# Vývoj hry pro výuku programovacího jazyka C#

Tomáš Jahoda

Bakalářská práce

2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Tomáš Jahoda** |
| Osobní číslo: | **A21602** |
| Studijní program: | **B0613A140020 Softwarové inženýrství** |
| Forma studia: | **Kombinovaná** |
| Téma práce: | **Vývoj hry pro výuku programovacího jazyka C#** |
| Téma práce anglicky: | **Development of a Game for Learning the C# Programming Language** |

## Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Navrhněte hru pro výuku programování v jazyce C#, předpokládejte využití herního enginu Unity.
3. Vytvořte výukovou hru dle návrhu.
4. Otestujte funkčnost výukové hry.
5. Implementaci a testování vhodně popište.

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

Seznam doporučené literatury:


1. TYKOSKI, Scott. *Mastering Game Design with Unity 2021*. Delhi: BPB Publications, 2023. ISBN 9789355512178.
2. OKITA, Alex. *Learning C# programming with Unity 3D*. Second edition. Boca Raton, FL: A K Peters/CRC Press, 2020. ISBN 9780429810251.
3. VIRIUS, Miroslav. *Programování v C#: od základů k profesionálnímu použití*. Grada, 2020. ISBN 978-80-271-1216-6.
4. UNITY TECHNOLOGIES. Unity User Manual 2022.3 (LTS). *[Unity Documentation]* [online]. 2023 [cit. 2023-11-09]. Dostupné z: https://docs.unity3d.com/Manual/index.html.
5. C# Tutorial. *W3Schools* [online]. c1999-2023 [cit. 2023-11-09]. Dostupné z: https://www.w3schools.com/cs/index.php.
6. SMITH, Matt a Shaun FERNS. *Unity 2021 cookbook : over 140 recipes to take your Unity game development skills to the next level*. 4. vyd., Birmingham; Mumbai: Packt, 2021. ISBN 978-1-83921-761-6.
7. MCGONIGAL, Jane. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. Ilustrované vydání, dotisk. Vintage Books, 2012. ISBN 9780099540281.


Vedoucí bakalářské práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

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

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

Ve Zlíně dne 5. ledna 2024

**Prohlašuji, že**

- beru na vědomí, že odevzdáváním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1198 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách, ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

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

Ve Zlíně, dne 10.5.2023

Tomáš Jahoda, v. r.
podpis studenta

## ABSTRAKT

Tato bakalářská práce se zabývá návrhem, vývojem a testováním výukové hry zaměřené na osvojení programovacího jazyka C#. Teoretická část práce se zabývá implementací naučných her do výuky a jejich vlivem, přibližuje programovací jazyk C# a vývojářský engine Unity. Nakonec je čtenář seznámen s vývojem naučných her včetně ukázek již existujících a přiblížení pojmů z herního prostředí, jako jsou 2D nebo RPG. Praktická část se následně zabývá návrhem naučné hry, jejím vývojem a testováním na různých operačních systémech.



Klíčová slova: výuková hra, herní engine, gamifikace, unity, c#



## ABSTRACT

This bachelor thesis deals with the design, development and testing of an educational game aimed at learning the C# programming language. The theoretical part of the thesis deals with the implementation of educational games in the educational setting and their impact, introducing the C# programming language and the Unity development engine. Finally, the reader is introduced to the principles of educational game development, including examples of existing games and an introduction to concepts from the game environment, such as 2D or RPG. The practical part then deals with the design of an educational game, its development and testing on different operating systems.



Keywords: educational game, game engine, gamification, unity, c#

Můj největší dík patří vedoucímu práce Ing. Tomáši Vogeltanzovi, Ph.D., za jeho cenné rady, trpělivost, ochotu, odborný přístup a čas, který mi v průběhu celého procesu psaní této bakalářská práce věnoval.

Dále bych rád poděkoval svým přátelům a rodině, kteří mě po celou dobu mého studia podporovali.

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

# OBSAH

# ÚVOD

V dnešní době jsou videohry multimiliardovým průmyslem. Není se čemu divit, když v jeden moment až desítky miliónů hráčů hrají různé počítačové hry po celém světě. Tyto hry nejsou jen zdrojem zábavy, ale stávají se nástrojem pro odreagování se od každodenního stresu, seriózní prací ba dokonce i platformou pro sociální interakce. Videohry však mohou přesáhnout rámec zábavy a stát se významným nástrojem pro vzdělávání.

Cílem této práce je čtenáře seznámit s potenciálem videoher jako edukativního nástroje, přiblížení a prohloubení znalostí programovacího jazyka C# a představení enginu pro vývoj her Unity.

V teoretické části jsou popsány důležité metody, jakými se hry mohou do výuky implementovat a analýza jejich vlivu na uživatele. Následně je popsán jazyk C# včetně jeho základních i pokročilých funkcí a jeho ukázka v praxi. Teoretická část také přibližuje engine Unity, což je jeden z mnoha enginů, za pomoci kterého je možno hry vyvíjet. Poslední část řeší samotný princip vývoje naučných her, kde je objasněno, jaké faktory je nutno při vývoji brát v potaz. V závěru praktické části je uveden příklad již existujících naučných her, které jsou aktivně ve výuce používány.

Praktická část práce je zaměřena na přímý vývoj naučné hry, která má za úkol uživateli přiblížit základy i pokročilé funkce programovacího jazyka C#. Na začátku je hra graficky navržena spolu s funkčními a nefunkčními požadavky a následně je detailně popsán její celkový vývoj za použití enginu Unity. Po její implementace je funkčnost vytvořené hry otestována na různých operačních systémem pro zjištění kompatibility.

Koncem praktické části je řešena budoucnost hry a možnosti na další implementace.

# I. TEORETICKÁ ČÁST

# 1 ÚVOD DO VZDĚLÁVÁNÍ POMOCÍ VIDEOHER

Videohry ve vzdělávání začaly nabývat na významu počátkem 80. let, kdy se vývojáři her snažili využít rostoucího zájmu o domácí počítače jako prostředek vzdělávání. Zajímavým příkladem je hra "The Oregon Trail", původně vyvinutá v polovině 70. let pro studenty v Minnesotě, která se stala široce dostupnou a populární po svém uvedení na trh pro Apple II v roce 1985. Cílem této hry, stejně jako dalších, jako například "Where in the World Is Carmen Sandiego?" z roku 1985, jejíž ukázku můžeme vidět na obrázku 1., bylo zábavnou formou naučit hráče zeměpis. Tyto hry ukázaly, že videohry mohou být účinným a poutavým nástrojem vzdělávání [1].


Obrázek 1 Where in the World Is Carmen Sandiego z roku 1985

Důvodem pro využití videoher ve výuce bylo zvýšení motivace a zapojení studentů do procesu učení. Hry jako vzdělávací nástroje umožňují studentům učit se prostřednictvím praxe a přímé interakce, což může vést k lepšímu pochopení a zapamatování látky. Vzdělávací hry také umožňují individualizované tempo učení, což je výhodné pro studenty s různým stylem a rychlostí učení [2].

Vzdělávací videohry navíc často využívají principy gamifikace ke zvýšení motivace studentů prostřednictvím systému odměn, úrovní a výzev. Učení se tak stává poutavějším a interaktivnějším zážitkem, který může podpořit trvalý zájem o vzdělávací obsah.

Pojmem gamifikace se označuje proces, který využívá herních prvků a principů v nehráčském kontextu s cílem zvýšit angažovanost, posílit motivaci a podpořit konkrétní chování. To může zahrnovat prvky jako jsou body, odznaky, žebříčky a výzvy, které jsou implementovány do vzdělávacích programů, marketingových kampaní, online komunit, pracovních procesů a dalších aplikací.

Přínosy videoher ve vzdělávání byly pozorovány nejen v základních akademických dovednostech, jako je čtení, matematika a zeměpis, ale také v rozvoji kritického myšlení, řešení problémů a týmové práce.

## 1.1 Metody začlenění her do výuky

Začlenění videoher do výukového procesu vyžaduje promyšlený přístup, který respektuje pedagogické cíle a zároveň maximálně využívá potenciálu her jako motivačního a vzdělávacího nástroje. Tato podkapitola se věnuje různým metodám a strategiím, jak efektivně integrovat hry do učebních plánů.

### 1.1.1 Výběr vhodných her

Důležitým aspektem k úspěšné integraci her do výuky je pečlivý výběr her, které jsou nejen atraktivní a zábavné, ale také pedagogicky smysluplné. Hry by měly být vybrány na základě jejich schopnosti rozvíjet specifické dovednosti nebo znalosti, které jsou předmětem výuky [2].

To zahrnuje:

- **Zaměření na konkrétní vzdělávací cíle:** Vybrané hry by měli přímo korespondovat s vzdělávacím cílem předmětu či kurzu [2].
- **Přizpůsobení věkové skupině a předchozím znalostem:** Hry by měly být přizpůsobeny věku studentům a jejich dosud dosažených znalostí a dovedností [2].

### 1.1.2 Integrace do učebního plánu

Začlenění her do výuky vyžaduje promyšlenou integraci do učebního plánu, kdy hry slouží pouze jako doplněk k tradičním výukovým metodám. Hry by nikdy neměly plně nahradit celou výuku, měly by čistě sloužit jen jako propojení mezi herním obsahem a učivem tak, aby studenti rozuměli, jakým způsobem hry podporují jejich učení [2].

### 1.1.3 Diskuse

Využití videoher ve výuce by nemělo sloužit pouze jako náhrada učiva nebo forma samostudia. Zde samozřejmě záleží na typu, o jakou hru se jedná. Diskuse po herní aktivitě může být přínosem jak pro studenty, tak pro pedagogy, kteří se mohou dozvědět více o strategických postupech a logických řešeních od ostatních účastníků [2].

### 1.1.4 Zpětná vazba

Důležitým faktorem je také systém zpětné vazby a hodnocení, který umožňuje jak studentům, tak učitelům porozumět výsledkům učení a efektivity konkrétních her. Na základě těchto hodnocení je pak možno hry upravit a integrovat tak, aby se efektivita a porozumění látky zlepšilo [2].

## 1.2 Přínos a nevýhody výukových her

Existují různé studie, které potvrzují, že implementace her ve výukovém prostředí dokáže být přínosnou aktivitou. Každopádně se najdou i studie, které tvrdí pravý opak. Rozdíl v těchto studiích spočívá v tom, že jedny se zaměřují na výuku, kde jsou videohry propojené s klasickou výukou tak, jak ji známe, zatímco ty druhé čerpají data čistě z výuky postavené pouze na hraní.[3][4]

### 1.2.1 Výhody výukových her

Co se týká výhod výukových her, rozhodně zde najdeme vícero důležitých informací než u nevýhod. Už jen proto, že učení se pomocí her může být pro studenty zábavné a díky tomu jsou více motivovaní ve studiu. Kromě a zábavy a motivace mohou hry pomoci v jiných oblastech, jako jsou například:

- Rozvoj kritického myšlení a řešení problémů [3][4].
- Ovlivnění chování [3][4].

- Možnost praktické aplikace teoretických znalostí v bezpečném a kontrolovaném prostředí [3][4].
- Spolupráce a komunikace s ostatními [3][4].

### 1.2.2 Nevýhody výukových her

Jak již bylo zmíněno, valná většina nevýhod vyplývá ze studií spojených s výukou kompletně nahrazenou videohrami. Hry nikdy plně nezvládnou nahradit klasickou výuku, nicméně určitými nevýhodami mohou být:

- Příliš mnoho času tráveného u obrazovky [3][4].
- Hry nejsou přizpůsobeny jednotlivcům [3][4].
- Ne vždy se řídí výukovým cílem předmětu nebo kurzu [3][4].
- Hry mohou být zdrojem rozptýlení [3][4].

V kontrastu s výhodami se nejedná o záležitosti, které by nebylo možné minimalizovat.

# 2 PROGRAMOVACÍ JAZYK C#

Programovací jazyk C# byl oficiálně uveden veřejnosti firmou Microsoft roku 2002, původně především pouze pro Windows zařízení, jelikož jazyk byl navržen pro .NET Framework 1.0, který nebyl multiplatformní a nepodporoval rozdílné operační systémy (OS). Cílem bylo vytvořit moderní, objektově orientovaný jazyk, který by se našel někde mezi tehdejšími C++ a Javou. [5][6]

Cílem C# bylo vytvořit bezpečný, efektivní a snadno použitelný jazyk pro vývoj různorodých aplikací. Při tvorbě se usilovalo o vytvoření silného typového systému pro minimalizaci chyb, zjednodušení složitějších konceptů, jako jsou například ukazatele a přetěžování operátorů u C++, a automatickou správu paměti (garbage collection). To vše pro to, aby byl C# výkonným a flexibilním programovacím jazykem [5][6].

## 2.1 Základní koncepty C#

Klíčem pro efektivní programování za pomocí C# je znát jeho základní koncepty, které jsou nezbytné k pochopení jeho struktury a funkcí.

### 2.1.1 Syntaxe a struktura

Syntaxe jazyka C# definuje pravidla pro strukturování a psaní kódu. Jako moderní, objektově orientovaný jazyk C# využívá čitelnosti kódu, umožňující vývojářům efektivně psát udržitelný a snadno pochopitelný kód. Porozumění těmto základním pravidlům a struktuře je stěžejní pro každého, kdo se C# chce naučit, jelikož se jedná o základ, na kterém je celý jazyk postaven [5][7].

Základními pravidly syntaxe jsou:

- **Středníky (` ; `)** – Každý výraz napsaný v C# je nutno zakončit středníkem. Ten dává kompilátoru vědět, že se jedná o konec příkazu, díky čemuž je možné oddělit jeden příkaz od druhého [5][7].
- **Závorky (` ( ) `, ` { } ` , ` [ ] `)** – Závorky jsou nepostradatelnou součástí C#, kdy kulaté závorky ` ( ) ` jsou využívány pro parametry metod a řízení operátorů. Složené závorky ` { } ` definují bloky kódu, jako jsou například těla metod, tříd, či smyček (loopů). Hranaté závorky ` [ ] ` se používají pro indexy polí či listů [5][7].
- **Komentáře ` / ` -** Jedná se o části kódu, které jsou ignorovány kompilátorem, takže se nikdy neprovedou. Z pravidla se využívají k vysvětlení určité části kódu či

k omezení jeho funkčnosti při případném testování. Komentáře mohou být jednořádkové za použití ` // `, které můžeme vidět na obrázku 2, nebo naopak více řádkové, kdy budou všechny řádky komentáře ohraničené ` /* ` a ` */ ` [5][7].

```
using System;

namespace HelloWorld
{
  class Program
  {
    static void Main(string[] args)
    {
      // This is a comment
      Console.WriteLine("Hello World!");
      // This is also a comment
     }
  }
}
```

Obrázek 2 Ukázka použití komentářů v C#

Strukturu programu tvoří:

- **Namespaces** – Pomáhají organizovat kód a předchází chybám ve jménech. Základním příkladem může být namespace 'System', ve kterém se nachází základní standardní třídy .NET Frameworku [5][7].
- **Třídy a metody** – Třídou je myšlen základní blok C# programu, který definuje data a chování jednotlivých objektů. Metody jsou pak funkce definované ve třídách, které dle zadaných dat provádějí různé akce [5][7].
- **Main** – Jedná se o takzvanou Entry Point metodu, která musí být obsažena v každém z jednoho C# programů. Bez ní by program nefungoval, jelikož se jedná o vstupní bod programu [5][7].

### 2.1.2 Datové typy a proměnné

Pro to, aby bylo možno ukládat a manipulovat s daty v programu, je nutno znát a porozumět datový typům a proměnným a jejich správné deklaraci. Datových typů existuje vícero druhů, jako například primitivní datové typy:

- **Celá čísla (` int` , ` long `)** – Existuje mnoho variant, jakými lze uložit celočíselnou hodnotu do proměnné. Nejpoužívanějším datovým typem je každopádně ` int `, který dokáže uložit hodnoty od -2,147,483,648 do 2,147,483,648. Pokud by to z nějakého důvodu nebylo dostačující, je zde možnost použití ` long ` datového typu, který dokáže uložit několikanásobně větší čísla. Deklarace by poté vypadala ` int a = 5; ` [8].
- **Desetinná čísla (` float `, ` double `, ` decimal `)** - Pokud je potřeba uložit čísla obsahující desetinnou čárku, například pro přesnější výpočty, je možno použít jeden ze zmíněných datových typů. Liší se v přesnosti a počtu čísel za desetinnou čárkou. Vytvoření proměnné pro desetinný datový typ je obdobný jako u celočíselného typu: ` double b = 5.1; `. Datové typy float a decimal od uživatele vyžadují použití takzvaného `precision specifieru`, který je zapsán v podobě `float c = 5.1f `, kdy přípona f za hodnotou značí, že se jedná specificky o datový typ float [9].
- **Logické typy (` bool `)** – Reprezentuje pravdivostní hodnoty ` true ` = 1 – jedná se o pravdivý výrok, a ` false ` = 0 – zde se jedná o nepravdivý výrok [10].
- **Znak (` char `)** - Pomocí tohoto datového typu je možno ukládat jednotlivé Unicode znaky, jako může být například písmeno `o` nebo číslice `4` [11].

Dále existují složené datové typy:

- **Pole** – Za pomocí polí můžeme do jedné proměnné uložit více hodnot stejného datového typu. Například můžeme vytvořit pole číslic, které bude obsahovat tři číslice: ` int poleCisel[] = new int[3]; ` [12].
- **Řetězec (` string `)** – Jedná se o seskupení znaků za sebou, ideální řešení pro výpis textu: ` string text = "Ahoj!"; ` [13].

Ukázku použití datových typů můžeme vidět na obrázku 3.

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      int myNum = 7;
      double myDoubleNum = 5.99D;
      float myFloatNum = 6.5f;
      char myLetter = 'D';
      bool myBool = true;
      string myText = "Hello";
      Console.WriteLine(myFloatNum+" "+myDoubleNum+" "+myNum);
      Console.WriteLine(myLetter+ " is " +myBool);
      Console.WriteLine(myText);
    }
  }
}
```

Obrázek 3 Ukázka použití datových typů v C#

Každá nová proměnná musí být deklarována s určitým datovým typem, který určuje, jaké hodnoty daná proměnná může obsahovat. Kromě datového typu obsahuje deklarace i název proměnné, se kterým pak následně je v programu pracováno. Deklarovaná proměnná se následně může inicializovat, tím je myšleno dosazení hodnoty do proměnné [14].

- **Deklarace** – ` int a; ` deklaruje proměnnou ` a ` datového typu ` int ` [14].
- **Inicializace** – ` a = 5; ` vloží do proměnné ` a ` celočíselnou hodnotu ` 5 ` [14].

Deklaraci a inicializaci je možné provést v jednom kroku, jak již bylo ukázáno v předchozím seznamu datových typů.

C# umožňuje deklarovat proměnné za pomoci použití ` var `, kdy je kompilátoru umožněno, aby automaticky proměnné přiřadil vhodný datový typ na základě vložené hodnoty. Příkladem může být ` var shape = "Circle“; `, kde kompilátor sám určí, že proměnná ` shape ` bude typu ` string ` [14].

Pro správnou práci s pamětí je dobré znát rozdíly mezi hodnotovými a referenčními typy proměnných:

- **Hodnotový typ (` struct `, primitivní datové typy)** – Hodnota se přímo ukládá do proměnné [15].

- **Referenční typ (` class `, ` string `, pole)** – Proměnná uloží odkaz na místo v paměti, kde jsou data proměnné uložená [16].

### 2.1.3 Operátory a výrazy

Základními nástroji pro například provádění výpočtů nebo porovnávání hodnot jsou operátory a výrazy, ve kterých jsou použity. Pracujeme zde s pojmy jako jsou aritmetické operátory, relační operátory, logické operátory, přiřazovací operátory a unární operátory [17].

Aritmetické operátory:

- **Sčítání (` + `)** – Slouží k sečtení dvou hodnot [17].
- **Odčítání (` - `)** – Slouží k odečtení jedné hodnoty od druhé [17].
- **Násobení (` * `)** – Slouží k násobení dvou hodnot [17].
- **Dělení (` / `)** – Slouží k dělení dvou hodnot [17].
- **Modulo (` % `)** – Vrací zbytek po dělení dvou čísel [17].

Relační operátory:

- **Rovnost (` == `)** – Ověřuje, zda jsou dvě hodnoty rovné [18].
- **Nerovnost (` != `)** – Ověřuje, zda jsou dvě hodnoty rozdílné [18].
- **Větší než (` > `)** – Ověřuje, zda je hodnota vlevo větší než hodnota vpravo [19].
- **Menší než (` < `)** – Ověřuje, zda je hodnota vlevo menší než hodnota vpravo [19].
- **Větší nebo rovno (` >= `)** – Ověřuje, zda je hodnota vlevo větší nebo stejná jako hodnota vpravo [19].
- **Menší nebo rovno (` <= `)** – Ověřuje, zda je hodnota vlevo menší nebo stejná jako hodnota vpravo [19].

Logické operátory:

- **AND (` && `)** – Vrací pravdivý výrok, pokud jsou dva či více porovnávaných hodnot také pravdivými výroky (true) [20].
- **OR (` || `)** – Vrací pravdivý výrok, pokud alespoň jedna z porovnávaných hodnot je také pravdivým výrokem (true) [20].

- **NOT (` ! `)** – Invertuje logickou hodnotu z pravdivého výroku (true) na nepravdivý výrok (false) [21].

Přiřazovací operátory:

- **Přiřazení (` = `)** – Přiřazuje hodnotu vpravo od operátoru proměnné vlevo [17].
- **Sčítání a přiřazení (` += `)** – Přičte hodnotu vpravo k hodnotě vlevo a následný výsledek přiřadí k proměnné [22].

Stejným způsobem by fungovalo například odečítání a přiřazení, násobení a přiřazení nebo dělení a přiřazení.

Unární operátory:

- **Inkrementace (` ++ `)** – Zvyšuje hodnotu proměnné o 1 [17].
- **Dekrementace (` -- `)** – Snižuje hodnotu proměnné o 1 [17].

Výrazem je myšleno kombinování všech dosavadních znalostí – proměnných, hodnot a operátorů k dosažení výpočtu nové hodnoty proměnné. Například ` int result = 1 + 3; `. Jedná se o výraz, kdy se obě hodnoty prvně sečtou a následně se přiřadí k proměnné.

### 2.1.4 Podmíněné výrazy

Práce podmíněných výrazů spočívá v tom, aby rozhodovali, jaká část kódu se vykoná v závislosti na daných podmínkách, anebo umožní, aby se jedna část kódu vykonala vícekrát.

**if (if-else)** – Asi tou nejzákladnější formou rozhodování. V případě, že máme podmínku ve stavu ` true `, vykoná se blok kódu uvnitř ` if `. Pokud je podmínka v jiném stavu, a je dostupný blok kódu v rámci ` else `, vykoná se místo toho ten [23], jako vidíme na obrázku 4.

**else if** - Jde o větev rozhodování, která následuje po **if** a umožňuje další úroveň kontroly podmínek. Pokud podmínka uvnitř **if** není splněna, **else if** nabízí alternativní blok kódu, který se vykoná, pokud je jeho podmínka pravdivá. Je možné použít více **else if** větví za sebou, čímž vytváříme komplexní strukturu rozhodování, kde každá větev má svou vlastní podmínku [23], jak zobrazuje obrázek 4.

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      int time = 22;
      if (time < 10)
      {
        Console.WriteLine("Good morning.");
      }
      else if (time < 20)
      {
        Console.WriteLine("Good day.");
      }
      else
      {
        Console.WriteLine("Good evening.");
      }
    }
  }
}
```

Obrázek 4 Ukázka podmíněných výrazů if, if-else a else if

**switch** – O něco rozsáhlejší ` if`-else `. Vhodný, pokud je k dispozici větší počet možností. Zároveň je přehlednější a rychlejší pro porovnávání jedné proměnné s několika konstantními hodnotami [23]. Ukázku můžeme vidět na obrázku 5.

