Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj 2D herní aplikace pro mobilní zařízení

Jan Doležal

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Doležal

Informatika

Název práce

Vývoj 2D herní aplikace pro mobilní zařízení

Název anglicky

Development of a 2D game application for mobile devices

Cíle práce

Cílem bakalářské práce je charakteristika vývoje mobilní hry pro platformu Android. Práce bude obsahovat popis základních postupů a technik při práci s herním enginem Unity, díky tomu bude moci sloužit jako návod nejen začínajícím vývojářům. Dílčím cílem práce je implementace hry, která po důkladném uživatelském testování, bude publikována na Google Play. Součástí práce bude také podrobný popis postupu samostatného zveřejnění hry.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu odborné literatury a dokumentace Unity. V teoretické části budou popsány základní postupy při práci v Unity a jejím asset store, také jak probíhá vývoj mobilní aplikace a v neposlední řadě budou představeny různé druhy herních enginů a rozdíly mezi nimi. Pomocí znalostí z teoretické části bude probíhat samotná implementace hry se scripty psanými v jazyce C#.

Doporučený rozsah práce

35-40 stran

Klíčová slova

C#, Unity, herní engine, mobilní hra, OS Android

Doporučené zdroje informací

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

SCHELL, Jesse. The Art of Game Design: A Book of Lenses. Amsterdam; Boston: Elsevier/Morgan Kaufmann, 2008. ISBN 978-0-12-369496-6

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

VIRIUS, Miroslav. Programování v C#: od základů k profesionálnímu použití. Praha: Grada Publishing, 2021. Knihovna programátora (Grada). ISBN 978-80-271-1216-6

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. Tomáš Benda

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 9. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 27. 02. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj 2D herní aplikace pro mobilní zařízení" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2024 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) vedoucímu Ing. Tomáši Bendovi za odborné vedení mé bakalářské práce, za pomoc, ochotu, cenné rady a připomínky.

Vývoj 2D herní aplikace pro mobilní zařízení

Abstrakt

Bakalářská práce poskytuje ucelený pohled na vývoj mobilních her s využitím herního enginu Unity a zároveň ukazuje, jak se tyto hry mohou dostat k širokému publiku prostřednictvím platformy Google Play. Je tak ideálním průvodce pro čtenáře při tvorbě vlastní hry.

V teoretické části práce jsou čtenáři představeny známé herní enginy včetně jejich výhod a nevýhod a jejich srovnání subjektivním hodnocení autora. Dále následuje podrobný popis herního enginu Unity, včetně jeho rozhraní, funkcí a komponent. Tyto znalosti jsou dále uplatněny v praktické části práce, kde pomocí nich probíhá vývoj samostatné hry.

Výsledkem praktické části je publikování funkční 2D roguelike mobilní hry na platformu Google Play. Cílem této hry je přežití v aréně po určitý čas. Během této doby se kolem hráče objevují nepřátelé, které musí přemoci pomocí kouzel, jenž si postupně během hry vybírá a vylepšuje.

Klíčová slova: C#, Unity, herní engine, mobilní hra, OS Android

Development of a 2D game application for mobile devices

Abstract

The bachelor thesis provides a comprehensive view of the development of mobile games using the Unity game engine while also demonstrating how these games can reach a wide audience through the Google Play platform. Therefore, it serves as an ideal guide for readers in the creation of their own game.

In the theoretical part of the thesis, readers are introduced to well-known game engines. The work shows their advantages and disadvantages and compared them according to the author's subjective evaluation. This is followed by a detailed description of the Unity game engine, which includes its interface, functions, and components. These insights are further applied in the practical part of the thesis, where they are used for development of an independent game.

The result of the practical part is the publication of a functional 2D roguelike mobile game on the Google Play platform. The main goal of the game is that the player tries to survive in an arena for a certain time. During this time, enemies appear around the player. They must overcome them by using spells that they gradually choose and improve throughout the game.

Keywords: C#, Unity, Game engine, Mobile game, Android OS

Obsah

1 Úvod

Hry se stávají čím dál tím významnější součástí každodenního života a jejich popularita výrazně roste. Tento trend je spojen s neustálým pokrokem ve vývoji herních technologií. Zatímco v minulosti bylo vytváření her komplikovaným a náročným procesem, dnes je k dispozici široká škála herních enginů, které značně usnadňují a urychlují tvorbu.

Herní engine je základem herního vývoje a představuje klíčový nástroj, který umožňuje vývojářům vytvářet hry efektivněji a s menším úsilím. Tyto enginy poskytují vývojářům sadu funkcí a nástrojů, které jim umožňují realizovat jejich nápady. Výběr správného herního enginu je klíčovým krokem pro úspěšný vývoj hry, a proto je důležité porozumět různým druhům a jejich vlastnostem.

Bakalářská práce se věnuje vytváření mobilních her pomocí herního enginu Unity. Unity je jedním z nejpopulárnějších herních enginů na trhu a nabízí rozsáhlou škálu nástrojů a funkcí, které usnadňují tvorbu her pro různé platformy. Cílem této práce je nejen prostudovat teoretické aspekty vývoje her a herních enginů, ale také provést konkrétní implementaci mobilní hry v Unity a následně ji publikovat na platformě Google Play.

Teoretická část práce se zaměřuje na popis a porovnání různých herních enginů pro lepší porozumění jejich výhodám a omezením. Dále se detailně zabývá Unity, jeho nástroji a funkcemi, které jsou klíčové pro tvorbu her. Tyto znalosti budou následně aplikovány při implementaci samostatné mobilní hry se zaměřením na praktickou část procesu vývoje.

Hlavním cílem této bakalářské práce je poskytnout ucelený pohled na vývoj mobilních her s využitím herního enginu Unity a zároveň ukázat, jak se tyto hry mohou dostat k širokému publiku prostřednictvím platformy Google Play.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem bakalářské práce je charakteristika vývoje mobilní hry pro platformu Android. Práce bude obsahovat popis základních postupů a technik při práci s herním enginem Unity, díky tomu bude moci sloužit jako návod nejen začínajícím vývojářům. Dílčím cílem práce je implementace hry, která po důkladném uživatelském testování, bude publikována na Google Play. Součástí práce bude také podrobný popis postupu samostatného zveřejnění hry.

2.2 Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu odborné literatury a dokumentace Unity. V teoretické části budou popsány základní postupy při práci v Unity a jejím asset store, také jak probíhá vývoj mobilní aplikace a v neposlední řadě budou představeny různé druhy herních enginů a rozdíly mezi nimi. Pomocí znalostí z teoretické části bude probíhat samotná implementace hry se scripty psanými v jazyce C#.

3 Teoretická východiska

Tato kapitola seznámí čtenáře s nejznámějšími herními enginy současnosti, přičemž větší důraz klade na herní engine Unity.

3.1 Herní engine

Herní engine je prostředí pro zjednodušení vývoje videoher a aplikací. Výrazně urychluje a usnadňuje práci vývojářům při tvorbě her. Zpravidla obsahuje mnoho nástrojů, pomocí nichž se tvoří vizuál hry, herních mechaniky a ozvučení. [1]

Existuje mnoho herních enginů a nejde říct, který z nich je nejlepší, protože každá hra má na engine jiné požadavky. Proto si velké herní společnosti vyvíjejí vlastní herní enginy, které přesně pasují a umožňují vše, co daný titul zrovna potřebuje. [29]

V následujících podkapitolách jsou představeny některé nejznámější herní enginy.

3.1.1 Unreal Engine

Unreal engine je jeden z nejpopulárnějších herních enginů současnosti, vyvíjí ho společnost Epic Games od roku 1998. Tento engine je velmi dobře zdokumentovaný, a proto je vhodný pro začátečníky ale také i pro pokročilé vývojáře. Mezi jeho silné stránky patří například výkonost oproti ostatní konkurentům, dále poskytuje velmi rozsáhlé nástroje pro vývoj her pro virtuální realitu. Mezi nejznámější hry vytvořené v Unreal enginu patří například Fortnite anebo Ark. [2]

3.1.2 Godot

Godot je herní engine, který má unikátní přístup ke scéně pomocí nodes, kde každému uzlu je možné přiřadit script, pomocí kterého se kontroluje chování objektů na scéně. Na rozdíl od ostatních herních enginů, Godot poskytuje možnost psát scripty ve speciálním jazyce gdcript, což je kombinace mezi programovacími jazyky typescrip a python. Tento herní engine je zdarma a open-source a je velmi dobrou volbou pro tvorbu 2D a 3D her. Unity nebo Unreal jsou ale známější a populárnější, a tak pro tento engine existuje méně návodů a matriálu pro práci s ním. Mezi známe tituly vytvořené pomocí tohoto enginu patří například hra Brotato. [2] [3]

3.1.3 CryEngine

CryEngine je herní engine vyvíjen společností CryTek od roku 2002. Pomocí něho je vytvořeno mnoho populárních her jako například všechny hry od společnosti CryTek a také známé tituly jako například Kingdom Come: Deliverence od společnosti Warhorse Studios. Mezi přednosti se řadí jednoduchost, výkonný editor a podpora virtuální reality. [2]

3.1.4 Amazon Lumberyard

Lumberyard je herní engine od společnosti Amazon. Jeho technologie jsou založeny na CryEnginu, které jsou vylepšeny a rozšířeny. Tento engine je zdarma, obsahuje 3D modelovací nástroje a také podporu virtuální reality. Mezi nejznámější tituly vytvořené pomocí tohoto enginu patří The Grand Tour Game a Crucible. [4]

3.1.5 Unity

Unity je herní engine pro vývoj 2D a 3D her vyvíjen společností Unity Technologies od roku 2005. Jedním z klíčových momentů v historii bylo vydání Unity 5 v roce 2015, které přineslo řadu vylepšení, včetně zvýšení grafické kvality a podpory pro více platforem. Tato verze získala pozornost vývojářského světa a přivedla Unity ještě blíže k pozici jednoho z hlavních hráčů na trhu s herním vývojem. [5]

Přednosti Unity:

- Snadné použití – nabízí intuitivní uživatelské rozhraní a velké množství nástrojů pro vývoj her
- Multiplatformní podpora – umožňuje vývojářům vytvářet hry a aplikace pro širokou škálu platforem, včetně PC, konzolí a mobilních zařízení
- Velká komunita – má velkou a aktivní komunitu vývojářů, která poskytuje mnoho návodů a rad, jak s Unity pracovat
- Asset Store – Unity Asset Store je místem, kde vývojáři mohou zakoupit a prodávat herní modely, pluginy a nástroje, které usnadňují vývoj her

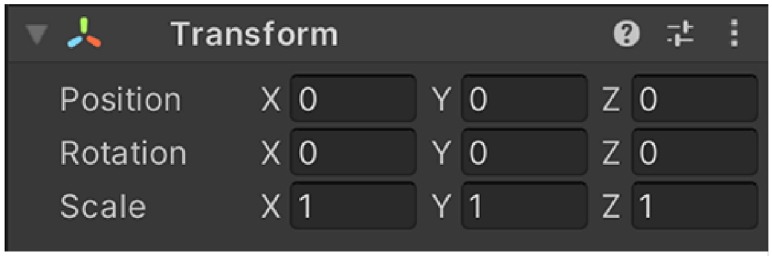
V následujících podkapitolách budou představeny, všechny důležité koncepty, se kterými se setká každý, kdo vyvíjí hru v Unity.

3.1.5.1 GameObject

Jako GameObject se nazývá všechno, co je vidět na obrazovce, ať už se jedná o postavu hráče, stěnu budovy nebo zbraň nepřítele. Tento herní objekt, ale sám o sobě nic nedělá, jedná se pouze o obal, ke kterému se dále připojují komponenty, které určují chování daného GameObjektu. Existuje mnoho komponent, přičemž každá má své vlastní využití. Níže jsou uvedeny ty nejznámější, bez kterých se neobejde skoro žádný komplexnější herní objekt.

Transform

Jedná se o komponent, který je vždy připnutý k hernímu objektu a nejde odebrat. Určuje pozici, rotaci a měřítko objektu na scéně. Jednotlivé hodnoty se dají nastavit v inspektoru daného objektu nebo přímo ze scriptu. [6]



Obrázek 1 – Transform [6]

Následující ukázka kódu nastaví hernímu objektu, ke kterému je tento script připnutý, souřadnice na 0, 0 a 0.

```
void Start()
{
    transform.position = new Vector3(0, 0, 0);
}
```

Vlastnosti komponenty Transform jsou používány v celé hierarchii herního objektu. Proto pokud se změní pozice rodiče, tato změna ovlivní také všechny jeho potomky v hierarchii.

Collider

Komponent, který slouží k řešením kolizí mezi objekty ve scéně. Existuje několik, předpřipravených tvarů, které se nazývají **primitive colliders**. Ty jsou díky své jednoduchosti málo náročné pro procesor. Patří mezi ně například Box Collider, Sphere Collider, Capsule Collider a jejich zjednodušené verze ve 2D.

