**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačního inženýrství**

# Bakalářská práce

## Vývoj a optimalizace vlastní videohry v prostředí Unity

**Petr Anděl**

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Petr Anděl

Informatika

Název práce

**Vývoj a optimalizace vlastní videohry v prostředí Unity**

Název anglicky

**Development and optimization of own video game in Unity engine**

---

**Cíle práce**

Cílem práce je optimalizovat a rozšířit existující počítačovou hru vytvořenou pomocí frameworku Unity. Prvním dílčím cílem je analyzovat stávající implementaci a na základě této analýzy provést optimalizaci stávajícího řešení. Druhým dílčím cílem je navrhnout a implementovat novou funkcionalitu pro tuto hru.

**Metodika**

Bakalářská práce sestává ze dvou částí, teoretických východisek a vlastního řešení.

Metodika zpracování teoretické části vychází ze studia odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována východiska pro zpracování praktické části.

Praktická část práce spočívá v optimalizaci a rozšíření konkrétní hry implementované v prostředí Unity. Bude provedena analýza stávající implementace a budou navrženy možnosti její optimalizace, které budou posléze implementovány. Dále bude navržena a implementována nová funkcionalita této hry.

V průběhu řešení bude využito standardních metod a nástrojů softwarového inženýrství.

**Doporučený rozsah práce**

35-40 stran

**Klíčová slova**

Unity, vývoj, optimalizace, videohra, C#, skript, software

---

**Doporučené zdroje informací**

BISHOP, J. M. C#: návrhové vzory. Brno: Zoner Press, 2010. Encyklopedie Zoner Press. ISBN 978-80-7413-076-2.

Dokumentace jazyka C#, web: https://docs.microsoft.com/en-us/dotnet/csharp/

Dokumentace Unity 2019.3 – Editor manual, web: https://docs.unity3d.com/2019.3/Documentation/Manual/index.html

Dokumentace Unity 2019.3 – Scripting reference, web: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/

RAJLICH, Vaclav. Software Engineering : The Current Practice [online]. CRC Press. ISBN 9781466510357.

---

**Předběžný termín obhajoby**

2021/22 ZS – PEF

**Vedoucí práce**

Ing. Jiří Brožek, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 15. 11. 2021

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 16. 01. 2022

**Čestné prohlášení**

Prohlašuji, že svou bakalářskou práci "Vývoj a optimalizace vlastní videohry v prostředí Unity" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2022 _____

**Poděkování**

Rád bych touto cestou poděkoval svému vedoucímu bakalářské práce Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení a ochotnou pomoc v podobě konzultací v průběhu zpracovávání práce.

# Vývoj a optimalizace vlastní videohry v prostředí Unity


## Abstrakt

Tato bakalářská práce se zabývá optimalizací a rozšířením původní implementace konkrétní počítačové hry vyvíjené v prostředí Unity za použití objektově orientovaného programovacího jazyka C#.

V teoretické části práce jsou čtenáři přiblíženy základní pojmy či východiska z oblastí vývoje softwaru, do kterých tato práce zasahuje. V praktické části práce je zachycen výsledek provedené analýzy společně s navržením a realizací konkrétních dílčích změn. Dále je v práci popsáno začlenění nové, rozsáhlejší funkce ovlivňující větší část dosavadní implementace. Práce taktéž obsahuje průběžné ukázky zdrojového kódu hry, které vhodně doplňují provedené změny či demonstrují aplikování teoretických poznatků.



**Klíčová slova:** Unity, vývoj, optimalizace, videohra, C#, skript, software

# Development and optimization of own video game in Unity engine

**Abstract**

This Bachelor thesis deals with optimization and extension of the original implementation of a specific computer game developed in the Unity environment using the object-oriented programming language C#.

In the theoretical part of the work, the reader is approached with basic concepts and starting points from the areas of software development that this work interferes with. The practical part of the work shows the result of the analysis carried out together with the design and implementation of specific changes. The work also describes the implementation of a new, larger function affecting a larger part of the implementation to date. The work also includes samples of the source code of the game, which suitably complement the changes made and demonstrate the application of theoretical knowledge.

**Keywords**: Unity, development, optimization, video game, C#, script, software

# Obsah

# Seznam obrázků

# 1 Úvod

Vývojové prostředí Unity se v dnešní době mimo jiné řadí také mezi populární nástroje pro usnadnění tvorby počítačových videoher. Poskytuje především soubor již fungujících tříd a metod, které utváří základní logickou vrstvu, na které lze dále stavět. Kromě toho je pro vývojáře dostupný i přidružený editor, kde lze definovat a utvářet strukturu a výslednou podobu projektu.

Předmětem této práce je úprava a vylepšení jedné konkrétní počítačové hry, která byla v tomto vývojovém prostředí v minulosti sestavena. Práce je zaměřena především na specifické změny a problémy, které byly definovány v rámci úvodní analýzy projektu, čímž jsou čtenáři přiblíženy aplikované postupy řešení konkrétních situací včetně ukázek zdrojového kódu.

# 2 Cíl práce a metodika

## 2.1 Cíl práce

Cílem práce je optimalizovat a rozšířit existující počítačovou hru vytvořenou pomocí frameworku Unity. Prvním dílčím cílem je analyzovat stávající implementaci a na základě této analýzy provést optimalizaci stávajícího řešení. Druhým dílčím cílem je navrhnout a implementovat novou funkcionalitu pro tuto hru.

## 2.2 Metodika

Bakalářská práce sestává ze dvou částí, teoretických východisek a vlastního řešení.

Metodika zpracování teoretické části vychází ze studia odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována východiska pro zpracování praktické části.

Praktická část práce spočívá v optimalizaci a rozšíření konkrétní hry implementované v prostředí Unity. Bude provedena analýza stávající implementace a budou navrženy možnosti její optimalizace, které budou posléze implementovány. Dále bude navržena a implementována nová funkcionalita této hry.

V průběhu řešení bude využito standardních metod a nástrojů softwarového inženýrství.

# 3 Teoretická východiska

Tato část práce slouží pro shrnutí teoretických poznatků nabytých před započetím vypracovávání této práce či při jejím vypracovávání. Na následující text lze pohlížet jako na výpis nezbytně nutných informací pro pochopení dalších částí práce.

## 3.1 Objektově orientované programování

Objektově orientované programování je dnes běžnou softwarovou technologií či přístupem k řešení problémů – programovacím paradigmatem. Objektově orientované jazyky využívají pro řešení různých problémů spojených s komplexností požadovaného výsledku především abstrakci. O abstrakci lze hovořit u objektů, které sdílejí či napodobují vlastnosti a parametry jejich protějšků v reálném světě ve formě metod a poskytnutých dat. Lze také definovat abstrakci na úrovni existence různých tříd, které sdílejí vlastnosti podobných objektů [1].

Lze tedy říci, že objektově orientované programování je založeno na konceptu objektů obsahujících data v podobě atributů či proměnných a funkce ve formě metod definující specifické chování.

## 3.2 Návrhové vzory

Při psaní a definování struktury kódu programu lze použít takzvané návrhové vzory. Návrhový vzor je možné definovat jako pojmenované a popsané řešení typického problému, které je použito z důvodu uplatnění strukturovaného přístupu, který zajistí efektivní a přehledné složení a propojení jednotlivých částí kódu [2].

Pro objektově orientovaný jazyk C#, který byl v této práci využit, existuje velké množství různých návrhových vzorů. Níže jsou vybrány a blíže specifikovány takové návrhové vzory, u kterých bylo předpokládáno, že se budou v projektu, který je předmětem této práce, vyskytovat či bude vhodné je v projektu uplatnit, pokud se pro to naskytnou vhodné podmínky.

### 3.2.1 Kompozit (Composite)

Pomocí vzoru Kompozit lze vytvořit strukturované hierarchie, díky čemuž lze s jednotlivými komponenty pracovat stejně jako se skupinami komponentů. Mezi typické operace, kterých

se tento návrhový vzor týká, lze zařadit například přidávání, odstraňování, zobrazování, vyhledávání či seskupování komponentů [3].

Jako příklad, kde takový vzor aplikovat, je možno uvést fotoalbum, přehrávač hudby či jakýkoliv jiný program, který pracuje s vícero stejnými či podobnými objekty.

### 3.2.2 Jedináček (Singleton)

Účelem tohoto vzoru je zajistit, aby v daném programu či projektu existovala pouze jedna instance třídy a že k dané třídě (objektu) bude existovat jen jeden globální přístupový bod. Důsledkem by mělo být to, že je daná třída instanciována jen jednou a že všechny požadavky jsou směřovány k tomuto jedinému objektu. Daný objekt by taktéž neměl být vytvořen do té doby, dokud není skutečně potřeba. Za toto chování by měla být při užití tohoto vzoru zodpovědná samotná třída, nikoliv klient třídy [4].

### 3.2.3 Stav (State)

Na tento vzor lze pohlížet jako na dynamickou verzi vzoru Strategie (Strategy), pomocí kterého je typicky zapisována sada instrukcí chování pro daný objekt. Podobné využití má i vzor Stav s tím, že pokud se stav uvnitř objektu změní, může být změněno jeho chování přepnutím na sadu jiných operací či instrukcí. Toho může být dosaženo například nahrazením objektu jiným objektem ze stejné hierarchie tříd [5].

### 3.2.4 Šablonová metoda (Template method)

Tento vzor definuje možnost rozdělit jednotlivé kroky algoritmu do podtříd či metod. Struktura daného algoritmu se nemění, nicméně malé, dobře definované části jeho operací jsou řešeny na různých místech [6].

Jedná se víceméně o dodržování takové struktury kódu, kdy každá podtřída či metoda řeší pouze jeden konkrétní, ideálně ne moc komplexní problém či úlohu. Tím lze zajistit nejen lepší přehlednost sepsaného kódu, ale i zvýšit znovupoužitelnost jednotlivých dílčích částí.

## 3.3 Analýza a implementace plánovaných změn

K implementaci změn do existujícího projektu může docházet z řady různých důvodů. Všechny tyto důvody mají ale stejnou podstatu – cílem je úprava aktuálního chování softwaru, na který je změna zaměřena. Aby k této změně chování mohlo dojít bez zbytečných rizik a komplikací, je potřeba si vymezit očekávané dopady změny a přizpůsobit tomu zamýšlené řešení v podobě úpravy zdrojového kódu.

### 3.3.1 Analýza dopadu změny

„Impact analysis“ aneb analýzu dopadu lze označit za jednu z fází vývoje či modifikace softwarového projektu. Během této fáze by mělo dojít k co nejreálnějšímu odhadu či předpovědi požadovaných modifikací a následnému výběru nejvhodnějšího řešení či strategie.

Sada počátečního dopadu

Během fáze analýzy jsou například identifikovány oblasti projektu (zdrojového kódu), kde později dojde ke změně – takové oblasti pak lze zahrnout do množiny označované jako „sada počátečního dopadu“. Pokud se jedná pouze o malou změnu, jsou do této množiny zahrnuty veškeré (i podpůrné a vedlejší) moduly a části projektu, kde se změna má pravděpodobně projevit [7].

Sada odhadovaného dopadu

Pokud se jedná o větší změny v rozsáhlejším projektu, může jedna konkrétní změna přesahovat do několika modulů či částí projektu. Při takových změnách sada počátečního dopadu obsahuje pouze nejpodstatnější části projektu, které mají být zahrnuty (neobsahuje veškeré, podrobně členěné moduly). Podrobněji členěné moduly jsou pak zpravidla zahrnuty například v množině, kterou lze označit jako „sada odhadovaného dopadu“ či jiných množinách obsahující konkrétnější informace [7].

Propagační subjekty

Během této analýzy je taktéž doporučeno nalézt a označit části projektu, které jsou součástí změny, nicméně ke změně uvnitř těchto částí projektu nedojde. Takové části projektu mohou být například jednotlivé třídy či skripty, které se samy nemění, ačkoliv provedenou změnu dále šíří a interpretují svým sousedům (přidruženým třídám či skriptům), které je v průběhu

implementace změny taktéž nutno překontrolovat, aby nedošlo k nežádoucím a hůře zachytitelným chybám. Takové třídy či skripty lze označit jako „propagační subjekty“ [8].

### 3.3.2 Implementace změn

Během fáze implementace změn většinou dochází k připsání nové funkcionality v podobě nového kódu, který musí být následně řádně připojen a zabudován ke zbytku starého zdrojového kódu. Při tomto začlenění je taktéž kladen důraz na kontrolu dříve zmíněných propagačních subjektů. Způsob provedení změn v projektu se odvíjí především od jejich rozsahu. K různě obsáhlým změnám je přistupováno různými způsoby. Obecně lze definovat několik základní skupin změn [9].

Změny malého rozsahu

První skupinou jsou změny malého či středně malého rozsahu, které ovlivňují jen malé či dílčí části projektu. V takových případech je změna většinou implementována modifikováním již existujících tříd či skriptů – tvorba nových tříd či skriptů by mohla být zbytečně komplikovaná či rozsáhlá [10].

Změny velkého rozsahu

Druhou skupinu tvoří větší změny, kde je již potřeba kromě úpravy staré implementace také zahrnout tvorbu nových tříd či skriptů. V takovém případě musí následně dojít k výše zmíněnému začlenění nového kódu do starého kódu. V tomto případě lze narazit na několik navzájem se lišících situací.

V případě, že navrhovaná změna obsahuje implementaci funkcí, které nejsou ani okrajově zahrnuty v původním kódu, dochází především k připsání nových tříd či skriptů, které se označují jako „new supplier classes“ nebo-li nové doplňující (dodavatelské) třídy, které jsou do projektu začleněny jako nové deklarace.

Naopak v případě, kdy se dílčí části nově požadované funkce již v projektu nachází, je potřeba tyto dílčí části upravit či doplnit a poskládat tak, aby fungovaly jako celek. V takovém případě se dané třídy označují jako třídy nového klienta („new client classes“).

Pokud již ve starém kódu existuje původní (zastaralá) verze požadované funkce, která má být nahrazena novou sadou instrukcí (novým kódem), je ve většině případů starý modul či zdrojový kód zcela odstraněn a nahrazen novým modulem (zdrojovým kódem). Taková výměna obvykle znamená rozšíření provedených změn ve všech možných směrech přes přidružené propagační subjekty a kontrola a aktualizace interakcí všech zasažených částí projektu je tak neméně rozsáhlou a náročnou součástí daného procesu [10].

### 3.3.3 Příklad

Jako příklad lze uvést situaci, kdy máme počítačovou hru, kde hráč ovládá postavu. V aktuální implementaci může postava chodit pouze doleva či doprava bez možnosti skoku do výšky. Změna má zahrnovat přidání možnosti výskoku do výšky.

Při analýze například zjistíme, že hra obsahuje animace postavy a možnost změny ovládání (přemapování kláves). V takovém případě by bylo předpokládáno, že se změna bude týkat několika modulů – logiky pohybu hráče, animací pohybu, nastavení ovládání a vizuální interpretace změn v uživatelském nastavení ovládání, které by byly zahrnuty v sadě počátečního dopadu. Jejich dílčí či přidružené podpůrné části by byly zahrnuty v sadě odhadovaného dopadu. Jednotlivá propojení mezi těmito moduly a dalšími vnějšími funkcemi by byla označena jako propagační subjekty.

Při implementaci nové funkce by se jednalo o změnu většího rozsahu, která by byla provedena úpravou staré implementace.

### 3.3.4 Shrnutí

Úpravy kódu, ať už malé nebo velké, často narušují interakce mezi jednotlivými částmi projektu. Tyto narušené interakce musí být při každé takové úpravě opraveny. K opravě dochází podobným postupem jako při analýze dopadu změny, avšak tentokrát se již provádějí skutečné úpravy kódu. Tyto sekundární úpravy bývají menší než primární úpravy a provádějí se jen ve starém kódu [11].

## 3.4 Software pro vypracování praktické části

Při předešlém vývoji projektu, kterým se tato práce zabývá, byl využíván pouze bezplatný a volně dostupný software. Bezplatný a volně dostupný software byl proto využíván i nadále pro tuto práci. Jednalo se především o software uvedený níže.

### 3.4.1 Unity3D

Vizualizační a vývojový software, který lze zároveň označit za framework, na jehož základě byl projekt, jímž se tato práce zabývá, vytvořen. Projekt byl v době zpracovávání postaven na verzi Unity 2019.3 a v průběhu zpracovávání této práce k žádnému přechodu na novější či starší verzi softwaru nedošlo z důvodu zachování kompatibility a minimalizace vedlejších komplikací.

Součástí editoru Unity je i rozsáhlá sada interních skriptů a funkcí, které definují vlastní nastavení a vývojové prostředí jako takové (včetně implementace fyziky, metody vykreslování objektů a celkové správy projektu jakožto systému), na kterých lze konkrétní projekt nadále rozšiřovat.

### 3.4.2 GIMP 2.1

GIMP nebo-li Gnu Image Manipulation Program je rastrový grafický editor vhodný pro téměř jakoukoliv manipulaci s obrázky či fotografiemi. Daný software je dostupný na všech široce dostupných platformách (Linux, macOS, Windows) [12]. Jedná se o nástroj, pomocí kterého byla dříve vytvořena a následně upravována grafická stránka projektu.

### 3.4.3 Pomocná vývojová prostředí (IDE)

Prostředí Unity defaultně nedisponuje žádným editorem pro sepisování a úpravu zdrojového kódu (vlastních skriptů). Pro sepisování či úpravy vlastního kódu projektu byla proto před započetím vypracovávání této práce použita volně dostupná vývojová prostředí. Jmenovitě se jednalo především o vývojové prostředí Visual Studio Code a prostředí Atom. Oba editory disponují velkým počtem možností, jak je lze přizpůsobit pro práci na konkrétním projektu či pro práci se specifickým jazykem a předpokládá se, že budou využívány i pro účely vypracování této práce.

## 3.5 Pořadí spouštění událostí v prostředí Unity

Pro vhodné aplikování změn a sepsání nového kódu, který má být začleněn do prostředí Unity, bude pravděpodobně potřeba mít přehled o definovaném pořadí spouštění a činnostech interních funkcí, pomocí kterých lze získat širší přístup k ovládání a monitorování vývojového prostředí a jeho logiky. K některým z daných funkcí lze připisovat a vkládat vlastní bloky kódu, které pak ovlivňují chování výsledné aplikace na základě toho, do jaké z funkcí byly vloženy.

Jednotlivé funkce byly pro lepší přehlednost a čitelnost rozděleny do množin na základě jejich doby spuštění a definovaných činností s tím spojených. Každá z popisovaných funkcí má přiřazené číslo, které označuje pozici v pořadí, které určuje, v jakém jsou metody (poprvé) volány při jednom obnovovacím cyklu aplikace [13].

### 3.5.1 Inicializace

Tato fáze nastává při načtení scény projektu. Během ní je pro každý objekt dané scény zavoláno několik funkcí. Jedná se o funkce, pomocí kterých lze definovat (počáteční) stav objektů či skriptů přiřazených k daným objektům. Tyto funkce nejsou standardně volány při každém průchodu obnovovacím cyklem, nýbrž pouze při konkrétních událostech. V této fázi nalezneme především funkce:

- **Awake (1)** – užívá se například pro načtení komponentů a logiky skriptu, které by měly být dostupné již ve všech instancích funkce „Start“ či funkce „Update“ (napříč všemi ostatními skripty) [14].
- **OnEnable (2)** – je zavolána ve chvíli, kdy je objekt, ke kterému je skript přiřazen, přepnut ze stavu „neaktivní“ do stavu “aktivní“. V případě, že je při spuštění scény objekt nastaven jako aktivní, lze tuto funkci použít obdobně jako funkci „Awake“ [15].
- **Reset (3)** – je zavolána ve chvíli, kdy je v projektu skript přiřazen k objektu, ale vývojové prostředí se nenachází ve hracím režimu (např. při přepnutí z hracího režimu do editovacího režimu, přidání komponentu, ruční obnova scény atd.) [16].
- **Start (4)** – je zavolána vždy pouze jednou pro daný skript. Typicky zde lze definovat výchozí nastavení či zavolání dalších vlastních metod určující výchozí nastavení či logiku skriptu, kterou není potřeba volat opakovaně [17].

### 3.5.2 Fyzika

Po fázi inicializace následuje fáze, ve které figurují funkce spojené s aktualizací fyzikální stránky vývojového prostředí. Do daného procesu jsou zapojeny následující funkce:

- **FixedUpdate (5)** – volání této metody je nezávislé na snímkové frekvenci a používá se především pro fyzikální výpočty objektů. Může být zavolána i několikrát během jednoho snímku či naopak jen jednou během uplynutí několika snímků. Frekvence volání je standardně nastavena na 0.02 sekundy [18].
- **Internal animation update 1 (6)** – první množina funkcí, které aktualizují případnou polohu a vlastnosti objektů, které mají přiřazený animátor.
- **Internal physics update (7)** – Blíže nespecifikovaná množina funkcí, která zajišťuje aktualizaci pozic a vlastností objektů ve scéně.
- **Internal animation update 2 (8)** – druhá množina funkcí (odlišných od funkcí z množiny 1), které zapisují případnou provedenou změnu polohy a vlastností objektů, které ovlivnil animátor [13].
- **OnTriggerXXX (9)** – množina funkcí (OnTriggerEnter, OnTriggerStay, OnTriggerExit atd.), které zachycují kolizi či překrytí dvou či více objektů na základě definovaných kolizních hranic jednotlivých objektů [19].
- **OnCollisionXXX (10)** – množina funkcí (OnCollisionEnter, OnCollisionExit atd.), které zachycují kolizi dvou či více objektů na základě definovaných kolizních hranic jednotlivých objektů. Zde je na rozdíl od funkcí „OnTrigger“ volána v případě kontaktu jiná vnitřní funkce, která zahrnuje i parametry jako nárazovou rychlost, váhu a působení dalších podobných proměnných, které mohou kolizi ovlivnit [20].
- **Yield WaitForFixedUpdate (11)** – vyčká, dokud nemá proběhnout další tik metody „FixedUpdate“. Jedná se o funkci, od které lze opakovat cyklus funkcí zahrnujících fyziku projektu [21].

### 3.5.3 Vstup a herní logika (aktualizace stavu)

V této fázi se volají funkce, ve kterých lze definovat základní herní logiku v podobě hlavních metod pro aktualizaci stavu programu. Jedná se o funkce:

- **OnMouseXXX (12)** – množina funkcí (OnMouseOver, OnMouseDrag, atd.), které jsou zavolány při specifické interakci uživatele a překrytí kolizní zóny objektu kurzorem myši [22].

- **Update (13)** – je standardní funkce používaná pro zápis příkazů, které jsou závislé na průběžném obnovování při běhu programu. Daná funkce je během obnovovacího cyklu zavolána právě jednou [23].
- **Yield (null, WaitForSeconds, StartCoroutine etc.) (14)** – řada funkcí používaných uvnitř struktury „Coroutine“, na základě kterých lze definovat, řídit či upravovat chování dané struktury.
- **Internal animation update 1 + 2 (15)** – zavolání všech funkcí z obou množin pro aktualizaci animací [13].
- **LateUpdate (16)** – je zavolána ve chvíli, kdy byly zavolány metody „Update“ ve všech skriptech, které danou metodou disponují. Lze zde tak implementovat logiku, u které potřebujeme mít jistotu, že veškerá přidružená předcházející logika v metodě „Update“ napříč skripty byla vykonána [24].

### 3.5.4 Vykreslování scény

Tato fáze zahrnuje metody, které souvisí s vykreslováním objektů ve scéně. Lze tak v těchto metodách aplikovat logiku, která se má projevit až při samotném vykreslování.

- **OnWillRenderObject (17)** – bývá zavolána právě jednou pro každou aktivní kameru, pokud je nějaký objekt viditelný a lze ho vykreslit.
- **OnPreCull (18)** – je zavolána těsně předtím, než kamera zastaví (vyřadí) scénu.
- **OnBecameVisible (19)** – je zavolána, když se daný objekt stane viditelným pro jakoukoliv z dostupných kamer.
- **OnBecameInvisible (20)** – volána v případě, kdy se přiřazený objekt stane neviditelným pro jakoukoliv z dostupných kamer.
- **OnPreRender (21)** – zavolána předtím, než kamera začne vykreslovat scénu.
- **OnRenderObject (22)** – je volána poté, co jsou všechna předešlá vykreslování dokončena. Lze zde použít například „Graphics.DrawMeshNow“ pro vykreslení geometrických obrazců.
- **OnPostRender (23)** – volána ve chvíli, kdy kamera dokončila vykreslování scény
- **OnRenderImage (24)** – lze použít pro aplikování „Post-processing“ efektů [13].

### 3.5.5 Další vykreslování

Kromě vykreslování, které se týká objektů ve scéně a zobrazení objektů ve výsledném programu, vývojové prostředí Unity poskytuje i funkce sloužící převážně pro testování a vývoj projektu.

- **OnDrawGizmos (25)** – určená pro vykreslování značek a obrazců do náhledu scény v editoru vývojového prostředí. Uplatňuje se pouze při vývoji, ve výsledném produktu se dané značky a obrazce nezobrazují.
- **OnGUI (26)** – voláno automaticky několikrát během jednoho vykreslování snímku. Slouží pro zachycení odezvy GUI událostí. Nejprve vrací odezvu z interních funkcí „Layout“ a „Repaint“, následně například odezvu z klávesnice a myši (vstupu uživatele) [13].

### 3.5.6 Ukončení

V poslední fázi se volají funkce, ve kterých lze ošetřit chování při dokončení jednoho cyklu obnovení či přerušení cyklu obnovení a vypnutí programu.

- **Yield WaitForEndOfFrame (27)** - volá se po dokončení veškerého vykreslování kamery včetně GUI elementů, těsně předtím, než je snímek zobrazen uživateli.
- **OnApplicationPause (28)** – zavolána pro všechny objekty při pozastavení scény
- **OnApplicationQuit (29)** – zavolána pro všechny objekty předtím, než je aplikace ukončena nebo pokud vývojář vypne hrací mód.
- **OnDisable (30)** – je zavolána ve chvíli, kdy je objekt, ke kterému je skript přiřazen, přepnut ze stavu „aktivní“ do stavu “neaktivní“.
- **OnDestroy (31)** – zavolána při přepnutí z jedné scény do druhé (při ukončení scény) či při celkovém vypnutí aplikace [13].

# 4 Vlastní práce

## 4.1 Rozdělení činností praktické části

Praktická část této práce se skládala z analýzy projektu, tvorby ve vývojovém prostředí Unity a sepisování kódu v jazyce C#. Tyto činnosti byly na sobě vzájemně závislé.

Příklad: V editoru lze navrhnout a sestavit novou množinu objektů (např. novou část uživatelského rozhraní), která ale nenabude požadovaných funkcí (nebude schopna plnit požadovanou úlohu) bez přiřazení či implementace vhodných skriptů (sepsaného kódu). Stejně tak ve většině případů nelze uplatnit chování či funkce skriptů, které nejsou přiřazeny ke konkrétním objektům v editoru za konkrétním účelem. K efektivnímu řešení stanovených cílů je taktéž potřeba hlubší znalost a orientace nejen v oblasti softwaru, se kterým se pracuje, ale i znalost konkrétního projektu, které je docíleno úvodní analýzou projektu.

### 4.1.1 Tvorba ve vývojovém prostředí

V případě tvorby ve vývojovém prostředí Unity se jednalo převážně o práci s objekty či množinami objektů a jejich veřejnými parametry.

Náplň této činnosti spočívala ve větší míře v úpravě již existujících objektů dle nových požadavků, v menší míře pak v tvorbě a začlenění úplně nových objektů, které se v projektu doposud nenacházely. Nové požadavky představovaly jak vizuální, tak i funkční úpravy, které přímo či nepřímo navazovaly na psaní kódu.

### 4.1.2 Psaní a úpravy zdrojového kódu

Manipulaci s kódem lze na základě informací z teoretických východisek dále rozdělit na dvě hlavní podmnožiny.

Do první z podmnožin spadá návrh a sepsání zcela nových skriptů, které jsou psány za účelem implementace nových funkcí. Do druhé z podmnožin pak lze zařadit úpravy již vytvořených a do projektu začleněných skriptů.

K úpravě skriptů při zpracovávání této práce docházelo z několika důvodů:

- Do skriptu bylo potřeba připsat dodatečnou metodu a bylo zbytečné ji implementovat samostatně do nového skriptu
- Ve skriptu bylo potřeba upravit funkci již existující metody
- Skript byl (např. z důvodu postupného rozšiřování) špatně strukturovaný (nebylo dodrženo správné užití návrhového vzoru) a bylo potřeba provést celkovou revizi a následné přepsání zdrojového kódu z důvodu lepší orientace a snazších úprav daného skriptu v budoucnu

## 4.2 Analýza projektu

Pro vhodné provedení optimalizace a uplatnění standardů programování a informací z literatury studované v rámci teoretické části práce bylo potřeba projít pracovní verzi projektu a zároveň otestovat projekt jakožto výsledný produkt.

Navržení nové funkcionality produktu se odvíjelo od poznatků získaných během testování výsledného produktu a subjektivního uvážení autora této práce na základě již implementovaných funkcionalit, plánovaných cílů a dostupné časové dotaci pro tuto práci.

Produktem (předmětem zájmu této práce) byla počítačová videohra, kterou autor této práce v minulosti sám vytvořil. Pro bližší specifikaci se jednalo o 2D RPG akční hru dostupnou především na systémech Windows, Linux a MacOS, která byla od roku 2020 dostupná na internetové herní platformě Steam.

Testování výsledného produktu (hry) bylo věnováno přibližně 80 hodin před započetím vypracovávání této práce a přibližně dalších 40 hodin při samotném vypracovávání. Testováním je v tomto případě myšleno mnohočetné procházení dílčích částí hry a testování různých scénářů, ve kterých se může v průběhu hraní hry koncový uživatel ocitnout.

Analýza pracovní verze projektu byla zaměřena především na vlastní dříve sepsané skripty. Projekt při zahájení analýzy obsahoval přibližně 218 vlastních skriptů. Většina těchto skriptů plnila pouze menší dílčí úlohy, ze kterých se skládala hlavní herní logika.

V průběhu analýzy byla pozornost směřována především na:

- Funkce a jejich povely, které se ve skriptech nacházely
- Celkovou strukturu a členitost skriptů
- Propojení a vazby mezi jednotlivými skripty, na základě kterých bylo možné definovat podmnožiny a zkoumané skripty do nich roztřídit

### 4.2.1 Nalezené návrhové vzory

Dle struktury, obsahu a funkcí těchto skriptů bylo rozpoznáno aplikování několika návrhových vzorů a celková struktura projektu. V projektu bylo nalezeno použití těchto návrhových vzorů:

- Vzor Stav (State)
- Vzor Šablonová metoda (Template Method)
- Vzor Kompozit (Composite)

Všechny tyto vzory jsou podrobněji popsané výše v teoretické části práce. V průběhu vypracovávání praktické části práce byl kladen důraz na dodržení daných návrhových vzorů v případě, že se manipulovalo se skriptem obsahující dané návrhové vzory, případně v situaci, kdy se vytvářel nový skript, který měl být přímo či nepřímo napojen na skripty, ve kterých se návrhové vzory vyskytovaly.

### 4.2.2 Struktura projektu dle skriptů

Na základě provedené analýzy skriptů a jednotlivých uskupení objektů a scén ve vývojovém prostředí Unity bylo možné definovat několik subsystémů, ze kterých se projekt skládá.

Skripty byly pro demonstrativní účely a další činnosti rozděleny do množin definovaných dále v této práci. Některé skripty, plnící funkci propagačních subjektů, mohou mít přesah do více množin najednou.

Množina 1: Reprezentace a chování postavy ovládané hráčem

Do této kategorie spadají veškeré skripty, které definují a upravují chování postavy, či jinak zasahují do reprezentace postavy, kterou uživatel ovládá. Jedná se o skripty zajišťující pohyb a animace postavy, skripty definující statistiky hráče a odezvu na různé podněty.

Množina 2: Interaktivní předměty

V této množině lze najít skripty, které slouží k manipulaci interaktivních předmětů v rámci hráčem vyvolaných událostí. Spadají sem skripty definující dané předměty, obchodování s předměty, jejich uchování v úložišti (truhlách či hráčově batohu) a účinky při jejich použití.

Množina 3: Vykreslení uživatelského rozhraní

Zde lze najít skripty zajišťující zobrazování informací a vykreslování grafických prvků na uživatelské rozhraní, tedy na vrstvu, která se uživateli zobrazuje „před" sestavenou scénou. Byly sem zařazeny skripty vykreslující herní nabídku a její jednotlivé sekce, dialogy mezi hráčem a jinou postavou, hráčovy statistiky, obsah hráčova batohu atp.

Množina 4: Ostatní objekty herního prostředí

Tato množina zahrnuje skripty definující podobu herního prostředí a jeho logiku, která je hráči prezentována v podobě sestavených množin objektů v jednotlivých scénách. Jedná se tedy o skripty definující chování specifických objektů (změna stavu, deformace a animace prostředí), chování nepřátelských jednotek, provoz záchytných bodů atp.

Množina 5: Nastavení, ovládání a korigování běhu hry

Do této množiny byly zařazeny skripty sloužící k zajištění celkového fungování jednotlivých dílčích částí projektu. Jedná se o skripty ukládání a načítání dat, skripty nastavení uživatelských preferencí, ovládání a zobrazení, skripty pro přepínání jednotlivých scén, uchování informací o stavu definovaných objektů a skripty řídící načítání a spouštění definovaných komponentů.

### 4.2.3 Struktura projektu dle scén

Podobné dělení do množin na základě společných prvků bylo provedeno i pro jednotlivé scény projektu.

Množina 1: Scény hlavní nabídky

V této množině se nachází scény, ve kterých je hráči zobrazena např. úvodní obrazovka a nastavení, obecné informace o hře, výběr herního režimu či výběr konkrétní úrovně.

Množina 2: Scény herního prostředí

Do této množiny spadají všechny scény, ve kterých může uživatel ovládat svou postavu. Jedná se tedy o jednotlivé úrovně obou dostupných herních režimů, které lze dále dělit na jednotlivé kapitoly či podmnožiny na základě charakteristik prostředí (objektů) v jednotlivých scénách.

### 4.2.4 Části projektu vyhodnocené jako vhodné pro optimalizaci

Na základě testování a analýzy projektu byly po syntéze s informacemi z teoretické části práce vybrány pro optimalizaci tyto části:

Objekty uživatelského rozhraní a nastavení rozlišení herního okna

Pro věrohodnější a profesionálněji působící dojem z výsledného produktu bylo potřeba upravit rozvržení prvků uživatelského rozhraní hry pro správné zobrazení při aplikování poměru stran 4:3 a jemu podobným. Před zahájením zpracovávání této práce hra plně podporovala pouze poměr stran 16:9 a blízké alternativy, přičemž při vyobrazení při poměru stran 4:3 a jemu podobným docházelo k nežádoucím chybám.

U této změny spadají do sady počátečního dopadu dříve definované množiny 3 a 5, tedy množiny skriptů zajišťujících vykreslování uživatelského rozhraní a skriptů nastavení a korigování běhu hry. Z hlediska zdrojového kódu by se množina číslo 3 dala taktéž vnímat spíše jako množina obsahující propagační subjekty.

Skripty objektů reprezentující úložný prostor hráče

Skripty, které obsahovaly metody a funkcionalitu ovládání objektů uživatelského rozhraní představující úložný prostor hráče, obsahovaly množství redundantního kódu, který mohl být nahrazen jinou, efektivnější formou interpretace příkazů.

Při této změně spadá do sady počátečního dopadu dříve definovaná množina 2. Předpokládá se, že množina 1 (skripty reprezentace a chování postavy) nebude v tomto případě zahrnuta či bude obsahovat skripty plnící propagační úlohu společně s množinou 3 (vykreslování uživ. rozhraní). Hlavní bod zájmu se nachází v množině 2 (skript úložného prostoru hráče, který je součástí sady skriptů pro manipulaci s interaktivními předměty). Vzhledem k povaze

změny by neměla být tato změna mimo množinu 2 znatelná a propagační subjekty byly určeny pouze pro následnou kontrolu, že tomu tak skutečně bude.

Popisy interaktivních předmětů

Před zahájením zpracovávání této práce obsahovaly objekty pro zobrazování textových řetězců (názvy, popisy) interaktivních objektů pouze anglickou verzi (anglický překlad) textu, ačkoliv byl zbytek projektu (výsledné hry) dostupný jak v anglickém, tak i v českém jazyce.

V tomto případě se opět jedná o změnu, která bude prováděna především v množině číslo 2, přičemž změna bude pro koncového uživatele znatelná především kvůli skriptům z množiny číslo 3, které bude nutné po připsání nového kódu správně napojit či nový kód uzpůsobit tak, aby ke změnám dojít nemuselo.

Skript sloužící pro řízení ukládání a načítání dat

Skript, který obsahoval souhrn metod představující ukládání dat do souboru a zároveň pak i jejich opětovné načítání ze souboru, obsahoval špatně strukturovaný kód a příliš robustní metody, díky čemuž mohlo dojít k nežádoucímu chování při ukládání či načítání dat. Ve skriptu bylo nalezeno použití třídy „BinaryFormatter“, který v současné době není podporován a je vyhodnocen jako nebezpečný. Skript byl proto vyhodnocený jakožto vhodný k celkovému přepsání.

Tato změna vzhledem ke své povaze zasáhne pouze skripty z množiny číslo 5. Nepočítá se s tím, že by změna měla přesah do jiné množiny.

### 4.2.5 Části projektu vyhodnocené jako vhodné pro další rozvoj

Nastavení ovládání

Hra obsahovala řadu hráčem spustitelných či proveditelných akcí (událostí), které byly pevně přiřazené k určitému vstupu ovládání (tlačítka klávesnice či myši) a nemohly být před spuštěním hry a ani v jejím průběhu přenastaveny na jiné, hráčem preferované vstupy ovládání. Vzhledem k většímu množství dostupných akcí a absenci možnosti měnit jejich vstupy ovládání bylo rozhodnuto tuto možnost změny konkrétních vstupů ovládání pro konkrétní akce implementovat.

V tomto případě se jedná o přidání zcela nové funkce, která prozatím nemá v projektu své zastoupení ani v podobě dílčích částí, tudíž půjde o tvorbu nových doplňujících (dodavatelských) tříd v rámci úprav velkého rozsahu. Stará implementace zachycení uživatelských vstupů (která je s tímto úzce spojena) je postavena na konkrétních systémových funkcích, které jsou součástí vývojového prostředí, a tudíž tyto funkce nebudou odebrány, nýbrž bude navrhnuto takové řešení, které bude na těchto systémových funkcích dále stavět.

### 4.2.6 Shrnutí

Po provedení analýzy projektu si autor práce stanovil 4 dílčí části optimalizace. Dvě z těchto dílčích optimalizací jsou zaměřeny především na předělání struktury zdrojového kódu vybraných skriptů. Pro koncového uživatele by tyto změny neměly mít žádný vizuální dopad – změny mohou být znatelné pouze z funkčního pohledu.

Další dílčí optimalizace je zaměřena na úpravu možností nastavení projektu a vylepšení grafické reprezentace objektů vykreslujících hlavní nabídku hry. Tato změna by měla být znát jak z funkčního, tak i z vizuálního hlediska.

Poslední dílčí optimalizace je opět zaměřena na změny ve zdrojovém kódu. Ačkoliv tato dílčí optimalizace neobsahuje žádné přímé zásahy do vizuální stránky projektu, úpravou zdrojového kódu bude i přesto vizuální stránka projektu ovlivněna.

Všechny tyto změny by měly být v rámci malého až středně velkého rozsahu.

Pro implementaci nové funkcionality si autor práce stanovil přidání možnosti přenastavení ovládání základních akcí, které jsou ve hře dostupné. V tomto případě se jedná o změnu velkého rozsahu, při které pravděpodobně vznikne nový modul projektu a zároveň s tím bude několik již existujících modulů projektu ovlivněno.

## 4.3 Optimalizace 1: uživatelské rozhraní a nastavení rozlišení okna hry

Výsledný produkt neumožňoval v nastavení rozlišení změnit poměr stran zobrazovaného okna z poměru stran 16:9 na jiné alternativy. Tato změna nebyla dostupná především z důvodu, že po aplikaci změny poměru stran by došlo k chybnému zobrazení většiny prvků uživatelského rozhraní na několika na sobě nezávislých místech herní nabídky, což by pravděpodobně mělo negativní dopad na celkový dojem z produktu.

### 4.3.1 Návrh na optimalizaci

Cílem této optimalizace byla měla být úprava prvků uživatelského rozhraní, pomocí které by bylo docíleno toho, že po provedení optimalizace bude uživatelské rozhraní jakožto celek více responzivní a jeho jednotlivé prvky se budou zobrazovat co nejvhodnějším způsobem vůči standardnímu rozmístění.

Zároveň s tím by měla být tato optimalizace zaměřena na revizi skriptu, který obsahuje metody sloužící pro manipulaci s nastavením zobrazení, a připsání metody pro vyfiltrování a selekci podporovaných a nepodporovaných rozlišení v návaznosti na poměr stran herního okna a na přepsání metody pro přepínání mezi danými rozlišeními.

Výše zmíněný skript se nachází ve scéně hlavní herní nabídky, do které se může uživatel opakovaně vracet, což zapříčiňuje opětovnou inicializaci skriptu. Je tedy potřeba ošetřit tuto situaci, aby nedocházelo k automatickému přenastavení rozlišení okna programu pokaždé, kdy k takové situaci dojde.

### 4.3.2 Postup implementace optimalizace

Práce v editoru

Pro vykreslování uživatelského rozhraní je v projektu využito „Plátno“ nebo-li „Canvas“. Jedná se o jeden ze základních komponentů vývojového prostředí Unity. Všechny objekty představující uživatelské rozhraní musí být potomky tohoto „Plátna“ [25].

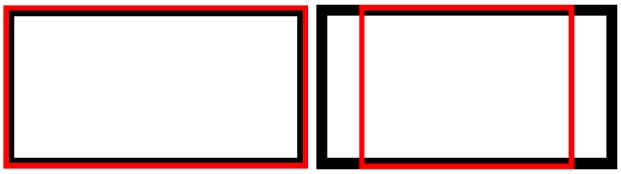
Každý z vybraných potomků (objektů) „Plátna“ defaultně obsahuje komponent „Rect Transform“, kde lze nastavit základní parametry objektu, jakými jsou například pozice na X, Y a Z souřadnicích, výška, šířka, rotace a předvolby ukotvení.

Nejprve bylo nutné najít objekty uživatelského rozhraní, které při změně zobrazení či poměru stran vykazovaly problematické chování. U těchto nalezených objektů pak bylo potřeba upravit parametry škálování, umístění a především předvolbu ukotvení.

U některých vybraných objektů úprava základních parametrů nestačila a takové objekty pak musely být dodatečně modifikované či zcela nahrazené jinými, více přizpůsobivými alternativami. V případě menších úprav byly úpravy provedeny v prostředí Unity, v případě větších úprav k úpravám docházelo pomocí programu GIMP.

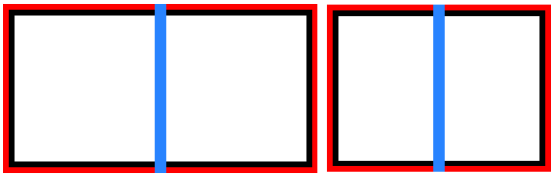
Příklad: Objekt představující rám ohraničení zobrazovaného okna se při změně poměru stran zobrazoval chybně. Vzhledem k povaze objektu ho nebylo možné pomocí parametrů zobrazení upravit tak, aby splňoval požadavky optimalizace. Demonstrováno na obrázcích dále.

- Červená barva představuje prostor, který je vykreslován uživateli
- Černá barva představuje vykreslení objektu rámu ohraničení
- Modrá barva představuje rozdělení objektů



*Obrázek 1 – původní vykreslení rámu UI jedním objektem, vlevo při poměru 16:9, vpravo při poměru 4:3 (zdroj: vlastní)*

V tomto konkrétním případě bylo nutné původní objekt odebrat a nahradit dvěma dílčími objekty, které jakožto celek plnily funkci původního objektu.



*Obrázek 2 - nové vykreslení rámu UI dvěma objekty, vlevo při poměru 16:9, vpravo při poměru 4:3 (zdroj: vlastní)*

Po aplikaci této úpravy a přenastavení parametrů ukotvení se při změně poměru stran rám ohraničení zobrazuje správně, případně se zobrazuje s minimálními odchylkami, které jsou

zanedbatelné. Podobné úpravy byly provedeny i u dalších, dříve chybně zobrazovaných objektů uživatelského rozhraní.

Psaní kódu

Okno, ve kterém se hra zobrazuje, má vypnutou funkci změny velikosti okna pomocí myši či systémových tlačítek pro maximalizaci / obnovení z maximalizace. Tato funkce byla dříve deaktivována nejen kvůli neresponzivně ukotveným objektům, jimiž se tato kapitola zabývá, ale taktéž kvůli možnosti snadného podvádění a obcházení zamýšlených herních mechanik pomocí přizpůsobení zobrazované části herního prostředí při hraní hry. Jediným způsobem, jak změnit rozlišení okna hry, tak zůstává uživatelské rozhraní nastavení v hlavní nabídce hry, které je ovládáno přes vlastní skript k tomu určený.

Do skriptu sloužícího pro manipulaci uživatelských nastavení bylo nutné připsat dvě nové metody pro inicializaci a přenastavení zobrazení okna hry.

Inicializace rozlišení

První metoda zastává funkci inicializace dostupných rozlišení připojené obrazovky a třídění nalezených rozlišení na podporovaná a nepodporovaná.

Nejprve je potřeba definovat počáteční stav a načíst dostupná rozlišení pomocí dostupných metod zahrnutých v Unity.

```
private void InitializeResolution(){
     resolutions = Screen.resolutions;
     resolutionDropdown.ClearOptions();
     resolutionFound = false;
     optimalResolutionFound = false;
     List<string> resOptions = new List<string>();
     currentResolutionIndex = 0;
     supportedResolutionIndex = 0;
```

*Ukázka kódu - skript SettingsUI; metoda inicializace rozlišení; 1. část*

Následně jsou všechna načtená rozlišení testována a tříděna. První část testování se zabývá hodnotami šířky a výšky zvoleného rozlišení. Pokud testované rozlišení nedosahuje minimálních požadavků na šířku nebo výšku, je později vyhodnoceno jako nepodporované. Tento stejný princip lze uplatnit i v případě, že by bylo potřeba limitovat maximální hodnoty pro šířku a výšku.

```
for(int i = 0; i < resolutions.Length; i++){
      resDivision =
      (float)resolutions[i].width / (float)resolutions[i].height;

      if(resolutions[i].width < windowSize_widthMin ||
      resolutions[i].height < windowSize_heightMin)
      {
            resDivision = 0;
      }
```

*Ukázka kódu - skript SettingsUI; metoda inicializace rozlišení; 2. část*

Druhá část třídění probíhá na základě podílu hodnot šířky a výšky u každého rozlišení. Po provedení změn u objektů v uživatelském rozhraní nabídky hry je hra přizpůsobena pro jakékoliv rozlišení, jehož podíl šířky a výšky vyjde jakožto hodnota mezi 1.1 a 1.8. Do těchto hodnot spadají všechna rozlišení s poměrem stran 16:9, 4:3 a variantami mezi nimi. Toto testování má identifikovat veškerá rozlišení, která by pro hru nebyla přípustná, např. širokoúhlá („ultra-wide“) rozlišení.

Pokud je rozlišení vyhodnocené jako podporované, je přidané do pole dostupných rozlišení a jeho index uložen do pomocné proměnné. V opačném případě, kdy je rozlišení vyhodnocené jako nepodporované, je z důvodu zachování správného indexování přiřazené do pole dostupných rozlišení, nicméně zároveň s tím je jeho index zaznamenán do pole představující nepodporovaná rozlišení.

```
if(resDivision > resDivisionMin && resDivision < resDivisionMax)
{
      string option =
      resolutions[i].width + " x " + resolutions[i].height;
      resOptions.Add(option);
      supportedResolutionIndex = i;
      supportedResolutionFound = true;

      if(resolutions[i].width == Screen.width &&
      resolutions[i].height == Screen.height)
      {
            currentResolutionIndex = i;
            resolutionFound = true;
      }
}
else
{
      string option = "not supported (" + resolutions[i].width
```

```
    + " x " + resolutions[i].height + ")";
    resOptions.Add(option);
    notSupportedResolutions.Add(i);
}
```

*Ukázka kódu - skript SettingsUI; metoda inicializace rozlišení; 3. část*

Dále je pak potřeba rozhodnout, jaké rozlišení bude uplatněno:

- Pokud bylo v průběhu testování nalezeno aktuálně používané rozlišení okna programu, je jeho index uložen v proměnné (v ukázkách kódu proměnná „currentResolutionIndex“) již z fáze testování. Ke změně rozlišení tedy v podstatě nedojde.
- Pokud aktuální rozlišení okna programu neodpovídá žádné z testovaných možností, je zvolené to nejvyšší podporované rozlišení, jehož index byl dosud uložený v proměnné (v ukázkách kódu proměnná „supportedResolutionIndex“). V takovém případě později dojde ke změně rozlišení okna programu.
- Pokud aktuální rozlišení okna programu neodpovídá žádné z testovaných možností a zároveň není nalezeno žádné podporované rozlišení, je zvolené nejvyšší dostupné (nepodporované) rozlišení. V takovém případě později taktéž dojde ke změně rozlišení okna programu.

```
    if(resolutionFound == false){
        if(supportedResolutionFound == true)
        {
            currentResolutionIndex = supportedResolutionIndex;
             Debug.Log
            ("current resolution index set to supported value");
        }
        else
        {
            currentResolutionIndex = resolutions.Length - 1;
            Debug.Log
            ("current resolution index set to unsupported value");
        }
    }
```

*Ukázka kódu - skript SettingsUI; inicializace rozlišení; 4. část*

V posledním kroku jsou dostupná rozlišení nahrána ve formě textových řetězců do definovaného komponentu uživatelského rozhraní, který je následně aktualizován.

Aktualizací daného komponentu zároveň dojde k případné změně rozlišení, jelikož je na daný komponent navázána metoda pro přepínání rozlišení, která je popsána dále.

```
resolutionDropdown.AddOptions(resOptions);
resolutionDropdown.value = currentResolutionIndex;
resolutionDropdown.RefreshShownValue();
```

*Ukázka kódu - skript SettingsUI; inicializace rozlišení; 5. část*

Přepínání rozlišení

Druhá metoda zastává funkci přepínání rozlišení na základě vstupního (zvoleného) indexu. Před přepnutím je nutné otestovat, zda-li je vybrané rozlišení podporováno nebo ne.

```
public void SetResolution(int resolutionIndex)
{
     bool _changeResolution = true;
     int _targetedResolutionIndex = resolutionIndex;
     foreach(int res in notSupportedResolutions)
     {
          if(res == _targetedResolutionIndex)
          {
               _changeResolution = false;
          }
     }
     Debug.Log("resolution change state: " + _changeResolution);
```

*Ukázka kódu - skript SettingsUI; přepínání rozlišení; 1. část*

Pokud není rozlišení na seznamu nepodporovaných rozlišení, je následně provedena změna rozlišení okna programu. Tato změna rozlišení nemá vliv na aktuální stav režimu celé obrazovky. Pokud je tedy při změně okno programu v režimu celé obrazovky, zůstává v něm i po změně. Pokud se v něm okno nenachází, nebude se v něm nacházet ani po aplikaci změny.

Pokud je rozlišení na seznamu nepodporovaných rozlišení, je provedeno nepřímé opětovné zavolání této funkce s původním indexem rozlišení pomocí komponentu uživatelského rozhraní, čímž je ponechané stávající rozlišení a ke změně nedojde.

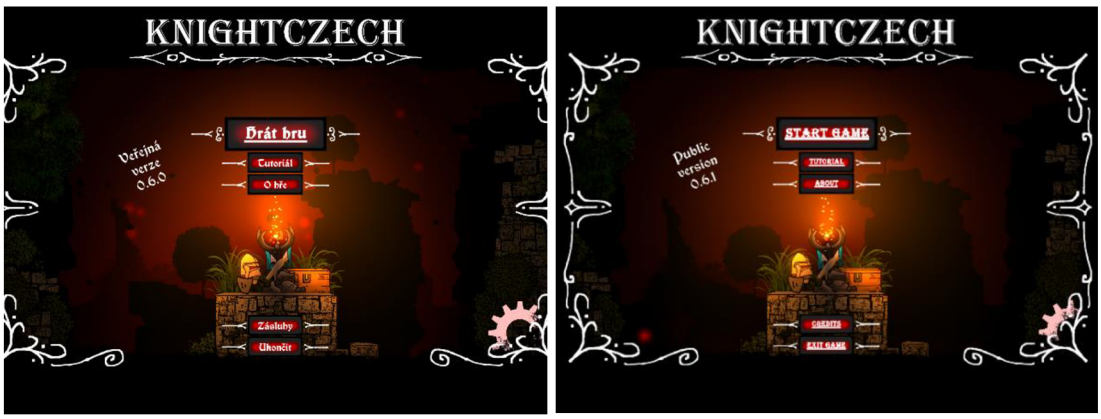
```
if(_changeResolution == true)
{
     currentResolutionIndex = _targetedResolutionIndex;
     Screen.SetResolution(resolutions[currentResolutionIndex].width,
     resolutions[currentResolutionIndex].height,Screen.fullScreen,60);
}
else
{
     resolutionDropdown.value = currentResolutionIndex;
     resolutionDropdown.RefreshShownValue();
}
```

*Ukázka kódu - skript SettingsUI; přepínání rozlišení; 2. část*

Místo této části funkce lze samozřejmě implementovat jakékoliv jiné chování v případě, že by zastupovalo více vyhovující řešení. Jednou z alternativ k (aktuálnímu) řešení uvedenému v této ukázce může být například povolení provedení změny rozlišení na nepodporované rozlišení za doprovodu vyskakovacího okna upozorňujícího uživatele na možné nedostatky a komplikace a zobrazující případnou žádost o potvrzení provedení dané akce.

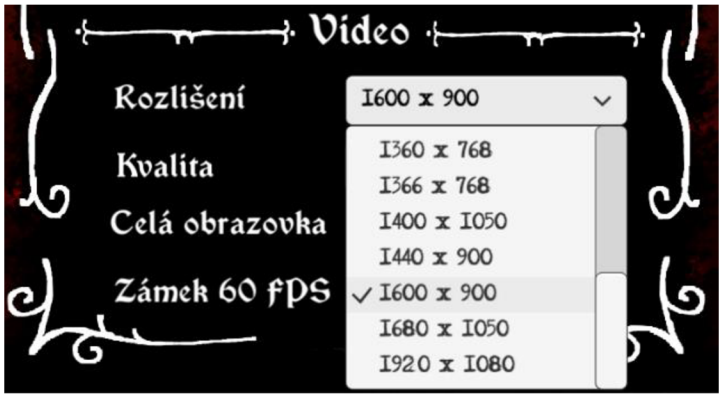
### 4.3.3 Výsledek optimalizace

Pomocí provedených úprav v editoru Unity bylo docíleno kvalitnějšího, méně chybového zobrazování objektů uživatelského rozhraní v nabídce hry. Jako příklad je v této práci uvedeno srovnání staré a nové implementace zachycující ukotvení a zobrazení orámování v hlavní nabídce hry při poměru stran okna 4:3, viz obrázek 3. Vlevo se nachází podoba před úpravou, vpravo podoba po úpravě.




*Obrázek 3 - porovnání změn uchycení objektů uživatelského rozhraní při poměru stran 4:3 (zdroj: vlastní)*

Po revizi a provedení úprav ve skriptu pro manipulaci uživatelských nastavení je nyní možné si zobrazit seznam všech dostupných podporovaných (i nepodporovaných) rozlišení a libovolně mezi nimi přepínat pomocí výběru v rozbalovací nabídce rozlišení, která je součástí uživatelského rozhraní, viz obrázek 4.




*Obrázek 4 - náhled na (nové) možnosti nastavení rozlišení v uživatelském rozhraní (zdroj: vlastní)*

Tímto byl počáteční cíl této optimalizace z pohledu autora splněn. Zdrojový kód byl sepsán v souladu s návrhovými vzory, respektive v souladu se vzorem Šablonová metoda, díky čemuž lze tuto novou implementaci v budoucnu snáze upravovat.

## 4.4 Optimalizace 2: skripty reprezentující úložný prostor hráče

Skript, který zajišťoval manipulaci a ovládání úložného prostoru hráče skrze objekty uživatelského rozhraní, obsahoval špatně strukturovaný kód, zejména v podobě téměř identických metod, které plnily stejné instrukce s rozdílnými komponenty a proměnnými. Proměnné tohoto skriptu byly pevně přiřazené ke konkrétním metodám, ačkoliv zde byla možná jiná implementace daných povelů, které metody prováděly, což v konkrétním případě značně snižovalo čitelnost a efektivnost daného skriptu.

### 4.4.1 Návrh na optimalizaci

Struktura skriptu byla vyhodnocena k celkovému předělání s podmínkou zachování původní výsledné funkcionality. Cílem této optimalizace není implementace nových funkcionalit nýbrž přepsání skriptu do čitelnější a efektivnější podoby, která by více vyhovovala aktuálním standardům programování v jazyce C#.

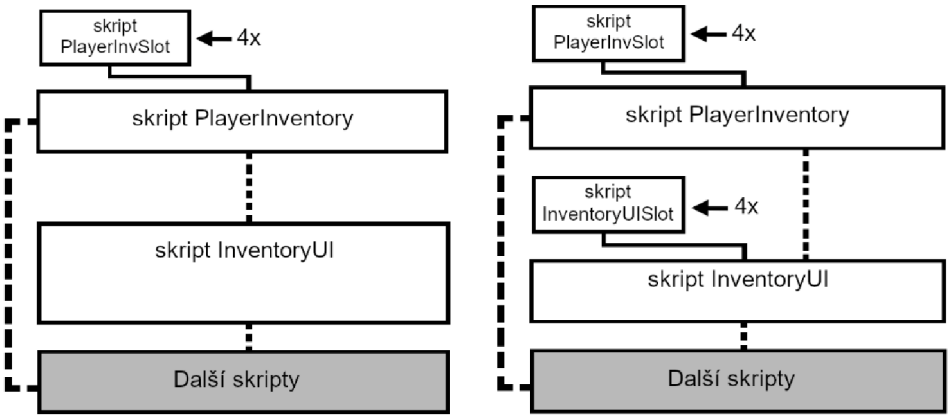
### 4.4.2 Postup implementace optimalizace

Skript „InventoryUI“

Skript obsahoval jednotlivé proměnné a komponenty pro vizuální reprezentaci celého úložného prostoru hráče. V konkrétním případě, kdy se úložný prostor hráče skládal ze čtyř míst a pro propojení jednoho místa úložného prostoru s uživatelským rozhraním bylo potřeba sedm proměnných, skript obsahoval dvacet osm proměnných pouze pro vizuální zpracování, identifikaci a manipulaci s konkrétním místem úložného prostoru.

Byl zde proto aplikován návrhový vzor Kompozit (Composite), který lze použít k vytvoření strukturované hierarchie za doprovodu zachování vzoru Šablonová metoda (Template method) [3].

Proměnné byly přesunuty do nově vytvořeného skriptu, který představuje množinu proměnných a komponentů (přiřazených referencí) potřebných pro vizuální zpracování a manipulaci jednoho místa úložného prostoru. Místo 28 proměnných stačí v původním skriptu přiřadit čtyři instance nového skriptu, viz obrázek 5.




*Obrázek 5 - znázornění provedení úpravy přesunutí vybraných proměnných ze skriptu InventoryUI do nového skriptu InventoryUISlot (zdroj: vlastní)*

```
public class InventoryUISlot
{
     public GameObject slotImageHolder; public Sprite slotImage;
     public Text slotDescription; public Text slotCounter;
     public Text slotTimerText; public int slotTimeCounter;
     public bool slotCanBeUsed;
}
```

*Ukázka kódu – (nově vytvořený) skript InventoryUISlot obsahující proměnné pro identifikaci a manipulaci s 1 místem úložného prostoru*

Metody plnící stejné instrukce s rozdílnými a pevně přiřazenými komponenty či proměnnými byly doplněny novými a více univerzálními metodami, kterým pro plnění daných instrukcí stačí specifikovat konkrétní proměnné či komponenty ve formě vstupních hodnot.

Původní metody byly zachovány kvůli propojení skriptu s uživatelským rozhraním hry – každá z metod byla přiřazena konkrétnímu objektu (tlačítku) uživatelského rozhraní úložného prostoru. Cílem přepsání skriptu reprezentující úložný prostor hráče nebylo tuto výslednou implementaci měnit, tudíž zůstala zachována. Z původních metod byly původní instrukce vyjmuty a následně nahrazeny příkazem zavolání jedné z univerzálních metod.

```
private void RefreshSlot(PlayerInvSlot playerInvSlot, InventoryUISlot
invUISlot)
{
     invUISlot.slotImage = playerInvSlot.image;
     invUISlot.slotCounter.text = playerInvSlot.amount + "";
```

```
     if(itemDescriptionList != null)
     {
          invUISlot.slotDescription.text = itemDescriptionList
          .GetDescription (playerInvSlot.name, "long");
     }
     else
     {
          invUISlot.slotDescription.text =
          playerInvSlot.abilityDescription;
     }
     invUISlot.slotImageHolder.GetComponent<UnityEngine.UI.Image>().
     sprite = invUISlot.slotImage;
}
public void RefreshSlot1()
     { RefreshSlot(playerInventory.slot1, inventoryUISlot1); }
public void RefreshSlot2()
     { RefreshSlot(playerInventory.slot2, inventoryUISlot2); }
public void RefreshSlot3()
     { RefreshSlot(playerInventory.slot3, inventoryUISlot3); }
```

*Ukázka kódu - skript InventoryUI – příklad univerzální metody pro načtení hodnot ze skriptů úložného prostoru hráče do skriptů uživatelského rozhraní úložného prostoru hráče a příklad jejího volání v původních metodách*

Tyto úpravy byly provedeny u všech důležitých operací, které se ve skriptu nacházely. Jednalo se o načtení a vymazání hodnot z úložného prostoru či použití specifického místa úložného prostoru.

### 4.4.3 Výsledek optimalizace 2

Skript reprezentující úložný prostor hráče disponuje novou strukturou kódu. Funkcionalitu metod, které byly dříve navázané na konkrétní komponenty a proměnné, nyní vykonávají metody, které na konkrétní komponenty a proměnné přímo navázané nejsou.

U některých proměnných byla zvolena vhodnější pojmenování, která lépe vystihují účel daných proměnných. Nově sepsaný kód byl doplněn o komentáře vysvětlující jeho funkce a využití (tyto komentáře byly z ukázek kódu vyjmuty).

Výsledná funkcionalita skriptu a propojení skriptu s uživatelským rozhraním zůstaly téměř beze změn.

## 4.5 Optimalizace 3: popisy interaktivních předmětů

V příběhovém režimu hry je dostupná sada interaktivních předmětů, které hráč může najít nebo koupit, uložit do svého úložného prostoru a následně je například použít nebo je z úložného prostoru odebrat.

Tyto předměty jsou ve hře kromě náhledového obrázku hráči reprezentovány i popisem ve formě textového řetězce, který se typicky zobrazuje pod náhledovým obrázkem v daném uživatelském rozhraní.

V každé instanci skriptu definující interaktivní předmět se nacházely dvě proměnné typu textového řetězce, do kterých mohly být v editoru vývojového prostředí vepsány textové hodnoty (popisky) vystihující účel či statistiky daného předmětu.

První zmíněná proměnná typu textového řetězce sloužila pro zapsání standardního (dlouhého) popisu interaktivního předmětu, který byl použit například pro zobrazení popisu předmětu v úložném prostoru hráče. Druhá zmíněná proměnná sloužila pro zapsání zkráceného (krátkého) popisu interaktivního předmětu, který byl zobrazen například v nabídce obchodníka.

Popisy interaktivních předmětů byly dostupné pouze v anglickém jazyce, ačkoliv hra v dané době disponovala kromě anglického jazyka i jazykem českým. Popisy interaktivních předmětů se proto zobrazovaly v anglickém jazyce (nepřeložené) i v případě, že byl v nastavení zvolen český jazyk.

Pokud by byla česká varianta popisů interaktivních předmětů přidána do současné struktury tohoto subsystému projektu pouhým vytvořením dvou dodatečných proměnných ve skriptu reprezentující interaktivní předmět, nebylo by možné nově vytvořené popisy v českém jazyce efektivně napojit na stávající strukturu implementace bez nutnosti vyřešení několika souvisejících problémů.

### 4.5.1 Návrh na optimalizaci

Stávající implementace zobrazování textového popisu interaktivních předmětů by měla být upravena tak, aby se popisy interaktivních předmětů zobrazovaly v českém jazyce, pokud je český jazyk zvolen v nastavení hry.

Výsledná úprava implementace by měla mít strukturu, do které by mělo být možné v budoucnu přidat další jazykové možnosti efektivnějším a snadnějším způsobem, než jakým by byly nové jazykové možnosti přidány ve stávající, neupravené implementaci.

### 4.5.2 Postup implementace optimalizace

Jak již bylo zmíněno výše, stávající implementace distribuce dat mezi jednotlivými skripty, které se podílejí na fungování interaktivních předmětů a jejich zobrazování, se v případě přidávání dalších proměnných pro popis předmětu v dalších jazycích jeví jako implementace dlouhodobě neudržitelná a neefektivní. To bylo pravděpodobně způsobeno tím, že se při tvorbě této části projektu nepočítalo s budoucím rozšiřováním či úpravami, které by zasahovaly do funkční struktury této části projektu.

Pokud by byla realizována varianta, kdy se do skriptu interaktivních předmětů přidají další dvě proměnné typu textového řetězce pro překlad zobrazovaných informací do českého jazyka, vyskytlo by se pravděpodobně několik níže popsaných problémů.

Prvním problémem by byla distribuce dat mezi jednotlivými skripty. Stará struktura projektu by nedovolovala přenést mezi skripty více než jednu jazykovou variantu popisu předmětu, což by způsobilo sekundární problémy například při ukládání dat. Pro alespoň částečné vyřešení tohoto problému by bylo potřeba upravit způsob distribuce a načítání dat v několika různých skriptech.

Další problém by představovalo načítání textových řetězců do uživatelského rozhraní ve správném jazyce. Do každého skriptu, který s těmito textovými řetězci pracuje, by bylo potřeba připsat kód, pomocí kterého by se u každého načítání kontrolovalo, jaký textový řetězec má být použit na základě právě zvoleného jazyka.

Bylo proto rozhodnuto, že se textové popisy předmětů oddělí do samostatného skriptu, který bude následně napojen na zbytek projektu tak, aby byly výše popsané problémy efektivně vyřešeny.

Nově vytvořený skript obsahuje ve formě textových řetězců veškeré popisy ve všech aktuálně podporovaných jazycích. Tyto textové řetězce jsou uspořádány v několika polích, které představují jednotlivé skupiny, respektive sady textových řetězců. Rozdělení textových řetězců do skupin bylo provedeno na základě jazyka, pro který jsou sepsané, a jejich způsobu využití.

V případě, že jsou v projektu rozlišeny dva druhy popisů interaktivních předmětů (krátké a dlouhé) a projekt disponuje dvěma dostupnými jazyky (anglickým a českým), jsou textové řetězce rozděleny do čtyř polí.

```
//LONG description in ENGLISH
private string[] longDescription_en = new string[15]{
      "Empty slot",                    //0 - "empty" item
      "Ammo for canon",                //1 - canon ammunition item
      "Restores energy when used",     //2 - energy potion item
```

*Ukázka kódu – skript ItemDescriptionList; příklad skupin (polí) textových řetězců, část 1*

```
//SHORT description in ENGLISH
private string[] shortDescription_en = new string[15]{
      "Empty",            //0
      "Ammunition",       //1
      "Restores energy",  //2
```

*Ukázka kódu - skript ItemDescriptionList; příklad skupin (polí) textových řetězců, část 2*

Ve skriptu se dále nachází metoda, která v případě vhodně zadaných vstupních argumentů navrátí hodnoty konkrétního textového řetězce. Při volání dané metody je nutné zadání dvou argumentů, pomocí kterých je následně vhodný textový řetězec vybrán. První argument představuje název předmětu, pro který má být obsah jednoho z textových řetězců navrácen, druhým argumentem je specifikováno, o jaký typ popisu se jedná.

Vzhledem k nízkému počtu dostupných interaktivních předmětů v projektu je výběr vhodného textového řetězce na základě shody jména se zadaným parametrem realizován

pomocí přepínače (switch), kdy při jeho použití nedochází ke znatelným prodlevám v odezvě.

V případě nalezené shody je pomocí dodatečných podmínek dotazujících se na požadovaný typ popisu a právě zvolený jazyk hry vybrán konkrétní textový řetězec.

```
public string GetDescription
(string nameOfItem, string typeOfDescription){
     string descriptionValue = "";
     switch (nameOfItem)
     {
          case "canonball":
          if(typeOfDescription == "short")
          {
               if(language.english_language == true){
                    descriptionValue = shortDescription_en[1];}
               else if(language.czech_language == true){
                    descriptionValue = shortDescription_cz[1];}
          }
          else if(typeOfDescription == "long")
          {
               if(language.english_language == true){
                    descriptionValue = longDescription_en[1];}
               else if(language.czech_language == true){
                    descriptionValue = longDescription_cz[1];}
          }
     break;
```

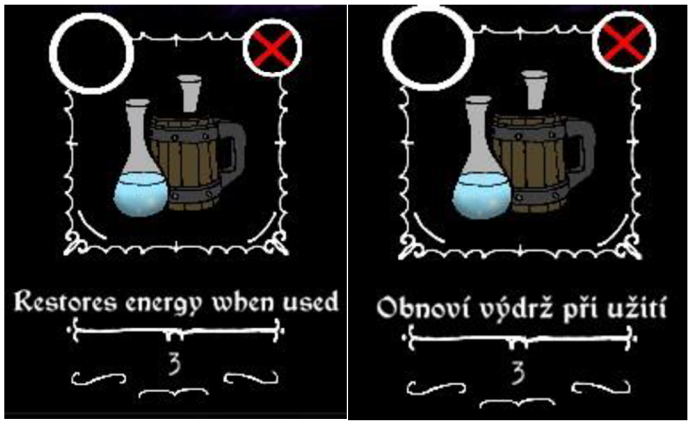
*Ukázka kódu - skript ItemDescriptionList; ukázka části metody pro výběr popisu interaktivního předmětu*

V ostatních skriptech, které s interaktivními předměty a jejich textovými popisy pracují, bylo nutné upravit načítání hodnot textových řetězců, které se v této nové implementaci nerealizuje z konkrétní instance skriptu představující interaktivní předmět, nýbrž z nově vytvořeného, výše popsaného skriptu – pomocí metody „GetDescription“.

Mezi skripty, ve kterých tato změna musela být provedena, se řadí skripty definující obchodníka s interaktivními předměty, skripty sloužící k reprezentaci úložného prostoru hráče a skripty reprezentující truhly s kořistí (interaktivními předměty). Příklad úpravy skriptu je k nalezení v ukázce kódu znázorňující obnovení (zobrazení) dat z úložného prostoru hráče v předešlé optimalizaci (Optimalizace 2, metoda „RefreshSlot“).

### 4.5.3 Výsledek optimalizace 3

V případě, že je v nastavení zvolen český jazyk, se popisy interaktivních předmětů v úložném prostoru hráče, truhlách s kořistí či nabídce u obchodníka zobrazují s popisy v českém jazyce, nikoliv v anglickém, jak tomu bylo před předěláním této části projektu (viz obrázek 6).




*Obrázek 6 - příklad nově zobrazovaného popisu interaktivního předmětu (zdroj: vlastní)*

Veškeré popisy interaktivních předmětů se nyní nachází ve formě textových řetězců v jednom centrálním skriptu, ke kterému mohou v průběhu hry přistupovat ostatní skripty a získávat tak hodnoty zadané v daných textových řetězcích. Jednalo se o změnu většího rozsahu, kdy nově přidaný zdrojový kód musel být patřičně začleněn do starého kódu tak, aby vhodně nahradil starou implementaci zobrazování popisu předmětů. Kontrola a úprava propojení nového kódu se starým kódem musela být provedena napříč několika různými skripty, které byly při plánování této změny zařazeny do sad odhadovaného dopadu a propagačních subjektů.

Nově sepsaný skript s popisy interaktivních předmětů se dá v budoucnu efektivně rozšiřovat o nové popisy předmětů či jejich jazykové varianty. Při budoucí manipulaci, přepisování a rozšiřování popisů interaktivních předmětů nově stačí akci provést pouze v tomto novém skriptu, není potřeba akci provádět na několika různých místech a není potřeba upravovat či kontrolovat několik různých skriptů.

## 4.6 Optimalizace 4: skript sloužící pro ukládání a načítání dat

Data, která jsou v průběhu hraní ukládána do souboru, jsou určena pouze pro opětovné načítání dosaženého postupu či nastavení hry – tedy pro uchování stavu hry mezi jednotlivými relacemi. Nepředpokládá se, že by měla s těmito daty probíhat i jiná manipulace (např. ruční přepisování, či zápis do databáze) a tudíž není vyžadována taková forma dat, která by byla bez pomocných nástrojů čitelná pro lidi (.json, .xml).

Po celou předešlou dobu vývoje hry byla naopak upřednostňována taková forma zápisu dat, při které se data v souborech jevila na první pohled jako nečitelná, což do určité míry stěžovalo manipulaci s těmito daty mimo hru a chránilo tak data před úmyslným pozměněním jejich hodnot (podváděním, urychlením postupu).

Herní data proto byla doposud do souboru ukládána pomocí třídy „BinaryFormatter“. Jedná se o třídu dostupnou v jazyce C#, která slouží k serializaci a deserializaci objektů v binárním formátu [26].

Původní implementace metod ukládání a načítání dat obsahovala nevhodně strukturované části kódu. Skript původně obsahoval pouze dvě metody – jednu pro ukládání dat a jednu pro jejich opětovné načítání.

Tyto metody byly navržené pro jednotné ukládání a načítání dat při stěžejních specifických událostech, jakými byly například spuštění a ukončení hry (programu). Kvůli postupnému rozšiřování funkcí hry se tento způsob ukládání a načítání dat začal projevovat jako nedostatečný a problematický.

Při spuštění hry docházelo k načítání dat, která v danou chvíli nebyla potřebná. Při nestandardním ukončení hry pak naopak nemuselo dojít k uložení dat z poslední relace hry (programu). Při průběžném ukládání dat se operace prováděla se všemi daty, což vedlo k tomu, že se častokrát ukládaly celé bloky dat, ve kterých nedošlo k žádným změnám. Tato implementace byla z výše uvedených důvodů vyhodnocena jako nevhodná.

Další problematický prvek je pak samotný „BinaryFormatter“, který byl dle oficiální dokumentace jazyka C# označen jako nebezpečný a bylo vydáno doporučení

„BinaryFormatter“ nadále nepoužívat, jelikož jeho samotná konstrukce nedovoluje nalezené bezpečnostní chyby opravit [26].

Bezpečnostní problém se týká metody „Deserialize“, ve které bylo nalezeno několik bezpečnostních nedostatků, které se týkají především špatné ověřitelnosti autenticity a charakteru zpracovávaných dat, což ve výsledku může vést k tomu, že se touto cestou mohou do ekosystému programu dostat nežádoucí data či kusy spustitelného kódu, který může být škodlivý [27].

Nalezené bezpečnostní problémy nelze potlačit ani pomocí třídy „SerializationBinder“ či jiných pomocných nastavení, která upravují způsob načítání dat [27].

### 4.6.1 Návrh na optimalizaci

Události pro ukládání a načítání dat rozšířit či nahradit jinými, častějšími událostmi. Novými událostmi mohou být například: otevření či zavření specifické části herní nabídky, změny v konkrétních proměnných či jiné, uživatelem vyvolané akce.

Intuitivně rozdělit ukládaná a načítaná data do skupin. Metody ve skriptu pro ukládání a načítání dat rozdělit do menších metod, které budou ukládat či načítat pouze konkrétní části dat dle vytvořených skupin.

Ukládání a načítání dat nerealizovat přes „BinaryFormatter“. Místo něj najít a implementovat jiný vhodný způsob a formát pro ukládání dat.

### 4.6.2 Postup implementace optimalizace

Skupiny a přiřazené události

Pro eliminaci výše popsaných problémů s načítáním a ukládáním dat byla data rozdělena na základě různorodých uplatnění do skupin, kterým byly přidělené různorodé události pro ukládání a načítání.

Skupina 1 – Data příběhového režimu hry

Zde se nachází veškerá data spojená s příběhovým režimem hry. Lze zde nalézt data obsahující poslední uloženou pozici hráče, název scény (úrovně) ve které se hráč nacházel,

informace o již aktivovaných předmětech a místech v dané úrovni, počet průběžných úmrtí (nezdařených pokusů) hráče, maximální a aktuální úrovně zdraví a výdrže, počet nashromážděných mincí, obsah hráčova batohu a informace o aktuálním stavu hráčových schopností.

Tato data jsou nově načítána při stisknutí tlačítka „Načíst hru“ v sekci výběru herního režimu v hlavní nabídce hry. K „manuálnímu“ uložení těchto dat dojde v případě, že uživatel opustí příběhový režim stisknutím tlačítka „Uložit a odejít“ v herní nabídce při pozastavení hry. K „automatickému“ uložení těchto dat dojde v případě, že hráč dokončí některou z kapitol příběhového režimu.

Pozn: Příběhový režim obsahuje čtyři kapitoly, každá z kapitol obsahuje přibližně 6 až 15 úrovní.

Skupina 2;3;4 – Data první; druhé; třetí lokace klasického režimu

Tyto skupiny obsahují informace o milnících jednotlivých úrovní klasického režimu – ukládá se stav jednotlivých milníků (dokončen/ nedokončen). Každá lokace obsahuje deset úrovní. Každá z úrovní má k dispozici tři milníky, které hráč musí splnit pro úplné dokončení dané úrovně.

Tato data jsou načítána při spuštění hry, aby na základě jejich hodnot mohly být nastavené objekty a komponenty konkrétní scény bez prodlení v okamžiku, kdy se přiřazená scéna zobrazí. Načítání dat až v okamžiku zobrazení přiřazené scény by mohlo vyvolat uživatelem postřehnutelnou prodlevu mezi spuštěním scény a správným zobrazením objektů a jejich komponentů, což by mohlo působit neprofesionálním dojmem.

K ukládání těchto dat dochází při dokončení kterékoliv z dostupných úrovní. Spouštěčem je v tomto případě kliknutí na tlačítko „Pokračovat“, které je součástí části uživatelského rozhraní, které se při dokončení úrovně zobrazí.

Skupina 5 – Data exkluzivních úrovní klasického režimu

Zde jsou obsažena data týkající se exkluzivních úrovní, která jsou sice součástí klasického režimu, nicméně nejsou přiřazena k žádné konkrétní lokaci. Ukládá se pouze informace o

tom, zda uživatel úroveň dokončil nebo nikoliv – je zde tedy pouze jeden milník pro každou úroveň.

Tato data jsou načítána při spuštění hry za stejným účelem jako u skupin 2, 3 a 4 – tedy z důvodu co nejrychlejšího načtení objektů a komponentů specifické scény. Uložení těchto dat je vyvoláno při dokončení úrovně, kdy hráč překročí předem definované kolizní hranice objektu, který definuje konec dané úrovně.

Skupina 6 – Data obecného nastavení programu

Tato skupina obsahuje data ohledně obecného nastavení hry a jejího chování během používání. Do této skupiny spadá především nastavení hlasitosti hudby a zvukových efektů, rozlišení a velikost okna hry, nastavení kvality zobrazovaných objektů, jazyk uživatelského rozhraní a dodatečná nastavení pro prvky uživatelského rozhraní během hraní.

Data této skupiny jsou načítána při spuštění hry. Jsou téměř bez prodlení vyžadována příslušnými skripty nastavení a uživatelského rozhraní, které k nim přistupují s krátkou prodlevou (v rozmezí 0.1 až 0.75 sekund) po spuštění hry.

K ukládání těchto dat dochází při několika různých událostech. Většinou se jedná o události vyvolané změnou hodnot proměnných reprezentujících nastavení. Taková událost může být vyvolána výběrem jedné z hodnot v rozbalovací liště nabídky, případně změnou hodnoty zaškrtávacího pole.

Zbytek událostí je pak přiřazen ke kliknutí na tlačítka, která slouží k zavření specifických částí uživatelského rozhraní reprezentující nastavení hry, kde k průběžnému ukládání při změně hodnot proměnných nedochází. Jedná se například o část uživatelského rozhraní, kde se nastavuje úroveň hlasitosti hudby a zvukových efektů – ukládání hodnot do souboru při každé změně hodnot na posuvníku, ke které může docházet i několikrát za sekundu, by bylo zbytečně výpočetně náročné.

Skupina 7 – Data nastaveného ovládání (mapování kláves)

Do této skupiny spadají data reprezentující uložené nastavení kláves pro ovládání hry. Data této skupiny jsou od dat předešlé skupiny (skupiny 6) oddělena nikoliv z důvodu

různorodosti (pořád se jedná o data reprezentující nastavení hry), ale z důvodu odlišného způsobu přenastavení a ukládání změn v těchto datech.

K ukládání těchto dat dochází ihned po změně jejich hodnot, konkrétně při přenastavení vstupu (klávesy či tlačítka) u kterékoliv z dostupných akcí ovládání.

Bylo by zbytečné, aby se při každém pokusu o přemapování kláves pro jednu z dostupných akcí ovládání hry ukládala a přepisovala i zbylá data reprezentující ostatní nastavení, která s daty nastavení ovládání hry nemají kromě podobnosti typu využití (skupiny) dat nic společného. Z toho důvodu se data této skupiny zapisují (načítají) do (z) odděleného souboru při aktivaci jiných událostí.

K načítání dat dochází ihned po spuštění hry.

Nahrazení třídy „BinaryFormatter“

Pro způsob ukládání dat přes „BinaryFormatter“ se zde nabízelo několik možných alternativ. Žádná z alternativ neměla značně ovlivňovat dosavadní proces ukládání dat. Mělo se jednat pouze o zvolení jiného serializátoru bez nutnosti významně zasahovat do struktury a typů ukládaných dat.

Obdoby bezpečnostních rizik nalezených ve třídě „BinaryFormatter“ lze nalézt i v dalších serializátorech, mezi které lze zařadit například i „SoapFormatter“ [27]. Ačkoliv byl původně „SoapFormatter“ vybrán jakožto možná alternativa, použitím této třídy by nebylo docíleno žádných změn z pohledu zabezpečení práce s daty.

Při zohlednění konkrétního způsobu manipulace s daty, který je ve hře implementován, jsou dle oficiální dokumentace jazyka C# preferované a bezpečné alternativy pro „BinaryFormatter“ především „XmlSerializer“ pro XML, „BinaryReader – BinaryWriter“ pro XML a JSON, případně ještě „System.TextJson“ API výhradně pro JSON formáty [27].

Zde jde pak už jen o subjektivní preferenci toho, který formát dat se zdá pro ukládání a načítání vhodnější. XML i JSON jsou pro lidi čitelné a lze je bez větších obtíží ručně upravovat. Ať je tedy zvolena kterákoliv varianta, zlepšení zabezpečení načítání dat s sebou

v tomto konkrétním případě nese i lepší čitelnost uložených dat, čímž zaniká předešlá preference těžší čitelnosti uložených dat.

Přihlédne-li se k faktu, že se v tomto případě jedná pouze o data malé počítačové hry pro jednoho hráče, která nedisponuje žádnou nadstandardní ochranou kódu či generovaných dat, je zbytečné implementovat další bezpečnostní opatření, která by stěžovala úpravu dat mimo hru, např. šifrování XML pomocí AES (šifrování XML elementů pomocí symetrických klíčů).

Po seznámení se s vhodnými dostupnými alternativami bylo na základě subjektivní preference rozhodnuto použít pro nahrazení třídy „BinaryFormatter“ třídu „XmlSerializer“. Jedná se o třídu určenou pro serializaci a deserializaci objektů do a z dokumentů ve formátu XML. „XmlSerializer“ tedy umožňuje řídit, jak jsou objekty do formátu XML kódovány [28].

Níže je uveden příklad přepsané metody pro uložení a načtení dat reprezentující hráčův postup příběhovým režimem hry, tedy data ze skupiny 1, která je popsaná výše.

V tomto konkrétním případě vyžaduje metoda vstup v podobě čtyř různorodých skriptů, které obsahují data, která se mají uložit. Nejprve se definuje cílová adresa umístění a název souboru pro ukládání. Následně je vytvořena instance skriptu „PlayerData“, který obsahuje proměnné pro vybraná data ze vstupních skriptů. Poté jsou tyto proměnné ve skriptu „PlayerData“ naplněny pomocí metody uvnitř skriptu.

Následně dojde k zapsání dat z instance skriptu „PlayerData“ do souboru přes třídu „XmlSerializer“.

```
public static void SavePlayerDataToFile
(GameMaster gamemaster, PlayerStats playerstats,  PlayerInventory
playerinventory, LoadEntityLevelManager loadentitylevelmanager)
{
     String path = Application.persistentDataPath +
     _storymode_data_path;
     PlayerData dataToStore =
     new PlayerData (gamemaster, playerstats, playerinventory,
     loadentitylevelmanager);
```

```
        using(Stream stream = File.Open(path, FileMode.Create))
        {
            XmlSerializer serializer =
            new XmlSerializer(typeof(PlayerData));

            XmlWriter writer = new XmlTextWriter(stream, Encoding.UTF8);
            serializer.Serialize(writer, dataToStore);
            writer.Close();
        }
}
```

*Ukázka kódu – skript SaveToFile; příklad serializace dat skupiny 1*

Při načítání dat je opět nejprve definována adresa umístění společně s názvem souboru. Pokud je cesta k souboru validní a soubor s daným názvem existuje, proběhne skrz „XmlSerializer“ načtení dat do instance skriptu „PlayerData“, který je po dokončení načítání dat vrácen jako výstup původního volání z jiného skriptu. Pokud není cesta k souboru validní nebo soubor není vytvořen, k načítání dat nedojde a v konzoli je vypsána chybová hláška.

```
public static PlayerData LoadStoredData()
{
    string path =
    Application.persistentDataPath + _storymode_data_path;
    if(File.Exists(path))
    {
        using(Stream stream = File.Open(path, FileMode.Open))
        {
            XmlSerializer serializer =
            new XmlSerializer(typeof(PlayerData));
            PlayerData loadedStoredData =
            (PlayerData)serializer.Deserialize(stream);
            stream.Close();
            return loadedStoredData;
        }
    }
    else
    {
        Debug.Log("Story mode data could not be loaded -
        either file not exists or path is not valid");
        return null;
    }
}
```

*Ukázka kódu – skript SaveToFile; příklad deserializace dat skupiny 1*

Obdobně byly vytvořeny metody pro serializaci a deserializaci dat i u zbylých 6 definovaných skupin. Každá ze skupin se pak kromě samotných dat liší i definovaným názvem souboru a vyžadovanými skripty, ze kterých jsou při ukládání data čerpána.

### 4.6.3 Výsledek optimalizace 4

Po analýze původní implementace ukládání dat byly na základě získaných informací definovány a do projektu začleněny nové události pro ukládání dat do souborů. Tyto události zajišťují průběžné ukládání různých skupin dat v častějších intervalech na základě uživatelských interakcí se hrou.

Tímto krokem byla téměř eliminována náchylnost na ztrátu dat z dané relace, která nebyla uložena například kvůli neočekávaným komplikacím či nesprávnému ukončení hry.

Společně s implementováním nových událostí pro ukládání dat byla ukládaná data roztříděna do skupin dle jejich významu, způsobu a potřeby využití tak, aby se daly nově vytvořené události pro ukládání dat efektivně využít. Při změně hodnot obecného nastavení hry se tak například ukládá pouze skupina dat, která reprezentuje právě obecné nastavení hry.

Třída „BinaryFormatter“, která byla v projektu využívána pro ukládání dat do souboru a jejich opětovné načítání ze souboru byla nahrazena její alternativou – třídou „XMLSerializer“. Data již nejsou do souboru zapisována ve formě binárního kódu, který nelze předem spolehlivě identifikovat. Data jsou do souboru ukládána ve formátu XML, který v tomto případě představuje bezpečnější variantu.

## 4.7 Nová implementace 1: Nastavení ovládání

Ve hře se nachází řada akcí, pomocí kterých hráč ovládá svou postavu či interaguje s dostupným uživatelským rozhraním. K těmto akcím byly původně pevně přiřazené specifické klávesy či tlačítka myši a nebylo možné daným akcím vstupy přenastavit.

S postupným rozšiřováním obsahu a přidáváním nových dostupných akcí se pak mohlo rozložení pevně přiřazených kláves začít jevit jako příliš individuální, založené na základě subjektivních preferencí, se kterými se hráči nemuseli ztotožňovat a mohlo tak docházet ke zhoršení spokojenosti koncového uživatele při hraní hry z důvodu absence možnosti dané ovládání akcí přenastavit.

V předchozích verzích vývojového prostředí Unity bylo možné zahrnout do finálního sestavení programu (hry) dialogové okno, které se uživateli zobrazilo před spuštěním samotného programu. V tomto okně bylo možné nastavit například cílený monitor, rozlišení okna a především vstupy pro ovládání, pokud byl projekt správně napojen na třídy a funkce z kategorie „System.Input“ [29]. Toto dialogové okno bylo z verzí vývojového prostředí Unity 2019 a novějších variant odstraněno a nelze ho tedy ve verzi 2019.3, která je tímto projektem využívána, nadále používat [30]. Náhrada za tuto funkci ve verzi Unity 2019.3 neexistuje.

### 4.7.1 Návrh implementace

I v případě, že by k odstranění daného dialogového okna v novějších verzích vývojového prostředí Unity nedošlo a dané dialogové okno by bylo použito i v tomto projektu, problém s absencí přenastavení ovládání by nebyl pravděpodobně zcela vyřešen. Změny v nastavení ovládání by bylo možné provést vždy pouze před spuštěním hry, což by výše popisovaný problém řešilo jen částečně, vzhledem k tomu, že hráč by musel před každou změnou nastavení ovládání hru vypnout a opět zapnout.

Z daného důvodu bylo navrženo implementovat možnost přenastavení uživatelských vstupů ovládání, kterou by mohl uživatel využít přímo při běhu hry, nikoliv pouze před jejím spuštěním. Provedené změny v nastavení ovládání by měly zůstat zachovány mezi jednotlivými relacemi. Tato nová implementace by měla využívat třídy a funkce „System.Input“, které vývojové prostředí Unity poskytuje. Jednalo by se tedy o vytvoření

nového subsystému skriptů fungujícího na základě dostupných zdrojů poskytnutých vývojovým prostředím.

### 4.7.2 Realizace implementace

Implementace nastavení ovládání (libovolné přenastavení kláves pro klíčové akce) byla do hry přidána v podobě pěti nových skriptů. Společně s přidáním nových skriptů bylo pro začlenění těchto skriptů a správné fungování hry nutné provést úpravy několika již existujících skriptů a v editoru vývojového prostředí navrhnout a sestavit novou množinu objektů představující dodatečnou sekci uživatelského prostředí nastavení hry. Jednoznačně se tak jednalo o změnu velkého rozsahu, která byla na začátku předpokládána.

Dva nové skripty, které obsahují většinu důležité logiky pro změnu nastavení ovládání, jsou níže detailněji popsané včetně ukázek kódu. Tři zbylé nové skripty a řada již existujících skriptů, ve kterých musely být provedeny úpravy, jsou z důvodu doporučeného rozsahu práce popsány pouze velmi stručně. Pro popis a ukázku byly vybrané pouze úpravy, které nevykazují známky duplicity či značné podobnosti s ostatními úpravami vybranými pro popis a ukázku.

Většina provedených úprav v již existujících skriptech se týká změn ve vyvolání specifických akcí ovládání, pro které je v nové implementaci použita stejná metoda (případně velmi podobné metody) jako ve staré implementaci. Jedná se tedy pouze o drobné úpravy, které bylo nutné provést pro úplné a funkční začlenění nového zdrojového kódu.

Skript „PlayerInputManager“

Jedná se o první ze dvou důležitých skriptů, které slouží k realizaci nastavení ovládání hry. Tento skript obsahuje slovník pro dostupné akce a jejich hodnoty (klávesové zkratky). Hodnoty slovníku lze pomocí externího povelu (zavolání metody) měnit při běhu hry a lze tak docílit změn v nastavení ovládání.

V tomto skriptu se zároveň nachází dva výchozí seznamy hodnot. První seznam obsahuje všechny dostupné a hrou podporované akce. Druhý seznam se skládá z klávesových zkratek. Tyto seznamy představují výchozí nastavení ovládání. Pořadí zapsaných hodnot určuje, jak jsou jednotlivé hodnoty v seznamech párovány (první hodnota ze seznamu číslo jedna má

být párována s první hodnotou ze seznamu číslo dva, atd.). V případě implementace nové akce ovládání je potřeba do těchto seznamů připsat nové hodnoty.

```
        string[] keyBindings = new string[13]
        {
        "MoveRight",    //movement on x axis to right (positive)
        "MoveLeft",     //movement on x axis to left (negative)
        "Jump",         //impulse movement on y axis (positive)
        "ClimbUp",      //movement on y axis (positive)
        "ClimbDown",    //movement on y axis (negative)
        "Inventory",    //show - hide inventory UI
        "Abilities",    //show - hide abilities UI
        "UseInvSlot1",  //use first slot of inventory
        "UseInvSlot2",  //use second slot of inventory
        "UseInvSlot3",  //use third slot of inventory
        "UseInvSlot4",  //use fourth slot of inventory
        "Attack",       //proceed to attack
        "Confirm"       //confirm - continue - select action
        };
        KeyCode[] defaultKeys = new KeyCode[13]
        {
        KeyCode.D, KeyCode.A, KeyCode.Space, KeyCode.W, KeyCode.S,
        KeyCode.E, KeyCode.Q, KeyCode.Alpha1, KeyCode.Alpha2,
        KeyCode.Alpha3, KeyCode.Alpha4, KeyCode.Mouse0, KeyCode.Return
        };
```

*Ukázka kódu – skript PlayerInputManager; seznam akcí (seznam číslo 1) a výchozích hodnot (seznam číslo 2)*

Při inicializaci skriptu je zavolána metoda pro načtení akcí a hodnot do slovníku. Následně je s krátkým opožděním zavolána metoda pro zachycení aktuálního stavu slovníku. Pokud neexistuje kopie slovníku z předchozí relace v podobě uložených hráčských dat, nachází se ve slovníku výchozí akce a hodnoty z dostupných seznamů, tudíž nedošlo k přepisu výchozích dat. Pokud kopie slovníku existuje, byly do slovníku během úvodní prodlevy mezi inicializací skriptu a zavoláním metody pro ověření stavu slovníku vepsány dříve uložené hodnoty, které se pravděpodobně liší od těch výchozích.

```
void Awake() { SetDefaultKeybidings(); }
void Start() { CheckKeybindingsWithDelay(0.25f); }

public IEnumerator CheckKeybindingsWithDelay(float delay)
{
     yield return new WaitForSeconds(delay);
     CheckKeybindings();
}
```

```
public void CheckKeybindings(){
     if(recievedDataFromFile == false)
     {
          Debug.Log("PlayerInputManager: Did not recieved
          data from file, default keybindings persists");
     }
     else
     {
          Debug.Log("PlayerInputManager: Data from file
          recieved, default keybindings were overridden");
     }
}
private void SetDefaultKeybidings()
{
     keyBindingDictionary = new Dictionary<string, KeyCode>();
     for(int i = 0; i < keyBindings.Length; i++)
     {
          keyBindingDictionary.Add(keyBindings[i], defaultKeys[i]);
     }
}
```

*Ukázka kódu - skript PlayerInputManager; metody vyvolané při inicializaci skriptu*

Pro změnu hodnoty (přemapování) určité akce obsahuje skript metodu, jejíž zavolání je podmíněno dvěma parametry – textovým řetězcem představujícím označení žádané akce pro kterou má být změna provedena a označení konkrétního vstupu, na který má být žádaná akce přemapována.

Pokud slovník žádanou akci obsahuje, je provedena změna její hodnoty ze stávající na nově požadovanou. Pakliže se žádaná akce ve slovníku nenachází, není provedena změna, jelikož ji není kde provést a jedná se pravděpodobně o nežádoucí chybu v konfiguraci specifických hodnot v editoru nebo souboru, ze kterého byla načítána uložená data, či jiném behaviorálním skriptu, který tuto metodu zavolal.

```
public void SetKeyBinding(string bindingAction, KeyCode key)
{
     if(keyBindingDictionary.ContainsKey(bindingAction) == false)
     {
          Debug.Log("action " + bindingAction +
          " could not be found in dictionary 'keyBindingDictionary'");
     }
```

```
        else
        {
            keyBindingDictionary[bindingAction] = key;
            Debug.Log("action " + bindingAction + " was binded to " +
            key.ToString());
         }
}
```

*Ukázka kódu - skript PlayerInputManager - metoda SetKeyBinding*

Kromě přenastavení hodnoty konkrétní akce na základě vstupních hodnot skript obsahuje metodu pro získání hodnoty určité akce v podobě textového řetězce. Tato metoda je v případě potřeby volána vně tohoto skriptu jinými skripty, které danou hodnotu vyžadují pro různorodé účely – například pro zobrazení aktuálně používaných hodnot v uživatelském rozhraní.

```
public string GetKeyCodeStringValue(string nameOfAction)
{
    KeyCode outputKeyCode = keyBindingDictionary[nameOfAction];
    string keyCodeText = "" + outputKeyCode.ToString();
    return keyCodeText;
}
```

*Ukázka kódu – skript PlayerInputManager - metoda GetKeyCodeStringValue*

Jednou z hlavních funkcí skriptu je také přemostění komunikace mezi jednotlivými skripty provádějícími konkrétní akce se systémem vstupu vývojového prostředí Unity – jedná se tedy o prostředníka pro propojení hráčem vyvolaných vstupů ovládání s odezvou hry. Pro tento účel jsou ve skriptu připraveny tři metody – kvůli třem různým druhům uživatelského vstupu, viz ukázka níže. Každá z metod vrací logickou hodnotu „true“ nebo „false“ na základě poskytnutého vstupu a aktuálního stavu systému vstupů.

První metoda využívá funkci „Input.GetKey“. Metoda „Input.GetKey“ vrací logickou hodnotu pro specifickou klávesu - pokud je v daný moment klávesa držena ve stlačené poloze, vrací metoda hodnotu „true“, v jakémkoliv jiném případě vrací hodnotu „false“ [31]. Nezáleží na délce intervalu držení klávesy, důležité je, že systém snímá klávesu jako „právě používanou“ (stlačenou). Tato metoda je využívána například pro realizaci horizontálního a vertikálního pohybu postavy ovládané hráčem – chůze vpravo či vlevo, skok do výšky atp.

V případě chůze či skoku postavy se metoda volá opakovaně z jiného skriptu za cílem rozhodnutí, zda má postava hráče nadále vykonávat zvolený pohyb či nikoliv. Hráč pak může například pomocí změn intervalu držení klávesy ovlivnit výšku skoku či vzdálenost, kterou postava ujde na jedno stisknutí klávesy.

```
public bool KeyPressing(string bindingAction){
      return Input.GetKey(keyBindingDictionary[bindingAction]);}
```

*Ukázka kódu - skript PlayerInputManager; metoda KeyPressing, využito Input.GetKey*

Druhá metoda využívá funkci „Input.GetKeyDown“. Funkce „Input.GetKeyDown“ vrací logickou hodnotu „true“ pouze pokud je v daném „snímku“ (tiku metody „Update“) klávesa stisknuta – nikoliv držena [32]. Stisk a držení klávesy či jiný alternativní vstup vývojové prostředí Unity registruje jako dvě rozdílné události. Poté, co metoda zachytí daný druh vstupu a navrátí tak při dotazu na stav logickou hodnotu „true“, musí být pro další navrácení logické hodnoty „true“ nejprve klávesa uvolněna [32].

Pokud tedy uživatel stiskne a podrží specifickou klávesu, bude pro danou klávesu při opakovaném volání této metody navrácena logická hodnota „true“ pouze při prvotním zachycení stisknutí klávesy. Následně bude vždy navrácena logická hodnota „false“ až do té doby, dokud nedojde k uvolnění dané klávesy a jejímu opětovnému stisknutí.

```
public bool KeyIsDown(string bindingAction){
      return Input.GetKeyDown(keyBindingDictionary[bindingAction]);}
```

*Ukázka kódu – skript PlayerInputManager; metoda KeyIsDown, využito Input.GetKeyDown*

Třetí metoda využívá funkci „Input.GetKeyUp“. Ta slouží k navrácení logické hodnoty pro událost uvolnění specifické klávesy [33]. Stejně jako u stisku a držení klávesy se i zde jedná o další odlišný druh události, kterou vývojové prostředí Unity zaznamenává. Logická hodnota „true“ je pak navrácena pouze při zachycení uvolnění klávesy, tedy při změně stavu z držení klávesy do stavu uvolněné (nepoužívané) klávesy [33]. Ve všech ostatních případech vrací metoda logickou hodnotu „false“.

```
public bool KeyIsReleased(string bindingAction){
      return Input.GetKeyUp(keyBindingDictionary[bindingAction]);}
```

*Ukázka kódu – skript PlayerInputManager; metoda KeyIsReleased, využito Input.GetKeyUp*

Skript „KeybindingsUI“

V tomto skriptu se nachází metody zajišťující propojení uživatelského rozhraní, které se nachází v nově vytvořené sekci nabídky nastavení, s funkcemi ostatních skriptů, zejména pak s funkcemi výše popsaného skriptu „PlayerInputManager“. Jedná se tedy o skript, který koriguje a vyvolává změny hodnot uživatelského nastavení ovládání skrze dostupné uživatelské rozhraní a přidružené skripty.

Při inicializaci skriptu je definována reference na skript „PlayerInputManager“ popsaný výše a zavolána metoda pro načtení dostupných (vývojovým prostředím podporovaných) uživatelských vstupů.

```
void Start()
{
     _playerInputManager = GameObject.FindWithTag("God").
     GetComponent<PlayerInputManager>();
     KeyCodeInitializer();
}
```

*Ukázka kódu – skript KeybindingsUI; metoda Start, inicializace základních nastavení potřebných proměnných*

Vývojové prostředí Unity disponuje přibližně 400 různými označeními pro vstupy ovládání. V těchto označeních jsou kromě vstupů klávesnice a myši zahrnuty i vstupy ovladačů herních konzolí, joysticky a případně další specifická zařízení.

Tento projekt počítačové hry je určený převážně pro stolní počítače a přenosné laptopy s operačními systémy Windows, Linux, případně MacOS. Zařízení s jinými vstupy ovládání či jiným operačním systémem nejsou podporována, tudíž by neměly být podporované ani jejich vstupy ovládání. Z tohoto důvodu jsou při načítání dostupných vstupů ovládání uplatněna specifická kritéria popsaná níže.

Vývojové prostředí Unity má v dokumentaci, která je dostupná online na jejich webových stránkách, uvedený seznam všech podporovaných vstupů a jejich interní označení [34].

Na základě tohoto seznamu podporovaných vstupů jsou při načítání daných vstupů uplatněna kritéria, která omezují dané vstupy pouze na vstupy klávesnice a myši. Kritéria splňují pouze ty vstupy, které se v seznamu nachází nad vstupem (položkou) „JoystickButton0“, tedy

veškeré vstupy (položky) od „None“ až po „Mouse6“, který označuje poslední vstup z kategorie klávesnice a myši.

Dostupné vstupy, které splňují definovaná kritéria, jsou přidány do pole určeného k jejich dočasnému uložení a pozdějším operacím. Výsledkem této operace je podmnožina vybraných vstupů, která je oproštěna od hrou nepodporovaných a tím pádem nevyužívaných vstupů ovládání. Je tak docíleno menšího počtu prvků v poli a tedy i snížení potřebného výpočetního výkonu při následujících operacích s daným polem – například při iteraci skrz dané pole se nemusí operace provádět u cca 400 prvků, nýbrž jen u cca 150.

Po provedení načtení podporovaných vstupů ovládání je skript připraven k realizaci změn nastavení ovládání na základě událostí vyvolaných uživatelem.

```
private void KeyCodeInitializer()
{
     keyCodes =
     System.Enum.GetValues(typeof(KeyCode)).Cast<KeyCode>().Where
     (key => ((int)key <= (int)KeyCode.Mouse6)).ToArray();
     _keyCodesInitialized = true;
}
```

*Ukázka kódu – skript KeybindingsUI; metoda KeyCodeInitializer, inicializace označení kláves*

Proces změny nastavení ovládání, respektive proces provedení změn ve skriptu „PlayerInputManager“, je realizován postupem popsaným níže.

Pokud je vyvolána událost zahajující změnu nastavení ovládání (přemapování specifické akce na jiný vstup), je zavolána metoda, kde je proměnná typu „bool“ (sloužící jako přepínač) přepnuta do opačného stavu, než ve kterém se dosud nacházela, což umožní další průběh procesu změny nastavení ovládání.

Zahajující událostí je v tomto případě stisknutí tlačítka (interakce s objektem), čímž je nepřímo zavolána zde popisovaná metoda v tomto skriptu. Proces změny nastavení ovládání je vždy zahajován z externího skriptu jiného objektu.

Pokud byla metoda zavolána při zahájení procesu změny nastavení ovládání, je zároveň s přepnutím stavu proměnné aktivován přiřazený objekt uživatelského rozhraní – v tomto

případě se jedná o vyskakovací okno, které uživateli sděluje postup při dalším kroku, kterým je zvolení nového vstupu.

```
public void InputListenerSwitch()
{
     if(_shoudListenToInput == false)
     {
          _shoudListenToInput = true;
          enableObjectByCall.ButtonClick_enable();
     }
     else
     { _shoudListenToInput = false; }
}
```

*Ukázka kódu – skript KeybindingsUI; metoda InputListenerSwitch*

Společně s metodou pro přepnutí stavu přepínače je typicky zavolána i metoda, která přiřazuje do připravených proměnných informace o tom, pro který komponent byla metoda zavolána a specifikuje název akce, pro kterou má být změna nastavení ovládání provedena. I tato metoda je volána z externích zdrojů (jiných skriptů).

```
public void LoadNameOfAction(string nameOfAction, Text buttonText )
{
     _nameOfActionToSwitch = nameOfAction;
     _buttonTextToReturn = buttonText;
}
```

*Ukázka kódu – skript KeybindingsUI; metoda LoadNameOfAction*

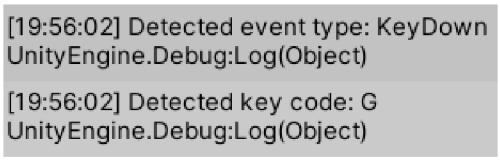
Po přepnutí stavu proměnné a načtení názvu akce je skript připraven na zachycení dalšího uživatelského vstupu, který by měl dle očekávání reprezentovat nově zvolený vstup pro vybranou akci.

Vývojové prostředí Unity disponuje rozhraním, které při zavolání vrátí označení konkrétního uživatelského vstupu (události), který je právě používán (byl právě použit). Jedná se o rozhraní „Event“ ve spojení s voláním „MonoBehaviour.OnGUI“ metody, která byla stručně popsána v teoretické části práce. Pomocí níže uvedeného příkladu kódu lze zachytit konkrétní typ události vyvolané uživatelem a zjistit označení vyvolané události – v tomto případě označení uživatelského vstupu ve formě stisknutí klávesy.

```
public class InputDetectTest : MonoBehaviour
{
     void OnGUI()
     {
          Event myEvent = Event.current;
          if(myEvent.isKey && myEvent.type == EventType.KeyDown
          && myEvent.keyCode != KeyCode.None)
          {
               Debug.Log("Detected event type: " + myEvent.type);
               Debug.Log("Detected key code: " + myEvent.keyCode);
          }
     }
}
```

*Ukázka kódu – skript InputDetectTest; metoda OnGUI, test zachycení vstupu z podporované klávesy či tlačítka*

Při krátkém stisknutí klávesy takovýto skript zanechá v konzoli výstup zobrazený níže na obrázku 7.




*Obrázek 7 - výpis zachycených událostí a jejich typu při aplikování limitujících podmínek přiloženým kódem (zdroj: vlastní)*

Pro získání jednoho konkrétního označení při stisknutí klávesy bylo potřeba pevně specifikovat, jaké parametry a hodnoty u požadované události očekávat. V tomto konkrétním případě bylo specifikováno následující:

- Událost musí být vyvolána klávesnicí (uživatelský vstup musí pocházet z klávesnice)
- Typ dané události musí odpovídat typu označující stisknutí klávesy
- Označení události nesmí nabývat hodnot definující „žádnou“ klávesu (KeyCode.None)

Bez aplikace těchto podmínek by při krátkém stisknutí klávesy bylo zachyceno a do konzole sepsáno několik desítek až stovek různých (uživatelských) vstupů, jak je uvedeno na obrázku 8.


*Obrázek 8 - Výpis zachycených událostí v metodě OnGUI bez aplikovaných podmínek (zdroj: vlastní)*

Mezi snímkem vlevo a snímkem vpravo bylo vynecháno přibližně 100 řádků zachycujících označení a typ jiných událostí, které proběhly mezi stisknutím a uvolněním tlačítka (opakovaně události typu "Layout", "repaint", atp.).

Příčinou tohoto chování je především to, že metoda „OnGUI“ je volána několikrát během jednoho tiku obnovovací metody (během jednoho průchodu metody „Update“). Dále je nutno vzít v potaz fakt, že metoda „OnGUI“ je využívána nejen pro zachycení uživatelského vstupu, ale i pro zachycení jakékoliv jiné události, například vykreslování prvků uživatelského rozhraní atp.

V průběhu testování a implementace možnosti změny nastavení ovládání bylo rozhodnuto tuto variantu zjišťování označení použitých vstupů neuplatnit a využít místo toho jiných prostředků.

Zachycení uživatelského vstupu je proto v tomto případě realizováno přes rozhraní „System.Input“, ze kterého vychází i zbytek této implementace.

Pokud je v metodě „Update“ skriptu „KeybindingsUI“ zachycen jakýkoliv vstup (pravděpodobně z klávesnice či tlačítek myši), je opět zavolána metoda pro přepnutí přepínače, aby nedošlo k zachycení dalších, v tomto případě nežádoucích, vstupů. Následně je zavolána metoda pro provedení další části procesu.

```
void Update()
{
     if(_shoudListenToInput == true)
     {
          if(Input.anyKey == true)
          {
               Debug.Log("Update (KeybindingsUI): input detected");
               InputListenerSwitch();
               KeyCodeChanger();
          }
     }
}
```

*Ukázka kódu – skript KeybindingsUI; metoda Update, volání metod při zachycení vstupu od uživatele*

V tento moment se celý proces nachází ve stavu, kdy změna nastavení ovládání probíhá, nicméně zatím se ví pouze to, že byl detekován neurčitý, blíže nespecifikovaný vstup neznámého označení.

Před realizací změny nastavení ovládání pro specifickou akci je tedy potřeba zjistit, který konkrétní vstup byl uživatelem „aktivován“, aby mohlo dojít k přiřazení tohoto vstupu k vybrané akci. Pro tento účel bylo krátce po inicializaci skriptu provedeno výše popsané načtení dostupných označení vstupů, jejichž užší výběr na základě určitých kritérií byl uložen do pole.

Skript se stále nachází v jednom určitém zavolání („tiku“) metody „Update“, kde byl zatím nespecifikovaný vstup zachycen. Daný nespecifikovaný vstup je v tento časový úsek stále vnímán jako aktivní (při dotazu na jeho stav je navrácena logická hodnota „true“) a lze zároveň provést dotaz na stav konkrétních vstupů pomocí jejich označení. Pomocí provedení takového dotazu nad různými vstupy lze určit, o který konkrétní vstup se jedná.

Dotaz na stav konkrétního vstupu je proveden nad každým prvkem pole, kde jsou označení podporovaných vstupů uložena. Pokud je nalezen prvek pole, jehož stav odpovídá hodnotě „true“, je aktivní a dříve neznámý vstup identifikován a jeho označení je vráceno pro pokračování procesu.

Pakliže u žádného z prvků pole není aktuální stav vyhodnocen jako aktivní, jedná se pravděpodobně o vstup, který je vývojovým prostředím podporován, nicméně není podporován projektem – hrou. V takovém případě je navráceno označení zastupující „žádnou“ klávesu – „Keycode.None“.

```
private KeyCode KeyCodeCheck()
{
     if(_keyCodesInitialized == true)
     {
          for (int i = 0; i < keyCodes.Length; i++)
          {
               if(Input.GetKey(keyCodes[i]) == true)
               {
                    return keyCodes[i];
               }
          }
     }
     return KeyCode.None;
}
```

*Ukázka kódu – skript KeybindingsUI; metoda KeyCodeCheck, vyhledávání aktivního vstupu*

Na základě výše popsaného pokusu o identifikaci konkrétního aktivovaného vstupu se pokračuje v procesu přenastavení vstupu ovládání.

Pokud se nalezené označení aktivního vstupu shoduje s označením „žádné“ klávesy („KeyCode.None“) nebo se shoduje s označením klávesy typicky určené pro zrušení probíhající akce („KeyCode.Escape“), není provedena žádná změna v nastavení ovládání a uživatelské rozhraní se vrátí do původního stavu – objekt představující vyskakovací okno s instrukcemi pro uživatele je skryt.

Jestliže se nalezené označení neshoduje s ani jedním z výše zmíněných označení pro přerušení operace, je zavolána metoda ve skriptu „PlayerInputManager“ pro provedení změny ve slovníku, kde jsou akce a jim přiřazené označení vstupů uložené. Následně je aktualizován textový komponent objektu (tlačítka), přes který byla změna zahájena. Poté je slovník obsahující novou hodnotu uložen do specifikovaného souboru v místním úložišti. Posledním krokem změny nastavení ovládání je i v tomto případě navrácení uživatelského rozhraní do původního stavu – deaktivace vyskakovacího okna.

```
private void KeyCodeChanger()
{
     KeyCode pressedKey = KeyCodeCheck();
     if(pressedKey != KeyCode.None &&
        pressedKey != KeyCodeCheck.Escape)
     {
          _playerInputManager.SetKeyBinding (_nameOfActionToSwitch,
          pressedKey);
          _buttonTextToReturn.text = "" + pressedKey.ToString();
          SaveKeybindindsToFile();
     }
     enableObjectByCall.ButtonClick_disable(true);
}
```

*Ukázka kódu – skript KeybindingsUI; metoda KeyCodeChanger, část procesu přenastavení*

```
private void SaveKeybindindsToFile()
{
     SaveToFile.SaveKeybindings(_playerInputManager);
     Debug.Log("SaveKeybindindsToFile: save method called");
}
```

*Ukázka kódu – skript KeybindingsUI; metoda SaveKeybindingsToFile pro uložení slovníku ze skriptu PlayerInputMnager a jeho hodnot do souboru*

Ve skriptu se taktéž nachází metoda pro přepsání hodnoty konkrétního textového komponentu na základě definovaného vstupu. Tato metoda je výhradně volána z jiných skriptů. Slouží především k aktualizaci textových objektů zobrazovaných v uživatelském rozhraní (například pro aktualizaci objektů po inicializaci základních hodnot slovníku či načtení uložených hodnot slovníku ze souboru).

```
public void UpdateButtonText(string nameOfAction, Text buttonText )
{
     string keyCodeText =
     _playerInputManager.GetKeyCodeStringValue(nameOfAction);
     buttonText.text = "" + keyCodeText;
}
```

*Ukázka kódu – skript KeybindingsUI; metoda UpdateButtonText*

Tímto byl nově implementovaný princip přenastavení a uložení hodnot ovládání popsán od začátku do konce. Následující text v této podkapitole je věnován dalším skriptům, které byly v průběhu implementace tohoto chování sepsány, nicméně z pohledu stanoveného cíle plní pouze podpůrné a méně důležité povely.

Skript „KeybindingsUIButtonText“

Jedná se o skript obsahující metody, které slouží například k zavolání metod pro načtení či aktualizování textových komponentů objektů, ke kterým je tento skript přiřazen. Jinými slovy se jedná z pohledu výše popisovaného skriptu „KeybindingsUI“ o skript, který je jedním z externích zdrojů, odkud jsou některé metody skriptu „KeybindingsUI“ volány.

Po inicializaci skriptu je provedeno přepsání hodnoty textového komponentu přiřazeného objektu na základě uložených hodnot ve slovníku skriptu „PlayerInputManager“ a názvu akce, který byl objektu přiřazen v editoru. Skript tak lze použít např. pro úvodní aktualizaci vizuální odezvy pro uživatele v nabídce nastavení ovládání.

V editoru lze taktéž nastavit hodnoty veřejných proměnných, na základě kterých je určeno, jestli má být před provedením operace použito zpoždění či o jaký druh textového komponentu se v daném konkrétním případě jedná.

Skript „PlayerDataKeybindings“

V tomto případě se jedná o skript, jehož základem není třída „MonoBehaviour“. Třída „MonoBehaviour“ je jednou ze základních tříd, které se ve skriptech pro Unity prostředí používají. Pokud ve skriptu není použita, nelze využívat například následující funkce specifické pro Unity: Start(), Update(), FixedUpdate(), LateUpdate(), OnGUI(), OnDisable(), OnEnable() a další [35].

Skript slouží pouze jako třída obsahující proměnné, do kterých jsou načteny hodnoty, které mají být uloženy do souboru. V tomto případě je skript použit při ukládání nastavení ovládání. Ze skriptu „PlayerInputManager“ je do tohoto skriptu načten slovník, jehož hodnoty mají být do souboru uloženy.

Skript „LoadKeybindings“

Tento skript slouží k načtení dat ze souboru a následné distribuci načtených dat do míst, odkud byla data při předešlém ukládání do souboru kopírována.

Nejprve je vytvořena reference na skript, odkud byla data při předešlém ukládání převzata. Následně je do instance skriptu sloužícího pro ukládání a načítání dat ze souboru načten

dříve uložený slovník nesoucí hodnoty představující nastavení ovládání. Poté jsou hodnoty slovníku v cílovém skriptu přepsány přes vytvořenou referenci a proces načtení dat je dokončen.

Úpravy dalších skriptů

Pro začlenění nových skriptů do ekosystému projektu bylo nutné provést úpravy v již sepsaných a dříve začleněných skriptech. Jednalo se především o aktualizaci skriptů, přes které byly realizovány různé akce na základě hráčského vstupu. Skripty plnící takový účel dříve využívaly dostupné zdroje a metody poskytnuté vývojovým prostředím Unity (System.Input etc.) a musely být upraveny tak, aby využívaly metody a zdroje nově dostupného skriptu „PlayerInputManager“.

Jako příklad lze uvést aktualizaci implementace odezvy na uživatelský vstup pro horizontální pohyb postavy. V původní implementaci byla odezva navázána na konkrétní vstup definovaný přímo v daném skriptu. Ověřovací podmínka obsahovala dotaz přímo na „System.Input“ vývojového prostředí Unity.

```
if(Input.GetKeyDown(KeyCode.D))
{ moveInput = 1; }
if(Input.GetKeyDown(KeyCode.A))
{ moveInput = -1; }
if(Input.GetKey(KeyCode.D) == false
&& Input.GetKey(KeyCode.A) == false)
{ moveInput = 0; }
```

*Ukázka kódu - nespecifikovaný skript; stará implementace odezvy na uživ. vstup pro pohyb postavy*

V nové implementaci ověřovací podmínka obsahuje dotaz směřovaný na metodu v nově začleněném a výše popsaném skriptu „PlayerInputManager“. Bylo tím docíleno toho, že provedení dané odezvy není závislé na jednom uživatelském vstupu definovaném přímo ve skriptu vykonávajícím odezvu (není závislé na jednom konkrétním označení klávesy), nýbrž na vstupu, který je uložený ve slovníku a přiřazen názvu činnosti, na základě které lze identifikovat a rozlišovat konkrétní odezvy.

```
if(playerInputManager.KeyIsDown("MoveRight"))
{moveInput = 1; }
if(playerInputManager.KeyIsDown("MoveLeft"))
{moveInput = -1; }
if(playerInputManager.KeyPressing("MoveLeft") == false
&& playerInputManager.KeyPressing("MoveRight") == false)
{ moveInput = 0; }
```

*Ukázka kódu - nespecifikovaný skript; nová implementace odezvy na uživ. vstup pro pohyb postavy*

Podobné úpravy byly provedeny i ve všech ostatních skriptech, které obsahovaly funkce závislé na uživatelském vstupu.

Tvorba v editoru

Pro dokončení začlenění nové implementace nastavení ovládání bylo nutné vytvořit novou část uživatelského rozhraní v hlavní nabídce hry, na kterou byly výše popsané skripty přímo či nepřímo navázány tak, aby bylo docíleno zamýšleného chování nové implementace.

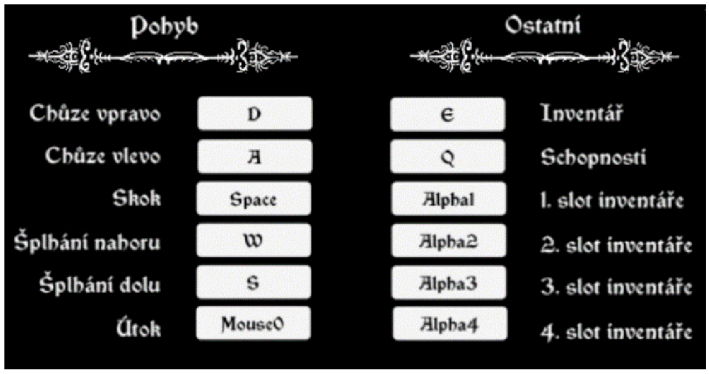
Pro každou ve hře dostupnou akci, u které bylo povoleno provádět změny uživatelského vstupu, bylo potřeba vytvořit objekt či skupinu objektů, pomocí které by bylo možné zobrazit právě přiřazený uživatelský vstup či provést změnu přiřazeného uživatelského vstupu pro danou akci. Každá taková akce je v uživatelském prostředí reprezentována dvěma hlavními objekty – tlačítkem a popisným textem. Objekt představující popisný text zobrazuje název konkrétní akce. Objekt tlačítka obsahuje textový komponent, který zobrazuje právě přiřazený název vstupu. Tlačítko zároveň při jeho sepnutí slouží jakožto spouštěč změny nastavení ovládání.

### 4.7.3 Výsledek

U všech nově přidaných skriptů byly dodrženy postupy pro použití návrhových vzorů popsaných v teoretických východiscích této práce, pokud bylo jejich použití v daném skriptu opodstatněné, a veškerý zdrojový kód byl řádně okomentován.

Do projektu byly zakomponovány nové skripty pro nastavení ovládání, pro které byla zároveň vytvořena nová sekce uživatelského rozhraní, viz obrázek 9. Nová sekce uživatelského rozhraní se nachází v hlavní nabídce hry jakožto jedna z vedlejších nabídek nastavení. Pomocí nových prvků projektu si může hráč zobrazit aktuální vstupy ovládání určitých akcí, kterými hra disponuje. Tyto vstupy lze zároveň i měnit dle libosti na

jakoukoliv dostupnou klávesu či tlačítko myši, které projekt a samotné vývojové prostředí Unity podporují. Případné změny, které jsou v nastavení ovládání provedeny, jsou ukládány do souboru. Nastavení ovládání tak přetrvává mezi jednotlivými relacemi – pokud dojde k provedení změny vstupu u konkrétní akce v nastavení ovládání a proběhne zápis dat do souboru, hodnoty nastavení ovládání jsou při dalším spuštění hry ze souboru opětovně načteny.




*Obrázek 9 - náhled na novou sekci už. rozhraní interpretující aktuální stav nastavení akcí (zdroj: vlastní)*

Změnu vstupu pro konkrétní akci hráč zahájí kliknutím na tlačítko zobrazující aktuálně přiřazený vstup pro danou akci. Následně se spustí proces změny nastavení ovládání popsaný výše a objeví se dialogové okno vybízející uživatele k dalšímu kroku, viz obrázek 10.


*Obrázek 10 - náhled na dialogové okno, které se zobrazí během procesu změny nastavení ovládání (zdroj: vlastní)*

Po stisknutí klávesy či tlačítka myši dialogové okno zmizí a proces změny nastavení ovládání pokračuje. Pokud byl zadán validní vstup splňující definovaná kritéria, je proces úspěšně dokončen. Tlačítko reprezentující vstup pro danou akci, u které proběhla změna, nyní ukazuje textovou hodnotu nově zvoleného vstupu. Pokud poskytnutý vstup nesplňuje požadovaná kritéria, je proces přerušen a ke změně nastavení nedojde. Ke konkrétní akci pak zůstává přiřazena původní hodnota.

# 5 Výsledky a diskuse

Výsledkem práce jsou čtyři úspěšné optimalizace původních funkcí projektu a jedna rozsáhlejší implementace zcela nové funkce. Výsledný produkt v podobě počítačové hry nyní disponuje aplikovanými úpravami v podobě nových či upravených množin objektů a novými či upravenými částmi zdrojového kódu, které se zbytkem projektu tvoří funkční celek zbavený nalezených závažnějších chyb či nedostatků a je rozšířený o novou funkcionalitu.

## 5.1 Výsledky optimalizací

Jak již bylo zmíněno výše, provedení a výsledek všech čtyř zvolených optimalizací odpovídají dříve stanoveným a popsaným požadavkům, které byly formulovány na počátku zpracovávání.

Pomocí první optimalizace (optimalizace uživatelského rozhraní a nastavení rozlišení okna hry) bylo docíleno méně chybového a flexibilnějšího zobrazování grafických prvků, pomocí kterých je koncovému uživateli interpretováno uživatelské rozhraní hlavní nabídky, i při zvolení dříve nepodporovaných rozlišení okna hry (týká se především variant s poměrem stran 4:3 a jejich blízkých alternativ).

V rámci druhé a čtvrté optimalizace (optimalizace skriptu hráčova úložného prostoru a skriptů pro ukládání a načítání dat hry) bylo docíleno úprav zdrojového kódu, které splňují předpoklady pro efektivněji sepsanou a lépe udržitelnou strukturu projektu, na základě které lze projekt v daných částech bez větších komplikací dále rozšiřovat, pokud by to bylo nutné.

Zbývající optimalizace (optimalizace interaktivních předmětů) zajistila jak docílení přehlednější a efektivnější struktury projektu se znovupoužitelností zdrojového kódu, tak i ve výsledku vylepšenou vizuální interpretaci interaktivních objektů koncovému uživateli.

Podrobněji popsané výsledky jsou dostupné jednotlivě na koncích podkapitol v kapitole Vlastní práce.

## 5.2 Výsledky nové implementace

Implementace nové funkcionality přináší možnost nastavení uživatelských vstupů ovládání pro jednotlivé akce a události ve hře, čímž už není koncový uživatel vázán na jednu konkrétní konfiguraci nastavení ovládání, která mu mohla z důvodu dřívější absence dodatečné konfigurace přijít nevhodná.

To bylo docíleno sepsáním vlastního subsystému skriptů, jehož součástí je i skript, který lze označit jako vlastní adaptér, který spojuje část projektu (hry) s funkcemi vývojového prostředí Unity a pomocí kterého lze v budoucnu snadněji a přehledněji implementovat nastavení ovládání pro nové akce či události hry. Předpokládá se, že takový adaptér či jeho části by bylo možné použít i v jiných projektech, které jsou na prostředí Unity postavené.

## 5.3 Možná vylepšení a další rozvoj

V rámci dalších úprav projektu, kterým se tato práce zabývala, by bylo možné dále rozvíjet obsah na aktuální poupravené struktuře projektu – tedy zaměřit se více na obsahovou stránku hry, která nebyla hlavním předmětem této práce. Nové funkcionality by tak pravděpodobně byly směřovány do oblasti přidávání herního obsahu v podobě nových akcí či událostí, nových interaktivních předmětů či objektů herního prostředí a nových herních úrovní, případně překladu textových prvků hry do dalších jazykových variant. Taktéž by se mohlo dále upravovat rozložení objektů uživatelského rozhraní pro vysoce širokoúhlé („ultra-wide“) rozlišení, které nebyly do úprav zobrazení herního okna popisovaných v této práci zahrnuty.

# 6 Závěr

Cílem práce bylo optimalizovat a rozšířit již existující počítačovou hru, která byla vytvořena pomocí prostředí Unity. Úpravy a tvorba nových částí zdrojového kódu společně s úpravami provedenými ve vývojovém prostředí tento cíl naplnily v plném rozsahu včetně dílčích cílů.

V rámci teoretických východisek je čtenář seznámen se základy problematiky různých softwarových odvětví, které byly součástí vypracování vlastní práce (praktické části). Jednalo se především o přiblížení podstaty a způsobu provedení analýzy, která byla na projekt aplikována. Dále bylo čtenáři nastíněno používání návrhových vzorů v objektově orientovaném jazyce C# a představení základních funkcí vývojového prostředí Unity, ve kterém byl projekt zpracováván.

Během vypracovávání řešení vlastní práce byly k dosažení různých dílčích cílů použity různé metody, které vycházely z poznatků nabytých při vypracovávání části práce zahrnující teoretická východiska. Práce se zabývala především funkční stránkou projektu, kdy hlavním předmětem zájmu byla ve dvou případech úprava struktury zdrojového kódu, v dalších dvou případech úprava již existujících funkcí z hlediska zdrojového kódu i jejich finální vizuální interpretace koncovému uživateli. Na závěr byla úspěšně implementována pro projekt zcela nová, rozsáhlá funkcionalita, kterou lze bez větších komplikací aplikovat i v jiných, podobných projektech. Konkrétní postup řešení problémů a provedení implementace byly popsány v jednotlivých kapitolách takovým způsobem, aby rozsah popisu odpovídal rozsahu a náročnosti daných dílčích úkolů.

Vývojové prostředí Unity se ve spojení s jazykem C# a odbornými teoretickými poznatky jeví jako dobré vývojové prostředí či nástroj pro tvorbu a úpravu počítačových her, což bylo demonstrováno vyřešením dříve nalezených chyb či problémů a úspěšným aplikováním změn.

# 7 Seznam použitých zdrojů


1. RAJLICH, Vaclav. *Software Engineering : The Current Practice*. CRC Press, Elektronická verze - N/A (Tištěná verze roku 2011), strany 31 - 46. ISBN 9781466510357.
2. GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 0201633612.
3. BISHOP, Judith. *C# návrhové vzory*. Zoner Press, 2010, strany 65 - 90. ISBN 978-80-7413-076-2.
4. BISHOP, Judith. *C# návrhové vzory*. Zoner Press, 2010, strany 133 - 140. ISBN 978-80-7413-076-2.
5. BISHOP, Judith. *C# návrhové vzory*. Zoner Press, 2010, strany 169 - 179. ISBN 978-80-7413-076-2.
6. BISHOP, Judith. *C# návrhové vzory*. Zoner Press, 2010, strany 180 - 185. ISBN 978-80-7413-076-2.
7. RAJLICH, Vaclav. *Software Engineering : The Current Practice*. CRC Press, Elektronická verze - N/A (Tištěná verze roku 2011), strana 106. ISBN 9781466510357.
8. RAJLICH, Vaclav. *Software Engineering : The Current Practice*. CRC Press, Elektronická verze - N/A (Tištěná verze roku 2011), strana 115. ISBN 9781466510357.
9. RAJLICH, Vaclav. *Software Engineering : The Current Practice*. CRC Press, Elektronická verze - N/A (Tištěná verze roku 2011), strany 116 - 117. ISBN 9781466510357.
10. RAJLICH, Vaclav. *Software Engineering : The Current Practice*. CRC Press, Elektronická verze - N/A (Tištěná verze roku 2011), strany 125 - 126. ISBN 9781466510357.
11. RAJLICH, Vaclav. *Software Engineering : The Current Practice*. CRC Press, Elektronická verze - N/A (Tištěná verze roku 2011), strany 127 - 132. ISBN 9781466510357.
12. GIMP.org – základní informace. *Gimp.org* [online]. [cit. 2021-12-17]. Dostupné z: https://www.gimp.org/
13. Unity3D - *pořadí spouštění klíčových funkcí* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/Manual/ExecutionOrder.html
14. Unity3D - Awake metoda. *Docs.unity3d.com* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.Awake.html
15. Unity3D - OnEnable metoda. *Docs.unity3d.com* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.OnEnable.html
16. Unity3D - Reset metoda. *Docs.unity3d.com* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.Reset.html

17. Unity3D - Start metoda. *Docs.unity3d.com* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.Start.html
18. Unity3D - FixedUpdate metoda. *Docs.unity3d.com* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.FixedUpdate.html
19. Unity3D - OnTrigger metoda. *Docs.unity3d.com* [online]. [cit. 2021-12-29]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Collider.OnTriggerEnter.html
20. Unity3D - OnCollision, příklad OnCollisionEnter. *Docs.unity3d.com* [online]. [cit. 2022-01-04]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Collider.OnCollisionEnter.html
21. Unity3D – WaitForFixedUpdate. *Docs.unity3d.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/WaitForFixedUpdate.html
22. Unity3D - OnMouseXXX functions. *Docs.unity3d.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/30_search.html?q=OnMouse
23. Unity3D - Update metoda. *Docs.unity3d.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.Update.html
24. Unity3D - LateUpdate metoda. *Docs.unity3d.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.LateUpdate.html
25. Unity3D - Canvas. *Docs.unity3d.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html
26. BinaryFormatter třída. *Docs.microsoft.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.formatters.binary.binaryformatter?view=net-5.0
27. BinaryFormatter zabezpečení. *Docs.microsoft.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide
28. XML serializer. *Docs.microsoft.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer?view=net-6.0
29. Resolution dialog window – legacy docs. *Docs.unity3d.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.unity3d.com/560/Documentation/Manual/class-PlayerSettingsStandalone.html
30. Resolution dialog window removal. *Blog.unity.com* [online]. [cit. 2021-11-15]. Dostupné z: https://blog.unity.com/technology/introducing-unity-2019-1#Engine

31. Unity Input system - GetKey. *Docs.unity3d.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Input.GetKey.html
32. Unity Input system - GetKeyDown. *Docs.unity3d.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Input.GetKeyDown.html
33. Unity Input system - GetKeyUp. *Docs.unity3d.com* [online]. [cit. 2021-11-15]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/Input.GetKeyUp.html
34. Unity3D - KeyCode a celkový přehled značek. *Docs.unity3d.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/KeyCode.html
35. Unity3D - třída MonoBehaviour. *Docs.unity3d.com* [online]. [cit. 2022-01-05]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/ScriptReference/MonoBehaviour.html

# 8 Přílohy

Upravené či nově vytvořené skripty obsahující zdrojový kód projektu jsou dostupné v přiloženém .zip souboru či na poskytnutém paměťovém médiu. Tyto skripty sami o sobě netvoří funkční celek. Výsledný produkt (počítačová hra) není v přílohách zahrnut z důvodu kapacitních omezení úložiště a faktu, že jeho oficiální verze vyžaduje ověření platnosti kopie na příslušné internetové platformě.