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      int day = 4;
      switch (day)
      {
        case 1:
          Console.WriteLine("Monday");
          break;
        case 2:
          Console.WriteLine("Tuesday");
          break;
        case 3:
          Console.WriteLine("Wednesday");
          break;
        case 4:
          Console.WriteLine("Thursday");
          break;
        case 5:
          Console.WriteLine("Friday");
          break;
        case 6:
          Console.WriteLine("Saturday");
          break;
        case 7:
          Console.WriteLine("Sunday");
          break;
      }
    }
  }
}
```

Obrázek 5 Ukázka použití podmínky switch v C#

**for –** Jedná se o smyčku, kdy se daný blok kódu bude opakovat po určitý počet iterací, který je ve většině případů znám předem [24] (Obr. 6).

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      for (int i = 0; i < 5; i++)
      {
        Console.WriteLine(i);
      }
    }
  }
}
```

Obrázek 6 Ukázka použití podmínky for v C#

**while** – Obdoba smyčky ` for `. Jediný rozdíl je v tom, že není znám počet iterací. Kód se provádí tak dlouho, dokud je podmínka pravdivá [24].

**do-while** – Funguje na stejném principu jako ` while ` s tím rozdílem, že kód se pokaždé provede minimálně jednou. I v případě, že je podmínka nepravdivá [24].

**foreach** – Jedná se o speciální formu smyčky, která provádí iterace na základě proměnné a kolekce nebo pole [24].

## 2.2 Objektově orientované programování v C#

Objektově orientované programování (OOP) je vzor, který umožňuje programátorům psát kód tak, že data a operace nad daty jsou spojeny do jednotek, které se nazývají objekty. C# jako objektově orientovaný jazyk poskytuje plnou podporu pro OOP principy, včetně tříd a objektů, dědičnosti, polymorfismu, abstrakce a zapouzdření. Díky těmto konceptům je možné kód lehce udržovat, rozšiřovat a opakovaně používat [25].

### 2.2.1 Třídy a objekty

Třída je základním kamenem pro vytváření objektů v C#. Definuje jejich stav a jejich chování pomocí dat a metod. Třída obsahuje členské proměnné pro ukládání těchto dat a metody pro práci s nimi. Kromě toho může obsahovat konstruktory, které slouží k inicializaci nové instance objektu a další speciální metody, kterými mohou být například destruktory nebo indexery [26].
Objekty jsou instance jedné konkrétní třídy, ve které jsou definovány jeho stav a chování. Každá třída může mít více objektů, s tím, že každý objekt je samostatnou entitou nezávislou na ostatních objektech [26].
Tvorbu třídy a jejího objektu můžeme vidět na obrázku 7.

```
using System;

namespace MyApplication
{
  class Book
  {
    string theme = "fantasy";

    static void Main(string[] args)
    {
      Book myObj = new Book();
      Console.WriteLine(myObj.theme);
    }
  }
}
```

Obrázek 7 Ukázka vytvoření třídy a objektu v C#

### 2.2.2 Dědičnost

Jedná se o nezbytný koncept OOP, který umožňuje konkrétní třídě zdědit stavy (proměnné) a chování (metody) od jiné třídy. Tento mechanismus implementuje základní hierarchii tříd [27].

Třída, od které se dědí, je známá jako základní třída (rodič, předek) a třída, která dědí je známá jako odvozená (potomek) [27].

Odvozená třída automaticky získává veřejné a chráněné atributy a metody rodičovské třídy a tím získává možnost rozšíření nebo úpravy chování základní třídy [27].

### 2.2.3 Polymorfismus

Dalším konceptem OOP je polymorfimus (Obr. 8), který umožňuje objektům různých tříd reagovat na volání stejné metody různými způsoby. V kontextu C# to znamená, že stejná metoda má jiné chování podle toho, který objekt ji volá [28].

```
using System;

namespace MyApplication
{
  class Animal  // Base class (parent)
  {
    public virtual void animalSound()
    {
      Console.WriteLine("The animal makes a sound");
    }
  }

  class Cat : Animal  // Derived class (child)
  {
    public override void animalSound() // Overrides the virtual method
    {
      Console.WriteLine("The cat says: meowazing");
    }
  }

  class Dog : Animal  // Derived class (child)
  {
    public override void animalSound() // Overrides the virtual method
    {
      Console.WriteLine("The dog says: woof bark");
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Animal myAnimal = new Animal();  // Create a Animal object
      Animal myCat = new Cat();  // Create a Cat object
      Animal myDog = new Dog();  // Create a Dog object

      myAnimal.animalSound();
      myCat.animalSound();
      myDog.animalSound();
    }
  }
}
```

Obrázek 8 Ukázka funkce polymorfizmu v C#

### 2.2.4 Abstraktní třídy

Abstraktní třídou nazýváme takovou třídu, která nemůže vytvořit svou instanci na přímo a je používána pouze jako základní třída pro odvozené. Abstraktní třídy mohou obsahovat abstraktní metody, které jsou deklarované, ale nejsou implementované. Odvozené třídy jsou tak povinny tyto metody implementovat. [29]

Abstraktní třídy se dají využít v případě, kdy je možno vytvořit šablonu pro skupinu podobných tříd, přičemž každá z odvozených tříd bude mít vlastní implementaci jedné nebo více metod [29].

### 2.2.5 Zapouzdření

Aby si program udržel určitou integritu, OOP využívá principu zapouzdření. Jde o takovou ochranu tříd před vnějším kódem, aby ke třídám nepřistupoval tak, jak se mu zachce a neoprávněně ji nemodifikoval data a metody. Pomocí zapouzdření si třída může určité funkce či proměnné uschovat [30].

Zapouzdření se dosahuje pomocí modifikátorů přístupu. Těmi jsou ` public `, ` private `, ` protected `, ` internal `, ` private protected ` a ` protected internal `ty. Tyto modifikátory určují, z kterých částí programu lze k členským proměnným a metodám přistupovat [30].

## 2.3 Pokročilé funkce jazyka C#

Pro ještě flexibilnější a výkonnější kód existuje další řada funkcí, které nám pomohou dosáhnout tohoto cíle. Takovými funkcemi mohou být například generické programování, výjimky a jejich ošetření, LINQ, asynchronní programování, použití delegátů anebo událostí.

### 2.3.1 Generické programování

Technika generického programování se využívá při definování tříd, metod, rozhraní a delegátů za použití generických typů. To jsou placeholderové typy, jež umožňují specifikovat konkrétní typ při instanci nebo volání metody. Generické typy zvyšují bezpečnost typů tím, že umožňují kompilátoru kontrolovat a omezit typy dat, s nimiž se pracuje. To minimalizuje riziko běhových chyb spojených s nesprávnými typy dat a
Generická třída ` List<T> ` je jeden z nejpoužívanějších příkladů generického programování. Třída umožňuje ukládat seznam položek libovolného typu bez nutnosti přetypovávání nebo obav z typových chyb [31].

### 2.3.2 Delegáty a události

Když mluvíme o delegátech, mluvíme o typech, které reprezentují odkazy na metody s určitým seznamem parametrů a návratovým typem. Lze je použít k předání metod jako argumentů, přiřazení vícero metod k jednomu delegátu (zde mluvíme o multicast delegátech) nebo k vytvoření událostně orientovaných programů [32].
Události jsou na druhou stranu speciálním druhem delegátů, které umožňují definovat, kdy může být delegát volán. Události jsou zároveň způsob, jak třída může informovat vnější komponenty, že došlo k nějaké situaci [33].

### 2.3.3 Výjimky a jejich ošetření

Výjimky jsou objekty, které jsou vyvolány během specifických situací, kdy aplikace narazí na problém, který nejde řešit běžným řízením toku programu. C# poskytuje širokou škálu tříd výjimek pro různé druhy chyb – od obecných, jako je například ` Exception `, až po specifické, kterou může být ` IOException ` [34].

K ošetření výjimek se používá blok kódu ` try ` a ` catch `, kde ` try ` je obalen kolem kódu, ve kterém výjimka může nastat, zatímco ` catch ` definuje, jakým způsobem program na specifickou výjimku zareaguje [34].

# 3 UNITY

Unity je všestranný herní engine a vývojové prostředí používané k vytváření interaktivního obsahu napříč širokou škálou platforem. S jeho pomocí mohou vývojáři a designéři vytvářet, testovat a nasazovat aplikace od 2D mobilních her až po 3D tituly pro konzole a VR zařízení. Jeho schopnost integrovat fyzikální simulace, umělou inteligenci a síťové funkce, spolu s pokročilým grafickým renderováním, činí z Unity robustní řešení pro tvorbu komplexních herních světů [35].

Unity je navrženo tak, aby byl přístupný i pro ty s menšími technickými znalostmi, zároveň však nabízí hluboké možnosti pro zkušené programátory prostřednictvím jeho skriptovacího API v C#. Unity přináší integraci s řadou nástrojů a služeb, které usnadňují kolaborativní vývoj a zjednodušují procesy jako je animace, audio design a tvorba uživatelského rozhraní [35].

## 3.1 C# v Unity

### 3.1.1 Úvod do programování v Unity

Programování v Unity je základním kamenem pro vytváření interaktivního obsahu, ať už jde o hry, simulace, nebo vzdělávací aplikace. Unity poskytuje flexibilní a uživatelsky přívětivé prostředí pro tvorbu 2D, 3D nebo VR/AR her s použitím jazyka C#, který je díky své síle a přehlednosti oblíben mezi vývojáři na všech úrovních. V této bakalářské práci se nicméně zaměříme pouze na 2D rozhraní vývoje her [35].

Když uživatel začne programovat v Unity, musí se naučit základy práce s Unity Editorem, vytvářením skriptů, manipulací s herními objekty a používáním Unity API pro přidávání interaktivity do projektů. Vývojáři začleňují skripty do herních objektů pomocí komponent, což jsou modulární bloky kódu, které definují chování objektů ve hře [35].

S Unity je možné rychle prototypovat a iterovat herní design, což umožňuje vývojářům testovat a upravovat herní mechaniky s okamžitou zpětnou vazbou. Platforma podporuje množství funkcí, jako jsou fyzikální simulace, AI, animace a mnoho dalších, které usnadňují vývoj komplexních 2D her [35].

### 3.1.2 Základy C# pro Unity

C# je primární programovací jazyk používaný v Unity pro skriptování herní logiky a interakce. Díky své flexibilitě a síle je C# ideální volbou pro vývojáře hledající efektivní způsob, jak oživit herní prvky a implementovat komplexní mechaniky [36].

#### *3.1.2.1 Základní koncepty*

- **Proměnné a datové typy – C#** je staticky typovaný jazyk, což znamená, že typ každé proměnné je znám při kompilaci. Unity podporuje všechny základní datové typy C# jako **int** (celá čísla), **float** (desetinná čísla), **string** (řetězce), a **bool** (logické hodnoty), které jsou základem pro uchovávání a manipulaci s daty ve hrách [36].
- **Metody –** Jsou způsobem, jakým se v C# organizuje kód do logických bloků. V Unity se často používají metody, jako jsou **Start()** a **Update()**, které jsou automaticky volány Unity enginem. **Start()** se spouští při prvním načtení scény, zatímco **Update()** se volá každý snímek a je ideální pro kód, který sleduje stisknutí kláves, pohyby myši nebo jiné interaktivní prvky [36].
- **Třídy a objekty –** V Unity je každý skript C# třídou, která dědí z Unity třídy **MonoBehaviour**. To umožňuje skriptům přistupovat k funkcím a životnímu cyklu enginu. Třídy mohou obsahovat proměnné, metody a jiné třídy, což umožňuje složitější struktury a funkce [36].
- **Práce s Unity API –** Pro interakci s herním enginem Unity používá C# rozsáhlé API, které umožňuje manipulaci s herními objekty, scénami, fyzikou a mnohem více [37].

#### *3.1.2.2 Praktický příklad*

Následující je jednoduchý skript v C#, který by mohl být použit ve hře Unity pro pohyb objektu:

```
using UnityEngine;
public class SimpleMovement : MonoBehaviour
{
   public float speed = 5.0f;
   void Update()
   {
      float moveHorizontal = Input.GetAxis("Horizontal");
      float moveVertical = Input.GetAxis("Vertical");
      Vector3 movement = new Vector3(moveHorizontal, moveVertical, 0.0f);
      transform.position += movement * speed * Time.deltaTime;
   }
}
```

Tento skript využívá metodu **Update** pro zpracování vstupů od hráče a pohybuje objektem v horizontálním a vertikálním směru.

### 3.1.3 Interakce skriptů a herních objektů

Interakce mezi skripty a herními objekty v Unity je základem pro vytváření dynamických a interaktivních her. Skripty napsané v C# umožňují programově manipulovat s objekty ve scéně, reagovat na uživatelské vstupy a řídit herní logiku [35].

#### *3.1.3.1 Jak skripty komunikují s herními objekty*

**Připojení skriptů** – Skripty jsou připojovány k herním objektům jako komponenty. Tímto způsobem může skript přímo ovlivňovat objekt, ke kterému je přiřazen, a využívat jeho vlastnosti a komponenty, jako je Transform nebo Rigidbody [38].
**Komunikace mezi skripty** – Skripty mohou také komunikovat mezi sebou, i když jsou přiřazeny různým objektům. To umožňuje vytvářet složité interakce mezi objekty ve hře, jako je aktivace pasti, když hráč dosáhne určitého místa [39].

#### *3.1.3.2 Praktický příklad*

Níže je uveden jednoduchý příklad, který ukazuje, jak skript může změnit barvu objektu, když uživatel stiskne klávesu:

```
using UnityEngine;
public class ColorChanger : MonoBehaviour
{
   void Update()
   {
      if (Input.GetKeyDown(KeyCode.Space))
      {
         GetComponent<SpriteRenderer>().color = Color.red;
      }
   }
}
```

Tento skript nejprve čeká na stisk klávesy mezerníku (**KeyCode.Space**). Když je klávesa stisknuta, skript najde komponentu **SpriteRenderer** objektu, ke kterému je připojen, a změní jeho barvu na červenou.

### 3.1.4 Interakce se skriptovacím API Unity

Pro interakci s herním světem Unity jsou nezbytné skripty C#, které komunikují s rozsáhlým API Unity. Toto API umožňuje skriptům manipulovat s objekty ve scéně, zachytávat uživatelské vstupy, řídit zvuk a grafiku a mnoho dalšího. Skriptovací API poskytuje vývojářům nástroje pro vytváření sofistikovaných a interaktivních prvků, od jednoduchých klikatelných tlačítek až po komplexní herní mechaniky [37].

Efektivní práce s API zahrnuje využití předdefinovaných objektů, jako jsou **GameObject**, **Transform** a **Rigidbody2D**, a základních metod, jako **GetComponent** pro přístup k dalším komponentám objektů nebo **Instantiate** pro vytvoření nových instancí objektů. Naučení se, jak správně používat eventy a delegáty, je klíčové pro reagování na herní události v real-time aplikacích [37].

## 3.2 Rozhraní pro 2D v Unity

Unity je výkonný nástroj nejen pro 3D hry, ale také pro 2D vývoj. Jeho 2D prostředí poskytuje sadu nástrojů a funkcí speciálně navržených pro tvorbu 2D her, od platformových her po puzzle hry. Díky integraci fyzikálního enginu, podpory pro sprity a tilemapy, animace a uživatelské rozhraní, Unity umožňuje vývojářům snadno převést své nápady na hratelné hry [40].

### 3.2.1 Práce s 2D sprity a tilemapami

V Unity, sprity a tilemapy jsou základními stavebními bloky pro 2D hry, umožňující vývojářům vytvářet vizuálně bohaté a dynamické herní prostředí.

#### *3.2.1.1 Sprity*

Sprity jsou 2D grafické assety používané pro reprezentaci postav, objektů a scén. Unity podporuje snadný import spritů, kterým lze poté v editoru nastavit různé vlastnosti, jako jsou rozměry, pivot points nebo slicing. Po importu je možné sprity umístit přímo do scény nebo je dynamicky manipulovat pomocí skriptů C#, což vám umožňuje měnit jejich vzhled nebo chování během hry [41].

#### *3.2.1.2 Tilemapy*

Tilemapy v Unity umožňují vývojářům efektivně vytvářet a spravovat rozsáhlé 2D plochy pomocí opakujících se dlaždic (tiles). Unity nabízí Tilemap editor, s jehož pomocí lze snadno

definovat vzhled a strukturu herních světů, od jednoduchých pozadí až po komplexní herní úrovně s různými terény a překážkami. Tilemap systém také podporuje přidávání fyzikálních vlastností k objektům, což rozšiřuje možnosti pro design herních mechanik [42].

**Tiles a Tilesets –** Tiles jsou základními stavebníky bloky v Tilemap systému. Je možno je mít buď jednotlivě anebo v setech zvaných Tilesety. Tyto sety obsahují více Tiles a pro level design jsou mnohem efektivnější [43].

## 3.3 Komponenty v Unity

Komponenty v Unity jsou pilířem, které definují funkce, chování a vzhled herních objektů, zvaných **GameObjects**. Tyto komponenty nabízejí vývojářům možnost dodávat nové funkce objektům bez nutnosti psát komplexní kód od základu. Unity nabízí široké spektrum předdefinovaných komponent, jako jsou transformace, renderery, fyzikální komponenty a skripty [44].

### 3.3.1 Vytváření 2D animací pomocí Animator Controller

Animator Controller v Unity je mocný nástroj umožňující vytvářet složité animace pro 2D objekty. Pomocí tohoto nástroje je možno definovat stavy animace, přechody mezi stavy a podmínky, za kterých k těmto přechodům dojde, což vám umožní oživit postavy, objekty a další prvky ve vaší hře [45].
Animator Controller pracuje s tzv. "Animation Clips", což jsou jednotlivé animace, jako jsou chůze, skok nebo útok. Tyto klipy se poté přiřazují do stavů v rámci Animator Controlleru, který spravuje, kdy a jak budou tyto animace přehrávány [45].

#### *3.3.1.1 Vytvoření Animator Controlleru a práce s ním*

- Pro vytvoření nového Animator Controlleru je potřeba pravým tlačítkem kliknout na Project panel a vybrat **Create > Animator Controller**. Ten je pak nutno pojmenovat, ať už podle dané postavy nebo objektu, pro který se Controller vytváří [45].
- Následně je nutno mít přichystané různé snímky pro objekt, který se bude animovat. Ty se přetáhnu do okna Animator, kde se ke každé animaci vytvoří určitý stav spojený s konkrétní animací [45].
- Pro přidané animace lze poté definovat různé přechody, které zajistí, že změna z jedné animace na druhou proběhne plynule. K přechodům se může přiřadit

podmínka, která pro aktivaci daného přechodu musí být splněna. Těmito podmínkami mohou být například různé hodnoty proměnných anebo konkrétní událost v herním prostředí bez nutnosti psaní skriptů [45].

### 3.3.2 Transform a práce s 2D objekty

V Unity, komponenta Transform je základem pro umisťování, rotaci a škálování objektů ve 2D i 3D prostředí. Ve 2D je práce s Transform klíčová pro efektivní manipulaci s herními objekty, umožňující vývojářům snadno určit jejich polohu, orientaci a velikost [46].

Transform komponenta je přístupná a manipulovatelná jak přes Unity Editor, tak dynamicky pomocí skriptů. Skriptování umožňuje programově měnit pozici, rotaci a škálu objektů během runtime, čímž umožňuje efektivně testovat a nastavovat objekty na scéně [46].

#### *3.3.2.1 Základní transform komponenty*

- **Pozice –** Určuje umístění objektu ve scéně. Ve 2D hře se typicky pracuje s x a y souřadnicemi, kde x reprezentuje horizontální a y vertikální osu [46].
- **Rotace –** I ve 2D prostředí je možno otáčet objekty pro dosažení požadované orientace. Rotace se obvykle aplikuje vzhledem k ose z, což způsobí otáčení objektu kolem jeho středu [46].
- **Škálování –** Jedná se o změnu velikosti objektu. Škálování umožňuje objekt zvětšovat nebo zmenšovat bez nutnosti upravovat původní sprite. Ve 2D je možno škálovat pouze podle os x a y [46].