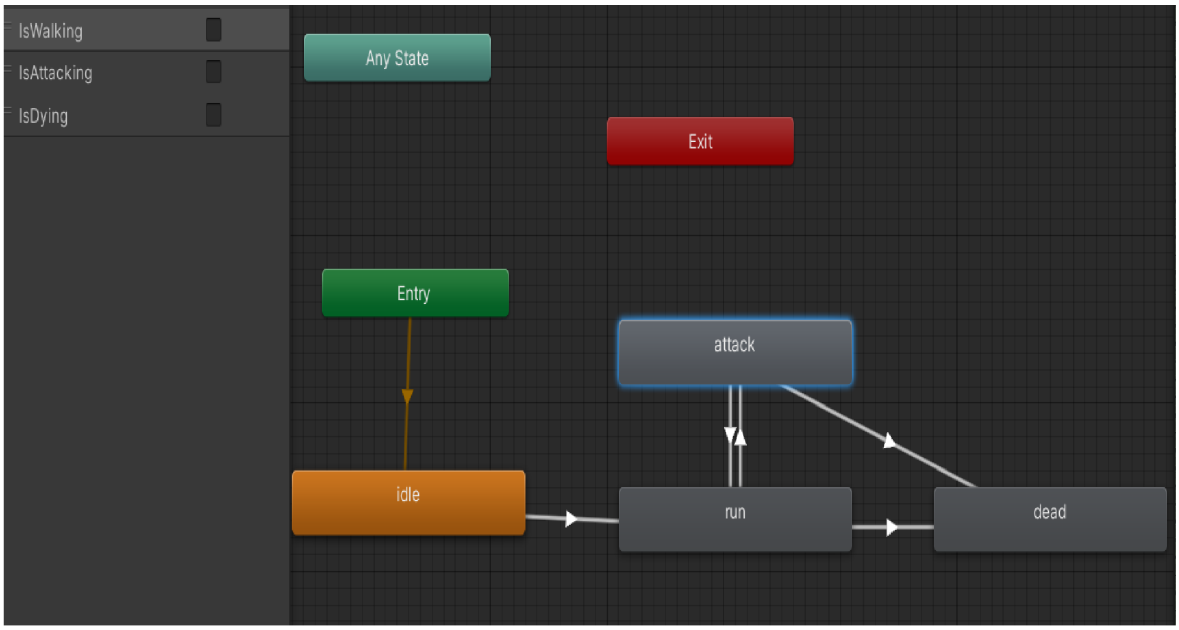
Aby se na objekty ve scéně vztahovaly klasické fyzikální veličiny jako jsou gravitace, hmotnost, odpor a hybnost je potřeba k danému objektu připnout komponentu **Rigid Body**, která společně s Collider docílí přesného řešení kolizí.

Trigger, neboli komponent Collider, který má zapnutou možnost „Is Trigger‘‘ je speciálním druhem collideru, který slouží ke spouštění funkcí při kolizi dvou objektů. Na rozdíl od normálních kolizí tyto objekty spolu nijak neinteragují. Velmi častým příkladem může být střela hráče, u které není potřeba, aby na hráče působila z fyzické stránky (posunula ho). [7] [8]

Animator

Je komponent, pomocí něhož se provádí animace objektu, ke kterému je připojen. Ke správné funkčnosti vyžaduje **Animator Controller**, který slouží k přepínání jednotlivých klipů a vytváření přechodů mezi nimi.

Animator Controller se skládá z několika obdélníků, který představují jednotlivé stavy daného objektu. Mezi těmito stavy se dají vytvořit přechody, které obsahují podmínky, pomocí niž se řídí přepínání mezi jednotlivými stavy. Základní Animator Controller, který řídí animace obyčejného nepřítele může vypadat jako je uvedeno na obrázku níže.



Obrázek 2 - Animator Controller (vlastní zpracování)

Každý Animator Controller zpravidla obsahuje tři základní stavy:

- Entry – stav, ve kterém každý objekt začíná a ze kterého se dále přechází do dalších stavů
- Exit – tento stav slouží k ukončení animace a navracení do počátečního stavu Entry
- Any State – slouží k přechodu do určitého stavu bez ohledu na to, ve kterém stavu se objekt právě nachází. [9]

Audio

Audio Listener je komponent, který slouží k zaznamenávání všeho zvuku ve scéně. Nejčastěji se umisťuje na hlavní kameru, protože je to místo, ze kterého je možné slyšet všechen zvuk ve 3D.

Audio Source je zodpovědný za přehrávání zvukových nahrávek. Přidává se k objektům, které někdy během pobytu na scéně přehrají nějaký zvuk. Jedná se například o objekt hráče, který přehrává zvuky při určitých situacích jako je obdržení zranění atd. Tento komponent umožňuje nastavit zvukové nahrávce několik vlastností, přímo z inspektoru jako je například spuštění nahrávky při vzniku objektu nebo hlasitost. [10]

3.1.5.2 Scripty

Jsou velmi důležitou součástí každého objektu ve hře. Dá se pomocí nich řídit celý objekt a přistupovat k ostatním komponentám objektu. Při vytvoření nového scriptu vznikne třída se jménem daného scriptu, která automaticky dědí ze třídy **MonoBehaviour**. Tato třída nabízí funkce životního cyklu, které usnadňují vývoj s Unity. Mezi nejpoužívanější funkce patří **Start** a **Update**. Funkce Start se provede při inicializaci daného objektu, což se hodí zejména pro nastavování referencí na jiné objekty ve scéně, se kterými se dále v tomto scriptu pracuje. Naopak funkce Update se provádí při každé aktualizaci obrazovky, proto sem patří vše, co je potřeba provádět neustále, jako je například pohyb vystřelené střely.

```
using UnityEngine;
using System.Collections;

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

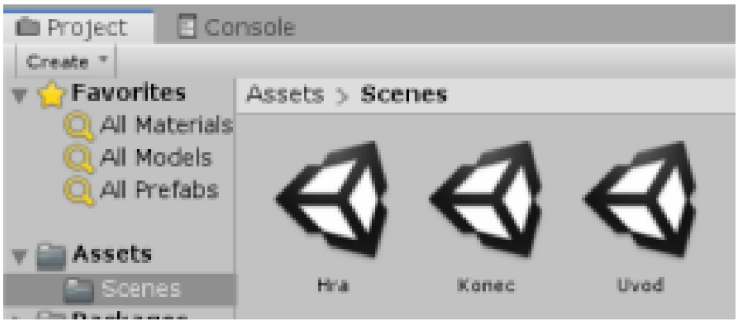
Obrázek 3 - Základní script [6]

Další velmi důležitou funkcí, kterou třída MonoBehaviour poskytuje je **Awake**. Tato funkce se stejně jako Start provede pouze jednou na začátku. Awake je ale na rozdíl od Start volán hned potom, co je objekt s daným scriptem inicializován neboli všechny funkce Awake se provedou před voláním funkcí Start. Tato funkcionalita je užitečná v případech, kdy inicializace objektu A závisí na objektu B, který je již inicializován. Proto inicializace objektu B by měla být provedena ve funkci Awake, zatímco objektu A by měla být provedena ve funkci Start.

Příkladem použití může být script, který vytváří a inicializuje hráče, v něm je Awake nezbytný pro správnou funkci UI scriptu, který zobrazuje informace o hráči na obrazovce. Pokud by hráč nebyl vytvořen a inicializován před spuštěním UI scriptu, mohlo by dojít k chybě při pokusu o získání vlastností hráče pro jejich zobrazení na obrazovce. Z tohoto důvodu je důležité provést vytvoření a inicializaci hráče v metodě Awake, aby bylo zajištěno, že hráč je připraven k použití dříve, než je spuštěn jakýkoliv kód UI scriptu. [24]

3.1.5.3 Scéna

Další důležitým prvkem, který se v Unity vyskytuje jsou scény. Scény obsahují všechny objekty, které daná hra potřebuje. Jednodušší hry si většinou vystačí jen s jednou scénou, ale naopak hry složitější jich obsahují několik, například každý level hry je v jiné scéně. To má za důsledek rychlejší načítání hry a zlepšení její plynulosti. Každá hra by měla obsahovat aspoň tři základní scény jako je úvod nebo hlavní menu, samotnou hru a konec jak je uvedeno na obrázku číslo 4. [27]



Obrázek 4 – Scény [18]

3.1.5.4 Prefabs

Prefab je šablona objektu, která obsahuje všechny komponenty daného objektu, jejich hodnoty nastavení a také jeho potomky. Oproti klasické kopii se liší v tom, že když provedeme úpravu tohoto prefabu, například změníme jeho velikost, tato změna se nám promítne do všech jeho instancí.

Kdykoliv když chceme použít jeden objekt, například nepřátelskou postavu na více místech ve scéně anebo ve více scénách je doporučené z tohoto objektu vytvořit prefab. Do scény se tento prefab přidá klasickým přetažením do okna hierarchie nebo přímo z kódu, což je nejčastější způsob práce s prefaby. Vytvoření instance obstará funkce Instantiate(), která přijímá následující parametry: prefab, pozici a otočení. Příklad použití téhle funkce je uveden v kódu níže.

