. Ihned po zapnutí hry, se převezme tato vlastnost z kamery, která je přiřezaná pomocí proměnné „virtualCamera“ v inspektoru. Pokud nastane kolize s nějakým objektem, tak se vyhledá, jestli tento objekt měl štítek „Player“, pokud tomu tak je, tak se spustí metoda „StartCoroutine()“ s parametrem metody „ShakeCamera()“.

Výpis kódu 14: Skript pro zatřesení kamery

```
private void Start()
{
    virtualCameraNoise = virtualCamera.GetCinemachineComponent<CinemachineBasicMultiChannelPerlin>();
}

Event function  N0rki
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        StartCoroutine( routine: ShakeCamera());
    }
}
```

Zdroj: Autor (2023)

Druhá část tohoto skriptu se stará o samotné chvění. V této části kódu se upravuje hodnoty chvění a frekvenci šumu. Tyto hodnoty jsou přiřazeny virtuální kameře pomocí hodnot „m_AplitudeGain“ a „m_FrequencyGain“. Tyto vlastnosti jsou součástí komponenty „Basic Multi Channel Perlin“ připojené k virtuální kameře. Pomocí proměnných „shakeAmplitude“ a „shakeFrequency“, které jsou nastavené v inspektoru, se tyto proměnné změní na chtěné hodnoty. Dále se pomocí toho, že tato metoda byla vyvolána metodou „StartCoroutine()“ pozastaví provádění na počet sekund zadaných v inspektoru v proměnné „shakeDuration“. Po uplynutí zadané doby se jak amplituda, tak frekvence šumu nastaví zpět na nulu a tím se vrátí na původní hodnotu

6.2.6 Změna gravitace objektů

Při některých událostech je nutné změnit gravitační váhu. Například pro vytvoření iluze padajícího kamene stačí místo vytvoření více objektů změnit gravitaci z nuly na více, a tato změna gravitace umožní realistický pád kamene.

Výpis kódu 15 ukazuje průběh změny gravitace. Tento proces se skládá pouze ze dvou kroků. Při začátku aplikace se převezme komponent „Rigid Body 2D“ z objektu, který byl tomuto skriptu předán v inspektoru a poté se čeká na to, dokud nějaký objekt nevstoupí do hraniční kolize. Pokud se tomuto stane a tento objekt má štítek „Player“, tak se gravitace tohoto komponentu změní na proměnnou „gravityScale“, která se taktéž nastaví v inspektoru.

Výpis kódu 15: Změna gravitace objektu

```
private void Start()
{
    rb = GetComponent<Rigidbody2D>();
}

Event function  NOrki *
private void OnTriggerEnter2D(Collider2D col)
{
    if(!col.CompareTag("Player")) return;

    rb.gravityScale = gravityScale;
}
```

Zdroj: Autor (2023)

6.2.7 Květina pro větší skok

Pro dosažení platform, které se jinak nacházejí příliš vysoko pro jakýkoliv pohyb, slouží objekty, jež vypadají jako květiny a tyto objekty hlavní postavu vyhodí do vzduchu, pokud se jich dotkne. V okně projektu se nejdříve zvolí všechny sprity této květiny, poté se přesunou na hlavní scénu, kde Unity vyzve k uložení nové animace, tato animace se uloží pod unikátním jménem. Do hlavní scény se poté jakýkoliv ze spritů květiny přesune a tomuto objektu se přidá komponent „Animator“, tomuto komponentu se do vlastnosti „Controller“ přesune soubor animace. Objektu se dále přidají komponenty „Rigid Body 2D“, „Box Collider 2D“ a skript pod názvem „JumpPlant“. Pro tento objekt je potřeba, aby s ním nešlo nijak hýbat, proto se v „Rigid Body 2D“ povolí možnost „Freeze Position X“, „Freeze Position Y“ a „Freeze Rotation Z“. V komponentu „Box Collider 2D“ se přizpůsobí kolizní hranice pro sprite květiny. Ve výpisu kódu 16 lze vidět, jak probíhá skok pomocí této květiny. Pokud nastane kolize s objektem se štítkem „Player“, tak proměnná „playerRigidBody“ převezme od hlavní postavy komponent „RigidBody 2D“. Tomuto komponentu je poté přidána rychlost vektoru pomocí „new Vector2()“ přičemž osa x převezme rychlost hlavní postavy a ose y se zamění hodnota za proměnnou „jumpForce“.

Výpis kódu 16: Vyšší skok pomocí objektu květiny

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        Rigidbody2D playerRigidbody = collision.gameObject.GetComponent<Rigidbody2D>();

        playerRigidbody.velocity = new Vector2(playerRigidbody.velocity.x, y: jumpForce);
    }
}
```

Zdroj: Autor (2023)

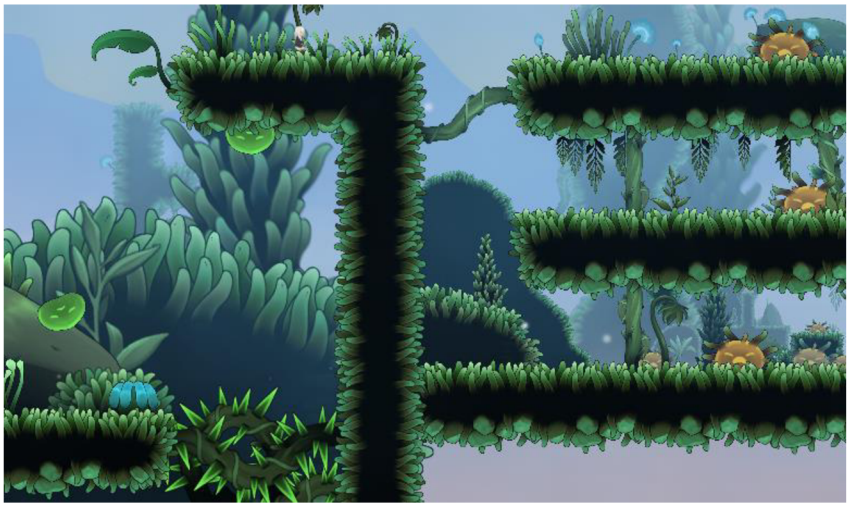
6.3 Nepřátelé

V herním světě se nadále vyskytují objekty, které se pohybují z místa na místo a chovají se jinak jak projektily, tyto objekty se dají nazvat obecně jako nepřátelé. Tito nepřátelé jsou vytvořeny jako objekty ve světě. Pro zobrazení spritu je potřeba komponent „Sprite Renderer“ s přidaným požadovaným spritem ve vlastnosti „Sprite“. Většina spritů těchto nepřátel jsou animované, a proto je také potřeba přidat komponent „Animator“. Pro vytvoření nové animace je nutné vytvořit nový soubor animace, tohoto je dosaženo zvolením všech spritů této animace v okně projektu. Všechny tyto sprity se přesunou do hlavního okna scény a zde se zobrazí nové okno pro uložení animace v hierarchii souborů, po výběru vhodného jména a lokace se tento soubor uloží. Komponentu „Animator“ se poté tento soubor animace přidá přesunutím na vlastnost „Controller“.

Pro požadovanou kolizi s hlavní postavou je potřeba těmto objektům přidat komponent „Capsule Collider 2D“ nebo „Circle Collider 2D“, tento komponent pro kolizní hranice nabývá tvaru kapsule nebo kruhu. Kvůli tvaru nepřátel ve hře je „Circle Collider 2D“ nejlepší možnost. Po zvolení tohoto komponentu v inspektoru se musí upravit tvar a velikost kolizních hranic, tohoto se dosáhne kliknutím na vlastnost „Edit Collider“, v editoru se poté dají upravit hranice pomocí myši, nebo v inspektoru pomocí vlastností „Offset“ a „Size“. Jako další je potřeba tomuto komponentu potvrdit vlastnost „isTrigger“ pro skript „TeleportToPlace“, který zajistí přemístění hráče. Tomuto komponentu se přiřadí osa x a osa y kam se při dotyku má hlavní postava přesunout.

V obrázku 6 lze vidět nepřátele „Jello“ a „Jumpy“, kteří jsou popsáni níže.

Obrázek 6: Nepřátele „Jello“ a „Jumpy“



Zdroj: Autor (2023)

6.3.1 Nepřítel „Jello“

První typ nepřítele je nazván „Jello“, pohybuje se po v dané lokaci a nijak neregistruje hráče. Tento nepřítel je považován za nejméně obtížným a rozmístěn na místech, ve kterých by uživatel měl použít speciální pohyb „Dash“. Pohyb zajišťuje skript s názvem „MoveFromSideToSide“, který lze vidět ve výpisu kódu 17.

Při zapnutí hry se do proměnné „initialPosition“ uloží hodnota „transform.Position“, to zajistí, že si skript bude pamatovat pozici z které se tento objekt začal hýbat. V metodě „Update()“ se zjišťuje, zda se objekt pohybuje doprava pomocí proměnné „movingRight“, pokud má proměnná „movingRight“ hodnotu true, tak se pozice, kam se má objekt posunout, vypočítá použitím proměnné „initialPosition“ a vynásobením proměnné „moveDistance“ s vektorem vpravo. Pokud „movingRight“ má hodnotu false, tak se provede stejný postup, ale proměnná se vynásobí s vektorem vlevo. Metoda „MoveTowards()“ zajišťuje pohyb, přičemž proměnné se vloží v tomto pořadí: nynější pozice, chtěná pozice a rychlost * čas. Pokud se objekt dostane na chtěnou pozici, tak proměnná „movingRight“ nabude hodnoty false a objekt se začne pohybovat na opačnou stranu.

Výpis kódu 17: Pohyb pro nepřítele „Jello“