### 3.3.3 Renderer

V Unity renderery umožňují objektům být vizuálně reprezentovány na obrazovce. Tyto komponenty jsou klíčové pro definováné, jak objekty vypadají během hraní hry, a umožňují detailní nastavení grafických aspektů [47].

#### *3.3.3.1 Typy rendererů v Unity pro 2D*

Ve 2D projektech v Unity jsou nejčastěji využívány následující typy rendererů:

**Sprite Renderer –** Primární komponent pro vykreslení 2D obrázků a grafiky. Každý 2D sprite má přiřazen Sprite Renderer, který umožňuje kontrolu nad vizuálním zobrazením sprite v herní scéně [47].

**Line Renderer –** I když je často spojován s 3D grafikou, může být použit i ve 2D projektech pro vykreslení čas s různou tloušťkou a barvou. Je to užitečné například pro kreslení trajektorií nebo mapování cest [48].

**Trail Renderer –** Ve 2D prostředí se používá primárně pro tvorbu efektů, jakou jsou například stopy za pohybujícími se objekty, například za běžící postavou [49].

**Particle System Renderer –** Particle systémy, ať už ve 2D nebo 3D, jsou používány ke tvorbě efektů, kterými mohou být například exploze, oheň, dým či různorodé magické efekty [50].

**Canvas Renderer –** Pro všechy UI prvky, jakými jsou například tlačítka, scoreboardy, texty atd., se používá právě Canvas Renderer, který je součástí Canvas komponenty v Unity [51].

### 3.3.4 Kontroléry a správa pohybu

Kontroléry a správa pohybu jsou v Unity velmi důležité z hlediska implementace dynamických a plynulých pohybů postav a objektů napříč hrami. Umožňují totiž vývojářům definovat a spravovat různé typy pohybů, od jednoduchých posunů objektů až po komplexní animace a interakce [52].

Klíčovými komponenty pro pohyb jsou:

- **Rigidbody –** Pomocí této komponenty můžeme přidat fyzikální vlastnosti k objektům, čímž jim umožníme pohyb dle zákonů fyziky. Rigidbody je nezbytné pro simulaci realistických reakcí na síly a kolize [53].
- **Character Controller –** Alternativa Rigidbody pro pohybové účely, kdy není potřeba manuálně zpracovávat fyziku nebo detekci kolizí [54].
- **NavMesh Agent –** Automatizace pohybu a cest, kdy Unity za využití umělé inteligence umožňuje objektům se inteligentně pohybovat v prostředí za účelem najít nejvhodnější cestu [55].

## 3.4 Fyzika v 2D prostředí

Z hlediska realističnosti a interaktivity je fyzika u valné většiny 2D her stejně nutná jako u 3D her. Fyzikální engine v Unity zvládne zpracovat vše, od jednoduchého pádu objektu po složité interakce mezi více objekty, jakými mohou být například srážky či odrazy objektů.

Pro správné použití fyziky je důležité zvážit, jaké vlastnosti jsou pro danou hru klíčové, aby nedošlo k negativnímu ovlivnění výkonu hry [56].

### 3.4.1 Rigidbody2D

V předchozí kapitole jsme se dočetli, že pro pohyb objektů v závislostech na fyzikálních zákonech, například hmotnost či tření (drag), je možno využití komponenty jménem Rigidbody. Ve 2D prostředí se využívá jeho alternativy – **Rigidbody2D**, které je aplikováno na 2D sprity. Jeho implementace je nezbytná v případě, že hra využívá mechanik jako je skákání, střelba objektů, zničitelné prostředí, ale i obyčejný pohyb v platformových hrách. V těchto případech Rigidbody2D může ovlivnit výšku, do které může objekt vyskočit, vzdálenost a úhel letu objektu, či závislost kolizí objektů pro správnou interakci zničení. Nicméně všechny tyto interakce musí mít správně nastavené hodnoty u parametrů, aby nedocházelo k extrémním případům – gravitace má větší hodnotu než síla skoku a postava se tím pádem nepohne ani o jeden pixel nahoru [57].

### 3.4.2 Kolize a její detekce

Pokud je potřeba definovat reakci objektu při kontaktu s jiným, je na místě využití kolizí – v Unity dosáhnuto použitím Collider 2D. Nejenže mohou znemožňovat například objektu hráče procházet skrze neprůchozí stěny tak, aby se hráč nedostal mimo hrací plochu, kolize mohou spouštět různorodé eventy (události) po střetu objektu s daným místem, kde se právě tady ta kolize nachází. Příkladem mohou být například pasti, které se spustí, jakmile se hráč ocitne na určitém místě, nebo eventuálně sbírání předmětů, přes které hráčský objekt přejde [58].

Aby kolize fungovaly, je potřeba je detekovat. Toho se dosáhne nasazením Collideru 2D, který je pro hráče neviditelný, na 2D sprite. Nemusí odpovídat přesnému vzhledu spritu, většinou se používá ve tvarech obdelníku nebo čtvece. Díky tomu pak engine ví, kde přesně kolize s různými objekty vznikají a co má při kolizi nastat – například pomocí funkcí **OnCollisionEnter2D**, **OnCollisionStay2D**, **OnCollisionExit2D** [58].

### 3.4.3 Fyzikální materiály ve 2D

Pro autentické chování různých objektů je potřeba znát, o jaký typ materiálu se jedná. Díky tomu je pro vývojáře jednodušší nastavení jeho hodnot. Například ledová plošina bude mít vyšší kluzkost – menší tření než obyčejná travnatá plocha, která ho bude mít naopak větší. To bude platit i pro například kovový či fotbalový míč, kdy jeden bude méně pružnější než

ten druhý. Z toho důsledku se pak při kolizi kovový míč neodrazí tak moc, jako ten fotbalový [59].

## 3.5 Zvuk a hudba v Unity

Každá hra je lepší, když je v ní zvuková a hudební stopa, ať už na pozadí nebo při akci hráče. Unity má integrovaný audio systém, který umožňuje právě tyto prvky jednoduše přidávat a manipulovat s nimi tak, aby byl herní zážitek hráče mnohem lepší [60].

### 3.5.1 Audio komponenty

V Unity je k dispozici několik komponent pro základní práci s vloženým audiem, jako jsou **AudioSource**, který slouží k přehrávání zvuku, a **AudioListener**, který se většinou přichytává přímo ke kameře a sám o sobě zachytává všechny aktuální zvuky z herního prostředí. Díky tomu mohou být různým být specifickým objektům přiřazeny různé zvukové efekty. Příkladem může být chůze postavy po různém povrchu [60] [61].

Pro přidání efektů, faderů nebo různých zvukových kanálů je možnost použít **AudioMixer**, který umožňuje v reálném čase tyto zvukové efekty upravovat [62].

## 3.6 Testování a ladění

Pro zvýšení kvality konečného produktu je důležitým krokem mít hru otestovanou. Pomocí testování a ladění dosáhneme stability hry a zbavíme ji nechtěných bugů, které by mohly negativně ovlivnit hráčský zážitek.

### 3.6.1 Testování

Pro testování v Unity může být použito nástroje Unity Test Runner, pomocí kterého může programátor testovat v reálném čase přímo ve vývojovém prostředí. Tento nástroj využívá framework NUnit pro definici a spuštění testů [63].

### 3.6.2 Ladění

Pro identifikaci a řešení problémů ve skriptech a v herním chování je v Unity možnost použití různých forem ladění. Jednou může být konzole, která zaznamenává chyby, varování a další důležité informace, které mohou způsobovat problémy. Taktéž je možnost použít debugger, který je součástí Unity Editoru a umožňuje krokování kódu a inspekci všech proměnných v reálném čase [64].

# 4 PROCES VÝVOJE HER

Jedná se o velice rozsáhlou a multidisciplinární oblast, která integruje různé aspekty technologie, umění a designu. Je charakteristická svou dynamikou a neustálým pokrokem, což umožňuje velmi časté inovace a rozvoj nových technik a metod. V praxi se jedná o širokou škálu disciplín, včetně základního designu, grafického zpracování, programování anebo testování, což v konečném důsledku vyžaduje komplexní pochopení jak technických, tak kreativních prvků [65].

## 4.1 Dokumentace vývoje her

Aby bylo možno na projektu vůbec začít, je třeba si prvně vytvořit její dokumentaci. Ta slouží jako taková navigační mapa pro vývojářský tým a zajišťuje, že všechny aspekty projektu jsou jasně definovány a aktivně je na ně brán zřetel. Dokumentace by neměla být podceňován, jelikož poskytuje strukturu a efektivní práci týmu [65].

### 4.1.1 Účel dokumentace

Zásadním účelem dokumentace vývoje hry je zaznamenání všech klíčových informací potřebných pro plánování, realizace a testování hry. Takto dokumentace zahrnuje technické specifikace, designové dokumenty, plány testování a uživatelské příručky. V případě dobře strukturované dokumentace má vývojářský tým k dispozici kvalitní podklady pro zorientování se v práci a referenční bod pro eventuální budoucí úpravy a iterace projektu [65].

### 4.1.2 Typy dokumentace

- **Designový dokument hry (GDD) –** Základní dokument pro každý herní projekt. Obsahuje detailní popis herní světa, pravidel, postav, cíle hry, herní mechaniky a příběh. S tím, jak se vývoj hry posouvá dopředu, se GDD často vyvíjí a mění [65].
- **Technická dokumentace** – Zahrnuje specifikace software, hardwaru a technické požadavky – engine, síťové technologie, databáze a integrační nástroje. Důležitá je pro programátory a vedoucí týmu [65].
- **Dokumentace uživatelského rozhraní (UI)** – Jsou v ní obsaženy plány a mockupy pro UI projektu, včetně menu, ovládacích prvků a jakékoliv informace zobrazované hráčům a konečným uživatelům. Tato dokumentace je klíčová pro designéry UI/UX [65].

- **Plány vývoje a časové osy** – Zahrnují detailní plány, kdo co a kdy dělá, zásadní milníky projektu a harmonogramy. Pomáhají celému týmu udržet projekt v časových a rozpočtových limitech [65].

## 4.2 Návrh výukové hry

V případě návrhu výukové hry je třeba brát v potaz pečlivé propojení pedagogických cílů a principy herního designu tak, aby výsledný produkt byl nejen funkční z programátorského hlediska, ale i toho výukového. Při procesu je třeba dbát na: [66]

- **Definici vzdělávacích cílů** – Jde o první a základní krok, kdy je potřeba si jasně definovat, jaký je vůbec vzdělávací cíl. Tím myšleno, co by měla hra hráči po jejím dokončení přinést. Mělo by se jednat o jasně specifické a měřitelné cíle, které jsou přímo spojeny s požadovanou učební osnovou. Zároveň musí být cíle realistické, dosažitelné a musí být možné je implementovat do herního prostředí [66].
- **Výběr herního žánru** – Taktéž je důležité se zaměřit, o jaký žánr hry se bude jednat a zda daná látka či učivo je možné v daném žánru nějakým způsobem předat. Všeobecně se dá říct, že puzzle hry jsou asi nejčastějším žánrem využívajícím určitou formu výuky. Ty se ale pak mohou kombinovat například s adventurami či RPG hrami [66].
- **Vytvoření herního designu** – Ve fázi tvorby herního designu jde o to navrhnout základní herní mechaniky a pravidla, která jsou nejen zábavné, ale také efektivně implementují konkrétní vzdělávací cíle z prvního bodu. Herní design by měl zahrnovat interakce, které jsou intuitivní a přímo podporují proces učení formou opakování, zpětné vazby a postupného zvyšování obtížnosti [66].
- **Vývoj prototypu** – Pro testování a iteraci herních nápadů je ideální mít určitý prototyp hry, což je zjednodušený model finální hry, který umožňuje vývojářům či pedagogům testovat, jak herní mechaniky a učební proces funguje v praxi. Měl by obsahovat základní funkce a funkční gameplay, na kterém lze hrubě otestovat hlavní myšlenku a na základě zpětné vazby by se měl následně upravit do finální podoby [66].
- **Grafický a audiovizuální design** – Grafická stránka výukové hry by měla být přitažlivá a vhodná pro věkovou skupinu cílového publika. Velká pozornost by se

měla věnovat čitelnosti textu anebo grafům či jiným věcem v rámci učeného tématu. Vždy musí být jasné, na co se uživatel dívá a co čte [66].

- **Implementace a testování** – Při implementaci je důležité zvolit vhodnou platformu a nástroje právě pro vývoj konkrétní hry. Během fáze implementování je kritické průběžně testovat hru, aby se zajistilo, že je po technické stránce stabilní, funkční a že splňuje všechny konkrétní požadavky. Testování by mělo zahrnovat jak interní týmy lidí, tak testy přímo s cílovou skupinou uživatelů. Zpětná vazba je u testování nejdůležitější, jelikož se v této fázi mohou objevit bugy, které je nutno co nejrychleji opravit, anebo můžou přijít nové nápady, které by finální hru mohly udělat mnohem zajímavější [66].
- **Iterace a zlepšení** – Oprava bugů a implementace zajímavých nápadu přichází právě ve fázi iterace, kde se vezme všechna zpětná vazba a herní mechaniky se začnou ladit [66].
- **Spuštění a hodnocení** – Konečným krokem je pak spuštění hry a její nasazení do vzdělávacího prostředí. Po jejím spuštění je důležité monitorovat a vyhodnocovat, jakým způsobem hra ovlivňuje učení a zapojení dané skupiny lidí. Shromažďování a analýza těchto dat může poskytnout důležité informace potřebné pro další zlepšování hry nebo pro vývoj budoucích projektů [66].

### 4.2.1 2D Hry

2D herní žánr se týká videoher, které jsou vizuálně reprezentovány ve dvourozměrném prostoru. To v praxi znamená, že všechny grafické prvky hry (postavy, prostředí či objekty) jsou vykresleny na jedné rovině s výškou a šířkou, ale bez hloubky – tudíž pouze osy X a Y. Ve 2D hrách se objekty a postavy obvykle pohybují pouze vodorovně (zleva doprava nebo zprava doleva), anebo svisle (shora dolů či zdola nahoru) [67].

Do 2D her mohou spadat různé herní žánry:

- **Platformovky** – V tomto žánru jde o to pouze skákat mezi platformami a zdolávat různorodé překážky. Příkladem může být Super Mario Bros.
- **Adventury** – Důraz je kladen na průzkum herního světa a vyprávění příběhů, například česká herní série Polda
- **Puzzle hry** – Kde je cílem řešit hlavolamy všech možných druhů. Jednou z nejznámějších je hra Tetris.

- **Bojové hry** – Jak název napovídá, v těchto hrách jde převážně o boj mezi hráčem a počítačem. Spadá sem například Street Fighter.
- **Střílečky** – V těchto hrách je hlavním cílem zneškodnit nepřítele primárně střelbou. Jednoznačným příkladem bude Space Invaders.

### 4.2.2 RPG

RPG, zkratka pro role-playing game (hra na hrdiny), je žánr videoher, kde hráči přebírají role fiktivních postav a prožívají různé příběhy v často rozsáhlých světech. Hráči mají většinou možnost ovlivnit vývoj jak postav, tak samotného příběhu svými rozhodnutími, které se mohou odrazit například v morálních rozhodnutích či vztazích s ostatními postavami. RPG hry jsou typicky charakterizovány složitým příběhem, hlubokým vývojem postav a světa, a často obsahují prvky v podobě úkolů, boje, průzkumu světa a interakce s dalšími nehráčskými či hráčskými postavami [68].

RPG hry mohou být rozděleny do několika subžánrů, včetně:

- **Tradiční RPG** – Tyto hry jsou silně inspirovány stolními RPG, jakou jsou Dungeons & Dragons, kde hráči přebírají role hrdiny, vylepšují si statistiky svých postav a sledují lineární příběhovou linku. Asi nejlepším příkladem bude relativně nová hra Baldur's Gate.
- **Akční RPG** – Tyto hry jsou v základu to samé, jako tradiční RPG, ale jsou obohaceny akčnějšími herními mechanikami. Příkladem bude například polský Zaklínač.
- **JRPG (Japanese RPG)** – Specifický styl RPG pocházející z Japonska. Velmi populární mezi hráči, kdy klade hodně velký důraz na příběhovou linku a často obsahuje turn-based bojový systém. Nejznámější hrou z tohoto subžánru bude Final Fantasy.
- **MMORPG (Massively Multiplayer Online RPG)** – Online hry, kde tisíce hráčů zároveň mohou spolu hrát, prozkoumávat svět a interagovat s ním. Tady bych jako příklad uvedl jedno z neumírajících MMORPG, World of Warcraft.

# 5 OBDOBNÉ HRY NA TRHU

Na internetu se dá sehnat mnoho různých naučných her, konkrétně i těch, které se specializují na programování. Většinou jsou hry mířeny na nově začínající programátory, tudíž obsahují kompletní základy daného jazyka a na ty nabalují více nových informací, které jsou potřebné k postupu ve hře.

Mezi hry, které mají za cíl přiblížit a naučit hráče programování, můžeme zařadit například:

### 5.1.1 CSS Diner

Jedná se o interaktivní webovou aplikaci, která učí uživatele základům a pokročilým technikám CSS selektorů. Je navržena jako série úrovní, v nichž každá jedna úroveň přestavuje specifickou výzvu, která souvisí s výběrem různých prvků HTML webové stránky pomocí CSS selektorů. Účelem hry je pomoci uživatelům naučit se, jakým efektivním způsobem využívat selektorů k určení a stylizaci specifických prvků na webové stránce.

CSS Diner funguje následovně:

1. Uživatelé vidí na své obrazovce jednoduché zobrazení struktury HTML, která vizuálně připomíná jídelní stůl s různými HTML prvky, jakými jsou například talíře, příbory a hrníčky.
2. Každá úroveň uživateli předloží specifický úkol pro výběr určitých HTML prvků pomocí CSS selektorů. Příkladem může být vybrání jednoho konkrétního talíře, například toho nejmenšího, a na hráči je pak, aby napsal správný CSS selektor, který tento prvek vybere.
3. Hráči zadávají své CSS selektory do vstupního pole a hra na jejich vstup okamžitě reaguje. Hráč v reálném čase vidí, jak jeho selektory dané prvky na zmíněném jídelním stole ovlivňuje.
4. V případě zadání správného CSS selektoru se hráč posouvá na další úroveň. Hra se postupně stává složitější skrze komplikovanější selektory a jejich kombinace, což hráčům umožňuje postupně zlepšovat jejich dovednosti. K dispozici na oficiálním repozitáři GitHubu [69].

### 5.1.2 CodeCombat

CodeCombat je interaktivní platformou pro výuku programování v prohlížeči. Hra je navržena tak, aby sloužila jako zdroj informací a zkušeností pro studenty, učitele či jednotlivce, kteří mají zájem o zlepšení svých programovacích dovedností pomocí interaktivního a gamifikovaného prostředí.

Hlavními rysy CodeCombat jsou:

1. Využívání RPG prvků, kde hráči ovládají svého hrdinu v dobrodružném fantasy světě. Zadané úkoly a mise jsou vyřešeny systémem psaní skutečného kódu, který umožňuje se postavě navigovat přes úrovně, bojovat s nepřátelskými postavami a řešit logické hádanky.
2. CodeCombat nabízí výuku vícera programovacích jazyků, včetně například JavaScriptu a Pythonu, které jsou také velmi populární a široce rozšířené v programování a IT průmyslu.
3. CodeCombat je strukturovaný do řady úrovní, kde každá má vlastní cíle a výzvy. Tím pádem si hráči postupně rozšiřují vědomosti daného programovacího jazyka, jako jsou například struktury, funkce, smyčky a další základní programovací koncepty, zatímco postupují skrze herní úrovně.
4. CodeCombat je vhodnou implementací i do výuky na školách, jelikož učitelům a profesorům nabízí speciální dashboard, ve kterém mohou sledovat pokrok studentů, plány následujících lekcí a další zdroje pro výuku, které usnadňují integraci této platformy do výukového plánu.
5. Zároveň CodeCombat nabízí možnost interakce mezi vícero hráči, kteří se mohou účastnit různých soutěží, které tato platforma pravidelně pořádá. Tyto akce podporují motivaci pro další rozvoj dovedností.

CodeCombat má pár prvních úrovní k dispozici zdarma. V případě, že se uživateli líbí, jakým stylem je hra podána a vyhovuje mu vizuál a koncept předávání informací, může si za určitý poplatek koupit přístup k celé hře. CodeCombat nabízí možnost předplatného buď jednou za měsíc či jednou za rok. Aplikace k dispozici na oficiálních stránkách CodeCombat [70].

### 5.1.3 Scratch

I když se nejedná přímo o hru, hodí se zmínit interaktivní platformu Scratch, která byla vytvořena týmem na MIT (Massachusetts Institute of Technology) s cílem umožnit především dětem a začátečníkům snadno se dostat do základů programování. Scratch umožňuje jeho uživatelům vytvářet vlastní interaktivní hry nebo animaci pomocí vizuálního programovacího jazyka, který je založen na systému Drag and Drop neboli přetahování bloků kódu a jeho skládání na sebe v logické posloupnosti. Tato metoda uživatelům umožňuje vizuálně sestavit logiku programu bez nutnosti psaní tradičního textového kódu.

Základní charakteristiky platformy Scratch jsou:

1. Programování ve Scratchi probíhá přetahováním a spojováním různorodých bloků. Každý blok má jinou funkcionalitu, ale všechny reprezentují určité programové struktury, jakou jsou například smyčky, podmínky, proměnné a další funkce. Díky tomu je platforma ideální pro začátečníky.
2. Vytvořené projekty uživatelů mohou být sdíleny přímo uvnitř platformy. Ty mohou být okomentovány od ostatních uživatelů. Scratch má velmi aktivní a podporující komunitu, což může být pro začátečníky a mladé programátory velmi motivující.
3. Kromě nápomocné komunity Scratch obsahuje mnoho vzdělávacích materiálu, jakými mohou být například návody či lekce pro učitele. Díky těmto zdrojům je jednoduché Scratch implementovat do samotné výuky na školách.