```
public spawnEnemy()
{
    Instantiate(prefab, new Vector3(0,0,0), Quaternion.identity);
}
```

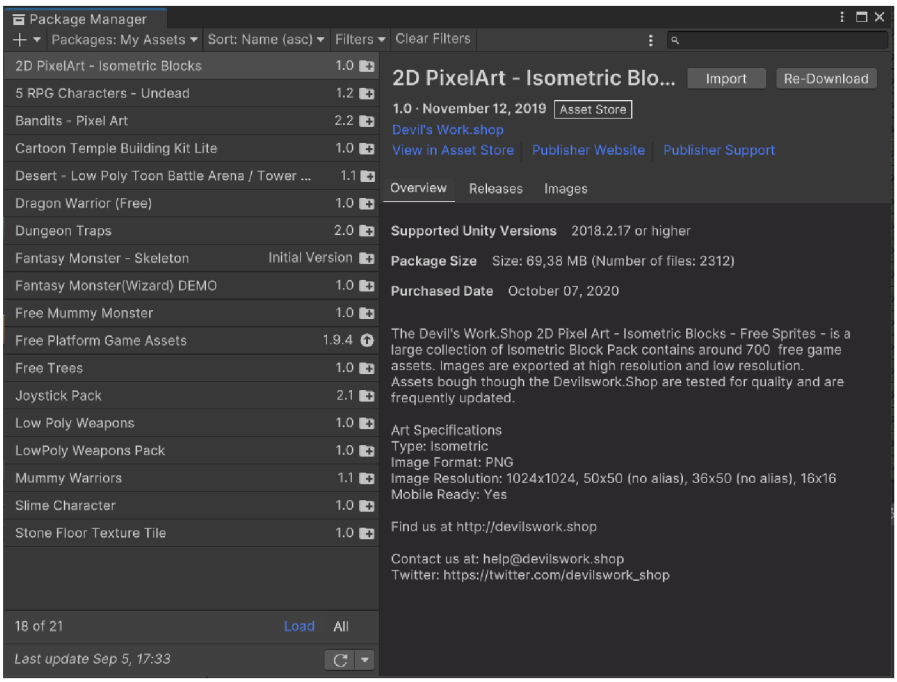
Prefaby se dají různě kombinovat a vytvářet takzvané **nested prefabs**, což jsou prefaby, které obsahují jiné prefaby. To může být užitečné, když je potřeba zkombinovat

více opakovaně použitelných objektů dohromady a vytvořit něco nového, ale zároveň mít možnost aktualizovat každý z nich jednotlivě. [11][26]

3.1.5.5 Asset Store

Unity Asset Store je rozsáhlá platforma pro vývojáře her a aplikací, která nabízí širokou škálu nástrojů, modelů, textur a scriptů. Tato platforma slouží jako nepostradatelný zdroj pro ty, kteří chtějí zrychlit a obohatit svůj vývojový proces. Unity Asset Store umožňuje vývojářům rychle najít a zakoupit komponenty, které potřebují k vytvoření vysoce kvalitních her a aplikací. Díky komunitnímu přístupu mohou také vývojáři sdílet své vlastní zdroje a získat z nich příjem.

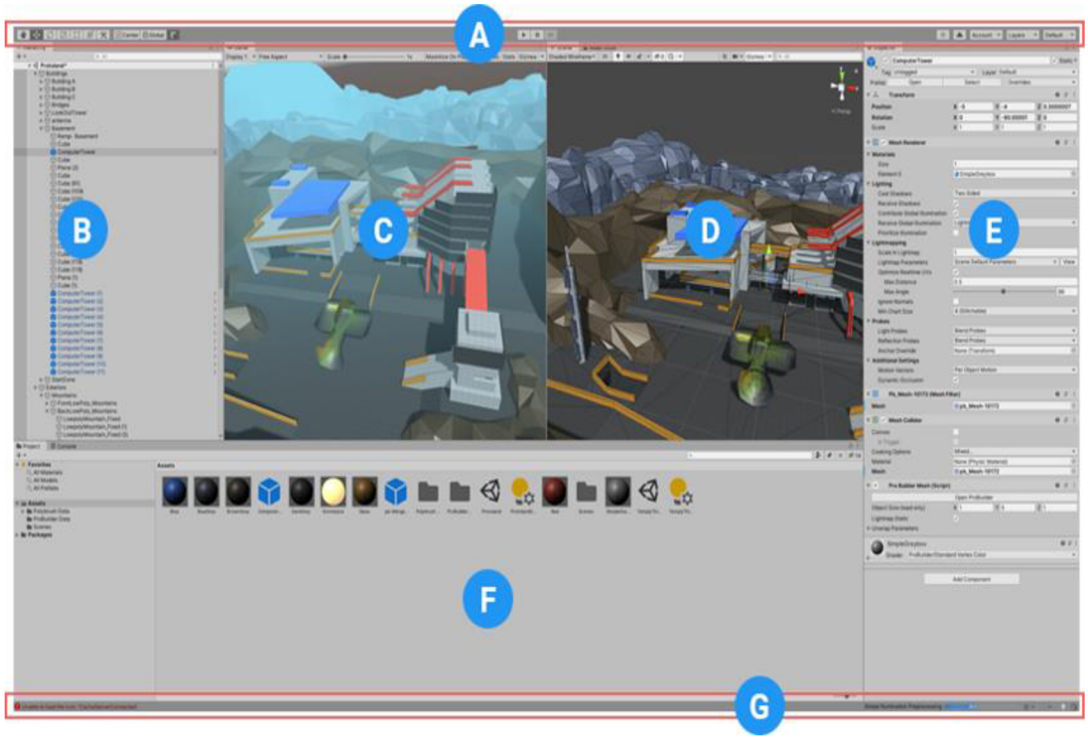
K zahrnutí jednotlivých modelů z Asset Store do projektu složí nástroj **Package manager**, pomocí nějž je možné jednoduše spravovat všechny uživatelovy balíčky.



Obrázek 5 - Package manager (vlastní zpracování)

3.1.5.6 Uživatelské rozhraní

Prostředí je navrženo tak, aby bylo snadno ovladatelné a zároveň flexibilní pro různé potřeby vývojářů. Všechna okna se dají přesouvat a dá se měnit jejich velikost dle potřeby.



Obrázek 6 - Uživatelské rozhraní [6]

Na obrázku číslo 6, který je zobrazen výše je základní uživatelské rozhraní. Přičemž jednotlivá písmena označují důležité části:

- (A) Panel nástrojů – poskytuje přístup k nejdůležitějším funkcím.
- (B) Okno hierarchy – obsahuje všechny objekty ve scéně.
- (C) Zobrazení hry – simuluje, jak bude vypadat vykreslená hra.
- (D) Zobrazení scény – zde se provádí všechny úpravy.
- (E) Okno inspektora – umožňuje prohlížet a upravovat vlastnosti vybraného objektu.
- (F) Okno projekt – zobrazuje všechny soubory v daném projektu.
- (G) Stavový řádek – poskytuje upozornění na různé procesy Unity. [23]

3.1.5.7 Osvětlení

Osvětlení scény je jeden z nejdůležitějších prvků při vývoji hry. Existuje mnoho způsobu, jak jej realizovat, v závislosti na tom, jakého výsledku se snažíme docílit. Unity poskytuje čtyři základní druhy světel.

- **Directional Light** – jedná se o velké, vzdálené světlo, které se používá nejčastěji k simulaci slunečního světla
- **Point Light** – světlo, které má pevně danou pozici a poloměr dosahu paprsků, které cestují všemi směry
- **Spot Light** – na rozdíl od Point Light tento druh světla poskytuje možnost vyslat paprsky pouze jedním směrem pod určitým úhlem
- **Area Light** – světla, která mají obdélníkový tvar a promítají se pouze jedním směrem

Při vytváření světla je dále možné výběru ze tří režimů. **Real-time Lighting**, jak už název vypovídá, je metoda, kdy všechny výpočty potřebné k tvorbě světla se provádí každý jednotlivý snímek. Tento režim je vhodný pro osvětlení postav nebo jiné pohyblivé geometrie. Kvůli neustálým výpočtům zatěžuje však tento režim CPU a GPU. Proto Unity poskytuje režim **Baked Lighting**, který naopak všechny výpočty provede předem a uloží si jejich výsledky na disk, ze kterého jsou čteny dle potřeby. [12]

3.1.5.8 Tag

Jako Tag neboli Štítky se v Unity označuje systém, pomocí kterého je možné jednoduše identifikovat objekty mezi sebou. V inspektoru každého objektu je možné najít políčko Tag, kde se přiřazuje danému objektu štítek už z předem vytvořených štítků nebo v případě potřeby je možné nějaký nový vytvořit. K takto přiřazeným štítkům lze přistupovat přímo z kódu například při kontrole kolizí, jak je uvedeno v kódu níže.

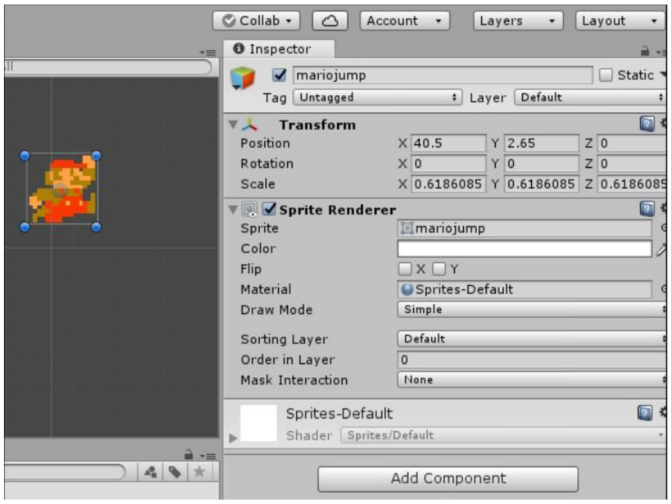
```
private void OnTriggerEnter2D(Collider2D other){
    if(other.CompareTag("Enemy")){
        Destroy(this.gameObject);
    }
}
```

3.1.5.9 Unity 2D

Unity engine je známý hlavně pro svou schopnost vytvářet 3D hry, to ale neznamená, že pomocí něho není možné vytvořit i 2D hry. Všechny postupy a informace, které byly výše zmíněné se týkají všech typů her bez závislosti na tom, jak daná hra vypadá. Existuje ale několik nástrojů a speciálních funkcí, které Unity poskytuje pro vývoj 2D her a ty více rozebírají podkapitoly této kapitoly.

Grafika

Pro vytváření obrazu ve 2D hrách se používají **Sprites**. Jedná se o jednoduché objekty neboli obrázky, které jsou vždycky otočené ke kameře. Tyto sprites se přidávají k objektům pomocí komponentu Sprite Renderer, který umožňuje měnit mnoho vlastností objektu jako je například barva obrázku či rotace podle os. Všechny vlastnosti tohoto komponentu jsou uvedeny na obrázku níže.



Obrázek 7 - Sprite [13]

Unity také poskytuje nástroj **Sprite Editor,** jenž slouží k úpravě jednotlivých obrázků a nabízí možnost vyříznutí několika na sobě nezávislých spritů z jednoho většího obrázku. [13]

Fyzika

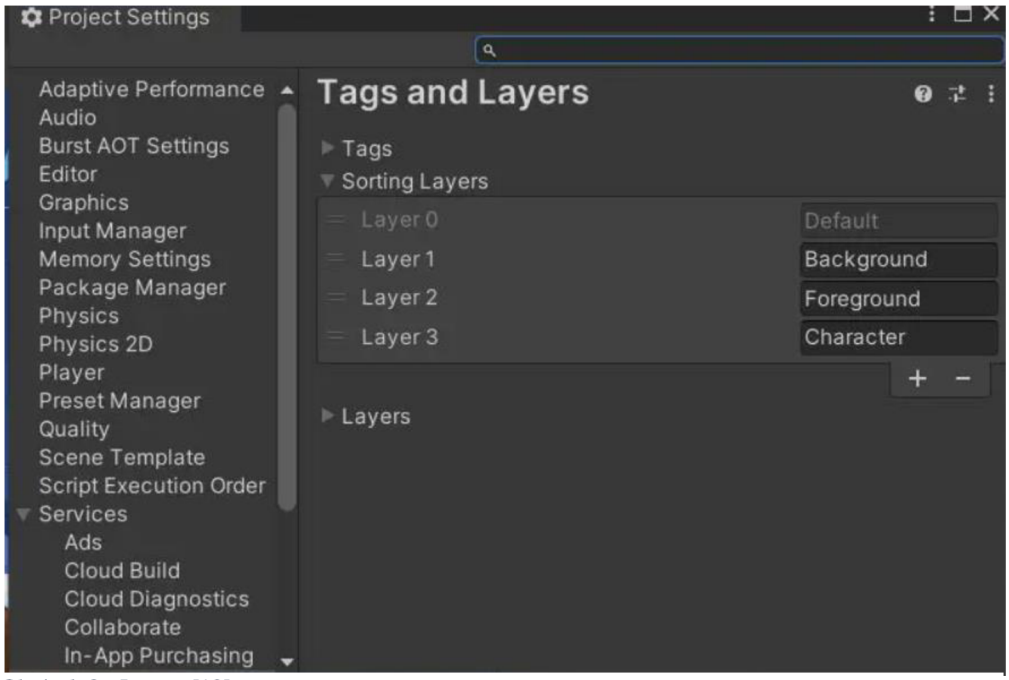
Fyzika je ve 2D hrách oproti 3D hrám velmi zjednodušená, protože Unity používá odlišný systém, aby dosáhl maximální optimalizace pro 2D hry. Komponenty jako Box Collider, které se běžně používají ve 3D hrách mají svojí zjednodušenou verzi, která se zpravidla jmenuje stejně s příponou 2D na konci (Box Collider2D).

Řazení

Objekty se na obrazovce zobrazují v závislosti na vzdálenosti od kamery, což funguje bez problémů ve 3D hrách. Avšak ve 2D hrách jsou všechny objekty stejně vzdálené, což způsobuje problém vzájemného překrývání. Proto Unity poskytuje několik způsobů, jak tento problém řešit.

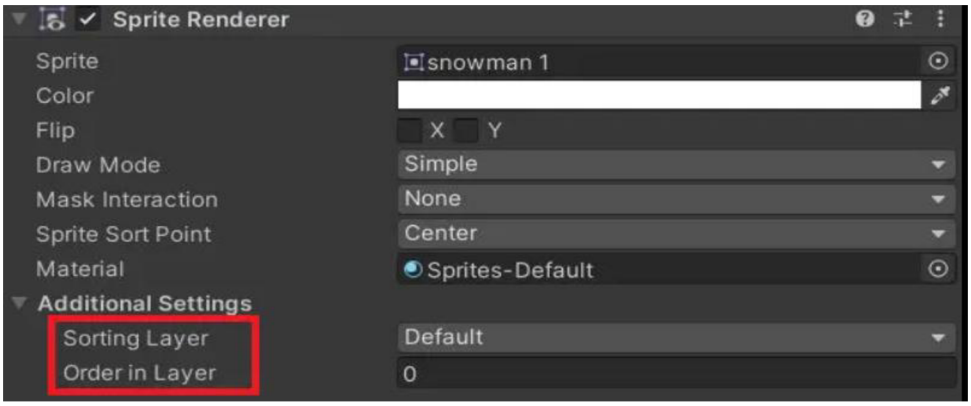
Sorting Layer

Prvním způsobem jsou Sorting Layers neboli hladiny, které říkají v jakém pořadí se budou zobrazovat jednotlivé objekty ve scéně. V základu se všechny objekty dávají na stejnou vrstvu Default, kde Unity nabízí možnost nové vrstvy přidávat a starší odebírat. Při vytváření těchto hladin závisí na jejich pořadí, vrstva s nejmenším číslem se zobrazuje jako první a další se na ní postupně přidávají. Pro lepší představu obrázek 8 ukazuje případ, kdy se nejdříve zobrazí všechny objekty, které mají přiřazenou vrstvu Background, nad ní se zobrazí vrstva Foreground a úplně nahoře vrstva s objekty Character. [14]



Obrázek 8 - Layers [19]

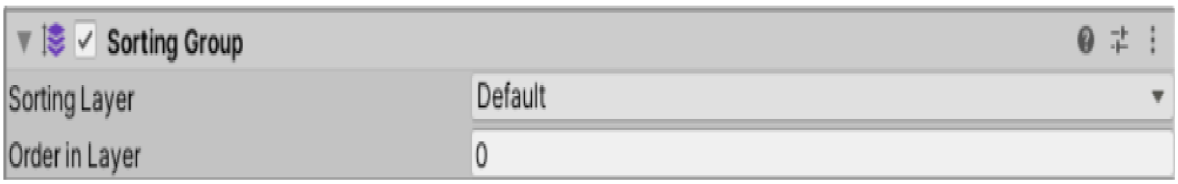
Každý objekt zobrazený na scéně musí mít připnutý komponent Sprite Renderer, který nabízí dvě možnosti dalšího nastavení: Sorting Layer a Order in Layer. Sorting Layer slouží k přiřazení vrstvy, jež byla vytvořena dříve v nastavení projektu. Dále nabízí možnost nastavit pořadí daného objektu v rámci zvolené vrstvy pomocí Order in Layer. Toto nastavení je zobrazeno na obrázku níže.



Obrázek 9 - Order in Layer [19]

Sorting Groups

Je komponent, který nabízí možnost spravovat pořadí zobrazení několika sprites najednou. Nejčastěji se používá na objektech jako je postava hráče nebo postava protivníka, které se skládají z velkému počtu spritů a bylo by obtížné až přímo nemožné řešit jejich vzájemné překrývání. Přidáním komponentu Sorting Group do rodičovského objektu se všechno řazení potomků spojí do jednoho. Toto spojení zařídí, že všechno řazení v rámci objektu se řeší pouze v tomto objektu nezávisle na okolí a o řazení celého objektu ve scéně se postará Sorting Group komponent. Zobrazení této komponenty je uvedeno na obrázku níže. [15]



Obrázek 10 - Sorting group [15]

Specify Render Queue

Dalším způsobem, jak řešit řazení objektů na scéně je pomocí vykreslovacích řad. Zatímco Sorting Layers jsou vyvinuty zásadně pro sprites, může nastat případ dvou spritů s odlišnými materiály, kde je potřeba extra řazení. V tento moment se hodí specifikovat řadu vykreslování. Jednotlivé typy řad se dají nastavit v inspektoru objektu v sekci nastavení materiálu. Unity nabízí 5 základních řad:

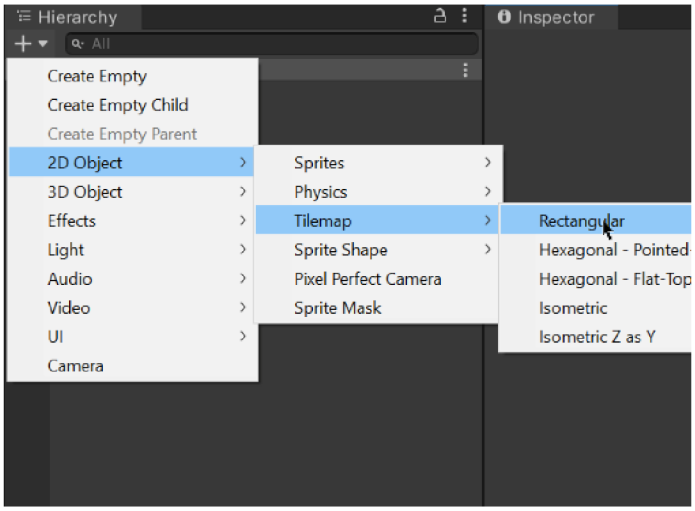
- Background
- Geometry
- Alpha Test
- Transparent
- Overlay [16]

Vzdálenost od kamery

Většina 2D her, používá řazení pomocí výše zmíněných postupů, ale existuje několik výjimek, kdy je jednodušší a lepší používat řazení pomocí vzdálenosti od kamery. Například hra s vlastní perspektivou, kde objekty umístěné na horní části obrazovky jsou menší než objekty umístěné dole na obrazovce. Tohle řazení se provádí na základně nastavení kamery, která má dva typy: **Perspektivní** a **Ortografický**.

Tilemap

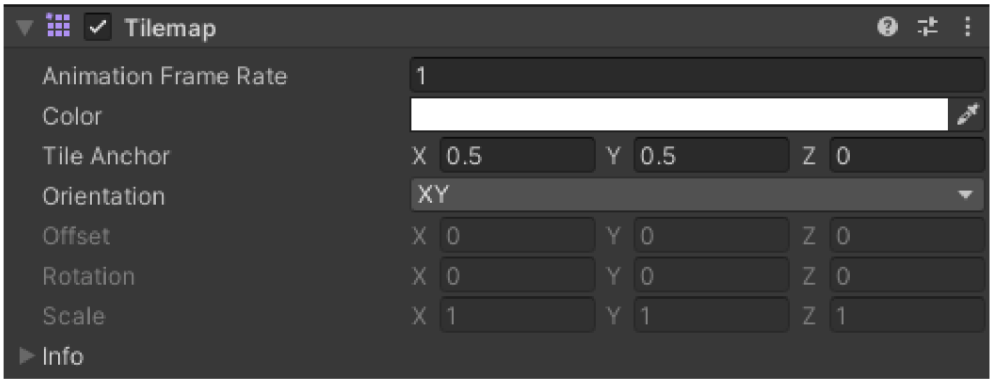
Jedná se o systém pro vytváření 2D levelů ve hrách. Vytváření komplexních levelů ve 2D hrách je pomocí Tilemaps velmi jednoduchá a rychlá záležitost. Při vytváření tilemapy je na výběr z několika druhů, které jsou uvedeny na obrázku níže.



Obrázek 11 - Tilemap (vlastní zpracování)

Po vytvoření vznikne objekt, který má k sobě připnutý komponent Tilemap, jenž nabízí spoustu možností nastavení, které jsou uvedeny na obrázku č. 12.

- Amination Frame Rate – mění rychlost animací animovaných dlaždic
- Color – dovoluje nastavit barvu všech dlaždic v Tilemap
- Tile Anchor – určuje střed dlaždice
- Orientation – určuje směr otočení dlaždic [17]

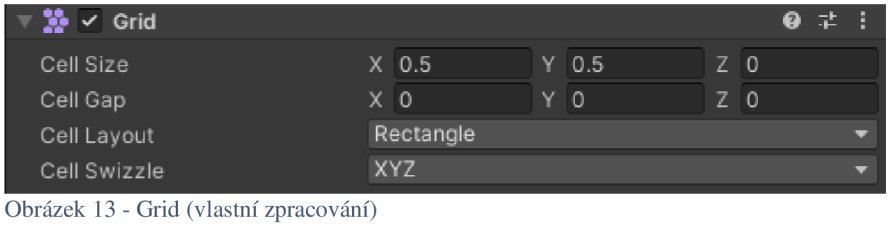


Obrázek 12 - Tilemap komponent (vlastní zpracování)

K použití Tilemaps je potřeba také několik dalších nástrojů jako je Grid, Tiles, Tile Palette a pro řešení kolizí Tileamp Collider 2D.

Grid

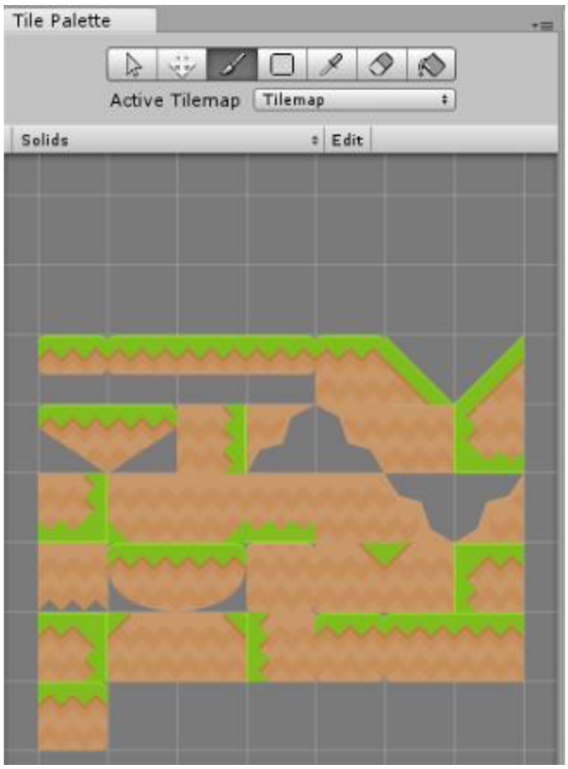
Mřížka, která se automaticky vytvoří při přidání Tilemap do scény. Poskytuje možnost nastavit velikost jednotlivých buněk a také mezery mezi nimi. Toto nastavení by mělo být prováděno s opatrností, protože každá změna se promítne do všech tilemaps. Ve scéně je zpravidla pouze jeden Grid ale ten může obsahovat několik Tilemaps. Zobrazení této komponenty v inspektoru je uvedeno níže.



Obrázek 13 - Grid (vlastní zpracování)

Tile Palette

Jedná se o panel, který umožňuje organizovat a rychle přistupovat k sadám dlaždic, které se používají při tvorbě hry. S touto paletou se dají snadno přetahovat dlaždice ze sad na tilemapu a rychle vytvářet složité struktury a mapy. Tento nástroj také umožňuje definovat různé varianty dlaždic pro různé situace, což přidává flexibilitu do tvorby herního prostředí. Ukázka tohoto nástroje je uvedena níže. [17]



Obrázek 14 - Tile Palette [17]

3.2 Porovnání

Dle Vohery a kolektivu [21] je Unity ideální volbou pro začátečníky díky své dobře strukturované dokumentaci a intuitivnímu uživatelskému rozhraní. Unreal Engine je rovněž vysoce hodnocen, ale je navržen spíše pro zkušenější uživatele kvůli jeho pokročilým grafickým funkcím. Naopak Cry Engine zaostává především kvůli nedostatečné dokumentaci a návodům, což může představovat výzvu pro uživatele při jeho používání.

Podle jiného článku od autorů Sharifa a Ameena [22] se jako nejlepší volby objevují Unity, Unreal a Cry Engine. Následuje je Godot, který za nimi pozůstává kvůli horším funkcím, které poskytuje. Porovnání bylo prováděno v několika kategoriích, včetně funkcí enginu, uživatelské přístupnosti a rozlišení.

V následující tabulce je uveden subjektivní porovnání autora podle předností a nedostatků jednotlivých enginů z předchozích kapitol, kde hodnota pět značí nejlepší a hodnota jedna nejhorší. Z tabulky vychází Unreal Engine jako nejlepší volba. Avšak pro tvorbu jednodušších her nejsou potřeba žádné speciální funkce, tedy pro jednoduší hry je Unreal engine srovnatelný s Unity. Autor se rozhodl pro Unity z osobní preference a také kvůli předchozím zkušenostem s tímto enginem.

Parametr	Unreal Engine	Godot	CryEngine	Amazon Lumberyard	Unity
Podporované platformy	5	4	5	4	4
Dokumentace	5	4	2	2	5
Grafický výkon	5	3	5	3	3
Komunita a podpora	5	4	2	2	5
Rozvoj a aktualizace	5	4	2	2	5

Tabulka 1 - porovnání enginů

4 Vlastní práce

Arena Survival je 2D roguelike mobilní hra pro zařízení android s jednoduchým ovládáním. Hráč ovládá postavu ducha, který se snaží přežít v aréně pět minut. Během tohoto času se kolem hráče objevují nepřátelé, který musí přemoci pomocí kouzel. Ty si postupně během hry vybírá a vylepšuje. Po skončení hry je hráč odměněn mincemi, pomocí nichž si může vylepšovat jednotlivé schopnosti, díky čemuž je schopen pokořit těžší obtížnosti této hry.

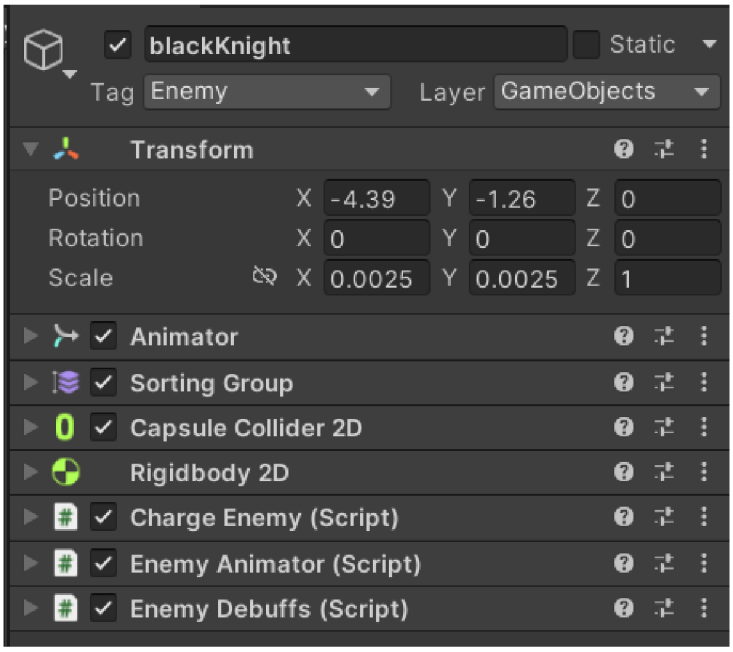
4.1 Implementace hry

V této kapitole se podrobně představí implementace všech herních prvků, které se v této hře vyskytují.

4.1.1 Nepřátelé

Hra obsahuje celkem čtyři modely nepřátel, jejich sprity byly zakoupeny z Asset Storu od uživatele „Pixel Cattle Games“. [20] Přidání jednotlivých balíčků do projektu se věnovala kapitola 3.1.5.5 v teoretické části.

Každému protivníkovi byl vytvořen prefab, který obsahuje komponenty uvedené na obrázku 15.

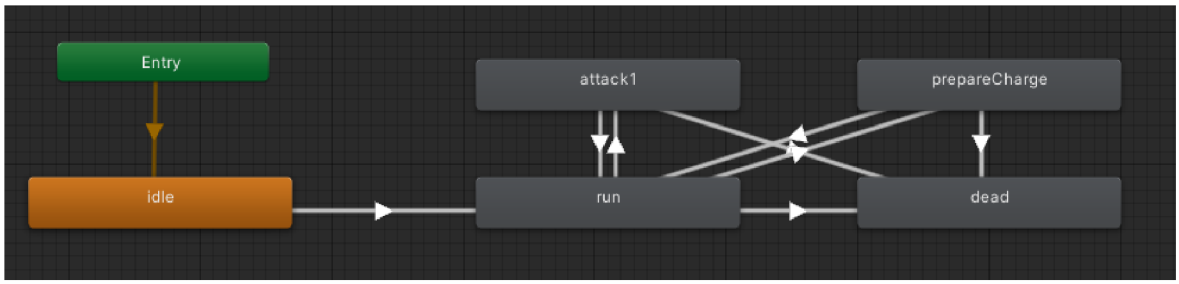


Obrázek 15 - Enemy prefab (vlastní zpracování)

Prefab nepřítele obsahuje Animátor a Sorting Group, který řídí animace a správné zobrazování jednotlivých spritů. Capsule Collider 2D a Rigidbody2D, se starají o detekci kolizí s hráčem, důkladný popis těchto komponent byl proveden v teoretické části. Dále obsahuje tři scripty, který řídí správné chování protivníka. Všechny druhy nepřátel obsahují script Enemy Animator a Enemy Debuffs. Script Charge Enemy je specifický pro blackKnight protivníka.

4.1.1.1 Animace

Každý základní nepřítel obsahuje komponentu Animator, která má tři základní stavy Run, Attack a Dead. Pokud se ale jedná o nepřítele schopného se na hráče rozběhnout, tak ten obsahuje ještě navíc stav prepareCharge.



Obrázek 16 - Animator nepřátele (vlastní zpracování)

Přechod mezi těmito stavy, řídí script EnemyAnimator, kde je funkce HandleAnimation, pomocí které se jednotlivé stavy přepínají v závislosti na tom, co právě daný nepřítel dělá.

```
private void HandleAnimation(){
  if (enemy.GetState() == Enemy.State.ChasePlayer)
  {
      animator.SetBool(IS_ATTACKING, false);
      animator.SetBool(IS_WALKING, true);
  }
}
```

4.1.1.2 Druhy nepřátel

Hra obsahuje dva druhy nepřátel BasicEnemy a ChargeEnemy. Všechny scripty těchto nepřátel dědí z hlavního scriptu Enemy, který obsahuje funkce a proměnné, které se používají napříč všemi druhy protivníků.

Enemy script obsahuje:

- Enum State, který obsahuje všechny stavy, ve kterých se daný nepřítel může nacházet
- Abstraktní metodu Act, jež se musí implementovat v každém potomku a řídí chovaní určitého nepřítele
- Funkci na pohyb k hráči MoveToPlayer, která mimo jiné otáčí nepřítele podle toho, jakým směrem jde
- Funkci CheckDespawn, která odstraní nepřítele ze hry, pokud je od hráče vzdálený více jak 50 f, zavedení této funkce významně snížilo náročnost hry na zařízení
- Funkci TakeDamage, která se volá, když má daný nepřítel obdržet zranění
- Funkce Die, která se volá, pokud nepřítel zemře

Basic Enemy

Jedná se o jednoduchého nepřítele, který celou dobu pronásleduje hráče. Proto jeho Act metoda vypadá následovně:

```
protected override void Act() {
  switch (state)
  {
      case State.ChasePlayer:
          MoveToPlayer();
          break;
      case State.Attack:
          Attack();
          break;
  }
}
```

Přepínání mezi jednotlivými stavy se provádí pomocí kolizí. Při kolizí s hráčem se nastavuje stav na Attack a naopak při skončení kolize se stav zpátky nastaví na ChasePlayer, jak je uvedeno v příkladu níže.

```
private void OnTriggerEnter2D(Collider2D collider2D){
    GameObject gm = collider2D.gameObject;
    if(gm.CompareTag("Player")){
        state = State.Attack;
    }
}


private void OnTriggerExit2D(Collider2D collider2D) {
    GameObject gm = collider2D.gameObject;
    if(gm.CompareTag("Player")){
        state = State.ChasePlayer;
    }
}
```

Charge Enemy

Jedná se o složitější nepřátele, kteří se na určité vzdálenosti od hráče zastaví a začnou nabíjet jejich rozběh. Po nabití se rozběhnou na pozici, kde hráč byl, v okamžiku jejich nabíjení. Po tomto rozběhu se chovají stejně jako Basic Enemy. Jejich Act metoda vypadá následovně:

```
protected override void Act() {
    switch (state)
    {
        case State.ChasePlayer:
            if(InChargeRange()){
                state = State.PrepareToCharge;
            }else{
                MoveToPlayer();
            }
            break;
        case State.Attack:
            Attack();
            break;
        case State.PrepareToCharge:
            PrepareToCharge();
            break;
        case State.Charge:
            HandleChargeMovement();
            break;
    }
}
```

4.1.1.3 Enemy debuffs

Kvůli tomu, že existují čtyři typy kouzel (blesk, vítr, oheň a mráz), kterými může hráč zranit protivníky a udělit jim příslušný stav je nutné implementovat script, který řídí všechna tato omezení. Na začátku se v metodě Start vytvoří Dictionary, který spravuje jednotlivé stavy a jejich čas působení, inicializace tohoto slovníku je ukázána níže.

```
debuffs = new Dictionary<PlayerAttacks.TypeSpellElement, float>{
            {PlayerAttacks.TypeSpellElement.Frost, 0f},
            {PlayerAttacks.TypeSpellElement.Fire, 0f},
            {PlayerAttacks.TypeSpellElement.Wind, 0f},
            {PlayerAttacks.TypeSpellElement.Lightning, 0f},
        };
```

Dále se v tomto scriptu nachází funkce HandleDebuff, která se volá v metodě Update. Tato funkce při každém volání, přičte k aktivním omezení Time.deltaTime a pokud výsledná hodnota překročila konstantu „debuff duration" odstraní toto omezení z daného nepřítele, jak je uvedeno v příkladu níže pro ohnivé omezení:

```
        if(fireDmgDot != 0) debuffs[PlayerAttacks.TypeSpellElement.Fire] +=
      Time.deltaTime;
        if(debuffs[PlayerAttacks.TypeSpellElement.Fire] > DEBUFF_DURATION){
            fireDmgDot = 0;
            debuffs[PlayerAttacks.TypeSpellElement.Fire] = 0;
            StopCoroutine(fireDot);
        }
```

4.1.1.4 Enemy spawner

Jedná se o neviditelný objekt ve scéně, kterému je přiřazený script EnemySpawner, jehož úkolem je vytváření nepřátel, v závislosti na čase daného levelu a obtížnosti. Obsahuje několik základních proměnných, pomocí nichž protivníky tvoří. Proměnná timeUntilSpawn, říká, kolik času musí uplynout do vytvoření nové vlny nepřátel. currTimeUntiSpawn značí aktuální čas od minulé vlny. V Update metodě se volá funkce SpawnEnemies, která odečítá od proměnné currTimeUntiSpawn Time.deltaTime a pokud výsledná hodnota je menší než 0 vytvoří se nová vlna nepřátel náhodně umístěná okolo hráče pomocí funkce SpawnEnemiesAroundPlayer, jak je ukázáno níže:

```
    private void SpawnEnemiesAroundPlayer(int count){
        Vector3 center = player.transform.position;
        for (int i = 0; i < count; i++){
            float ang = i * (360/count);
            Vector3 pos = RandomCircle(center, distanceFromPlayer, ang);
            pos.x = pos.x + Random.Range(0, distanceFromPlayer/2);
            pos.y = pos.y + Random.Range(0, distanceFromPlayer/2);
            SpawnRandomEnemy(pos);

        }
    }
```

Aby bylo možné jednotlivé protivníky přidávat do hry, je nutné inicializovat jejich prefaby. Popis prefabu a jeho inicializace, byla uvedena v teoretické části 3.1.5.4. V konkrétním případě této hry, je inicializace docílena pomocí toho, že každý script poskytuje public funkci na vytvoření nepřátele, která je dostupná ze všech ostatní scriptů. Dále nabízí funkci Setup pro nastavení proměnných. Následující příklad je ukázka této funkce pro třídu BasicEnemy.

```
    public static BasicEnemy Create(Vector3 position, GameObject gm, int hp,
float attackSpeed, float speed, int dmg){
        Transform enemyTransform = Instantiate(gm, position,
Quaternion.identity).transform;
        BasicEnemy enemy = enemyTransform.GetComponent<BasicEnemy>();
        enemy.Setup(hp, attackSpeed, speed, dmg);
        return enemy;
    }

    private void Setup(int hp, float attackSpeed, float speed, int dmg){
        base.health = hp;
        base.attackSpeed = attackSpeed;
        base.moveSpeed = speed;
        base.dmg = dmg;
        timeToAttack = 0;
    }
```

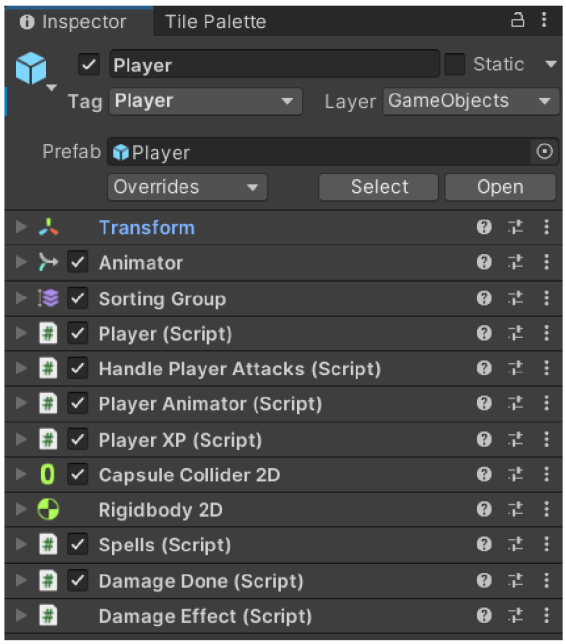
Přehled všech protivníků a jejich vlastností je uveden v tabulce 2.

NAME	HP	DMG	SPEED	ATACK SPEED	CHARGE RANGE
UndeadSkeleton	5	2	0,8	2	x
UndeadZombie	10	3	0,8	1,75	x
UndeadVampire	10	3	0,9	1,6	3
UndeadBlackKnight	30	5	1	3	3

Tabulka 2 - souhrn nepřátel

4.1.2 Hráč

Model hráče pochází ze stejného balíčku jako modely nepřátel. Jeho prefab obsahuje stejně jako ostatní prefaby komponenty pro správu animací a kolizí. Dále také obsahuje mnoho scriptů pro správu chování hráče, které jsou uvedeny na obrázku 17.



Obrázek 17 - komponenty hráče (vlastní zpracování)

4.1.2.1 Player Script

Spravuje všechny funkce, které jsou nutné pro fungování i nejjednoduššího hráče jako jsou například funkce pro pohyb, počet životů a řešení kolizí. Hráč ovládá svou postavu pomocí joysticku, pro jehož implementaci bylo použito už hotové řešení na Asset Storu a to Joystick Pack od Fenerax Studios. [25] Tato třída poskytuje input od uživatele, který se dále využil ve funkci HandleMovement pro pohybování modelu hráče, jak je uvedeno v příkladu níže:

```
    private Vector3 HandleMovement(){
        Vector2 inputVector = new Vector2(joystick.Horizontal,
joystick.Vertical).normalized;
        Vector3 moveDir = new Vector3(inputVector.x, inputVector.y, 0f);
        if(inputVector != Vector2.zero){
            isWalking = true;
            if(inputVector.x > 0){
                transform.eulerAngles = new Vector2(0, 0);
            }else{
                transform.eulerAngles = new Vector2(0, 180);
            }
        }else{
            isWalking = false;
        }
        return moveDir * moveSpeed * Time.deltaTime;
    }
```

Dále tento script poskytuje public metodu TakeDamage, která se volá ve scriptech nepřátel k udělení poškození hráči. Tato funkce nejenom odečte hodnotu zranění od hráčových životů, ale také zobrazí vizuální efekt zranění hráče na obrazovce (model hráče na chvíli zčervená) a zkontroluje, jestli hráčovi životy jsou vetší než 0, pokud ne zavolá metodu na zobrazení koncového menu.

```
    public void TakeDamage(int damage){
        Health -= damage;
        healthBar.SetHealth(Health);
        gameObject.GetComponent<DamageEffect>().ShowDamageEffect();
        if(Health <= 0){
            endMenu.SetActiveEndMenu(false);
        }
    }
```

Tento script také spravuje kolize hráče s objekty ve scéně. Pomocí porovnávání štítků jednotlivých objektů v kolizi se rozhoduje o řešení dané kolize. Existují tři objekty ve hře, který hráč může sebrat.

1. Zkušenost, která je reprezentována zelenou kuličkou na obrazovce, která se objeví po zabití nepřítele
2. Srdce, které hráči obnoví životy. Stejně jako zkušenost se objevuje po zabití nepřítele s tím rozdílem, že vytvoření tohoto objektu má pouze 2% pravděpodobnost
3. Mince, které se náhodně objevují v průběhu hry, může hráč využít k vylepšování své postavy

4.1.2.2 Player Animator

Hráčova postava má dvě animace (Idle, Walking). Tento script tedy není rozsáhlý a jeho funkcí je nastavovat komponentě Animator proměnou isWalking na true nebo false podle toho, jestli hráč je v pohybu nebo ne.

4.1.2.3 Player XP

Hráč sbírá v průběhu hry zkušenost, díky níž získá nový level a dostává možnost si vybrat nová kouzla. O správu tohoto procesu se stará právě tento script. Obsahuje několik funkcí, ze kterých je nejdůležitější public funkce SetPlayerXP, která připočte hodnotu z parametru ke stávající zkušenosti. A provede příslušné postupy na základě výsledku.

```
public void setPlayerXP(int amount){
    XP += amount;
    xpBar.SetXP(XP);
    if(XP >= XPForLevel){
        XP = XPForLevel - XP;
        XPForLevel += 5;
        playerLevel += 1;
        if(CheckIfPlayerCanLevelUp()) LevelUp();
    }
}
```

4.1.2.4 Damage Effect

Tento script slouží k zobrazení a varovaní hráče, že byl zraněn. Toho se docílí pomocí funkce StartCoroutine, kterou poskytuje třída MonoBehaviour. Tato funkce je schopná počkat určitý čas během vykonávání. Toho se využívá tak, že při zranění hráče se změní barva hráčovy postavy na červenou. Počká se jednu sekundu a po uplynutí této doby se barva hráče vrátí do normální podoby.

4.1.2.5 Spells

Hráč si během hry vybírá nová kouzla a jejich vylepšení. O to se stará právě tento script. Nejdříve se v metodě Start vytvoří dvě struktury List obsahující textové řetězce. „availableSpells“, ve kterém jsou uloženy všechna možná kouzla, která se lze naučit a list „learnedSpells“, jenž obsahuje vše, co se hráč již naučil. Tento script dále poskytuje public funci LearnNewSpell, která se volá v LevelUpMenu, po kliku hráče na nějaké tlačítko. Protože každé kouzlo má čtyři úrovně vylepšení a tato úroveň je zapsaná v názvu jednotlivého kouzla, je nutné ve funkci LearnNewSpell provést patřičné operace jako je

například odstraňování posledního znaku z názvu, pro správné vyhledávání mezi už naučenými kouzly. Celá funkce je uvedena níže:

```
    public void LearnNewSpell(string newSpellWithoutRank){
        string newSpell = GetSpellWithRank(newSpellWithoutRank);
        // remove rank
        string spellToLearn = newSpell.Substring(0,  newSpell.Length - 1);

        bool spellAdded = false;
        string removedSpell = "";
        // check if player already knows this spell (diffrent rank)
        for (int i = 0; i < learnedSpells.Count; i++)
        {
            string spell = learnedSpells[i];
            string spellToCheck = spell.Substring(0,  spell.Length - 1);

            if(spellToCheck == spellToLearn){
                //remove old one and add spell with bigger rank
                removedSpell = learnedSpells[i];
                learnedSpells.Remove(learnedSpells[i]);
                learnedSpells.Add(newSpell);
                spellAdded = true;
                break;
            }
        }
        // its new spell
        if(!spellAdded) learnedSpells.Add(newSpell);

        UpdateAvailableSpells(newSpell);
        transform.GetComponent<HandlePlayerAttacks>().AddSpellToSpellBook
(newSpell, removedSpell);
    }
```

4.1.2.6 Handle Player Attacks

Jak už z názvu vyplývá tento script se stará o spouštění všech útoků neboli kouzel, které má hráč k dispozici. Při spuštění tohoto scriptu se v metodě Start vytvoří List „spellBook“, který obsahuje třídu SpellInfo, ve které jsou všechny potřebné informace k práci s kouzly. Více o této třídě je v kapitole 4.1.3.

V Update metodě se volá funkce Attack, která projde všechny naučená kouzla ve for cyklu, každému kouzlu sníží dobu do vyčarování o hodnotu Time.deltaTime. Zároveň se podívá, jestli dané kouzlo nemá být vyčarováno. Pokud ano, tak ho vytvoří a vyresetuje dobu do vyčarování. Celá tato funkce je uvedena níže:

```
    private void Attack(){
        for (int i = 0; i < spellBook.Count; i++)
        {
            spellBook[i].TimeUntilCast -= Time.deltaTime;
            if(spellBook[i].TimeUntilCast < 0){
                if(!IsEnemy()) return;
                spellBook[i].SpawnSpell(transform.position);
                spellBook[i].TimeUntilCast = spellBook[i].CastTime * (1f -
(GameData.GetPlayerCdReduction() / 100f));
            }
        }
    }
```

4.1.2.7 Damage done

Poslední script, který je připnutý k hráči je DamageDone script, jenž se stará o zapisování všech zranění, které hráč během hry udělí. Tato tabulka se poté zobrazí na konci hry. Script na začátku vytvoří strukturu Dictionary, která obsahuje klíče (jména kouzel) a hodnotu zranění, které udělily. Dále poskytuje public metodu AddRows, která vezme hodnoty z Dictionary a přidá je jako řádky do tabulky, která se zobrazuje v konečném menu. Obrázek 18 této tabulky je uveden níže.

Obrázek 18 - DamageDone Menu (vlastní zpracování)

4.1.3 Kouzla

Ve hře jsou čtyři druhy elementů, kterých můžou kouzla nabývat, oheň, mráz, vítr a blesky. Jednotlivé typy jsou dostupné jako enum ve scriptu PlayerAttacks což je hlavní script, ze kterého dědí všechny ostatní scripty, jenž implementují jednotlivé typy kouzel.

Tato třída dále poskytuje funkce pro hledání nepřátel, které se využívají ke správnému pohyby daného kouzla. Ukázka jedné z funkcí je uvedena níže:

```
    protected GameObject GetNearestEnemyToPlayer(){
        arrayEnemies = GameObject.FindGameObjectsWithTag("Enemy");
        GameObject nearestEnemy = null;
        float nearestEnemyDistance = float.PositiveInfinity;
        foreach (GameObject gm in arrayEnemies)
        {
            float distance = Vector3.Distance(player.transform.position,
gm.transform.position);
            if(distance < nearestEnemyDistance){
                nearestEnemy = gm;
                nearestEnemyDistance = distance;
            }
        }
        return nearestEnemy;
    }
```

Jak už bylo napsáno výše existuje několik druhů kouzel, které se dělí podle toho, jak se dané kouzlo chová. V následujícím seznamu je přehled všech typů scriptů pro implementaci těchto kouzel.

- PlayerBasicAttack – jednoduché kouzlo, které letí na pozici nejbližšího nepřítele v době vyčarování tohohle kouzla
- PlayerChainAttack – jedná se o kouzlo, které nezmizí při kolizi s nepřítelem, ale vyhledá nejbližšího dalšího nepřítele vzhledem k pozici, kde se zrovna nachází a letí za ním
- PlayerWaveAttack – kouzlo, které se chová jako PlayerBasicAttack ale na rozdíl od něho nezmizí při první kolizi a pokračuje dál ve směru vystřelení
- PlayerStaticPointAttack – typ kouzla, které se objeví na náhodném místě okolo hráče a zůstane na něm určitý čas
- PlayerRotationAttack – jedná se o kouzla, která rotují okolo hráče a nikdy nezmizí
- Explosion – kouzlo, které se objeví na určitém místě a hned zmizí
- PlayerExplosionAttack – má podobné chování jako obyčejné kouzlo s tím rozdílem, že při kolizi s nepřítelem vytvoří explozi

Popis všech kouzel a hodnot, které se vyskytují ve hře je nutné mít zaznamenaný a snadno přístupný, k tomu slouží třída SpellsData. Ta obsahuje třídu SpellInfo, která nabízí proměnné ohledně daného kouzla, jak je uvedeno na příkladu níže:

```
public class SpellInfo
{
    public string SpellName { get; set; }
    public string SpellDescription { get; set; }
    public float CastTime { get; set; }
    public float TimeUntilCast { get; set; }
    public Action<Vector3> SpawnSpell { get; set; }
}
```

Dále tato třída SpellsData poskytuje public funkci, která vrátí Dictionary <string, SpellInfo> obsahující všechny kouzla ve hře. Přehled všech kouzel je uveden v příloze č. 1.

4.1.4 UI

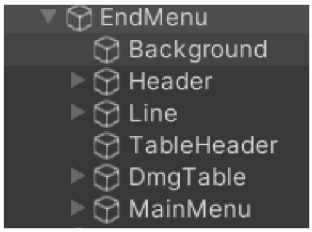
Jedná se o všechny většinou statické prvky, které jsou vidět na obrazovce, patří mezi ně životy hráče, zkušenost anebo čas od startu dané hry. UI element se vytvoří pravým klikem na okno Hierarchie a vybráním UI kategorie. Zde je přehled všech dostupných prvků.

V tomto projektu byly použity následují prvky. Slider, pro zobrazení hráčových životů a pro zobrazení získaných zkušeností bodů. Dále bylo použito tlačítko pro zastavení hry a také element Text – TextMeshPro, který zobrazuje aktuální čas od startu daného levelu. Všechny tyto prvky jsou zobrazeny na obrázku 19.

Obrázek 19 - UI ve hře (vlastní zpracování)

4.1.4.1 Menu

Každé menu ve hře je tvořeno jako UI element, který je zpravidla vypnuté a v době potřeby se mu zapne viditelnost. Struktura všech menu je podobná, skládají se z elementu Background, které ztmaví hru na pozadí. Dále obsahují nadpis, pod nímž se nachází obsah daného menu a na spodku jsou tlačítka korespondující s obsahem. Ukázka takového menu v hierarchii je uvedena na obrázku 20.



Obrázek 20 – EndMenu (vlastní zpracování)

Ve hře je celkově sedm menu, přičemž každé má svůj specifický script, který řídí jeho chování. Ve hře se vyskytují tato menu:

1. MainMenu – zobrazí se po spuštění hry
2. OptionsMenu – dostupné z hlavního menu
3. DifficultyMenu – před spuštění hry se hráči objeví toto menu
4. UpgradesMenu – obsahuje všechny dostupné vylepšení
5. PauseMenu – zobrazí se, když hráč zastaví hru
6. LevelUpMenu – při přechodu na novou úroveň se hráči zobrazí toto menu
7. EndMenu – objeví se na konci hry

4.1.5 Ukládací systém

Kvůli tomu, že hráč má možnost sbírat mince a následně je používat pro vylepšování své postavy je nutné, aby se tyto informace uchovávaly i po vypnutí hry. K tomu byla použita třída PlayerPrefs, kterou poskytuje Unity. Tato třída je schopná ukládat tři typy proměnných: int, float a string.

Pro implementaci toho systému byla vytvořena obyčejná static třída GameData, která obsahuje všechny proměnné, které je potřeba uložit a jejich get a set funkce, které jsou dostupné ze všech ostatních scriptů. Dále tato třída obsahuje konstruktor, který nastaví všechny proměnné pomocí funkcí, které poskytuje třída PlayerPrefs. První parametr je

zpravidla klíč uložené proměnné a druhý je základní hodnota, která se nastaví, pokud tato hodnota ještě nikdy nebyla uložena. Ukázka celého konstruktoru je uvedena níže.

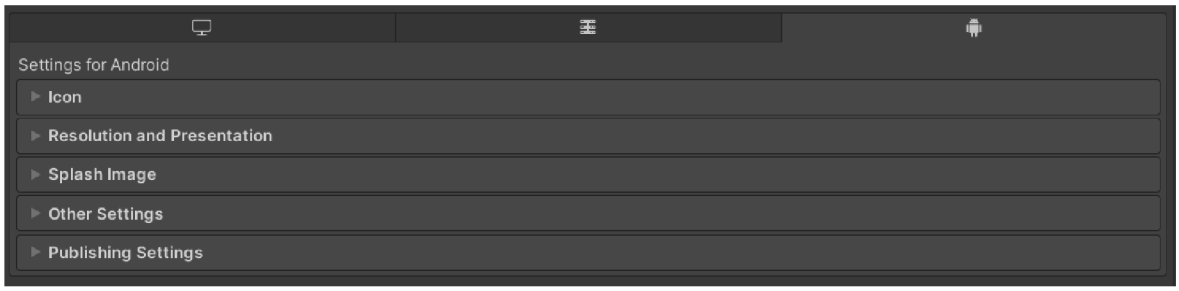
```
static GameData(){
    maxDiffLevel = PlayerPrefs.GetInt("maxDiffLevel", 1);
    currDiffLevel = PlayerPrefs.GetInt("currDiffLevel", 1);
    playerCritChange = PlayerPrefs.GetInt("playerCritChange", 0);
    playerBonusHp = PlayerPrefs.GetInt("playerBonusHp", 0);
    playerBonusDmg = PlayerPrefs.GetInt("playerBonusDmg", 0);
    playerBonusSpeed = PlayerPrefs.GetInt("playerBonusSpeed", 0);
    playerCdReduction = PlayerPrefs.GetInt("playerCdReduction", 0);
    coins = PlayerPrefs.GetInt("coins", 0);
}
```

4.2 Publikace

Hra byla úspěšně publikovaná na platformě Google Play. V následujících kapitolách je podrobně popsáno, jak probíhala publikace. To zahrnuje, jak provést nastavení a vyplnění potřebných údajů v Unity pro správné sestavení aplikace do spustitelné podoby a také v neposlední řadě, jak si založit účet pro publikaci a vyplnění všech nezbytných informací v obchodě Google Play.

4.2.1 Build

Předtím než se může aplikace zkompilovat je důležité vyplnit všechna potřebná nastavení. V Project Settings je záložka Player, která obsahuje několik kategorií, které jsou uvedeny na obrázku 21.



Obrázek 21 - Nastavení projektu, seznam kategorií (vlastní zpracování)

V kategorii Icon, je nutné nahrát ikonu, která se bude zobrazovat jako ikonka hry. Další dvě kategorie zpravidla mají už automaticky nastavené vše potřebné, takže je není potřeba nijak měnit. Naopak kategorie Other Settings, obsahuje spoustu nastavení, mezi

nimž se nachází sekce Identification, ve které je nutné nastavit verzi dané aplikace a minimum API level, který určuje, jak starý telefon si může spustit tuto hru. Příklad tohoto nastavení je uveden níže.

Obrázek 22 - příklad nastavení Identification (vlastní zpracování)

Poslední kategorií je Publishing Settings, kde je nutné buď nahrát už vytvořený Keystore anebo si nechat nový vygenerovat. Tento klíč je nutný mít pro nahrávání aplikací na Google Play nebo App Store, protože slouží k podpisu aplikace. Klíč je jedinečný a nelze jej znovu vytvořit. Při ztrátě klíče již nejde danou aplikaci aktualizovat.

Potom co je nastavení vyplněné je na čase aplikaci zkompilovat. Po otevření okna Build Settings, je nutno vybrat platformu v případě této hry Android. Skoro vše je tam automaticky předvyplněné, zbývá jen zakliknout položku Build App Bundle (Google Play) a nastavit Create symbols.zip na Public. Potom, co se klikne na tlačítko build se vytvoří na předem určeném místě soubor s příponou aab. Tento soubor je nutný nahrát do Google Play Console. Postup vytvoření účtu a nahrání aplikace je uveden v další kapitole. [28]

4.2.2 Google play console

Pro publikování hry na obchodě Google Play je nutné si nejdříve založit google play developer účet. Po zaplacení poplatku za vytvoření účtu dostane tvůrce přístup do console, kde může spravovat všechny své hry a přidávat nové.

Kliknutí na tlačítko „Create App“ se otevře krátký dotazník, kde je potřeba vyplnit jméno aplikace. Dále zakliknout, zda se jedná o aplikaci nebo hru a jestli tato daná aplikace je zdarma nebo placená. Po tomto procesu se daná hra přidá do console, kde je nutné nastavit základní informace ohledně dané hry.

1. Set privacy policy – zde se musí umístit link, na stránku obsahující zásady ochrany osobních údajů, které si můžeme nechat vygenerovat pomocí online generátoru
2. App access – informace o tom, jestli je v dané aplikaci nějaký způsob přihlášení
3. Ads – zde se vyplní, jestli daná aplikace obsahuje reklamy
4. Content ratings – obsahuje dotazník ohledně toho, co se nachází v dané hře
5. Target audience – zde se uvede pro jaké publikum je hra určená
6. News app – zde se pouze zaklikne, jestli se jedná o aplikaci se zprávami
7. COVID-19 – jednoduchá otázka ohledně toho, jestli aplikace obsahuje něco souvisejícího s covidem
8. Data safety – dlouhý dotazník ohledně toho jaké data aplikace sbírá
9. Government apps – zde se určuje, jestli aplikace je či není vládní

Po dokončení již zbývá jen nastavit, jak se vytvořená hra bude zobrazovat v obchodě. Mezi to patří jméno hry, krátký i dlouhý popis a výběr štítků. V neposlední řadě je potřeba nahrát ikonu dané hry a snímky zachycující hru, která se bude zobrazovat v obchodě.

4.2.2.1 Testování

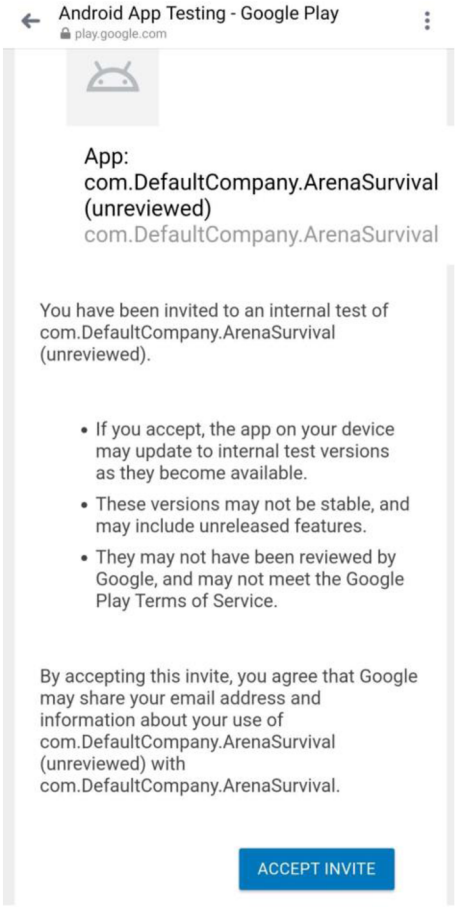
Testování her je rozsáhlým a klíčovým prvkem ve vývoji her. V dnešní době manuální testování hratelnosti her, které se také nazývá play-testing, je hlavní testovací technikou používanou herními vývojáři. Automatizace testování her je obtížná až nemožná činnost, což vede k tomu, že vývojáři najímají specializované testery k manuálnímu testování jejich her. Tito lidští testeři pak zejména hodnotí „fun factor“ a „game balance“ jednotlivých her. [30] Další způsob získávání zpětné vazby je přímo od uživatelů pomocí přímé diskuze, z uživatelského hodnocení, fóra vývojářů her či herní výstavy. Kde se na konci testování sjednotí všechny získané připomínky, které se předají vývojářům a slouží tak jako základ pro vylepšení hry v dalším produkčním cyklu. [31]

Při testování této hry se využívaly logy, neboli zaznamenávání důležitých událostí ve hře do herní konzole, aby bylo možné zpětně analyzovat vzniklé chyby během testování. S testery probíhala komunikace online anebo přímo osobně. Ve zpětné vazbě popsali problémy, které se vyskytly během hraní hry. Nastavení "game balance" bylo realizováno úpravami vlastností nepřátel a kouzel, s cílem dosáhnout co nejvyváženějšího stavu, podle subjektivního hodnocení autora.

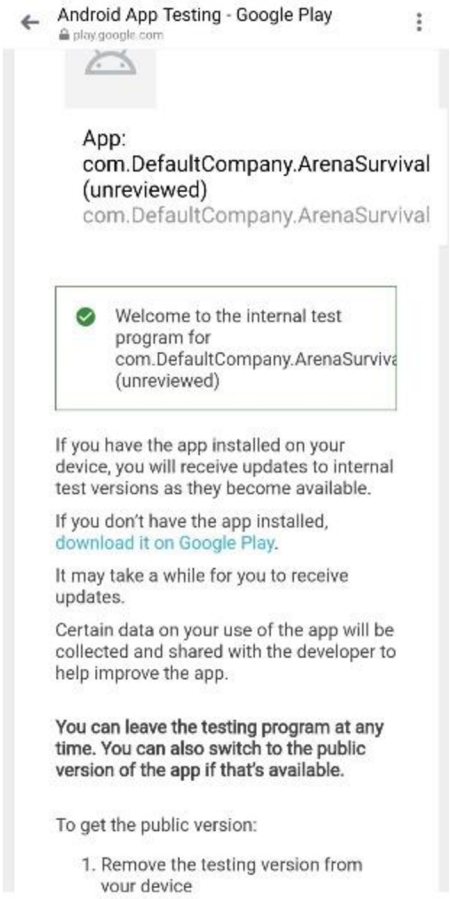
Google play console poskytuje nástroje pro testování her před publikací pro veřejnost.

V záložce Internal testing je nutno nejdříve přidat testery, kteří budou hru testovat. Po přidání jejich google emailů je možné kliknout na tlačítko „Create new release“. Zde se musí přidat zkompilovaná hra a vyplnit políčko s Release notes.

Potom, co byla hra přidána k testování se v záložce Testers, nachází link, přes který si testeři mohou hru stáhnout. Po kliknutí na tento odkaz se zobrazí okno, kde je nutné přijmout pozvánku k testování viz obrázek 24.



Obrázek 24 - pozvánka k testovaní 1 (vlastní zpracování)



Obrázek 23 - pozvánka k testování 2 (vlastní zpracování)

Po přijmutí pozvánky se zobrazí další okno, které už obsahuje odkaz přímo do obchodu, kde se hra dá stáhnout. Toto okno je zobrazeno výše na obrázku 23.

Během uživatelského testování, do kterého byli zapojeni jak zkušení hráči, tak úplní nováčci bylo zjištěno několik zásadních problémů, které bylo potřeba vyřešit.

První problém, na který uživatelé narazili, spočíval v optimalizaci hry, kdy se ve hře vyskytovaly stovky nepřátel. Uživatel hrál hru neobvyklým způsobem a tím, že utíkal před nepřáteli a nesnažil se s nimi bojovat. To mělo za následek velký počet objektů ve hře, a

proto bylo nutné zavést funkci ve třídě Enemy, která odstraňuje nepřátele ze hry, pokud se nachází ve velké vzdálenosti od hráče. Tato funkce je uvedena v příkladu níže.

```
    private void CheckDespawn(){
        float distance = Vector3.Distance(player.transform.position,
gameObject.transform.position);
        if(distance > 50f){
            Destroy(gameObject);
        }
    }
```

Další problém se týkal výsledného menu, které zobrazuje poškození udělené hráčem určitým kouzlem nepřátelům. Zde se vyskytla chyba s kouzly, které mají podle úrovně vylepšení různé chování, a proto se používají různé prefaby pro určité úrovně vylepšení. Zde bylo nutné zavést jednoduchou funkci, která sjednocovala tyto úrovně vylepšení do jednoho názvu, pod kterým se dané kouzlo zobrazilo ve výsledné tabulce.

Posledním zásadním problémem, na který bylo uživateli poukázáno, bylo neobjevování se léčivých srdcí ve hře. Tento problém spočíval v tom, že bylo nastaveno, že ve hře může být jenom pouze určitý počet těchto srdcí najednou. Uživatel je tedy musel přehlídnout a nesebrat je, a to způsobilo neobjevování dalších těchto srdcí. Tento problém byl vyřešen přidáním funkce na odstranění těchto objektů ze hry, při určité vzdálenosti od hráče.

V neposlední řadě bylo zjištěno pár menších problémů týkajících se hlavně uživatelského rozhraní, jako například překlepy v popisu kouzel.

Po vyřešení všech zjištěných problému bylo možné tuto verzi hry v záložce Production publikovat pro veřejnost. Toto publikování ale trvá několik dní, protože během nich probíhá kontrola hry.

5 Výsledky a diskuse

Konečný prototyp obsahuje všechny plánované funkce, které byly na začátku stanovené. Na začátku tvorby práce měl autor základní znalosti s tvořením her pomocí herního enginu Unity.

Během implementace hry se vyskytlo pár problémů, které se povedlo vyřešit pomocí návodů a podrobné studie Unity dokumentace. Jedna z největších komplikací bylo navržení systému pro všechna kouzla vyskytující se ve hře s důrazem na znovu použitelnost a možnost rozšíření. Tento problém se nakonec vyřešil pomocí implementace struktury Dictionary.

Další problémy, se kterými se autor setkal bylo správné navržení rychlosti a počtu objevování nepřátel v závislosti na obtížnosti, kterou si hráč vybere na začátku hry. Tento problém byl řešen prostřednictvím testování, kdy se postupně zvětšoval počet nepřátel do té doby, dokud hra nebyla možná dokončit.

Poslední problém, se kterým se setká každý vývojář mobilních her je správné rozlišení a navržení herních menu pro různé typy mobilních zařízení a tabletů. Tento problém byl řešen pomocí funkce Device Simulator, kterou Unity nabízí k testování hry na různých typech obrazovek.

Jak bylo zmíněno výše během vývoje hry se klad důraz na možnosti rozšíření, a proto jsou veškeré scripty napsány tak, že přidání nových věcí, jako například kouzel nebo protivníků, je snadná záležitost jak z hlediska logiky, tak správného fungování.

Tato práce poskytuje návod pro čtenáře ať už úplné začátečníky nebo pro pokročilé, jak vytvořit 2D mobilní hru pomocí herního enginu Unity. Během vývoje hry se autor naučil základy vývoje mobilní hry i s její publikací na Google Play, kde je možné ji najít pod jménem Arena Survival. Dále si autor také rozšířil už dosavadní dovednosti s Unity a programovacím jazykem c#.

6 Závěr

Cílem této bakalářské práce bylo nejen vytvořit mobilní hru využívající herní engine Unity, ale také podrobně popsat celý proces vývoje her.

Teoretická část práce se věnovala důkladnému zkoumání herních enginů a srovnání jejich vlastností. Dále bylo podrobně popsány komponenty a nástroje, které Unity nabízí, což umožní lepší porozumění tomu, jak tento engine funguje a jakým způsobem je možné vytvářet a spravovat herní svět.

Praktická část práce popisovala vývoj hry prostřednictvím Unity a všech scriptů. Nejdříve byl popsán vývoj nepřátel, jejich druhů a implementace jejich AI. Následně byla podrobně rozebrána samotná postava hráče, kde byly představeny a detailně popsány veškeré scripty, které tvořily hráčovu postavu. Důraz byl také kladen na vysvětlení fungování systému kouzel a jejich různých typů. V neposlední řadě byly také představeny všechny menu v rámci hry a způsoby jejich spouštění a zobrazení, což přispělo k celkovému pochopení uživatelského rozhraní.

Nakonec byl podrobně popsán postup publikování hry na platformě Google Play od úplného začátku. Čtenář byl seznámen s postupem kompilace dané hry do spustitelné podoby a následně s procesem založení vývojářského účtu v Google Play. Na závěr bylo představeno testování ať už samotným autorem nebo testery, kteří byli pozváni pomocí platformy Google Play.

Výsledkem této práce je mobilní hra s názvem "Arena Survival", která je určena pro zařízení s operačním systémem Android. Arena Survival je 2D roguelike mobilní hra s jednoduchým ovládáním. Hráč ovládá postavu ducha, jehož cílem je přežít v aréně pět minut. Během tohoto času se kolem hráče zjevují nepřátelé, který musí přemoci pomocí kouzel. Ty si postupně během hry vybírá a vylepšuje. Po skončení hry je hráč odměněn mincemi, pomocí nichž si může vylepšovat jednotlivé schopnosti, díky čemuž je schopen pokořit těžší obtížnosti této hry.

Tato hra byla důkladně testovaná testery a na základě jejich připomínek byla upravena. Následně byla hra úspěšně publikována na platformě Google Play a je plně funkční.

7 Seznam použitých zdrojů

1. What is a Game Engine? [online]. 2021 [cit. 2023-09-07]. Dostupné z: https://fullscale.io/blog/what-is-game-engine/
2. What Are the Best Game Engines? [online]. 2023 [cit. 2023-09-07]. Dostupné z: https://www.perforce.com/blog/vcs/most-popular-game-engines
3. Fireship, 2022, Godot in 100 Seconds, YouTube video. [cit. 2023-09-07]. Dostupné z: https://www.youtube.com/watch?v=QKgTZWbwD1U
4. Amazon Lumberyard [online]. 2023 [cit. 2023-09-07]. Dostupné z: https://www.incredibuild.com/integrations/amazon-lumberyard
5. What is Unity? – A Top Game Engine Solution for Video Games [online]. 2023 [cit. 2023-09-17]. Dostupné z: https://gamedevacademy.org/what-is-unity/
6. Unity User Manual 2022.3 (LTS) [online]. Unity Technologies, 2023 [cit. 2023-09-17]. Dostupné z: https://docs.unity3d.com/Manual/index.html
7. HILTON, Chris. Collider's and Triggers in Unity — Understanding the Basics [online]. 2021 [cit. 2023-09-17]. Dostupné z: https://christopherhilton88.medium.com/colliders-and-triggers-in-unity-understanding-the-basics-7192714f3440
8. Unity - Rigidbodies and Physics [online]. [cit. 2023-09-20]. Dostupné z: https://www.tutorialspoint.com/unity/unity_rigidbodies_and_physics.htm
9. Animation and Animator in Unity 3D [online]. [cit. 2023-09-20]. Dostupné z: https://www.studytonight.com/game-development-in-2D/animation-animator
10. Brackeys, 2017, GIntroduction to AUDIO in Unity, YouTube video. [cit. 2023-09-20]. Dostupné z: https://www.youtube.com/watch?v=6OT43pvUyfY
11. FRENCH, John. How to use Prefabs in Unity [online]. [cit. 2023-09-20]. Dostupné z: https://gamedevbeginner.com/how-to-use-prefabs-in-unity/#how_to_use_prefabs
12. ELEMUWA, 2022, Fimber. The basics of lighting in Unity [online]. [cit. 2023-09-20]. Dostupné z: https://blog.logrocket.com/lighting-basics-unity/#directional-light
13. Unity - Creating Sprites [online]. [cit. 2023-10-05]. Dostupné z: https://www.tutorialspoint.com/unity/unity_creating_sprites.htm
14. Sorting Layer/Order Setting [online]. [cit. 2023-10-05]. Dostupné z: https://rainyrizzle.github.io/en/AdvancedManual/AD_SortingLayer.html

15. FRENCH, John. How to use Sorting Layers in Unity [online]. 2021 [cit. 2023-10-05]. Dostupné z: https://gamedevbeginner.com/how-to-use-sorting-layers-in-unity/
16. SCHRUTE, Ahmed. Rendering Queues in Unity [online]. 2020 [cit. 2023-10-05]. Dostupné z: https://www.linkedin.com/pulse/rendering-queues-unity-a-hassan
17. BONZON, Tim. How to Build 2D Levels in Unity – Tilemap Editor Tutorial [online]. 2023 [cit. 2023-10-05]. Dostupné z: https://gamedevacademy.org/mastering-unitys-new-tilemap-editor-building-2d-levels
18. HOLAN, Tomáš. Unity: první seznámení s tvorbou počítačových her. Praha: CZ.NIC, z.s.p.o., 2020. CZ.NIC. ISBN 978-80-88168-57-7.
19. VIONIX. Sorting Layer in Unity 2D [online]. 2023 [cit. 2023-10-14]. Dostupné z: https://vionixstudio.com/2023/07/06/sorting-layer-in-unity-2d/?utm_content=cmp-true
20. UNITY. *Unity Asset Store* [online]. [cit. 2024-02-11]. Dostupné z: https://assetstore.unity.com/publishers/23567
21. C. Vohera, H. Chheda, D. Chouhan, A. Desai and V. Jain, "Game Engine Architecture and Comparative Study of Different Game Engines," 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2021, pp. 1-6, doi: 10.1109/ICCCNT51525.2021.9579618.
22. K. H. Sharif and S. Yousif Ameen, "Game Engines Evaluation for Serious Game Development in Education," 2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Hvar, Croatia, 2021, pp. 1-6, doi: 10.23919/SoftCOM52868.2021.9559053.
23. S. Singh and A. Kaur, "Game Development using Unity Game Engine," 2022 3rd International Conference on Computing, Analytics and Networks (ICAN), Rajpura, Punjab, India, 2022, pp. 1-6, doi: 10.1109/ICAN56228.2022.10007155.
24. ÇAMÖNÜ, İSMAIL. *What is MonoBehaviour?* [online]. [cit. 2024-03-05]. Dostupné z: https://www.codinblack.com/what-is-monobehaviour
25. UNITY. Unity Asset Store [online]. [cit. 2024-03-05]. Dostupné z: https://assetstore.unity.com/publishers/32730
26. GOLDSTONE, Will. *Unity Game Development Essentials*. Packt Publishing, 2009. ISBN 1847198198, 9781847198198.

27. GOLDSTONE, Will. *Unity 3.x Game Development Essentials*. Přepracované vydání. Packt Publishing, 2011. ISBN 1849691452, 9781849691451.
28. ZHI ENG, Lee. *Building a Game with Unity and Blender*. Packt Publishing, 2015. ISBN 1785280740, 9781785280740.
29. Kuznetsov, V., Moiseienko, M., Moiseienko, N., Rostalny, B. and Kiv, A. Using Unity to Teach Game Development. DOI: 220/0010933400003364 Proceedings of the 1st Symposium on Advances in Educational Technology (AET 2020) - Volume 2, pages 506-515. ISBN: 978-989-758-558-6
30. C. Politowski, F. Petrillo and Y. -G. Guéhéneuc, "A Survey of Video Game Testing," 2021 IEEE/ACM International Conference on Automation of Software Test (AST), Madrid, Spain, 2021, pp. 90-99, doi: 10.1109/AST52587.2021.00018.
31. R. Ramadan and B. Hendradjaya, "Development of game testing method for measuring game quality," 2014 International Conference on Data and Software Engineering (ICODSE), Bandung, Indonesia, 2014, pp. 1-6, doi: 10.1109/ICODSE.2014.7062694.

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam grafů

8.4 Seznam použitých zkratek

Přílohy

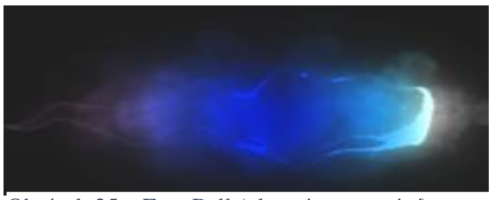
Příloha č. 1 – Přehled kouzel

Příloha č. 1 – Přehled kouzel

NAME	CAST TIME [s]	DMG	TYPE	SPEED
FireBall_1	2	5	FIRE	5
FireBall_2	1,5	8	FIRE	5
FireBall_3	1	10	FIRE	5
FrostBall_1	1,5	5	FROST	5
FrostBall_2	1	5	FROST	5
FrostBall_3	0,8	8	FROST	5
FrostBall_4	0,5	10	FROST	5

Tabulka 3 - PlayerBasicAttack

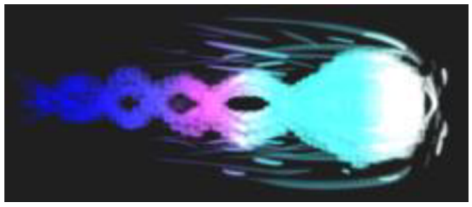
Obrázek 26 – FireBall (vlastní zpracování)



Obrázek 25 – FrostBall (vlastní zpracování)

NAME	CAST TIME [s]	EXPLOSION DMG	TYPE	SPEED
FireBall_4	1	10	FIRE	7,5
FrostNova_1	5	2	FROST	5
FrostNova_2	4	5	FROST	5
FrostNova_3	3	5	FROST	5
FrostNova_4	2	10	FROST	5

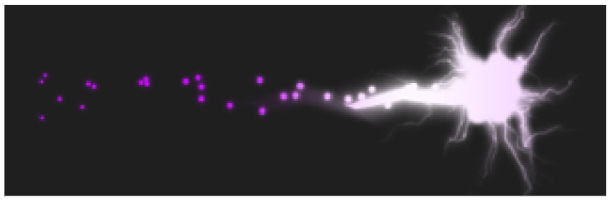
Tabulka 4 - PlayerExplosionAttack



Obrázek 27 – FrostNova (vlastní zpracování)

NAME	CAST TIME [s]	DMG	TYPE	SPEED	ADDINATIONAL CAST
ChainLightning_1	2	5	LIGHTNING	10	1
ChainLightning_2	2	5	LIGHTNING	10	2
ChainLightning_3	2	5	LIGHTNING	10	3
ChainLightning_4	2	5	LIGHTNING	10	5

Tabulka 5 - PlayerChainAttack

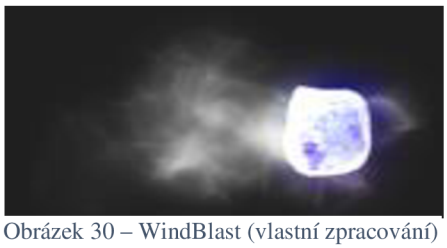


Obrázek 28 – ChainLightning (vlastní zpracování)

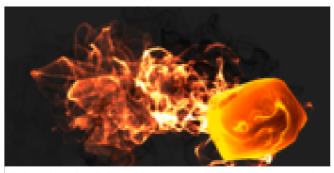
NAME	CAST TIME [s]	DMG	TYPE	SPEED	DISTANCE
LightningBolt_1	4	10	LIGHTNING	7	5
LightningBolt_2	3	10	LIGHTNING	7	5
LightningBolt_3	2	10	LIGHTNING	7	5
LightningBolt_4	1	15	LIGHTNING	7	5
WindBlast_1	3	5	Wind	7	5
WindBlast_2	2,5	7	Wind	7	5
WindBlast_3	2	7	Wind	7	5
WindBlast_4	1	10	Wind	7	5
PyroBlast_1	4	10	Fire	7	5
PyroBlast_2	3,5	12	Fire	7	5
PyroBlast_3	3	15	Fire	7	5
PyroBlast_4	1,5	15	Fire	7	5

Tabulka 6 - PlayerWaveAttack

Obrázek 31 – LightningBolt (vlastní zpracování)



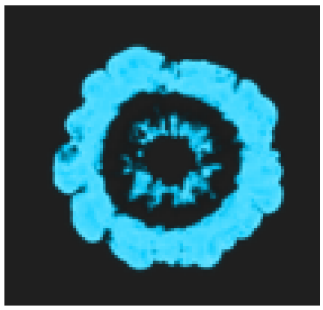
Obrázek 30 – WindBlast (vlastní zpracování)



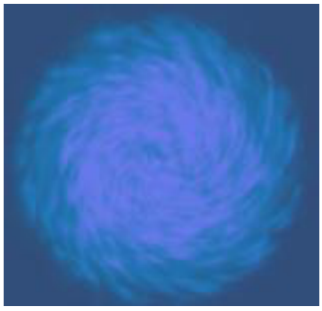
Obrázek 29 – PyroBlast (vlastní zpracování)

NAME	CAST TIME [s]	DMG	TYPE	DURATION	DISTANCE FROM PLAYER
FrostRing_1	5	5	Frost	3	3
FrostRing_2	5	6	Frost	4	3
FrostRing_3	5	8	Frost	5	3
FrostRing_4	5	10	Frost	10	3
Tornado_1	5	3	Wind	3	3
Tornado_2	5	5	Wind	4	3
Tornado_3	5	5	Wind	6	3
Tornado_4	5	10	Wind	10	3

Tabulka 7 - PlayerStaticPointAttack



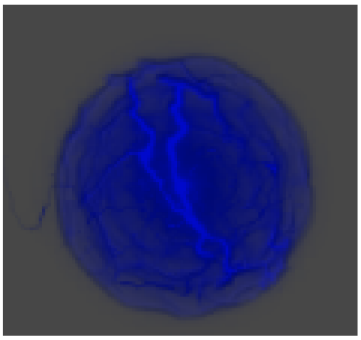
Obrázek 33 – FrostRing (vlastní zpracování)



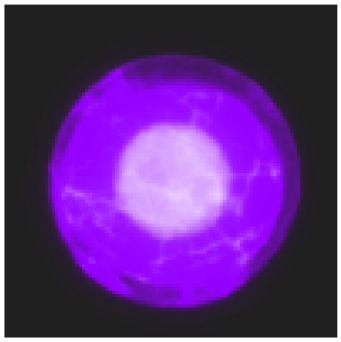
Obrázek 32 – Tornado (vlastní zpracování)

NAME	CAST TIME [s]	DMG	TYPE	SPEED	DISTANCE FROM PLAYER
FrostOrb_1	x	5	Frost	2	2
FrostOrb_2	x	5	Frost	3	2
FrostOrb_3	x	5	Frost	4	2
FrostOrb_4	x	10	Frost	5	2
LightningOrb_1	x	7	LIGHTNING	2	2
LightningOrb_2	x	7	LIGHTNING	3	2
LightningOrb_3	x	7	LIGHTNING	4	2
LightningOrb_4	x	12	LIGHTNING	5	2

Tabulka 8 - PlayerRotationAttack



Obrázek 34 – FrostOrb (vlastní zpracování)



Obrázek 35 – LightningOrb (vlastní zpracování)