```
private void Update()
{
    Vector3 targetPosition;
    if (movingRight)
    {
        targetPosition = initialPosition + Vector3.right * moveDistance;
    }
    else
    {
        targetPosition = initialPosition + Vector3.left * moveDistance;
    }

    transform.position = Vector3.MoveTowards( current: transform.position, targetPosition,  maxDistanceDelta: moveSpeed * Time.deltaTime);

    if (transform.position == targetPosition)
    {
        movingRight = !movingRight;
    }
```

Zdroj: Autor (2023)

6.3.2 Nepřítel „Slimper“

Druhý typ nepřítele lze rozpoznat pod názvem „Slimper“, tento nepřítel také neregistruje hlavní postavu, a proto je společně s „Jello“ považován za nejméně obtížné. „Slimper“ skáče do vzduchu v různých časových intervalech. Tento proces lze dosáhnout pomocí skriptu ve výpisu kódu 18, kde je vidět proces kontroly před skokem. Na začátku programu se spustí metoda „StartCoroutine()“ s parametrem metody „JumpWithDelay()“. V metodě „Update()“ se při tom, když je parametr „_isGrounded“ true, zaznamenává do proměnné „elapsedTime“ kolik času uběhlo .V metodě „JumpWithDelay()“se čeká, dokud proměnná „_isGrounded“ nabude hodnoty true. Když se tak stane tak se porovnají hodnoty „elapsedTime“ a „jumpDelay“ a pokud je uběhlý čas vyšší, tak se spustí metoda „Jump()“ a proměnná „elapsedTime“ se přepíše na hodnotu 0. Samotný skok s pomocí metody „Jump()“ je vytvořen pomocí vynásobení vektoru nahoru proměnnou síly skoku a toto se načte do rychlosti „Rigid Body 2D“.

Výpis kódu 18: Metoda skoku nepřítele „Slimper“

```
private void Update()
{
    if (_isGrounded)
    {
        elapsedTime += Time.deltaTime;
    }
}

Frequently called  1 usage  NOrki *
private IEnumerator JumpWithDelay()
{
    while (true)
    {
        yield return new WaitUntil(() => _isGrounded);

        if (elapsedTime >= jumpDelay)
        {
            Jump();

            elapsedTime = 0;
```

Zdroj: Autor (2023)

6.3.3 Nepřítel „SlimeFly“

Další nepřítel nese název „SlimeFly“, tento typ nepřítele hlídkuje mezi bodem A a bodem B. Pokud se hlavní postava nalezne v předem daném rozsahu útoku, tak „SlimeFly“ provede útok, přestane hlídkovat a pokusí se spadnout na hlavní postavu, pokud hlavní postavu netrefí, tak se pokaždé v daném časovém intervalu pokusí o nový útok, ovšem tentokrát provede pouze malé skoky. Po tom, co se hlavní postava dostane do vzdálenosti, která se dá nastavit v inspektoru, tak nepřítel „SlimeFly“ znovu začne hlídkovat mezi bodem A a bodem B.

V hierarchii se přidá nový prázdný objekt a pojmenuje se „SlimeFly“, do tohoto objektu se přidá objekt tohoto nepřítele, a vytvoří se nové sprity kruhu pomocí kliknutí na tlačítko plus, poté „2D Object“, „Sprites“ a zde se vybere „Circle“. Tyto kruhy se přesunou na místa, ve kterých se chce, aby „SlimeFly“ hlídkoval. Těmto kruhům se v komponentu „Sprite Renderer“ a parametru barvy přepíše hodnota alfy na 0, aby byly zcela neviditelné. V komponentu skriptu se poté rozbalí parametr „Patrol Points“ a zde se klikne na tlačítko plus, tolikrát, kolik se pro tento objekt nachází hlídkovacích bodů. Tyto kruhy se poté jeden po jednom z hierarchie přesunou do parametru „Patrol Points“. Ve výpisu kódu 19 lze vidět, jak tento nepřítel provádí hlídkování. Proměnná „patrolPoints“ obsahuje body hlídkování a proměnná „currentPatrolIndex“ pořadí tohoto bodu. Odečtením pozice nepřítele „SlimeFly“ od chtěného bodu hlídkování se získá směr k tomuto bodu. Rychlost tohoto nepřítele se nastaví vynásobením směru a rychlosti, kterou lze nastavit. Nakonec kód porovnáním vzdálenosti mezi aktuální a cílovou pozicí zkontroluje, zda „SlimeFly“ dosáhl cílové pozice. Pokud je vzdálenost menší než hodnota 0,1 jednotky, znamená to, že nepřítel dosáhl cílového bodu. V takovém případě se zvýší index „currentPatrolIndex“ a získá se se hlídkový bod z „patrolPoints“.

Výpis kódu 19: Hlídkování nepřítele „SlimeFly“

```
private void Patrol()
{
    Vector2 targetPosition = (Vector2)patrolPoints[currentPatrolIndex].position;
    Vector2 direction = targetPosition - (Vector2)transform.position;
    direction.Normalize();

    rb.velocity = direction * moveSpeed;

    if (Vector2.Distance(a: (Vector2)transform.position, b: targetPosition) < 0.1f)
    {
        currentPatrolIndex = (currentPatrolIndex + 1) % patrolPoints.Length;
```

Zdroj: Autor (2023)

Výpis kódu 20 ukazuje průběh útoku nepřítele „SlimeFly“. Metoda „Attack“ je zodpovědná za zahájení útoku. Po zahájení metody útoku se proměnná „isAttacking“ nastaví na true, toto je důležité kvůli podmínkám pro metodu hlídkování. Dále tato metoda určuje směrový vektor odečtením pozice postavy hráče od pozice „SlimeFly“. Tento vektor představuje orientaci na hráče a směřuje trajektorii útoku nepřítele tímto směrem. Jako další se používá funkce „Mathf.Atan2“ k určení úhlu mezi směrovým vektorem a osou x, aby byla zajištěna správná orientace vůči hráči. Konstanta „Mathf.Rad2Deg“ se pak použije k převodu získaného úhlu z radiánů na stupně. Rotace se provádí pomocí „Quaternion.AngleAxis“ a to tak, že se úhel změní o konstantní hodnotu -90 stupňů kolem vektoru „Vector3.forward“. Při této úpravě jsou zohledněny případné rozdíly v orientaci spritu nebo modelu. Poslední krok zajišťuje spuštění metody „Invoke()“ která po časovém intervalu v sekundách z proměnné „attackCooldown“ spustí metodu „ResetCooldown()“, kde se proměnná „isAttacking“ přepne na false a útok tedy může znovu nastat.

Výpis kódu 20: Útok nepřítele „SlimeFly“

```
private void Attack()
{
    isAttacking = true;

    Vector2 direction = (Vector2)(playerTransform.position - transform.position);
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    transform.rotation = Quaternion.AngleAxis(angle: angle - 90f, Vector3.forward);

    isCooldown = true;
    Invoke(nameof(ResetCooldown), attackCooldown);
```

Zdroj: Autor (2023)

Výpis kódu 21 pak ukazuje metodu „Update“ která kontroluje jaký pohyb se má provést pomocí proměnných a pomocí metody „Distance(), která vypočítá vzdálenost mezi uživatelem a mezi proměnnou „attackRange“, která se pro každou instanci nepřítele nastaví.

Výpis kódu 21: Kontrola pro pohyb nepřítele „SlimeFly“

```
private void Update()
{
    if (!isAttacking)
    {
        Patrol();
    }

    float distanceToPlayer = Vector2.Distance(a: (Vector2)transform.position, b: (Vector2)playerTransform.position);
    if (distanceToPlayer < attackRange && !isAttacking)
    {
        Attack();
```

Zdroj: Autor (2023)

6.3.4 Nepřítel „Squint“

Nepřítel „Squint“ čeká ve vzduchu, dokud do rozsahu útoku nevstoupí hráč, pokud se tak stane, tak „Squint“ provede rychlý útok, v podobě rychlého přesunu na pozici hráče, toho testuje uživatelovi reflexy a je to velký skok v obtížnosti od předešlých nepřátel. Pokud se tento nepřítel netrefí do hlavní postavy, tak se zastaví a začne se pomalu pohybovat na pozici hráče, přičemž po určitém časovém úseku provede útok znova. Pokud se uživatel dostane mimo rozsahu útoku, tak „Squint“ přestane hráče pronásledovat.

Ve výpisu kódu 22 lze vidět kontrolu pro pohyb nepřítele „Squint“. Metoda „Update“ provádí měření vzdálenosti mezi polohou hlavní postavy a aktuální polohou „Squint“ která je označena atributy „playerTransform.position“ a „transform.position“. K výpočtu vzdálenosti slouží metoda „Vector2.Distance()“, která vypočítá vzdálenost mezi oběma body. Pokud „Squint“ může provést pohyb „Dash“ a zároveň je hlavní postava v rozsahu útoku, tak tento pohyb provede.

Výpis kódu 22: Kontrola pro pohyb nepřítele „Squint“

```
private void Update()
{
    float distanceToPlayer = Vector2.Distance( a: (Vector2) transform.position,  b: (Vector2) playerTransform.position);

    if (canDash && distanceToPlayer < attackRange)
    {
        DashMove();
```

Zdroj: Autor (2023)

První část skriptu pro pohyb „Dash“ tohoto nepřítele je provázen pomocí kódu ve výpisu kódu 23. Nejdříve se zjišťuje, zdali „Squint“ neprovádí tento pohyb, k tomuto slouží proměnná „isDashing“. Pokud je proměnná false, tak metoda pokračuje výpočtem pohybu. Pro určení směru pohybu „Dash“ se vypočítá vektor pohybu odečtením pozice hlavní postavy, ke kterému se přistupuje pomocí proměnné „target.position“, od aktuální pozice nepřátelské postavy, kterou reprezentuje proměnná „transform.position“. Pro určení vzdálenosti mezi hlavní postavou a nepřítelem, je pak použita metoda „Vector2.Distance()“ s proměnnými polohy nepřítele a polohy hráče. Pokud je vzdálenost k hlavní postavě menší nebo se rovná hodnotě, která je nastavená v inspektoru, tak se provede další část kódu. Proměnná „isDashing“ se nastaví na true a proměnná „canDash“ se nastaví na false. Tyto hodnoty nám zajišťují správné chování podmínek. Metoda Invoke() se používá k naplánování vyvolání metody „ResetDash()“ po uplynutí zadané doby trvání v proměnné „dashCooldown“. Jako další se aktualizuje proměnná „originalPosition“ na aktuální pozici nepřítele „Squint“, čímž zachová výchozí pozice pro pohyb

„Dash“. Dále se vypočítá vektor „targetPosition“ přičtením součinu vektoru směru a proměnné „dashDistance“ k proměnné „originalPosition“. Výsledkem je vypočtená pozice, ke které se nepřátelská postava vrhne. Pro zahájení samotného pohybu se pak využije metoda „StartCoroutine()“ s metodou „DashCoroutine(targetPosition)“ jako parametr. Tato metoda slouží pro plynulý přechod z původní pozice na pozici hráče. Pokud se hlavní postava nenachází dostatečně blízko k začátku pohybu „Dash“, tak se nepřítel začne pohybovat k uživatelovi s regulární rychlostí.

Výpis kódu 23: První část pohybu „Dash“ nepřítele „Squint“

```
private void DashMove()
{
    if (!isDashing)
    {
        Vector2 direction = (Vector2)(target.position - transform.position);

        direction.Normalize();

        float distanceToTarget = Vector2.Distance(a: (Vector2)transform.position, b: (Vector2)target.position);
        if (distanceToTarget <= dashDistance)
        {
            isDashing = true;
            canDash = false;
            Invoke(nameof(ResetDash), dashCooldown);

            originalPosition = (Vector2)transform.position;

            Vector2 targetPosition = originalPosition + (direction * dashDistance);

            StartCoroutine(routine: DashCoroutine(targetPosition));
        }
        else
        {
            rb.velocity = direction * speed;
```

Zdroj: Autor (2023)

Část skriptu ve výpisu kódu 24 slouží jako druhá část pohybu „Dash“. Při vyvolání se inicializuje proměnná „elapsedDuration“ na hodnotu 0, tato hodnota představuje dobu, která uplynula během pohybu „Dash“. Dokud je proměnná „elapsedDuration“ menší jak proměnná „dashDuration“, kterou lze nastavit v inspektoru, tak proběhne další část kódu. V každé iteraci cyklu while se vypočítá hodnota interpolace „t“ vydělením uběhlého času a proměnné „dashDuration“. Tato hodnota se dále použije při výpočtu vektoru nynější pozice „currentPosition“ pomocí metody „Vector2.Lerp“. Pro aktualizování pozice nepřítele se použije metoda „MovePosition()“ s parametrem vektoru „currentPosition“. Na konci smyčky se vypočítá uběhlý čas přičtením času „Time.deltaTime“ do proměnné „elapsedTime“, tato metoda se znovu použije při zahájení nové iterace smyčky. Po dokončení pohybu „Dash“ se proměnná „isDashing“ nastaví na false a pro všechnu

zbývající rychlost z pohybu „Dash“ se nastaví rychlost na nulu. Jako další se vypočte vzdálenost mezi hráčskou postavou a nepřítelem, pokud se bude uživatel stále nacházet v hraniční vzdálenosti pro útok, tak se „Squint“ začne pomalu pohybovat směrem k hlavní postavě pomocí metody „Move()“.

Výpis kódu 24: Druhá část pohybu „Dash“ nepřítele „Squint“

```
private IEnumerator DashCoroutine(Vector2 targetPosition)
{
    float elapsedDuration = 0f;

    while (elapsedDuration < dashDuration)
    {
        float t = elapsedDuration / dashDuration;
        Vector2 currentPosition = Vector2.Lerp( a: originalPosition,  b: targetPosition, t);

        rb.MovePosition(currentPosition);

        elapsedDuration += Time.deltaTime;
        yield return null;
    }

    isDashing = false;
    rb.velocity = Vector2.zero;

    float distanceToPlayer = Vector2.Distance( a: (Vector2)transform.position,  b: (Vector2)playerTransform.position);

    if (distanceToPlayer < attackRange)
    {
        Move();
```

Zdroj: Autor (2023)

Výpis kódu 25 pak ukazuje tento pomalý pohyb. Směr se vypočte odečtením pozice nepřítele od pozice hlavní postavy. Tento směr je poté normalizován pro zajištění konzistentní rychlosti pohybu pomocí metody „Normalize(). Samotný pohyb je pak vypočítán pomocí vynásobením směru s proměnnou rychlosti „speed“.

Výpis kódu 25: Metoda pomalého pohybu nepřítele „Squint“

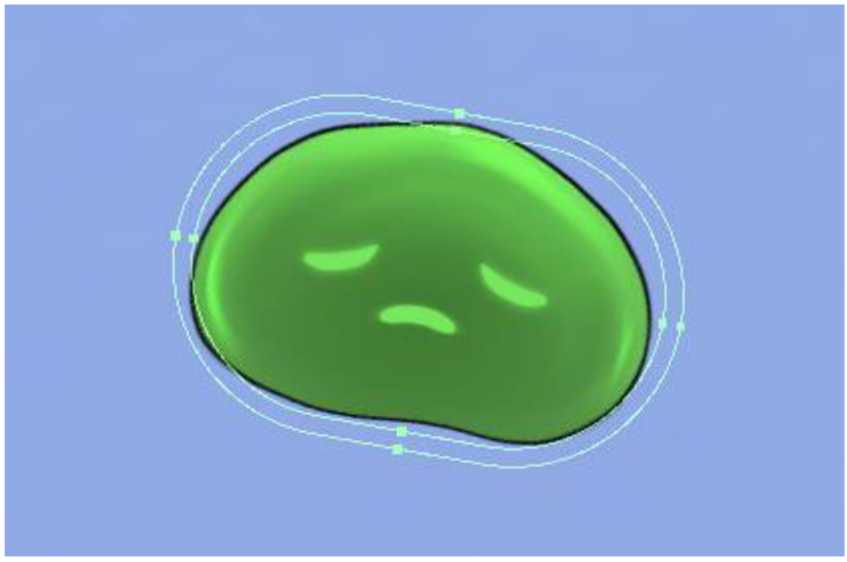
```
private void Move()
{
    Vector2 direction = (Vector2)(target.position - transform.position);

    direction.Normalize();

    rb.velocity = direction * speed;
```

Zdroj: Autor (2023)

6.3.5 Nepřítel „Hoppter“

Posledním nepřítelem je „Hoppter“. Tento nepřítel nijak neregistruje hráče, ale i tak se řadí mezi více obtížné nepřátele. Pro správné chování je potřeba v hierarchii objektů vytvořit prázdný objekt, který bude sloužit jako „rodič“ pro objekty tohoto nepřítele. Tomuto nepřítelovi je potřeba přiřadit druhý detektor kolize, který je menší jak detektor kolize s hráčem. Toho se dosáhne přiřazením nového komponentu detektoru kolize „Capsule Collider 2D“. Tomuto komponentu se poté upraví hranice kolize tak, aby při dotyku hráče stále fungoval skript na přemístění, to znamená že hranice musí být při editaci viditelně rozdílné. V obrázku 7 lze vidět rozdíl těchto kolizních hranic, přičemž menší hranice slouží ke kolizi s vrstvou „Ground“ a větší hranice s hlavní postavou.

Obrázek 7: Kolizní hranice nepřítele „Hoppter“

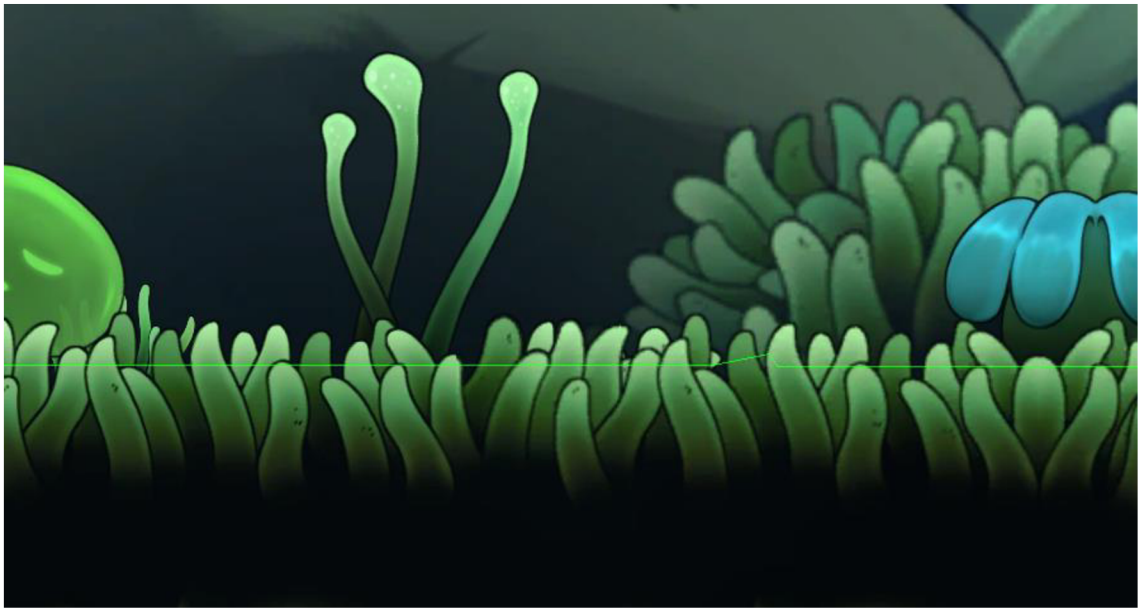


Zdroj: Autor (2023)

Do prázdného rodičovského objektu se poté vloží dva objekty spritu kruhu pomocí pravého kliknutí na prázdný objekt, přejetí na „2D Objects“, „Sprites“ a zde se zvolí „Circle“. Tyto kruhy se rozmístí na místa, ve kterých má tento nepřítel hlídkovat, jelikož je „Hoppter“ závislý na vrstvě „Ground“ a gravitace na něj normálně funguje, tak záleží pouze na ose x těchto kruhů, po rozmístění se pojmenují jako „Point A“ a „Point B“ pro lepší přehled v hierarchii. Těmto kruhům se nakonec nastaví hodnota alfa na nulu a tím se zneviditelní. Jako další je potřeba vytvořit objekt pro kontrolu vrstvy „Ground“. Na tvaru nezáleží, ale pro tento případ je to tvar kapsule, vytvoří se stejným postupem jako kruhy výše, ale místo „Circle“ se zvolí „Capsule“. Tato kapsule se vloží na objekt nepřítele tak, aby kousek přesahoval pod sprite, tímto se bude testovat vrstva na které nepřítel

bude. Nakonec se tomuto objektu kapsule také nastaví hodnota alfa na nulu. Pro lepší chování nepřítele ve hře, se při editaci kolizních hranic udělá malý výsek nahoru ve tvaru trojúhelníku. Při případném skoku, kdy nepřítel dopadne „vzhůru nohama“, tak tento výsek napraví rotaci nepřítele a tím zajistí neustálé zamyšlené chování. Na obrázku 8 lze vidět tento výsek v neonově zelené barvě.

Obrázek 8: Malý výsek v kolizní hranici „Ground“ pro nepřítele „Hoppter“



Zdroj: Autor (2023)

Výpis kódu 26 ukazuje, jak se nepřítel chová při spuštění aplikace. Ihned po spuštění se získá komponent „Rigid Body 2D“, který se bude používat pro změny směru a rychlosti. Poté se spustí funkce „InvokeRepeating()“, k opakování metody „Jump“ v určitých časových intervalech. Do funkce „InvokeRepeating()“ je potřeba vložit tři argumenty: metodu „Jump“, prodlevu mezi prvním vyvoláním, která je reprezentována proměnnou „jumpInterval“, a interval mezi následujícími vyvoláními, což je opět reprezentováno proměnnou „jumpInterval“.

Výpis kódu 26: Metoda Start() pro nepřítele „Hoppter“

```
private void Start()
{
    rb = GetComponent<Rigidbody2D>();

    InvokeRepeating( methodName: "Jump",  time: jumpInterval,  repeatRate: jumpInterval);
}
```

Zdroj: Autor (2023)

Pomocí kódu ve výpisu kódu 27 se provádí skok a kontrola uzemnění nepřítele. Metoda „FixedUpdate()“ se provádí v pevném časovém intervalu, který nezávisí na snímkové frekvenci, což zajišťuje konzistentní a přesné aktualizace. V této metodě se zjišťuje, zdali objekt kapsule je v kolizi s vrstvou „Ground“, k tomuto slouží tři proměnné, do kterých se hodnoty nastaví v inspektoru. Těmito hodnotami jsou: pozice kapsule, poloměr kontroly a proměnná „whatIsGround“, kde se vybere vrstva „Ground“. Pokud se nepřítel dotýká vrstvy „Ground“, tak provede skok, který je vypočítán pomocí vektoru nahoru a síly skoku, kterou lze přiřadit v inspektoru.

Výpis kódu 27: Metoda skoku a kontrola uzemnění nepřítele „Hoppter“

```
private void Jump()
{
    if (_isGrounded)
    {
        rb.velocity = Vector2.up * jumpForce;
    }
}

Event function  NOrki
private void FixedUpdate()
{
    _isGrounded = (bool)Physics2D.OverlapCircle((Vector2)groundCheck.position, checkRadius, (int)whatIsGround);
}
```

Zdroj: Autor (2023)

V metodě „Update()“, která je závislá na snímkové frekvenci, se volá metoda „Move()“, tato metoda je vyobrazená ve výpisu kódu 28. Proměnná „moveDirection“ je určena na základě hodnoty proměnné „isMovingRight“. Pokud je „isMovingRight“ false, tak je hodnota „moveDirection“ nastavena na 1, v opačném případě je hodnota „moveDirection“ nastavena na -1. Tato hodnota se dále používá při výpočtu rychlosti nepřítele „Hoppter“ a určuje směr pohybu. Atribut rychlosti je aktualizován na nový vektor pomocí proměnných rychlosti pohybu „moveSpeed“, proměnné „moveDirection“ a aktuální osy y tohoto nepřítele. Následně je použita podmínka pro kontrolu pohybu směru pohybu a aktuální lokace nepřítele vzhledem ke chtěnému hlídkovacímu bodu. Pokud se nepřítel pohybuje doprava a osa x je větší, nebo rovná se ose x bodu B, tak je proměnná „isMovingRight“ nastavena na false a zavolá se metoda „Flip()“. V opačném případě když se nepřítel pohybuje doleva a pozice přesáhne nebo rovná se bodu A, tak je proměnná „isMovingRight“ nastavena na true a metoda „Flip()“ se zavolá taktéž. Tato Metoda převezme hodnotu velikosti spritu do nové proměnné vektoru osy x „scale.x“ a nastaví mu hodnotu -1, poté tuto hodnotu použije na přepsání hodnoty velikosti spritu čímž jej přetočí na druhou stranu.

Výpis kódu 28: Pohyb nepřítele „Hoppter“

```
Frequently called  1 usage  N0rki
private void Move()
{
    float moveDirection = isMovingRight ? 1f : -1f;
    rb.velocity = new Vector2(x: moveSpeed * moveDirection, rb.velocity.y);

    if (isMovingRight && transform.position.x >= pointB.position.x)
    {
        isMovingRight = false;
        Flip();
    }
    else if (!isMovingRight && transform.position.x <= pointA.position.x)
    {
        isMovingRight = true;
        Flip();
    }
}

Frequently called  2 usages  N0rki
private void Flip()
{
    Vector3 scale = transform.localScale;
    scale.x *= -1;
    transform.localScale = scale;
}
}
```

Zdroj: Autor (2023)

6.4 Tvorba spritů

Většina spritů v této hře je použita z již vytvořených balíčků z itch.io pod licencí pro neomezené personální a komerční užití nebo z Unity Store pod „Standard Unity Asset Store EULA“. Použití pouze těchto balíčků však přináší omezení v podobě opakovatelnosti spritů a malé originality. Pro tento problém je použita generativní umělá inteligence pro generování obrázků pomocí „prompt“, což je textový popis obrázku, který se má generovat. Ke generování těchto obrázků je dosáhnuto pomocí webové aplikace Leonardo AI. Pro obrázky z Leonardo AI v bezplatné úrovni „Free-tier“, ve které jsou tyto sprity vytvořeny, je licence celosvětová, nevýhradní a bezplatná pro jak personální, tak komerční použití.

6.4.1 Leonardo AI

Při generování spritů je potřeba hledět na několik okolností, těmi jsou počet tokenů na účtu, rozlišení, jaký model pro generaci je vybrán a další funkce, pomocí kterých by vygenerované obrázky stáli více tokenů, jelikož každý účet má pouze omezený počet tokenů, který se po půlnoci vyresetuje. Pro generování spritů ve hře jsou použity modely „Leonardo Diffusion“ a „Leonardo Creative“ ve velikosti 768x768. Při generaci obrázku se musí přesně definovat, jak má výsledný obrázek vypadat v kolonce „Prompt“, zde se můžou vložit jak pouze jednotlivá slova, tak věty. Při psaní do této kolonky obecně platí, že čím více, tím lepe. Pokud se do tohoto spritu nemá něco generovat, tak se povolí možnost „Add Negative Prompt“ a zde se vypíšou všechny vlastnosti, které jsou ve výsledném obrázku nechtěné. Jako poslední se vybere, kolik těchto obrázků se při jedné generaci má vygenerovat. Po těchto krocích se vygeneruje specifikovaný počet obrázků, ale při tomto typu generování často dochází k nedorozumění mezi uživatelem a umělou inteligencí, a proto je potřeba generovat desítky takovýchto obrázků.

Pro lepší přesnost generování obrázků je použita funkce generování dle předlohy. Tato funkce funguje na bázi nahrání již existujícího obrázku jako vstupu, místo použití pouze kolonky „Prompt“. Po nahrání obrázku je potřeba nastavit vlastnost „Init Strength“. Tato vlastnost označuje vzor šumu, který se vytváří na začátku postupu. Tato vlastnost nabývá hodnot od 0 do 1, přičemž čím menší hodnota, tím bude výstup méně podobný originálu. Při generování těchto obrázků je použit „Init Strength“ na hodnotu 0.7 a více. Do kolonky „prompt“ je potřeba napsat „sprite“ a další vlastnosti, které jsou na obrázku chtěné. Slovo „sprite“ je použito z důvodu pozadí, z testování generace s tímto slovem je

vyvozeno, že toto slovo generuje obrázky s nejméně objekty v pozadí. Jakmile se vygeneruje obrázek, který splňuje všechny podmínky pro použití ve hře, tak je potřeba tomuto obrázku vytvořit kopii ve vyšší kvalitě. Leonardo AI tuto funkci podporuje a to tím, že se pod obrázkem klikne na možnost „Creative Upscale“. Po několika minutách je tento obrázek vygenerován ve velmi vysoké kvalitě s minimálními změnami v detailech, v některých případech se však tyto detaily ztratí, a proto je použita možnost „Upscale Image Alternate“, tato možnost trvá déle, ale zanechává více detailů z původního obrázku. V obrázku 9 lze vidět několik těchto vygenerovaných obrázků.

Obrázek 9: Vygenerované obrázky pomocí generativní AI platformy Leonardo AI

Zdroj: Autor (2023)

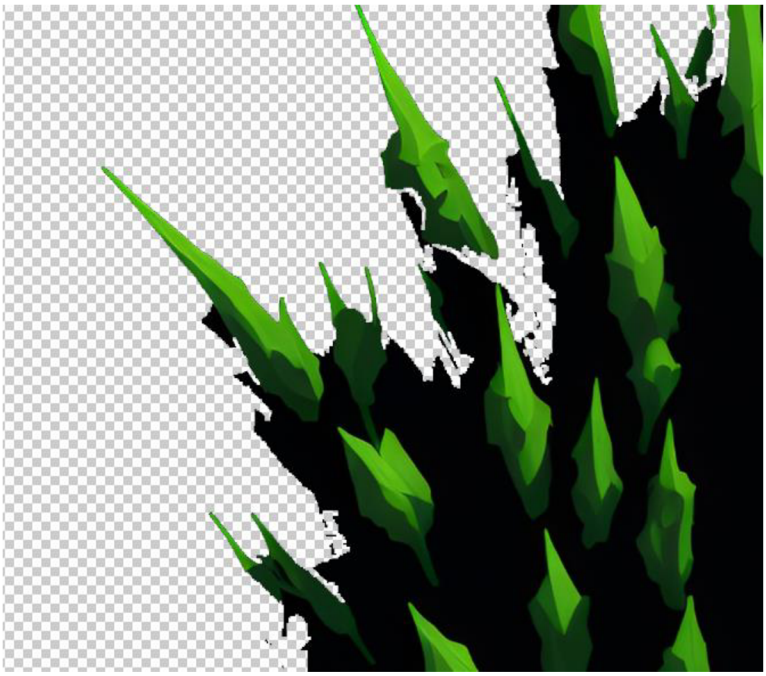
6.4.2 Photoshop

Po vytvoření obrázků je stále potřeba tyto obrázky upravit. Pokud byl obrázek vytvořen tak, aby seděl do hry, tak je mu pouze potřeba odstranit pozadí, ne všechny obrázky jsou ale takto vytvořeny, a tak je potřeba je doladit. U obrázků, ve kterých je velký rozdíl mezi barvou pozadí a barvami hlavního objektu častokrát stačí vybrat v možnosti nástrojů nástroj „kouzelná hůlka“ a kliknout na barvu pozadí. Jako další je potřeba kliknout v horní nabídce na možnost „Výběr“ a poté na „Inverze“, nebo pomocí kláves zmáčknout „Shift+Ctrl+I“. Pokud se stane, že se vyberou nějaké části hlavního objektu, tak lze vybrat nástroj „rychlý výběr“, při držení klávesy levý alt se poté pomocí myši vyberou části objektu, které se mají zanechat, častokrát je potřeba upravit velikost štětce a to tím, že se v horní nabídce klikne na ikonku kruhu a zde se vybere menší velikost. Po těchto krocích se pravým klikem myši vybere možnost „Vrstva kopírováním“. Pro export tohoto obrázku se v horní nabídce možností vybere „Soubor“ a zde „Export“ a „Exportovat jako…“ nebo

se sem lze dostat pomocí kombinace kláves „Alt+Shift+Ctrl+W“. Zde se provedou poslední úpravy jako změna plátna, jelikož častokrát je plátno příliš velké pro hlavní objekt, který byl vybrán, a tak je ve hře s ním těžší pracovat. Jako poslední se tento obrázek vyexportuje pomocí tlačítka „Export“ do námi zvolené složky, v tomto případě přímo do složky assetů hry. Tento soubor Photoshopu se poté do unikátní složky s těmito soubory, pro případné další úpravy. Toto se provede pomocí možnosti „Uložit jako“ nebo kombinace kláves „Shift+Ctrl+S“, zde se vybere složka pro uložení a klikne se „Uložit“.

U obrázků, ve kterých je menší rozdíl mezi barvou pozadí a barvou hlavního objektu je postup před export trochu jiný. Výběr tohoto objektu se provází pomocí nástrojů „Laso“, „Magnetické laso“ a pomocí nástrojů „Pero“ a „Cesta od ruky“. Při použití těchto nástrojů je potřeba dbát na detail, a proto se obrázek přiblíží pomocí lupy, při této práci se velikost přiblížení pohybovala v rozmezí od 300% výše. Pomocí myši se pak důkladně vyberou všechny části objektu, který by automatický nástroj jinak smazal. Při tomto stylu není zcela možné vše vybrat jedním tahem, proto si lze dát pauzu a pokračovat ve výběru pomocí stisknutí klávesy levého shiftu a vybrat další část objektu pomocí myši. Pokud se z omylu vybere část, která neměla, tak klávesou levého altu a myši lze tuto část odstranit. Po zvolení celého objektu je postup vytvoření vrstvy, exportu a uložení stejný jako v prvním postupu. V obrázku 10 lze vidět ztrátu detailů při použití nástrojů pro automatický výběr.

Obrázek 10: Ztracení detailů při automatickém výběru



Zdroj: Autor (2023)

6.5 Design herní mapy

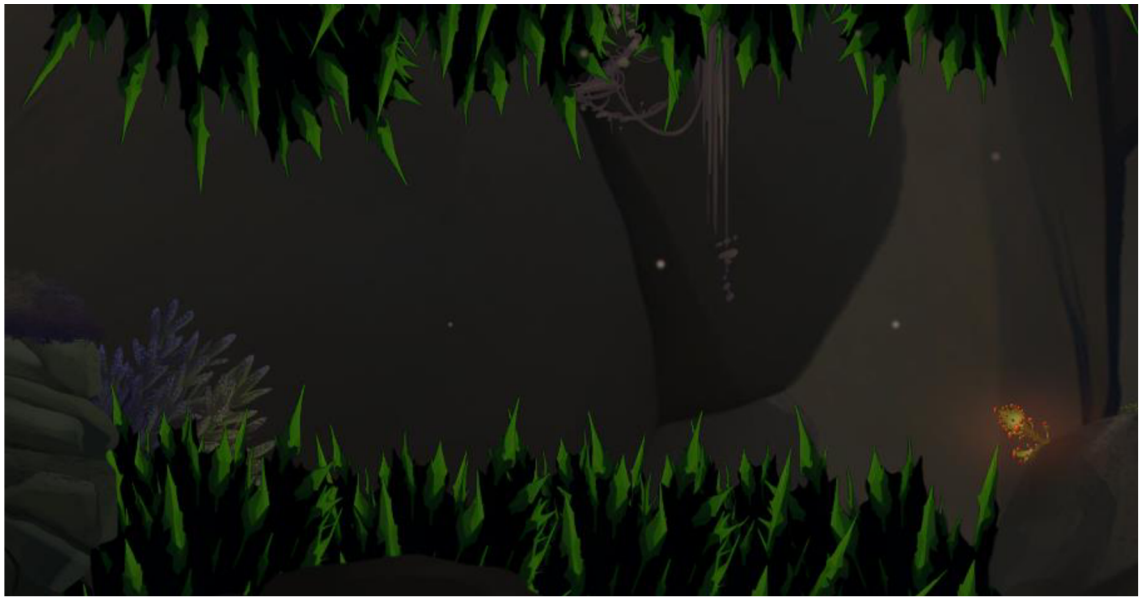
Hlavní mapa je vytvořená na základě toho, že se předpokládá, že se uživatel dokáže naučit ovládat postavu lépe v průběhu této hry. To znamená, že mapa je rozdělená do několika úseků na základě obtížnosti. Ze začátku záleží pouze na uživatelovo načasování a přesnosti jeho skoků, proto jsou první úseky navržené tak, aby byly relativně snadné a uživatel mohl získat cítění základního ovládání postavy. Jakmile uživatel dokončí tyto začátečnické úseky a zdokonalí své skokové dovednosti, tak se postupně dostává do obtížnějších částí mapy. Tento postupný nárůst obtížnosti umožňuje hráčům rozvíjení svých dovedností. V některých úsecích hry je možné, že si uživatel nebude jistý kudy pokračovat, proto jsou ve hře implementovány pomocné navigační prvky ve stylu rostlin, které produkují světlo. Tyto rostliny lze vidět na obrázku 11.

Obrázek 11: Navádění v mapě pomocí rostlin vydávajících světlo

Zdroj: Autor (2023)

Na herní mapě jsou poté jsou úseky, ve kterých musí uživatel použít speciální schopnosti a načasovat své skoky tak, aby přes tyto překážky prošel. Tyto úseky se skládají jak z úseků, které testují uživatelovi reflexy, tak úseků, ve kterých jde pouze o načasování. Tyto úseky jsou naplánovány tak, aby se k nim uživatel dostal až poté, co získá základní dovednosti, co se týče ovládání hlavní postavy. Jeden z takovýchto úseků lze vidět na obrázku 12.

Obrázek 12: Úsek mapy, pro který je nutné použít speciální pohyb „Dash“



Zdroj: Autor (2023)

V posledním typu úseku se nachází nepřátelé, kteří se uživatelovi snaží zabránit v postupu hry. Tyto úseky jsou navržené tak, aby poskytovaly uživatelovi více místa a šancí na vyhnutí se nepřátelům. Zároveň jsou navrženy podle toho, jaký typ nepřítele se v tomto úseku nachází. Pro létající nepřátele jsou vytvořeny větší místnosti, ve kterých má uživatel dostatečně místa na uhýbaní, přičemž pozemní nepřátelé jsou více v úzkých prostorech. Celkově jsou všechny tyto typy úseků postupem ve hře spojeny dohromady a nabízí tak dynamický nástup obtížnosti. Úsek s nepřáteli lze vidět na obrázku 13.

Obrázek 13: Úsek s pozemními nepřáteli

Zdroj: Autor (2023)

6.5.1 Speciální efekty

Pro lepší vtažení uživatele do herního světa, jsou ve hře využity speciální efekty v podobě kapání vody, světelných paprsků a poletujících částic. Tyto efekty jsou implementovány pomocí komponentu s názvem „Particle System“. Tento komponent se přiřadí do prázdného objektu, který se poté přesune na místo, kde se tyto částice mají vyskytovat. Jako první je potřeba v komponentu otevřít soubor vlastností „Renderer“, zde se klikne na vlastnost „Material“ a vybere se sprite, který tyto částice má tvořit. V případě této hry jsou poletující částice pod názvem „Dust1“, světelné paprsky pod názvem „Shaft“ a padající kapky vody pod názvem „Drop“.

Pro kapky vody je v základním souboru vlastností nastavena vlastnost „Gravity Modifier“ na 1, pomocí této vlastnosti budou kapky vody padat směrem dolů. Poté jsou zde nastaveny vlastnosti velikosti, času trvání, počínající a končící barvy, velikosti a maximálního počtu kapek. Tyto vlastnosti jsou nastaveny tak, aby vytvářeli dojem normálně vypadající kapky, přičemž maximální počet kapek je nastaven na jednu. Vlastnost „Rate over Time“ udává kolik kapek se bude vytvářet za sekundu, v tomto případě je nastavena na jednu. V souboru vlastností „Shape“ se upraví velikost vytvářeče částic na oblast, ve které se tyto kapky mají vyskytovat, toto je možné buď pomocí vlastnosti „Scale“ na ose x a ose y, nebo pomocí myši po kliknutí na možnost „Shape gizmo editing mode“. Jako poslední se v souboru vlastností „Color over Lifetime“ nastaví gradient barvy pomocí módu „Blend“ k dosažení iluze zmizení kapky.

V případě slunečních paprsků je síla gravitace nastavena na 0, díky tomu zůstanou paprsky na místě. Hlavními vlastnosti pro vytvoření dobře vypadajících paprsků je doba životnosti těchto paprsků, počet paprsků a rychlost simulace. U těchto vlastností se životnost nastaví na 6 sekund, počet na 6 paprsků a rychlost na 0.1 neboli 10 %. „Rate over Time“ je poté nastaven na 1 paprsek za sekundu. Gradient těchto paprsků je nastaven tak, aby vypadalo, že snižují a zvyšují na intenzitě pomocí vlastnosti „Color over Lifetime“.

Efekt poletujících částic je nejvíce měněný efekt kvůli lokaci úseků, kde je tento efekt zobrazen. U tohoto efektu je nově nastavená vlastnost „Start Lifetime“ o dvou hodnotách, tyto hodnoty přestavují, jak dlouho budou jednotlivé částice na mapě. Při vytvoření částice se vybere náhodné číslo mezi těmito hodnotami a přiřadí se jí jako životnost. V souboru vlastností „Shape“ se vybere vlastnost „Shape“ a zde „Edge“. Pomocí tohoto tvaru se částice budou vytvářet pouze z této přímky. Tato přímka se poté nastaví pomocí

možnosti „Shape gizmo editing mode“ na velikost úseku, kde se tyto částice mají vyskytovat. V souboru vlastností „Velocity over Time“ se pod vlastností „Linear“ vyskytují dvě pole pro každou osu. Do těchto polí se přiřadí čísla, podle kterých se určuje rychlost částice. Tyto pole fungují stejně jako u vlastnosti „Start Lifetime“ a to tak, že se při vytvoření částice vybere náhodné číslo mezi těmito hodnotami. V souboru vlastností „Color over Lifetime“ se vybere vlastnost „Random between two gradients“ a vytvoří se dvě barvy, přičemž spodní barva bude zcela průhledná. Pomocí této vlastnosti vznikne iluze lineárního zmizení částice. V obrázku 14 lze vidět všechny tyto efekty najednou.

Obrázek 14: Speciální efekty paprsků slunce, částic a kapky vody

Zdroj: Autor (2023)

6 Závěr

V této práci se podařilo vytvořit kompletní 2D plošinovou hru inspirovanou jinými plošinovými hrami. K dosažení tohoto cíle byl použit herní engine Unity a bylo zde vytvořeno chování všech typů nepřátel, překážky, chování objektů ve světě, normální pohyb a speciální pohyby hlavní postavy, přepínání kamer a animace různých objektů. Dále zde byla vytvořena celá herní mapa, ve které se mění obtížnost tím, jak daleko uživatel pokročil. Mapa také obsahuje světelné prvky v podobě rostlin, které nabízí navigaci hráči. Pro doplnění atmosféry je ve hře použito několik speciálních efektů, v podobě slunečních paprsků, prachu a kapek vody.

Pro vytváření spritů byla zvolena aplikace generativní umělé inteligence Leonardo AI, kde byly použity assety, které byly stáhnuty z Unity Store a z itch.io. Bylo zde vytvořeno mnoho prvků, jako jsou kusy skal, vegetace a části nepřátel. Pro vylepšení těchto prvků a odstranění pozadí byl použit program Photoshop. Po vylepšení byly tyto sprity dále použity ve hře a plynule tak zapadají do mapy.

Po dokončení vývoje byla hra předána lidem, kteří hru otestovali a předali zpětnou vazbu. Podle zpětné vazby se hra upravila do té podoby, dokud nedosáhla příjemné hratelnosti a na mapě se nacházelo minimum chyb.

Na základě těchto výsledků lze považovat náplň této práce za dokončenou.

I. Summary and keywords

This thesis focuses on the complete development of a 2D game created in the Unity game engine using the C# programming language. The development includes programming game functions with the C# language and visualizing the player model, world, and objects with different graphic sprites. The work aims to emulate other well-known platformer games, incorporating gameplay mechanics that enable the player to traverse the map faster through various types of movement. The project involves programming different hostile AIs, and the game follows the platformer style with a large level where the difficulty increases as the player progresses. Some sprites are custom-created using generative AI and Photoshop. The final product is an entertaining game that aims to captivate its audience.

Key words: 2D game development, Unity game engine, C# programming language, sprite creation, level design, game design

II. Seznam literatury

1. Arsalan. (2023). What are Gaming Assets? ITS. Získáno z https://it-s.com/what-are-gaming-assets/
2. BillWagner. (2023, Květen 4). A tour of C# - Overview. Získáno z https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
3. Burtt, G. (2020). How An Ionizing Particle From Outer Space Helped A Mario Speedrunner Save Time. TheGamer. Získáno z https://www.thegamer.com/how-ionizing-particle-outer-space-helped-super-mario-64-speedrunner-save-time/
4. Christensson, P. (2023). Script. techterms.com. Získáno z https://techterms.com/definition/script
5. CODE Magazine, EPS Software Corp., Chris Woodruff and Maarten Balliauw. (b.r.). Building a .NET IDE with JetBrains Rider. Získáno z https://www.codemag.com/article/1811091/Building-a-.NET-IDE-with-JetBrains-Rider
6. freeCodeCamp.org. (2021). Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There. freeCodeCamp.org. Získáno z https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/
7. Gaming, N. (2022). The evolution of platform games in 9 steps. Red Bull. Získáno z https://www.redbull.com/in-en/evolution-of-platformers
8. Griffiths, M. D. (2005). Video games and health. BMJ, 331(7509), 122–123. https://doi.org/10.1136/bmj.331.7509.122
9. Hollow Knight review. (2017, Červenec 10). Získáno z https://www.pcgamer.com/hollow-knight-review/
10. How do Vector Graphics Work? | CorelDRAW. (b.r.). Získáno z https://www.coreldraw.com/en/learn/guide-to-vector-design/how-do-vector-graphics-work/
11. iStock Blog. (2023). What Are Vector Graphics and How Best to Use Them. iStock Blog. Získáno z https://marketing.istockphoto.com/blog/vector-graphics/
12. Juviler, J. (2022, Říjen 31). What Is GitHub? (And What Is It Used For?). Hubspot. Získáno z https://blog.hubspot.com/website/what-is-github-used-for
13. Kinsta. (2022, Prosinec 13). What is GitHub? A beginner's introduction to GitHub. Získáno z https://kinsta.com/knowledgebase/what-is-github/
14. Klappenbach, M. (2021). What is a Platform Game? Lifewire. Získáno z https://www.lifewire.com/what-is-a-platform-game-812371
15. Küçükkarakurt, F. (2021). INTRODUCTION TO GAME GRAPHICS. DEV Community. Získáno z https://dev.to/fkkarakurt/introduction-to-game-graphics-3l89
16. MasterClass. (2021, Červenec 9). Learn About Platform Game: 7 Examples of Platform Games - 2023 - MasterClass. Získáno z https://www.masterclass.com/articles/platform-game-explained

17. Ori and the Will of the Wisps Game Review. (2020, Březen 16). Získáno z https://www.commonsensemedia.org/game-reviews/ori-and-the-will-of-the-wisps
18. Patrick_Yu. (2022). What are Platform Games? Acer Corner. Získáno z https://blog.acer.com/en/discussion/212/what-are-platform-games
19. PCMag. (b.r.). Definition of video game. Získáno Červenec 9, 2023, z https://www.pcmag.com/encyclopedia/term/video-game
20. Ramlochan, S. (2023). Leonardo AI: A Game-Changer in AI Art. Prompt Engineering. Získáno z https://www.promptengineering.org/leonardo-ai-a-game-changer-in-ai-art/
21. Research Guides: All About Images: Raster vs. Vector Images. (b.r.). Získáno z https://guides.lib.umich.edu/c.php?g=282942&p=1885352
22. Sayers, S. (2022, Září 5). Why Graphics Are So Important In. . . PlayStation Universe. Získáno z https://www.psu.com
23. Sinicki, A. (2021). What is Unity? Everything you need to know. Android Authority. Získáno z https://www.androidauthority.com/what-is-unity-1131558/
24. Smith, J. (b.r.). What is Photoshop. Získáno z https://www.agitraining.com/adobe/photoshop/classes/what-is-photoshop
25. Sprite (Concept) - Giant bomb. (b.r.). Získáno z https://www.giantbomb.com/sprite/3015-491/
26. Unity Technologies. (2018, červenec 31). Unity - Manual: Prefabs. Získáno z https://docs.unity3d.com/Manual/Prefabs.html
27. Video game - New World Encyclopedia. (b.r.). Získáno Červenec 9, 2023, z https://www.newworldencyclopedia.org/entry/Video_game#cite_note-1
28. Von Ehren, S. (2020, Červen 12). Why Do People Love Games? The New York Times. Získáno z https://www.nytimes.com
29. Weinreb, B. (b.r.). What Are Raster Graphics? Definition, Terms, and File Extensions. Získáno z https://learn.g2.com/raster-graphics
30. What is a 2D Computer Graphic? (2023). Easy Tech Junkie. Získáno z https://www.easytechjunkie.com/what-is-a-2d-computer-graphic.htm
31. Worrall, W. (2021). What Is a Platformer in Gaming? MUO. Získáno z https://www.makeuseof.com/platform-games-explained/
32. Zenva. (2023). What Is Unity? - A Guide For One Of The Top Game Engines - GameDev Academy. GameDev Academy. Získáno z https://gamedevacademy.org/what-is-unity/

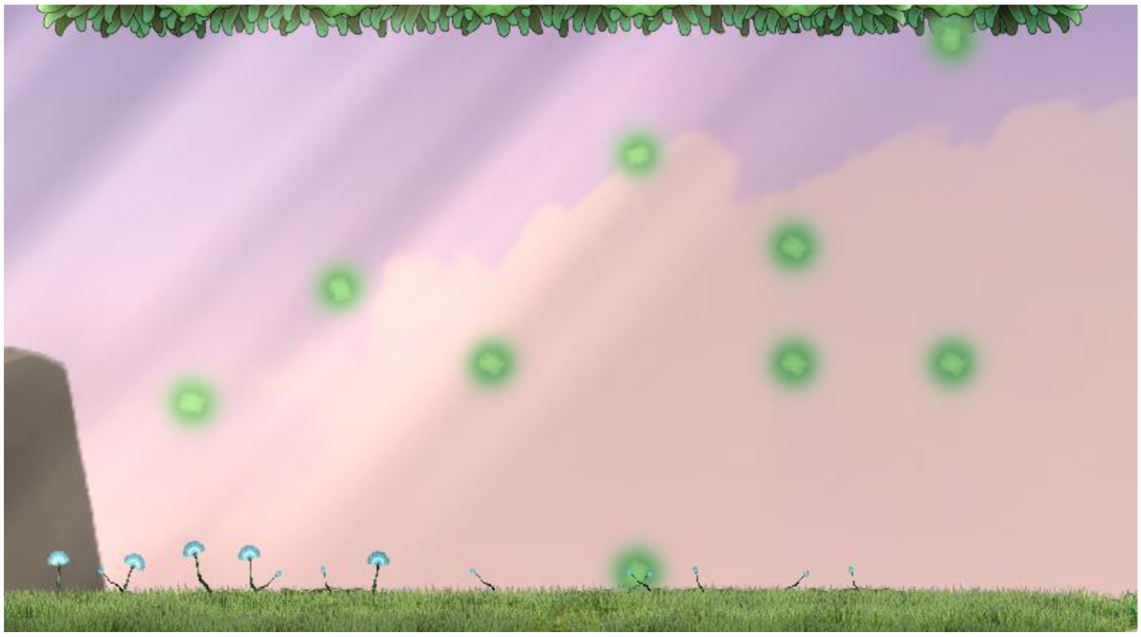
III. Seznam obrázků

IV. Seznam výpisů kódu

V. Seznam příloh

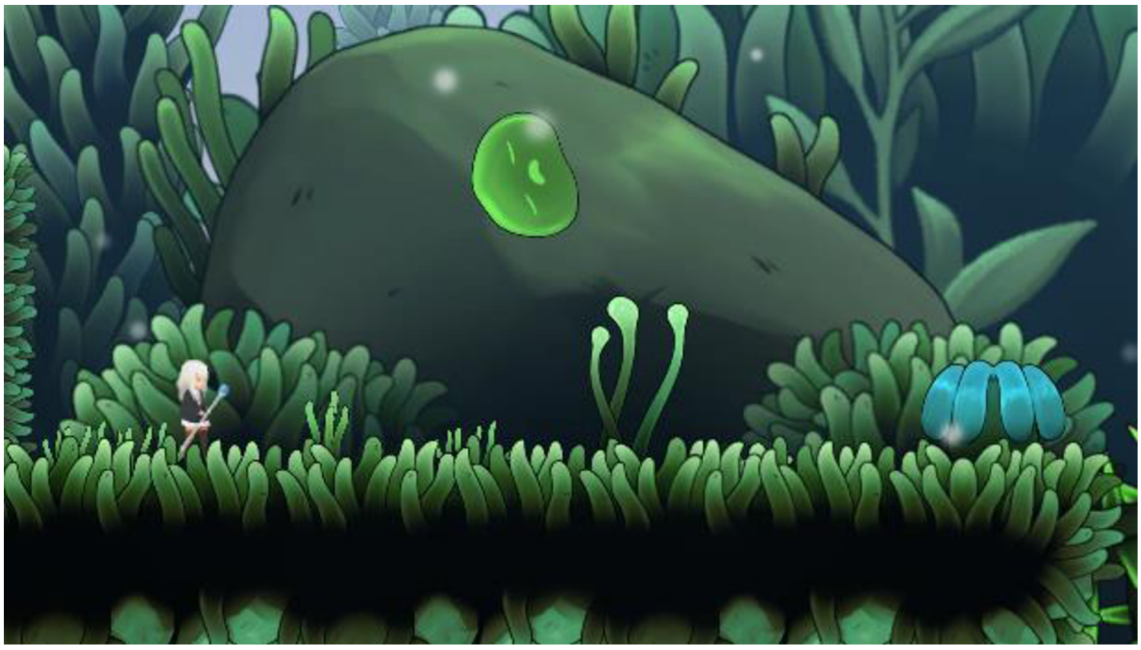
VI. Přílohy

Příloha 1: Úsek s projektily



Zdroj: Autor (2023)

Příloha 2: Úsek s nepřítelem Hoppter



Zdroj: Autor (2023)

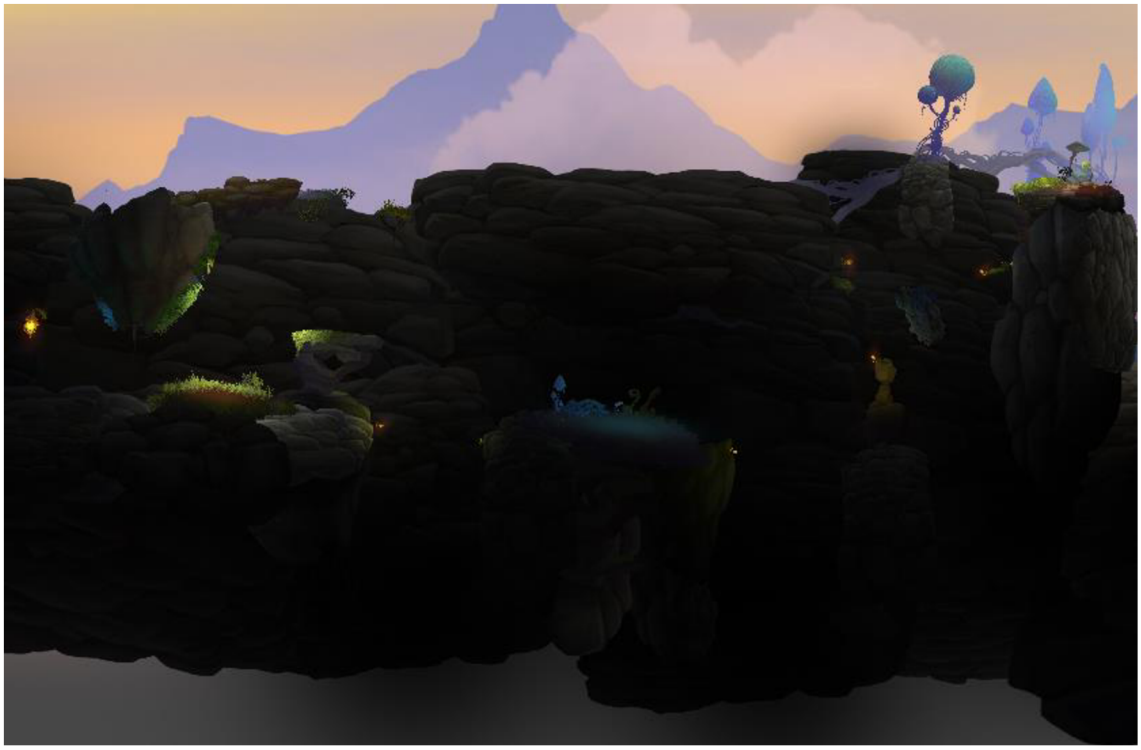
Příloha 3: Úsek s bodáky

Zdroj: Autor (2023)

Příloha 4: Úsek s naváděcími rostlinami

Zdroj: Autor (2023)

Příloha 5: Část mapy v jeskyni



Zdroj: Autor (2023)

Příloha 6: Část mapy se sprity vytvořené pomocí Leonardo AI

Zdroj: Autor (2023)

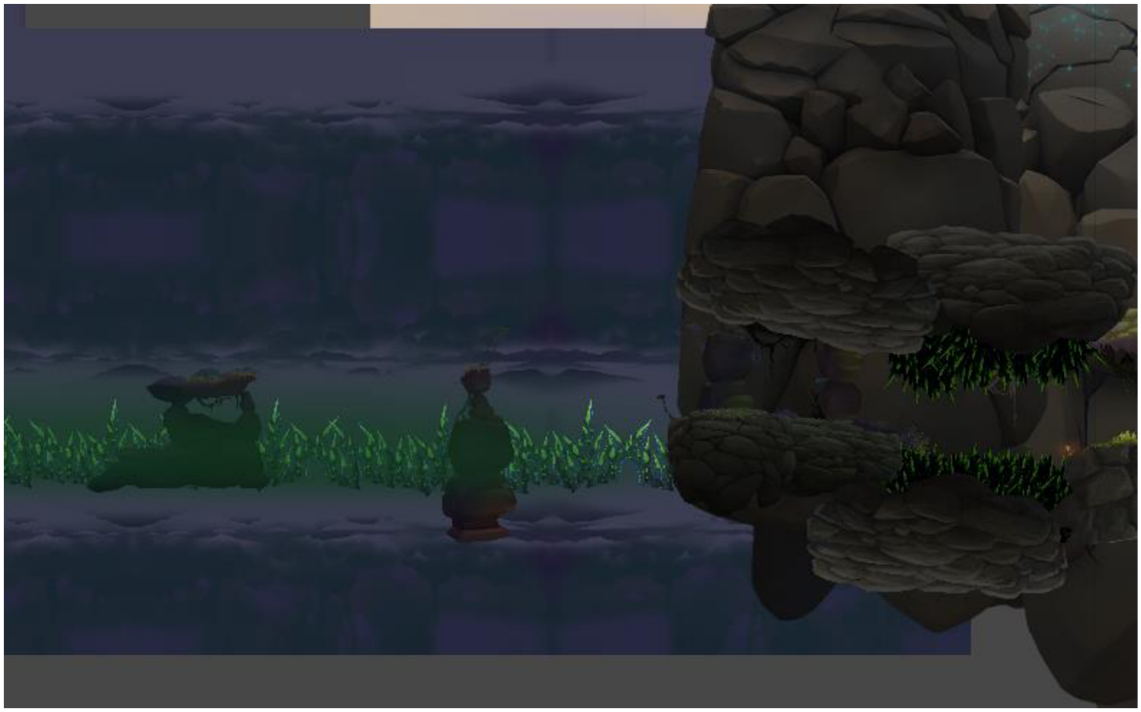
Příloha 7: Část „podzemní“ mapy

Zdroj: Autor (2023)

Příloha 8: Ostrov v „podzemí“, kde si hráč odemkne speciální pohyb „Dash“

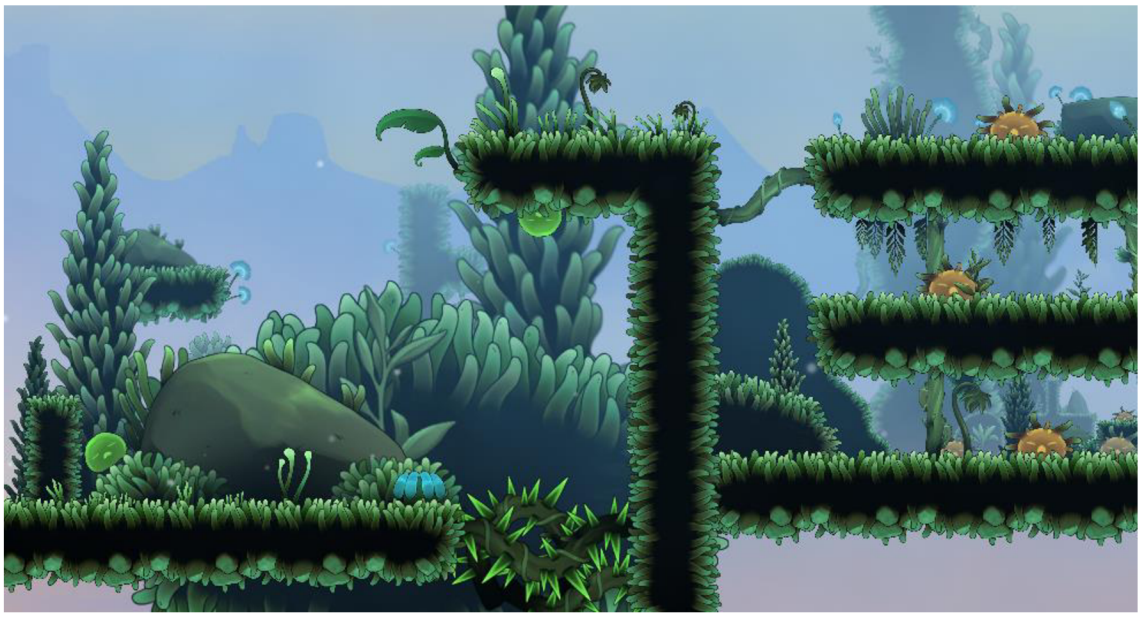
Zdroj: Autor (2023)

Příloha 9: Úsek, kde se uživatel dostane zpět na „povrch“



Zdroj: Autor (2023)

Příloha 10: Úsek s nepřátelskou umělou inteligencí



Zdroj: Autor (2023)