Scratch je dostupný zdarma, což z něj dělá snadno dostupný nástroj pro vzdělávání ať už doma, tak na školách. Je navržen tak, aby poskytoval intuitivní a zábavný způsob vzdělávání se v oblasti programování. Aplikace je k dispozici na oficiálních webových stránkách [71].

### 5.1.4 Tynker

Jedná se o vzdělávací webovou platformu, která funguje na obdobném principu jako předchozí Scratch. Je vhodnou volbou pro děti a začínající programátory, s cílem poskytnou zábavný a poutavý způsob, jak se naučit programovat za použití herních prvků.

Důležitými rysy platformy Tynker jsou:

1. Využívá se stejně jako u platformy Scratch blokového programování, kde si uživatelé skládají jednotlivé bloky kódu. Toto umožňuje snadnou vizualizaci toho, co kód vlastně dělá a pomáhá pochopit základní koncepty programování, kterými jsou

smyčky, proměnné, podmínky či funkce, aniž by se uživatel musel přímo setkat se syntaxí textového programování.

2. Tynker nabízí velice rozsáhlé spektrum kurzů a projektů, které pokrývají různé tematické oblasti, včetně herního designu, animace, robotiky a webového vývoje. Tyto kurzy jsou navrženy tak, aby pomáhaly ve vzdělávání a zároveň si zanechaly určitou formu zábavy, což je klíčové pro motivaci v používání pokračovat.
3. K dispozici má také Tynker komplexní nástroje pro učitele, včetně plánů lekcí, průvodců učivem a dashboard, který slouží pro sledování pokroku studentů. Díky tomu je integrace Tynkeru do školního prostředí jednoduchá a umožňuje efektivněji spravovat výuku programování v hodinách.
4. Kromě předdefinovaných lekcí a kurzů Tynker nabízí uživatelům možnost vytvářet vlastní projekty, což podporuje kreativitu a inovaci. Platforma taktéž podporuje přechod od blokového programování k psaní tradičního textového kódu v jazycích, jakými je Python a JavaScript.
5. Tynker také nabízí integraci s různými hardwarovými platformami, jakými mohou být například drony. To umožňuje studentům aplikovat své programovací dovednosti v reálných technologických projektech.

Základní forma platformy je zdarma ke zkoušce, následně je na uživateli, zda se rozhodne platit předplatné ve formě půl roku, roku, či si zaplatí plnou verzi bez dalších měsíčních poplatků. Aplikace je k dispozici na oficiálních webových stránkách [72].

### 5.1.5 CodinGame

CodinGame je jedna z dalších webových platforem, která umožňuje programátorům všech úrovní zlepšovat své dovednosti prostřednictvím hraní her a řešením programovacích výzev. Platforma nabízí unikátní přístup k učení a zdokonalování se v programování tím, že programovací koncepty zavádí do interaktivních her, kde úkoly vyžadují psaní skutečného kódu pro jejich splnění.

Důležitými vlastnostmi CodinGame jsou:

1. Podporuje více než 25 programovacích jazyků, včetně těch populárních, jakými jsou Python, Java, C++ či dokonce C#. Díky tomu umožňuje programátorům pracovat v jazyce, který jim je blízký, anebo si mohou vybrat nějaký, který by se chtěli naučit.

2. Nabízí obsáhlou herní knihovnu a širokou škálu výzev, od jednoduchých úkolů pro úplné začátečníky až po komplexní algoritmické hádanky a multiplayerové soutěže pro ty pokročilejší. Hry jsou navrženy způsobem, který testuje a rozvíjí aspekty programovacích dovedností, jako jsou algoritmické myšlení, správa datových struktur, optimalizace a mnoho dalších.
3. Motivací můžou být také online soutěže, které CodinGame nabízí. Tyto soutěže jsou skvělou motivací pro stálé zlepšování se jednotlivců.
4. Silná komunita je součástí CodinGame, kde funguje velmi aktivně fórum či online chaty, ve kterých si uživatelé mohou sdílet své kódy, řešení různých výzev nebo získávat zpětnou vazbu na své nápady.
5. Mimo jiné, CodinGame nabízí nástroje pro profesní rozvoj. Služba CodinGame for Work poskytuje programátorům možnost se spojit přímo s potenciálními zaměstnavateli dle jejich zkušeností, které prokázali používáním platformy. Nicméně se jedná o placenou službu, která má měsíční poplatky, ale se zaručeným spojením se zaměstnavateli.

CodinGame je k dispozici na oficiálních webových stránkách [73].

# II. PRAKTICKÁ ČÁST

# 6 KONCPEPT HRY

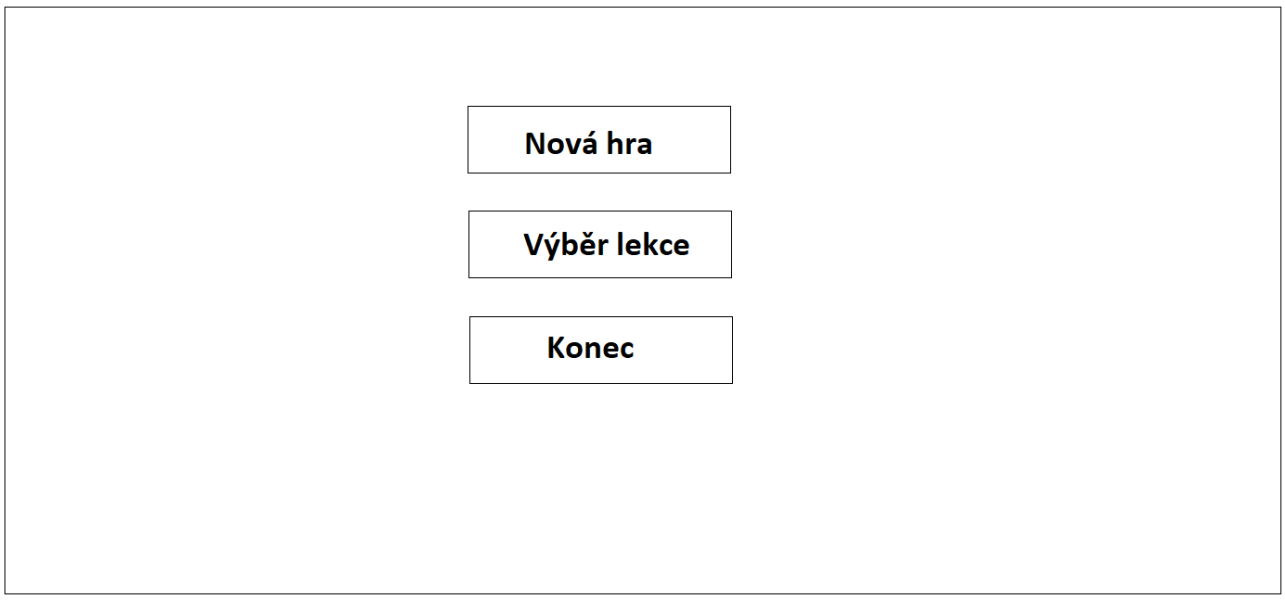
Základem této naučné 2D RPG hry je propojení zábavných herních prvků s edukativním obsahem. Hra je navržena tak, aby hráče nenásilně provázela virtuálním světem. V této kapitole popíšeme herní mechaniky, které budou v další kapitole implementovány přímo do hry.

## 6.1 Herní mechaniky

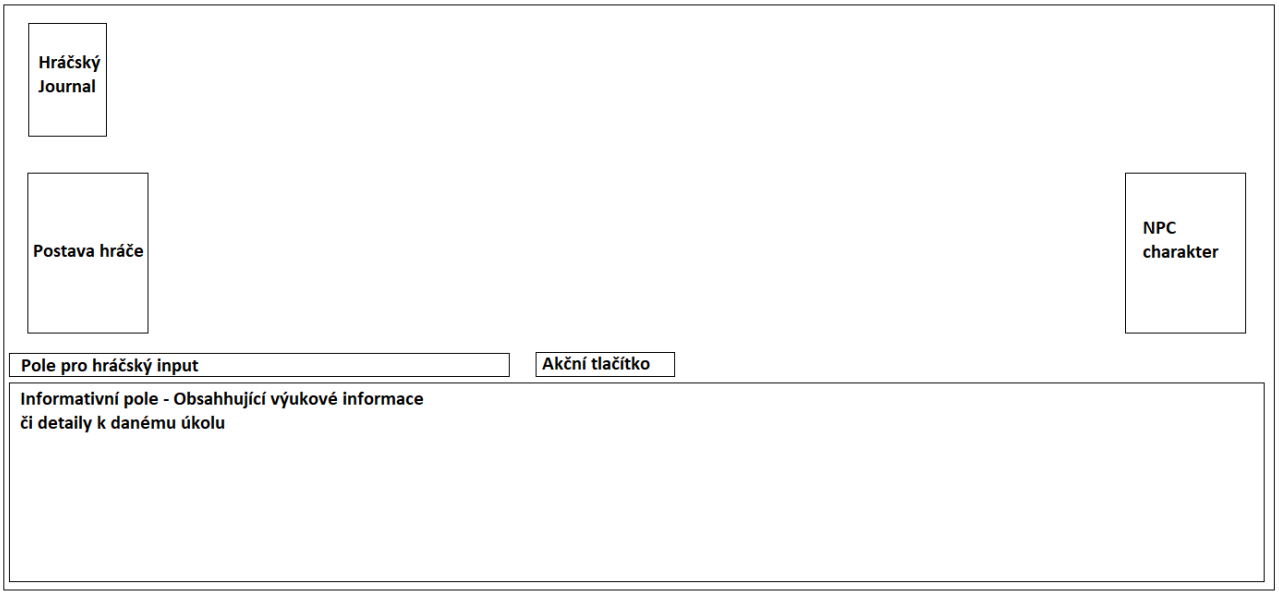
Hlavní herní mechanika spočívá v plnění různých úkolů a kvízů, které hráč dostává v průběhu svého průchodu hrou. V určitých částech hry narazí na NPC, se kterým může vést konverzaci a také od něj dostane hodnotné informace v podobě nových programovacích zkušeností, které následně jsou otestovány určitou formou kvízu nebo souboje. Hráč dle zadání vkládá své inputy do textového pole a následně záleží, zda byli zadány korektně či ne. V obou případech bude hráč informován. Všechny nově naučené zkušenosti má hráč uloženy ve svém Journal systému, který si může kdykoliv otevřít a podívat se zpětně na informace, které používal s využitím v reálném programování.

## 6.2 Grafický návrh vyvíjené hry

Pro tvorbu grafického návrhu hry byla použita aplikace Malování vyvíjena společností Microsoft Corp. Jako návrh byla vytvořena hlavní obrazovka (Obr. 9), hlavní hrací plocha (Obr. 10) a hráčský Journal (Obr. 11).




Obrázek 9 Návrh hlavního menu




Obrázek 10 Návrh hlavní hrací obrazovky


Obrázek 11 Návrh hráčského Journalu

## 6.3 Funkční a nefunkční požadavky

V procesu návrhu hry bylo nezbytné podrobně specifikovat jak funkční, tak nefunkční požadavky, aby bylo zajištěno efektivní a hladké fungování aplikace.

| Funkční požadavky | | |
|---|---|---|
| **ID** | **Požadavek** | **Popis** |
| FR1 | Spuštění aplikace | Hra lze stisknutím tlačítka „Nová hra“ spustit. |

| | | |
|---|---|---|
| FR2 | Načtení levelu | Lze si vybrat úroveň pomocí tlačítka „Výběr lekce“. |
| FR3 | Ukončení aplikace | Hra lze stisknutím tlačítka „Konec“ ukončit. |
| FR4 | Uživatelské rozhraní | Hra má intuitivní uživatelské rozhraní. |
| FR5 | Hráčský input | Hra umožňuje uživateli využití textového vstupu přímo ve hře. |
| FR6 | Vzdělávací moduly | Hra poskytuje vzdělávací moduly pro jazyk C#. |
| FR7 | Interaktivní úkoly | Hra poskytuje integrované úkoly a kvízy pro aktivní testování znalostí hráče. |
| FR8 | Zpětná vazba | Hra poskytuje okamžitou zpětnou vazbu na akce uživatele. |
| **Nefunkční požadavky** | | |
| **ID** | **Požadavek** | **Popis** |
| NFR1 | Multiplatformní podpora | Hra je kompatibilní s různými operačními systémy. |
| NFR3 | Výkon | Hra je optimalizována pro plynulý běh na různých zařízeních. |
| NFR4 | Rozšiřitelnost | Hra je rozšiřitelná o další programovací jazyky a vzdělávací moduly. |

Tabulka 1 Funkční a nefunkční požadavky

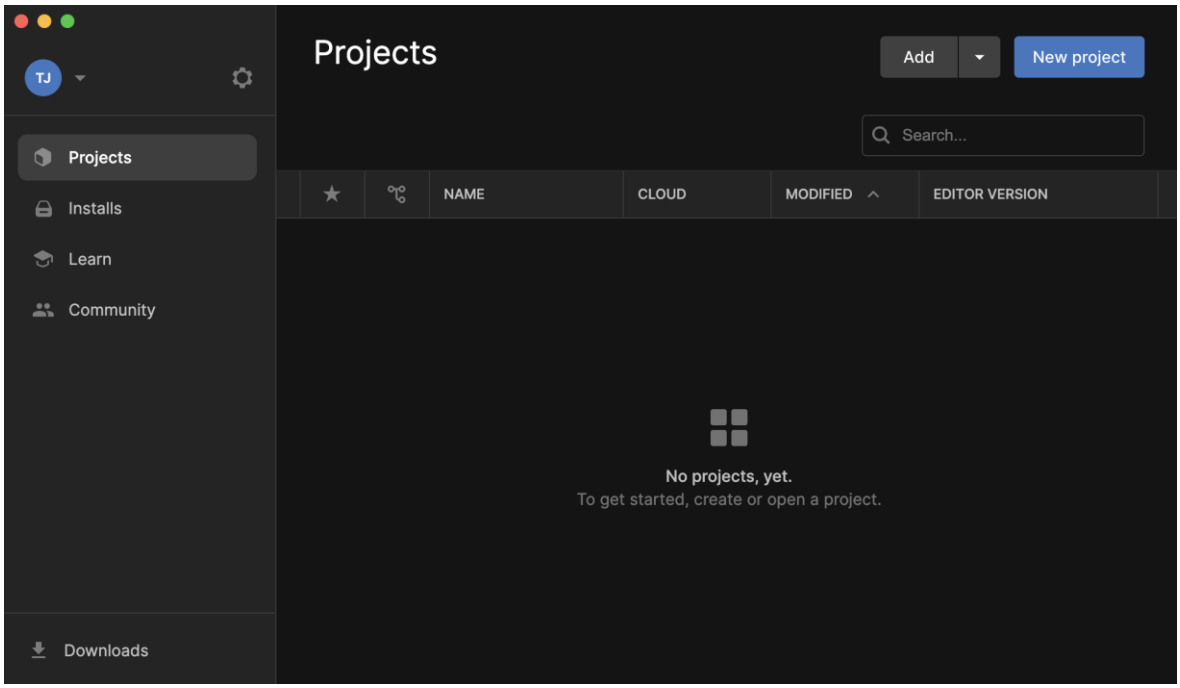
# 7 PRÁCE S ENGINEM UNITY

Pro vytvoření naučné 2D RPG hry bylo zvoleno Unity jako nejvhodnější kandidát, jelikož nabízí komplexní a robustní nástroje pro tvorbu 2D her s aktivní uživatelskou komunitou.

## 7.1 Stáhnutí a instalace Unity

Pro stáhnutí Unity je potřeba otevřít jejich webovou stránku [74], kde je možno si zvolit ze tří různých variant Unity. V případě, že se nejedná o vývojářskou firmu, tak je nejvhodnější první varianta, která je zcela zdarma a její používání není ničím omezeno [75].

Po vybrání varianty ke stažení je uživatel přesměrován na webovou stránku, kde si může stáhnout Unity Hub (Obr. 12) pro svou platformu, ať už jí je Windows, Linux nebo MacOS. Unity Hub slouží jako samostatná aplikace, ve které uživatel vidí všechny své projekty a může v ní spravovat nainstalované verze Unity [76].




Obrázek 12 Hlavní obrazovka Unity Hub

Následně se po výběru začne stahovat konkrétní varianta Unity Hubu do počítače. Dále je spuštěn instalační soubor a poté stačí následovat instrukce instalace. [76]

Při prvním stažení a instalaci, Unity Hub spustí uvítací obrazovku, ve které je možností se buď přihlásit pod Unity ID, nebo si vytvořit nové. Pro práci v Unity je toto ID nutno mít [75].

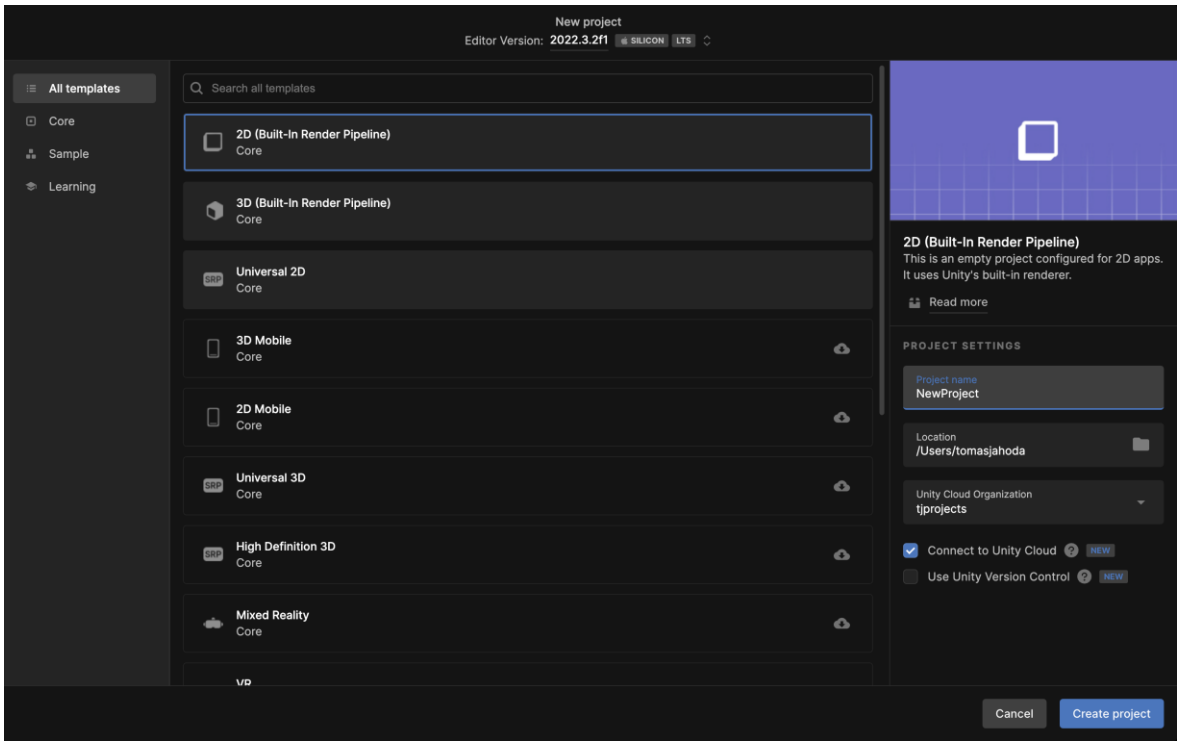
### 7.1.1 Systémové požadavky

Pro správnou funkčnost, aplikace Unity Hub podporuje následující operační systémy:

- Windows 7 SP1+, 8, 10 (64-bitová verze), 11
- macOS X 10.13+
- CentOS 7
- Rocky
- Ubuntu 18.04, 20.04, 22.04[75]

## 7.2 Tvorba projektu

Jakmile je Unity Hub nainstalován a uživatel je přihlášen, stačí v pravém horním rohu kliknout na tlačítko New Project, které otevře okno se základním nastavením projektu (Obr 13). Na výběr má z více možností šablon.

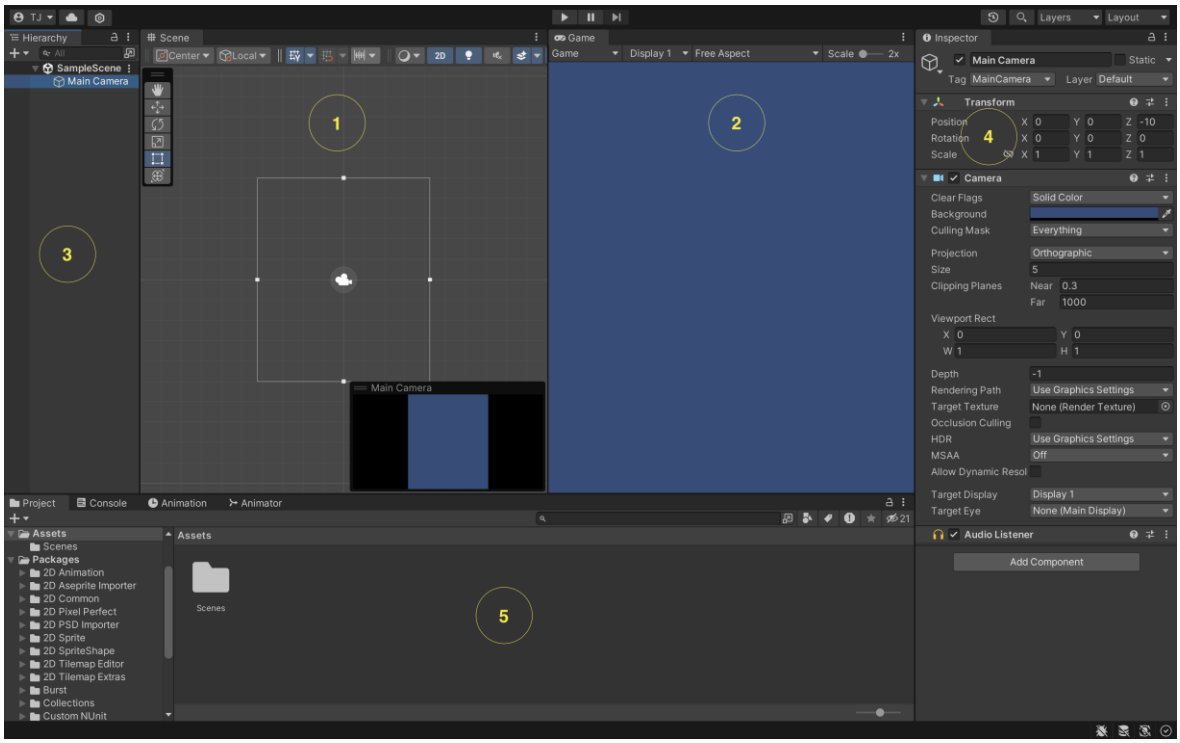



Obrázek 13 Tvorba nového projektu

Na obrázku jde vidět, že pro vytvářený projekt jsem zvolil základní 2D šablonu. Následně je nutno nový projekt pojmenovat a vybrat složku, do které se náš projekt uloží. Jakmile máme vše připraveno, klikneme na Create Project.

