Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Implementace počítačové hry s použitím enginu Godot

2021

Martin Popelka

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Martin Popelka
Název práce:	Implementace počítačové hry s použitím enginu Godot
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	35
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Martin Popelka
Title:	Computer game using Godot engine
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	35
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem práce je vytvoření počítačové hry za pomoci herního enginu Godot. Čtenáře seznámím s enginem Godot a procesem tvorby her v něm. Poté představím implementovanou hru a popíšu podstatu herních elementů.

Synopsis

The aim of the work is to create a computer game using the game engine Godot. I will introduce the reader to the Godot engine and the process of creating games in it. Then I will introduce the implemented game and describe the essence of game elements.

Klíčová slova: počítačová hra; Godot; herní engine

Keywords: computer game; Godot; game engine

Děkuji svému vedoucímu Mgr. Petru Osičkovi, Ph.D. za vedení této práce a užitečné připomínky. Děkuji také své rodině a přátelům za nepřetržitou podporu.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

1 Úvod

Videohry patří mezi globálně vyhledávanou formu zábavy. Stejně jako se technologie neustále vylepšují, spolu s nimi se zlepšuje i herní průmysl. Tvorba her je v dnešní době dostupná téměř pro každého a existuje mnoho úspěšných her, za jejichž vznikem stojí pouze jedniná osoba [1, 2].

Herní engine je jedním z nejdůležitějších prvků vývoje her. Jedná se o softwarový framework sloužící k abstrakci nízkoúrovňových programovacích úkonů jako jsou vykreslování, fyzika či rozpoznávání vstupů. Díky tomu se vývojáři mohou soustředit na detaily, které jejich hry činí jedinečnými [3].

1.1 Motivace

Hraní videoher bylo nevyhnutelnou součástí mého dětství. Při pozdějším získání základních zkušeností v programování mě začala fascinovat myšlenka tvorby vlastních her, které by poskytovaly zábavu ostatním lidem jako mně v mládí.

První zkušenosti s tvorbou her jsem získal díky hernímu enginu Game Maker Studio, ve kterém jsem už několik jednoduchých her naprogramoval. Práce v něm mi však dostatečně nevyhovovala. Později jsem se doslechl o rozrůstajícím se enginu Godot a začal se o něj zajímat. Čtení mnoha pozitivních ohlasů a jeho uživatelská přívětivost vzbudily můj zájem. Rozhodl jsem tedy využít příležitosti a věnovat se herní tvorbě s využitím enginu Godot v rámci své bakalářské práce.

Godot mimo jiné nabízí podporu programovacímu jazyku C#, ve kterém jsem se rozhodl hru psát a aplikovat tak studiem získané zkušenosti.

2 Godot Engine

Godot je open source engine pro tvorbu 2D i 3D her. Poskytuje komplexní sadu nástrojů a hry lze exportovat na řadu platforem. Je nezávisle řízený komunitou dobrovolníků pod *MIT*[1] licencí, což znamená, hry uživatelů jsou právoplatně jejich a mohou je šířit bez žádných poplatků. Godot je dostupný zdarma stažením jediného spustitelného souboru, který má navíc ve srovnání s ostatními herními enginy velmi nízkou velikost. Je vhodný jak pro začátečníky, tak pro pokročilejší vývojáře [4].

Godot se od ostatních enginů odlišuje inovativním systémem scén a uzlů. Všechny hry v tomto enginu jsou založeny na tomto systému. Uzly jsou atomické části herních prvků a jejich kombinace tvoří scény, které se mohou dále kombinovat a tvořit tak scény ještě komplexnější [5].

2.1 Uzly

Základní prvky tvorby hry jsou uzly. Uzly mají různé typy, jako např. *sprity*[2], *tilemapy*[3], přehrávače animací, oblasti, zvuky, kamery, světla, časovače aj. Mají vlastní název, nastavitelné vlastnosti, speciální funkce a mohou přijímat zpětná volání [5]. Ve smyslu objektově orientovaného programování uzly představují objekty.

Object je základní třída pro všechny předdefinované typy. Z této třídy dále dědí *Node*, což je základní třída pro uzly. Potomky této třídy jsou např. *AnimationPlayer*, *Timer*, *Viewport* a *CanvasItem*. *CanvasItem* obsahuje vlastnosti související se zobrazením (např. viditelnost, barva, světelná maska). Potomkem této třídy je třída *Node2D*, což je základní třída pro všechny 2D objekty jako např. *Sprite*, *TileMap*, *Camera2D* nebo *CollisionShape2D*. Obsahuje vlastnosti související s polohou a velikostí objektů. Umožňuje s objekty pohybovat, otáčet je a měnit velikost pomocí dostupných metod.

Pro každý uzel je možné vytvořit vlastní třídu pomocí skriptu (viz kapitola 2.2.1). Určením rodičovské třídy se zdědí potřebné vlastnosti a metody odpovídající danému typu uzlu. Ve stromové struktuře uzel dále dědí hodnoty vlastností, jako pozice, velikost, rotace, viditelnost apod. To znamená, že např. rotací konkrétního uzlu se změní rotace všech jeho potomků o stejnou hodnotu.

2.2 Scény

Kombinace uzlů tvoří scénu. Scéna je stromová struktura obsahující kořenový uzel s libovolným množstvím potomků. Scény se v Godot Enginu ukládají ve formátu *.tscn*. Na takto uložené scény je pak možné se odkazovat přes skript, nebo z nich skládat komplexnější scény (např. úrovně hry) tzv. instancováním (viz kapitola 2.2.1).

[1] Zkratka pro Massachusetts Institute of Technology.
[2] Označení pro bitmapovou grafiku.
[3] Mřížka dlaždicových spritů sloužící k reprezentaci herního prostředí.

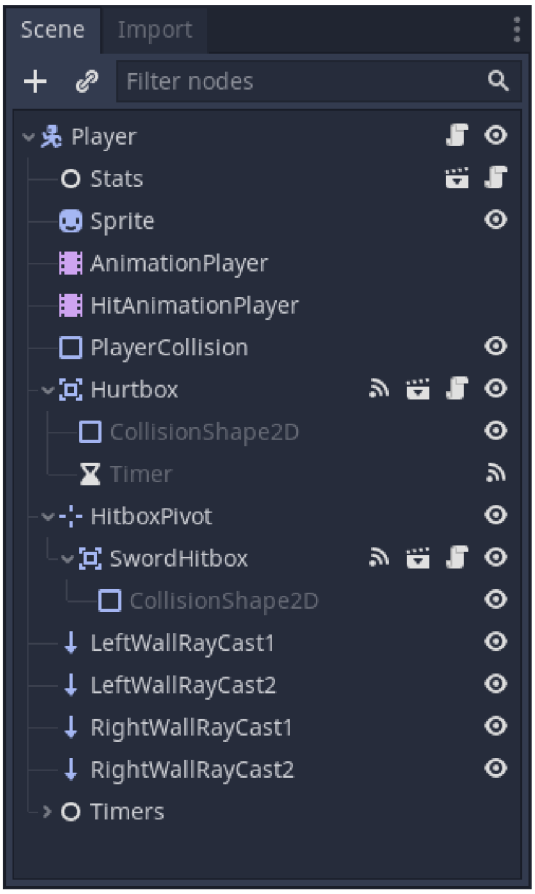
Propojení uzlů ve stromové struktuře doprovází trojice pravidel [5]:

- Uzly jsou zpracovávány v pořadí daném stromovou strukturou od kořenového uzlu k jeho potomkům.
- Uzly jsou vykreslovány v pořadí daném stromovou strukturou. Rodiče jsou na obrazovku vykresleny před potomky. To znamená, že potomkovské uzly překrývají rodiče.
- Uzly dědí transformaci jejich rodičů. Pokud uzel změní pozici, rotaci nebo velikost, tato změna se aplikuje na všechny jeho potomky. Změny potomků jsou relativní ke svým rodičům.

Jakmile uzel vstoupí do stromu scény, stane se aktivním a získá přístup ke všemu, co potřebuje ke zpracování: údaje o uživatelských vstupech, schopnost zobrazovat 2D prvky, přijímaní a odesílání upozornění, přehrávání zvuků atd. Tyto schopnosti uzel ztrácí při odebrání ze scény.

2.2.1 Instancování scén

Při návrhu úrovní hry často používáme více stejných objektů. Při složitějších hrách by kopírování uzlů vedlo k nepřehlednosti a pozdější provádění změn by se mohlo stát velmi časově náročným. Z tohoto důvodu Godot nabízí instancování scén. Pro objekty, jako např. nepřátele, si stačí vytvořit samostatnou scénu a tu potom vkládat (instancovat) do vybraných úrovní. Případné změny v hlavní scéně se poté projeví na všech jejich instancích. Instancovaná scéna se chová jako uzel, tím zajišťuje pohodlnou manipulaci ve stromové struktuře a přehlednou správu projektu.



