Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj pixel art 2D hry v Unity

Dmitriy Shemet

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Dmitriy Shemet

Systémové inženýrství a informatika
Informatika

Název práce

Vyvoj pixel art 2D hry v Unity

Název anglicky

Development of pixel art 2D games in Unity

Cíle práce

Bakalářská práce je zaměřena na vývoj herních aplikací v prostředí Unity. Hlavním cílem práce je popis metod a postupů souvisejících s vývojem 2D plošinových her v grafickém stylu pixel art. Dílčím cílem práce je vývoj prototypu herní aplikace, která bude dané postupy demonstrovat.

Metodika

Teoretická část práce bude založena na studiu odborných informačních zdrojů. Na základě získaných informací budou stanoveny teoretická východiska pro praktickou část práce.

Praktická část bude věnována vývoji prototypu 2D herní aplikace, který bude zaměřen zejména na demonstraci pixel art grafického stylu, vytváření animací, úpravu fyziky a implementaci databáze. Aplikace bude následně otestována a budou stanoveny možnosti dalšího vývoje.

Doporučený rozsah práce

35-40 stran

Klíčová slova

2D, Herní aplikace, Unity Engine, Vývoj SW, Pixel art, C#

Doporučené zdroje informací

CALABRESE, Dave, 2014. Unity 2D Game Development. 1.vyd. Packt Publishing Limited. ISBN 978-1-84969-256-4.

C# documentation [online]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/

HALPERN, Jared, 2019. Developing 2D Games with Unity: Independent Game Programming with C#. 1.vyd. New York: Apress. ISBN 978-1-4842-3771-7.

MAREŠ, Amadeo, 2011. 1001 tipů a triků pro C# 2010. Brno : Computer Press. ISBN: 978-80-251-3250-0.

Unity Documentation [online]. Dostupné z: https://docs.unity3d.com/Manual/index.html

VIRIUS, Miroslav, 2020. Programování v C#. Praha: Grada. ISBN: 978-80-271-1216-6.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Tomáš Benda

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 09. 10. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Vývoj pixel art 2D hry v Unity“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury i dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Tomáši Bendovi za odborné vedení a cenné rady. Děkuji Vám za čas a úsilí, které jste věnoval recenzi mé práce, a Vaše konstruktivní kritika a rady výrazně pomohly zlepšit její kvalitu.

Vývoj pixel art 2D hry v Unity

Abstrakt

Tato práce je zaměřena na vytvoření návrhu 2D pixelové plošinové hry ve vývojovém prostředí Unity s využitím programovacího jazyka C# spolu s editorem kódu Visual Studio a programů pro pixelovou grafiku, jako jsou Krita a Pixilart, a také programu pro tvorbu zvukových efektů Jsfxr.

Teoretická část je zaměřena na popis hlavních částí vývojového prostředí Unity potřebných při vývoji 2D her. Úvodem je čtenář seznámen s procesem vývoje hry. Dále jsou popsány klíčové součásti herního enginu Unity, historie pixel artu a možné perspektivy 2D zobrazování. Poté následuje popis klíčových funkcí programů Krita a Pixilart spolu s hlavními funkcemi aplikace Visual Studio určenými pro práci s Unity. Nakonec je představen systém Jsfxr.

Výsledkem praktické části je plně funkční prototyp hry. Ten se skládá z jedné lokace, kde se hráč musí dostat do obchodu, a přitom se vyhnout všem překážkám. Tento prototyp je připraven k dalšímu vývoji a vylepšení.

Klíčová slova: plošinové hry, Unity, C#, 2D, prototyp, pixel art

Development of pixel art 2D game in Unity

Abstract

This work focuses on creating a 2D pixel platform game in the Unity development environment using the C# programming language along with the Visual Studio code editor and pixel graphics programs such as Krita and Pixilart, as well as the sound effects program Jsfxr.

The theoretical part focuses on describing the main parts of the Unity development environment needed for 2D game development. The introduction introduces the reader to the game development process. Then, the main components of the Unity game engine, the history of pixel art and possible perspectives of 2D imaging are described. Then follows a description of the key features of Krita and Pixilart, along with the main Visual Studio features aimed at working with Unity. Finally, the Jsfxr system is introduced.

The practical part results in a fully functional prototype of the game. This consists of a single location where the player must get to the store and avoid all obstacles. This prototype is ready for further development and improvement.

Keywords: platform games, Unity, C#, 2D, prototype, pixel art

Obsah

1 Úvod

V poslední době je pixel art opět velice populárním způsobem pro vytváření herní grafiky, vrátil se zpět – i s mnoha vývojáři, kteří se obrací k tomuto médiu, aby vytvořili vizuálně působivé a esteticky příjemné hry.

Každá hra má základnu pomoci, která je nebo bude stvořena. Jednou z takzvaných základen je Unity engine, který je používán nejen pro vytvoření her, ale například v architektuře, inženýrství atd. V tomto enginu mají vývojáři k dispozici širokou řadu nástrojů a funkcí pro vytváření 2D a 3D her. Vývoj 2D pixel art her zahrnuje celou řadu úkolů a úvah, včetně tvorby sprite sheetů, animací, používání technik pro stylizaci a úpravu pixel artu, navrhování a implementaci herních mechanik a funkcí pomocí skriptů. Sprite sheet je grafický soubor složený z několika menších obrázků, které jsou spojeny do jednoho velkého celku (Löw, 2023). Design úrovní je také důležitým aspektem vývoje her a pixel art může být využit k vytvoření zajímavých a zábavných herních prostředí. Navíc lze použít různé efekty, jako je „Parallax scrolling“, k přidání hloubky do herního světa. „Parallax scrolling“ označuje efekt, při kterém se zdá, že vzdálenější objekty se pohybují pomaleji než blízké. Tento efekt se často využívá v různých oblastech, jako jsou filmové efekty, animace a videohry (Encora, 2021).

Tato bakalářská práce se věnuje zejména vývoji 2D pixel art her v Unity enginu. Rovněž slouží k seznámení čtenáře s etapami vývoje her a složkami spojenými s vývojem 2D her Unity enginu.

2 Cíl práce a metodika

2.1 Cíl

Tato bakalářská práce je zaměřena na vývoj herních aplikací v prostředí Unity. Jejím hlavním cílem je popis metod a postupů souvisejících s vývojem 2D plošinových her v grafickém stylu pixel art. Dílčím cílem je seznámit čtenáře s procesem vývoje prototypu hry a demonstrovat klíčové části vývojového prostředí Unity, a také představit styl grafiky pro hry zvaný pixel art.

2.2 Metodika

Teoretická část práce bude založena na studiu odborných informačních zdrojů. Na základě získaných informací budou stanovena teoretická východiska pro praktickou část práce. Ta bude věnována vývoji prototypu 2D herní aplikace, který bude zaměřen zejména na demonstraci pixel art grafického stylu, vytváření animací, úpravu fyziky. Aplikace bude následně otestována a budou stanoveny možnosti dalšího vývoje.

3 Teoretická východiska

3.1 Vývoj her

Tato kapitola se zabývá seznámením čtenáře s hlavními etapami vývoje her, kterými prochází hra jak u velkých společností, tak také u jednotlivých vývojářů.

Vývoj hry je proces, který vyžaduje spolupráci v týmu, složeném z herních designérů, umělců, programátorů, animátorů, testerů a projektových manažerů. Přesto existují případy, kdy byly hry vytvořeny jednotlivci nebo malými skupinami herních vývojářů (Unity Technologies, 2023e; Pulse College, 2023).

Vývoj her je projekt podobný jako například založení podniku, musí mít svůj „workflow“, tedy systém, který umožňuje řídit opakující se procesy a úkoly v určitém pořadí. Tento systém se používá k organizaci činností v různých oblastech, včetně výroby, poskytování služeb, zpracování informací a dalších činností, které přinášejí hodnotu. „Workflow“ umožňuje lidem i podnikům účinně vykonávat svou práci a zajišťuje, že jsou všechny kroky vykonány v pořádku a včas (IBM, 2023b). „Workflow“ je používán pro uspořádání i řízení pracovního toku a zahrnuje všechny kroky od začátku vývoje hry, jako jsou koncept a plánování, až po dokončení hry. Je to systém, který pomáhá uspořádat práci tak, aby všichni členové týmu věděli, co mají dělat a kdy. „Workflow“ také pomáhá správně rozvrhnout čas a rozpočet pro vývoj hry, což zvyšuje efektivitu. Proces „workflow“ může být různý v různých herních studiích a projektech, ale základní princip zůstává stejný, ať už se jedná o „AAA“, „indie“, nebo mobilní hry. Hry označované jako „AAA“ jsou vytvářeny a vydávány středně velkými nebo velkými vydavateli (Unity Technologies, 2023e). „Indie“ hra je označení pro hru, která byla vytvořena malou skupinou lidí nebo jednotlivcem, bez podpory vydavatele (Computer Hope, 2017). „Workflow“ musí být dostatečně flexibilní, aby se daly zohlednit změny a revize. Vývoj her bývá často rozdělen do tří fází, jimiž jsou předprodukce, produkce a postprodukce (Stefyn, 2022).

Následující kapitoly popisují jednotlivé fáze, které jsou součástí vývoje videoher.

3.1.1 Předprodukce (Pre-production)

Začátkem každého projektu je předprodukční fáze, která slouží k definování herního konceptu, zdrojů a potřeb pro jeho realizaci. V této fázi se kladou základy projektu a hledají se odpovědi na důležité otázky, jako například, kdo je cílové publikum, jaká je konkurence nebo jaký bude odhadovaný rozpočet (Stefyn, 2022; Perforce, 2023).

Předprodukční fáze obvykle trvá od jednoho týdne do jednoho roku a zabere až 20 % celkové doby výroby. V této fázi pracuje poměrně malý tým, v němž může být producent, programátor a koncepční umělec (Stefyn, 2022; Perforce, 2023).

Předprodukce také zahrnuje fázi zpracování nápadu, která obsahuje budování příběhu a definování hlavní postavy, zasazení děje a celkového tématu hry. Tým vývojářů určuje cíle hry, ovládací prvky a začíná skicovat herní grafiku. Tato fáze je důležitá pro vytvoření kostry hry a určení estetického stylu, který se bude projevovat v celém projektu. Data získaná během předprodukční fáze jsou podkladem pro vytváření dokumentu, který je nazván „Game design document“ (dále jen GDD) (Stefyn, 2022; Tsekhansky, 2019).

3.1.1.1 Game design document

GDD je profesionální dokument, který slouží herním vývojářům k úplné definici a zdůvodnění hry, kterou vytvářejí nebo plánují vytvořit. GDD obvykle slouží jako součást nabídky pro vydavatele. Dokument obsahuje detailní popisy hry, včetně příběhu, herního stylu, postav, návrhu úrovní a dalších klíčových prvků hry (Unity Technologies, 2023e).

GDD může být užitečným nástrojem pro udržení pořádku v projektu vývoje hry, identifikaci potenciálních rizik a plánování potřebných zaměstnanců či subdodavatelů. Při vytváření dokumentu mohou vývojáři zjistit, že jejich původně zdánlivě jednoduchý nápad na hru je ve skutečnosti náročný a vyžaduje významné zdroje. Bez plánu může být pravděpodobné, že takový projekt překročí stanovené časové i finanční limity. Dále může GDD sloužit jako užitečná pomůcka pro prezentaci a financování projektu. Potenciální investoři totiž před vstupem do projektu obvykle chtějí vidět detailní plán. Navíc GDD může být také užitečným nástrojem pro uvádění hry na trh, jakmile bude dokončena. V průběhu výroby je GDD průběžně aktualizován a zpřesňován. Tyto změny mohou být způsobeny technickými nebo finančními omezeními, případně přezkoumáním, že prvotní záměry nejsou účinné ve vizuálním vzhledu nebo hratelnosti (Stefyn, 2022).

3.1.1.2 Prototypování

Prototypování se zaměřuje na vytváření raných verzí hry, které umožní zkoumat různé herní mechanismy i funkce a rozhodnout se, které z nich budou nejvhodnější pro finální hru (Unity Technologies, 2023e).

V průběhu předprodukční fáze se obvykle vytvářejí assety a prototypy (Perforce, 2023). Označení „asset“ se používá pro jakýkoli prvek, který je součástí videohry, jako jsou postavy, objekty, zvukové efekty, mapy, prostředí a další prvky (Unity Technologies, 2023e).

Prototypování slouží k ověření funkčnosti, uživatelského zážitku, hratelnosti, herních mechanismů a uměleckého směru. Tento proces má za cíl ověřit, zda má herní nápad smysl a bude fungovat. Je důležité, aby prototyp byl testován i ostatními, protože některé věci, které jsou pro vývojáře zřejmé, pro ostatní být nemusí. Prototypování může také odhalit neočekávané problémy, které mohou mít vliv na celý průběh projektu (Stefyn, 2022).

Při tvorbě prototypů se vytvářejí minimální makety herních mechanismů a pravidel, které slouží k ověření základní funkčnosti. Velké herní studie obvykle používají minimální grafické prostředky, jako skladové nebo dříve použité modely, a textury jsou použity minimálně, nebo vůbec nejsou. Tato fáze vývoje má několik označení, jako např. „bílý box“, „šedý box“ nebo „modrý box“, což se všechno vztahuje k tomu, že se jedná o hrubý náčrt hry. Tým v této fázi stráví čas psaním kódu, ověřováním základů herního designu a hraním prototypu k vylepšení návrhu (KinematicSoup, 2016).

3.1.2 Produkce (Production)

Ve výrobním procesu hry se nejvíce času věnuje fázi produkce, na níž se podílí celý tým. Tato fáze může trvat od jednoho roku do čtyř let a během ní se hra skutečně formuje. Pracuje se na dokonalosti příběhu, vytvářejí se prvky, například postavy, rekvizity a prostředí, stanovují se pravidla hry, staví se úrovně a světy, píše se kód a mnoho dalšího (Stefyn, 2022).

Producenti spolupracují s týmem, aby zajistili, že všichni pracují na stejném harmonogramu. V této fázi se také začíná s marketingem, aby byl obsah hry viděn širším publikem (Mignano, 2016).

Během fáze produkce her:

- vývojáři a designéři vytvářejí prostředí hry, které zahrnuje herní mechaniky, umělecký styl a příběh,
- modely hlavních postav a NPC jsou navrženy, animovány a vykresleny. Non-Player Character (dále jen NPC) je herní postava, která je řízena umělou inteligencí a není ovládána hráčem (Unity Technologies, 2023e),
- hlasoví herci nahrávají scénáře a dialogy, aby zajistili správný tón,
- zvukoví designéři se starají o zvukové efekty, včetně zvuků menu a různých herních situací,
- scenáristé se zabývají psaním scénářů a pojmenováváním NPC a popisů předmětů (Bramble, 2023).

Existuje řada etap, které je třeba absolvovat v průběhu celého procesu vývoje hry. První z nich je **prototyp** . Prototypy umožňují rychle vytvářet různé iterace hry, testovat je a vybírat nejlepší variantu. To může pomoci ušetřit čas i zdroje, které by mohly být potřebné pro delší vývojový cyklus. Následuje „**First Playable**“ verze, která poskytuje lepší představu o grafice a hratelnosti. „**Vertical slice**“ je plně hratelná ukázka, kterou lze použít k představení hry investorům a dalším studiím. Fáze „**Pre-alpha**“ je většinou zaměřena na vývoj obsahu a rozhodnutí, co by mělo být přidáno, nebo odstraněno, aby se zlepšila hratelnost. V „**Alpha**“ fázi jsou přidány všechny hlavní prvky a hra je plně hratelná od začátku do konce, ale některé prvky mohou být ještě nedokončené. „**Beta**“ fáze se zaměřuje na integraci veškerého obsahu i prostředků a na optimalizaci hry. Fáze „**Gold master**“ je pak finální verze hry, která je připravena k odeslání do vydavatelství a uvolnění pro veřejnost (Stefyn, 2022).

3.1.3 Postprodukce (Post-production)

Po dokončení výroby a vydání hry probíhá další fáze vývoje, během níž se někteří členové týmu zaměřují na údržbu hry (opravy chyb, tvorba patchů) nebo tvorbu doplňkového obsahu (DLC). Ostatní členové týmu se mohou přesunout k dalšímu projektu. Marketingový tým se v této fázi snaží co nejvíce propagovat hru, aby ji bylo možné co nejvíce rozšířit mezi publikum. Projektová dokumentace, zdrojové kódy a veškeré zdroje jsou shromážděny a uloženy pro případné budoucí potřeby (Stefyn, 2022; Mignano, 2016).

3.1.4 Shrnutí

Tvorba videoher představuje náročný a složitý proces, který vyžaduje hodně času. Hry mohou být vytvářeny jak týmy, tak i nezávislými vývojáři. Rozsah projektu a velikost týmu jsou klíčové faktory, které ovlivňují délku vývoje hry. Na základě zkoumaného materiálu bylo vybráno téma „Vývoj pixel art 2D hry v Unity“, které je vhodné i pro jednoho vývojáře.

3.2 Unity

K pochopení základů vývojového prostředí Unity je potřeba znát jeho hlavní součásti využívané v procesu tvorby her. Účelem této kapitoly je seznámit čtenáře s těmito komponentami za účelem lepšího porozumění vývojovému prostředí Unity.

Unity je populární herní engine, který nabízí mnoho výhod v porovnání s ostatními herními enginy na trhu. Má vizuální pracovní postup s funkcí „drag-and-drop“, podporuje skriptování pomocí jazyka C# a je vhodný pro tvorbu jak 3D, tak 2D her. S každou verzí se sady nástrojů pro obě tyto oblasti stávají propracovanějšími a snadno použitelnými.

Unity má několik úrovní licencí, včetně bezplatné verze pro projekty s příjmy do 100 tisíc dolarů, a nabízí podporu pro 27 různých platforem s využitím různých grafických rozhraní API, jako jsou Direct3D, OpenGL a Vulkan. API (Application Programming Interface), také známé jako rozhraní pro programování aplikací, je soubor pravidel definujících, jak mohou různé aplikace vzájemně komunikovat. Toto rozhraní slouží jako prostředník, který zpracovává přenosy dat mezi systémy a umožňuje společnostem otevřít data i funkce svých aplikací třetím stranám, obchodním partnerům a interním oddělením ve společnosti (IBM, 2023a). Unity Teams poskytuje spolupráci na projektech v cloudu a nepřetržitou integraci.