## 7.3 Unity UI

Po chvilce se nám otevře úvodní obrazovka našeho nového projektu (Obr. 14), která je rozdělena do několika částí, kdy každá slouží k něčemu jinému.




Obrázek 14 Unity UI

1. Část Scene (Scéna), která slouží k vizuálním úpravám hry. Vývojář může do Scény vkládat herní objekty, se kterými pracuje. Může s nimi libovolně pohybovat v rámci 3D prostoru.

2. Game view přebírá objekty ze Scény a ukazuje vývojáři, jakou bude mít jeho projekt finální podobu skrze kamery ze Scény. Je možno spustit simulaci hry skrze tlačítko Play v levém horním rohu Game view okna.

3. Hierarchie obsahuje všechny herní objekty, se kterými vývojář pracuje na konkrétní Scéně. Hierarchie dovoluje herní objekty rozdělit do skupin pro lepší přehlednost. Kromě objektů může Hierarchie obsahovat i jiné scény, mezi kterými se dá následně jednoduše navigovat.

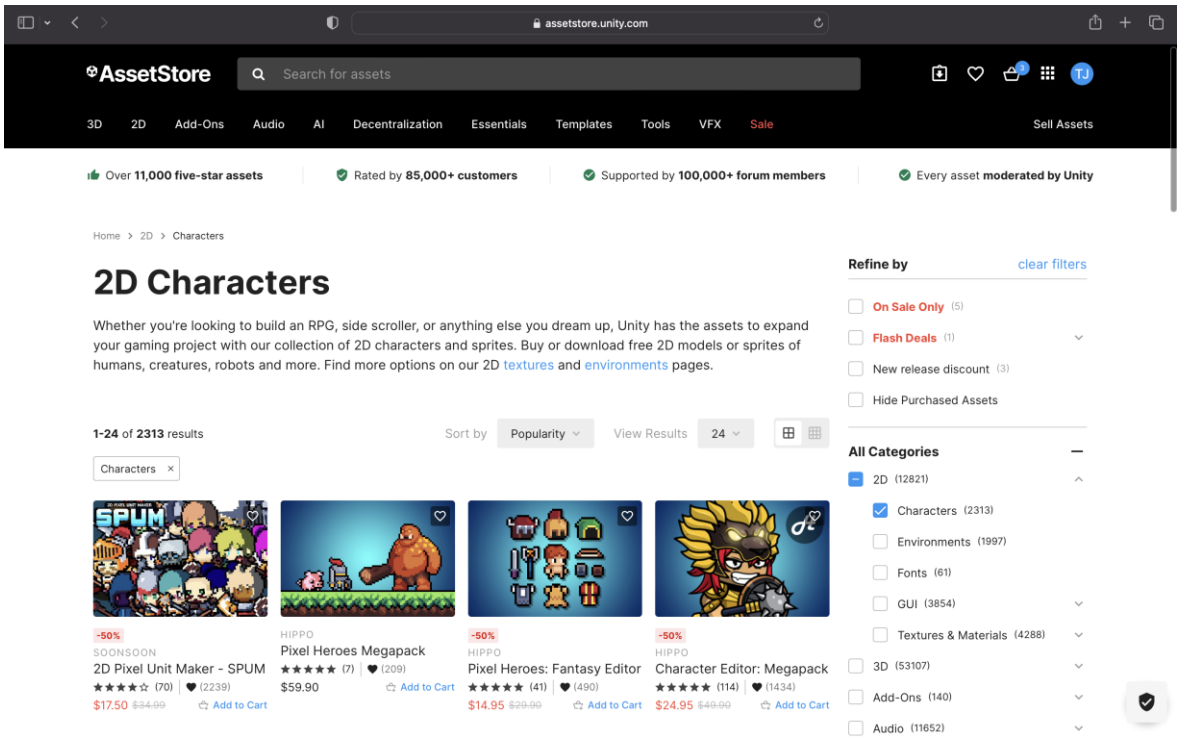
4. Inspector slouží k zobrazení a úpravě všech vlastností aktuálně vybraného objektu. Každý objekt má jiné vlastnosti a tím pádem bude Inspector pro každý objekt vypadat jinak.

5. Project zobrazuje všechny složky komponenty, které jsou aktuálně v projektu. Lze zde najít například všechny importované grafické sprity, animace, skripty či vytvořené scény. Přístup ke všem komponentům přímo v Unity výrazně usnadňuje práci na projektu.

## 7.4 Tvorba hry

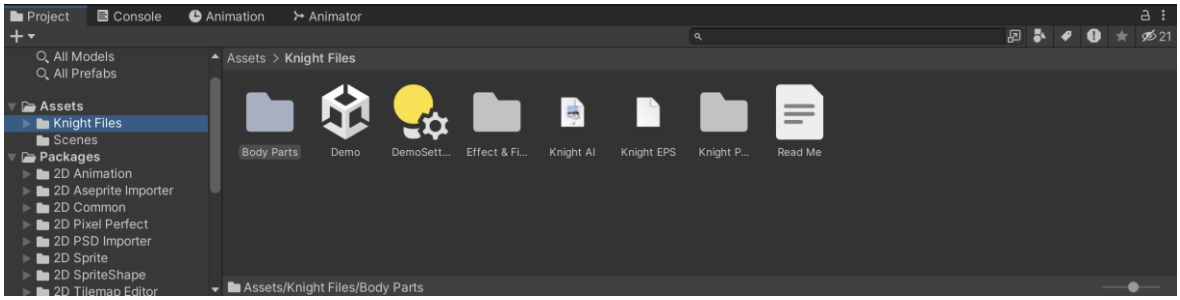
### 7.4.1 Přidání grafických komponentů

Pro začátek si za použití Unity Asset Storu [77] (Obr. 15) do projektu nahrajeme grafické sprity a tile mapy, které budou fungovat jako referenční bod pro budoucí úpravu. Přihlásíme se do Asset Storu, kde si rozklikneme 2D a následně přejdeme do sekce Characters.




Obrázek 15 Unity Asset Store

Jelikož se nejedná o finální grafiku, nebude potřeba si pořizovat placené grafické charaktery. Proto si zobrazíme pouze ty zdarma a vybereme si nějaký jednoduchý charakter, který přichází už i s animacemi. Jakmile najdeme takový, který se nám líbí, rozklikneme si jeho stránku v obchodě a pomocí tlačítka **Add to My Assets** si ho přidáme do Unity. Po úspěšném přidání se nás Asset Store zeptá, zda chceme grafický balíček našeho nového charakteru otevřít v Unity. Když jej otevřeme, zobrazí se nám Packet Manager okno, ve kterém si můžeme balíček nahrát přímo do Unity. Po úspěšném přidání nalezneme náš charakter v okně Project (Obr. 16).

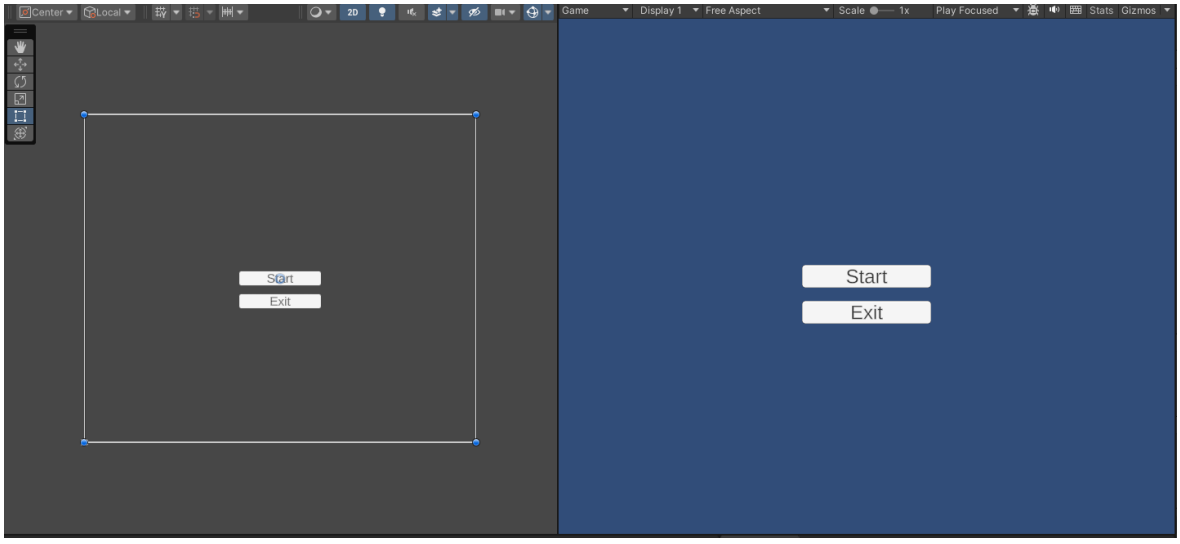



Obrázek 16 Okno Project view

Stejným způsobem si do projektu přidáme více charakterů, které mohou představovat rozdílná NPC a Tilemap sety, pomocí kterých si vytvoříme jednoduché prostředí.

### 7.4.2 Vytvoření hlavního menu

Pro začátek si vytvoříme základní herní menu (Obr. 17.), které hru buď ukončí nebo spustí novou hru. Toho dosáhneme tak, že si vytvoříme novou scénu ve složce Scenes a pojmenujeme ji StartMenu. Následně bude potřeba vytvořit nový Canvas object, který zvládne zobrazovat herní tlačítka. Vytvoříme dvě, jedno pro Start a druhé pro Exit.




Obrázek 17 Scene view a Game view s hlavním menu

Nyní máme na scéně dvě tlačítka, která ale zatím nic nedělají. Proto si vytvoříme nový skript ve složce Scripts a pojmenujeme jej StartMenuManager. Jeho výsledná podoba je ukázána na obrázku 18.

```
public class StartMenuManager : MonoBehaviour
{
    // Start is called before the first frame update
    Zpráva Unity | Počet odkazů: 0
    void Start()
    {

    }

    Počet odkazů: 0
    public void StartFirstScene()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    Počet odkazů: 0
    public void doExitGame()
    {
        Application.Quit();
    }

}
```

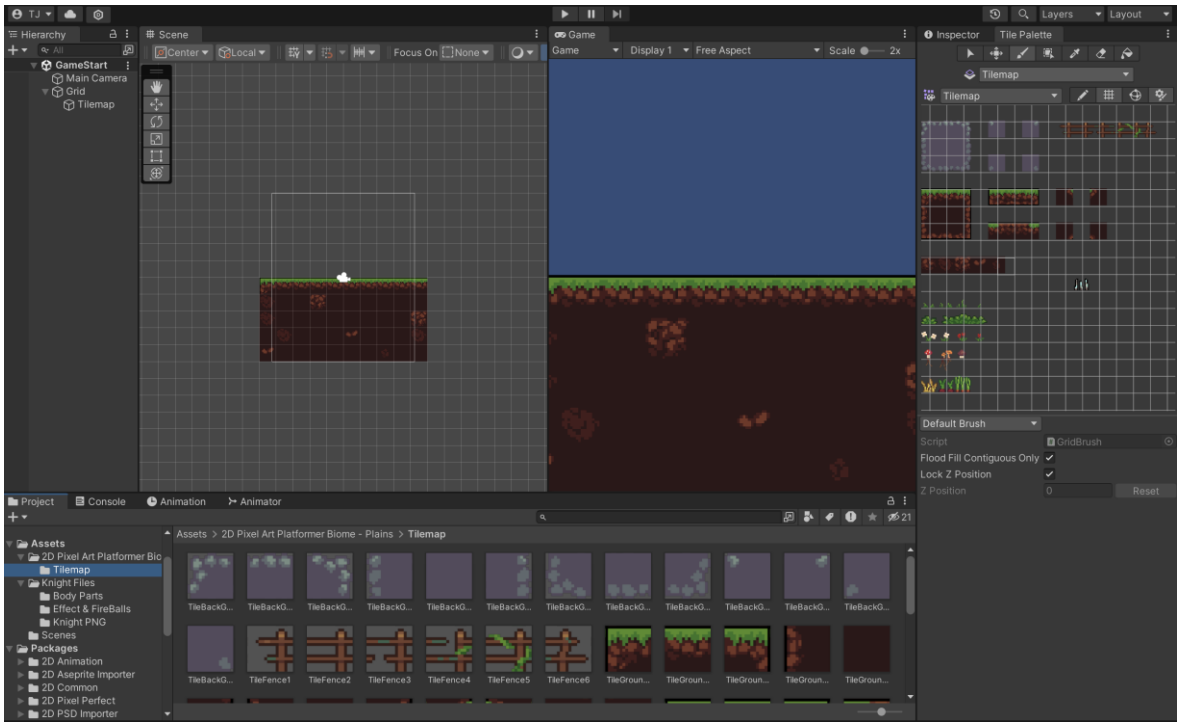
Obrázek 18 Ukázka kódu pro hlavní menu

Aby skript fungoval, vytvoříme si nový Empty Object, který pojmenujeme StateManager. Tomu následně přidělíme náš skript. Nyní se vrátíme k našim tlačítkům, které obsahují funkci OnClick(). Přidělíme jim správné funkce, tím myšleno tlačítku Start přidělíme funkci StartFirstScene() a tlačítku Exit přidělíme funkci doExitGame(). Poté oběma přidáme objekt StateManager. Nyní, když hru spustíme, tak bude fungovat pouze Exit tlačítko. To proto, jelikož nemáme nastavenou scénu, na kterou Start bude odkazovat. Můžeme to otestovat tak, že vytvoříme novou prázdnou scénu, kterou pojmenujeme TestScene. Následně použijeme zkratku Ctrl+Shift+B, což nám otevře okno Build Settings, ve kterém si můžeme nastavit posloupnost našich scén. Jakmile máme nastaveno, můžeme projekt zapnout a vidíme, že obě tlačítka fungují správně.

### 7.4.3 Základní prostředí

Pro základ každé hry je důležité mít nějaký svět, ve kterém se všechno odehrává. Proto si vytvoříme novou scénu, která bude sloužit jako základní stavební kámen začátku hry. Tu vytvoříme v Project okně, kde si najdeme složku Scenes, ve které klikneme pravým tlačítkem, zvolíme Create a následně Scene. Jakmile máme naši novou scénu připravenou, bude potřeba vytvořit Tilemapu. Tu jsme si připravili a importovali do Unity v podkapitole 8.4.1. Nyní bude potřeba si vytvořit Tilemap Game Object, který vytvoříme tak, že v Hierarchy okně klikneme pravým tlačítkem, vybereme 2D Object a následně Tilemap. Nyní bude potřeba si otevřít okno Tile Palette, které najdeme, když si v horní nabídce

rozklikneme možnost Window > 2D > Tile Pallete. To nám umožní vybrat naši importovanou Tilemapu, kde pak stačí vybrat blok grafiky, který budeme nanášet do nově vytvořeného Tilemap objektu. Tímto způsobem si můžeme poskládat naše úplně první prostředí naší hry (Obr. 19).

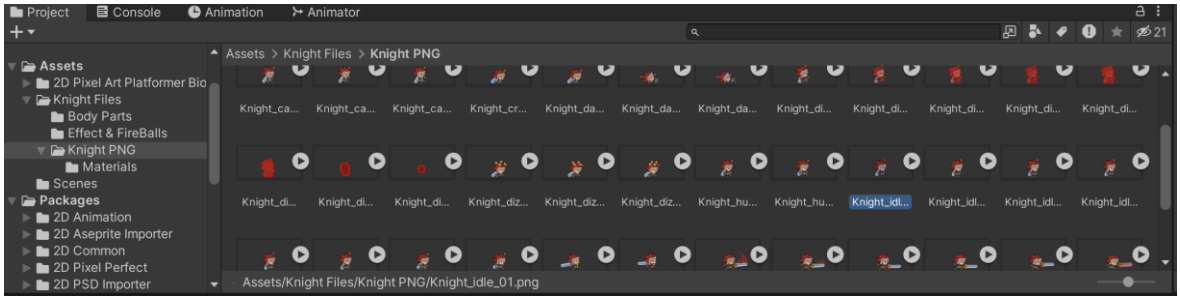



Obrázek 19 Ukázka aplikovaných Tilesetů

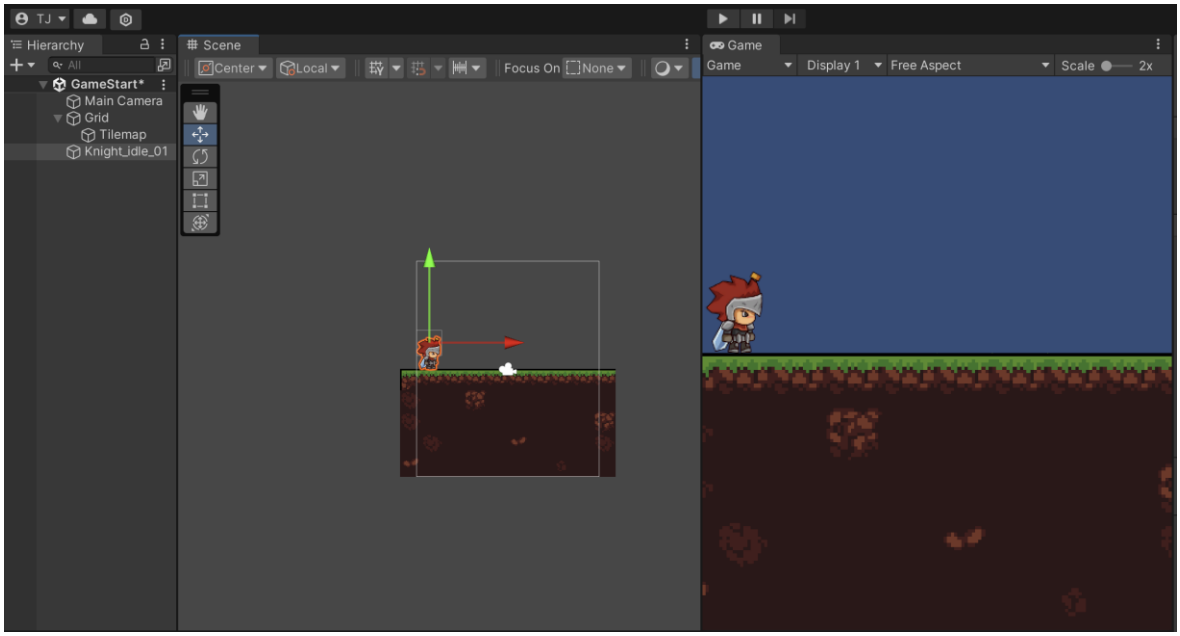
Jak lze na obrázku vidět, všechny změny provedeny v okně Scene jsou automaticky zobrazeny i v Game view okně.

### 7.4.4 Implementace hráčských postav

Vložení charakteru na scénu probíhá tak, že si v okně Project najdeme náš implementovaný charakter, který jsme si přidali do Unity a vybereme si v jaké podobě ho budeme chtít přidat do hry. Následně ho pouze přetáhneme na Scénu. V našem případě vybereme sprite v Idle animaci a přetáhneme jej na scénu (Obr. 20 a 21).




Obrázek 20 Složka se sprity hráčského charakteru

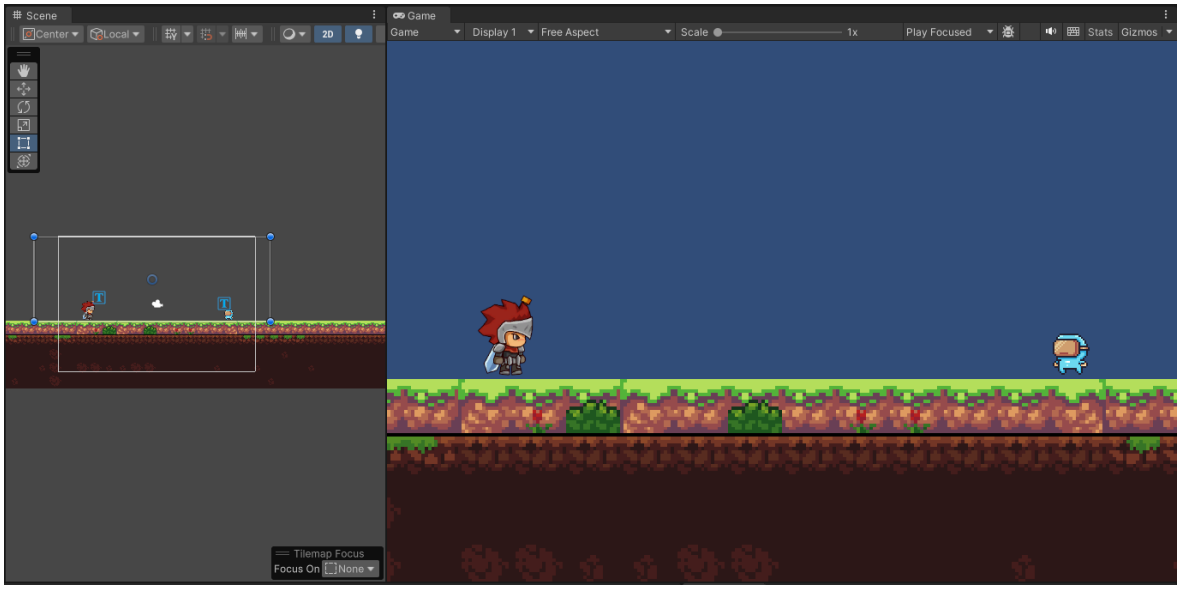



Obrázek 21 Přidání charakteru na scénu

Jak jde vidět na obrázku 21, charakter se automaticky zobrazuje i v Game view. Nicméně pokud bychom nyní spustili simulaci hry, nic se nestane, jelikož ještě nemáme nastavenou žádnou logiku ani animace hry.