Obrázek 1: Scéna hlavní postavy hry

2.3 Programovací jazyk

Godot podporuje 4 hlavní programovací jazyky. Jsou to GDScript, VisualScript, C# a nově i C++.

GDScript je hlavní jazyk enginu Godot, navržen speciálně pro něj a plně integrován. Syntaxí se podobá jazyku Python. VisualScript je naopak, z důvodu jeho implementace pomocí „bloků a spojení“ určen především pro uživatele bez zkušenosti s programováním. K použití dalších jazyků včetně C# je zapotřebí vnějšího programovacího rozhraní [6]. GDScript se od jazyka C# liší kromě syntaxe i notací. Zatímco GDScript využívá *snake_case*[4], metody v C# jsou psány notací *PascalCase*[5]. Godot tyto rozdíly v jazycích respektuje a není nutné při psaní kódu v jazyce C# notace střídat.

Jelikož jsem pro účely této bakalářské práce využíval jazyka C#, budu se na jednotlivé metody odkazovat notací právě tohoto jazyka.

[4]Způsob psaní víceslovných názvů, jednotlivá slova jsou oddělena podtržítkem.
[5]Způsob psaní víceslovných názvů, jednotlivá slova začínají velkým písmenem.

2.4 Skriptování

Každý uzel může být obohacen o jeden skript, který dodává požadované chování, ovládá jeho funkčnost a řídí interakce s ostatními uzly. Godot vychází z objektově orientovaného programování, takže potomek dědí vlastnosti ze svého předka [7]. Ve skriptu se vytvoří třída s libovolným názvem a určí se rodičovská třída. Rodičovská třída obvykle odpovídá typu daného uzlu (např. ve skriptu pro hráče je vytvořena třída *Player*, která dědí z třídy *KinematicBody2D*), ale lze vytvořit i vlastní třídy a dědit z nich.

2.4.1 Virtuální metody

Uzly obsahují virtuální metody, které programátor může přepsat. To umožňuje aktualizovat uzel v reakci na konkrétní událost. Ať už ve chvíli, kdy vstoupí do scény, nebo při každém zobrazeném snímku za sekundu.

Virtuální metody jsou od ostatních metod odlišeny podtržítkovým prefixem a mezi ty nejvíce využívané patří `_Ready` a `_Process`, které následně představím.

2.4.1.1 Metoda `_Ready`

Jakmile kořenový uzel vstoupí do scény a aktivuje všechny své potomky, je zavolána metoda `_Ready`. Metoda nevyžaduje argument a lze ji využít k nastavení výchozího stavu daného objektu či skriptu [5].

Pokud je potřeba získat přístup k dalším uzlům, lze použít metodu `GetNode`, která jako argument vyžaduje cestu k požadovanému uzlu, ať už relativní (aktuální stromovou strukturou) nebo absolutní (souborovým systémem).

2.4.1.2 Metoda `_Process`

Hry jsou ovládány tzv. herní smyčkou, která běží nepřetržitě během hraní hry. Před každým vykreslením snímku je potřeba zpracovat vstup od uživatele a aktualizovat stav hry [8]. Godot k tomu poskytuje všem uzlům dvě virtuální metody, které po definování volá engine automaticky: `_Process` a `_PhysicsProcess`.

Rozdíl mezi těmito dvěmi metodami je ve frekvenci jejich volání. Metoda `_Process` aktualizuje uzel každým snímkem tak často, jak je to možné. Závisí na snímkové frekvenci, která se však časem i napříč zařízeními může měnit. O konstantní průběh se naopak snaží metoda `_PhysicsProcess`, standartně 60krát za sekundu a nezávisle na snímkové frekvenci. To se hodí pro cokoli, co vyžaduje výpočet fyziky, jako např. pohyb těla kolidujícího s prostředím.

Obě tyto metody používají parametr „delta“, která reprezentuje uplynutý čas v sekundách od posledního volání těchto metod. Tento parametr umožňuje provádět výpočty nezávislé na snímkové frekvenci, jako např. rychlosti pohybujících se objektů [9].

2.4.2 Signály

Signály jsou forma zpětného volání. Každý typ uzlu má k dispozici své vlastní signály, ale je možné ve skriptech definovat i vlastní [7].

Jedná se o např. zmáčknutí tlačítka, vypršení času v časovači, vstup objektu do oblasti atd. Pokud na tyto signály navážeme funkci určitého skriptu, tato funkce se bude volat vždy, když k dané akci dojde.

3 Další použitý software

I přes rozsáhlé množství nástrojů, které Godot poskytuje, se tvorba her neobejde bez vnějších prostředků, ať už pro tvorbu grafiky nebo zvuků. Volba programování v jazyce C# navíc vyžaduje vnější programovací prostředí.

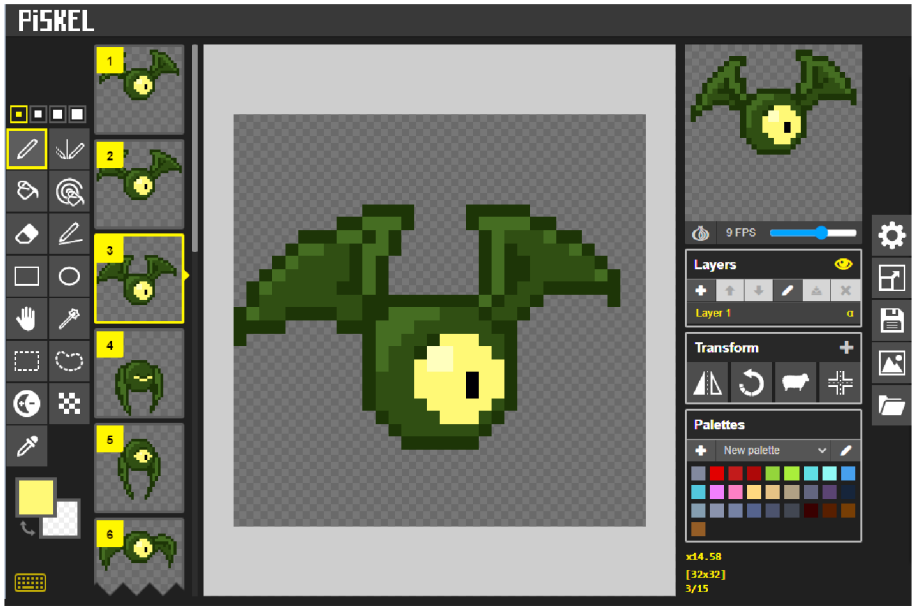
3.1 Visual Studio Code

Visual Studio Code je editor zdrojového kódu vyvíjený společností Microsoft s nabídkou bohatého ekosystému rozšíření pro různé jazyky [10]. Pro spolupráci editoru s Godotem je potřeba (kromě jazyka C#) nainstalovat rozšiřující balíčky „Mono Debug“ a „godot-tools“. V nastavení editoru v Godotu lze poté aktivovat možnost „Použít externí editor“, vložit k němu cestu a otevírané skripty se tak budou automaticky zobrazovat ve Visual Studiu Code.

Tento editor jsem zvolil díky jeho jednoduché synchronizaci s Godotem a také díky zkušenostem, které jsem s ním během studia získal.

3.2 Piskel

Pro tvorbu a úpravu *spritů* jsem zvolil bezplatný online editor *Piskel*[6]. Nabízí základní nástroje pro práci s *pixel art*[7] grafikou, ať už statickými obrázky nebo animaceni. Umožňuje navíc i ukládání rozpracovaných projektů, takže je možné se k nim kdykoli vrátit.