Od roku 2005, kdy bylo Unity uvedeno na trh, se v něm vyvinuly tisíce her a aplikací pro stolní počítače, mobilní zařízení a konzole. Některými ze známých her, které byly vytvořeny pomocí enginu Unity, byly RimWorld (Sylvester a Ludeon, 2013), Hearthstone (Blizzard Entertainment, 2014) a Cuphead (Cuphead, 2017; Halpern, 2019).

3.2.1 Unity Hub

Hlavním účelem Unity Hubu je usnadnění procesu vývoje her v Unity. Toho dociluje tím, že poskytuje uživatelům snadný přístup k různým verzím herního enginu Unity a různým nástrojům i zdrojům potřebným pro vývoj her. Kromě toho může také zlepšit správu projektů a zefektivnit práci vývojářů. Tyto funkce jsou užitečné nejen pro zkušené vývojáře, ale i pro ty, kteří s vývojem her v Unity teprve začínají.

Následující funkce je možné využít pomocí Unity Hubu. Jsou to:

- správa, stahování a instalace modulů a verzí Unity Editoru,
- vytváření a správa projektů Unity,
- prohlížení šablon, vzorových projektů a vzdělávacích materiálů pro různé úrovně dovedností,
- správa uživatelského profilu, nastavení a licencí Unity,
- zaslání zpětné vazby a získání pomoci od Unity (Docfx, 2021).

3.2.2 Pracovní plochy a prvky

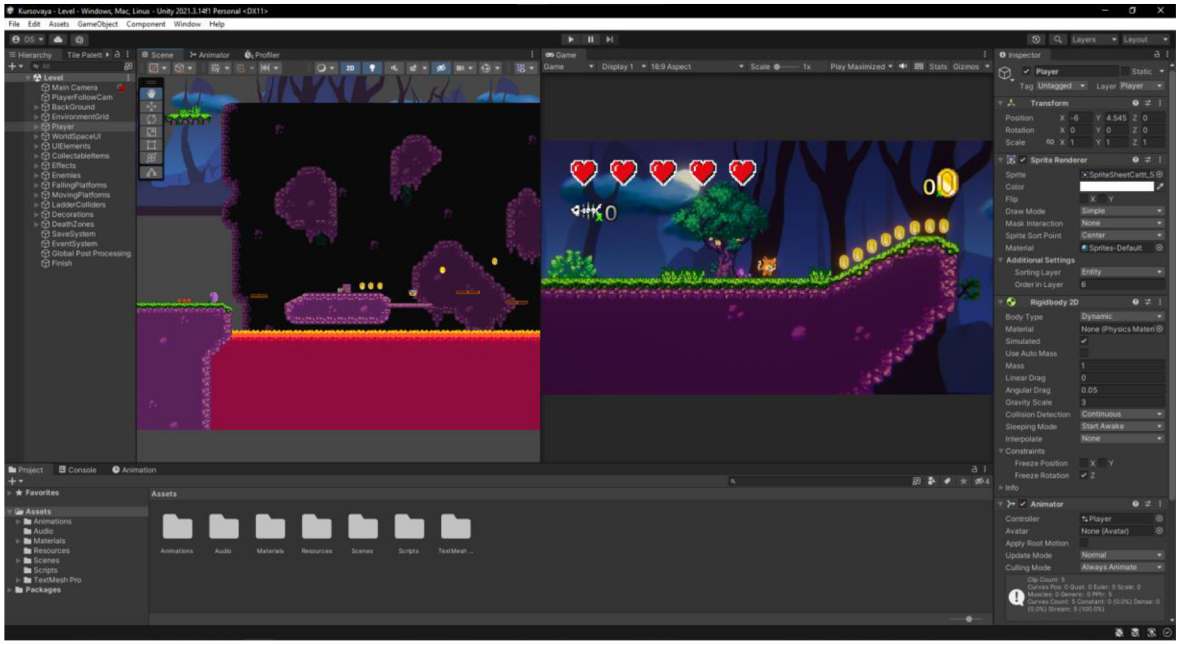
Unity obsahuje mnoho oken a klíčových prvků, které jsou základem pro práci s tímto enginem. Vývoj v Unity probíhá na dvou místech – v Unity Editoru a ve vývojovém prostředí pro editaci skriptů. Programátor má k dispozici dvě možnosti pro práci s kódem, a to jazyk C# nebo JavaScript.

3.2.2.1 Základní okna

Rozhraní editoru Unity se skládá z několika základních oken a panelů, z nichž každé má specifickou funkci. Okna lze libovolně přesouvat a přizpůsobovat. Uživatel může také vytvořit vlastní okno s novými funkcemi. Mezi nejčastěji používaná okna patří „Project“ a „Hierarchy“, „Inspector“, „Scene view“, „Game view“, „Console“ a „Profiler“. Existuje také okno „Animation“ a mnoho dalších.

Funkce a možnosti těchto oken budou vysvětleny na základě oficiálního manuálu od Unity.

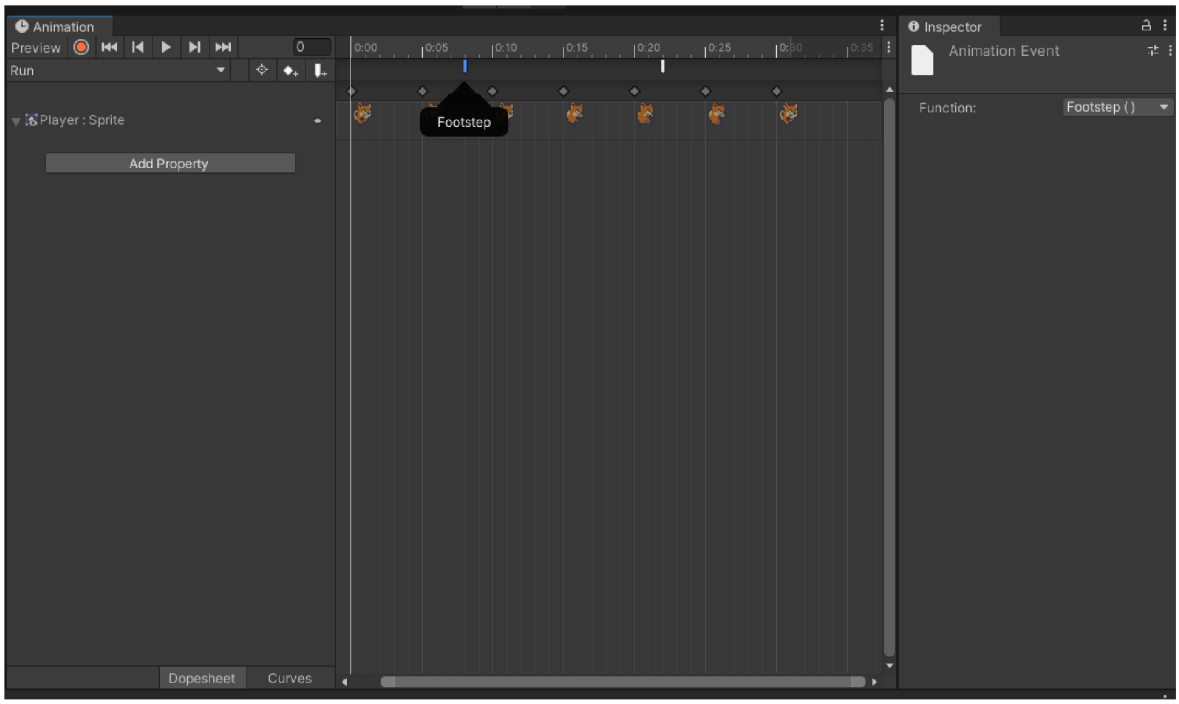
Obrázek 1: Vzhled Unity Editoru (vlastní zpracování)



- **„Project“** zobrazuje všechny datové zdroje, které jsou k dispozici pro použití v projektu. Pokud jsou do projektu importovány nové datové zdroje, zobrazí se zde jako nové položky.
- **„Hierarchy“.** Toto okno reprezentuje každý herní objekt ve scéně v hierarchické textové formě. Každý prvek na scéně má svou položku v hierarchii, takže obě okna jsou navzájem propojena. „Hierarchy“ poskytuje informaci o struktuře vzájemného propojení herních objektů.
- **„Scene view“** umožňuje uživateli vizuálně procházet a upravovat scénu. V závislosti na typu projektu, na kterém se pracuje, může zobrazovat herní prostředí v 2D nebo 3D perspektivě.
- **„Game view“** slouží k simulaci finálního vykreslení hry prostřednictvím kamer ve scéně. Po klepnutí na tlačítko „Přehrát“ se spustí simulace a umožní vizuálně prohlédnout hru tak, jak ji uvidí hráči.
- **„Inspector“.** V tomto okně je možné vidět a editovat všechny vlastnosti herního objektu, který je momentálně vybrán. Jelikož různé typy herních objektů mají různé sady vlastností, rozložení a obsah okna „Inspector“ se mění pokaždé, když je vybrán jiný herní objekt.
- **„Toolbar“.** Tento panel umožňuje přístup k účtu Unity a ke službám „Unity Cloud“. Obsahuje také ovládací prvky pro režim přehrávání, „Unity Search“, nabídku viditelnosti vrstev a nabídku rozvržení oken editoru.
- **„Console“**. V tomto okně se zobrazují zprávy generované editorem, jako jsou chyby a varování, které pomohou najít problémy v projektu, například při kompilaci skriptů. Tyto zprávy také upozorňují na akce, které editor provedl automaticky, například nahrazení chybějících metasouborů, což může vést k problémům v projektu.
- **„Profiler“** je nástroj, který se používá k získání informací o výkonu herní aplikace. Je možné jej připojit k zařízením v síti nebo k zařízením připojeným k počítači a otestovat, jak hra běží na platformě zamýšlené verze. Je možné tento nástroj také spustit v editoru a získat přehled o přidělování prostředků při vývoji hry. Profiler shromažďuje informace o výkonu aplikace v oblastech, jako jsou procesor, paměť, vykreslování a zvuk, a zobrazuje je v řadě grafů, takže je možné identifikovat oblasti, kde je třeba zlepšit výkon aplikace a iterovat je.

- **„Animation“** v Unity je nástroj umožňující tvorbu a úpravu animačních klipů přímo v aplikaci. Jeho design je zaměřen na to, aby byl silnou a jednoduchou alternativou k externím 3D animačním programům. Kromě animace pohybu má tento editor také schopnost animovat materiálové proměnné i komponenty a umožňuje rozšíření animačních klipů o animační události, které jsou volány v určitých bodech na časové ose (Unity Technologies, 2023a).

Obrázek 2: Ukázka okna „Animation“ v Unity Editoru



3.2.2.2 GameObject

GameObject, zvaný také herní objekt, je klíčovým prvkem v Unity Editoru. Tento prvek představuje každý objekt ve hře, jako jsou postavy, sběratelské předměty, světla, kamery a speciální efekty. Samotný herní objekt však nemá žádnou funkčnost, dokud mu nejsou přidány další komponenty. Herní objekty jsou základními stavebními bloky, ze kterých se skládají postavy, rekvizity a prostředí v Unity. Tyto objekty slouží jako kontejnery pro komponenty, které poskytují další funkcionalitu. V herní scéně jsou herní objekty neviditelné a obsahují pouze komponentu „Transform“. Pokud jsou herní objekty součástí uživatelského rozhraní, namísto komponenty „Transform“ obsahují komponentu „RectTransform“ (Unity Technologies, 2023c).

3.2.2.3 Komponenty

V Unity Editoru jsou komponenty klíčovými funkčními částmi každého herního objektu. Každá komponenta definuje chování i vlastnosti určitého objektu a lze je upravit v okně „Inspector“ po výběru daného herního objektu. Jeden herní objekt může mít mnoho komponent, k těm nejčastěji používaným se řadí například „Transform“, „Sprite Renderer“, „Collider 2D“, „Rigidbody 2D“ a „Animator“. Dále je možné vytvořit vlastní komponentu a přidat ji do herního objektu pomocí okna „Inspector“.

Oficiální manuál od Unity poskytuje podrobné vysvětlení funkcí a možností komponent, které jsou dostupné v Unity. Na základě tohoto manuálu budou následující komponenty vysvětleny.

- **„Transform“.** Tato komponenta slouží k ukládání informací o pozici, natočení a velikosti herního objektu. Tyto informace se vztahují k předkovi daného objektu nebo k počátku scény, pokud objekt nemá žádného předka. „Transform“ je vždy přítomen v každém herním objektu a nelze ho odstranit ani vytvořit herní objekt bez něj.
- **„Sprite Renderer“** je komponenta v Unity, která slouží k vykreslení spritů a řídí jejich vizuální vzhled v rámci 2D i 3D projektů. V kontextu 2D grafiky jsou sprity 2D grafické prvky, které mohou být použity k vykreslení obrazů, postav a pozadí ve hře (Unity Technologies, 2023d). Když je vytvořen nový sprite, automaticky se vytvoří i herní objekt s připojenou komponentou „Sprite Renderer“. Tuto komponentu je možné přidat i do již existujícího herního objektu pomocí okna „Inspector“.
- **„Collider 2D“.** Jde o komponentu určující tvar 2D herních objektů pro účely fyzických kolizí. „Collider 2D“ pomáhá definovat fyzický tvar herních objektů a určuje, jak budou interagovat s ostatními herními objekty ve scéně. Tyto kolizní prvky jsou neviditelné a nemusí mít přesně stejný tvar jako samotný herní objekt. Většinou je hrubá aproximace dostatečná a při hře není rozdíl patrný. Je důležité si uvědomit, že všechny „Colliders“ pro 2D herní objekty mají názvy končící na „2D“. Kolizní prvky, které nemají v názvu „2D“, jsou určeny k použití pro 3D herní objekty. Je důležité dodržovat pravidlo, že nelze míchat 3D herní objekty a „Colliders 2D“, ani 2D herní objekty a „Colliders“.

Typy „Colliders 2D“ jsou nádledující:

- **„Circle Collider 2D“.** Tento Collider má tvar kruhu, jehož poloha a poloměr jsou definovány v lokálním souřadném systému sprite.
- **„Box Collider 2D“.** Definuje obdélníkový tvar s určenou polohou, šířkou a výškou v lokálním souřadném prostoru sprite. Je důležité poznamenat, že obdélník je osově zarovnaný a jeho hrany jsou rovnoběžné s osami X nebo Y místního prostoru.
- **„Polygon Collider 2D“.** Jeho tvar je definován libovolnou hranou složenou z úseček, kterou lze upravit tak, aby odpovídala tvaru sprite nebo jinému tvaru. Nicméně aby tato komponenta fungovala, musí hrana uzavírat celou oblast.
- **„Edge Collider 2D“.** Jeho tvar je volný a tvořený úsečkami, které lze upravovat, aby odpovídaly tvaru sprite nebo jinému tvaru. Počáteční a koncové body „Collider“ se nemusí setkávat nebo uzavírat oblast, aby fungovaly, na rozdíl od „Polygon Collider 2D“, a mohou tvořit přímku nebo jiný tvar hrany.
- **„Capsule Collider 2D“.** Má tvar kapsle, který se vyznačuje absencí vrcholových rohů a souvislým kruhovým obvodem. Díky této konstrukci je „Capsule Collider 2D“ schopen efektivně se vyhýbat nárazům v rohových oblastech, které by mohly způsobit nežádoucí chování simulace. Tvar kapsle je geometricky pevný a postrádá dutiny, což umožňuje, aby se všechny ostatní 2D kolizní prvky uvnitř „Capsule Collider 2D“ považovaly za kontaktní a byly s ním v interakci. V důsledku toho jsou tyto prvky schopny být v průběhu času z „Capsule Collider 2D“ vytlačeny.
- **„Composite Collider 2D“.** Na rozdíl od většiny kolizních těles nevykazuje samostatný tvar, ale využívá kombinace tvarů „Box Collider 2D“ a „Polygon Collider 2D“, které jsou mu přiřazeny. V kontextu „Composite Collider 2D“ jsou použity vrcholy z těchto tvarů a jsou sloučeny do nové geometrie, která je řízena „Composite Collider 2D“.
- **„Effector 2D“** je komponentou, která umožňuje ovlivňovat směr fyzikálních sil v situaci, kdy se v prostředí dotýkají kolizní tělesa dvou různých herních objektů.

Existuje několik typů „Effector 2D“, ale uveden jen ten, který bude následně použit v praktické části:

- **„Platform Effector 2D“** mění chování plošiny, například umožňuje jednosměrné kolize nebo odstranění bočního tření a odrazu.

- **„Rigidbody“.** Pro aktivaci fyzikálního ovládání pohybu a polohy herního objektu je možné použít komponentu „Rigidbody“. Ta umožňuje použít simulované fyzikální síly a točivý moment k pohybu herního objektu namísto využití komponenty „Transform“. Tímto způsobem je umožněno fyzikálnímu enginu (systému, který umožňuje simulovat fyzikální vlastnosti objektů, aby mohly správně reagovat na různé vlivy, jako jsou kolize, gravitace nebo další síly, a přitom zachovávat přesné zrychlení) vypočítat výsledné efekty pohybu objektu. Mnoho konceptů ze standardní komponenty „Rigidbody“ je přeneseno i do „RigidBody 2D“, jenom s tím rozdílem, že objekty se mohou pohybovat pouze v rovině XY a mohou se otáčet výhradně kolem osy kolmé k této rovině.
- **„Particle System“** umožňuje simulovat a animovat tekutiny, jako jsou kapaliny, mraky a plameny, pomocí generování velkého počtu malých 2D obrázků ve scéně. Tyto obrázky jsou nazývány částice a tvoří vizuální efekt. Každá částice v systému představuje jednotlivý grafický prvek efektu a systém simuluje všechny částice současně pro vytvoření celkového dojmu vizuálního efektu (Unity Technologies, 2023f).

3.2.2.4 Prefab

„Prefab“ je speciálním typem herního objektu, který umožňuje uložit plně nakonfigurované herní objekty do projektu pro opakované použití. Tyto objekty pak lze sdílet mezi scénami, nebo dokonce jinými projekty, aniž by bylo nutné je znovu konfigurovat. To je velmi užitečné pro objekty, které se budou používat mnohokrát, například plošiny. Hlavní výhodou „Prefabu“ je, že se v podstatě jedná o propojené kopie assetů, které existují v okně „Project“. To znamená, že změny provedené nebo aplikované na původní „Prefab“ se rozšíří do všech ostatních instancí. To umožňuje rychlé a efektivní úpravy chyb, výměnu výtvarných prvků a další stylistické změny (Unity Technologies, 2020).

3.2.2.5 Scripting

V Unity je k dispozici široká škála komponent, v některých případech je však nutné vytvořit vlastní komponentu pro implementaci vlastní herní logiky a chování objektů. Tuto

vlastní komponentu lze vytvořit pomocí skriptů, které pak mohou být připojeny k herním objektům jako komponenty. Každý takový skript pak komunikuje s vnitřními mechanismy Unity implementací třídy odvozené od vestavěné třídy „MonoBehaviour“. Komponenty založené na skriptech pak umožňují spouštět herní události, kontrolovat kolize objektů, aplikovat fyzikální vlastnosti, programovat reakce na uživatelské ovládací prvky a další akce (Unity Technologies, 2023b).

Základní třídou každého skriptu v Unity je „MonoBehaviour“. Při vytvoření nového skriptu v jazyce C# v okně „Project“ Unity se automaticky odvozuje od této třídy a poskytuje vzorový skript. Třída „MonoBehaviour“ poskytuje framework pro připojení skriptu k hernímu objektu v editoru a také poskytuje předem definované body, jako jsou „Start()“ a „Update()“, které se dají využít pro implementaci herní logiky (Unity Technologies, 2023g).

Metoda „Start()“ je v Unity vyvolána v prvním snímku po aktivaci skriptu a před prvním voláním metody „Update()“. Podobně, jako je tomu u metody „Awake()“, i metoda „Start()“ je vyvolána pouze jednou za dobu existence skriptu. Nicméně metoda „Awake()“ je volána při inicializaci objektu skriptu, bez ohledu na to, zda je skript v danou chvíli povolen, nebo ne. Pokud skript není v době inicializace povolen, metoda „Start()“ by neměla být volána ve stejném snímku jako funkce „Awake()“ (Unity Technologies, 2023h).

Metoda „Update()“ v Unity se volá pro každý snímek, každý skript uvnitř této metody se provede jednou za snímek. To znamená, že jakýkoli kód uvnitř metody „Update()“ bude vykonán jednou za každý snímek hry. Pokud například v „Update()“ bude vložen kód, který přidá jedničku do nějaké proměnné, a hra bude spuštěna při 30 snímcích za sekundu, pak se hodnota proměnné bude zvyšovat o hodnotu třicet každou sekundu (Vionix, 2021).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ExampleClass : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Ukázka kódu 1: Kód, který je generován automaticky při vytvoření nového skriptu

3.2.3 Unity Asset Store

Unity Asset Store obsahuje knihovnu bezplatných a komerčních datových zdrojů, které vytváří společnost Unity Technologies a členové komunity. K dispozici je široká řada datových zdrojů, včetně textur, modelů, animací, příkladů celého projektu, výukových lekcí a rozšíření pro Unity Editor (Unity Technologies, 2023e). Assety z Unity Asset Store mohou pomoci vytvořit animace pro postavy, objekty a rozhraní. Je možné najít hotové materiály, textury a sprity, které usnadní tvorbu krásných a efektních grafických prvků pro hry. Také assety jsou užitečné při prototypování nápadů, což může výrazně zkrátit dobu vývoje.

3.3 Pixel art

Pixel art je forma digitálního umění, ve které jsou obrázky vytvářeny a upravovány na úrovni pixelů pomocí softwaru pro úpravu grafiky. Jeho jedinečný vizuální styl je založen na použití jednotlivých pixelů jako stavebních bloků pro tvorbu obrazu. Tento styl bývá často používán pro vytvoření malých a detailních obrázků nebo animací s nízkým rozlišením, které jsou obvykle zobrazovány na malých obrazovkách, jako jsou například obrazovky mobilních telefonů nebo hracích konzol. Pixel art může být také použit pro vytvoření stylizovaného vzhledu pro hry nebo animace. Bývá často spojován s retro styly a používá se pro vytváření efektu nostalgie (Techopedia, 2023). Například hry jako Celeste (Extremely OK Games, 2018) nebo Stardew Valley (ConcernedApe, 2016) dobře zachycují atmosféru starých pixelových her, vypadají však nově a moderně.

Obrázek 3: Vzhled hry Celeste (Extremely OK Games, 2018)

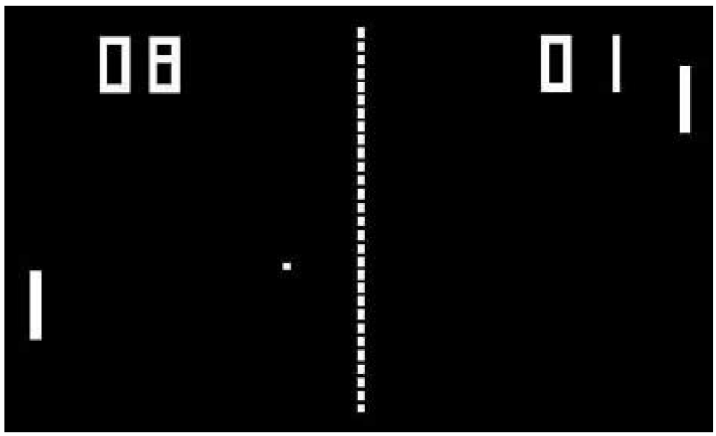
3.3.1 Historie pixelového umění v herním vývoji

Vznik pixel artu se datuje do 70. let 20. století, kdy byl v porovnání s dnešním digitálním uměním velmi jednoduchý. Vzhledem k tehdejším technologickým omezením a nedostatečnému rozvoji oboru byli tvůrci omezeni pouze na určitou úroveň grafických prvků, kterou mohli vytvořit. Některé z prvních pixel artů byly založeny pouze na jednoduchých tvarech, jako jsou čtverce a obdélníky (Barnes, 2022).

První léta (1972–1983)

Pixel art se objevil v raných časech počítačových her a byl ovlivněn technologickými omezeními i nedostatkem zkušeností vývojářů s pohyblivými hrami. To vedlo ke vzniku jednoduchých pixelových bloků, které se snažily znázornit objekty a velmi spoléhaly na představivost hráčů, aby doplnily prázdná místa. Konzole jako Magnavox Odyssey, ColecoVision a Atari 2600 byly mezi prvními, které používaly pixel art (Griffiths, 2018).

Obrázek 4: Screenshot z PONG z Atari Arcade Hits (Bumm13, 2006)



Osmibitová éra (1983–1987)

S rozšiřujícími se schopnostmi technologií a získáváním zkušeností se vývojáři začali snažit vytvořit bohatší herní světy s rozpoznatelnými postavami a přidali více detailů do pozadí i skrytých oblastí. Došlo také k omezeným pokusům o replikaci filmových cutscén. Používání zkratek pro vytváření příběhů vedlo k vývoji mnoha moderních herních příběhů. V tuto dobu konzole jako Nintendo Entertainment System (NES), Sega Master System a Game Boy používaly pixel art ve svých hrách (Griffiths, 2018).

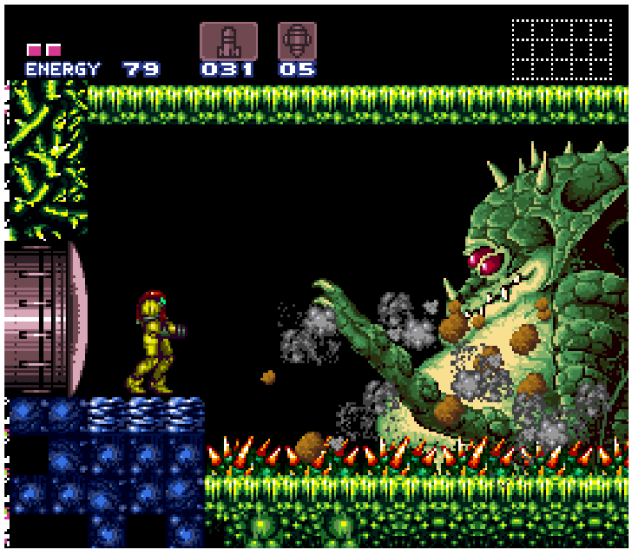
Obrázek 5: Screenshot z Mega Man 2 (Capcom, 1989)

Šestnáctibitová éra (1987–1993)

Postupem času se pixel art vyvinul tak, že už mohl plně replikovat to, co bylo dříve dostupné pouze v arkádách. Tvůrci her se cítili dostatečně sebevědomě, aby se odklonili od svých arkádových kořenů, vytvořili vlastní odlišné herní světy a mohli se měřit s novějšími hrami (Griffiths, 2018).

Některé hry se dokonce pokoušely kombinovat pixel art s raným ztvárněním 3D ve snaze zvýšit účinnost. Konzole jako Super Nintendo Entertainment System (SNES), Sega Genesis a Neo Geo používaly tou dobou pixel art ve svých hrách (Griffiths, 2018).

Obrázek 6: Screenshot ze Super Metroid (Metroid Recon, 2014)



Pomalý pád pixel artu (1993–2006)

S příchodem konzolí jako PlayStation a Nintendo 64, které se snažily rozvíjet použití 3D modelů pro zobrazování postav, pixel art začal postupně ztrácet na popularitě. Od té doby se pixel art příliš nezměnil a většina her pouze vylepšila umění z předchozích konzolí, místo aby vytvořila úplně nový styl. Několik společností se rozhodlo neúčastnit se vývoje ve 3D, dokud nebylo dokonalejší, ale tyto společnosti často ztratily význam v průmyslu následkem této volby. V tuto dobu konzole jako Sega Dreamcast a Gameboy Advance používaly pixel art ve svých hrách (Griffiths, 2018).

Moderní doba (2006–současnost)

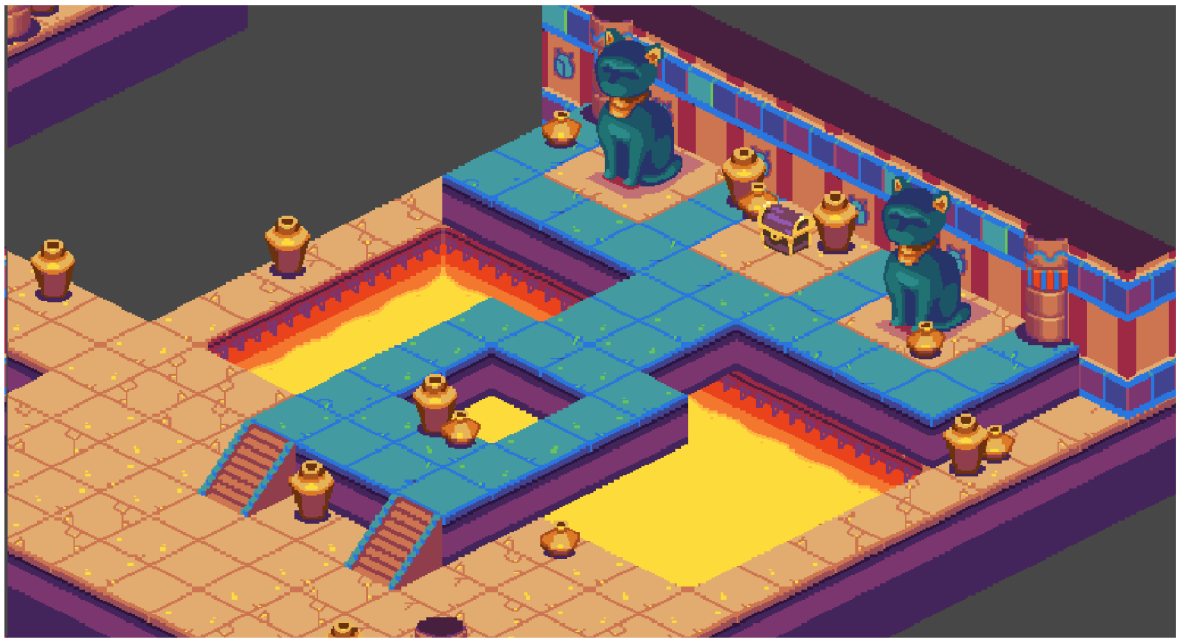
I když se 3D stalo dominantním způsobem zobrazování her, pixel art stále existuje, třebaže v menším množství než dříve. V současné době je pixel art hlavně omezen na kapesní konzole a indie hry, které se snaží o „retro“ styl. I když jeho přítomnost klesá, pixel art stále vyvíjí a má herní světy, které soutěží, a dokonce překonávají své polygonální protějšky. Konzole jako Nintendo DS/3DS a Nintendo Switch používají pixel art ve svých hrách (Griffiths, 2018).

3.3.2 2D perspektivy

Ve 2D hrách existují tři typické perspektivy: izometrická, shora dolů a boční. Každá z nich se liší v tom, jak svět hry zobrazuje, a vyžaduje jiný způsob kreslení. Izometrická perspektiva se často používá ve strategiích a simulátorech, shora dolů je více zastoupena v RPG a adventurách, boční se využívá v plošinových hrách. RPG (Role-playing games) jsou žánr elektronických her, ve kterých hráči postupují přes příběhový úkol a často řeší mnoho vedlejších úkolů, aby získali zkušenosti, které zlepšují různé atributy i schopnosti postavy nebo skupiny postav (Hosch, 2023). Tyto perspektivy poskytují různé úhly pohledu na svět hry a rozličné druhy kontroly a navigace pro hráče.

Izometrická perspektiva. Existují hry, které jsou vytvořeny s izometrickou perspektivou. To znamená, že se na krychli vidí tři viditelné strany se stejnou velikostí. Tato perspektiva umožňuje umělcům zobrazit části světa, které by jinak nebyly viditelné. Nevýhodou je, že vytváření může trvat déle, než je tomu u jiných perspektiv pixel artu (Wain, 2022).

Obrázek 7: Ukázka izometrické 2D demo hry vytvořené pomocí Unity (Unity Technologies, 2023i)



Perspektiva shora dolů znamená, že hra vypadá, jako by kamera byla nad hlavou. Existují různé způsoby kreslení pixel artu pro hry s touto perspektivou. Například ve FPS (first person shooter) v reálném čase, jako je Hotline Miami (Dennaton Games, 2012), pixel art vypadá, jako by šlo o pohled z ptačí perspektivy. Na druhé straně v hrách jako Undertale (Fox,

2015) je kamera více skloněna. Způsob, jakým se kreslí postavy v těchto hrách, je boční pohled a vytvoření jedné stěny, aby bylo dosaženo tohoto efektu (Wain, 2022).

Obrázek 8: Ukázka 2D demo hry s perspektivou shora dolů vytvořené pomocí Unity (Unity Technologies, 2023i)

Boční perspektiva se používá v 2D plošinových hrách, jako je například všem známá Super Mario Bros. (Nintendo, 1985). Ačkoliv 2D plošinové hry dosahovaly vrcholu popularity v 80. a 90. letech, moderní klasiky jako Celeste (Extremely OK Games, 2018) poukazují na úspěšnost spojení pixel artu s 2D plošinovými hry (Wain, 2022).

Obrázek 9: Ukázka prototypu 2D hry s boční perspektivou vytvořené pomocí Unity (vlastní zpracování)

3.4 Ostatní nástroje pro tvorbu pixel art 2D grafiky, kódu a zvuku

3.4.1 Krita

Krita je aplikace s otevřeným zdrojovým kódem, která je zdarma a dostupná na mnoha platformách. Tato aplikace nabízí ucelené řešení pro tvorbu digitálních uměleckých souborů a je vhodná pro pravidelné, dlouhodobé a zaměřené používání. Krita se specializuje na oblasti, jako jsou ilustrace, konceptuální umění, matná malba, textury, komiksy a animace. Aplikace byla vyvinuta ve spolupráci s uživateli, což zaručuje, že podporuje skutečné potřeby a pracovní postupy. Krita také dodržuje otevřené standardy a umožňuje spolupráci s dalšími aplikacemi (Krita, 2018).

- **File Formats**. V rámci podpory souborových formátů nabízí Krita širokou škálu možností. Některé formáty však nejsou podporovány tak docela, u jiných je možný pouze import, nikoliv export. Krita umožňuje využívat metadata pro formáty souborů .kra, .ora, .tiff, .jpeg a .png.
- **Color models**. Krita neumožňuje práci s indexovanými barevnými modely a většinou neumožňuje práci s barevnými modely, které nezahrnují alfa kanál. Místo toho Krita umožňuje použití různých hloubek kanálů, včetně rozsahu od osmi bitů celých až po 32 bitů s plovoucí desetinnou čárkou na kanál.
- **Layer types**. Krita umožňuje práci s vrstvami a maskami. Masky jsou přiřazeny k určité vrstvě a slouží k omezení úprav, které se provádějí na této vrstvě. Vrstvy jsou hierarchicky seskupeny, kde vrstvy nahoře jsou viditelné, vrstvy dole naopak skryté.

Díky této struktuře lze vytvářet složité obrazy s mnoha detaily a efekty pomocí jednotlivých vrstev, které jsou navrstvené přes sebe.

- **Tools**. Krita poskytuje uživatelům širokou škálu nástrojů pro různé účely. Tyto nástroje zahrnují vektorové nástroje, které umožňují tvorbu přesných vektorových objektů s hranami a křivkami, rastrové nástroje pro malbu s pixely, nástroje pro přesné kreslení linií a křivek pomocí navádění, nástroje pro úpravy a efekty na celém plátně a nástroje pro selekci a manipulaci s jednotlivými objekty nebo oblastmi v obraze. Kromě toho Krita nabízí také „tvary“, jako jsou rozšířený text, text na cestě a geometrické tvary, které mohou být vkládány do obrazu. Všechny tyto nástroje umožňují uživatelům tvorbu složitých obrazů s mnoha detaily a efekty.
- **Brush engines**. Krita se odlišuje od jiných aplikací tím, že podporuje pluginy brush engine pro rastrové nástroje pro malbu. Tyto pluginy umožňují uživatelům měnit chování štětců při kreslení, například simulovat chování reálného štětce nebo pastelky, přidávat textury, speciální efekty a další. Krita obsahuje několik přednastavených brush engine pluginů, ale také umožňuje uživatelům vytvářet a používat vlastní pluginy, aby naplnili své individuální potřeby.
- **Filters**. Krita nabízí širokou škálu filtrů pro úpravu obrazu, které lze aplikovat přímo na pixely v aktuální vrstvě nebo použít jako filtrační vrstvu či masku filtru. To umožňuje použít filtr pouze na určitou část obrazu nebo upravit jeho účinky pomocí masky. Mezi množství různých filtrů, které Krita poskytuje, patří změna barevného tónu, úprava jasu a kontrastu, přidávání textur a šumu a další (Krita, 2009).

3.4.2 Pixilart

Pixilart je online platforma pro vytváření pixelového umění. Umožňuje uživatelům vytvářet a sdílet pixelové obrázky pomocí webového prohlížeče. Pixilart obsahuje širokou škálu nástrojů pro kreslení a úpravy pixelů, včetně štětců, barviček, výběrových nástrojů a filtrů. Kromě samotného vytváření pixelového umění mohou uživatelé na Pixilartu také sdílet svá díla, navzájem se inspirovat a zapojit se do komunity pixelových umělců.

- **Layers**. V Pixilartu se používají vrstvy pro usnadnění úpravy složitých obrázků. Vrstvy jsou velmi užitečné při práci s obrazy, které je třeba přemístit, vypnout, případně u nich upravit krytí. Uživatelé mohou vytvářet složité obrazy bez nežádoucích vedlejších efektů díky tomu, že mohou duplikovat, přesouvat, slučovat a měnit krytí vrstev.

- **Frames**. V Pixilartu se používají snímky pro tvorbu animovaných souborů GIF (Graphics Interchange Format). Uživatel si může vytvořit jakýkoliv počet snímků a může je duplikovat nebo přesouvat, přičemž může nastavit rychlost každého snímku. K dispozici je také funkce, která umožňuje zobrazit předchozí snímek jako náhled na každém snímku, což je užitečné pro návaznost snímků při tvorbě animací.
- **Pixel Perfect**. Využitím této funkce lze v programu dosáhnout precizních a čistých tahů při kreslení. Tato funkce odstraňuje zdvojené pixely, které jsou označeny červenými tečkami, čímž pomáhá zajistit, že výsledný obraz bude vypadat hladce a přesně.
- **Custom Canvas Sizes**. V Pixilartu je možné vytvářet plátna o velikosti až 700 pixelů pro tvorbu pohyblivých spritů, ikon a uměleckých děl. Uživatelé mohou stáhnout své výkresy v různých velikostech (1× nebo 50×) prostřednictvím záložky ke stažení. Kromě toho je také možné stáhnout jednotlivé vrstvy, snímky nebo celý GIF.
- **Brush Tool**. V programu Pixilart je k dispozici nástroj štětec, který umožňuje uživatelům kreslit tahy a vytvářet různé textury i efekty. Tento nástroj lze použít k vytváření uměleckých děl. Uživatelé mohou vytvářet své vlastní štětce a také využívat funkci obrysu k vytváření obrysů, které vypadají podobně jako štětec (Pixilart, 2023).

3.4.3 Visual Studio

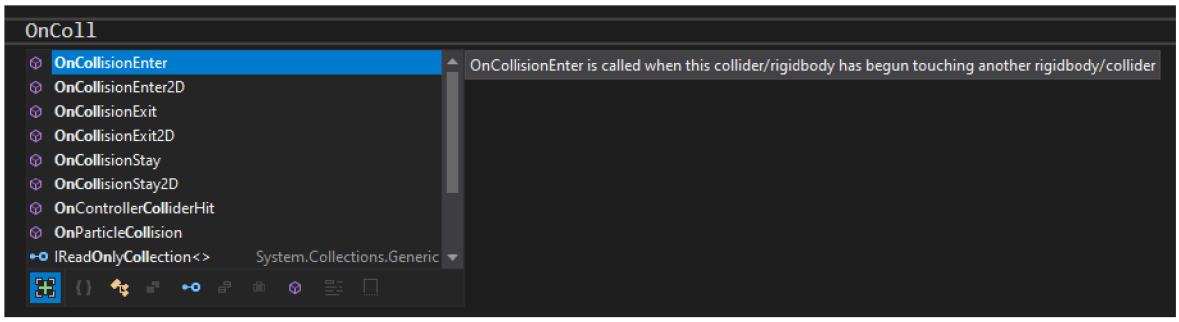
Visual Studio IDE (Integrated Development Environment) je vývojové prostředí společnosti Microsoft určené pro tvorbu aplikací pro různé platformy, jako jsou Windows, Mac, iOS a Android. Tento komplexní nástroj nabízí mnoho funkcí pro editaci, debuggování, profilování a testování kódu, stejně jako pro správu verzí a spolupráci s týmem. Existují různé verze této aplikace, jako jsou Visual Studio Community, Visual Studio Professional a Visual Studio Enterprise, které se liší v podporovaných jazycích a ve funkcích. Visual Studio IDE nachází oblibu mezi vývojáři v mnoha oblastech, jako jsou desktopové, webové i mobilní aplikace a hry (Microsoft, 2023b).

Pro vypracování praktické části práce byla použita verze Visual Studio Community. Visual Studio IDE nabízí mnoho užitečných funkcí pro práce s Unity.

Níže si představíme některé z nich:

- **Unity Event Functions**. Použitím technologie IntelliSense lze snadno a rychle přidávat funkce událostí Unity, například „Start“, „Update“ a „OnCollisionEnter“, do skriptů v jazyce C# pomocí několika klávesových zkratek. IntelliSense nabízí automatické dokončování návrhů, a usnadňuje tak rychlé a přesné přidávání funkcí událostí.

Obrázek 10: Příklad práce s IntelliSence (Microsoft, 2021)

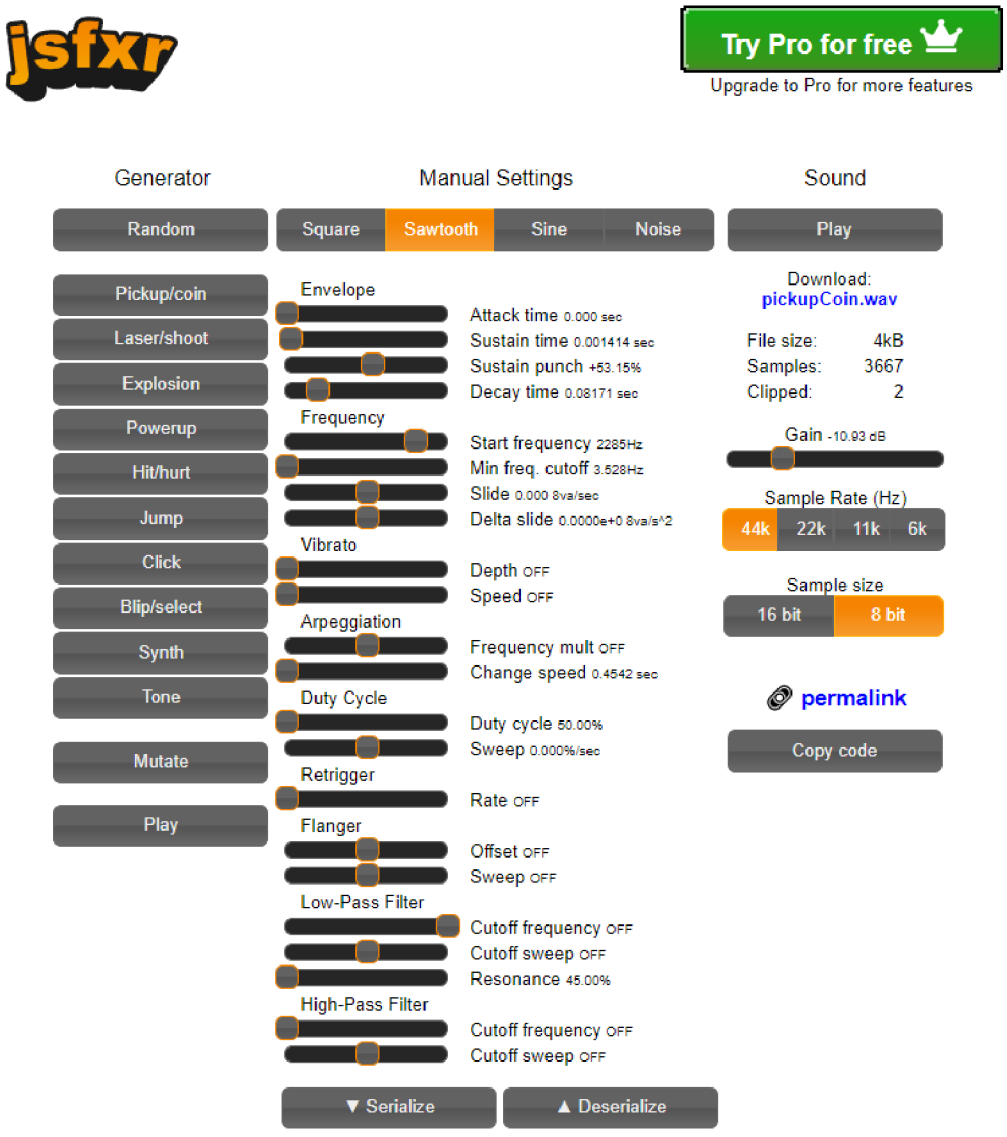


- **High-performance debugger**. Nástroje Visual Studio pro Unity nabízejí vývojářům robustní funkce ladění, které jsou standardní v rámci Visual Studia. Tyto funkce zahrnují nastavení breakpointů, včetně podmíněných breakpointů, vyhodnocování složitých výrazů v okně Watch, kontrolu a úpravu hodnot proměnných i argumentů a prohlížení složitých objektů i datových struktur. Tyto funkce jsou velmi užitečné pro vývojáře, kteří pracují na projektech v Unity.
- **Quick fixes and refactoring suggestions.** Pomocí sad Visual Studia, které mají hluboké porozumění projektům Unity, může uživatel využívat rychlé opravy a návrhy refaktoringu. Tyto funkce umožňují psát lepší kód s osvědčenými postupy a usnadňují efektivní práci na projektech v Unity.
- **CodeLens hints**. Funkce umožňuje identifikovat, kde se kód volá, a zobrazuje implicitní volání z Unity, což umožňuje snadno rozlišit kód od metod Unity. Tuto funkci lze využít pro každou funkci události Unity a při výběru nápovědy je možné zobrazit seznam implicitních volání. Dále lze při výběru konkrétního volání přejít přímo na objekt v editoru Unity.
- **Unity Project Explorer** je nástroj, který zobrazuje soubory projektu v hierarchické struktuře podobné oknu „Hierarchy“ v editoru Unity. Tento nástroj umožňuje vývojářům snadno přistupovat k souborům projektu a získat lepší přehled o jejich struktuře i organizaci. Díky tomu mohou vývojáři snadno spravovat soubory projektu a přidávat nové soubory i složky.

- **Unity documentation**. Při kontrole kódu se zobrazuje dokumentace k Unity přímo v nápovědách.
- **Automatically refresh Unity assets**. Visual Studio umožňuje automatickou aktualizaci zdrojů dat v Unity, což umožňuje vývojářům efektivnější práci a šetří čas, který by jinak museli trávit přepínáním mezi Unity a Visual Studiem. Při ukládání souborů se změny v kódu automaticky aktualizují v Unity, což umožňuje vývojářům sledovat změny v reálném čase a pracovat na projektech v Unity bez přerušení (Microsoft, 2021).

3.4.4 Jsfxr

Obrázek 11: Vzhled rozhraní webové aplikace Jsfxr (Fredricksen, 2011)



Jsfxr je JavaScript port desktopové aplikace Sfxr (Pettersson, 2012), která slouží k vytváření a generování zvukových efektů pro videohry. Jsfxr je navržen tak, aby byl snadno použitelný ve webových prohlížečích, a umožňuje uživatelům vytvářet zvuky pro své hry pomocí jednoduchého uživatelského rozhraní a algoritmů, které generují různé typy zvukových efektů, jako jsou například zvuky výstřelů, výbuchů nebo zvuky pohybu objektů (Fredricksen, 2011).

4 Vlastní práce – Prototyp hry Sandy

Tato kapitola se zaměřuje na tvorbu prototypu hry Sandy a ukazuje, jak může být na základě teoretické části této práce vytvořena 2D plošinová hra. Informace v této kapitole ohledně implementace jsou užitečné pro začínající vývojáře, kteří chtějí projít celou tuto práci a naučit se, jak vytvořit podobnou hru.

4.1 Game design document

Obsah GDD, ze kterého vychází následující kapitoly, je představen v kapitole 3.1.1.1.

4.1.1 Koncept

Sandy je plošinová 2D adventura s bočním pohledem. Je to klasická 2D plošinová hra, podobná známé hře Super Mario Bros. (Nintendo, 1985).

4.1.2 Příběh

Děj hry se odehrává na cestě hlavního hrdiny jménem Sandy do obchodu pro mléko. Sandy zjistí, že ztratil své peníze určené na nákup mléka. Hlavní hrdina se rozhodne najít peníze cestou do obchodu a koupit si mléko.

4.1.3 První minuta

Po spuštění hry a po značce „Unity Technologies“ se před hráčem objeví hlavní hrdina a text nad ním popisující děj. Text říká: „Ach ne, ztratil jsem peníze, které jsem potřeboval na nákup mléka. Možná najdu trošku peněz na cestě do obchodu.“ Tím jsou stanoveny cíle hry – najít peníze a dostat se do obchodu, aby bylo možné koupit mléko.

4.1.4 Postavy

Hlavním hrdinou je zrzavý kocour se zelenýma očima, který miluje dobrodružství a pití čerstvého mléka. Jmenuje se Sandy, umí házet kostmi z ryb a může se léčit rybími konzervami. Postava, kterou ovládá hráč.

Obrázek 12: Vzhled hlavní postavy (vlastní zpracování)

4.1.5 Hratelnost

Hratelnost znamená, jak budou hráč a hra vzájemně interagovat. Následující herní objekty a mechaniky vysvětlují, jak tato interakce bude probíhat.

- Nepřátelé
  - Magický červ – zmutovaný červ, který vylezl ze země a chová se nepřátelsky. Hází na hlavního hrdinu kusy hlíny a trávy.
  - Vzteklá krysa – krysa nakažená vzteklinou napadne hlavního hrdinu a pronásleduje ho, když se k ní přiblíží.
  - Orel – protivník, který útočí ze vzduchu, vidí v hlavním hrdinovi kořist a útočí a pronásleduje, pokud Sandy vstoupí do zóny viditelnosti.
  - Sliz – sám o sobě není nebezpečný, neútočí na hlavního hrdinu, ale pokud se ho dotkne, může se spálit kyselinou, která slimáka pokrývá. Pohybuje se v jedné oblasti.
  - Hroty – při setkání s nimi hlavní hrdina ztratí jedno červené srdce.
- Předměty
  - Mince – zlaté mince, které hlavní hrdina sbírá na nákup mléka.
  - Rybí kosti – kosti, které Sandy může házet na své nepřátele.
  - Konzervy – rybí konzervy, které hlavní hrdina jí, a obnovuje tím své zdraví.
  - Růžový kámen – kouzelný kámen, jehož zvednutím Sandy získá na krátkou dobu síly pro vysoké skoky.
- Herní mechaniky
  - Běh – hráč může pomocí kláves „Left Arrow“ a „Right Arrow“ nebo „A“ a „D“ na klávesnici přesouvat hlavního hrdinu vlevo nebo vpravo.
  - Skok – pomocí klávesy „mezerník“ může hráč skočit, směr skoku je kontrolován stejně jako při běhu.

- Interakce s interaktivními herními objekty (například tlačítka, šavle, výtahy atd.) probíhá stiskem klávesy „E“.
- Újma– při střetu s červenými trny nebo nepřáteli hlavní hrdina ztrácí život (odpovídá jednomu červenému srdci).
- Soubojový systém – jestliže hráč skočí shora na protivníka, pak protivník zemře, pokud je však hráč pod hlavou protivníka, hlavní hrdina získá zranění a jedno srdce mu bude odebráno.
- Smrt – když hlavní postava už nemá žádné červené srdce nebo když hráč spadne do lávy, hlavní postava zemře a úroveň začíná znovu, pokud hráč nedosáhl kontrolního bodu.
- Kontrolní bod – specifická část úrovně, když hráč dosáhne kontrolního bodu, jeho pokrok se uloží a při následujících smrtích úroveň začne od kontrolního bodu.
- Sběr předmětů – když se hráč dotkne sbíratelného předmětu (herního objektu, který lze sbírat, například peníze, hitpointy, náboje), předmět zmizí a následuje akce, která závisí na předmětu, například při sběru konzervy se obnovuje zdraví hlavního hrdiny.
- Schody – když se hlavní hrdina nachází u schodů, může po nich začít stoupat, jestliže hráč stiskne klávesy „Up Arrow“ a „Down Arrow“ nebo „W“ a „S“ na klávesnici.
- Plošiny – ve hře by měly být zastoupeny dva druhy plošin, pohyblivé a padající. Pohyblivé plošiny se mají periodicky pohybovat z jednoho bodu na druhý. Padající plošiny mají padat po určité době, když na ně hráč skočí.
- Střelba – pomocí tlačítek „Ctrl“ a „Left Mouse Button“ může hráč ve směru pohybu hlavní postavy hodit rybí kost, která může zabít nepřátele.

- Prohra
  - Prohra nastane, jestliže hráč nemá žádná červená srdce, při pádu do lávy nebo pádu z útesu. Pokud se to stane, hráč znovu „ožije“ na kontrolním bodu.

4.1.6 Dokončení úrovně

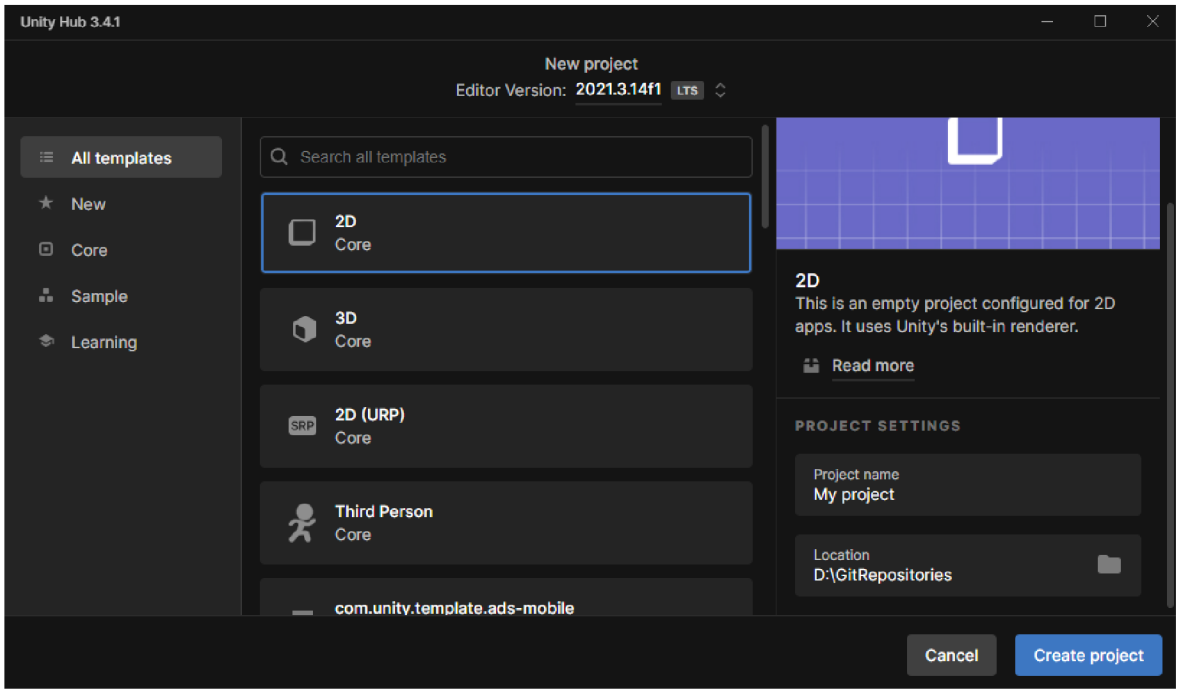
Cílem hry je najít správný počet mincí, dostat se do obchodu a koupit mléko. Když hráč splní tyto podmínky a dojde do obchodu, spustí se scéna s úspěšným koncem hry.

4.2 Implementace hry

4.2.1 Vytvoření projektu

Pro vytvoření projektu je nutné spustit Unity Hub (viz kapitolu 3.2.1 Unity Hub) a v záložce „Projects“ stisknout tlačítko „New project“. V objevujícím se okně budou k dispozici šablony pro 2D a 3D projekty, stejně jako možnost pojmenovat projekt a vybrat umístění na disku. Po výběru šablony a pojmenování projektu stisknutím tlačítka „Create project“ začne se projekt vytvářet a spustí se Unity editor.

Obrázek 13: Přehled šablon Unity Hubu (vlastní zpracování)



4.2.2 Background

Pozadí je jedním z důležitých prvků scény, protože určuje atmosféru úrovně. Pro nakreslení pozadí byla použita aplikace „Krita“ (viz kapitolu 3.4.1 Krita). Obrázek zadního pozadí byl nakreslen v pěti vrstvách: měsíc a noční obloha, vzdálené hory, mraky, blízké hory, vzdálené stromy, blízké stromy.

Pozadí je realizováno s efektem „Parallax scrolling“ (ve 2D hrách s pohledem z boku je to složení jednoho obrázku pomocí násobení vrstev, každá z nich se pohybuje vlastní rychlostí).

Pro realizace efektu je potřeba vytvořit herní objekt „Canvas“ a nastavit komponenty následujícím způsobem:

- Canvas
  - Render Mode → Screen Space – Camera
  - Render Camera → vybrat hlavní kameru (vytvořena automaticky při prvním spuštění projektu)
- Canvas Scaler
  - UI Scale Mode → Scale With Screen Size
  - Reference Resolution → napsat požadované rozlišení
  - Screen Match Mode → Expand

Potom je nutné vytvořit podřízený objekt „Image“ uvnitř „Canvasu“ a v komponentě „Image“ v nastavení „Source Image“ vybrat obrázek odpovídající první vrstvě pozadí.

Poté je třeba vytvořit ještě pět podřízených objektů typu „sprite“ a nastavit jejich komponenty následujícím způsobem:

- Sprite Renderer
  - Sprite → u každého z pěti „spritů“ postupně vybrat nutný obrázek odpovídající správnému pořadí vrstvy celého obrázku pozadí
  - Draw Mode → Tiled
    - Size → vybrat požadovanou šířku a výšku
    - Tile Mode → Continuous

Aby se vrstvy nepřekrývaly, je nutné v záložce „Layers“ kliknout na tlačítko „Edit layers“ a v otevřeném okně v sekci „Sorting Layers“ vytvořit novou vrstvu s názvem „Background“.

Potom v každém z pěti vytvořených „spritů“ nastavit komponenty následujícím způsobem:

- Sprite Renderer
  - Sorting Layers → vybrat vytvořenou vrstvu „Background“
  - Order in Layer → napsat požadované číslo odpovídající pořadí zobrazení vrstev

Posledním krokem je vytvoření skriptu, který bude připojen ke všem „spritům“. Poté, jakmile je vizuální část připravena, musí být vytvořen skript (viz kapitolu 3.2.2.6 Skriptování), který poskytuje funkce na pozadí.

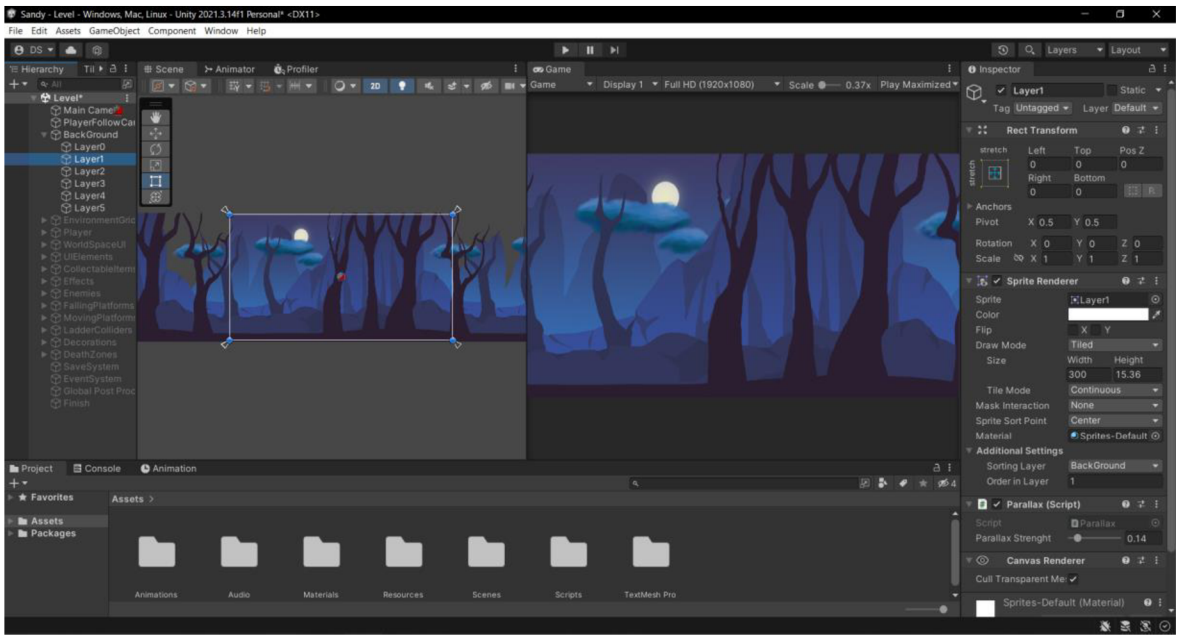
Následující skript popisuje implementaci efektu paralaxy:

```
private Transform _followingTarget;
[SerializeField, Range(0F, 1F)] private float _parallaxStrenght = 0.1F;
private Vector3 _targetPreviousPosition;

private void Start() {
    _followingTarget = Camera.main.transform;
    _targetPreviousPosition = _followingTarget.position;
}

private void LateUpdate() {
    Vector3 delta = _followingTarget.position - _targetPreviousPosition;
    delta.y = 0;
    _targetPreviousPosition = _followingTarget.position;
    transform.position += delta * _parallaxStrenght;
}
```

Ukázka kódu 2: Implementace efektu „Parallax scrolling“

Po napsání skriptu je nutné nastavit výše hodnot v nově vzniklé komponentě „Parallax Strength“. Pro vrstvy, které zobrazují další objekty, je nutné nastavit vysokou hodnotu, a naopak nižší hodnoty pro vrstvy, které zobrazují blízké objekty.

Obrázek 14: Background (vlastní zpracování)



4.2.3 Lokace

4.2.3.1 Foreground (Tilemap)

Komponenta „Tilemap“ v Unity je nástroj pro tvorbu 2D úrovní s využitím „Tile Assets“. Tato komponenta ukládá a zpracovává informace o jednotlivých „Tiles“, které jsou umístěny na ní. Dále přenáší tyto informace do souvisejících komponent, jako jsou „Tilemap Renderer“ a „Tilemap Collider 2D“. Při použití „Tilemap“ je také automaticky přidána komponenta

„Grid“, která slouží jako vodítko při rozmísťování „Tile“ na „Tilemap“. Tento nástroj usnadňuje tvorbu 2D úrovní v Unity a umožňuje vývojářům efektivněji pracovat s grafikou v jejich hrách (Unity Technologies, 2023j).

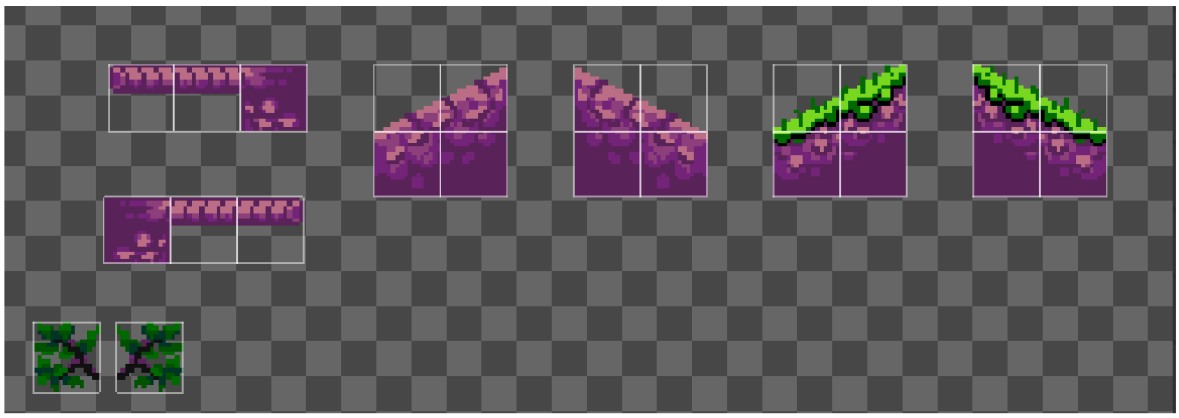
Pro vytvoření lokací je nutné mít sprity, které je možné nakreslit samostatně, nebo stáhnout z „Unity Asset Store“ (víz kapitolu 3.2.3 Unity Asset Store). V rámci tohoto projektu byl použit bezplatný balík assetů „Sunny Land“. Aby bylo možné ho importovat do projektu, je nutné ho přidat do účtu na stránce „Unity Asset Store“. Poté v editoru v záložce „Window“ kliknout na „Package Manager“, otevře se okno „Package Manager“ a v něm bude viditelný asset, který byl přidán do účtu. Nejprve je nutné ho stáhnout a poté pomocí tlačítka „Import“ importovat do projektu.

Dále je třeba na importovaný složený obrázek neboli „Tileset“ (skládající se z dalších obrázků, které se nazývají „Tile Assets“) kliknout a v otevřeném okně „Inspector“ nastavit komponenty následujícím způsobem:

- Texture Type → Sprite (2D and UI)
- Sprite Mode → Multiple
  - Pixels per Unit → 16

Potom kliknout na tlačítko „Sprite Editor“ a v otevřeném okně kliknout na složku „Slice“ a dále na tlačítko „Slice“. Jestliže nějaké obrysy nesouhlasí s obrázkem, lze to opravit ručně. Pokud je všechno správně, stačí stisknout tlačítko „Apply“ a zavřít okno.

Obrázek 15: Nakrájené části země v okně Sprite Editoru (vlastní zpracování)



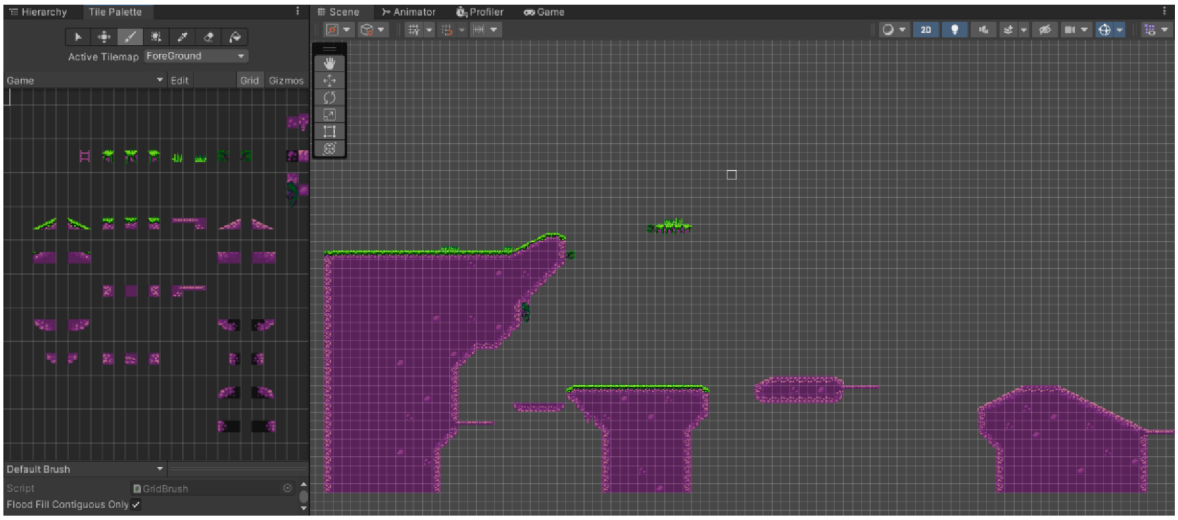
Aby bylo možné „nakreslit“ prostředí, po kterém bude hlavní postava chodit, je nutné otevřít okno „Tile Palette“. Je to okno, ve kterém lze vytvářet a ukládat „Tilesets“, upravovat jejich vlastnosti i nastavení. „Tile Palette“ umožňuje rychle a pohodlně vytvářet lokace a úrovně ve hrách, a to přesouváním a kombinováním různých „Tiles“ na mřížce, která bude vytvořena spolu s „Tilemap“. Po otevření okna je nutné vytvořit novou paletu cestou „Window“

→ „2D“ → „Tile Palette“ → „Create New Palette“, napsat název, kliknout na „Create“ a vybrat místo pro uložení palety. Poté je nutné přetáhnout oříznutý „Tileset“ do okna „Tile Palette“.

Potom je nutné vytvořit „Tilemap“ v okně „Hierarchy“ následujícím způsobem: kliknout na „Plus“ → „2D Object“ → „Tilemap“ → „Rectangular“. Je vytvořen herní objekt „Grid“ a automaticky přidán herní objekt „Tilemap“. Poté je v okně „Tile Palette“ potřeba vybrat vytvořený „Tilemap“ a už lze začít „kreslit“ zemi.

Následně stejným způsobem byly přidány další „Tilemaps“ potřebné pro rozdělení vrstev pozadí. Aby se vrstvy nepřekrývaly, používá se stejný princip s vytvořením vrstev a určením pořadí, jak bylo popsáno v kapitole 4.2.2 Background, ale odehrává se to v komponentě „Tilemap Renderer“.

Obrázek 16: Ukázka popředí (vlastní zpracování)



Aby hlavní hrdina i ostatní postavy mohli interagovat s povrchem, je nutné přidat fyziku jeho chování. Pro tento účel je do vytvořených herních objektů „Tilemaps“ nutné přidat komponentu „Tilemap Collider 2D“ a nastavit ji následujícím způsobem:

- Used By Composite → true

  Potom je nutné přidat komponentu „Rigidbody2D“ s následujícím nastavením:
- Body Type → Kinematic

  Poté je potřeba přidat komponent „Composite Collider 2D“ s následujícím nastavením:
- Used By Effector → true

  Nakonec se přidá komponenta „Platform Effector 2D“ s následujícím nastavením:
- Use One Way → false

Nyní je k dispozici vše potřebné pro vytvoření lokace, po které bude hlavní hrdina chodit.

4.2.3.2 Animace lávy

Technika animace, která byla v této práci použita, se nazývá „**snímek po snímku“**. Zahrnuje kreslení každého snímku animace samostatně, ale jako série, nikoli jednotlivě. Jednotlivé snímky se pak přehrávají v rychlém sledu, aby se vytvořila iluze pohybu. Tato technika se často používá k vytváření složitých animací s velkým počtem pohyblivých částí (Adobe, 2021).

„Sprity“ lávy byly nakresleny v aplikaci Krita, celkem bylo nakresleno pět „spritů“ pro animaci **snímek po snímku** a jeden „sprite“ pro vyplnění prostoru.

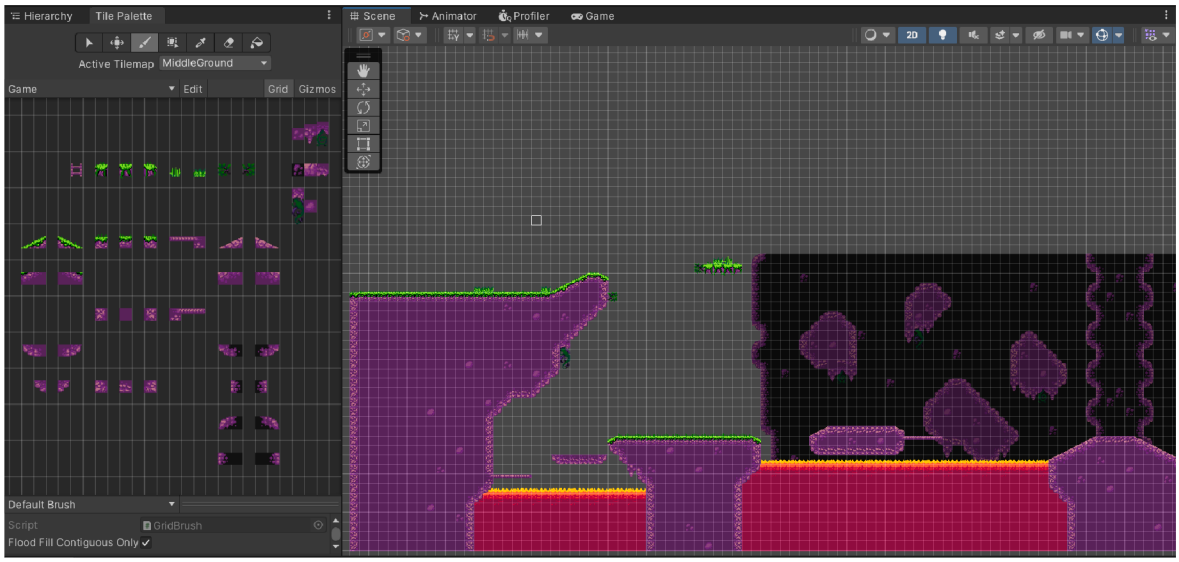
Pro vytvoření animace lávy je nutné v okně „Project“ kliknout na „Plus“ a v otevřeném kontextovém menu projít cestou „Create“ → „2D“ → „Tiles“ → „Animated Tiles“. Bude vytvořen soubor „Animated Tile“, ten je potřeba nastavit následujícím způsobem:

- Number of Animated → v tomto případě 5, protože celá animace má pět snímků.

Poté se objeví pět nastavení, ve kterých je třeba vybrat „sprity“ v pořadí, v němž bude animace sestavena. Také v parametrech „Minimum Speed“ a „Maximum Speed“ lze nastavit rychlost animace. Potom v okně „Tile Palette“ je nutné vytvořit novou paletu a také v okně Hierarchy uvnitř herního objektu „Grid“, který byl vytvořen v kapitole 4.2.3.1, vytvořit nový „Tilemap“ pro lávu.

Poté je třeba soubor „Animated Tile“ přesunout do okna „Tile Palette“ a v tomto okně vybrat vytvořený „Tilemap“ v nastavení „Active Tilemap“. Nyní je možné „nakreslit“ lávu.

Obrázek 17: Všechny prvky popředí (vlastní zpracování)



4.2.3.3 Plošiny

Jak již bylo popsáno v kapitole 4.1.5.3, ve hře by měly být zastoupeny dva typy plošin. „Sprity“ pro plošiny byly nakresleny pomocí Pixilartu (viz kapitolu 3.4.2 Pixilart).

Pro realizaci padající plošiny je nutné vytvořit herní objekt typu „sprite“. V komponentě „Sprite Renderer“ v nastavení „Sprite“ je nutné vybrat nakreslený obrázek odpovídající plošině. Potom je potřeba přidat komponentu „Box Collider 2D“ s následujícími nastaveními:

- Offset → 0(X), 0(Y)
- Size → nastavit podle velikosti „sprite“ plošiny na scéně
- Used By Effector → true

Poté se přidá komponenta „Platform Effector 2D“ s následujícími nastaveními:

- Collider Mask → Player. Je potřeba vytvořit tuto vrstvu, stejně jako to bylo popsáno v kapitole 4.2.2 Background, tentokrát to však není „Sorting Layer“, ale „Layers“.
- Use One Way → false

Pro realizaci mechaniky je nutné přidat komponentu „Rigidbody 2D“ následujícím nastavením:

- Body Type → Kinematic

Poté je nutné vytvořit skript, který bude řídit podmínky pro začátek padání plošiny. Jeho funkce je poměrně jednoduchá. Při kontaktu hráče s plošinou se volá metoda „Fall()“ s určitým zpožděním, která vypne parametr „isKinematic“ u komponenty „Rigidbody 2D“ a nutí plošinu se pohybovat, tím pádem plošina padá dolů po zpoždění a následně se zničí, aby nezatěžovala scénu.

```
    [SerializeField] private float _fallDelay;
    private const float _destroyingTime = 2.0f;
    private Rigidbody2D _rigidBody2D;

    private void Start() => _rigidBody2D = GetComponent<Rigidbody2D>();

    private void OnCollisionEnter2D(Collision2D collision) {
        if (collision.gameObject.TryGetComponent(out Player player)) {
            Invoke(nameof(Fall), _fallDelay);
            Destroy(gameObject, _destroyingTime);
        }
    }

    private void Fall() => _rigidBody2D.isKinematic = false;
```

Ukázka kódu 3: Implementace padající plošiny

Stejným způsobem jsou realizovány pohyblivé plošiny, s výjimkou u komponenty „RigidBody 2D“, tentokrát není nutné ho přidávat. Je potřeba vytvořit další vrstvu, stejně jako bylo uvedeno v kapitole 4.2.2 Background, ale tentokrát to není „Sorting Layer“, nýbrž „Layers“. Skript, který implementuje mechanismus pohybu, se také liší. Pro pohyb plošin po ose X a Y se používají čtyři body podle kontroly bodů změny směru a dvě rychlosti. Když plošina dosáhne nastaveného bodu v herním prostředí, mění se směr jejího pohybu. Zbývá jen rozmístit plošiny po scéně a přiřadit potřebné body.

```
    [SerializeField] private float _rightPoint;
    [SerializeField] private float _leftPoint;
    [SerializeField] private float _upPoint;
    [SerializeField] private float _downPoint;
    [SerializeField] public float SpeedX;
    [SerializeField] private float _speedY;
    [SerializeField] private bool _ableToChangeDirection;
    private bool _changeMoveDirection = true;

    private void FixedUpdate() {
        if (transform.position.y > _upPoint && !_ableToChangeDirection)
_changeMoveDirection = false;
        else if (transform.position.y < _downPoint && !_ableToChangeDirection)
_changeMoveDirection = true;

        if (transform.position.x > _rightPoint && _ableToChangeDirection)
_changeMoveDirection = false;
        else if (transform.position.x < _leftPoint && _ableToChangeDirection)
_changeMoveDirection = true;

        if (_changeMoveDirection) transform.position = new
Vector2(transform.position.x + SpeedX * Time.fixedDeltaTime, transform.position.y
+ _speedY * Time.fixedDeltaTime);
        else transform.position = new Vector2(transform.position.x - SpeedX *
Time.fixedDeltaTime, transform.position.y - _speedY * Time.fixedDeltaTime);
}
```

Ukázka kódu 4: Implementace pohybující se plošiny

4.2.3.4 Schody

Realizace schodů vyžaduje vytvoření „Tilemap“, což je nutné pro oddělení vrstev a možnost umístit schody na scéně. Poté by měl být vytvořen prázdný herní objekt a k němu přidána komponenta „BoxCollider2D“ s následujícími nastaveními:

- Offset → 0(X), 0(Y)
- Size → přizpůsobit rozměry pod sprite schodů na scéně, ale ponechat prostor dole a nahoře
- Is Trigger → true

Pro označení začátku a konce schodů jsou navíc potřeba dva další prázdné herní objekty s přidanou komponentou „BoxCollider2D“ a nastavenými stejným způsobem, ale umístěny na začátek i konec, kde byl ponechán prostor.

Následující metody jsou zásadní pro realizace mechaniky schodů:

```
    private void OnTriggerEnter2D(Collider2D collision) {
        if (collision.TryGetComponent(out Player player)) {
            switch (_part) {
                case LadderPart.complete:
                    player.CanClimb = true;
                    player.Ladder = this;
                    break;
                case LadderPart.bottom:
                    player.BottomLadder = true;
                    break;
                case LadderPart.top:
                    player.TopLadder = true;
                    break;
                default:
                    break;
            }
        }
    }
    private void OnTriggerExit2D(Collider2D collision) {
        if (collision.TryGetComponent(out Player player)) {
            switch (_part) {
                case LadderPart.complete:
                    player.CanClimb = false;
                    break;
                case LadderPart.bottom:
                    player.BottomLadder = false;
                    break;
                case LadderPart.top:
                    player.TopLadder = false;
                    break;
                default:
                    break;
            }
        }
    }
```

Ukázka kódu 5 Klíčové metody pro implementaci schodů

4.2.3.5 Dekorace

Dekorace se reprezentuje jako běžný herní objekt s přidanou komponentou „Sprite Renderer“ a vybraným nutným obrázkem. Hráč nemá žádnou interakci s dekoracemi, slouží pouze k vyplnění prostoru na scéně.

Obrázek 18: Dekorace na scéně (vlastní zpracování)

4.2.4 Hlavní postava

4.2.4.1 Sprite

Vzhled a sprity pro animace jednotlivých snímků pro hlavního hrdinu byly, stejně jako prostředí, importovány z bezplatného Asset Packu („Sunny Land“) a mírně upraveny.

4.2.4.2 Kamera

Hlavní kamera byla vytvořena při první inicializaci projektu. Parametr „Projekce“ musí být nastaven na „Orthographic“.

Pro vytvoření kamery sledující postavu je potřeba vytvořit prázdný herní objekt. Tento krok je proveden následujícím způsobem: v okně „Hierarchy“ je nutné kliknout na „Plus“ → „Cinemachine“ → „Virtuální kamera“. Komponenta „Cinemachine Brain“ se automaticky přidá do hlavní kamery. Následně je nutné provést nastavení virtuální kamery následujícím způsobem:

- Follow → Player (Transform)
- Look At → Player (Transform)
- Body → Framing Transposer. Rovněž je nutné upravit akční zóny kamery.
- Aim → Do nothing

4.2.4.3 Pohyb

Běh je realizován pomocí komponent „RigidBody 2D“ a „Box Collider 2D“. „RigidBody 2D“ je připojen ke hlavní postavě a má následující nastavení:

- Collision Detection → Continuous
- Freeze Rotation → false (X), false (Y), true (Z)

Nastavení „Box Collider 2D“:

- Material → Je nutné vytvořit materiál v okně „Project“ následujícím způsobem: „Create“ → „2D“ → „Physics Material“ a nastavit u komponenty „Friction“ požadovanou hodnotu. Potom ten materiál vybrat jako materiál.
- Offset, Size → přizpůsobit velikosti „spritu“

Aby se hráč mohl pohybovat, musí stisknout horizontální vstupní tlačítka („Left Arrow“, „Right Arrow“, „A“, „D“). Když hráč stiskne jedno z těchto tlačítek, postava se začne pohybovat doleva nebo doprava. Následují skripty popisují proces běhu ve hře:

```
    private void Update() {
        _horizontalMove = Input.GetAxisRaw("Horizontal") * _moveSpeed;
      …
}
```

Ukázka kódu 6: Přiřazování rychlosti a směru pohybu do proměnné

V metodě „Update()“ se přiřazuje hodnota rychlosti a směru do proměnné, poté se tato hodnota předává do následující metody:

```
    private void Run(float moveSpeed) {
        Vector3 targetVelocity = new Vector2(moveSpeed * 10f,
_rigidBody2D.velocity.y);
        _rigidBody2D.velocity = Vector3.SmoothDamp(_rigidBody2D.velocity,
targetVelocity, ref _velocity, 0f);
        if (_horizontalMove > 0f && !flipX) Facing();
        else if (_horizontalMove < 0f && flipX) Facing();
    }
```

Ukázka kódu 7: Implementace běhu

Při volání metody „Run()“ se jako parametr „moveSpeed“ použije hodnota „_horizontalMove * Time.fixedDeltaTime“. „Time.fixedDeltaTime“ je interval v sekundách, ve kterém se provádí aktualizace fyziky a dalších funkcí s pevnou snímkovou frekvencí, jako je „FixedUpdate“ funkce „MonoBehaviour“ (Unity Technologies, 2023k). Používá se k tomu, aby hra byla nezávislá na snímkové frekvenci obrazovky.

4.2.4.4 Skok

V realizaci skoku se aplikuje následující postup. Pokud je protagonista v kontaktu s povrchem („_isGrounded = true“) a hráč stiskne tlačítko „Space“, hlavní postava provede skok. Tyto kroky jsou řízeny skriptem, který stanovuje podmínky pro realizaci skoku.

```
private void Update() {
    …
    if (Input.GetButtonDown("Jump") && _isGrounded) Jump();
    …
}
private void Jump() {
    _rigidBody2D.AddForce(Vector2.up * _jumpForce);
    _animator.SetBool("Jumping", true);
    _jumpSound.Play();
    DustPlay();
}
```

Ukázka kódu 8: Implementace skoku

4.2.4.5 Střelba

Pro realizaci střelby je třeba nejprve realizovat náboj. Sprite kulky byl nakreslen v aplikaci Pixilart. Byl vytvořen prázdný herní objekt a na něj byly umístěny komponenty „Sprite Renderer“, „Box Collider2D“, „Rigidbody2D“ s následujícími nastaveními:

- Sprite Renderer
  - Sprite → vybrat potřebné zobrazení
- Box Collider2D
  - Is Trigger → true
  - Offset → 0(X), 0(Y)
  - Size → přizpůsobit rozměry pod sprite náboje na scéně
- Rigidbody2D
  - Body Type → Dynamic
  - Collision Detection → Continuous

Náboj je připraven, nyní je potřeba vytvořit jeho „Prefab“, k čemuž je třeba přetáhnout vytvořený herní objekt do okna „Project“. Dále je potřeba pro zadání rychlosti, směru a pro to, aby kulka rozeznala svého rodiče, napsat logiku jejího chování.

Pro implementaci střely jsou klíčové následující metody:

```
    private void Update() {
        transform.position = Vector3.MoveTowards(transform.position,
            transform.position + _startDirection,
            _speed * Time.deltaTime);
        transform.Rotate(0, 0, _rotationSpeed);
    }
    private void OnTriggerEnter2D(Collider2D collider) {
        if (collider.TryGetComponent(out EnemiesController unit) &&
unit.gameObject != _parentGameObject) {
            Destroy(gameObject);
        }
        if (collider.TryGetComponent(out Player player) && player.gameObject !=
_parentGameObject) {
            Destroy(gameObject);
        }
        Collider2D[] collider2D = Physics2D.OverlapCircleAll(transform.position,
0.1f, _groundLayer);
        for (int i = 0; i < collider2D.Length; i++) {
            if (collider2D[i].gameObject != gameObject) {
                Destroy(gameObject);
            }
        }
    }
```

Ukázka kódu 9: Hlavní metody pro implementaci náboje

Nyní je možné implementovat střelbu jednoduše tak, že ve třídě „Player“ je nutné napsat metodu zodpovídající za střelbu:

```
    private void Update() {
        …
        if (Input.GetButtonDown("Fire1") && Ammoes > 0) Shoot();
        …
    }
    private void Shoot() {
        Ammoes--;
        _shootSound.Play();
        Bullet newBullet = Instantiate(_bullet, transform.position,
_bullet.transform.rotation) as Bullet;
        newBullet.Parent = gameObject;
        newBullet.Direction = newBullet.transform.right * _scale.x;
    }
```

Ukázka kódu 10: Implementace střelby

4.2.4.6 Zvuky

Pro přehrávání různých zvuků (např. skákání, šlapání, zvedání předmětů atd.) je k hlavní postavě připojena komponenta „Audio Source“. Poté je v nastavení „AudioClip“ nutné vybrat požadovaný zvuk. Některé zvuky byly vygenerovány pomocí online generátoru zvuků „Jsfxr“ (viz kapitolu 3.4.4 Jsfxr) a některé byly převzaty z bezplatných balíčků z „Asset Store“.

Příklad skriptu pro spouštění zvuku při skoku je vidět v metodě „Jump ()“ v kapitole 4.2.4.4. Komponenta „AudioSource“ je předána poli „_jumpSound“. Pomocí metody „.Play ()“ lze v případě potřeby přehrávat zvuky, například při skoku se zvuk přehraje pokaždé, když hráč skočí.

```
    [SerializeField] private AudioSource _jumpSound;
```

Ukázka kódu 11: Pole nutné pro přehrávání zvuku skoku

4.2.4.7 Efekty

Efekt prachu

Pohybuje-li se hráč v jednom směru a následně změní směr na opačný, na zemi pod hlavní postavou se objeví prachové částice. K tomu je nutné vytvořit podřízený objekt typu „Particle System“ uvnitř hlavní postavy a nastavit následující parametry v okně „Inspector“:

- Duration → 0.1
- Looping → false
- Start Lifetime → 0,5
- Start Size → 0,4
- Simulation Space → World
- Play On Awake → false
- Emission → true
- Rate over Time → 100
    - Shape → true
    - Shape → Box
    - Scale → x = 0,5; y = 0; z = 1
- Velocity over Lifetime
    - Linear → x = -0,3; y = 0,3; z = 0
    - Space → World
- Color over Lifetime → Gradient (od bílého k transparentnímu)
- Size over Lifetime → Curve (křivka od 1 do 0)

Tato funkce je implementována pomocí třídy „ParticleSystem“ a implementace skriptu je podobná jako u přehrávání zvuku.

```
    [SerializeField] private ParticleSystem _dustParticle;

    private void DustPlay() => _dustParticle.Play();

    private void Facing() {
        flipX = !flipX;
        _scale = transform.localScale;
        _scale.x *= -1f;
        transform.localScale = _scale;
        DustPlay();
    }
```

Ukázka kódu 12: Implementace efektu prachu

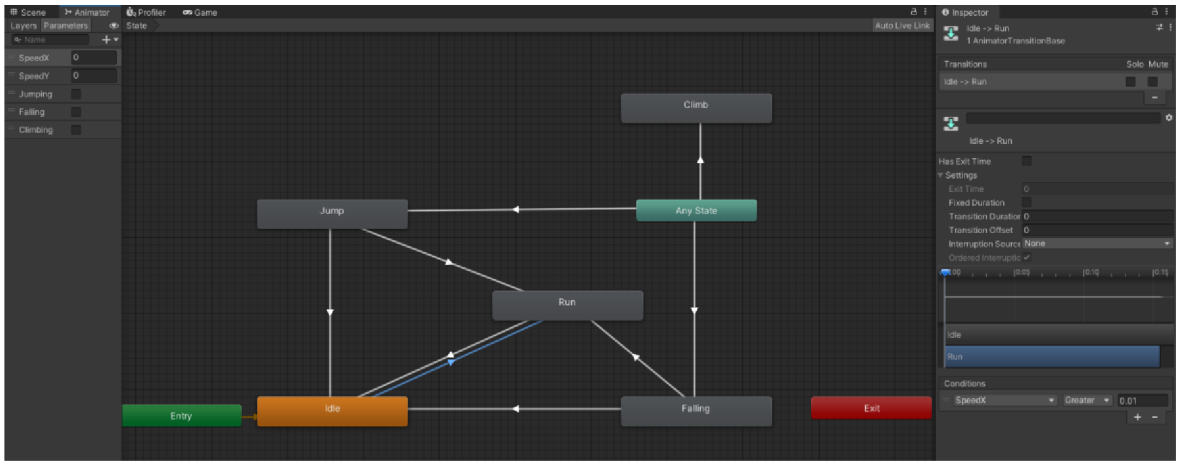
Efekt poškození

Tento efekt je vytvořen stejným způsobem jako efekt prachu, pouze se přehraje, utrpí-li hlavní postava škodu. Kromě toho jsou také nastaveny některé jiné parametry v komponentě „Particle System“.

4.2.4.8 Animace

Aby hlavní postava běžela, musí nejprve být vytvořen ovládač klipů pro tento herní objekt. Dále se v okně „**Animation“** vytvoří nová animace. V tomto okně je stopa, do které se přidávají klíčové snímky. Klíčový snímek je stav, ve kterém bude herní objekt v době zobrazení animace. Když jsou všechny klíčové snímky umístěny na místech, je možné kliknout na tlačítko přehrávání, pro kontrolu celé animace, a v případě potřeby provést opravy.

Obrázek 19: Stavy animací v okně „Animator“ (vlastní zpracování)



Přiřazování animací k hernímu objektu ve scéně probíhá pomocí komponenty „**Animator“**. Tato komponenta vyžaduje odkaz na řídicí jednotku aplikace „Animator“, která definuje, které animační klipy se mají použít, a určuje, kdy a jak se mezi nimi mají prolnout a přecházet (Unity Technologies, 2023l). Další kontrola podmínek pro spuštěni animace je ve skriptech.

4.2.5 Nepřátelé

Nepřátelé jsou vytvářeni podle jednoho z principů objektově orientovaného programování (dále OOP) – dědičnosti. Právě dědičnost je jednou z klíčových vlastností OOP. Umožňuje definovat podřízenou třídu, která využívá, rozšiřuje nebo modifikuje chování nadřazené třídy. Třída, která je zdrojem pro dědění, se nazývá základní třída. Třída, jež zděděné členy používá, se nazývá odvozená třída. C# a .NET podporují pouze jedno dědění, tj. třída může být odvozena pouze od jedné třídy. Navzdory tomu je dědičnost přenosná, což umožňuje vytvořit hierarchii dědičnosti pro sadu typů. Typ D tak může dědit z typu C, který dědí z typu B, a ten zase dědí od základního typu A. V důsledku tranzitivní povahy dědičnosti jsou členové typu A k dispozici i pro typ D (Microsoft, 2023a).

Základní třída – „EnemiesController“

Třída obsahuje pole a metody, které mohou být duplicitní, pokud není použita dědičnost. Touto metodou se například implementuje smrt, když hráč skočí na nepřítele:

```
    public virtual void JumpedOn(){
        _Animator.SetTrigger("Death");
        _RigidBody2D.constraints = RigidbodyConstraints2D.FreezeAll;
        ExplosionSound.Play();
        _Collider2D.enabled = false;
        _JumpedOn = true;
    }
```

Ukázka kódu 13: Implementace smrti nepřátel

V komponentě „Animator“ se aktivuje spouštěč, který byl pojmenován „Death“, a je přehrávána animace smrti, která byla dříve vytvořena v okně „Animation“. Na koncovém klíčovém snímku této animace byla pomocí tlačítka „Add event“ v okně „Animation“ přidána událost a byl zvolen veřejný metod „Destroying()“.

```
    protected virtual void Destroying() => Destroy(this.gameObject);
```

Ukázka kódu 14: Metoda pro realizace zničení nepřátel

Orel

Sprite byl importován z bezplatného balíčku assetů „Sunny Land“ a byl mírně vylepšen. Animace letu byla provedena stejným způsobem jako animace hlavní postavy. Řízení chování spočívá v tom, že orel hlídá specifický bod, a když se hráč přibližuje, začne ho pronásledovat. Poté, co se hráč dostane z určité vzdálenosti od orla, pronásledování se zastaví a orel se vrátí na své hlídací místo.

Obrázek 20: Vzhled orla v Unity (vlastní zpracování)

Jedna z klíčových metod tohoto skriptu, která se zabývá vším, co bylo popsáno dříve.

```
    private void StateControl() {
        if (Vector2.Distance(transform.position, _patrollingPoint.position) <
_positionOfPatrolling && _angryState == false) {
            _neutralState = true;
        }
        if (Vector2.Distance(transform.position, Target.position) <
_chaseStopDistance) {
            _angryState = true;
            _neutralState = false;
            _returnState = false;
        }
        if (Vector2.Distance(transform.position, Target.position) >
_chaseStopDistance) {
            _returnState = true;
            _angryState = false;
        }

    }
```

Ukázka kódu 15: Klíčová metoda nutná pro kontrolu stavů chování orla

Vzteklá krysa

Krysa má stejnou implementaci chování jako orel, avšak se pohybuje po povrchu země, zatímco orel létá. Má stejný model chování, jaký je popsán ve skriptu.

Obrázek 21: Vzhled krysy v Unity (vlastní zpracování)

Sliz

Sprite byl nakreslen v programu „Pixilart“, stejně jako dvě animace – skok a bez děje. Podle logiky popsané v dokumentu návrhu hry byl vytvořen skript, který jeho chování implementuje. Jsou vytvořeny dva body, které vymezují levou a pravou hranici konce pohybu sliza. Do animace bez děje byla přidána událost, která provádí metodu „Move()“.

Obrázek 22: Vzhled sliza v Unity (vlastní zpracování)

```
    private void Move() {
        if (_facingRight) {
            if (transform.position.x > _leftPoint) {
                if (transform.localScale.x != 1) transform.localScale = Scale(1f,
1f);
                if (_Collider2D.IsTouchingLayers(_groundLayer)) Jump(-
_jumpLength);
            } else _facingRight = false;
        } else {
            if (transform.position.x < _rightPoint) {
                if (transform.localScale.x != -1) transform.localScale = Scale(-
1f, 1f);
                if (_Collider2D.IsTouchingLayers(_groundLayer))
Jump(_jumpLength);
            } else _facingRight = true;
        }
    }
```

Ukázka kódu 16: Klíčová metoda pohybu sliza

Magický červ

Sprite a obrázky pro animaci byly nakresleny v programu „Krita“. Červ střílí částmi země s trávou směrem k hlavní postavě, zatímco sám zůstává na stejném místě, jen se otáčí.

Obrázek 23: Vzhled magického červa v Unity (vlastní zpracování)

Klíčovou metodou je „Shoot()“ a v metodě „Start()“ se provede metoda „InvokeRepeating(nameof(Shoot), _shootRate, _shootRate“ pro opakování výstřelů.

```
    private void Shoot() {
        _shootSound.Play();
        Vector3 position = transform.position;
        position.y += 0.2F;
        Bullet       newBullet       =       Instantiate(_bullet,       position,
_bullet.transform.rotation) as Bullet;
        newBullet.Parent = gameObject;
        newBullet.Direction = newBullet.transform.right * (_SpriteRenderer.flipX ?
1.0F : -1.0F);
        newBullet.SpriteRenderer.sprite = _bulletSprite;
        newBullet.Color = _bulletColor;
    }
```

Ukázka kódu 17: Implementace střelby červa

Hroty

Sprite byl importován z bezplatného balíčku assetů „Sunny Land“ a byl mírně upraven. Mechanika je podobná, stejně jako při kontaktu s nepřáteli ztrácí hrdina jedno červené srdce. Implementace v kódu se neliší od implementace kontaktu s nepřáteli.

4.2.6 Sběratelské předměty

Ve hře se vyskytují čtyři druhy sběratelských předmětů – konzervy (doplnění zdraví), rybí kosti („náboje“), zlaté mince (peníze) a magický kámen (dočasné posílení skoku).

Konzervy

Nejprve byl vytvořen nový herní objekt a poté k němu byly přidány komponenty s následujícími nastaveními:

- Box Collider 2D
  - Is Trigger → true
  - Offset → x = 0; y = 0
  - Size → je nastavena vhodná hodnota pro osy X a Y
- Sprite Renderer
  - Sprite → je vybrán potřebný sprite
  - Sorting Layer → dříve vytvořený s názvem „Entity“
  - Order in Layer → požadované pořadí zobrazení na scéně
- Audio Source
  - AudioClip → je vybrán příslušný zvuk

Dále se vytváří značka (tag), v okně „Inspector“ je třeba kliknout na rozbalovací seznam Tag a v tomto seznamu kliknout na „Add Tag“, v nově otevřeném okně kliknout na plus a napsat název značky, v tomto případě „HitPoint“, a kliknout na „Save“. Nyní je možné s pomocí této značky v kódu označit požadované objekty a vytvořit logiku chování. Například když se hráč dotkne předmětu označeného značkou „HitPoint“, tedy konzervy, a hlavní hrdina se netěší plnému zdraví, provede se následující metoda ve třídě „CollectableItems“, která zvýší počet zdraví postavy o jednotku.

```
    private Player _player;

    public void PickUpEvent() {
        _audioSource.Play();
        _animator.SetTrigger("PickUp");
        if (gameObject.CompareTag("Coin")) _player.Coins++;
        if (gameObject.CompareTag("Ammo")) _player.Ammoes++;
        if (gameObject.CompareTag("HitPoint")) _player.HitPoint++;
        if (gameObject.CompareTag("JumpUp")) _player.PowerUpEvent();
    }
```

Ukázka kódu 18: Implementace výběru předmětů

Následující kód je zapsán ve třídě „Player“ v metodě „OnTriggerEnter()“ a volá výše popsanou metodu.

```
        if (collider.CompareTag("HitPoint") && HP == 5) return;
        else  if  (collider.TryGetComponent(out  CollectableItems  collectable))
collectable.PickUpEvent();
```

Ukázka kódu 19: Kontrola podmínek pro volání metody „PickUpEvent()“

Všechny ostatní předměty jsou vyrobeny podobným způsobem, liší se pouze názvy značek. Tyto značky lze použít k rozlišení předmětů ve skriptu, aby bylo možné po jejich zvednutí provést potřebné akce.

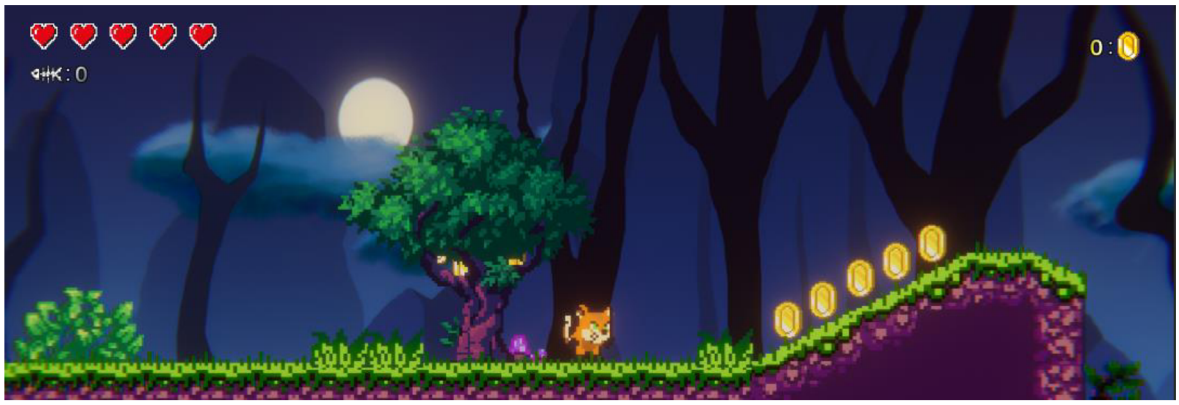
4.2.7 UI

Elementy uživatelského rozhraní se dělí na dva typy: na ty, které uživatel vidí na obrazovce vždy, a na ty, jež se objevují v herním prostoru v průběhu úrovně.

Rozhraní

Na rozhraní se zobrazují srdíčka (udávající počet jednotek zdraví hlavního hrdiny), počet nábojů a počet zlatých mincí.

Obrázek 24: Uživatelské rozhraní (vlastní zpracování)



Pro implementaci rozhraní byl vytvořen „Canvas“ a v jeho komponentě „Canvas“ bylo pro parametr „Render Mode“ vybráno „Screen Space – Overlay“.

Srdíčka

Pro vytvoření srdíček v dříve vytvořeném „Canvasu“ byl vytvořen objekt typu „Image“ (s názvem „HpBar“), který byl nastaven následujícím způsobem:

- Image
  - Color → alfa kanál je nastaven na nulu pro dosažení průhlednosti.

Poté se k tomuto objektu přidají komponenty „Horizontal Layout Group“ a „Content size Fitter“ s následujícími nastaveními:

- Horizontal Layout Group
  - Padding
    - Left → 5
    - Right → 5
    - Top → 10

        - Bottom → 5
        - Spacing → 15
    - Child Alignment → Upper Left
- Content Size Fitter
    - Horizontal Fit → Min Size
    - Vertical Fit → Min Size

Následně bylo uvnitř „HpBaru“ vytvořeno pět dalších objektů „Image“. V parametru „Image“ byl vybrán obrázek srdce a s těmito nastaveními byla přidána komponenta „Layout Element“:

- Min Width → 50
- Mac Width → 50

Skript zabývající se kontrolou zobrazení správného počtu srdíček na obrazovce:

```
private Transform[] _hearts = new Transform[5];
private Player _player;

public void Refresh() {
    for (int i = 0; i < _hearts.Length; i++)
    {
        if (i < _player.HP) _hearts[i].gameObject.SetActive(true);
        else _hearts[i].gameObject.SetActive(false);
    }
}
```

Ukázka kódu 20: Metoda určená pro kontrolu správnosti zobrazení zdraví

Náboje a zlaté mince

Náboje a zlaté mince se vytvářejí stejným způsobem. Skládají se ze tří částí: obrázku, dvojtečky a množství. Číslo zobrazené na obrazovce se řídí ve třídě „Player“ v metodě „Update()“ pomocí následujících řádků:

```
[SerializeField] private TextMeshProUGUI _ammoesText;
[SerializeField] private TextMeshProUGUI _coinsText;

public int Ammoes = 2;
public int Coins = 0;
private void Update() {
…
     _ammoesText.text = Ammoes.ToString();
     _coinsText.text = Coins.ToString();
…
}
```

Ukázka kódu 21: Implementace zobrazení počtu „nábojů“ a zlatých mincí

Elementy uživatelského rozhraní v herním prostoru

Po spuštění hry hráče přivítá text, který ho uvede do příběhu. Tento text se pohybuje spolu s hlavní postavou. Byl implementován následujícím způsobem: nejprve byl vytvořen další „Canvas“, v něm pak objekt typu „TextMeshPro“ a byla provedena tato nastavení:

- TextMeshPro – Text (UI)
  - Text Input → pole, kam se zadává text, který má být zobrazen
  - Font Style → Bold
  - Font Size → 17
  - Alignment → Justify + Middle
  - Outline → true
    - Color → #000000 (černá)
    - Thickness → 0,26

Aby text nesledoval hlavního hrdinu po celou dobu hry, je v metodě „Start()“ třídy „Player“ tento řádek: „`StartCoroutine(ResetIntroductionText());`“. Ten pak volá následující metodu:

```
    private IEnumerator ResetIntroductionText() {
        yield return new WaitForSeconds(8f);
        _introduction.enabled = false;
    }
```

Ukázka kódu 22: Implementace metody určené pro zmizení textu nad hlavní postavou po nějakém čase

Aby text následoval hráče, provede se v metodě „Update ()“ ve stejné třídě následující metoda:

```
    private void TextFollow() {
        Vector2 textPos = new Vector2(transform.position.x, transform.position.y +
2f);
        _introduction.transform.position = textPos;
    }
```

Ukázka kódu 23: Metoda, která umožňuje, aby text sledoval hlavní postavu

V dalším průběhu hry se objeví místa, kde je třeba hráči říci, co má dělat, například při zvednutí předmětu zvaného magický kámen, jelikož hráč neví, co tento předmět dělá (dočasně zvyšuje výšku skoku), se na obrazovce objeví zpráva „Jump!“ s podobným nastavením, jaké bylo popsáno výše.

4.2.8 Systém uložení

Ve fázi návrhu cílů této práce se předpokládalo, že pro ukládání herních dat bude potřeba malá databáze, ale v průběhu vývoje bylo rozhodnuto se této myšlenky zříci, protože objem ukládaných informací je příliš malý a vejde se do několika řádků textu v „JSON“ formátu.

```
{
"Position": {
"x":107.26359558105469,
"y":0.9669119119644165
},
"HP":4,
"Ammoes":2
}
```

Ukázka kódu 24: Příklad ukládaných dat

Ukládání bylo řešeno tak, že když hráč dosáhne kontrolního bodu, uloží se jeho pozice, zdraví a počet nábojů.

Při ukládání se zkontroluje, zda soubor, do kterého se budou data zapisovat, existuje. Pokud ano, pak se data přepíšou, jestliže soubor neexistuje, nejprve se soubor vytvoří a poté se do něj zadají informace. Následující dvě metody jsou klíčové pro ukládání a načítání:

```
private void Save() {
    _savedData.Position = _player.gameObject.transform.position;
    _savedData.HP = _player.HitPoint;
    _savedData.Ammoes = _player.Ammoes;
    string jsonString = JsonUtility.ToJson(_savedData);
    File.WriteAllText(_savePath, jsonString);
}
public void Load() {
    if (File.Exists(_savePath)) {
        string fileData = File.ReadAllText(_savePath);
        SaveData dataToSave = JsonUtility.FromJson<SaveData>(fileData);
        _player.transform.position = dataToSave.Position;
        _player.HitPoint = dataToSave.HP;
        _player.Ammoes = dataToSave.Ammoes;
    }
}
```

Ukázka kódu 25: Implementace klíčových metod pro uložení a načítání hry

4.2.9 Efekty

Pro oživení scény byly přidány efekty světlušek a malých výbuchů ohně z lávy. Tyto efekty byly implementovány stejným způsobem, jak bylo popsáno v kapitole 4.2.4.6 Efekty, pouze s některými rozdíly v nastavení.

4.2.10 Post-processing

Post-processing označuje obecnou praxi zpracování obrazu na celou obrazovku, které nastává po vykreslení scény kamerou, ale před zobrazením scény na obrazovce. Tyto následné procesy mohou významně zlepšit vizuální aspekt výsledku s relativně krátkým časem na nastavení (Docfx, 2022).

Za účelem následného zpracování byla k hlavní kameře přidána komponenta „Post-Process Layer“ a v parametru „Trigger“ byla vybrána hlavní kamera. V okně „Hierarchy“ byl také vytvořen prázdný objekt (s názvem „Global Post Processing“). Poté byla k tomuto objektu přidána komponenta „Post-Process Volume“. Nastavení použitá v rámci post-processingu:

- Is Global → true
- Profile → tlačítko „new“

Poté byly pomocí tlačítka „Add effect“ přidány čtyři efekty – „Bloom“, „Color Grading“, „Chromatic Aberration“, „Vignette“. Tyto parametry mají následující nastavení:

- Bloom
  - Intensity → 5,5
  - Threshold → 0,9
  - Color → #FFC000
- Color Grading
  - Mode → High-Definition Range
  - Tonemapping
    - Mode → ACES
- Chromatic Aberration
  - Intensity → 0,15
- Vignette
  - Intensity → 0,15

4.2.11 Dokončení úrovně a mechanika smrti

Smrtelné zóny, neviditelné stěny a ukončení úrovně jsou nastaveny podobným způsobem, přičemž hlavní funkce jsou prováděny pomocí značek a logiky popsané ve skriptech.

Smrtelné zóny

Představují prázdný objekt s vytvořenou značkou „DeathZone“ a komponentou „Box Collider 2D“. Parametr „Is Trigger“ je nastaven na „true“. Následující skript je uveden v metodě „OnTriggerEnter2D ()“ ve třídě „Player:

```
        if (collider.CompareTag("DeathZone")) {
            if (_saveSystem.CanSave) SceneManager.LoadScene(0);
            else _saveSystem.Load();
        }
```

Ukázka kódu 26: Kontrola podmínek pro restartovaní úrovně nebo načítání kontrolního bodu

Jestliže hlavní postava narazí na objekt se značkou „DeathZone“, úroveň se restartuje, pokud hráč nedosáhl kontrolního bodu. Zóny smrti jsou láva a útesy.

Neviditelné stěny

Neviditelné stěny jsou stěny, které se nacházejí na začátku i na konci úrovně a nedovolují hráči vyjít za ně. Představují prázdný objekt s komponentou „Box Collider 2D“ s hodnotou „true“ parametru „Used By Effector“ a komponentou „Platform Effector 2D“, ve které nebylo nic změněno.

Ukončení úrovně

Je implementováno stejným způsobem jako smrtelná zóna, liší se pouze názvem značky –„Finish“ – a logikou skriptu. Následující skript je uveden v metodě „OnTriggerEnter2D ()“ ve třídě „Player“:

```
        if (collider.CompareTag("Finish")) {
            if (Coins >= 25) SceneManager.LoadScene(1);
            else StartCoroutine(ResetText());
        }
```

Ukázka kódu 27: Kontrola podmínek pro spouštěni finální scény dokončeni úrovně

Pokud hráč našel pětadvacet zlatých mincí, načte se další scéna se závěrečným textem a hra se po několika sekundách ukončí.

4.3 Testování

Je-li hra vytvářena jedním člověkem, obvykle nebývá k dispozici tým testerů, kteří by mohli testovat herní mechanismy během vývoje. Proto hra byla otestována autorem a byly ověřeny všechny mechaniky stanovené v cílech práce.

Například během testování byla nalezena následující chyba: když se hlavní hrdina dostal do dlouhodobého kontaktu s nepřáteli nebo hroty, všechna červená srdce zmizela během jedné sekundy a úroveň začala znovu. Pro vyřešení této chyby byl napsán skript s poměrně jednoduchou logikou – při prvním kontaktu s nepřítelem nebo hroty se zapne ignorování vrstev hráče a nepřátel i hroty po dobu jedné sekundy a poté se ignorování vypne. Tímto způsobem bude hlavní hrdina po dobu jedné sekundy nezranitelný a v případě, že kontakt bude pokračovat, hlavní hrdina bude opět zraněn.

Následující skript demonstruje implementaci:

```
private void Start() {
    …
    ResetDamageResistance();
    …
}
private IEnumerator DamageFx() {
    _spriteRenderer.color = Color.red;
    yield return new WaitForSeconds(0.25F);
    _spriteRenderer.color = Color.white;
    yield return new WaitForSeconds(0.25F);
    _spriteRenderer.color = Color.red;
    yield return new WaitForSeconds(0.25F);
    _spriteRenderer.color = Color.white;
    yield return new WaitForSeconds(0.25F);
    ResetDamageResistance();
}
private void ResetDamageResistance() {
    Physics2D.IgnoreLayerCollision(3, 7, false);
    Physics2D.IgnoreLayerCollision(10, 7, false);
}
```

Ukázka kódu 28: Implementace dočasné nezranitelnosti při kontaktu s nepřáteli a hroty

Důležité je nezapomenout vyvolávat metodu „ResetDamageResistance()“ v metodě „Start()“, aby hlavní hrdina při restartu nebyl nezranitelný.

5 Výsledky a diskuse

Po dokončení práce byl úspěšně vytvořen prototyp hry Sandy, který splňuje stanovené cíle a obsahuje funkční herní mechaniky. S ohledem na další rozvoj hry jsou navrženy kroky, které pomohou zlepšit její kvalitu i kapacitu. Tyto kroky zahrnují vylepšení příběhu, rozšíření počtu úrovní, vytvoření hlavního menu a přidání více překážek i nepřátel. Kromě toho může být ovládání hry vylepšeno, například přidáním rychlých pohybů (Dash), což zajistí vytváření zajímavějších úrovní. Prototyp je momentálně v otevřeném stavu a připraven pro další rozvoj k plnohodnotné hře.

6 Závěr

V rámci této práce byl vytvořen návrh 2D pixelové plošinové hry pomocí vývojového prostředí Unity, což představuje splnění hlavního cíle.

Teoretická část zahrnovala popis procesu tvorby hry a informace o pixel artu. Dále byla představena použitelnost editoru kódu Visual Studia pro práci s vývojovým prostředím Unity a vysvětleny funkce programů Krita, Pixilart a Jsfxr pro tvorbu grafických a zvukových efektů.

V praktické části práce byl vytvořen koncept 2D plošinové hry a implementovány herní mechaniky na základě tohoto konceptu. Vývoj prototypu probíhal v iterativním procesu a byla prováděna celá řada testování za účelem zajištění kvality a funkčnosti. Tato část práce může být užitečná pro začínající vývojáře, kteří se chtějí naučit některé herní mechaniky a získat návod na řešení problémů, jež se mohou objevit v různých fázích vývoje 2D her.

Přestože vývojové prostředí Unity i různé editory sloužící k tvorbě kódu, grafiky a zvukové podpory usnadňují proces tvorby her, stále jde o velmi složitý proces. Jak je popsáno v teoretické části, tvorba her vyžaduje všestranné znalosti, počínaje uměleckými dovednostmi a schopností pracovat s hudbou a konče managementem a programováním.

7 Seznam použitých zdrojů

ADOBE, 2021. Frame-by-frame animation in Animate. In: *Helpx.adobe.com* [online]. 27. 4. 2021 [cit. 2023-03-03]. Dostupné z: https://helpx.adobe.com/animate/using/frame-by-frame-animation.html

BARNES, Sara, 2022. How the Humble Pixel Became a Building Block To Groundbreaking Art. In: *Mymodernmet.com* [online]. 23. 3. 2022 [cit. 2023-02-27]. Dostupné z: https://mymodernmet.com/what-is-pixel-art/#:~:text=Pixel%20art%20was%20born%20in,was%20merely%20squares%20and%20rectangles

BLIZZARD ENTERTAINMENT, 2014. Hearthstone. *Hearthstone.blizzard.com* [online]. 11. 3. 2014 [cit. 2023-02-26]. Dostupné z: https://hearthstone.blizzard.com

BRAMBLE, Ross, 2023. The Seven Stages of Game Development. In: *Gamemaker.io* [online]. 4. 1. 2023 [cit. 2023-02-27]. Dostupné z: https://gamemaker.io/en/blog/stages-of-game-development

BUMM13, 2006. Screenshot of PONG from the Atari Arcade Hits #1 software title released Hasbro Interactive - a conversion of the original 1972 Atari Pong. In: *Wikimedia Commons* [online]. © 2006 [cit. 2023-02-27]. Dostupné z: https://commons.wikimedia.org/wiki/File:Pong.png

CAPCOM, 1989. Mega Man 2 (NES) Screenshots. *Nintendolife.com* [online]. © 1989 [cit. 2023-02-27]. Dostupné z: https://www.nintendolife.com/games/nes/mega_man_2/screenshots

COMPUTER HOPE, 2017. Indie game. In: *Computerhope.com* [online]. 26. 4. 2017 [cit. 2023-03-06]. Dostupné z: https://www.computerhope.com/jargon/i/indie-game.htm

CONCERNEDAPE, 2016. Stardew Valley. In: *Stardewvalley.net* [online]. 26. 2. 2016 [cit. 2023-02-27]. Dostupné z: https://www.stardewvalley.net/

CUPHEAD, 2017. Cuphead: Don't Deal With The Devil. *Cupheadgame.com* [online]. © 2017 [cit. 2023-02-27]. Dostupné z: https://www.cupheadgame.com/

DENNATON GAMES, 2012. Hotline Miami. *Hotlinemiami.com* [online]. © 2012 [cit. 2023-02-27]. Dostupné z: https://www.hotlinemiami.com/

DOCFX, 2021. Get started with the Unity Hub. *Unity3d.com* [online]. © 2021 [cit. 2023-02-26]. Dostupné z: https://docs.unity3d.com/hub/manual/index.html

DOCFX, 2022. Post Processing Stack v2 overview. In: *Docs.unity3d.com* [online]. 14. 4. 2022 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.postprocessing@3.2/manual/index.html

ENCORA, 2021. How to Take Advantage of Parallax in Programming and Video Games. In: *Encora.com* [online]. 23. 9. 2021 [cit. 2023-02-28]. Dostupné z: https://www.encora.com/insights/how-to-take-advantage-of-parallax-in-programming-and-video-games

EXTREMELY OK GAMES, 2018. Celeste. *Exok.com* [online]. © 2018 [cit. 2023-02-27]. Dostupné z: https://exok.com/games/celeste/

FOX, Toby, 2015. Undertale. *Undertale.com* [online]. © 2015 [cit. 2023-02-27]. Dostupné z: https://undertale.com/

FREDRICKSEN, Eric, 2011. Jsfxr - 8 bit sound maker and online sfx generator. In: *Sfxr.me* [online]. © 2011 [cit. 2023-03-04]. Dostupné z: https://sfxr.me/#:~:text=Jsfxr%20is%20an%20online%208,sound%20effects%20in%20your%20games

GRIFFITHS, Devonte, 2018. The History Of Pixel Art. The Factory Times. In: *Thefactorytimes.com* [online]. 27. 9. 2018 [cit. 2023-02-27]. Dostupné z: http://www.thefactorytimes.com/factory-times/2018/9/27/the-history-of-pixel-art

HALPERN, Jared, 2019. *Developing 2D games with Unity: indipendent game programming with C#.* New York: APress. ISBN 978-1-4842-3771-7.

HOSCH, William L., 2023. Role-playing video game. In: *Britannica.com* [online]. 3. 2. 2023 [cit. 2023-02-27]. Dostupné z: https://www.britannica.com/topic/role-playing-video-game

IBM, 2023a. What is an API (application programming interface)? *Ibm.com* [online]. © 2023 [cit. 2023-02-23]. Dostupné z: https://www.ibm.com/topics/api

IBM, 2023b. What is a workflow? *Ibm.com* [online]. © 2023 [cit. 2023-02-23]. Dostupné z: https://www.ibm.com/topics/workflow

KINEMATICSOUP, 2016. Game Development Workflow. In: *Kinematicsoup.com* [online]. 26. 10. 2016 [cit. 2023-02-26]. Dostupné z: https://www.kinematicsoup.com/news/2016/10/26/game-development-workflow

KRITA, 2009. Krita Features. In: *Krita.org* [online]. 24. 12. 2009 [cit. 2023-02-27]. Dostupné z: https://krita.org/en/item/krita-features/

KRITA, 2018. Welcome to the Krita 5.0 Manual! *Docs.krita.org* [online]. © 2018 [cit. 2023-02-27]. Dostupné z: https://docs.krita.org/en/index.html

LÖW, Andreas, 2023. What is a sprite sheet?. In: *Codeandweb.com* [online]. © 2023 [cit. 2023-02-25]. Dostupné z: https://www.codeandweb.com/what-is-a-sprite-sheet

METROID RECON, 2014. The environments of Super Metroid, 1994. In: *Metroid.retropixel.net* [online]. 1. 5. 2014 [cit. 2023-02-27]. Dostupné z: https://metroid.retropixel.net/games/metroid3/screenshots/

MICROSOFT, 2021. Visual Studio Tools for Unity. In: *Learn.microsoft.com* [online]. 28. 12. 2021 [cit. 2023-03-05]. Dostupné z: https://learn.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity?pivots=windows

MICROSOFT, 2023a. Object-Oriented programming (C#). In: *Learn.microsoft.com* [online]. 3. 1. 2023 [cit. 2023-03-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop

MICROSOFT, 2023b. It's how you make software. Visual Studio: IDE and Code Editor for Software Developers and Teams. *Visualstudio.microsoft.com* [online]. © 2023 [cit. 2023-03-05]. Dostupné z: https://visualstudio.microsoft.com

MIGNANO, Marco, 2016. Game Development Pipeline: From Concept To Store. In: *Gamedevelopertips.com* [online]. 11. 7. 2016 [cit. 2023-02-26]. Dostupné z: https://gamedevelopertips.com/game-development-pipeline/

NINTENDO, 1985. The official home of Super Mario™ – History. *Nintendo.com* [online]. 18. 10. 1985 [cit. 2023-02-27]. Dostupné z: https://mario.nintendo.com/history/

PERFORCE, 2023. Game Development Process: How Game Development Works. *Perforce.com* [online]. © 2023 [cit. 2023-02-26]. Dostupné z: https://www.perforce.com/resources/game-development-process

PETTERSSON, Tomas, 2012. Sfxr. In: *Drpetter.se* [online]. 14. 9. 2012 [cit. 2023-03-05]. Dostupné z: http://www.drpetter.se/project_sfxr.html

PIXILART, 2023. Pixilart Pixel Editor Features. *Pixilart.com* [online]. © 2023 [cit. 2023-03-05]. Dostupné z: https://www.pixilart.com/features

PULSE COLLEGE, 2023. What is game development? Everything you need to know. *Pulsecollege.com* [online]. © 2023 [cit. 2023-02-25]. Dostupné z: https://www.pulsecollege.com/what-is-game-development-everything-you-need-to-know/

STEFYN, Nadia, 2022. How video games are made: the game development process. CG Spectrum. In: *Cgspectrum.com* [online]. 9. 5. 2022 [cit. 2023-02-25]. Dostupné z: https://www.cgspectrum.com/blog/game-development-process

SYLVESTER, Tynan a Ludeon STUDIOS, 2013. RimWorld. In: *Rimworldgame.com* [online]. 4. 11. 2013 [cit. 2023-02-27]. Dostupné z: https://rimworldgame.com/

TECHOPEDIA, 2023. Pixel Art. *Techopedia.com* [online]. © 2023 [cit. 2023-02-27]. Dostupné z: https://www.techopedia.com/definition/8884/pixel-art

TSEKHANSKY, Alex, 2019. Game Production Pipeline Infographic. In: *Theknightsofunity.com* [online]. © 2019 [cit. 2023-02-26]. Dostupné z: https://blog.theknightsofunity.com/game-production-pipeline/

UNITY TECHNOLOGIES, 2020. Prefabs. In: *Learn.unity.com* [online]. 12. 10. 2020 [cit. 2023-02-26]. Dostupné z: https://learn.unity.com/tutorial/prefabs-e?language=en#

UNITY TECHNOLOGIES, 2023a. Unity's interface. In: *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/UsingTheEditor.html

UNITY TECHNOLOGIES, 2023b. Scripting in Unity for Experienced Programmers. *Unity.com* [online]. © 2023 [cit. 2023-02-27]. Dostupné z: https://unity.com/how-to/programming-unity#scripting-components-unity

UNITY TECHNOLOGIES, 2023c. GameObjects. In: *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-02-27]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/GameObjects.html

UNITY TECHNOLOGIES, 2023d. Sprites. In: *Docs.unity3d.com* [online]. 24. 2. 2023 [cit. 2023-02-25]. Dostupné z: https://docs.unity3d.com/Manual/Sprites.html

UNITY TECHNOLOGIES, 2023e. Game Development Terms. *Unity.com* [online]. © 2023 [cit. 2023-02-23]. Dostupné z: https://unity.com/how-to/beginner/game-development-terms

UNITY TECHNOLOGIES, 2023f. Unity User Manual 2021.3 (LTS). In: *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/Manual/index.html

UNITY TECHNOLOGIES, 2023g. Important Classes – MonoBehaviour. *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/Manual/class-MonoBehaviour.html

UNITY TECHNOLOGIES, 2023h. MonoBehaviour.Start(). In: *Docs.unity3d.com* [online]. 24. 2. 2023 [cit. 2023-03-26]. Dostupné z: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html

UNITY TECHNOLOGIES, 2023i. Game perspectives for 2D games. In: *Docs.unity3d.com* [online]. © 2023 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/Manual/Quickstart2DPerspective.html

UNITY TECHNOLOGIES, 2023j. Tilemap. In: *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-03-03]. Dostupné z: https://docs.unity3d.com/Manual/class-Tilemap.html

UNITY TECHNOLOGIES, 2023k. Time.fixedDeltaTime. In: *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/ScriptReference/Time-fixedDeltaTime.html

UNITY TECHNOLOGIES, 2023l. Animator component. In: *Docs.unity3d.com* [online]. 3. 3. 2023 [cit. 2023-03-04]. Dostupné z: https://docs.unity3d.com/Manual/class-Animator.html

VIONIX, 2021. Unity Update function, Late update, and fixed update explained. In: *Vionixstudio.com* [online]. 28. 1. 2021 [cit. 2023-02-27]. Dostupné z: https://vionixstudio.com/2021/01/19/unity-update-function/

WAIN, Sam, 2022. How To Make Pixel Art For 2D Games. In: *Gamemaker.io* [online]. 24. 8. 2022 [cit. 2023-02-27]. Dostupné z: https://gamemaker.io/ru/blog/make-pixel-art-2d-games

8 Seznam obrázků