### 7.4.5 Implementace nehráčských postav

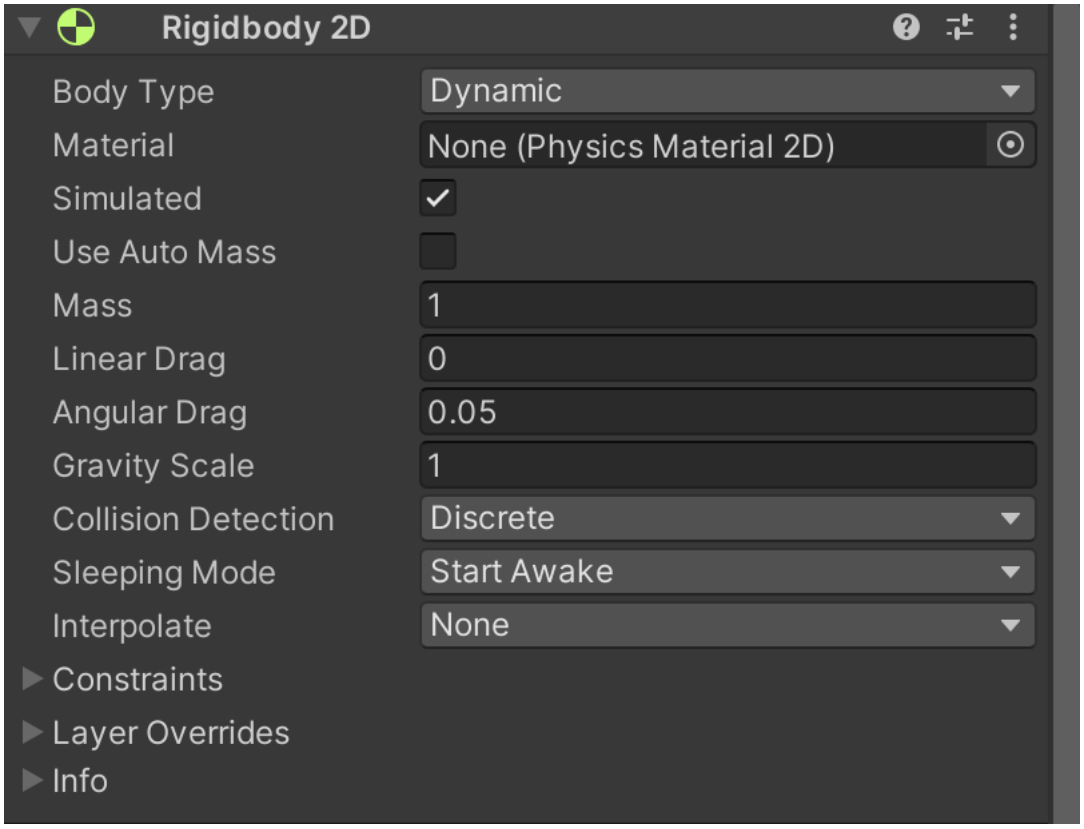
Přidání nehráčských postav (NPC) bude fungovat stejným způsobem, jako přidávání té hráčské. Stačí lokalizovat složku obsahující vybrané charaktery, které budou reprezentovat NPC a následně je přetáhnout na scénu (Obr. 22).




Obrázek 22 Přidání NPC na scénu

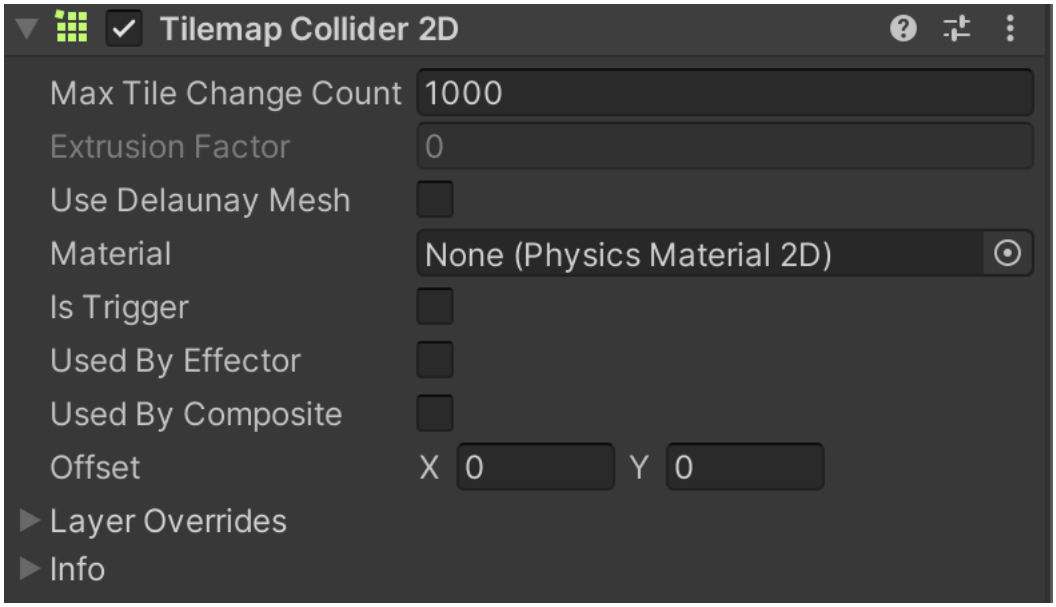
### 7.4.6 Fyzikální prvky

Důležitou součástí každé hry je mít funkční fyziku. Aby naše postava reagovala na gravitaci, je nutno jejímu objektu přidat nový komponent, kterým je RigidBody2D. To uděláme tak, že v Hierarchy okně vybereme objekt postavy a Inspektoru úplně dole klikneme na Add Component. Nyní vyhledáme RigidBody 2D a přidáme jej (Obr. 23). Tím docílíme, že náš charakter po zapnutí simulace začne padat.




Obrázek 23 Komponenta Rigidbody 2D

Abychom zabránili tomu, že charakter pokračuje v pádu a nezastavuje se, musíme mu přidat další komponentu, kterou je Box Collider 2D. Ten slouží k tomu, že kontroluje kolize mezi dvěma objekty. V základu to znamená, že dva objekty s aktivním Box Colliderem nikdy neprojdou přes sebe, pokud nespecifikujeme jinak, co se při kolizi má stát. To nám pomůže k tomu, aby nám postavy nepadaly dolů z obrazovky, ale zůstaly stát na naší vytvořené plošině. Nicméně nestačí pouze přidat Box Collider 2D k našim charakterům. Musíme kontrolovat kolizi i mezi plošinou, a to tak, že našemu Tilemap objektu přidáme komponentu Tilemap Collider 2D (Obr. 24) stejně, jako jsme dělali u charakterů.



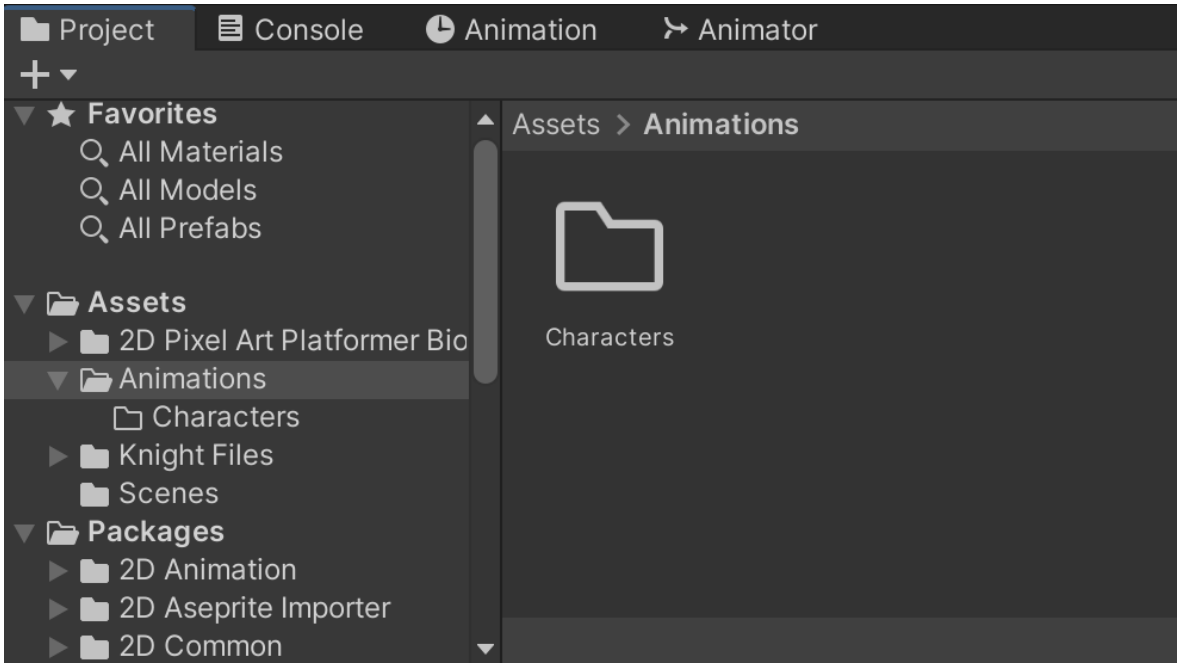

Obrázek 24 Komponenta Tilemap Collider 2D

Nyní nám po zapnutí simulace zůstanou postavy stát na platformě. To můžeme otestovat tak, že je ve Scene view dáme trochu výš nad platformu a po zapnutí simulace v Game view nám spadnou a zastaví se.

### 7.4.7 Animování

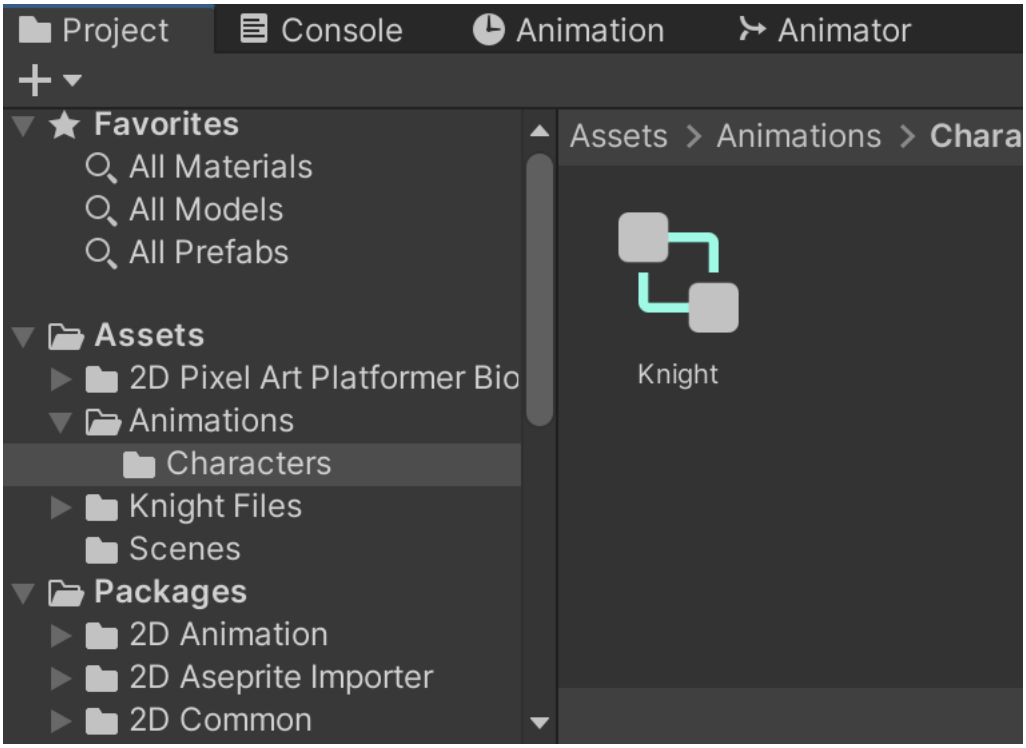
Aby postavy ve hře nebyly jen nepohybující se sprity, je potřeba k nim přidat animace. To hře dodá určitou hloubku a pro hráče ji udělá živější.

Pro začátek animování je potřeba vytvořit novou složku, ve které se bude nacházet Animator Controller spolu s animacemi. (Obr. 25)



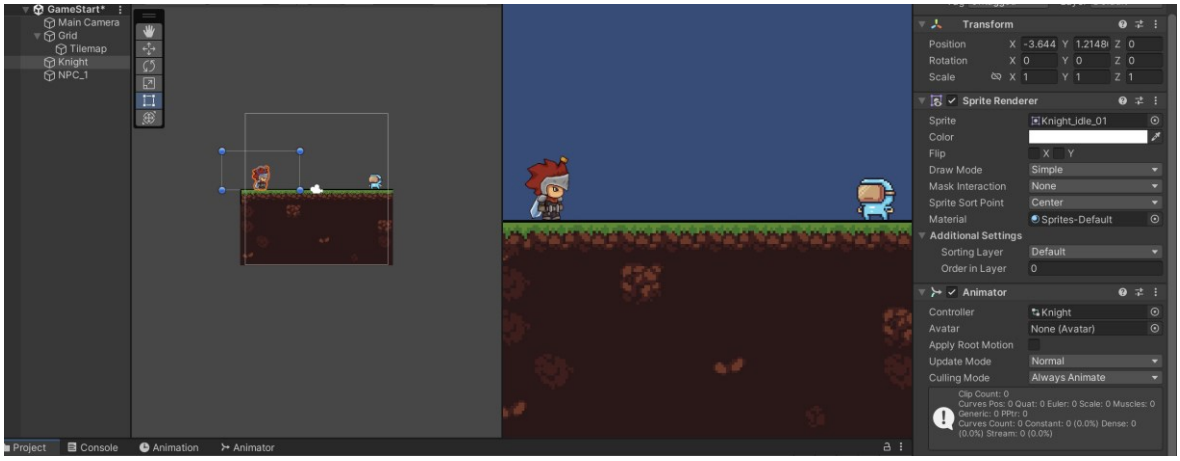

Obrázek 25 Složka s animacemi charakteru

Následně si v nově vytvořené složce vytvoříme Animator Controller, který si nějak pojmenujeme – doporučuji pojmenovávat podle objektů, které animujeme, jako je zobrazeno na obrázku 26.



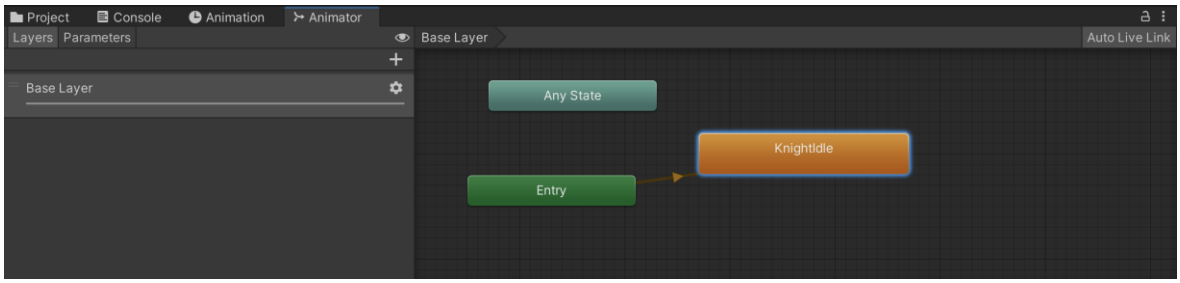

Obrázek 26 Vytvoření Animator Controlleru

Dalším krokem bude přesunout Animator Controller na objekt, který animujeme v Hierarchy okně. Po úspěšném přetažení můžeme vidět, že se nám k vlastnostem objektu v Inspectoru přidal Animator (Obr. 27).




Obrázek 27 Ukázka Animator komponenty u Game Objectu

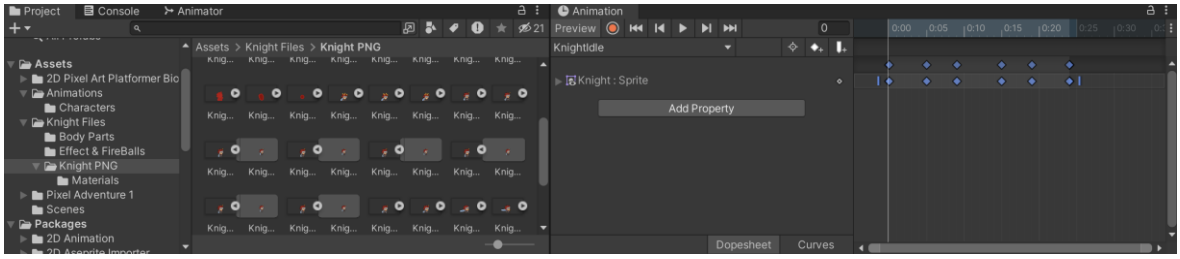
Následně stejným způsobem, jakým jsme vytvořili Animator Controller, vytvoříme Animation. Pro lepší přehlednost bude potřeba animaci pojmenovat podle možných stavů. Těmi mohou být například Idle stav, který se provede, když charakter nic neděl, nebo Move stav, který se přehrává v moment, kdy se charakter pohybuje. Po vytvoření a pojmenování našeho prvního stavu jej opět přetáhneme na objekt, který animujeme. To zaručí, že se náš stav přidá do Animator Controlleru, jak ukazuje obrázek 28.




Obrázek 28 Základní Animator window

Tímto způsobem vytvoříme animace pro každý stav, který náš charakter má a obdobně je přidáme.

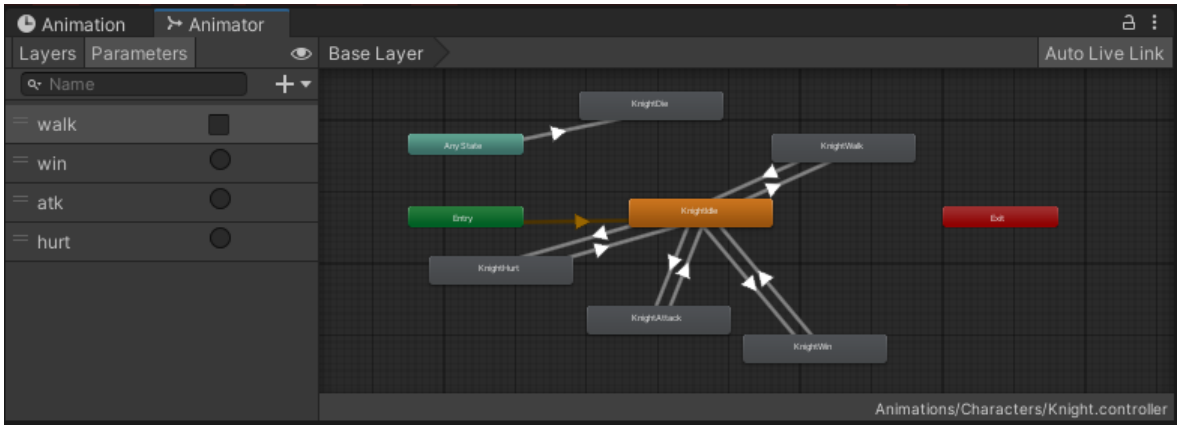
Poté přichází samotné animování, kdy si ve složce s naším charakterem najdeme všechny snímky pro daný stav animace, vybereme je a přetáhneme je do okna Animation. Důležité je u toho mít vybraný správný objekt v Hierarchy okně. Úspěšné přetažení poznáme tak, že se nám na časové ose vytvoří modré kosočtverce každý značící jeden snímek. Čas animace nastavíme na 24 sekund a tím máme hotovou první animaci. Tvorba animace je ukázána na obrázku 29.




Obrázek 29 Ukázka tvorby animace

V dalším kroku je potřeba nastavit zbytek animací stejným způsobem, jakým jsme dělali tu první.