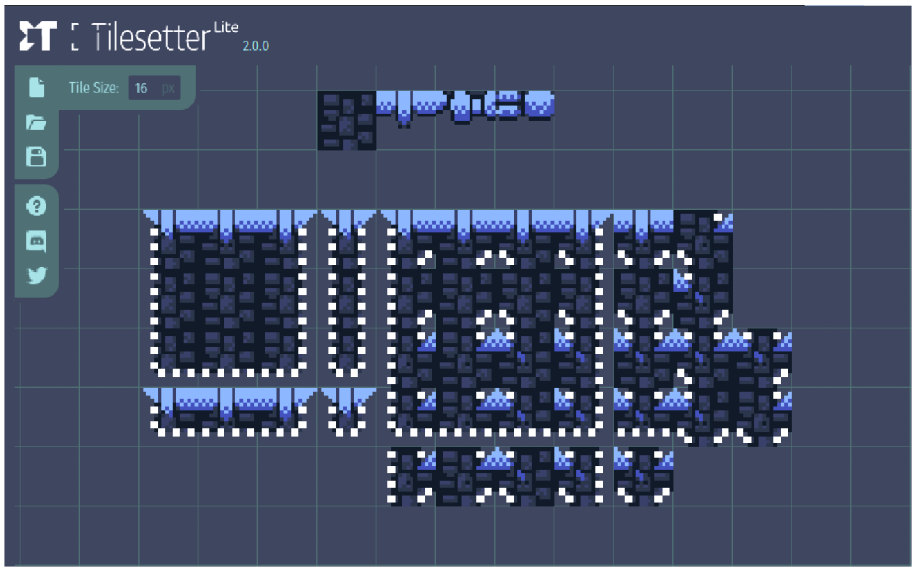
Obrázek 2: Uživatelské rozhraní aplikace Piskel

[6] https://www.piskelapp.com/

[7] Druh počítačové grafiky vytvářené po jednotlivých pixelech.

3.3 Tilesetter

Tilesetter je software zjednodušující tvorbu *tilesetů*[8]. Pro lépe působící vizuální stránku hry je vhodné měnit vzhled dlaždic na základě jejich vzájemném umístění. Do Tilesetteru stačí vložit pouze prostřední dlaždici s jejími okraji a software z poskytnutých materiálů vypočítá a vykreslí sadu dlaždic včetně jejich speciálních případů. Z takto sestavené sady lze potom v Godotu vytvořit tzv. *autotile*[9], poskytnutím údajů, na které pozici se mají dané dlaždice zobrazovat. Při tvorbě úrovní hry tak stačí „kreslit“ myší a dlaždicím se bude nastavovat *sprite* automaticky na základě jejich pozice vůči ostatním.



Obrázek 3: Tvorba tilesetu v aplikaci Tilesetter

3.4 Chiptone

Zvukové efekty použité ve hře jsem vytvářel pomocí webové aplikace Chiptone[10]. Aplikace nabízí výběr generování zvuku z několika přednastavených kategorií (skok, zranění, sebrání mince atd.) a poskytuje i základní nástroje pro další možnou úpravu těchto vyprodukovaných efektů.

[8] Sada dlaždicových spritů, jejichž opakováním je tvořeno prostředí úrovní hry.
[9] Systém vykreslování dlaždic, který podle okolí vybírá dlaždici vhodnou pro dané místo.
[10] https://sfbgames.itch.io/chiptone

Obrázek 4: Uživatelské rozhraní aplikace Chiptone

4 Obsah hry

Vytvořená hra je 2D akční plošinovka, inspirovaná tituly *The Messenger*, *Celeste* a *Hollow Knight*. Cílem hry je projít všemi úrovněmi hry. Dodatečným úkolem je zneškodnit všechny nepřátele a posbírat co nejvíce sběratelských předmětů v podobě jahod. Skóre s těmito údaji se zobrazí na konci hry.

Hráč má k dispozici meč, který mu umožňuje boj na blízko s nepřáteli. Jak hra postupuje, hráč získává nové schopnosti, jako jsou *dash*[11] a skok po zdech, a také vylepšení meče zvyšující jeho sílu. Hra je ukládána automaticky a při opětovném spuštění je hráči umožněno pokračovat od poslední dosažené úrovně. Ve hře je také rozmístěno několik záchytných bodů, tzv. *checkpointů*, aby hráč při ztrátě životů nemusel začínat od začátku. Na začátku hry má hráč na výběr ze tří možností obtížnosti: normální mód, pomocný mód (hráč má neomezený počet životů) a výzva (při ztrátě životů hráč nepokračuje od *checkpointu* a hra končí). Co se týče příběhu, hráč má za úkol uniknout z jeskyně, do které hlavní postava hry na začátku spadla. Podle toho hra nese název „Cave Escape“ (v překladu „Únik z jeskyně“).

4.1 Úrovně hry

Hra obsahuje 25 úrovní. Jedná se o kombinace statických úrovní, které svou velikostí odpovídají oknu hry a kamera se tím pádem nepohybuje, a tzv. horizontálních nebo vertikálních úrovní, ve kterých kamera následuje hráče v jednom z těchto směrů. Na základě získaných schopností hráče se hra dělí na 3 etapy. První etapa zahrnuje 10 úrovní, které hráče učí základní pohyb a útok, a je zakončena bojem proti *bossovi*[12]. Na začátku druhé etapy hráč získá schopnost *dash*. Tato část hry obsahuje 7 úrovní, které hráče naučí novou schopnost používat. Zde hráč také může získat vylepšení jeho meče. V poslední etapě se hráč naučí skákat po zdech a kombinovat tuto schopnost s už naučeným *dashem*. To vše je příprava na výzvu v podobě poslední úrovně hry. Jedná se o vertikální úroveň testující akrobatické dovednosti hráče v časovém presu, který vytváří zvyšující se hladina lávy. Zhruba v polovině úrovně navíc začnou shora padat kameny, které přidávají na obtížnosti. Po dokončení úrovně se hráči zobrazí čas, který oznamuje, jak dlouho mu poslední úroveň trvala a podle rychlosti hráče hra ohodnotí známkou.

4.1.1 Speedrun režim

Pokud má hráč zájem o vylepšení svého času, může poslední úroveň spustit z výběru úrovní v hlavním menu, což úroveň načte přizpůsobenou pro *speedrunning*[13]. V tomto režimu se hráči zobrazuje aktuální čas a pokud už úroveň dříve dokončil, zobrazuje se i jeho nejrychlejší průchod v podobě tzv. ducha hráče (postava

[11] Časově omezený většinou horizontální pohyb ve zvýšené rychlosti.
[12] Silnější nepřítel, jehož poražení vyžaduje určitou dávku strategie.
[13] Herní disciplína, jejímž cílem je dokončit danou hru, nebo vybranou část, co nejrychleji.

hráče se zvýšenou průhledností). Pro eliminaci náhodnosti, která je u *speedrunningu* frustrující, jsou zrušeny padající kameny. Pokud se hráči podaří dosáhnout lepšího času, starý záznam je přepsán novými daty a časem.

Záznam se ukládá tak, že každým snímkem hry (tzn. 60× za sekundu) se zapisuje pozice hráče v souřadnicích, aktuální obrázek animace a horizontální směr hráče. Průměrná doba průchodu úrovní se pohybuje kolem minuty, rekord je 33 sekund. Pokud hráč dosáhne osobního rekordu, zapsané údaje a nový čas se uloží do souboru typu JSON. Při novém spuštění úrovně se hráči přehrává tento nový záznam.

I přes zdánlivě vysoké množství údajů, které ukládáme, uložený soubor obsahuje pouze 200 kB. Toto je však možné optimalizovat jiným způsobem ukládání, který zapisuje vstupy hráče v závislosti na čase. Při stisknutí klávesy by se zapsal identifikátor klávesy a počet snímků představující délku držení dané klávesy. Tento způsob se používá např. v závodních hrách. Tento způsob je však komplikovanější a jelikož ve hře dochází k rychlému střídání kláves, zvolil jsem pro jednoduchost metodu ukládání údajů postavy každým snímkem hry, která při přehrávání zaručuje přesnou reprezentaci hráčova průchodu úrovní.

4.2 Životy hráče

Maximální počet životů hráče je 5. Životy jsou ubrány po kontaktu s nepřáteli nebo jinými nástrahami. Při ztrátě všech životů hráč pokračuje od posledního záchytného bodu hry. Životy je však možné dobíjet za body získané porážením nepřátel. Tyto body jsou ve hře reprezentované kulatou nádobou nalevo od ukazatele životů a jakmile je nádoba plná, hráči se doplní život, pokud zrovna nemá plný počet, a ukazatel se vynuluje.

4.3 Nepřátelé

Hráč ve hře narazí na tři hlavní typy nepřátel. Jsou to brouk, kostlivec a létající příšera s jedním okem (dále jen létající příšera). Brouk se pohybuje po vodorovných plošinách a mění směr, kdykoli narazí na konec plošiny nebo na zeď. Kostlivec se také pohybuje horizontálně, na rozdíl od brouka však drží meč, kterým na hráče útočí ve chvíli, kdy je na dosah. Na kostlivce je možné narazit také ve formě „hromady kostí“, která se v blízkosti hráče do kostlivce zformuje. Létající příšera je unikátní v tom, že se v klidovém stavu nachází na jednom místě v prostoru a jakmile se hráč přiblíží, letí přímo směrem k němu.

4.4 Nástrahy

Ve hře jsou dva typy nástrah: ostny a láva. Jakmile se jich hráč dotkne, je mu ubrán život a hráč je přemístěn na bezpečné místo, většinou na začátek úrovně. K přemístění hráče dochází z toho důvodu, aby nebyl omezován *level design*[14]

[14]Disciplína vývoje her, která zahrnuje vytváření úrovní, lokalit, misí nebo etap videoher.

ve vytváření hlubokých jam plných ostnů (nebo lávy), ze kterých by se hráč nemohl dostat ven.

4.5 Sběratelské předměty

Po úrovních je rozmístěno celkem 22 sběratelských předmětů v podobě jahod (inspirováno videohrou *Celeste*). Tyto předměty se většinou nachází na obtížněji dostupných místech a představují dodatečnou výzvu pro hráče, kteří jsou ochotni je posbírat. Při získání jedné z jahod se na obrazovce pod ukazatelem životů zobrazí, kolik jich hráč už posbíral a kolik mu chybí.

4.6 Grafika

Veškeré animace hlavní postavy, kostlivce a *bosse* jsem získal z webu *itch.io*. Animace hráče jsou od tvůrce *rvros* [15] a kostlivec s *bossem* je od *sanctumpixel*[16].

Použité fonty jsou *Adventurer*[17] od autora *Brian J Smith* a *TinyUnicode*[18] od autora *DuffsDevice*, dostupné v galerii webu *pentacom*.

Animace brouka s létající příšerou, pozadí a sprity ostatních prvků hry jsem vytvořil v aplikaci Piskel.

[15] https://rvros.itch.io/
[16] https://sanctumpixel.itch.io/
[17] http://www.pentacom.jp/pentacom/bitfontmaker2/gallery/?id=195
[18] http://www.pentacom.jp/pentacom/bitfontmaker2/gallery/?id=468

5 Implementace

V této kapitole čtenáře seznámím s implementací hlavních prvků vytvořené hry. Zaměřím se především na chování a ovládání hlavní postavy hry, na nepřátele a popíšu, jak je ve hře ukládán hráčův pokrok. Odlišným typem písma jsou znázorněny identifikátory z kódu.

5.1 Hlavní postava hry

Hlavní postava je z pohledu uzlů v Godotu implementovaná jako samostatná scéna následně instancovaná ve všech úrovních hry. Její kořenový uzel je typu `KinematicBody2D`, který umožňuje interakci s okolním prostředí díky povinnému podřízenému uzlu typu `CollisionShape2D` reprezentující kolize. Dále hráč obsahuje podřízený uzel typu `Sprite`, ve kterém je uložen *spritesheet*[19] animací postavy, `Area2D` reprezentující *hitbox*[20] meče, další uzel typu `Area2D` pro *hurtbox*[21] a `AnimationPlayer`, který podle vytvořených animací ovládá přepínání jednotlivých spritů uzlu `Sprite` a aktivuje či deaktivuje *hitbox* meče během animace pro útok. Dále scéna obsahuje uzly typu `RayCast` pro detekci zdí, druhý přehrávač animací zajišťující problikávání hráče po ztrátě životů, časovače a uzel pro přehrávání zvuku. Pro správnou synchronizaci všech těchto uzlů je kořen scény doplněn o skript, který tyto uzly ovládá a řídí jejich komunikaci.

5.1.1 Stavový automat

Chování hráče je implementováno s využitím tzv. stavového automatu. Tento princip funguje na rozdělení kódu podle stavů, ve kterých se hráč může nacházet a přechody mezi těmito stavy jsou přesně definovány. Výhoda tohoto řešení je v přehlednosti kódu a také v jeho optimalizaci, protože tak stačí provést vždy pouze zlomek kódu na základě stavu hráče. Kód stavového automatu se nachází v metodě `_PhysicsProcess` a vyhodnocuje se při každém vykresleném snímku hry.

Každý stav se dá rozdělit na 3 části: chování, přechody a animace. Chování udává, jak se bude postava v daném stavu chovat, přechody definují všechny možné cesty z aktuálního stavu do jiného a animace má za úkol nastavit postavě odpovídající animaci pro daný směr.

Celkem se hráč může nacházet ve 12 různých stavech, které jsem pro vyšší přehlednost rozdělil do 3 skupin. Mezi hlavní stavy jsem zařadil ty, do kterých se postava může dostat na základě vstupů od uživatele a jsou dostupné od začátku hry. Jedná se o stavy, v kódu pojmenované: `idle`, `run`, `air`, `attack`, `crouch`, `hurt` a `death`. Další stavy, které hráč „odemkne“ během hry jsou `dash` a `wallSlide`, reprezentující nové schopnosti hráče. V poslední skupině

[19] Obrázek poskládaný z několika menších spritů.
[20] Zásahová oblast způsobující poškození.
[21] Oblast monitorující kolize s nepřáteli.

jsou stavy, do kterých postavu může dostat hra samotná, což umožňuje vyšší kontrolu nad hráčem. Jsou to stavy `prepared`, `noInput` a `freeze`.

5.1.1.1 Stav `idle`

Stav `idle` je tzv. klidový stav, do kterého se postava dostane, pokud se shora dotýká plošiny a uživatel nedodává žádný vstup. K ověření, že se postava plošiny dotýká, slouží predikát `IsOnFloor`. Aby tento predikát fungoval správně, je potřeba na postavu neustále aplikovat gravitaci, neboli konstantní vertikální rychlost směrem dolů. I když se hráč vertikální rychlostí nepohybuje, kolize s plošinou předá signál predikátu `IsOnFloor`, který kontakt s plošinou potvrdí. I přes to, že tento stav reprezentuje stání na místě, pokud se při přechodu do tohoto stavu postava pohybovala určitou horizontální rychlostí, stav zajistí plynulé zpomalení postavy funkcí `Friction`. Tyto dvě zmíněné funkce počítají horizontální a vertikální rychlost postavy reprezentovanou vektorem `motion`. Tento údaj je nakonec předán metodě `MoveAndSlide`, která s ohledem na kolize s ostatními objekty hry pohyb vykoná. Dále jsou definovány přechody do stavů `run`, `air`, `attack` a `crouch`, znázorněny na diagramu. Přechody do stavů `hurt` a `death` jsou zajištěny zvnějška pomocí signálu, který je vyslán při kolizi *hurtboxu* hráče s *hitboxem* nepřátel. Pokud se žádný přechod neprovedl, podle směru hlavní postavy se uzlu `AnimationPlayer` sdělí, kterou animaci má přehrávat. Tímto způsobem jsou napsány i ostatní stavy.

5.1.1.2 Stav `run`

Do stavu pro běh se hráč dostane stisknutím šipky doleva nebo doprava, pokud se dotýká plošiny. Stav plynule akceleruje horizontální rychlost postavy, dokud rychlost nepřesáhne maximální rychlost nastavenou pro běh. Pokud ano, rychlost je nastavena na maximální. Dále se aplikuje gravitace a metoda `MoveAndSlide` pohyb vykoná. Přechody do ostatních stavů jsou stejné jako u stavu `idle`, pouze přechod do stavu run je nahrazen přechodem zpět do stavu `idle`. Podle směru pohybu se přehrává odpovídající animace.

5.1.1.3 Stav `air`

U některých stavů je potřeba vykonat určité akce před provedením přechodu. Před vstupem do stavu `air` je např. potřeba nastavit hráči počáteční vertikální rychlost, přehrát zvuk výskoku a taky vytvořit částice odražení od plošiny. Stav poté na danou vertikální rychlost plynule aplikuje gravitaci dokud postava nedopadne zpět na plošinu. Skok hráče má zároveň volitelnou výšku v závislosti na době držení klávesy pro skok. To je realizováno tím, že jakmile hráč pustí klávesu skoku, pokud je hráčova vertikální rychlost záporná (postava se pohybuje směrem nahoru), je tato rychlost okamžitě snížena na třetinu a skok se tímto sníží. Postava dále může nabývat 3 různé animace na základě vertikální rychlosti. Pohyb nahoru a dolů rozlišuje různé animace a zároveň, pokud hráč drží

šipku pro skok dostatečně dlouho, postava hráče provede ve vzduchu kotoul. To je zajištěno proměnnou `longJump`, která se se stisknutím klávesy skoku nastaví na `true` a jestliže hráč tuto klávesu uvolní, proměnná se nastaví na `false`. Část kódu, která má na starost přehrávání animací, pak sleduje vertikální rychlost hráče. Když se tato rychlost blíží k nulové hodnotě (postava začíná zpomalovat a začíná padat), zkontroluje se proměnná `longJump` a pokud je pravdivá, přehraje se odpovídající animace kotoulu.

5.1.1.4 Stav `crouch`

Do stavu pro dřep se hráč dostane zmáčknutím šipky dolů při doteku plošiny. Z tohoto stavu se hráč dostane do jiného uvolněním této klávesy. Když je hráč ve dřepu, jeho oblast představující kolize s nepřáteli se sníží, aby odpovídala *spritu* postavy.

5.1.1.5 Stav `attack`

Do stavu pro útok hráč může přejít ze všech předchozích stavů včetně `air` (lze útočit i ve vzduchu). Při snímku animace, ve kterém dochází ke švihu, se uzel `AnimationPlayer` postará o aktivování *hitboxu* meče i o následné deaktivování. Po přehrání animace nastane přesun do stavu podle toho, kde se hráč nachází a která klávesa je stisknutá. Pokud se postava nedotýká plošiny, přechod do stavu `air` už nevyžaduje nastavení vertikální rychlosti, jelikož hráč pokračuje v pádu.

5.1.1.6 Stavy `hurt` a `death`

Kdykoli hráč přijde do kontaktu s nepřáteli nebo jiným nebezpečím v podobě ostnů či lávy, kolize *hurtboxu* vyšle signál. V reakci na tento signál se vyhodnotí funkce `onHurtboxAreaEntered`. Tato funkce se spustí bez ohledu na to, ve kterém stavu se postava hráče právě nachází. Před přechodem do stavu `hurt` funkce odečte hráči život. Pokud hráči po odečtení zbývá kladný počet životů, funkce zařídí přehrátí zvuku pro zranění, zatřesení kamery a spustí problikávání hráče. Pokud se hráč dotknul ostnů či lávy, hra ho přesune na bezpečné místo, umístěné většinou na začátku úrovně a převede do stavu `noInput` (viz kapitola 5.1.1.8). Pokud kolizi s hurtboxem způsobil nepřítel, nastaví se hráči záporná vertikální rychlost simulující odhození a přejde se do stavu `hurt`, který pokračuje v přehrávání animace zranění, aplikuje gravitaci a pohybuje hráčem směrem od nepřítele. Po ukončení animace hráč pokračuje stavem `air`. Pokud však hráči životy došly, je vyslán signál, který zavolá funkci `onNoHealthSignal` a v ní se provedou odpovídající akce, jako např. přehrání animace úmrtí, zvuku, deaktivace kolizí, spuštění časovače, rozmazání a snížení saturace hry současně s přechodem do stavu `death`. V tomto stavu je pouze aplikovaná gravitace, protože po uplynutí času v časovači se změní scéna na úroveň podle posledního uloženého *checkpointu*.

5.1.1.7 Odemknutelné stavy

Do této kategorie patří stavy `dash` a `wallSlide`. Stavy jsou nazvány „odemknutelné“, jelikož nejsou hráči dostupné od začátku hry. Toto je zajištěno proměnnými `canDash` a `canWallJump`, které jsou od začátku nastaveny na hodnotu `false`. Podmínka přechodu do těchto stavů je v kódu realizována konjunkcí požadavků přechodu s odpovídající proměnnou. Hodnota těchto proměnných se přepíše na `true`, jakmile hráč danou schopnost ve hře získá.

Přechod do stavu `dash` je doprovázen nastavením časovače, zvukovým efektem, zatřesením obrazovky a *dash* efektem (hráč za sebou zanechá několik snímků sebe v dash animaci, které po chvíli zmizí). V tomto stavu se hráč pohybuje zvýšenou horizontální rychlostí v daném směru bez vlivu gravitace. To hráči umožňuje uskutečnit delší skoky nebo provést přesun na plošinu stejné výšky při nízkém „stropu“. Po vypršení doby trvání se hráč vrátí do vhodného stavu dle jeho pozice. Aby tato schopnost nebyla příliš silná, je omezena mírnou časovou prodlevou a ve vzduchu ji lze použít pouze jednou.

Druhý stav zpřístupněný v průběhu hry se nazývá `wallSlide` a představuje „sjíždění“ po zdi, které od této zdi umožňuje výskok. Přechod do tohoto stavu je možný, když je hráč ve vzduchu u libovolné zdi a drží klávesu pro pohyb směrem k ní. Přidržení této klávesy se zobrazí odlišná animace a vertikální rychlost hráče je oproti pádu ve stavu ve vzduchu poloviční. Tento stav umožňuje hráči výskok s přechodem do stavu `air`. Odraz od zdi je zajištěn pomocí krátkého časovače. Ten je spuštěn při přechodu do stavu `air` a dokud čas nevyprší, hráči je na chvíli nastavena při výskoku určitá horizontální rychlost směrem od zdi. Tento odraz plní především vizuální funkci, ale i z pohledu hratelnosti zjednodušuje skákání na protější zeď.

5.1.1.8 Vedlejší stavy

Poslední 3 stavy, do kterých se hráč může dostat, nejsou dostupné přechodem z jiných stavů a jejich přecházení rozhoduje herní systém.

Stav `prepared` jako jediný z této skupiny plní pouze vizuální funkci. Tento stav přehrává animaci dřepu hráče a pomáhá vykreslit osobnost postavy, aby nepůsobila příliš uměle. Postava se v tomto stavu nachází na začátku první úrovně, při přemístění nebo načtení úrovně jinak, než průchodem předchozí (*checkpoint* nebo načtení uložené hry). Z tohoto stavu se hráč dostane stisknutím klávesy pro pohyb.

Stav `noInput` odebírá hráči kontrolu nad postavou. Tento stav je použitý na začátku bitvy s *bossem* při čekání na zahájení. Pokud je hráč ve vzduchu, plynule dopadne na plošinu, horizontální rychlost se plynule zpomalí na nulu. Až je hra připravena, postavě jednoduše nastaví stav `idle` a hráč získá kontrolu nad ovládáním zpět.

Poslední stav je nazván `freeze`. Tento stav nejen odebere hráči kontrolu nad ovládáním, postava v něm doslova „zamrzne“ na jednom místě. Pro tento stav není potřeba žádný kód a je použit až na konci hry.

5.1.2 Tolerantní ovládání

Cílem vývojáře hry je zajistit, aby ovládací prvky hry reagovali tak, jak hráč v závislosti na jeho vstupech očekává. To někdy vyžaduje dodat ovládání určitou volnost. Hraní hry se může rychle stát špatnou zkušeností, pokud ovládání není přesné, ale to samé může nastat i pokud je přesné příliš [11]. V následujících kapitolách popíšu jednotlivé principy často používaných řešení tohoto problému, které jsou také implementované v této hře.

5.1.2.1 Coyote time

Dává smysl, že hráč může vyskočit, pouze když se dotýká plošiny. Pokud ale chce hráč skočit z plošiny co nejdál, může se stát, že stisknutí klávesy pro skok načasuje pár milisekund po tom, co přeběhl hranu. Z pohledu herní fyziky není nic špatného na tom, že postava nevyskočí, hráči však tento výstup může přijít nespravedlivý, protože z jeho pohledu stiskl tlačítko ve správnou chvíli.

Tento problém lze vyřešit pomocí nastavené doby, po kterou hráč může vyskočit i po tom, co opustil plošinu. Této vlastnosti se říká *coyote time*. Ve hře je toto implementováno pomocí časovače, který se začne odpočítávat po přechodu hráče ze stavu `run` do stavu `air`. Pokud hráč v tomto stavu stiskne tlačítko pro výskok, hra zkontroluje, zda nevypršel daný časovač a pokud ne, hráči se přidá určitá vertikální rychlost reprezentující výskok. Časovač se přitom vynuluje, aby hráč nemohl skočit vícekrát.

Tento princip je použitý i při skákání po zdech, kde hráč může vyskočit i chvíli po tom, co opustil stav `wallSlide` a přestal se dotýkat zdi.

5.1.2.2 Jump buffer

Další problém je velmi podobný předchozímu. Nastává v situaci, kdy hráč stiskne tlačítko pro skok těsně před dopadem. Opět se postava sice nedotýká plošiny a je pořád ve stavu `air`, jenže hráči se může zdát, že hra jeho stisknutí nezaznamenala. Toto lze však opět vyřešit pomocí dalšího časovače.

Pokud se postava nachází ve vzduchu (ve stavu `air`) a hra zaznamená stisknutí tlačítka pro výskok, spustí se odpočet časovače nastaveného na konkrétní dobu. Ve této hře je to realizováno 6 snímky[22] (0.1 s). Pokud v daném časovém okně po stisknutí klávesy postava hráče dopadne na plošinu, hráč automaticky vyskočí.

5.2 Nepřátelé

Ve hře jsou celkem 4 různí nepřátelé. Pro každý typ nepřítele byla vytvořena samostatná scéna, následně instancovaná v úrovních hry podle potřeby. Kořenový uzel těchto scén je typu `KinematicBody2D` a je doplněn o skript. Dále

[22] Hra běží rychlostí 60 snímků za sekundu.

scéna obsahuje uzel typu `Sprite` a `AnimationPlayer` pro přehrávání animací. Všem je nastavena oblast pro kolize s okolním světem a další 2 oblasti pro reprezentaci *hurtboxu* a *hitboxu*. Stejně jako postava hráče byli nepřátelé také realizováni pomocí stavového automatu.

5.2.1 Brouk

První nepřítel, na kterého hráč ve hře narazí, je brouk. Pohybuje se horizontálně po plošinách a jakmile narazí na stěnu nebo hranu plošiny, tak změní směr na opačnou stranu. Toto je realizováno pomocí uzlů typu `RayCast`. Tento uzel reprezentuje linku mezi 2 pozicemi a v tomto prostoru počítá kolize s jinými oblastmi. Brouk má na svých obou stranách nastaveny tyto linky směrem od sebe pro zjištění kolize se stěnami a směrem dolů pro detekci hrany plošiny.

Broukův stavový automat obsahuje 3 stavy: `walk`, `fall` a `death`. První stav reprezentuje chůzi konstantní rychlostí jedním směrem. Jeho směr je obrácen po zmíněné detekci konce plošiny, ale také po kontaktu *hurtboxu* s *hitboxem* hráčova meče. Po zásahu hráčem je broukovi také zvýšena horizontální rychlost v novém směru, která je následně plynule zpomalena funkcí `Friction`. Jelikož toto „odhození“ může brouka shodit z plošiny byl mu vytvořen i stav pro pád. V tomto stavu se přestane přehrávat animace pro chůzi a je na brouka aplikována gravitace funkcí `Gravity`. Po dopadu na jinou plošinu brouk přejde zpět do stavu `walk`. Pokud brouk dopadne na nebezpečný objekt (např. ostny), jsou mu také ubrány životy, stejně jako ostatním nepřátelům. Jakmile brouk přijde o životy, přejde do stavu `death`, přehraje se odpovídající animace a hráči se zvýší počet zdolaných nepřátel společně se zvýšením hodnoty ukazatele pro dobytí života.

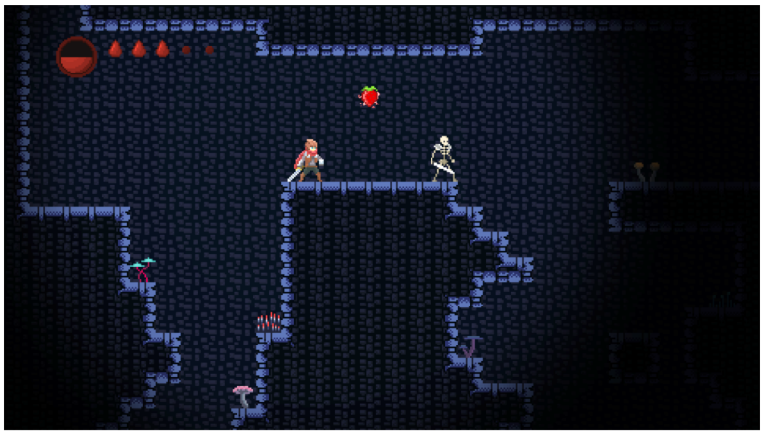
5.2.2 Kostlivec

Kostlivec stejně jako brouk chodí po plošinách horizontálním pohybem, ale pokud je hráč na dosah, zaútočí na něj. Kostlivec může být v 7 různých stavech: `idle`, `walk`, `attack`, `hurt`, `death`, `corpse` a `reborn`. Od brouka se liší také tím, že když dorazí na konec plošiny, tak na určitou dobu přejde do stavu `idle` a vyčkává. Kostlivec nemá samostatný stav pro pád, pád je vyřešen přechodem do stavu `idle`, který plní potřebné chování. Kostlivec se také pro prvek překvapení může na počátku nacházet ve stavu `corpse`, ve kterém je deaktivován jeho *hitbox* a nemůže hráči ubrat život, ale jakmile se hráč dostane do jeho blízkosti, přejde do stavu `reborn`, ve kterém se přehraje animace zformování a na konci této animace kostlivec přejde do `idle`, nebo rovnou `attack`, pokud je hráč stále v dosahu. Zjištění, zda je hráč v blízkosti je zařízeno dodatečnými uzly typu `RayCast`.

5.2.3 Létající příšera

Tento typ nepřítele má k dispozici 3 stavy: `idle`, `chase` a `returning`. V prvním stavu se příšera nachází na jednom místě, přehrává se animace létání a časovač střídá její směr pro vyjádření toho, že se příšera rozhlíží. Scéna tohoto typu nepřítele obsahuje oblast, do které když hráč vejde, příšera přejde do stavu `chase` a začne hráče pronásledovat. K tomu je využita funkce `MoveToward`, která požaduje 2 parametry pro udání směru a rychlosti. Rozdílem pozice příšery od aktuální pozice hráče se zjistí směr pohybu příšery. Ten je poté vynásoben konstantou s hodnotou maximální rychlosti příšery. Jako druhý parametr je zadána proměnná `delta` reprezentující čas uplynutý od posledního snímku. Tato hodnota je vynásobena konstantou akcelerace příšery pro úpravu výsledného pohybu. Jakmile hráč opustí danou oblast, příšera se začne vracet na počáteční místo, které je při spuštění úrovně uloženo do proměnné. K realizaci návratu je opět použita funkce `MoveToward`. Při návratu na toto místo přejde zpět do stavu `idle`.

Pro létající příšeru jsou dále napsány ještě další 3 stavy, které mají na starost chování příšery po úmrtí: `deathAir`, `deathFall` a `deathGround`.

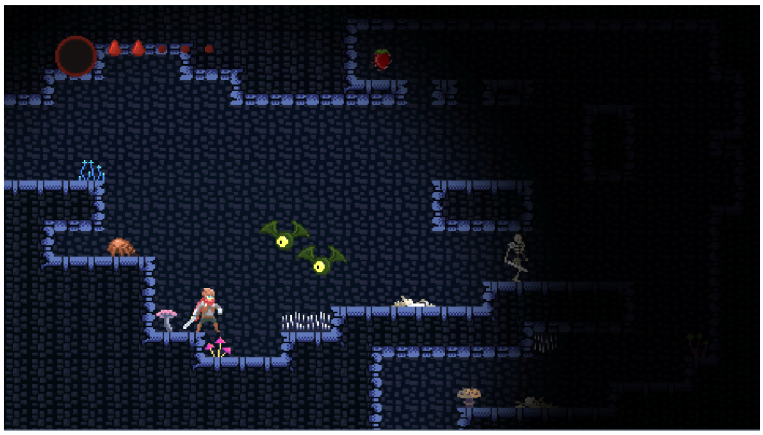


Obrázek 5: Ukázka ze 4. úrovně hry

5.2.4 Boss

Na *bosse* hráč narazí v 10. úrovni. Stejně jako u ostatních nepřátel má kořenový uzel typu `KinematicBody2D`, má vlastní *hurtbox* a *hitbox*. K detekování hráče používá uzly typu `RayCast`.

Chování *bosse* je implementováno pomocí 13 stavů, z toho 8 slouží pro různé části útoků. Při vstupu do místnosti je *boss* otočen směrem od hráče a nachází se ve stavu `initial`. Zhruba ve středu místnosti je přidána oblast, která při detekci hráče zajistí přechod hráče do stavu `noInput` a dveře „zavřou“ hráče v místnosti. Ten se otočí, „představí“ se výkřikem (stav `battlecry`) a na obrazovce ukáže jeho ukazatel životů. Dveře, které hráče zavřou v místnosti jsou



Obrázek 6: Ukázka ze 7. úrovně hry

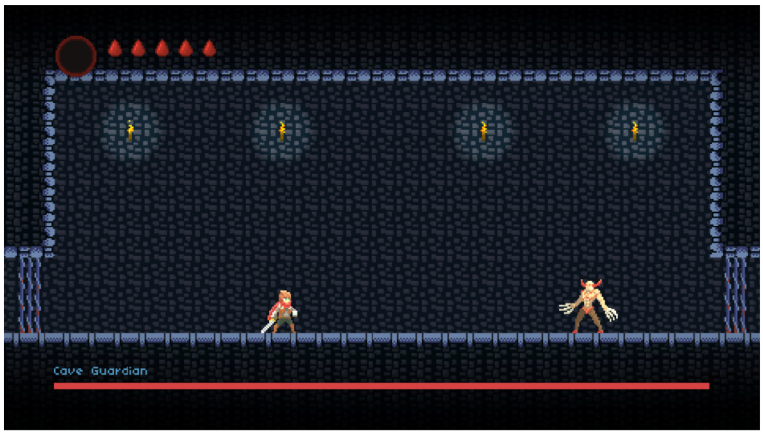
realizovány oblastmi, které jsou zpočátku deaktivované a po jejich aktivaci je zajištěna neprůchodnost. Po vypršení časovače *boss* opustí stav `battlecry` přechodem do stavu `idle` a hráči je vrácena kontrola nad postavou nastavením jejího stavu také na `idle`.

Boss má naprogramovány 4 různé typy útoků. Prvním je běh ve směru hráče a jakmile je v dosahu, zaútočí. Druhý typ útoku je skok na hráče. Skok vždy opakuje 3krát po sobě. Tyto 2 útoky střídá od začátku náhodně. Jakmile má *boss* méně než polovinu životů, přidá k náhodnému výběru útok vyvolávání padajících kamenů. Při tomto útoku *boss* stojí na místě a hráč se musí vyhýbat kamenům, které by mu při nárazu ubraly 1 život. Tento útok trvá 10 sekund. Poté je zajištěn přechod zpět do stavu *idle* a po daném čase nastane znovu náhodný výběr následujícího útoku. Pomocí proměnné, která si pamatuje poslední provedený útok, je zajištěno, aby *boss* neprovedl tento útok 2krát po sobě. Poslední útok je tzv. „útok na blízko“ a je použitý, když je hráč v těsném dosahu.

Jakmile hráč *bossovi* sebere všechny životy, *boss* přejde do stavu `deathFall`, ve kterém se přehraje animace pádu a rychlost hry je snížena na 20 % nastavením vlastnosti `Engine.TimeScale`. Po přehrání animace se provede přechod do stavu `death`, současně se hráči přičte skóre a rychlost hry je vrácena do původního stavu.

5.3 Ukládání

Při spuštění hry se automaticky načte scéna `playerVariables`, která je přítomná po celou hru. Takto automaticky načítaná scéna se nazývá *Autoload* [12]. Uchovává důležité proměnné, které jsou trvalé i při změně úrovně. Jsou to např. životy, počet získaných jahod, síla meče a taky to, zda hráč může používat *dash* či skok po zdech. Tyto data jsou při přechodu do následující úrovně aktualizovány ve skriptu `DataModel.cs`. Ten je průběžně serializován do souboru typu JSON, který uchovává data i po zavření hry. Tento soubor se nachází ve složce `AppData`. Po zavření a opětovném načtení hry, hráč pokračuje od za-



Obrázek 7: Ukázka úrovně s *bossem*

čátku úrovně, ve které skončil. Hra proto aktualizuje ukládaná data při průchodu úrovněmi.

Pokud však hráč ztratí všechny životy, je přemístěn k poslednímu uloženému *chechpointu*. Hra z tohoto důvodu musí ukládat nejen informace o tom, ve které úrovni hráč prošel posledním *checkpointem*, ale i kolik měl tou dobou posbíráno jahod, zda měl získanou schopnost *dash* a tak dále. Tyto údaje uchovávají proměnné s prefixem „checkpoint“ a jsou aktualizovány pouze při průchodu *checkpointem*. Hra také ukládá informaci o tom, jaký herní mód hráč na začátku hry zvolil a podle této informace hra reaguje.

5.4 Implementace úrovní

Pro každou úroveň hry je vytvořena samostatná scéna. Každá scéna reprezentující jednotlivé úrovně obsahuje instancovanou scénu hlavní postavy, ukazatele životů, uzel pro tilemapu, sprite pozadí a objekty jako nepřátele, nebezpečí atd. Dále úrovně obsahují uzly, které mají na starost vzhled. Je to uzel typu `CanvasModulate`, který vytváří efekt tmy, `WorldEnvironment`, animovaný uzlem typu `AnimationPlayer`. Tyto uzly se starají o desaturaci a rozmazání obrazu, jakmile má hráč pouze 1 život.

Instance uzlu hlavní postavy obsahuje dva další potomky. Jedním je uzel typu `Camera2D` plnící funkci kamery s nastavitelnými limity. Jelikož je potomkem postavy hráče, automaticky tak hráče následuje po úrovni. Druhým potomkem je uzel typu `Light2D` reprezentující světlo, které taktéž následuje hráče.

Na konci každé úrovně je instance scény `NextLevel`. Je to oblast, která zajistí po kolizi s hráčem změnu scény na následující úroveň a aktualizaci uložených dat. Scéna každé úrovně je nazvaná „Level_*číslo*“. Díky tomuto konzistentnímu pojmenování lze jednoduchým algoritmem zařídit automatické přepínání úrovní hry. Algoritmus zjistí číslo z názvu aktuální úrovně, toto číslo inkrementuje, připojí k němu zpátky prefix „Level_“ a koncovku „.tscn“. Scénu s tímto názvem najde ve složce „Levels“ a spustí ji.

6 Uživatelská příručka

V této kapitole uživatele seznámím s hlavním menu a popíšu jednotlivé volby (viz kapitola 8) a dále stručně popíšu základ hry s jejím ovládáním (viz kapitola 6.2).

6.1 Hlavní menu

Po spuštění aplikace se uživatel objeví v hlavním menu hry. Menu nabízí 5 možností: *New Game*, *Load*, *Select Level*, *Controls* a *Quit* (viz obrázek 8). Pro navigaci v menu lze použít myš i klávesnici.

Možnost *New Game* uživatele přesune k nabídce volby obtížnosti (viz obrázek 9). Na výběr jsou 3 režimy hry: *Normal mode*, *Assist mode* a *Challenge*. Režim *Normal mode* nabízí doporučený způsob hraní hry, při ztrátě všech životů se hráč objeví s dobitými životy u posledního *checkpointu*. *Assist mode* představuje pomocný režim hry, ve kterém se hráči životy neodečítají a hráč se tak může soustředit na akrobatické prvky úrovní. Poslední možný režim *Challenge* představuje výzvu pro pokročilé hráče. Při ztrátě všech životů se hráč neobjeví u *checkpointu*, ale hra skončí a uložená data jsou smazána. Při volbě jednoho z těchto režimů se načte první úroveň hry. Pro návrat do hlavního menu slouží tlačítko *Back*.

Druhou možností v hlavním menu je volba *Load*, která načte uloženou hru v případě dřívějšího hraní. Pokud nejsou k dispozici žádná uložená data rozehrané hry, tlačítko je deaktivované.

Volba *Select Level* uživatele přesune k nabídce volby ze 4 zásadních úrovní hry (viz obrázek 10). Nabídka slouží spíše pro demonstrační účely, jelikož zpřístupňuje jednotlivé etapy hry bez nutnosti průchodu předchozích úrovní. Po výběru se odpovídající úroveň načte v režimu *Normal mode*. Tato úroveň se nastaví jako záchytný bod, dokud hráč neprojde dalším *checkpointem*.

Volba *Controls* uživateli ukáže, jak se hra ovládá. Uživatel si může vybrat mezi hlavním či alternativním způsobem ovládání podle toho, jaká tlačítka se rozhodne používat.

Poslední volba *Quit* slouží pro ukončení aplikace.

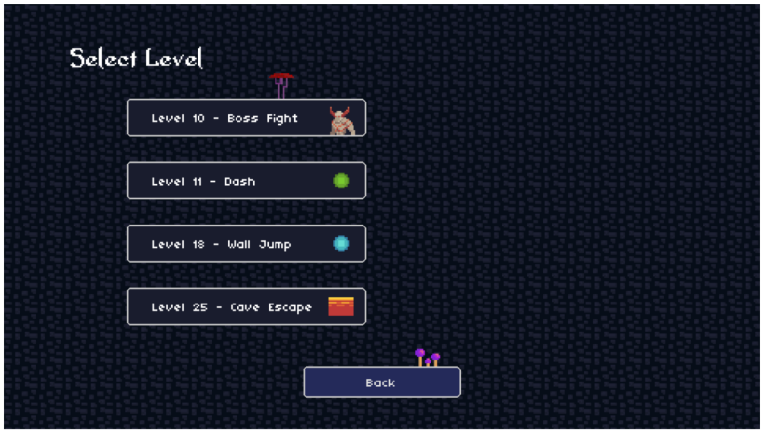
6.2 Hra

Cílem hry je projít všech 25 úrovní hry. Dodatečným úkolem je posbírat přitom co nejvíce sběratelských předmětů v podobě jahod a zneškodnit všechny nepřátele. Skóre s těmito údaji se zobrazí na konci hry. Životy hráče jsou zobrazeny ukazatelem v levém horním rohu společně s ukazatelem energie pro dobití životů.

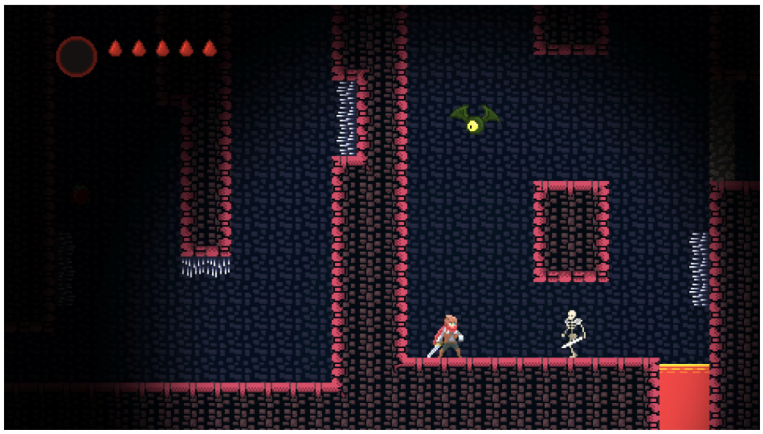
Hlavní postava hry se ovládá šipkami pro pohyb a výskok (výskok je volitelně vysoký v závislosti na době držení šipky nahoru), klávesa „X“ slouží pro útok a v pozdější fázi hry klávesa „C“ k provedení schopnosti *dash*. Alternativní ovládání je popsáno v menu v nabídce *Controlls*.

Obrázek 8: Hlavní menu

Obrázek 9: Výběr obtížnosti při zahájení nové hry

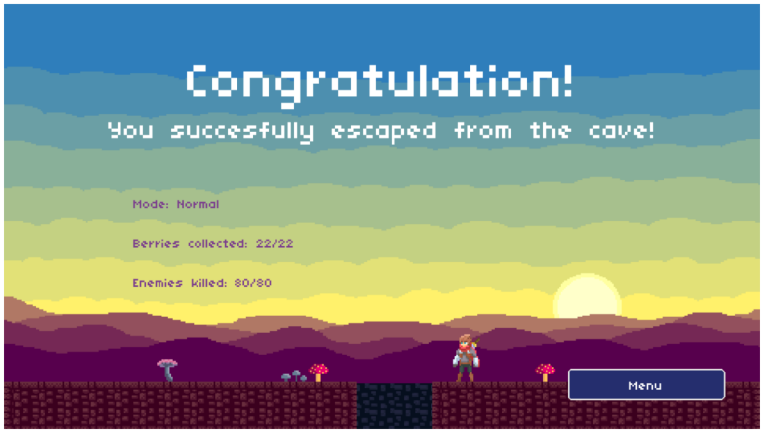


Obrázek 10: Výběr úrovní



Obrázek 11: Ukázka z 21. úrovně hry

Obrázek 12: Ukázka z poslední úrovně hry



Obrázek 13: Závěrečná obrazovka hry

Závěr

Výsledkem této bakalářské práce je 2D akční počítačová hra obsahující 25 úrovní, jejíž hraní může zabavit na více než hodinu. Pohyb hráče je zábavný na ovládání a hra má plynule narůstající obtížnost. Nepřátelé i překážky jednotlivých úrovní představují určitou výzvu. Hra nabízí tři druhy obtížnosti, tím zajišťuje přístupnost širšímu spektru hráčů. Herní pokrok je automaticky ukládán, takže není nutné hru dohrát v jednom sezení. S finálním výsledkem jsem velmi spokojený, i když prostor pro vylepšení by se určitě našel. Je například možné přidat více druhů nepřátel včetně bojů s *bossem* a nabídnout hráči více schopností (např. „dvojskok“). Hra sice obsahuje zvuky, ale bylo by ji možné obohatit o hudbu hrající v pozadí.

Co se týče enginu Godot, pracovalo se mi s ním pohodlně. Jeho intuitivní systém scén a uzlů je nenáročný na pochopení, a i když C# není hlavní programovací jazyk enginu, jeho podpora je velmi dobrá. Klíčové metody a funkce pro práci s uzly mají stejný název jako v hlavním jazyce enginu, jen je stačí psát notací specifickou pro C#. V případě nejasností Godot nabízí podrobnou dokumentaci dostupnou na webu a obsáhlá komunita vývojářů zaručuje jeho neustálé zdokonalování. Celkově je Godot velmi mocný a dostupný nástroj pro vývoj her.

Conclusions

The result of this bachelor's thesis is a 2D action computer game containing 25 levels, which can be enjoyed for more than an hour. The player's movement is fun to control and the game has a smooth learning curve. Enemies and obstacles at different levels represent a solid challenge. The game offers three types of difficulty, thus ensuring accessibility to a wider range of players. Game progress is automatically saved, so there is no need to complete the game in one session. I am very satisfied with the final result, although there would be room for an improvement. For example, it is possible to add more types of enemies including boss battles and offer more abilities for the player (e.g. double jump). Although the game contains sounds, it would be possible to enrich it with music playing in the background.

As for the Godot engine, I worked comfortably with it. Its intuitive system of scenes and nodes is easy to understand and although C# is not the main programming language of the engine, its support is very good. The key methods and functions for working with nodes have the same name as in the main language of the engine, you just need to write them with notation specific to C#. When in doubt, Godot offers detailed documentation available on the web and a comprehensive community of developers ensures that it is constantly improved. Overall, Godot is a very powerful and affordable game development tool.

A Obsah přiloženého CD/DVD

`bin/`
: Všechny soubory potřebné pro běh hry. Hru lze spustit otevřením souboru *CaveEscape.exe*.

`doc/`
: Text práce ve formátu PDF a všechny soubory potřebné pro bezproblémové vygenerování dokumentu.

`src/`
: Kompletní projekt včetně zdrojových kódů hry.

Literatura

[1] BEATTIE, Andrew. How the Video Game Industry Is Changing. *Investopedia* [online]. 2020, [cit. 2021-3-31]. Dostupný z: ⟨https://bit.ly/3fBR1nk⟩.

[2] K, Dustin. Dev Teams Make Great Games. Do You Know Successful Games Created By One Brave Soul? *Game Designing* [online]. 2021, [cit. 2021-3-31]. Dostupný z: ⟨https://bit.ly/2QWxnb7⟩.

[3] WARD, Jeff. What is a Game Engine? *Game Career Guide* [online]. 2008, [cit. 2021-3-31], s. 2. Dostupný z: ⟨https://bit.ly/3fvOA5w⟩.

[4] LINIETSKY, Juan; MANZUR, Ariel (ed.). Introduction. *Godot Docs* [online]. 2021, [cit. 2021-3-31]. Dostupný z: ⟨https://bit.ly/3x4Rlkr⟩.

[5] MANZUR, Ariel; GEORGE, Marques. *Godot Engine Game Development: The Official Guide to Godot 3.0.* 1st ed. Indianapolis (USA): Mark Taub, 2018. 412 s. Sams Teach Yourself. ISBN 0-13-483509-3.

[6] LINIETSKY, Juan; MANZUR, Ariel (ed.). C# differences to GDScript. *Godot Docs* [online]. 2021, [cit. 2021-4-8]. Dostupný z: ⟨https://bit.ly/3susQtw⟩.

[7] LINIETSKY, Juan; MANZUR, Ariel (ed.). Scripting. *Godot Docs* [online]. 2021, [cit. 2021-4-8]. Dostupný z: ⟨https://bit.ly/3alJndf⟩.

[8] NYSTROM, Robert. Game Loop. *Game Programming Patterns* [online]. 2014, [cit. 2021-4-16]. Dostupný z: ⟨https://bit.ly/3gh2rNx⟩.

[9] LINIETSKY, Juan; MANZUR, Ariel (ed.). Idle and Physics Processing. *Godot Docs* [online]. 2021, [cit. 2021-4-8]. Dostupný z: ⟨https://bit.ly/2P2QAHK⟩.

[10] MICROSOFT. *Documentation for Visual Studio Code* [online]. 2021 [cit. 2021-4-17]. Dostupný z: ⟨https://bit.ly/2OZFGCw⟩.

[11] HITTI, Alejandro. Coding Imprecise Controls to Make Them Feel More Precise. [online]. 2018, [cit. 2021-4-23]. Dostupný z: ⟨https://amzn.to/3er3vf5⟩.

[12] LINIETSKY, Juan; MANZUR, Ariel (ed.). Singletons (AutoLoad). *Godot Docs* [online]. 2021, [cit. 2021-4-24]. Dostupný z: ⟨https://bit.ly/3dKzOGP⟩.