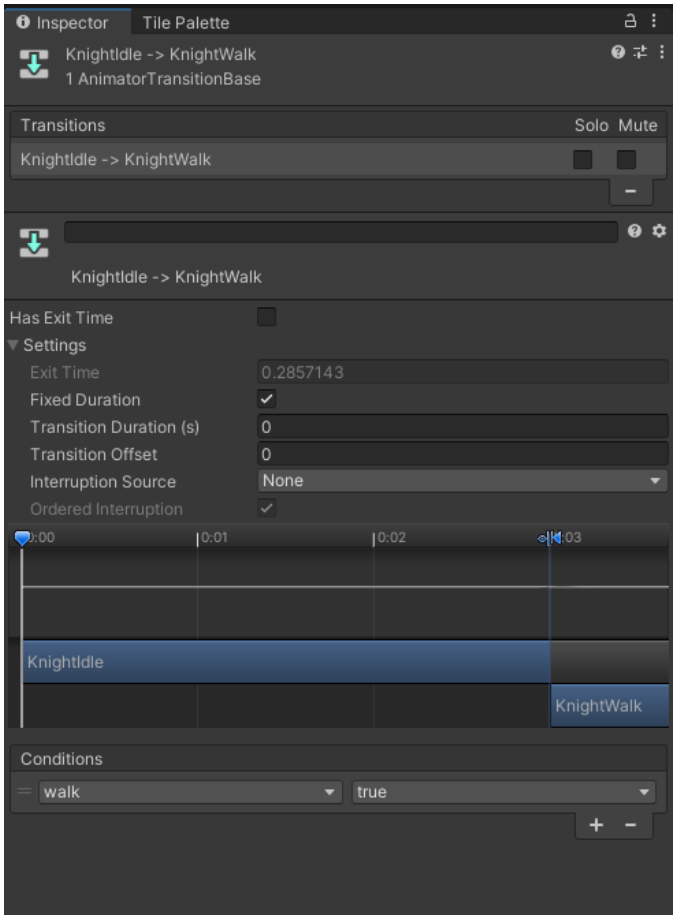
Jakmile máme všechny animace hotové, je potřeba mezi nimi udělat přechody. To zaručí plynulý přechod z jedné animace do druhé a zpět. Důležitým krokem je určit, která animace bude defaultní. V našem případě to bude KnightIdle, v jiném případě je nutno kliknout na danou animaci pravým tlačítkem a nastavit ji jako defaultní. Defaultní animace se pozná tak, že je v oranžovém boxu. Nastavení přechodu uděláme tak, že klikneme na defaultní animaci a vybereme Make Transition. Následně animaci propojíme se všemi ostatními, které jsme přidali. V případě, že chceme, abychom se z naší nové animace dostali zpět do defaultní, budu nutné nastavit přechod i zpět. Finální vzhled propojení animací je zobrazen na obrázku 30.




Obrázek 30 Ukázka finální animace

Na obrázku 30 lze vidět, že jsou zde nějaké parametry u animací. Ty fungují jako podmínky pro spuštění dané animace. Novou podmínku můžeme přidat pomocí symbolu „+“ u parametrů. V našem případě používáme bool podmínku u animace pro chůzi (walk) a pro animace výhry (win), útoku (atk) a zranění (hurt) používáme trigger.

Následně můžeme nastavit rychlost, jakou se přechod provede. Toto nastavení, které lze vidět na obrázku 31, získáme po kliknutí na konkrétní šipku u přechodu.




Obrázek 31 Nastavení přechodu u animací

V našem případě jsou všechny přechody nastaveny na hodnotu 0, aby byl přechod okamžitý. Je nutno podotknout, že pro každý přechod je toto nastavení nutno provést separátně.

Animace lze kontrolovat i skrze skripty a to tak, že si deklarujeme proměnnou Animatoru, kterou inicializujeme ve Start funkci na začátku skriptu a následně ji můžeme volat.

Příklad deklarace:

Private Animator anim;

Inicializace:

anim = GetComponent<Animator>();

Volání animace:

anim.SetBool(„walk“, false);

nebo

anim.SetTrigger(„atk“);

### 7.4.8 Rolovací efekt na pozadí

Abychom hře přidali nějakou hloubku, je potřeba přidat i pozadí, které reaguje na pohyb postavy. Když je postava ve stavu pohybu, pozadí se začne posouvat. Toho dosáhneme tak,

že vytvoříme nový objekt, konkrétně 3D Plane a nastavíme ho do pozadí. Nyní na něj bude potřeba nanést pozadí v podobě materiálu. Ve složce Materials si vytvoříme nový materiál, kterému přidáme vzhled našeho pozadí, a to následně naneseme na náš Plane, který si přejmenujeme na Background. Poté si vytvoříme nový skript, který bude řešit logiku posouvání. Skript je ukázán na obrázku 32.

```
public class ScrollingBackground : MonoBehaviour
{
    public GameObject stateObject;
    private StateHolder stateHolder;
    [Range(-1f, 1f)]
    public float scrollSpeed = 0.5f;
    private float offSet;
    private Material mat;
    // Start is called before the first frame update
    Zpráva Unity | Počet odkazů: 0
    void Start()
    {
        mat = GetComponent<Renderer>().material;
        stateHolder = stateObject.GetComponent<StateHolder>();
    }

    // Update is called once per frame
    Zpráva Unity | Počet odkazů: 0
    void Update()
    {
        if(stateHolder.isPlayerWalking == true)
        {
            offSet += (Time.deltaTime * scrollSpeed) / 10f;
            mat.SetTextureOffset("_MainTex", new Vector2(offSet, 0));
        }
    }
}
```

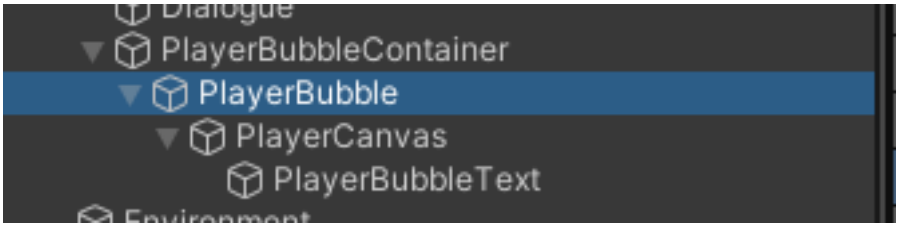
Obrázek 32 Ukázka skriptu pro řešení efektu skrolování

Nyní skript přidáme k našemu objektu pozadí a kdykoliv je hráčská postava v animaci pohybu, pozadí se zároveň s ní začne posouvat, což přidává na celkové hloubce prostředí.

### 7.4.9 Textové bubliny

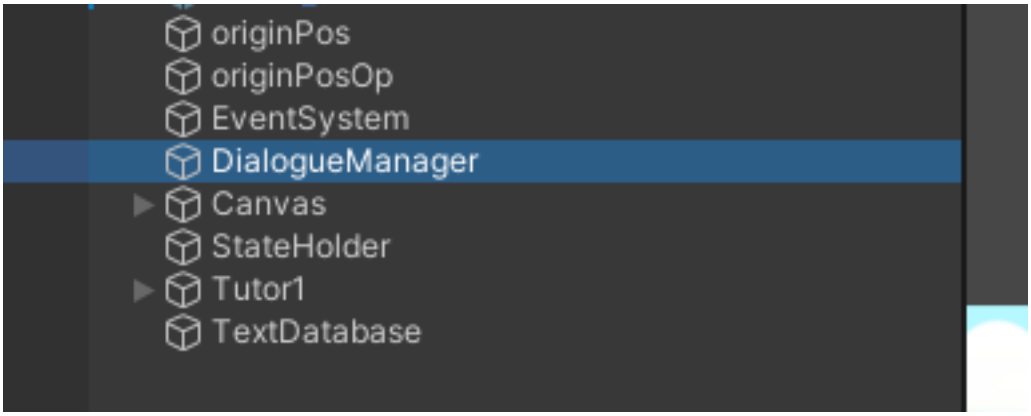
Hlavní způsob předávání informací a komunikace ve hře je text. Ten je zobrazován v textových bublinách nad hráčskou či NPC postavou a její výslednou podobu lze vidět na obrázku 37. Textová bublina je nicméně sama o sobě jen sprite, který přetáhneme nad charakter hráče a NPC zvlášť. Nesmíme zapomenout přidat animace na zobrazení textové bubliny a její ukončení.

Aby bublina mohla zobrazovat text, můžeme využít funkce TextMeshPro, která nám k tomu pomůže. Na pozici textové bubliny vytvoříme Canvas, na kterém následně můžeme pracovat s textem. Objektová posloupnost textové bubliny v Hierarchy view je ukázána na obrázku 33.

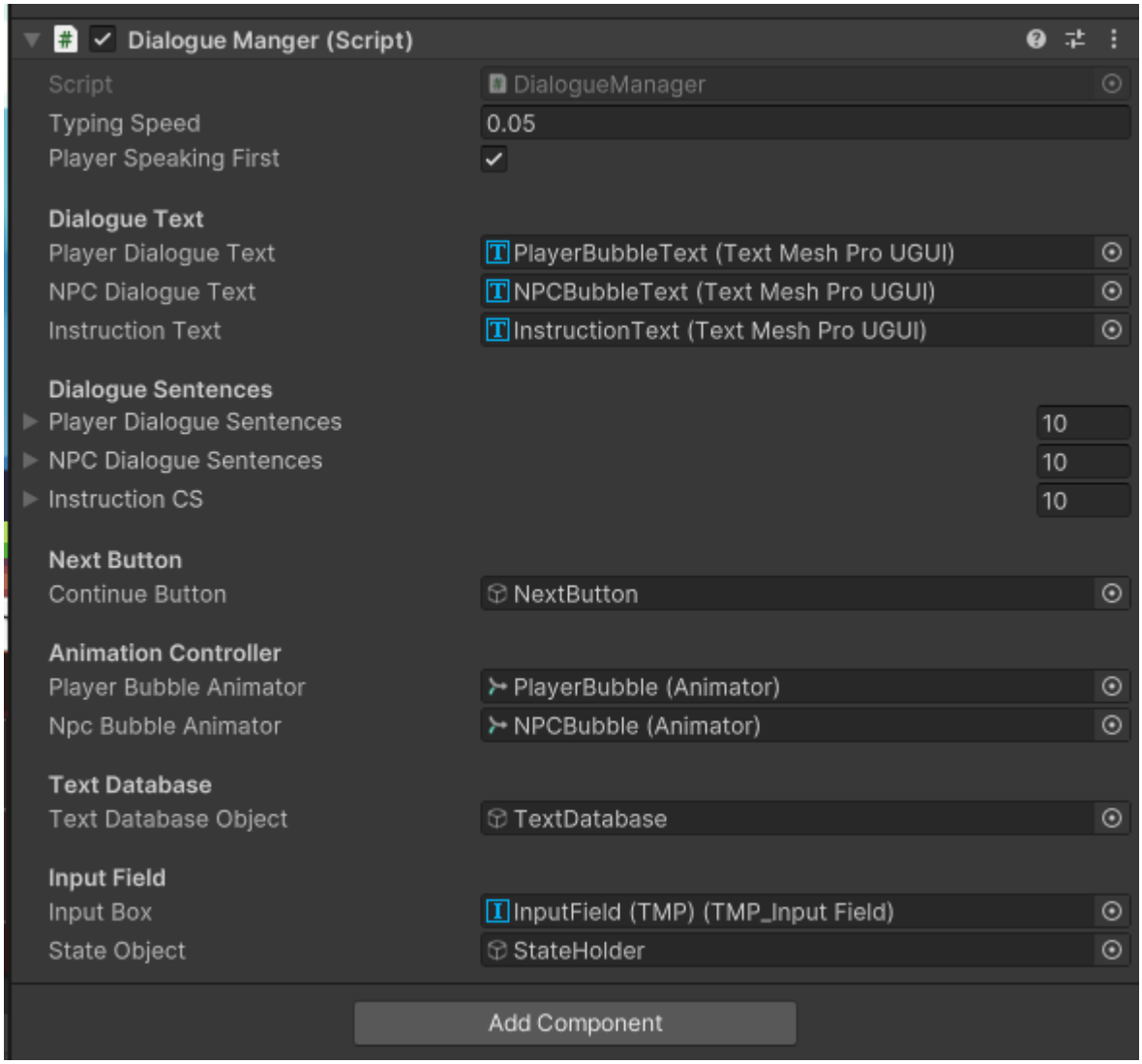



Obrázek 33 Textová bublina v Hierarchy view

Následné řešení všeho textu, dialogů a práce s textovými bublinami je řešeno ve skriptu Dialogue Manager (Obr. 35), který je připojen na objekt DialogueManager, který lze vidět na obrázku 34.




Obrázek 34 Game Object DialogueManager



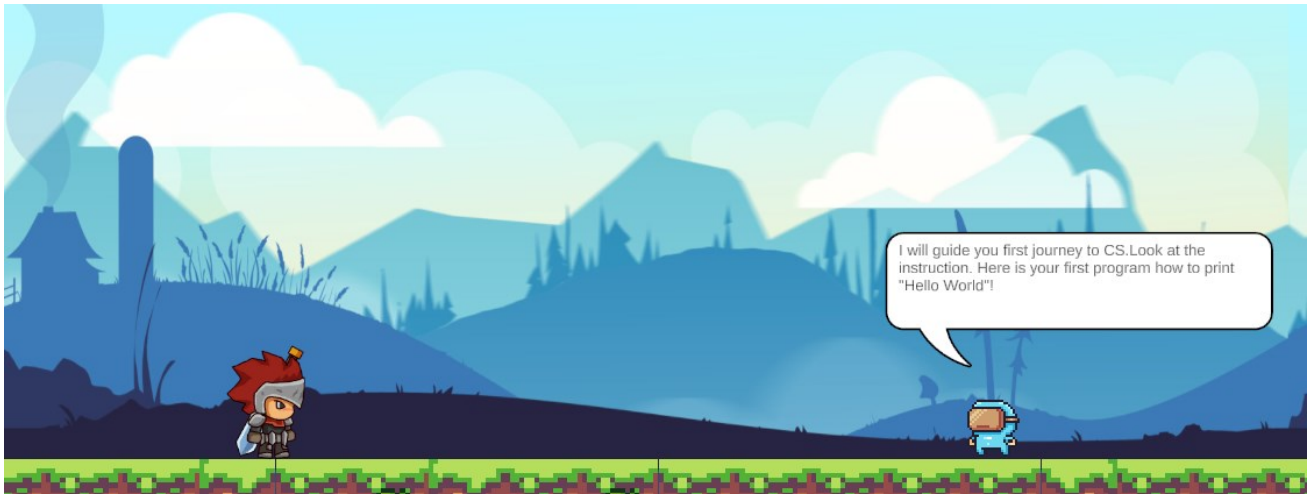

Obrázek 35 Dialogue Manager skript

Každá jedna z konverzací je zavolána metodou Coroutine – StartCoroutine(StartDialogue());. Pomocí této metody můžeme používat animace bublin a vypisování textu.

Pro samotné vypisování textu používáme následující kód na obrázku 36.

```
foreach (char letter in NPCDialogueSentences[NPCIndex].ToCharArray())
{
    NPCDialogueText.text += letter;
    yield return new WaitForSeconds(typingSpeed);

}
```

Obrázek 36 Kód pro vypisování textu




Obrázek 37 Ukázka textové bubliny ve hře

### 7.4.10 Hráčský input

Důležitou součástí hry bude také hráčský input. Ten slouží k tomu, aby hráč mohl plnit různé úkoly či souboje pomocí kódu, který vloží okna pro vstup. Abychom toho dosáhli, je potřeba si vytvořit nový TextMeshPro objekt uvnitř Canvasu pomocí pravého tlačítka myši, následně vybereme UI a Input Field. V Inspectoru mu následně přiřadíme On End Edit > DialogueManager objekt. Hráčův input je pak komplexně řešen ve skriptu Dialogue Manager v metodě InputHandling() na obrázku 38.

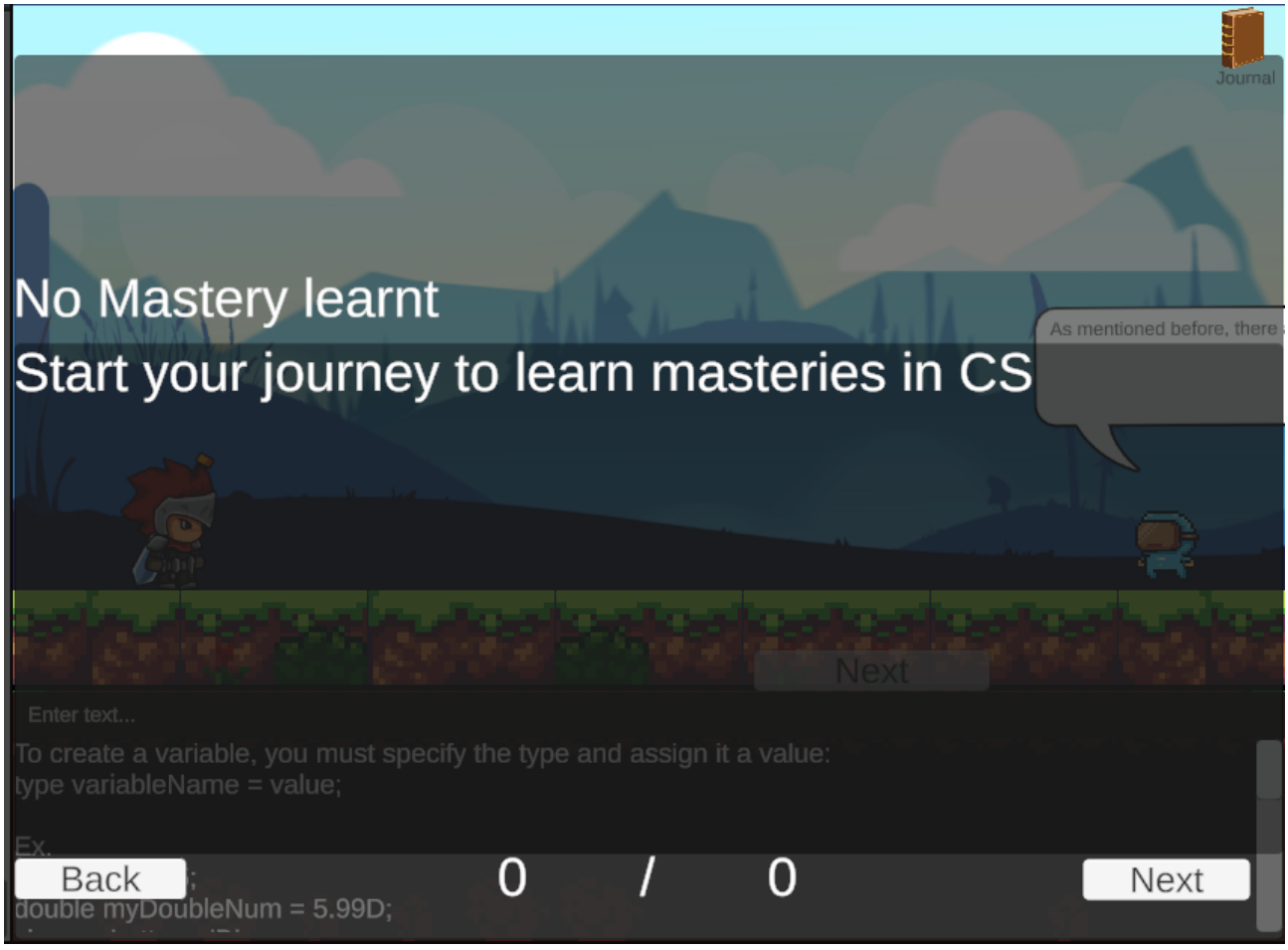
```
public void InputHandling()
{
    //var url = Path.Combine(Application.dataPath + "/TextResources", "types.txt");
    //string[] textList = File.ReadAllLines(url);

    inputText = inputBox.text;
    if(inputText == "Exit()")
    {
        NPCDialogueText.text = string.Empty;
        NPCDialogueText.text = "Ok, let's continue!";
        endInputStage = true;
        return;
    }
    switch (stateHolder.stage)
    {
        case 2:
            processText_2(inputText);
            break;
        case 3:
            processText_3(inputText);
            break;
        case 5:
            {
                var url = Path.Combine(Application.dataPath + "/TextResources", "types.txt");
                string[] textList = File.ReadAllLines(url);
                processText_5(inputText, textList);
                break;
            }
        case 6:
            {
                var url = Path.Combine(Application.dataPath + "/TextResources", "types.txt");
                string[] textList = File.ReadAllLines(url);
                processText_6(inputText, textList);
                break;
            }
        case 7:
            {
                var url = Path.Combine(Application.dataPath + "/TextResources", "operators.txt");
                string[] operatorsQuiz = File.ReadAllLines(url);
                processText_7(inputText, operatorsQuiz);
            }
            break;
        default:
            break;
    }

}
```

Obrázek 38 InputHandling() metoda

### 7.4.11 Hráčský Journal

Aby měl hráč možnost se vždy podívat zpětně na to, co ho hra naučila, je do hry implementován Journal systém, jehož ukázka je na obrázku 40. Ten funguje jako deník, do kterého se ukládají všechny nové informace. Pro jeho vytvoření nám opět poslouží Canvas, na kterém si vytvoříme nové tlačítko, které bude reprezentovat hlavní Journal. Následně přidáme další tlačítka pro navigování mezi stránkami a text, který se bude zobrazovat. Výsledné objekty lze vidět na obrázku 39.


Obrázek 39 Journal v Hierarchy view




Obrázek 40 Ukázka Journalu ve hře

Nyní se hráči každá nová informace uloží dovnitř Journalu, kde si mezi nimi může listovat a procházet si je.

### 7.4.12 Implementace audia

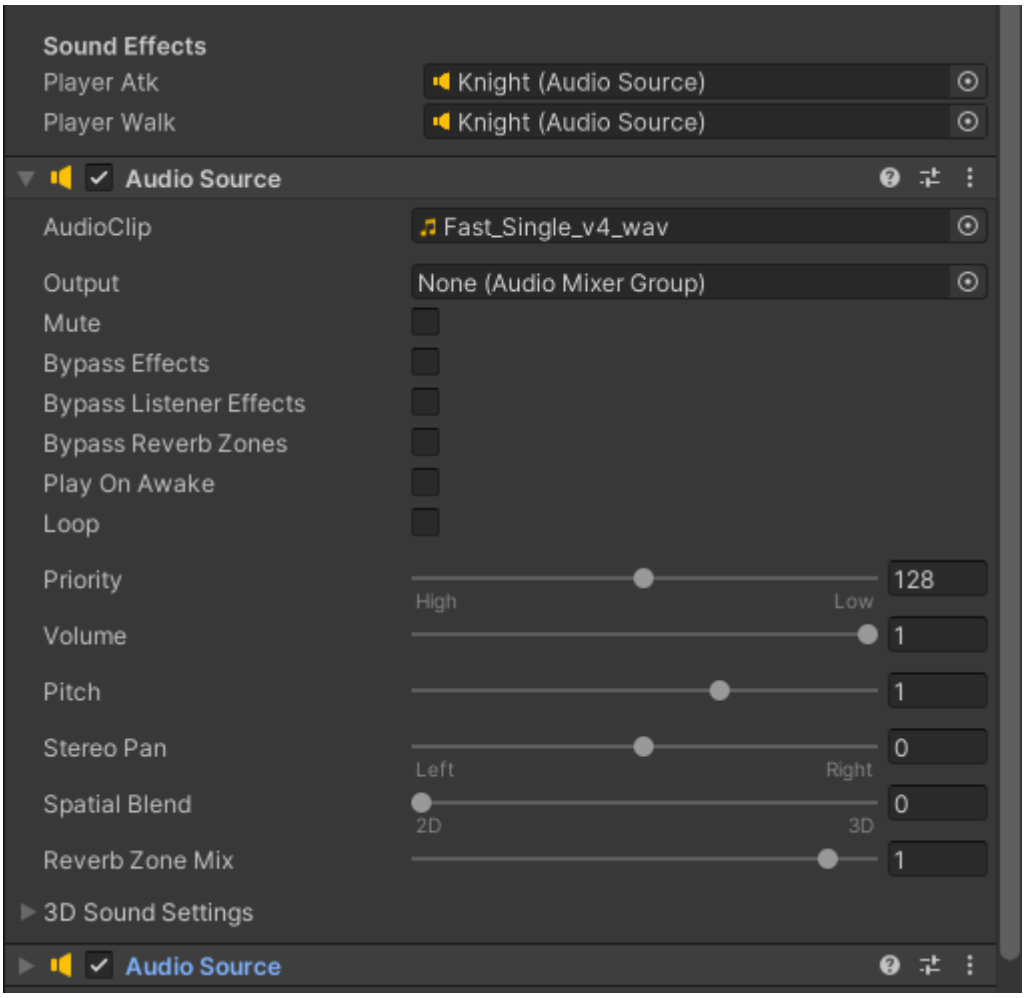
Pro implementaci audia, které bude hrát při různých událostech, jako je například chůze nebo zobrazení dialogů, je potřeba udělat pár kroků. Pro začátek je důležité si nějaké audioefekty

přidat do projektu. Následně je potřeba pro konkrétní objekty ve skriptech vytvořit SerializeField proměnné komponenty AudioSource, což je ukázáno na obrázku 41.

```
[Header("Sound Effects")]
[SerializeField] private AudioSource playerAtk;
[SerializeField] private AudioSource playerWalk;
```

Obrázek 41 Vytvoření audio proměnných

Poté je potřeba k objektu přidat samostatnou Audio Source komponentu (Obr. 42).




Obrázek 42 Audio Source komponenta pro Game Object

Je potřeba přidat tolik komponent, zvukových efektů bude daný objekt vydávat. V našem případě bude potřeba přidat pouze dva – jeden pro útok a druhý pro chůzi. Přidaným komponentám pak přiřadíme konkrétní audio efekt a následně je přetáhneme pod správné proměnné v okně Sound Effects.

Aktivaci zvuku následně řešíme ve skriptech, například jako na obrázku 43.

```
anim.SetBool("walk", true);
if (!playerWalk.isPlaying)
{
    playerWalk.Play();
}
```

Obrázek 43 Kód pro přehrání audia

### 7.4.13 Rozdělení levelů

Učivo je děleno na levely, kdy po každém levelu přichází test v podobě souboj s bossem, ve kterém jsou otestovány dosavadní zkušenosti hráče.

### 7.4.14 Úprava grafického rozhraní

Aby hra působila pro hráče přívětivěji, je vhodné grafickou stránku hry aktualizovat do modernějšího vzhledu. K tomu je nicméně za potřebí mít potřebné grafické materiály. Nejjednodušším způsobem, jak grafické komponenty získat, je Unity Asset Store, který jich má na výběr spoustu, ať už placených, tak těch zdarma. Pro přístup ke stažení komponentů je zapotřebí být přihlášen k Unity účtu. Do hry je následně můžeme aplikovat stejným způsobem, jako jsme to dělali s dočasnou grafikou.

# 8 TESTOVÁNÍ HRY

Po dokončení vývoje bylo nutno, aby aplikace hry podstoupila testování na různých operačních systémech. Byly zvoleny operační systémy Windows 10 a MacOS. Hra byla spuštěna na konkrétních zařízeních používající daný OS.

### 8.1.1 Testování na platformě Windows 10

Aplikace byla spuštěna na platformě Windows 10, kde byla otestována její funkčnost. Po spuštění aplikace došlo k načtení hlavního menu, ve kterém všechna tlačítka fungovala dle očekávání. Následný průchod hrou byl plynulý a nevykazoval žádné zásadní problémy, které by znemožnily hraní.

### 8.1.2 Testování na platformě MacOS

Při testování na MacOS došlo k obdobným závěrům, jako na Windows 10. Aplikace fungovala dle očekávání, bez závažných chyb znemožňující průchod hrou.

# 9 BUDOUCNOST HRY

Výsledná naprogramovaná hra nyní může sloužit nejen jako jedna z efektivních metod pro osvojení programovacího jazyka C#, ale také jako výchozí bod pro další rozvoj a zdokonalování programátorských dovedností. Existuje mnoho možností, jak hru dále rozšířit a vylepšit. Například by mohly být přidány nové úrovně nebo moduly, které by se zaměřily na pokročilé techniky a koncepty v C#, jako jsou LINQ dotazy, asynchronní programování nebo práce s databázemi.

Kromě toho, hra by mohla být rozšířena o podporu dalších programovacích jazyků. Tím by se stala univerzálnějším nástrojem pro učení, který by mohl oslovit širší spektrum uživatelů s rozličnými zájmy a úrovněmi dovedností. Například přidání jazyka Python by mohlo přilákat začínající programátory díky jeho snadné čitelnosti a širokému využití, zatímco implementace jazyka JavaScript by mohla přilákat ty, kteří se chtějí více zaměřit na vývoj webových aplikací.

Dále by bylo možné integrovat více interaktivních tutoriály a testů, které by hráčům umožnily nejen aplikovat nově naučené koncepty přímo ve hře, ale také ihned získat zpětnou vazbu a návrhy na zlepšení.

# ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a vytvořit hru pro výuku programovacího jazyka C# za předpokladu využití enginu Unity. Následně bylo za cíl vytvořenou hru řádně otestovat.

V teoretické části byl čtenář seznámen s důvody, jak a proč implementovat hry do školního prostředí. Byly zde rozebrány metody, jakými je třeba se řídit pro správné zavedení her do výuky a zároveň bylo provedeno porovnání mezi výhodami a nevýhodami výukových her. Dále byl čtenáři přiblížen programovací jazyk C#, popsány jeho funkce a využití v reálném kódování. Následně byl představen herní engine Unity, který je rozsáhle používán mezi herními vývojáři. Bylo ukázáno a popsáno, jak C# v Unity funguje a jaké funkce Unity pro vývoj 2D her nabízí. V poslední části teorie tak byl přiblížen proces vývoje naučných her, důležité kroky, kterými je třeba se řídit a zároveň byly přiblíženy již existující naučné hry na trhu, kde každá byla představena a popsána.

Praktická část pojednávala o vývoji samotné naučné hry, kde v první části byl vytvořen návrh struktury programované hry a následně probíhalo samotné vyvíjení a testování.

V části vývoje hry je postupně popsána struktura programování – konkrétně samotné vytvoření základního projektu, importování grafických assetů z Unity Asset Storu, vytvoření základního prostředí, implementace herních charakterů, fyzika pro 2D prostředí, animace postav a herního pozadí, dialogové bubliny, hráčský vstup a Journal, implementace audia, rozdělení levelů a úpravy grafického rozhraní.

Na konec byla hra úspěšně otestována na vícero operačních systémech a byla popsána eventuální budoucnost hry.

Na závěr bych zdůraznil, že vývoj mé vlastní hry byl pro mě bohatou zkušeností. Ačkoli výsledná hra nemá rozsah ani kvalitu velkých herních studií, jsem vděčný za možnost něco takového vytvořit a potvrdit si své schopnosti. Přiznávám, že hra by mohla být obsahově bohatější a lépe doladěná, ale konečný produkt je funkční a použitelný. Věřím, že každý jeden projekt posune člověka dále a ať už to budou nové implementace do této nebo jiné mé hry, výsledky se budou jen zlepšovat.

Při tvorbě této práce byl použit nástroj generativního modelu AI Chat GPT - https://chat.openai.com/ za účelem asistence při návrhu struktury práce. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

# SEZNAM POUŽITÉ LITERATURY

[1] TeachThought Staff. A Brief History Of Video Games In Education. Online. TeachThought. 2012, s. 1. Dostupné z: https://www.teachthought.com/technology/a-brief-history-of-video-games-in-education/. [cit. 2024-05-02].

[2] VLACHOPOULOS, Dimitrios a MAKRI, Agoritsa. The effect of games and simulations on higher education: a systematic literature review. Online. International Journal of Educational Technology in Higher Education. 2017, roč. 14, č. 1, s. 1. ISSN 2365-9440. Dostupné z: https://doi.org/10.1186/s41239-017-0062-1. [cit. 2024-05-02].

[3] NISBET, Jordan. Game-Based Learning: Pros, Cons & Implementation Tips for Educators: a systematic literature review. Online. Prodigy. 2023, s. 1. Dostupné z: https://www.prodigygame.com/main-en/blog/game-based-learning/. [cit. 2024-05-02].

[4] VANDERCRUYSSE, Sylke; VANDEWAETERE, Mieke a CLAREBOUT, Geraldine. Game-Based Learning: a systematic literature review. Online. Handbook of Research on Serious Games as Educational, Business and Research Tools. 2012, s. 628-647. ISBN 9781466601499. Dostupné z: https://doi.org/10.4018/978-1-4666-0149-9.ch032. [cit. 2024-05-02].

[5] VIRIUS, Miroslav. Programování v C#: od základů k profesionálnímu použití. Grada, 2020. ISBN 978-80-271-1216-6. Dostupné také z: https://www.bookport.cz/AccountSaml/SignIn/?idp=https://shibboleth.utb.cz/idp/shibboleth&returnUrl=/kniha/programovani-v-c-7262/

[6] Microsoft Corporation. The history of C#. Online. Microsoft Learn. 2024, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history#c-version-10-1. [cit. 2024-05-02].

[7] Microsoft Corporation. General Structure of a C# Program. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/. [cit. 2024-05-02].

[8] Microsoft Corporation. Integral numeric types (C# reference). Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types. [cit. 2024-05-02].

[9] Microsoft Corporation. Floating-point numeric types (C# reference). Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types. [cit. 2024-05-02].

[10] Microsoft Corporation. Bool (C# reference). Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/bool. [cit. 2024-05-02].

[11] Microsoft Corporation. Char (C# reference). Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/char. [cit. 2024-05-02].

[12] Microsoft Corporation. Arrays. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/arrays. [cit. 2024-05-02].

[13] Microsoft Corporation. Built-in reference types (C# reference). Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types#the-string-type. [cit. 2024-05-02].

[14] Microsoft Corporation. Declaration statements. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/declarations. [cit. 2024-05-02].

[15] Microsoft Corporation. Value types (C# reference). Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types. [cit. 2024-05-02].

[16] Microsoft Corporation. Built-in reference types (C# reference). Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types. [cit. 2024-05-02].

[17] Microsoft Corporation. Arithmetic operators (C# reference). Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators. [cit. 2024-05-02].

[18] Microsoft Corporation. Equality operators - test if two objects are equal or not. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators. [cit. 2024-05-02].

[19] Microsoft Corporation. Comparison operators (C# reference). Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators. [cit. 2024-05-02].

[20] Microsoft Corporation. Boolean logical operators - AND, OR, NOT, XOR. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators. [cit. 2024-05-02].

[21] Microsoft Corporation. ! (null-forgiving) operator (C# reference). Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-forgiving. [cit. 2024-05-02].

[22] Microsoft Corporation. Addition operators - + and +=. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/addition-operator. [cit. 2024-05-02].

[23] Microsoft Corporation. Selection statements - if, if-else, and switch. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/selection-statements. [cit. 2024-05-02].

[24] Microsoft Corporation. Iteration statements - for, foreach, do, and while. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/iteration-statements. [cit. 2024-05-02].

[25] Microsoft Corporation. Object-Oriented programming (C#). Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop. [cit. 2024-05-02].

[26] Microsoft Corporation. Introduction to classes. Online. Microsoft Learn. 2023, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes. [cit. 2024-05-02]

[27] VIRIUS, Miroslav. Programování v C#: od základů k profesionálnímu použití. Grada, 2020. ISBN 978-80-271-1216-6. Dostupné také z: https://www.bookport.cz/AccountSaml/SignIn/?idp=https://shibboleth.utb.cz/idp/shibboleth&returnUrl=/kniha/programovani-v-c-7262/

[28] VIRIUS, Miroslav. Programování v C#: od základů k profesionálnímu použití. Grada, 2020. ISBN 978-80-271-1216-6. Dostupné také z:

https://www.bookport.cz/AccountSaml/SignIn/?idp=https://shibboleth.utb.cz/idp/shibboleth&returnUrl=/kniha/programovani-v-c-7262/

[29] Microsoft Corporation. Abstract and Sealed Classes and Class Members (C# Programming Guide). Online. Microsoft Learn. 2021, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members. [cit. 2024-05-02].

[30] Microsoft Corporation. Access Modifiers (C# Programming Guide). Online. Microsoft Learn. 2024, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers. [cit. 2024-05-02].

[31] Microsoft Corporation. Generic classes and methods. Online. Microsoft Learn. 2024, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics. [cit. 2024-05-02].

[32] Microsoft Corporation. Introduction to delegates and events in C#. Online. Microsoft Learn. 2022, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/delegates-overview. [cit. 2024-05-02].

[33] Microsoft Corporation. Introduction to events. Online. Microsoft Learn. 2021, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/events-overview. [cit. 2024-05-02].

[34] Microsoft Corporation. Use exceptions. Online. Microsoft Learn. 2021, s. 1. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/using-exceptions. [cit. 2024-05-02].

[35] TYKOSKI, Scott. Mastering Game Design with Unity 2021. Delhi: BPB Publications, 2023. ISBN 9789355512178. Dostupné také z: https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=3469288&authtype=ip,shib&custid=s3936755

[36] Unity Technologies. Coding in C# in Unity for beginners. Online. S. 1. Dostupné z: https://unity.com/how-to/learning-c-sharp-unity-beginners. [cit. 2024-05-02].

[37] Unity Technologies. Welcome to the Unity Scripting Reference!. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/ScriptReference/. [cit. 2024-05-02].

[38] Unity Technologies. Creating and Using Scripts. Online. Unity Documentation. 2018, s. 1. Dostupné z: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. [cit. 2024-05-02].

[39] REED, Adam. Communication Between Scripts In Unity Using GetComponent. Online. Medium. 2021, s. 1. Dostupné z: https://adamwreed93.medium.com/communication-between-scripts-in-unity-using-getcomponent-da51d2634bba. [cit. 2024-05-03].

[40] Unity Technologies. Unity's interface. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/UsingTheEditor.html. [cit. 2024-05-02].

[41] Unity Technologies. Work with sprites. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/Sprites.html. [cit. 2024-05-02].

[42] Unity Technologies. Introduction to Tilemaps. Online. Unity Learn. 2023, s. 1. Dostupné z: https://learn.unity.com/tutorial/introduction-to-tilemaps. [cit. 2024-05-02].

[43] Unity Technologies. Tilemaps. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://learn.unity.com/tutorial/introduction-to-tilemaps. [cit. 2024-05-02].

[44] Unity Technologies. Introduction to components. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/Components.html. [cit. 2024-05-02].

[45] Unity Technologies. Animator Controller. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-AnimatorController.html. [cit. 2024-05-02].

[46] Unity Technologies. Transforms. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-Transform.html. [cit. 2024-05-02].

[47] Unity Technologies. Sprite Renderer. Online. Unity Documentation. 2018, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-SpriteRenderer.html. [cit. 2024-05-02].

[48] Unity Technologies. Line Renderer component. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-LineRenderer.html. [cit. 2024-05-02]

[49] Unity Technologies. Trail Renderer component. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-TrailRenderer.html. [cit. 2024-05-02].

[50] Unity Technologies. Renderer module. Online. Unity Documentation. 2018, s. 1. Dostupné z: https://docs.unity3d.com/Manual/PartSysRendererModule.html. [cit. 2024-05-02].

[51] Unity Technologies. Canvas Renderer. Online. Unity Documentation. S. 1. Dostupné z: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-CanvasRenderer.html. [cit. 2024-05-02].

[52] Unity Technologies. Introduction to character control. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/CharacterControllers.html. [cit. 2024-05-02].

[53] Unity Technologies. Rigidbody component reference. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-Rigidbody.html. [cit. 2024-05-02].

[54] Unity Technologies. Character Controller component reference. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-CharacterController.html. [cit. 2024-05-02].

[55] Unity Technologies. NavMesh Agent. Online. Unity Documentation. 2017, s. 1. Dostupné z: https://docs.unity3d.com/560/Documentation/Manual/class-NavMeshAgent.html. [cit. 2024-05-02].

[56] Unity Technologies. Physics 2D. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-Physics2DManager.html. [cit. 2024-05-02].

[57] Unity Technologies. Rigidbody2D. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/ScriptReference/Rigidbody2D.html. [cit. 2024-05-02].

[58] Unity Technologies. Collider 2D. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/Collider2D.html. [cit. 2024-05-02].

[59] Unity Technologies. Physics Material 2D. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-PhysicsMaterial2D.html. [cit. 2024-05-02].

[60] Unity Technologies. Audio Source. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-AudioSource.html. [cit. 2024-05-02].

[61] Unity Technologies. Audio Listener. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/class-AudioListener.html. [cit. 2024-05-02].

[62] Unity Technologies. Audio Mixer. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/AudioMixer.html. [cit. 2024-05-02].

[63] Unity Technologies. Unity Test Runner. Online. Unity Documentation. 2019, s. 1. Dostupné z: https://docs.unity3d.com/2019.1/Documentation/Manual/testing-editortestsrunner.html. [cit. 2024-05-02].

[64] Unity Technologies. Debug C# code in Unity. Online. Unity Documentation. 2024, s. 1. Dostupné z: https://docs.unity3d.com/Manual/ManagedCodeDebugging.html. [cit. 2024-05-02].

[65] SCHELL, Jesse. The art of game design: a book of lenses. 3rd edition. Boca Raton: CRC Press, Taylor & Francis Group, [2020]. ISBN 978-1-138-63205-9.

[66] Game-Ace. What Is The Best Way To Make Educational Games? Online. 2023, s. 1. Dostupné z: https://game-ace.com/blog/how-to-make-educational-game/. [cit. 2024-05-03].

[67] rpgwizardorg. What Defines a 2D Game: Exploring the Essence of 2D RPG Games. Online. 2D Computer Game RPG Hub. 2024, s. 1. Dostupné z: https://www.rpgwizard.org/what-defines-a-2d-game-exploring-the-essence-of-2d-rpg-games/#. [cit. 2024-05-03].

[68] rpgwizardorg. What does RPG mean in games: A Comprehensive Guide to Multiplayer RPGs. Online. 2D Computer Game RPG Hub. 2024, s. 1. Dostupné z: https://www.rpgwizard.org/what-does-rpg-mean-in-games-a-comprehensive-guide-to-multiplayer-rpgs/. [cit. 2024-05-03].

[69] FLUKEOUT, GitHub - flukeout/css-diner: CSS Diner. GitHub Online. Dostupné z: https://github.com/flukeout/css-diner

[70] WINTER, Nick; SAINES, George a ERICKSON, Scott. CodeCombat. Online. 2013, 2024. Dostupné z: https://codecombat.com. [cit. 2024-05-08].

[71] SCRATCH FOUNDATION. Scratch - imagine, program, share. Online. 2003, 2022. Dostupné z: https://scratch.mit.edu. [cit. 2024-05-08].

[72] TYNKER. Tynker: Coding For Kids, Kids Online Coding Classes & Games. Online. 2012, 2024. Dostupné z: www.tynker.com. [cit. 2024-05-08].

[73] CODINGAME. CodinGame. Online. 2012, 2024. Dostupné z: https://www.codingame.com/. [cit. 2024-05-08].

[74] UNITY TECHNOLOGIES. Plans and Pricing. Online. Unity. C2024. Dostupné z: https://unity.com/products. [cit. 2024-05-09].

[75] Unity Technologies. Install the Unity Hub. Online. Unity Documentation. S. 1. Dostupné z: https://docs.unity3d.com/hub/manual/InstallHub.html. [cit. 2024-05-02].

[76] Macondo. What is the Unity Hub? Online. Unity Support. 2024, s. 1. Dostupné z: https://support.unity.com/hc/en-us/articles/360061586571-What-is-the-Unity-Hub. [cit. 2024-05-02].

[77] UNITY TECHNOLOGIES. Unity Asset Store - The best assets for game making. Unity Asset Store. Online. C2024. Dostupné z: https://assetstore.unity.com/ [cit. 2024-05-02]

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

2D – dvoudimenzionální

3D – trojdimenzionální

AI – Artificial Intelligence

API – Application Programming Interface

AR – Augmentovaná realita

CSS – Cascading Style Sheets

HTML – Hypertext Markup Language

LINQ – Language Integrated Query

MIT – Massachusetts Institute of Technology

NPC – Non-Player Character

OOP – Objektově orientované programování

OS – Operační systém

RPG – Role-Playing Game

UI – User Interface

VR – Virtuální realita

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH