Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program: Aplikovaná informatika
Obor: Informatika

Framework Unreal Editor – jeho vlastnosti a použití pro tvorbu 2D hry

BAKALÁŘSKÁ PRÁCE

Student : Zbyněk Zelinka
Vedoucí : Ing. Rudolf Pecinovský, CSc.
Oponent : Ing. Martin Vondráček

2016

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 2. května 2016 . . . . . .

Zbyněk Zelinka

Poděkování

Děkuji Ing. Rudolfu Pecinovskému, CSc. za vstřícné jednání, cenné rady a pomoc s technickou stránkou práce.

Abstrakt

Tato bakalářská práce se zabývá popisem herního frameworku Unreal Engine 4. Nejprve je vysvětlen koncept tohoto stále populárnějšího softwarového archetypu a vybraný zástupce je zařazen do kontextu podobných alternativ. Dále je čtenář seznámen s jeho vývojovým prostředím, základními principy a programovacím jazykem Blueprint. Jednotlivé kroky jsou demonstrovány na vývoji jednoduché 2D aplikace, od založení až po export samostatné hry.

Klíčová slova

Herní framework, Unreal Engine, Sprite, Blueprint

Abstract

This bachelor thesis describes the game engine Unreal Engine 4. First, it explains the concept of this increasingly popular software archetype and the chosen representative is placed in the context of similar alternatives. The reader is then introduced to its development environment, basic principles and the Blueprint programming language. Individual steps are demonstrated in the development of a simple 2D application, from foundation to export of a standalone game.

Keywords

Game engine, Unreal Engine, Sprite, Blueprint

Obsah

1 Úvod 1
1.1 Koncepce výsledné aplikace 2
2 Rešerše 3
3 Herní frameworky 4
3.1 Unreal Engine 4
3.1.1 Historie UE 4
3.2 Unity3D 6
3.3 CryEngine 6
3.4 Game Maker 6
4 Instalace a základní operace 7
4.1 Instalace 7
4.2 Založení nového projektu 7
4.3 Rozložení editoru 8
4.4 Vytvoření nové úrovně 9
4.5 Vložení objektu do hry 9
4.6 Import obsahu 10
4.7 Objekty typu Sprite 10
5 Tvorba 2D úrovní 12
5.1 Dlaždicová sada 12
5.2 Dlaždicová mapa 13
6 Hráčova postava, vstupy a animace 15
6.1 Animace typu Flipbook 15
6.2 Mapování vstupů 16
6.3 Základy jazyka Blueprint 17
6.3.1 Komponentové okno 18
6.3.2 Konstrukční skript 19
6.3.3 Graf událostí 20
6.4 Herní režim 23
6.5 Spuštění hry 23
6.6 Přidání animací postavě 24
6.6.1 Definování proměnných 24
6.6.2 Uživatelské funkce 26
6.6.3 Výsledný graf animace 27

7 Životy a jak o ně přijít 30
7.1 Systém životů 30
7.1.1 Uživatelské události 30
7.1.2 Změny života 31
7.1.3 Odesílatel událostí 32
7.2 Statický nepřítel 33
7.2.1 Kolize a kolizní odezvy 33
7.2.2 Casting 34
7.3 Pohybliví nepřítel 34
7.3.1 Editovatelné proměnné 35
7.3.2 Časová osa 35
7.4 Smrt pádem 38
7.5 Interaktivní předměty 38

8 Podmínky vítězství a další úrovně 40

9 Zvuk 41
9.1 Zvukové efekty 41
9.2 Hudba 42

10 Uživatelská rozhraní 43
10.1 HUD 43
10.2 Nabídky 45

11 Export výsledné aplikace 48

12 Závěr 49

Terminologický slovník 50

Seznam použitých zdrojů 51

Seznam obrázků a tabulek 53

Příloha A: Další zdroje 53

1 Úvod

Počítačové hry jsou jedním z nejstarších a nejrozšířenějších typů aplikačního softwaru. První hry se objevovaly už v 50. letech minulého století, jednalo se však spíše o testovací projekty než o zábavu pro masy. [1] Od té doby se toho hodně změnilo a s rostoucími technologickými prostředky dochází k produkci stále složitějších a obsáhlejších her. Ruku v ruce s tím však rostou i požadavky na samotný vývoj, a proto velká část současných her staví na technologii herních frameworků (angl. game engine). Jejich hlavní myšlenkou je poskytnout co nejkomplexnější a nejefektivnější sadu nástrojů a možností pro tvorbu her a co nejvíc tak usnadnit vývoj jednotlivých her. Využitím takovéhoto principu se proces vývoje značně usnadňuje, jelikož již není třeba s každým projektem začínat od nuly

Tyto frameworky se tedy starají především o technickou stránku hry a koncový vývojář se tak může místo toho zaměřit např. na design, příběh nebo jiné faktory svojí hry. [2] Tyto nástroje navíc zpravidla obsahují grafická rozhraní, která mohou být, zejména pro začátečníky, přívětivější než psaní celého kódu v konvenčních nástrojích. Další z neopomenutelných předností dnešních herních frameworků je možnost vývoje pro několik platforem najednou.

Moderní herní frameworky zpravidla obsahují těchto několik základních funkcionalit:

- Grafický modul
- Fyzikální modul
- Animace
- Zvuk
- Umělou inteligenci
- Hraní po síti
- Uživatelská rozhraní
- Editor úrovní

Jelikož bych chtěl v budoucnu získat zaměstnání v herním průmyslu, je pro mě zkušenost s prací v některém z dostupných herních frameworků velmi cenná. Vybral jsem si Unreal Engine (dále UE) především na základě mých pozitivních zkušeností s hrami vyvíjenými právě na této platformě, a také se mi velice líbil přístup a vývojové prostředí programovacího jazyka Blueprint, které tento titul nabízí.

Cíle práce

- Analyzovat současnou situaci v oblasti nástrojů na podporu vývoje her a zařadit do jejich kontextu Unreal Engine 4.
- Seznámit čtenáře s prostředím UE4 a naučit ho pracovat s vybranými nástroji formou srozumitelnou i pro začátečníky.
- Podrobněji popsat jazyk Blueprint, jeho grafické rozhraní a metodiky.
- Demonstrovat použití UE4 na vývoji dvourozměrné hry od založení projektu až po export samostatné hry.

Pozn. Protože má práce výukový charakter, je v některých pasážích použita 1. osoba množného čísla.

1.1 Koncepce výsledné aplikace

Jedná o dvourozměrnou plošinovou hru pro jednoho hráče. Hry tohoto typu jsou pro svoji relativní jednoduchostí na vývoj i hratelnost často využívány jako výukové projekty. Jejím základem je postava ovládaná hráčem, která se umí pohybovat po herním světě, ve kterém na ni čeká řada překážek a nebezpečí.

Herní svět je rozdělen do několik úrovní se stupňující se obtížností. I přes ukázkovou povahu navrhovaného projektu je kladen důraz na design jednotlivých úrovní tak, aby byly přehledné, poutavé a různorodé. Hra by měla být snadno ovladatelná, nicméně natolik obtížná, aby byla výzvou i po opakovaném procházení. Ve hře se nalézá několik typů nepřátel. Každý typ je unikátní nejen po grafické stránce, ale obsahuje i unikátní funkcionalitu, zejména šablonu pohybu. Kromě hudby v pozadí má většina nepřátel unikátní zvukové efekty (např. žába kváká).

Projekt má také poměrně propracované nabídky, obsahuje vedle hlavního menu s podnabídkami také menu pro úspěšné i neúspěšné dokončení úrovně. Hra také obsahuje průhledový displej dynamicky zobrazující aktuální počet životů a stav inventáře.

K úspěšnému dokončení každé úrovně musí hráč získat klíč a použít ho k odemčení dveří nacházejících se na konci úrovně. Naopak k neúspěšnému dokončení (tj. smrti hráče) dojde, pokud hráči dojdou životy nebo pokud spadne do vody.

Pro usnadnění jsou použity veřejně dostupné grafické a zvukové prostředky s minimálními dodatečnými úpravami. Všechny jsou k dispozici online na adrese uvedené v příloze.

2 Rešerše

Publikace *Unreal Engine 4 Game Development Essentials* posloužila jako hlavní inspirace pro tuto práci. Jedná se o učebnici pro začátečníky, jejímž hlavním cílem je naučit čtenáře pracovat s UE. Zahrnuje mimo jiné i úvod do programování pomocí jazyka C++ a návody specifické pro trojrozměrné hry. Nevýhodou je nenávaznost jednotlivých kapitol, a jednotlivé praktické příklady spolu zpravidla nesouvisí, takže čtenář skončí pouze s několika rozpracovanými projekty.

Velmi podobným titulem je také publikace *Blueprints Visual Scripting for Unreal Engine*. Už z názvu vyplívá, že publikace se zaměřuje čistě na jazyk Blueprint. Jednotlivé kapitoly prakticky demonstruje na tvorbě válečné hry a čtenář si tak podle této učebnice dokáže vytvořit svoji vlastní hru, což je koncept převzatý i v této práci. Obsahem se i tak značně překrývá s výše zmíněným titulem.

Od stejného nakladatele existuje také publikace *Unreal Engine Game Development Blueprints*, která je opět zaměřena čistě na jazyk Blueprint. Jedná se o solidní titul, avšak po prostudování dvou výše zmíněných zdrojů nepřináší v podstatě nic nového. Podobně jako předchozí titul, také v této publikace na sebe kapitoly navazují a čtenář si tak vytvoří vlastní hru – tentokrát piškvorky.

Dalším zdrojem byla publikace *Master the Art of Unreal Engine*. Ve srovnání s předchozími tituly je tento o poznání méně kvalitní. Praktické příklady nejsou v podstatě vůbec vysvětleny a čtenář tedy může pouze slepě kopírovat autorův velmi specifický postup, aniž by tušil, co to vlastně dělá.

Zdaleka nejdůležitějším zdrojem informací pro mě však byla oficiální dokumentace k Unreal Engine 4. Čerpal jsem především teoretické znalosti, ale objevil jsem zde i mnoho relevantních praktických příkladů. Jak už to u dokumentací bývá, je extrémně obsáhlá, nicméně přehledně strukturovaná, včetně šikovného řazení podle obtížnosti jednotlivých témat.

S dokumentací úzce souvisí také oficiální fórum. Tam mohou uživatelé vkládat svoje tutoriály, které jsou většinou zaměřené na implementaci specifických funkcí. Fórum často slouží pouze jako rozcestník na jiné zdroje, z nichž bych rád jmenoval alespoň kanál *Muhammad A. Moniem* na serveru youtube.com. Ten se zabývá vývojem podobné hry jako tato práce. Jeho návody jsou sice o poznání delší, ale vždy srozumitelné a lze z nich dobře pochopit, co a proč právě dělá.

Unreal Engine má na serveru youtube.com také svůj vlastní kanál trefně pojmenovaný *Unreal Engine*. Lze zde najít série návodů na různé témata, včetně tvorby dvourozměrné hry pomocí jazyka Blueprint. Videa jsou profesionální a velmi informativní, nicméně ona relevantní série je velmi krátká a pokrývá pouze naprosté základy.

3 Herní frameworky

Vývojem herních frameworků se zabývá celá řada společností. Na trhu je v dnešní době nepřeberné množství kvalitních titulů, ať už komerčních nebo open-source. [9] Menší tituly jsou často specializované na určitých herní archetyp a jejich nástroje jsou potom uzpůsobeny tomuto účelu. Častější jsou však robustní a komplexní frameworky umožňující tvorbu celého spektra her. Jejich použití se ale často neomezuje pouze na tvorbu her. Zabudované grafické moduly poskytují možnosti natáčet animované filmy v reálném čase nebo vytvářet různé jiné grafické simulace. Většina dnešních předních titulů si je velmi podobná rozsahem funkcionality, výkonem i uživatelskými rozhraními svých editorů.

3.1 Unreal Engine

Prvním zástupcem je Unreal Engine od společnosti Epic Games. Tento titul nabízí kompletní sadu nástrojů pro herní vývojáře, ať už se jedná o jednoduché dvourozměrné hry nebo velké komerční trháky. Jeho sada nástrojů by neměla být překážkou pro jakýkoli typ hry a dokáže je také exportovat na většinu dnešních platforem. V současné době je framework volně ke stažení zdarma, přičemž poplatek za užívání je pouze ve formě procent z výnosů hry a to pouze v případě, že překročí určitou částku. To z něj činí zajímavou volbu pro malé vývojáře a studenty. Velkou předností tohoto titulu je možnost vytvořit celou aplikaci bez napsání jediného řádku kódu, pomocí programovacího jazyka Blueprint a jeho grafickému rozraní, což je běžné spíše u menších specializovaných frameworků. V jazyce Blueprint je možné udělat takřka vše, nicméně na určité velmi specifické věci přeci jen nestačí. V takovém případě musíme sáhnout po kódu. Poslední verze UE podporují pouze jazyk C++. [10]

3.1.1 Historie UE

Vše začalo v roce 1991, kdy americký student Tim Sweeney vytvořil svoji první hru, kterou nazval ZZT. Ta svým obsahem nijak nevynikala mezi tehdejší konkurencí, čím však zaujala, byl Timův přístup k programování. Pro tuto, i na tehdejší poměry jednoduchou hru, vytvořil vlastní jazyk, který poskytoval možnost jednoduché kontroly nad herními mechanismy. Tento jazyk implementoval do klíčových částí hry a umožnil tak hráčům upravit si tuto hru daleko větší měrou než soudobé editory. Nakonec se z tohoto editoru stala nejpopulárnější část hry, což dále vedlo k prvnímu UE. [11]

Unreal Engine 1

Nedlouho potom Tim založil společnost Epic Mega Games, které se už od počátku dařilo a mnoho z jejích raných produktů dnes patří mezi klasiku 90. let. V roce 1998 byl vydán první

Unreal Engine, který používal svůj vlastním jazyk – Unreal Script, což byl objektově orientovaný jazyk značně podobný Javě, který nabízel velkou volnost při provádění úprav. Verze UE1 byla zaměřena na tvorbu akčních her z pohledu první osoby, které v té době zažívali svůj největší boom.[11]

Unreal Engine 2

Hlavní myšlenkou pro Unreal Engine 2 bylo zahrnutí podpory pro konzole té doby. Dlouhé roky si šel vývoj pro počítače a vývoj pro konzole každý svou cestou a byly chápány jako dvě neslučitelné cesty. Unreal 2.0 to měl změnit a stát se tak prvním nástrojem pro multiplatformní vývoj. Kromě toho bylo záměrem nové verze hlavně nabídnout co nejvíc nových funkcí a možností. UE2 jich poskytoval už tolik, že umožňoval vytvořit téměř jakoukoli hru. Díky této všestrannosti se tento framework stal atraktivním pro velkou část developerů a vedl k vytvoření značné komunity, která od té doby pomáhá s dalším šířením znalostí. [11]

Unreal Engine 3

Poskytování licencí k UE2 se stalo pro společnost Epic Games hlavní formou příjmu, a proto bylo třetí vydání ještě více ambiciózní než ty dvě předchozí. Nicméně jeho komplexnost se nakonec ukázala až jako přepálená. Grafické možnosti byly vskutku revoluční, s čímž bohužel také souviselo značné zkomplikování vývoje a růst nákladů. Proto řada firem zůstala u Unreal 2.5, což byla značně upravená verze původního UE2. V Epic si však trvali na svém a snažili se propagovat nové funkce, které měli přesunout většinu kontroly nad designem od programátorů přímo na umělce. Zároveň byl UE3 optimalizován také pro mobilní telefony a jiná mobilní zařízení.

V roce 2009 byl vydán Unreal Development Kit (zkráceně UDK), což byla první, nekomerční, volně dostupná tohoto frameworku pro publikování vlastních her. [11]

Unreal Engine 4

O tři roky později byla vydána zatím poslední verze – Unreal Engine 4. Ta přinesla v první řadě celou řadu změn pro větší pohodlí při práci. Velkým překvapení byl přechod od zažitého jazyka Unreal Script na C++. Systém pro programování pomocí grafického rozhraní Kismet byl zdokonalen a aktualizován pod novým jménem Blueprint. Od března 2015 je plná verze UE4 dostupná zdarma pro veřejnost. [4]

3.2 Unity3D

Unity3D od společnosti Unity Technologies je v současné době jediným populárnějším produktem na trhu herních frameworků než UE4. Společnost uvádí, že jí v roce 2015 patřilo až 47% trhu. [12] Tyto dva nástroje jsou si v jejich aktuálních verzích do značné míry podobné a jsou si také vzájemně největšími konkurenty. Několik rozdílů ale přece jen mají. Zatímco Unreal staví striktně na C++, Unity používá především C# a JavaScript. Dalším rozhodujícím faktorem při výběru může být dostupnost. Poslední verze – Unity 5 nabízí dvě verze: měsíčně placenou *professional* a dále verzi *personal*, jenž postrádá několik prémiových funkcí a je zdarma. Co se týče výkonu, Unity5 se směle vyrovná UE4, a navíc dokáže exportovat snad na všechny běžně dostupné platformy. [9] Vývojové prostředí je také takřka srovnatelné. Co zde oproti UE4 chybí, je rozhraní pro vizuální programování.

3.3 CryEngine

Tato relativně nová platforma od německé společnosti Crytek, je určena především pro tvorbu velkých a propracovaných komerčních titulů, čemuž odpovídají i její parametry. Editor se nazývá Sandbox Editor a jeho nástroje byly přizpůsobeny primárně pro tvorbu rozsáhlých úrovní a špičkové grafiky. Začátky práce s tímto nástrojem proto můžou být poměrně obtížné, zejména protože se ještě nestačila utvořit dostatečná komunita poskytující výukové materiály a začátečníci si tak musí vystačit pouze s oficiálními zdroji. CryEngine nabízí vývoj pouze pro Windows, Linux, konzole Xbox a Play Station a futuristický Oculus Rift. Jedná se tedy o platformy, které dokážou plně využít jeho grafický potenciál. Zajímavý je model „pay what you want“ neboli „zaplať, kolik chceš“, který umožňuje získat plnou verzi tohoto softwaru i zcela zdarma. [13]

3.4 Game Maker

Dalším z populárních titulů na trhu herních frameworků je Game Maker od společnosti YoYo Games. Tento nástroj je určen primárně pro tvorbu dvourozměrných her a je oblíbený zejména pro svoji jednoduchost a přímočarý vývoj. Poskytuje grafická rozhraní, jenž většinou fungují na principu „drag and drop“ neboli „táhni a pusť". To znamená, že jednotlivé objekty a funkce se do hry jednoduše přetáhnou z nějaké nabídky. Pokud tyto nástroje nestačí, osahuje Game Maker také zabudovaný programovací jazyk GML, jehož základem je jazyk C. Podpora platforem je rozsáhlá, nicméně verze FREE je omezena na export pouze pro Windows, a ostatní verze softwaru Game Maker jsou bohužel velmi nákladné. [22]

4 Instalace a základní operace

4.1 Instalace

Před začátkem instalace Unreal Engine 4 je nejdříve třeba zkontrolovat, zdali naše hardwarová výbava splňuje alespoň tyto minimální doporučené požadavky [14]:

- Čtyř jádrový Intel nebo AMD procesor s frekvencí 2.5GHz nebo více
- 8GB RAM
- Grafická karta NVIDIA GeForce 470 GTX nebo AMD Radeon 6870 HD a lepší
- Windows 7 64-bit nebo Mac OS X 10.9.2 a novější

Plná verze Unreal Engine 4 včetně editoru je zdarma ke stažení na adrese www.unrealengine.com/dashboard. Pro přístup na tuto stránku a používání UE4 je třeba založit si na tomto webu uživatelský účet. Po registraci je ještě třeba zvolit platformu, na které budeme pracovat (v této práci je to Windows) a následně můžeme spustit stahování. V tomto kroku dojde ke stažení programu Epic Launcher, což je pouze obslužný software, sloužící k agregaci a spouštění všech ostatních produktů od této firmy. Tento program je třeba nainstalovat (ve Windows pomocí klasického průvodce instalací), spustit a přihlásit se do něj pomocí dříve vytvořeného účtu.

Samotný Unreal Engine se stahuje právě pomocí tohoto pomocného programu a pouze odtud ho lze také spustit. Stahování spustíme nepřehlédnutelným tlačítkem **Install Engine**. Tato fáze stahování je datově náročná a může trvat delší dobu.

Po provedení těchto kroků je Engine připravený pro zahájení práce. Editor spustíme pomocí tlačítka **Launch Engine**. Důležitá je také záložka **Library**, ve které se nacházejí všechny uživatelovy projekty a lze je odtud i přímo otevírat.

4.2 Založení nového projektu

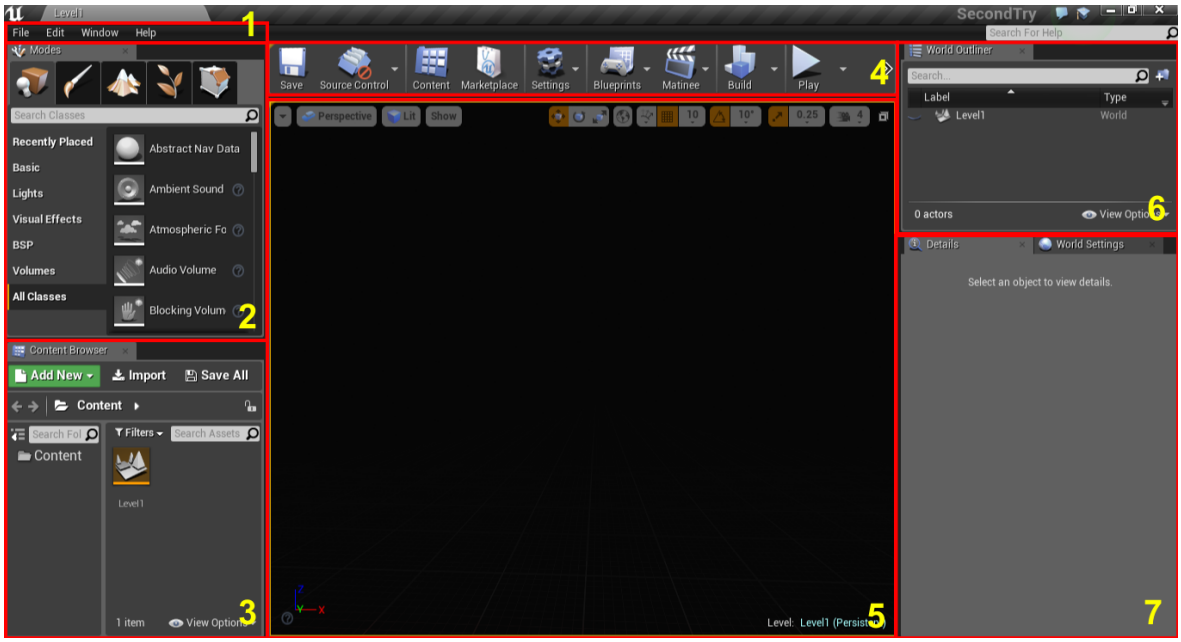
Po spuštění editoru je prvním krokem vytvoření nového projektu. Při spuštění UE se samo otevře okno, kde můžeme vybrat některý z vytvářených projektů, příp. založit nový. Při prvním spuštění se otevře nabídka s několika základními šablonami, kde každá je základem pro zcela jiný typ hry. Důležité je v horní části rozlišovat mezi záložkami Blueprint a C++. Tyto záložky definují přístup k programování použitý při tvorbě nabízených šablon. Každá z těchto šablon obsahuje již připravené základní mechanismy pro svůj typ hry. Výsledná aplikace této práce by mohla stavět na šabloně 2D Side-scroller. Pro zvládnutí základních principů práce v editoru a osvojení architektury UE4 bude však lepší začít zcela od začátku, a

proto zvolíme prázdnou (Blank) šablonu. V dolní části obrazovky je několik dalších nastavení, např. zda se jedná o klasickou hru pro PC a konzole nebo pro mobilní zařízení. Tuto i ostatní položky v této části necháme nezměněné. [9 str. 6]

Ještě níže je řádek pro zvolení úložiště na disku a nastavení názvu projektu.

Když máme tyto kroky za sebou, stiskneme tlačítko **Create Project** pro vytvoření projektu a konečně se dostaneme do samotného editoru.

4.3 Rozložení editoru



Obrázek 1
Domovské oko editoru s defaultním rozložením (Zdroj: autor)

Obrázek 1 slouží pouze k identifikaci jednotlivých částí okna editoru. Očíslované sekce reprezentují následující panely, jenž budou dále referovány podle svých českých názvů uvedených v závorkách.

1. Main Menu (hlavní menu) – obdobné jako u jiných programů.
2. Modes Menu (paleta) – slouží jako nabídka předdefinovaných objektů, které lze umisťovat přímo do herního světa.
3. Content Browser (průzkumník) – slouží pro navigaci mezi jednotlivými soubory a složkami projektu, jako jsou úrovně, třídy, importované soubory apod. Zároveň se zde vytváří všechny nové soubory a stará se i o import obsahu.
4. Toolbar (panel nástrojů) – obsahuje základní sadu nástrojů, jako uložení projektu, nastavení, nebo spuštění hry.
5. Viewport (náhled) – náhled právě otevřené úrovně, ve kterém se lze libovolně pohybovat pomocí šipek klávesnice nebo otáčet pomocí myši přidržením pravého tlačítka.

Lze sem umisťovat jednotlivé objekty a následně je manuálně transformovat – tzn. přemisťovat, otáčet nebo škálovat.

6. World Outliner (přehled úrovně) – seznam všech instancí nacházejících se v aktuální úrovni včetně jejich typu.
7. Details (detaily) – zde lze upravovat vlastnosti právě vybraného objektu, např. kolizi, nebo viditelnost. Také lze objekt transformovat vkládáním explicitních hodnot.

Rozložení editoru lze modifikovat dle vlastních potřeb pomocí škálování a přemisťování jednotlivých panelů.

4.4 Vytvoření nové úrovně

Když máme vytvořen nový projekt, je čas založit v něm novou úroveň. Úrovně jsou navazující části herního světa, kde pro postup do následující úrovně musí hráč v té aktuální splnit určitý úkol, nebo dosáhnout specifického bodu.

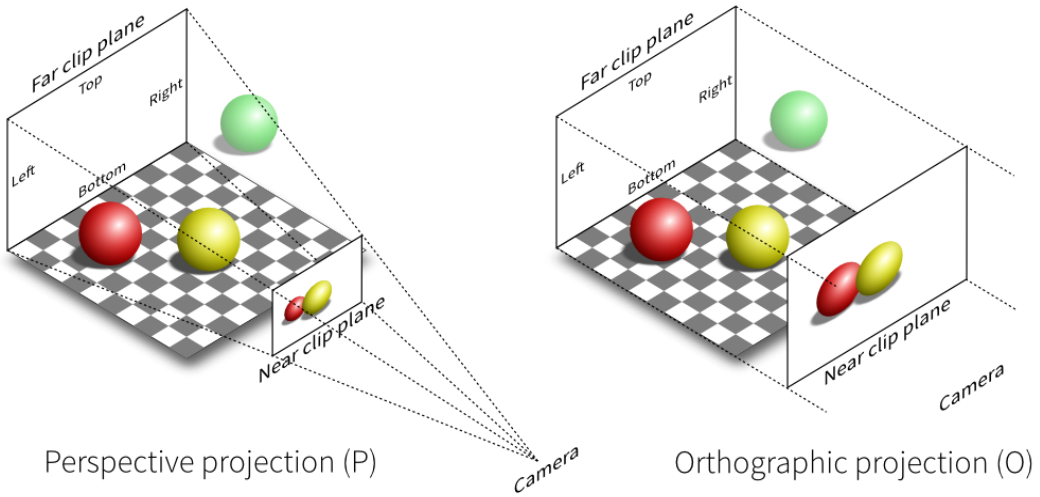
Pro vytvoření nové úrovně vybereme v hlavním menu **File** → **New Level**. V nabídce zvolíme **Empty** pro zcela holou úroveň. Tu na panelu nástrojů ihned uložíme. Dále chceme tuto úroveň nastavit jako startovní a to jak pro výslednou hru, tak pro editor, aby při jeho spuštění došlo rovnou k načtení této úrovně. V hlavním menu vybereme **Edit** → **Project Settings** a v novém okně zvolíme **Maps & Modes**. Položky Game Default Map a Editor Startup Map obě nastavíme na název námi vytvořené úrovně. [3 str. 47]

4.5 Vložení objektu do hry

Nejdříve je třeba v náhledu změnit pohled na úroveň, aby odpovídal účelům dvourozměrné hry. Příslušné nástroje se nachází v levém horním rohu náhledu. První nabídka definuje typ kamery. Vybereme **Front**, což je ortografický pohled zepředu (rozdíl mezi perspektivní a ortografickou projekcí ilustruje obrázek 2). Druhá nabídka reprezentuje zobrazovací režim, tzn. nasvícení úrovně. Zde vybereme možnost **Lit**, která představuje výsledný vzhled úrovně po aplikování všech světel. [15] Tato kombinace nám poskytne stejný pohled, z nějž bude tato úroveň zobrazována na hráčově obrazovce při hraní.

Nyní můžeme do úrovně přidat libovolný objekt. Vybereme například některý z objektů z karty Basic na panelu paleta. Vezmeme ho a přetáhneme přímo do okna náhledu. Takto se do úrovně vloží jedna instance vybraného objektu. Zároveň můžeme vidět, že se přidala také do přehledu vpravo nahoře a na kartě detailů se zobrazují její detaily. Tuto instanci lze následně transformovat. Pro tyto účely jsou určeny nástroje v pravé horní části náhledu, mezi nimiž lze libovolně přepínat. Pokud s instancí nechceme manipulovat manuálně, pouze ji označíme a na kartě s detaily jí můžeme zadat přímo specifické hodnoty těchto parametrů a instance se transformuje podle nich.

Jelikož vytváříme dvourozměrnou hru, je potřeba všechny objekty, se kterými chceme během hraní kolidovat a interagovat, tzn. všechny kromě těch zcela dekoračních, umisťovat na stejnou hodnotu osy Y. Běžně se používá nula.



Obrázek 2
Rozdíl mezi perspektivní a ortografickou projekcí (Zdroj: script-tutorials.com)

4.6 Import obsahu

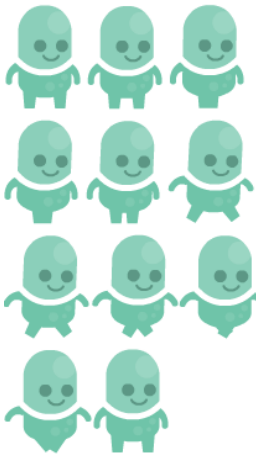
Další důležitou položkou je import externího obsahu. Pro další postup bude potřeba alespoň několik souborů, zejména grafické textury. Nejlepšími grafickými formáty pro import jsou PNG nebo PSD s transparentním pozadím a definovaným alfa kanálem. Pro získání těchto aktiv lze sáhnout do vlastních zdrojů, nebo použít stejné soubory jako v ukázkové aplikaci. Odkaz na ně je dispozici v příloze.

Pro import slouží stejnojmenné tlačítko v horní části karty průzkumníka, které otevře standartní Windows okno pro vybrání souboru. Importované soubory se zobrazují na kartě průzkumníka, kde s nimi lze dále manipulovat. Veškerý obsah je vhodné hned od začátku strukturovat. V průzkumníku se pod zeleným tlačítkem **Add New** nachází možnost **New Folder**, pro vytvoření nové složky.

4.7 Objekty typu Sprite

UE4 automaticky zahrnuje zásuvný modul Paper2D, což je systém speciálně navržený pro vývoj dvourozměrných projektů a v této práci bude využita celá řada jeho nástrojů. Základním stavebním kamenem tohoto systému je tvorba objektů typu Sprite, což jsou dvourozměrné grafické objekty, skládající se ze zdrojové textury a materiálu.

Tyto objekty lze vytvářet přímo z importovaných textur. Kliknutím pravým tlačítkem myši na texturu se otevře kontextové menu, kde vybereme **Sprite Actions** → **Create Sprite**. Výsledkem jsou vygenerované objekty typu Sprite, jenž se chovají stejně jako běžné objekty, tzn. jejich instance lze vkládat do úrovně, transformovat apod.



Obrázek 3
Sprite-sheet
(Zdroj: autor)

Jedna zdrojová textura může také obsahovat celou sadu objektů, potom ji označujeme jako tzv. Sprite-sheet a v nabídce musíme namísto **Create Sprite** zvolit **Extract Sprites**. Takto se z této sady vytvoří jednotlivé objekty typu Sprite, namísto jednoho obsahujícího celou zdrojovou texturu.

Pro editaci objektů typu Sprite obsahuje UE4 Sprite Editor, pro jeho otevření stačí dvojklik na některý objekt typu Sprite.

Sprite Editor

Editor obsahuje čtyři režimy [8]:

1. Viewport – slouží pro náhled a zobrazuje základní údaje.
2. Edit Source Region – zobrazuje celou zdrojovou texturu a umožňuje provádět úpravy zdrojové oblasti, což je využitelné především pro objekty vytvořené z textury typu Sprite-sheet.
3. Edit Collision Shapes – zobrazuje tvary zodpovědné za kolizi a umožňuje jejich editaci. Defaultně je kolizním tvarem nejmenší možný obdélník obsahující celý Sprite. Lze jej nastavit na přesnější tvar lépe kopírující tvar zdrojové textury. Přílišná přesnost však vede k větším hardwarovým nárokům výsledné hry.
4. Edit Rendering Geometry – upravuje geometrii vykreslování a umožňuje tak oříznutí grafiky tohoto objektu typu Sprite.

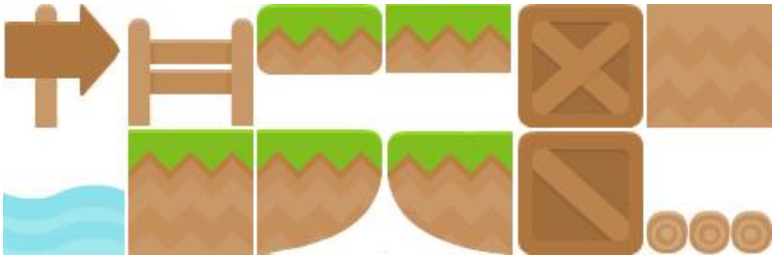
Napříč všemi těmito režimy je v pravé části karta nastavení. Za zmínku dále stojí možnost změnit materiál objektu typu Sprite, který dokáže zcela změnit jeho výsledný vzhled. Pro jednoduchost si jej lze představit jako „nátěr“ nanesený na tento objekt. Tato představa je nicméně poněkud zavádějící, jelikož materiál doslova definuje, o jaký se jedná povrch a z čeho objekt vypadá vyroben. V UE4 mají materiály svůj vlastní editor kde lze nové materiály vyrábět z textur a na ně aplikovaných matematických výrazů. [3 str. 35] Možnosti materiálů jsou nicméně zaměřeny především na trojrozměrné objekty interagující se světly. Účelům námi vyvíjené aplikace zcela vyhovuje materiál, který mají všechny objekty typu Sprite nastaven jako defaultní.

Instance objektů typu Sprite lze libovolně přidávat do úrovně a budou také dále používány, např. jako zdrojové snímky pro tvorbu animací.

5 Tvorba 2D úrovní

Prvním krokem na cestě k funkční hře bude vytvoření zajímavého prostředí, po kterém se bude hráč v průběhu hry pohybovat. Pro tyto účely obsahuje UE4 dlaždicový systém, který je součástí již zmíněného modulu Paper2D. Tento systém poskytuje rychlé a intuitivní nástroje pro tvorbu základních struktur dvourozměrného herního světa.

Nejprve budeme potřebovat importovat zdrojové textury. Pro dlaždicový systém je opravdu vhodné, aby byly ve formě Sprite-sheet. Ten musí mít navíc specifický formát, jelikož bude v dalším kroku „rozřezán“ podle pravidelné mřížky na jednotlivé dlaždice. Zpravidla by tyto dlaždice měli být čtvercové, což sice není podmínkou, avšak práce s jinými formáty je poměrně nešikovná. Mezi dlaždicemi může i nemusí být mezera. Samozřejmostí je transparentní pozadí.



Obrázek 4

Příklad vhodné zdrojové textury (Zdroj: opengameart.org)

5.1 Dlaždicová sada

Když máme vhodnou texturu, vytvoříme nový soubor, který bude sloužit jako zásobník dlaždic. Najdeme ho v průzkumníku pod tlačítkem **Add New** → **Paper2D** → **Tile Set**. Soubor pojmenujeme a otevřeme. Otevře se nám nové okno, kde můžeme tuto dlaždicovou sadu dále definovat. Nejprve do ní vložíme zdrojovou texturu. Vpravo dole se nachází panel Tile Set, na němž se potom nachází rozbalovací nabídka **Tile Sheet Texture**. V ní najdeme a zvolíme příslušnou importovanou texturu, která se ihned zobrazí v náhledu.

Nyní musíme specifikovat, jak má být tato textura rozdělena na jednotlivé dlaždice. Na kartě Tile Set tedy nastavíme hodnoty:

- Tile Size – jednotné rozměry všech dlaždic ze sady.
- Per-Tile Spacing – jednotná velikost mezery mezi dlaždicemi. [15]

Že jsme hodnoty nastavily správně, si můžeme ověřit tak, že v náhledu klikneme na dlaždici úplně vpravo dole. Tato dlaždice se ihned zobrazí na kartě Single Tile Editor vpravo. Pokud se zobrazuje správně, hodnoty jsou v pořádku. V opačném případě je budeme muset ještě zkalibrovat.

Dalším krokem je nastavit dlaždicím kolizní tvary. Tuto proceduru musíme provést pro každou dlaždici zvlášť. V náhledu tedy nejprve zvolíme jednu z dlaždic, pro kterou chceme nastavit kolizi. Z horní nabídky nástrojů potom vybereme jeden z nástrojů **Add Box/Polygon/Circle** podle toho, který tvar nejlépe vystihuje grafiku této dlaždice. Tento tvar se nám automaticky přidá přes dlaždici na kartě Single Tile Editor, kde ho můžeme dále upravovat. [15] Vybráním hrany nebo vrcholu tvaru s nimi můžeme pohybovat. Navíc Stisknutím SHIFT + LEVÉHO TLAČÍTKA MYŠI kamkoli na hranu tvaru se na tomto místě vytvoří další vrchol kolizního tvaru. [20] Tvar upravíme tak, aby zhruba odpovídal obrázku na dlaždici. Tvary nemusí být naprosto přesné, jelikož velké množství hran má negativní vliv na výkon. Mějme na paměti, že výsledná kolize bude odpovídat tomuto tvaru nehledě na grafiku dané dlaždice.

Pro dlaždice, které budou plnit čistě estetickou funkci (např. směrovka na obrázku 4) samozřejmě kolizní tvary definovat nemusíme, jelikož nepotřebujeme, aby jakkoli interagovali s hráčem.

Na horním panelu nástrojů se nachází šikovné tlačítko **Show Colliding Tiles**, které nám v náhledu vyznačí všechny dlaždice s již nastavenou kolizí. To je užitečné zejména při práci s většími zdrojovými texturami.

Když máme nastavené kolizní tvary podle našich představ, uložíme sadu a můžeme přistoupit k dalšímu kroku nebo si z jiné zdrojové textury vytvořit další sadu, protože dále budeme mít možnost kombinovat dlaždice z více sad. Všechny tyto sady by však měli mít stejné rozměry jednotlivých dlaždic.

5.2 Dlaždicová mapa

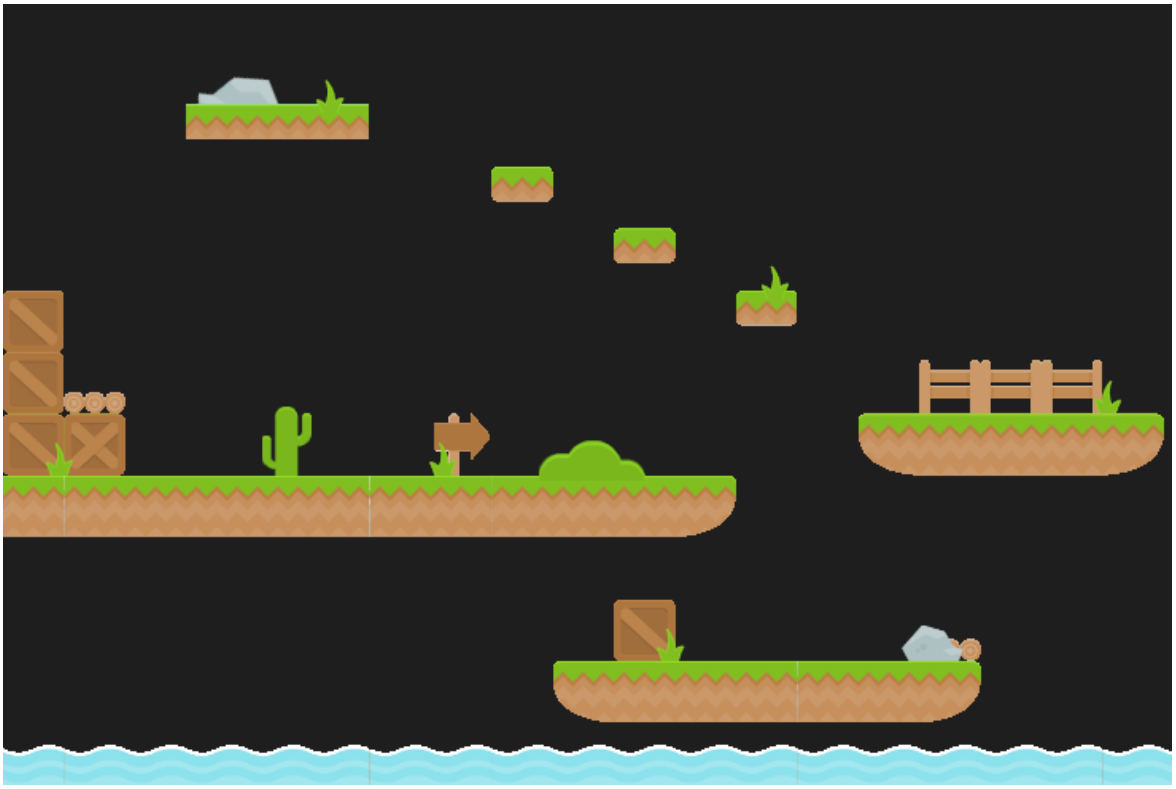
Máme vytvořený zásobník dlaždic a nyní ho můžeme využít pro výstavbu úrovně. Vytvoříme další soubor, tentokrát však typu Tile Map. V průzkumníku vybereme **Add New** → **Paper2D** → **Tile Set** a nový soubor otevřeme. V tomto okně budeme vytvářet základní architekturu úrovně. Uprostřed obrazovky se nachází malá mřížka, do níž budeme vkládat jednotlivé dlaždice. Nejdříve však musíme tuto mřížku upravit. V pravé časti okna je opět karta detailů. Na ní se nejprve zaměříme na část Setup. Zde musíme nastavit hodnoty Tile Width (šířka buňky mřížky) a Tile Height (výška buňky mřížky), aby odpovídaly velikosti dlaždic z naší sady (příp. sad). Dále také chceme navýšit celkový počet buněk. Nastavíme tedy hodnoty Map Width (šířka mapy) alespoň na 50 a Map Height (výška mapy) alespoň na 20.

Když máme mřížku připravenou, je čas ji naplnit. V levé části okna se nachází karta pro práci s dlaždicovou sadou. Nejprve zvolíme tlačítko se symbolem čtverečku na řádku Active Tile Set a z nabídky vybereme naši dlaždicovou sadu. Ta se ihned načte do levé sekce obrazovky. Nyní můžeme začít vkládat dlaždice ze sady do mapy. V horní části vybereme štětec a následně ze sady vybereme jednu z dlaždic. Kliknutím levým tlačítkem myši na jednu z buněk mřížky se do ní vloží vybraná dlaždice. Navíc pokud máme uspořádanou sadu, můžeme z ní dlaždice vkládat i po větších celcích.

Na horním panelu nástrojů je k dispozici mimo štětce také guma a kyblík. Guma funguje opačně než štětec, tzn., maže dlaždice. Kyblík pro změnu vyplní všechny prázdné buňky mřížky aktuálně vybranou dlaždicí. Vlevo od těchto nástrojů jsou ještě možnosti pro rotování a zrcadlové převracení dlaždic. Navíc pokud máme více dlaždicových sad, můžeme mezi nimi libovolně přecházet pomocí tlačítka se symbolem čtverečku a kombinovat dlaždice z obou sad do této jedné mapy.

Dlaždice lze přidávat také do vrstev. Nahoře na kartě detailů je sekce Tile Map, která obsahuje nástroje pro práci s vrstvami. Stačí vytvořit novou vrstvu pomocí tlačítka „+“. Nová vrstva se vytvoří vždy přes tu stávající, nicméně pomocí tlačítek se symboly šipek lze jejich pořadí měnit.

Když jsme spokojení s výsledkem, uložíme ho a okno zavřeme. Instanci vytvořené mapy nyní potřebujeme vložit do úrovně. Soubor s mapou najdeme v průzkumníku a přetáhneme rovnou do náhledu. Ujistíme se, že Y souřadnice je rovna nule. Jednu úroveň můžeme také složit z libovolného počtu stejných nebo různých dlaždicových map. Dále můžeme vylepšit vzhled úrovně přidáváním samostatných objektů. Stačí je naimportovat a vytvořit z nich objekty typu Sprite. Ty pak lze v libovolném množství vkládat do úrovně. Tyto předměty mají defaultně nastavenou kolizi, a proto pokud jsou čistě dekorační (např. kaktus na obrázku 5), posuneme je na ose Y o několik jednotek dopředu nebo dozadu. Naopak pokud chceme, aby kolidovali s hráčem, musí být jejich Y souřadnice rovny nule (např. kámen).



Obrázek 5

Část úrovně vytvořené pomocí dlaždic s přidanými předměty (Zdroj: autor)

6 Hráčova postava, vstupy a animace

Cílem v této kapitoly bude vytvořit ve hře grafickou reprezentaci hráče. Ta bude reagovat na vstupy z klávesnice, bude se umět pohybovat, tzn. běhat a skákat a bude na tyto změny adekvátně reagovat změnou své animace.

6.1 Animace typu Flipbook

Prvním krokem bude předpřipravení jednotlivých animací. Opět využijeme nástroje modulu Paper 2D, konkrétně nástroj pro tvorbu animací typu Flipbook. [15] Tento nástroj je speciálně určen pro práci s dvourozměrnými animacemi a díky tomu je mnohem jednodušší než většina systémů pro práci s animacemi trojrozměrnými. Lze si jej představit jako sérii po sobě jdoucích obrázků (zde objektů typu Sprite), kde každý další snímek je oproti svému předchůdci nepatrně pozměněn a rychlým průchodem touto sérií je vytvořena iluze pohybu.

Naše postava bude přecházet mezi třemi animačními stavy – stání (nečinnost), běh a skok. Pro každý tento stav bude potřeba vytvořit unikátní animaci. Jako první vytvoříme animaci běhu. Nejdříve budeme potřebovat vhodné objekty typu Sprite, aby výsledek vypadal, že postava opravdu běží. Importujeme si tedy adekvátní zdrojové textury a vytvoříme z nich objekty typu Sprite. Nový Flipbook soubor vytvoříme pomocí již známého tlačítka **Add New** na kartě průzkumníka, konkrétně v kategorii **Animations** jako **Paper Flipbook**.

Nový soubor můžeme otevřít a začít do něj vkládat jednotlivé snímky. V pravé části nového okna se opět nalézá karta detailů. Na ní je nahoře položka Frame Rate určující počet snímků za sekundu. Změna této hodnoty povede k zrychlení/zpomalení běhu animace. Hned pod ní je důležitá položka Key Frames (klíčové snímky), která zde slouží pro většinu práce. Můžeme u ní vidět, že aktuálně tento Flipbook neobsahuje žádný snímek. Pro vytvoření nového zvolíme symbol „+“. Do nové nabídky následně vložíme první objekt typu Sprite sekvence běhu. Výsledek se nám okamžitě zobrazí v náhledu. V dolní části okna se navíc nachází časová osa s ovládáním, kde můžeme běh animace zastavovat, obracet apod. Takto pokračujeme i pro další snímky dokud nevyčerpáme všechny objekty typu Sprite z této sekvence a animace není plynulá. [8]

Pro objekty typu Flipbook je také nezbytné definovat kolizi – buď žádnou, statickou (tj. kolizi prvního snímku po celou dobu běhu animace), nebo vlastní pro každý snímek. Nejčastěji používanou je kolize statická, neboť dynamická je i při běžné rychlosti střídání snímků velmi náročná na výkon. Pro její nastavení najdeme ve spodní části karty detailů položku **Collision Source** a vybereme možnost **First Frame Collision**.

Obdobně postupujeme také pro animaci skoku a stání. Pokud bychom chtěli některý stav reprezentovat pouze staticky a bez animace (typicky např. stání), je i tak s ohledem na další práci výhodnější vytvořit objekt typu Flipbook, i když bude obsahovat pouze jeden snímek.

6.2 Mapování vstupů

Dalším krokem než začneme se samotným programováním, bude nastavení mapování vstupů. Tato funkce slouží pro komunikaci mezi hardwarovými vstupy (stisk tlačítka klávesnice, pohyb myši apod.) a vytvářeným programem. [15] Existují dva typy mapování:

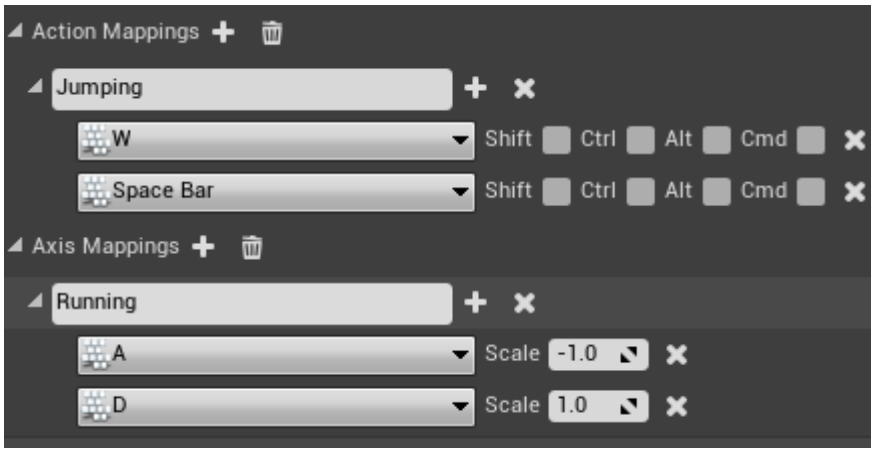
1. Mapování akcí – tato funkce slouží pro jednotlivé oddělené vstupy (jeden stisk tlačítka). Takovýto vstup vždy povede právě k jednomu zavolání nějaké herní funkce. Vhodné pro např. skok, otevření truhly apod.
2. Mapování os – tato funkce naopak slouží pro dlouhodobý vstup (držení tlačítka), kde komunikace s navazující herní logikou probíhá po celou dobu běhu hry (i pokud je přiřazený vstup neaktivní). Vhodné např. pro běh.

Pro naši postavu potřebujeme nastavit celkem 3 vstupy – skok, pohyb vpravo a pohyb vlevo. V hlavní nabídce navigujeme **Edit** → **Project Settings** a v novém okně vybereme **Input**.

Nejprve vytvoříme mapování pro skok. Jelikož se jedná o mapování akcí, zvolíme „+“ v části Action Mappings. Pojmenujeme a za okénkem pro jméno zvolíme opět „+“ pro navázání nového hardwarového vstupu. V rozbalovacím menu vybereme Keyboard a následně klávesu, kterou chceme využít pro skok, např. W. Na jedno mapování můžeme navázat více hardwarových vstupů se stejnou funkcí, např. pokud pro skok chceme použít kromě klávesy W také mezerník. Zvolíme tedy opět symbol „+“ a v menu najdeme Spacebar.

Dále potřebujeme vytvořit mapování pro pohyb. Postup je obdobný jako v předchozím případě. Protože se ale jedná o mapování os, zvolíme „+“ v sekci Axis Mappings. Pojmenujeme nové mapovaní a vytvoříme dva vstupy. Jeden bude sloužit pro pohyb doprava a jeden pro pohyb doleva. Každému vstupu přiřadíme jednu klávesu, např. A a D. Jelikož chceme, aby jedna z kláves reprezentovala pohyb v opačném směru, nastavíme její položku Scale (tj. váhu pohybu) na hodnotu -1.

Nyní máme vytvořené mapování vstupů a můžeme toto okno zavřít. Výsledek by měl vypadat jako na obrázku 6.



Obrázek 6
Mapování vstupů (Zdroj: autor)

6.3 Základy jazyka Blueprint

Blueprint je objektově orientovaný programovací jazyk určený primárně pro programování herních mechanismů. Od většiny podobných jazyků se liší, jelikož práce v něm nespočívá v psaní řádků kódu, ale místo toho nabízí propracovaný systém grafických prostředků. Ten je navržený tak, aby byl schopen poskytnout designérům celý rozsah funkcionality, která by jinak byla dostupná pouze pokročilým programátorům. Pomocí Blueprint lze implementovat například:

- Herní pravidla
- Postavy
- Kamery
- Vstupy
- Předměty
- Prostředí

Pomocí propojování uzlů, událostí, funkcí a proměnných pomocí spojů lze vytvořit i složité herní mechanismy bez napsání jediného řádku kódu. Blueprint používá při práci grafy s uzly a to hned pro několik účelů – konstruování objektů, samostatné funkce, herní události apod. Po osvojení základních principů jazyka Blueprint a získání dostatečné orientace v editoru je práce velmi rychlá a zábavná. Navíc jsou řádně uspořádané grafy na první pohled lépe vypovídající a zejména mnohem přehlednější než i dobře strukturovaný kód. Jazyk Blueprint byl vyvinut společně s UE4 a nikde jinde se s ním tudíž nelze setkat. Existuje však řada podobných jazyků používajících podobné principy. Jazyk Blueprint je následníkem jazyka Kismet, který plnil podobnou funkci v předcházejících verzích UE. [11]

Pro hráče chceme nejprve vytvořit novou třídu. Třída je objektem obsahujícím určité funkce a proměnné. Sama o sobě nelze vložit do hry, nýbrž z ní lze vytvořit jednotlivé instance. Tyto instance přebírají funkce své třídy a její proměnné nabývají specifických hodnot pro každou z nich. [15]

Pro vytvoření nové třídy navigujeme v průzkumníku **Add New** → **Blueprints** → **Blueprint Class**. Otevře se nové okno s výběrem tzv. rodičovských tříd. [3 str. 140] Každá z nich má specifickou sadu vlastností a funkcí, od těch nejjednodušších až po značně vybavené třídy. Vybrání jedné rodičovské třídy umožňuje zdědění těchto specifik, ke kterým lze následně přidávat další funkcionalitu. Správný výběr rodičovské třídy nám tudíž může značně ulehčit práci. Třídy, které v tuto chvíli vidíme v nabídce, jsou základními třídami frameworku vytvořenými v C++, každá uživatelsky vytvořená třída může však být rodičovskou třídou pro další tvořené třídy.

Nejpoužívanějšími rodičovskými třídami jsou:

- Actor (herec) – nejjednodušší, prázdná třída. Reprezentuje libovolný objekt, který lze přidat do herních světa.
- Pawn (pěšák) – typ herce s rozšířenou funkcionalitou, jenž může být ovládán ovladačem, přijímat z něj vstupy a reagovat na ně.

- Character (charakter) – typ pěšáka s rozšířenou funkcionalitou. Má např. schopnost pohybu a předdefinovaný kolizní tvar.
- Game Mode (herní režim) – určuje typ hry definováním jejích základních pravidel.

Pro potřeby hráčovy postavy se nám nejvíce hodí třída charakter. Modul Paper 2D však obsahuje ještě jednu velmi podobnou třídu upravenou přesně pro potřeby dvourozměrné postavy. Ve spodní části okna tedy vybereme **All classes** pro zobrazení všech tříd a do hledání napíšeme „Paper Character". Tuto možnost vybereme, pojmenujeme nový soubor a otevřeme.

Nyní se dostáváme do hlavního okna pro práci v jazyce Blueprint. Toto okno a práce v Blueprint se skládá z tří složek, přičemž každá z nich je určená pro jinou oblast vývoje. Jsou to:

- Components Window (komponentové okno)
- Construction Script (konstrukční skript)
- Event Graph (graf událostí)

6.3.1 Komponentové okno

Prvním oknem je komponentové okno (jeho záložka se jmenuje Viewport). Komponentou se zde rozumí určitá část funkcionality, kterou lze připnout k již existujícím komponentám této třídy. [15] [3 str. 118] Všechny komponenty třídy se nalézají vpravo na panelu Components. Lze si všimnout, že třída již obsahuje následující komponenty zděděné ze své rodičovské třídy:

- Capsule Component – kapsle, která má na starosti kolizi. Tato komponenta činí všechny ostatní kolizní tvary tohoto herce irrelevantní.
- Arrow Component – šipka pro orientaci jakým směrem je postava otočená čelem.
- Sprite Component – obsahuje grafickou reprezentaci postavy. Defaultně je prázdná a musí se naplnit.
- Character Movement Component – tato komponenta umožňuje postavě pohyb a obsahuje také vyčerpávající množství nastavení pohybu.

Komponentové okno slouží pro přidávání a editaci těchto komponent. Jeho rozhraní značně připomíná nástroje pro modelování objektů. Napříč všemi třemi okny je vpravo opět karta s detaily, která vždy ukazuje pouze podrobnosti právě vybrané komponenty.

Prvním krokem je obsazení zděděné Sprite komponenty. Z jejího názvu by se mohlo zdát, že pro tyto účely použijeme objekt typu Sprite, ale není tomu tak. Místo toho použijeme rovnou soubor typu Flipbook a postava tak bude již od počátku animovaná. Na kartě detailů v sekci Sprite zvolíme jako zdrojovou Flipbook animaci tu, kterou jsme nedávno vytvořili pro stav nečinnosti. Změna se ihned projeví a v kapsli by se měla objevit postava. Důležité

je, aby postava stála směrem, kterým ukazuje šipka, pokud tomu tak není, musíme Sprite komponentu otočit.

Dalším krokem je přizpůsobení velikosti kapsle k postavě. Označíme kapsli a na kartě detailů v sekci Shape kalibrujeme hodnoty, dokud není kapsle stejně velká jako postava uvnitř.

Pro komponentu Character Movement můžeme nastavovat velké množství parametrů pro pohyb, jako maximální výšku skoku, rychlost běhu apod. Tyto hodnoty se lépe kalibrují až po prvním spuštění hry, kdy víme, jak se postava chová ve vytvořeném herním světě. Prozatím potřebujeme postavě zakázat pohyb po ose Y, jelikož náš svět je pouze dvourozměrný. Najdeme tedy sekci Planar Movement, kde zaškrtneme položku **Constraint to Plane** a u položky **Plane Constraint Normal** nastavíme pro Y hodnotu -1.

Kamera

Dalším krokem je připevnění kamery k postavě. Kamera představuje bod, ze kterého hráč uvidí hru a herní svět na své obrazovce. Jelikož chceme, aby tento bod po celou dobu hraní následoval hráčovu postavu a to v konstantní vzdálenosti, připevníme k postavě pevné neviditelné rameno a na něj následně umístíme samotnou kameru.

Přidáme tedy novou komponentu. Na komponentové kartě vybereme zelené tlačítko **Add Component** a z nabídky zvolíme **Spring Arm**, což bude naše fixní rameno. V detailech provedeme několik nastavení. Důležitou položkou je zakázání kolize pro toto rameno. Toho docílíme odškrtnutím položky **Do Collision Test**. Dále potřebujeme v náhledu toto rameno otočit o 90° aby směřovalo kolmo na postavu. (pozn. Délka ramena nebude mít vliv na výslednou vzdálenost kamery.)

Nyní můžeme na rameno umístit samotnou kameru. Vytvoříme tedy opět novou komponentu, tentokrát typu **Camera**. Najdeme ji na kartě Components, vezmeme a přetáhneme na rameno, čímž se stane jeho následníkem a automaticky se umístí na jeho konec. Kamery mají dvě možnosti projekce, ortografickou a perspektivní. Pro naše účely se lépe hodí ortografická, a proto ji v detailech nastavíme v kolonce **Project Mode**. Hned o řádek níž se nachází možnost **Orto Width**. Ta slouží pro šířku záběru kamery, tzn. vlastně vzdálenost kamery od charakteru. Dále bude kameru třeba opět otočit o 90°, tentokrát však opačným směrem, aby směřovala přímo na postavu.

6.3.2 Konstrukční skript

Konstrukční skript slouží pro definování metod, které mají proběhnout při vytvoření instance této třídy. Obsahuje maximálně jeden uzlový graf, na jehož začátku se vždy nachází stejnojmenný uzel. Na něj potom navazuje zbytek grafu. Jeho rozhraní je takřka identické s následujícím grafem událostí a používá i stejných principů pro práci.

Toto okno není v této práci používáno.

6.3.3 Graf událostí

Graf událostí je srdcem každé třídy. Obsahuje uzlové grafy používající události a volání funkcí pro provádění akcí jako reakcí na herní události týkající se instancí této třídy. Právě zde lze implementovat interaktivitu a dynamické odezvy. Název graf událostí může být poněkud zavádějící, jelikož se zpravidla skládá z mnoha samostatných grafů.

Každý graf se skládá z uzlů a spojů. Uzly reprezentují vždy nějakou funkci nebo událost. Mohou mít vstupy, výstupy nebo obojí. Vstupy jsou vždy na levé straně uzlu a výstupy na pravé. Spoje vedou vždy od výstupu jednoho uzlu do vstupu druhého. Uzly i spoje jsou barevně označené podle svého typu. [3 str. 137] Jednotlivé typy uzlů se nachází v tabulce 1. Spojů existují dva typy – bílé označují postup průchodu grafem, zatímco spoje ostatních barev reprezentují datové toky. Jejich barva dále závisí na datovém typu (více v části 6.6.1 Definování proměnných).

Tabulka 1: *Typy uzlů*

Barva hlavičky	Typ uzlu
Červená	Uzel s událostí. Počáteční bod každého grafu.
Modrá	Tento uzel může být buď funkcí, nebo voláním události. Ikona na hlavičce se mění podle toho, o jaký z těchto dvou typů se jedná.
Fialová	Dostupný pouze v uživatelských funkcích, kde slouží jako vstup nebo výstup. Nelze jej manuálně vytvořit ani smazat.
Šedá	Uzel řízení procházení grafem nebo makro.
Zelená	Tento uzel představuje tzv. čistou funkci, což znamená, že pouze vrací nějakou hodnotu. Je napojen pouze datově a je automaticky iniciován, pokud jsou jeho data potřeba. [15]
Tyrkysová	Speciální uzel, který slouží pro komunikaci mezi třídami (více v kapitole 7.2.2 Komunikace mezi třídami, Casting).
Zlatá	Uzel Timeline generující křivky (více v kapitole 7.3.2 Časová osa).

Graf skoku

Nyní si zkusíme vytvořit co nejjednodušší graf. Výborně se na to hodí graf pro skok, jelikož většina funkcionality je již obsažena v komponentě Character Movement.

Nový uzel vytvoříme kliknutím pravým tlačítkem myši kamkoli v centrální sekci okna. Otevře se nám kontextové menu, kde můžeme najít všechny dostupné uzly zařazené v několika kategoriích. Uzlů je na výběr opravdu veliké množství, proto je lepší hledat je podle názvu. V horní části kontextového menu je textové pole, pomocí kterého můžeme uzly filtrovat.

Stejně jako všechny grafy musí i graf skoku začínat nějakou událostí. V tomto případě to bude událost navazující na mapování akcí. Napíšeme tedy název našeho mapování pro skok a vybereme, čímž dojde k vytvoření uzlu události. Můžeme si všimnout, že tento uzel má dva jeden bílé (exekuční) výstupy, horní reprezentuje událost stisk tlačítka definovaného v tomto mapování, spodní naopak při jeho uvolnění. My použijeme horní výstup označený Pressed. Najedeme na něj myší tak, aby se kurzor změnil na křížek, a tahem vytvoříme nový spoj. Automaticky se otevře nabídka pro výběr nového uzlu, do které napíšeme „Jump" a vybereme odpovídající možnost. [8]

Tím je graf skoku hotový. Říká, že pokud dojde k stisknutí některého z mapovaných tlačítek, zavolá funkci Jump, což je již předdefinovaná funkce komponenty Character Movement zajištují skok.

Graf pohybu

Graf pro pohyb už bude o poznání složitější, proto je postup uveden v jednotlivých krocích:

1. Začátek bude stejný. Stiskneme pravé tlačítko, do nabídky napíšeme název mapování pro běh a vybereme. Vytvoří se opět uzel s novou událostí.
2. Vezmeme jediný exekuční výstup a vytvoříme nový uzel. Z nabídky tentokrát vybereme Add Movement Input, což je funkce vytvářející instrukce pro pohyb pro komponentu Character Movement.
3. V uzlu Add Movement Input nastavíme položku World Direction pro X na hodnotu 1. Tzn., charakter se bude pohybovat po ose X.
4. Vrátíme se k uzlu s událostí a její druhý výstup (zelený) zapojíme do vstupu Scale uzlu Add Movement Input. Tento spoj bude zprostředkovávat přenos dat ze vstupu, které budou využity v uzlu Add Movement Input jako váhy pohybu. Vzpomeňte, že při mapování jsme nastavily hodnotu Scale pro klávesu A na -1 a pro klávesu D na 1. Když uzel obdrží hodnotu 1, posune postavu směrem popsaným v položce World Direction (v našem případě doprava po ose X), pokud však dostane -1, posune ji opačným směrem.

V této fázi je již postava schopná pohybu. Nicméně dále chceme, aby byla vždy otočená směrem, kterým se právě pohybuje.

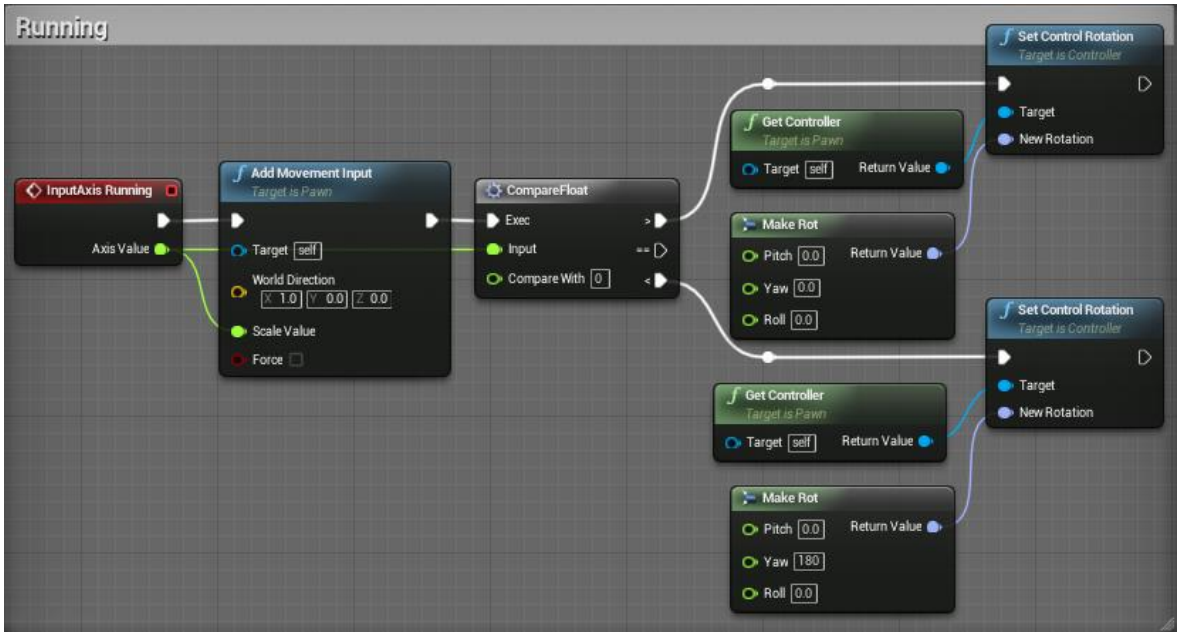
5. Vezmeme výstup funkce Add Movement Input a vytvoříme nový uzel Compare Float. Tento uzel porovnává data typu float (tj. reálná čísla) z obou svých vstupů a následně větví graf podle toho, který vstup je větší.
6. Znovu se vrátíme k uzlu s událostí a jeho zelený výstup zapojíme do vstupu Input uzlu Compare Float. Druhý vstup tohoto uzlu nastavíme ručně na hodnotu nula. Uzel nyní sleduje Scale data ze vstupu a porovnává je s nulou. My už víme, že pokud je hodnota Scale menší než nula, charakter se pohybuje doleva a naopak pokud je větší, pohybuje se doprava.

Pozn. Můžeme vidět, že pro datové toky platí, že z jednoho výstupu můžeme vyvést více spojů. Pro exekuční řetězec tomu tak není, pro jeho větvení existují speciální uzly.

7. Nyní vytvoříme nový nezávislý uzel Get Controller a na jeho výstup navážeme uzel Set Control Rotation. Controller neboli ovladač je třída zodpovědná za ovládání postavy a zprostředkovávání vstupů. [15]
8. Vytvoříme nový uzel Make Rot, který slouží pro vytvoření nové rotace. Hodnoty ponecháme a jeho výstup připojíme do vstupu New Rotation uzlu Set Control Rotation.
9. Exekuční výstup „>“ z Uzlu Compare Float zapojíme do vstupu uzlu Set Control Rotation.
10. Opakujeme kroky 7 a 8, s tím rozdílem, že při vytváření nové rotace nastavíme hodnotu Yaw na 180.
11. V posledním kroku připojíme tuto novou větev k výstupu „<„ uzlu Compare Float.

Nyní pokud bude v mapování hodnota Scale menší než nula, postava bude otočená o 180°.

Pozn. Dvojklikem na spoj dojde k vytvoření kotevního bodu, který lze využít k přehlednějšímu formátování. Dále existuje také možnost označit celý graf a pomocí klávesy C z něj vytvořit skupinu, kterou lze i okomentovat. [8]



Obrázek 7

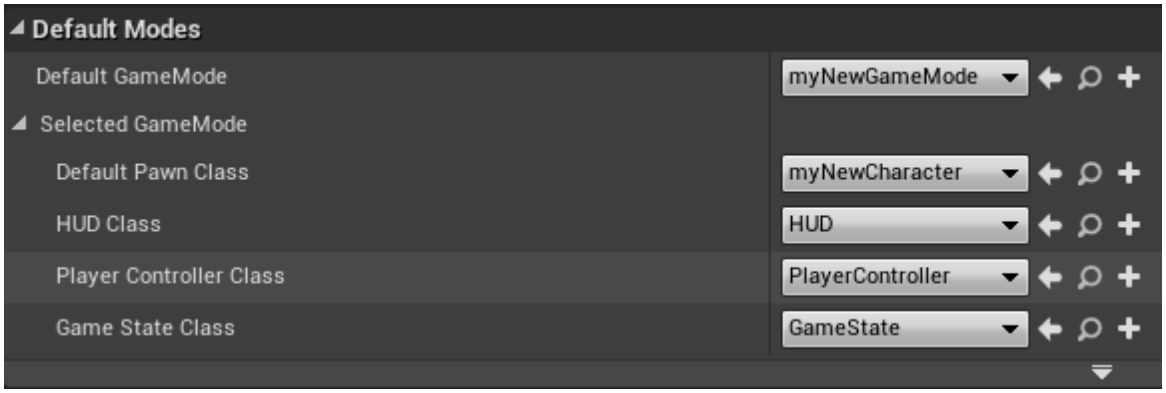
Dokončený graf pohybu (Zdroj: autor)

Tímto jsme prozatím v Blueprint hotoví. Stiskneme proto tlačítko Compile, čímž dojde ke kontrole vytvořených grafů a následně převedení celé třídy do jazyka, jemuž bude UE rozumět – konkrétně je to Unreal Script VM bajt kód, což je jazyk virtuálního stroje používaného již od verze UE 2.0. [15]

6.4 Herní režim

Před prvním spuštěním hry, je ještě třeba nastavit tzv. herní režim (angl. Game Mode). Herní režim je třídou stanovující základní herní pravidla a mechanismy. Každá nová funkce nebo proměnná zodpovědná za herní pravidla by měla být definována v rámci této třídy. Jedna hra může mít více herních režimů, nicméně aktivní může být vždy pouze jeden. Tato třída je inicializována při startu každé úrovně a po celou dobu hraní dohlíží na herní pravidla.

V hlavním menu vybereme **Edit** → **Project Settings**. V novém okně vybereme **Maps & Modes**. V sekci Default Mode vidíme, že hra již má definovaný herní režim, ten nám však nevyhovuje, vytvoříme si tedy novou třídu herního režimu. Zvolíme symbol „+“, vybereme umístění, pojmenujeme a vytvoříme. Položku **Default Game Mode** nyní změníme za právě vytvořenou třídu. Následně rozbalíme nabídku **Selected Game Mode** o řádek níže. Tady lze nastavit naprosto základní pravidla hry. My chceme, aby charakter, který jsme právě vytvořili, byl při startu hry přiřazen hráči a hráč tak nad ním získal kontrolu. Do kolonky **Default Pawn Class** proto dosadíme název hráčovy třídy. Ostatní hodnoty ponecháme.



Obrázek 8

Nastavení herního režimu (Zdroj: autor)

Posledním krokem je přepsat defaultní herní režim také v položce World Settings, jenž má na starosti základní atributy herního světa, které mohou být tímto režimem také ovlivněny. V hlavním okně editoru zvolíme na panelu nástrojů **Settings** → **World Settings** (pozn. toto okno se otevírá jako další karta přes panel detailů). Na této kartě jednoduše nastavíme hodnotu položky **Game Mode Override** na nově vytvořený herní režim. Navzdory názvu World Settings je toto nastavení třeba provést v každé z budoucích úrovní.

6.5 Spuštění hry

Nyní je konečně čas poprvé spustit samotnou hru. Než tak učiníme, musíme do úrovně vložit objekt Player Start, který najdeme na paletě v záložce Basic. Player Start vložíme do úrovně a musíme opět zkontrolovat, že jeho souřadnice na ose Y jsou rovny nule. Tento objekt představuje místo, kde se objeví hráčova postava při spuštění této úrovně. A zároveň má

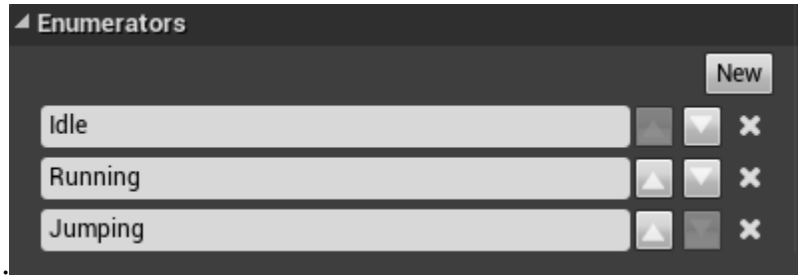
vždy stejnou velikost jako kapsle charakteru nastaveného v herním režimu, tudíž přesně vidíme, kde se postava objeví.

Když je vše připravené, spustíme hru pomocí tlačítka **Play** vpravo na panelu nástrojů. Na výběr máme několik režimů. Nejvhodnější jsou režimy **Selected Viewport**, který spustí hru přímo v náhledu, nebo režim **New Window**, jenž otevře hru v novém okně editoru. Hráčova postava by v této chvíli už měla být schopna reagovat na vstupy, tzn., běhat a skákat a také otáčet se podle toho, na jakou stranu se pohybuje.

Hraní se ukončí tlačítkem **Stop** na panelu nástrojů nebo z klávesnice klávesou ESC.

6.6 Přidání animací postavě

Prvním krokem bude vytvoření výčtu všech animací, které chceme použít. V průzkumníku proto vytvoříme nový soubor typu **Enumeration**, který najdeme v nabídce v kategorii **Blueprints**. Soubor otevřeme a vidíme, že se jedná vskutku pouze o jednoduchý výčet bez jakýchkoli dalších funkcí. Pomocí tlačítka New přidáme tři položky a pojmenujeme tak, aby odpovídaly našim připraveným animacím. Uložíme a zavřeme



Obrázek 9
Výčet Animací (Zdroj: autor)

Znovu si otevřeme graf událostí našeho charakteru. Tentokrát budeme pro nový graf potřebovat definovat vlastní proměnné a funkce.

6.6.1 Definování proměnných

Proměnné jsou vlastnostmi třídy držícími určitou hodnotu nebo odkaz na jiný objekt ve hře. Tyto hodnoty jsou specifické pro každou z instancí třídy, ve které se tyto proměnné nachází, přičemž třída pouze udává, jaké proměnné tato instance bude mít. Tyto atributy mohou být přístupné pouze vnitřně v rámci domovské třídy nebo editovatelné, tak aby byly upravitelné vně třídy. Jako u většiny ostatních programovacích jazyků zde nalezneme celou řadu typů proměnných. Ty se od sebe liší svým datovým typem. [15] V grafech jsou tyto typy reprezentovány různými barvami pro lepší přehlednost. Nejpoužívanější typy jsou uvedeny v tabulce 1.

Tabulka 2: *Datové typy*

Datový typ	Barva	Použití
Boolean	Červená	Pravda nebo nepravda
Integer	Tyrkysová	Čísla bez desetinných míst
Float	Zelená	Desetinná čísla
String	Tmavě růžová	Řetězec alfanumerických znaků
Vector	Žlutá	Skupina tří čísel definujících 3D vektor
Rotator	Fialová	Skupina čísel určujících rotaci v 3D prostoru
Transform	Oranžová	Agreguje lokaci, rotaci a měřítko
Object	Modrá	Reference jiných tříd

Nejdříve budeme potřebovat proměnné, do kterých budeme ukládat stavy, ve kterých se hráč právě nachází. Na záložce My Blueprint vpravo dole vybere **Add New** a z nabídky zvolíme **Variable**. Tato proměnná bude říkat, zdali postava zrovna běží nebo ne. Ihned se přidá do seznamu proměnných, který můžeme vidět na stejné kartě a kde ji lze také pojmenovat (např. *isRunning*. Na kartě detailů nyní můžeme upravit její specifikaci. Hned pod názvem je kolonka Variable type, určující její datový typ. Vidíme, že byla přednastavena jako typ boolean, což se nám přesně hodí. Pokud hráč poběží, bude hodnota této proměnné pravda, pokud ne, bude to nepravda.

Naprosto stejně vytvoříme ještě jednu proměnnou a pojmenujeme např. *isJumping*. Ta bude říkat, zda postava zrovna skáče.

Další proměnné, které budeme potřebovat, budou sloužit jako odkazy na jednotlivé soubory typu Flipbook obsahující naše animace. Vytvoříme tedy další proměnnou. Její datový typ už však nebude boolean, ale změníme ho v kolonce **Variable Type** na **Paper Flipbook**. Tento typ se nachází v kategorii Object Reference, jelikož sám o sobě nedrží žádnou explicitní hodnotu. Místo toho funguje jako reference na jiný již vytvořený objekt. Zatím však nikam neodkazuje, nastavily jsme pouze, na jaký typ objektu bude v budoucnu odkazovat. Abychom mohli tuto hodnotu nastavit, musíme třídu nejdříve zkompilovat pomocí tlačítka **Compile**. Následně se nám zpřístupní hodnota **Default Value** úplně dole na kartě detailů. Do ní vložíme Flipbook animaci pro běh.

Obdobně vytvoříme proměnné také pro animaci nečinnosti a skoku.

6.6.2 Uživatelské funkce

Funkce jsou samostatné uzlové grafy náležící určité třídě, které lze v rámci této třídy inicializovat z jejího grafu událostí. Funkce mají vždy jeden vstupní uzel nesoucí její jméno a žádný nebo maximálně jeden výstupní uzel. [3 str. 125]

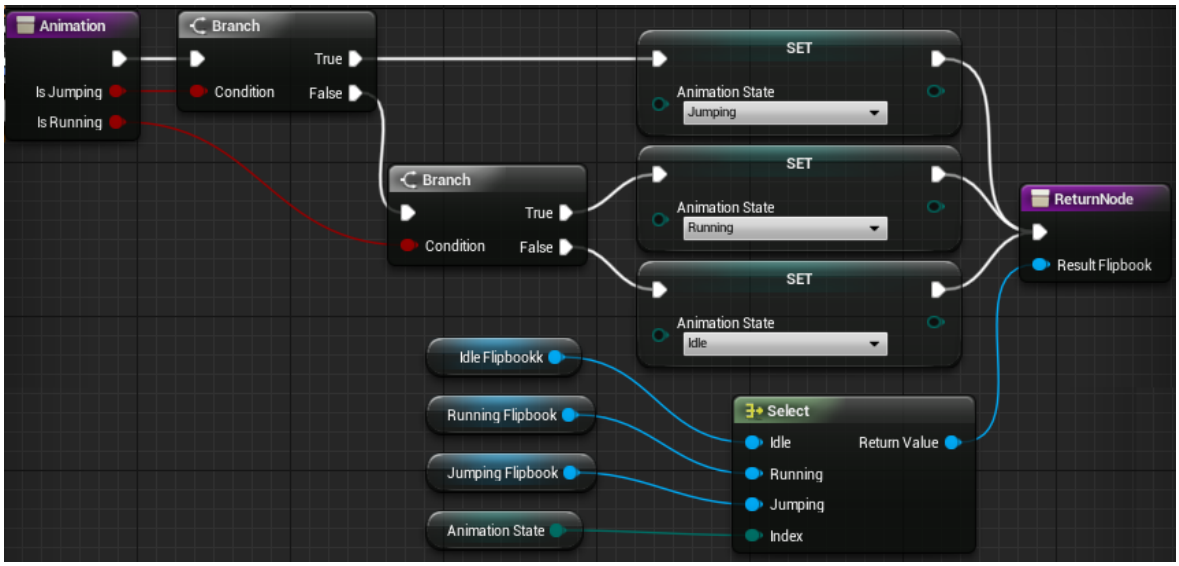
Budeme potřebovat vytvořit funkci, která zvolí vhodnou animaci na základě stavu, ve kterém se hráčova postava právě nachází. Pro její založení postupujeme jako při tvoření proměnných. Na kartě My Blueprint vybereme **Add New** a v nabídce položku **Function**. Novou funkci pojmenujeme např. „Animation".

Editor automaticky otevře nové okno, kde už čeká vstupní uzel a ve kterém lze definovat graf této funkce. Postup je následující:

1. Označíme vstupní uzel a na kartě detailů pomocí tlačítka **New** v sekci Inputs přidáme dva vstupy. Pojmenujeme je *isRunningLocal* a *isJumpingLocal* a jejich typ ponecháme jako boolean. Tudy budou do funkce vstupovat hodnoty proměnných, které jsme nedávno vytvořili.
2. Pomocí tlačítka **New** v sekci Outputs přidáme také jeden výstup, pojmenujeme ho *outputFlipbook* a jeho typ nastavíme na Paper Flipbook. Ihned se sám vytvoří uzel v výstupem, v němž bude muset funkce končit. Výstupem této funkce tedy bude jedna z animací typu Flipbook vybraná na základě logiky této funkce.
3. Nyní potřebujeme vytvořit podmínky. Tento účel plní v Blueprint uzel Branch, ale v nabídce ho lze nalézt i po zadání „if". Přidáme tedy jeden a do jeho datového vstupu typu boolean zapojíme *isJumpingLocal* ze vstupního uzlu. Tento uzel bude větvit graf. Dokud bude postava skákat (tzn. proměnná *isJumping* bude pravda), bude aktivní jeho horní výstup, avšak jakmile postava skákat přestane, aktivuje se jeho dolní výstup.
4. Postava může běžet nebo stát pouze v případě, že zrovna neskáče, tzn., že druhou podmínku navážeme na spodní výstup té první a do jejího datového vstupu navážeme *isRunningLocal*. Tento uzel bude fungovat naprosto stejně, pokud postava poběží, aktivuje horní výstup a naopak. Dále víme, že pokud ani neskáče ani neběží, musí nutně stát, jelikož více stavů už nemá. Vytvořili jsme tedy strukturu, která jednoznačně větví graf podle stavu, ve kterém se postava aktuálně nachází.
5. Nyní využijeme dříve vytvořený výčet. Nejprve definujeme ve funkci lokální proměnnou. Tu najdeme jako **Local Variable** na kartě My Blueprint, a pojmenujeme ji *animationState*. Lokální je proto, že bude existovat pouze v rámci této funkce. Její datový typ nastavíme jako referenci našeho animačního výčtu a to tak, že do nabídky napíšeme jeho název.
6. Na kartě My Blueprint vezmeme vtvořenou lokální proměnnou a přetáhneme ji do grafu. Nabídnou se dvě možnosti – **get** a **set**, tj. jestli chceme její hodnotu načíst nebo

naopak nastavit. Zvolíme **set**. Tím se vytvoří uzel, který má v sobě rozbalovací nabídku. V ní zvolíme hodnotu pro skok z našeho výčtu. Do vstupu tohoto uzlu připojíme větev grafu odpovídající skoku.

7. Opakujeme krok 6 pro běh a nečinnost.
8. Znovu vezmeme proměnnou *animationState*, ale tentokrát zvolíme get. Na výstup navážeme nový uzel Select. Ten obecně slouží pro výběr mezi možnostmi, kterých může jistá proměnná nabývat. V našem případě bude tedy vybírat právě z hodnot našeho animačního výčtu zpřístupněného pomocí proměnné *animationState*. Uzel si také automaticky doplní vstupy podle možností, kterých může proměnná *animationState* nabývat.
9. Do každého z těchto vstupů zapojíme odpovídající Flipbook proměnnou, jenž jsme vytvořili dříve.
10. Nakonec všechny výstupy uzlů Set Animation State zapojíme do výstupního uzlu a výstup uzlu Select do jeho datového vstupu. Tím je naše první uživatelská funkce hotová.



Obrázek 10

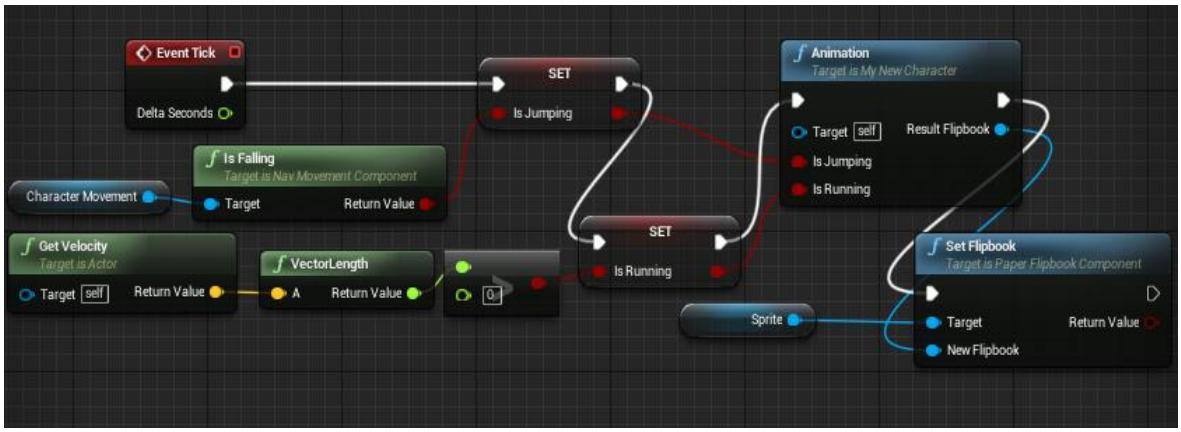
Hotový graf funkce Animation (Zdroj: autor)

6.6.3 Výsledný graf animace

Tím bohužel práce nekončí, tuto funkci je dále třeba navázat v grafu událostí, naplnit její vstupy a využít její výstup. [8] Vrátíme se proto do grafu událostí a postupujeme takto:

1. Využijeme událost Event Tick, ta je jednou ze tří zamlžených událostí, které zde byly předdefinovány už od založení třídy. Tato událost je specifická, jelikož k jejímu volání dochází při každé iteraci programu bez ohledu na ostatní faktory. Vezmeme ji a navážeme na ni nový graf.

2. Nejprve potřebujeme pro naše proměnné *isJumping* a *isRunning* najít nějaký herní mechanismus, na jehož základě jim bude možné přiřadit hodnoty. Jinak řečeno potřebujeme něco, co nám řekne, zdali postava skáče, příp. běží.
3. Pro *isJumping* je to vcelku jednoduché, pro komponentu Character Movement je již definována funkci is Falling, která obsahuje přesně tuto funkcionalitu. Přetáhneme tedy tuto komponentu z komponentové karty přímo do grafu a na výstup nového uzlu navážeme uzel is Falling.
4. Nyní vezmeme z karty My Blueprint proměnnou *isJumping* a zvolíme **set**. Do jejího exekučního vstupu připojíme událost Event Tick a do boolean vstupu zapojíme výstup uzlu isFalling.
5. Pro *isRunning* musíme nejdříve vytvořit uzel Get Velocity, který slouží pro zjištění rychlosti, kterou se postava právě pohybuje. Logicky pokud bude tato hodnota větší než nula, charakter je v pohybu. Výstupem této funkce je však vektor, složený z jednotlivých směrů. Ten je třeba převést na datový typ float, k čemuž slouží uzel Vector Length. [20]
6. Na výstup uzlu Vector Length navážeme porovnávací uzel, který najdeme, pokud do nabídky napíšeme „>“. Jeho druhý vstup nastavíme na hodnotu nula.
7. Z karty My Blueprint vezmeme tentokrát proměnnou *isRunning* a zvolíme opět **set**. Nový uzel bude exekučně navazovat na uzel Set *isJumping* a do jeho boolean vstupu připojíme výstup z porovnávacího uzlu.
8. Nyní můžeme zavolat naši animační funkci. Najdeme ji v nabídce podle jména v kategorii Call function. Vidíme, že její uzel obsahuje námi specifikované vstupy a výstupy. Do těchto vstupů připojíme výstupy z příslušných „set uzlů“ a také nesmíme zapomenout navázat na exekuční řetězec.
9. Nakonec exekučního řetězce nyní napojíme poslední uzel – Set Flipbook. Tento poslední uzel je zodpovědný za samotnou změnu objektu typu Flipbook ve Sprite komponentě, zatímco celý předcházející graf slouží pouze pro zjištění stavu, ve kterém se postava nachází a vybrání jedné z animací, kterou bude tento poslední uzel „nakrmen“. [8] Vstup Target se automaticky nastaví na Sprite komponentu naší postavy, tzn., že bude nastavovat Flipbook obsažený právě v této komponentě. Do posledního vstupu (New Flipbook) připojíme objektový výstup z funkce Animation. Tímto uzlu řekneme jaký Flipbook má nastavit.



Obrázek 11

Graf pro animace v grafu událostí (Zdroj: autor)

Když vytvořený graf připomíná ten na obrázku 11, zkompilujeme třídu a uložíme. Není od věci znovu si projít všechny grafy v této třídě a uvědomit si, jakou funkci každý z uzlů plní, a jak na sebe vzájemně navazují. Pochopení těchto základních struktur je klíčem k samostatné práci v jazyce Blueprint.

Hráčova postava nyní dokáže plynule přecházet mezi jednotlivými animacemi na základě toho, jakou činnost právě vykonává. Můžeme spustit hru a přesvědčit se na vlastní oči.

Tento postup je znovu aplikovatelný i pro další objekty, u nichž chceme přecházet mezi více animačními stavy. Rozdíly by mohli být pouze v definici podmínek, při kterých dochází ke změně stavu.

7 Životy a jak o ně přijít

V této kapitole budou vytvářeny a nastražovány různá nebezpečí, která budou na hráče čekat v průběhu hraní. Aby však byly relevantní, musí být pro hráčovu postavu nejprve definován systém životů a co se má stát když mu životy dojdou. Také je potřeba poskytnout hráči možnosti jak tato nebezpečí překonat. Během implementace těchto funkcí bude použito definování vlastních událostí, komunikace mezi třídami nebo práce s kolizí a časovými osami.

7.1 Systém životů

Nejprve vytvoříme pro hráče systém životů. Ten bude reprezentovaný číselnou hodnotou a při kolizi s nepřítelem dojde ke změně této hodnoty. Pokud se hodnota této proměnné dostane na nulu, hráč zemře. Po utržení zranění bude navíc hráč na krátkou dobu imunní na další zásahy.

Otevřeme si tedy graf událostí hráčovy třídy a definujeme novou proměnnou. Ta se bude jmenovat *health* a bude typu float. Dále zkompilujeme třídu, abychom mohli nastavit defaultní hodnotu této proměnné, kterou následně nastavíme na 1. S touto hodnotou pak bude hráč začínat při startu každé úrovně. [16] Ještě budeme potřebovat jednu proměnnou, pojmenujeme ji *immunity* a její datový typ bude boolean.

7.1.1 Uživatelské události

Co jsou události, už víme. V jazyce Blueprint jsou již předdefinovány tyto události, přičemž s některými jsme se již setkali [15]:

- Event Begin Play – tato událost se spustí na začátku každé úrovně
- Event Begin Overlap – k této události dojde pouze v případě že:
  a) Kolize mezi objekty umožňuje překrývání.
  b) Jeden objekt začne překrývat druhý objekt.
- Event Hit – podobné jako u překrytí, avšak pro vzájemně se blokující objekty
- Event Tick – tato událost je volána při každé iteraci programu

Pro implementaci zranění při kolizi s nepřítelem si však musíme vytvořit svoji vlastní událost. Stejně jako ostatní bude i tato událost počátečním bodem procházení nějakého grafu. Specifická bude tím, že bude explicitně inicializována z nějakého jiného vytvořeného grafu v této nebo i jiné třídě. Uživatelským událostem lze navíc také definovat vstupy.

Vytvoříme tedy nový uzel. Najdeme ho v nabídce jako Add Custom Event a pojmenujeme ho „*healthEvent*". Také mu na kartě detailů přidáme nový vstup, který se bude jmenovat

Value a bude typu float. Tuto událost budeme inicializovat z jiných tříd, když budeme chtít změnit stav hráčova života, přičemž vstup *Value* bude představovat, jak velká bude tato změna.

7.1.2 Změny života

Dále je třeba vytvořit graf reprezentující, co se má stát, pokud dojde k události *healthEvent*. Požadovanou funkcionalitou je změnit stav proměnné *health* a nastavit na krátký okamžik proměnnou *immunity* na pravdu. Následně zjistit jestli má hráč stále dostatek života a může tak pokračovat ve hře, nebo jestli přišel o svůj poslední život a má zemřít.

Postup tvorby grafu bude následující:

1. Na událost *healthEvent* navážeme větvící uzel Branch. Jeho datovým vstupem bude proměnná *immunity*.
2. Na spodní výstup uzlu Branch, (tj. pokud proměnná *immunity* je nepravda) navážeme uzel Set Health. Jinak řečeno pokud nemá hráč aktivní imunitu, pokračuj touto větví. Horní větev zůstane prázdná, jelikož pokud je hráč imunní nedojde k žádné změně života.
3. Vytvoříme uzel Get Health a dále na něj napojíme uzel Float – Float. Tento uzel slouží jako matematická operace mínus. Do jeho druhého vstupu tedy zapojíme výstup *Value* z uzlu s událostí a jeho výstup zapojíme do vstupu uzlu Set Health. Tato část grafu vezme aktuální stav proměnné health a odečte od ní hodnotu poškození způsobeného nepřítelem. Výsledkem následně přepíše stávající hodnotu proměnné *health*.

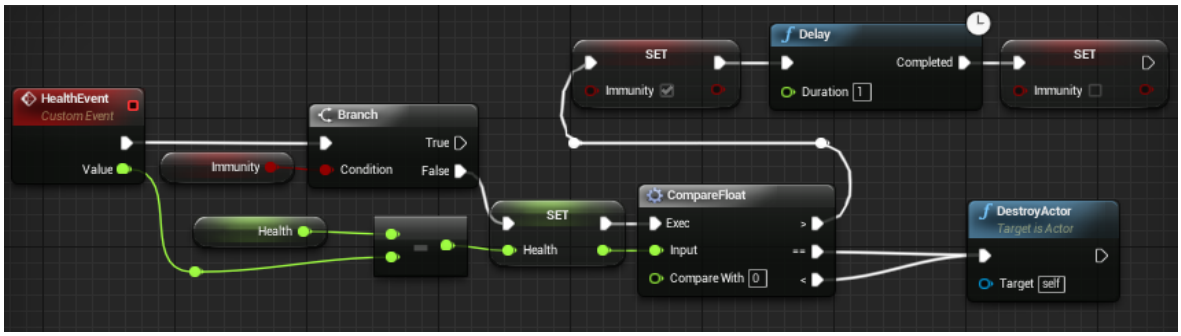
   Tímto máme dokončenou první část grafu, tj. část pro změnu ve stavu proměnné *health*. Dále potřebujeme vytvořit porovnávací část.
4. Na uzel Set Health navážeme nový uzel Compare Float jehož vstup Input bude navazovat výstup uzlu Set Health. Jeho druhý vstup nastavíme na hodnotu 0, jelikož stav hráčova života chceme porovnávat právě s nulou.
5. Na horní větev uzlu Compare Float (tj. pokud má hráč pořád dostatek života) navážeme uzel Set Immunity, a jeho hodnotu nastavíme na pravdu. Za něj zapojíme uzel Delay, který slouží jako pauza v průchodu grafem. Jeho hodnota je v sekundách a nastavíme ji na 1. Dále napojíme další uzel Set Immunity, tentokrát však nastavíme jeho hodnotu na nepravdu.

   Potom co hráč utrpí zranění, zapne tato větev hráčovy imunitu a po uplynutí jedné vteřiny ji opět vypne. Pokud bychom tuto větev grafu opomenuli, při kolizi s nepřítelem by došlo k volání události *healthEvent* při každé iteraci programu a hráč by tak ve většině případů zemřel, ještě než by stačil zareagovat.
6. Za tuto horní větev můžeme ještě přidat uzel Print String, do nějž lze napsat libovolný text. Tento uzel potom zobrazí uvedený text na hráčově obrazovce. Můžeme sem taky

napojit výstup z uzlu Set Health a potom bude místo toho vypisovat aktuální stav životů (na spoji také dojde k automatické změně datového typu z float na string).

Pozn. Tento krok nabízí pouze provizorní řešení, v kapitole 10 vytvoříme pro zobrazování aktuální stavu života grafické rozhraní.

7. Na zbývající dva výstupy uzlu Compare Float navážeme uzel Destroy Actor. Pokud tedy hráči dojdou životy, dojde ke zničení hráčovy postavy.

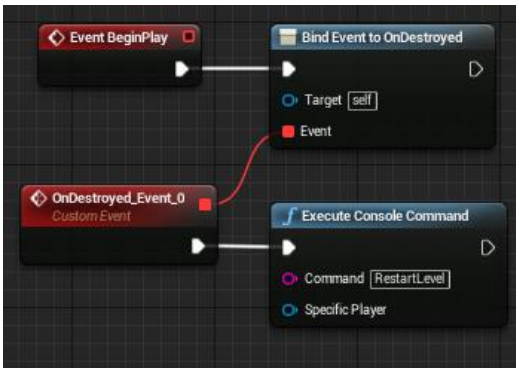


Obrázek 12
Systém životů (Zdroj: autor)

7.1.3 Odesílatel událostí

Dalším krokem bude definovat postup po zničení postavy. Požadované chování je, aby při jejím zničení došlo k restartování úrovně. Tuto funkcionalitu zajistíme pomocí tzv. odesílatele událostí (z angl. Event Dispatcher) [15], jenž na sebe dokáže připoutat nějakou událost, kterou jinak nelze volat přímo.

Najdeme tedy další ze zamlžených událostí, konkrétně Event Begin Play. To jest událost volaná jednou při začátku hraní úrovně. Na ní navážeme odesílatele událostí, najdeme ho v nabídce uzlů jako Assign On Destroyed. Automaticky se vytvoří nová událost a napojí se na speciální delegační spoj. Tento odesílatel událostí si tedy na začátku každé úrovně připoutá tuto událost, a pokud dojde ke zničení postavy, spustí ji.



Obrázek 13
Postup při zničení postavy (Zdroj: autor)

Zbývá definovat, co se má při zničení vlastně stát, např. můžeme chtít restartovat úroveň. Pro tuto funkci jazyk Blueprint nemá specifický uzel, a proto vytvoříme uzel Execute Console Command pro přímé zadání příkazu. Do jeho string vstupu Command tedy napíšeme příkaz pro restartování úrovně: „RestartLevel“.

7.2 Statický nepřítel

Nyní je na čase přidat do hry objekty, které budou mít vliv na hráče a jeho životy. Statická nástraha bude první a nejjednodušší překážkou. Její funkcionalita bude takováto: Pokud dojde ke kolizi s hráčem, zavolej v hráčově třídě událost *healthEvent* a sděl jí hodnotu způsobeného poškození.

Nejdříve si importujeme vhodnou texturu a uděláme z ní objekt typu Sprite. Následně na něj klikneme pravým tlačítkem a v nabídce vybereme **Asset Action → Create Blueprint Using This**. Takto si ušetříme práci, jelikož rodičovská třída se sama zvolí jako Actor a zároveň se v této nové třídě vytvoří Sprite komponenta již obsazená tímto objektem typu Sprite.

Pro tuto třídu definujeme novou proměnnou, kterou pojmenujeme *damageValue*. Ta bude typu float a bude udávat, jaké poškození tento objekt hráči způsobí. Hodnotu nastavíme např. na 0,25. Jelikož hráč začíná s hodnotou života 1, je schopen přežít tři kolize s tímto objektem a až ta čtvrtá se mu stane osudnou. Tuto hodnotu si lze samozřejmě upravit podle vlastních potřeb.

7.2.1 Kolize a kolizní odezvy

Nyní potřebujeme pro tuto nástrahu definovat kolizi. UE4 rozlišuje tři typy vzájemné kolize dvou objektů: [15] [17]

1. Objekty se blokují – možnost definovat kolizní odezvy při nárazu.
2. Objekty se ignorují – objekty mohou volně procházet jeden druhým, ale nelze jim definovat odezvy.
3. Objekty se překrývají – na první pohled se nijak neliší od ignorace, jediným rozdílem je možnost definovat kolizní odezvy v moment kdy dojde k překrytí, příp. pokud překrývání skončí.

Každý objekt a instance v UE4 má svůj kolizní typ, jenž stanoví, jak tento objekt interaguje s ostatními kolizními typy. Framework již obsahuje poměrně vyčerpávající nabídku těchto typů, např. typ Spectator.je nastaven, aby ignoroval všechny pohyblivé objekty a blokoval všechny statické. [15]

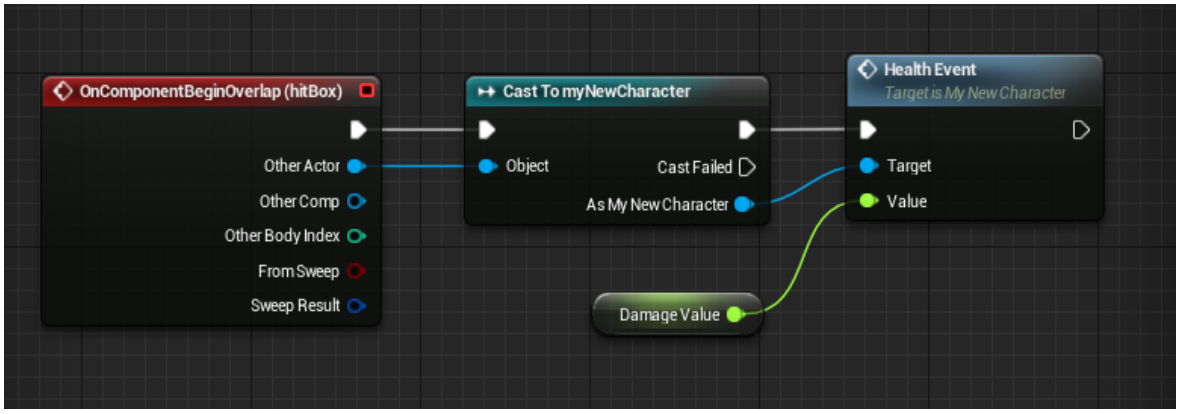
Třidě nyní přidáme novou komponentu Box Collision, jejíž kolizní typ je defaultně Overlap All, což znamená, že překrývá všechny objekty. Komponentu pojmenujeme např. *hitBox* a v náhledu transformujeme ji tak, aby některým směrem přesahovala kolizní tvar Sprite komponenty, jenž je vyznačen slabou šedou čárou.

Přesah můžeme nastavit například pouze vertikálně. Tím získáme funkcionalitu, jenž říká, že pokud postava přistoupí k nepříteli ze strany, nedojde ke kolizi s tímto kolizním tvarem a tím pádem ani ke zranění, pokud ale hráč dopadne na horní část, dojde ke kolizi a hráč se zraní. Fantazii se meze nekladou.

Dále chceme pro *hitBox* definovat kolizní odezvu, a proto přejdeme do grafu událostí. Na kartě Components si tuto kolizní komponentu označíme a vytvoříme nový uzel On Component Begin Overlap (aby se zobrazil v nabídce, musíme mít *hitBox* na kartě Components označený). Tento uzel představuje událost, kdy dojde k zahájení překrývání komponenty uvedené v závorce jiným objektem. Tento uzel má poměrně velké množství výstupů. Důležitý je výstup Other Actor, který referuje právě ten objekt, který tuto komponent začal překrývat.

7.2.2 Casting

Při kolizi nepřítele s hráčem potřebujeme inicializovat událost *healthEvent*. Ta se však nachází v hráčově třídě a v nabídce uzlů nepřítelovy třídy se tudíž nezobrazí. Musíme proto použít funkci Casting. [15] Tato funkce nám umožňuje přistoupit k událostem a proměnným jiné třídy. Založíme tedy uzel „Cast To + *jméno třídy s hráčovou postavou*“, např. na obrázku 14 je Casting pro třídu nazvanou myNewCharacter. Do vstupů tohoto uzlu připojíme exekuční a horní objektový výstup uzlu On Component Begin Overlap. Objektový spoj vybere právě tu instanci z hráčovy třídy, jenž překryje komponentu *hitBox*. Nyní můžeme navázat uzel s událostí *healthEvent*, jelikož se již zobrazuje v nabídce uzlů. Nesmíme také zapomenout připojit i objektový spoj, neboť spouštíme událost *healthEvent* pouze v jedné instanci hráčovy třídy. Posledním krokem bude do vstupu *Value* zapojit „get uzel“ proměnné *damageValue*.



Obrázek 14

Kolizní odezva a komunikace mezi třídami (Zdroj: autor)

7.3 Pohybliví nepřítel

Podobně jako statická past bude hlavní „funkcí“ pohyblivého nepřítele způsobit hráči zranění a zkomplikovat mu tak postup ve hře. Tento objekt bude mít navíc i další kolizní komponentu, díky které jej půjde zneškodnit. Oproti statickým nástrahám bude také vždy obsahovat alespoň jeden animační stav.

Nejjednodušší je nepřítel, jenž se nepřetržitě pohybuje mezi dvěma body. Jelikož je neustále v rovnoměrném pohybu, vystačí si pouze s jednou animací.

Opět importujeme vhodné zdrojové textury, vytvoříme z nich objekty typu Sprite, které dále použijeme v novém souboru typu Flipbook pro vytvoření kráčející animace tohoto nepřítele.

Na výsledný Flipbook klikneme pravým tlačítkem myši a vybereme z nabídky **Create Blueprint Using This**. Takto vytvoříme novou třídu a pojmenujeme ji podle typu našeho nepřítele.

Třídu si otevřeme a přidáme jí dvě kolizní komponenty typu **Box Collision** a pojmenujeme je např. *hitbox* a *ripBox*. *HitBox* bude sloužit pro zranění hráče a bude přesahovat horizontálně. Ke kolizi s touto komponentou tedy dojde v případě, že hráč naběhne na nepřítele z boku. *RipBox* bude naopak přesahovat vertikálně a zajišťovat, že pokud hráč na tohoto nepřítele dopadne od shora, dojde k zneškodnění tohoto nepřítele.

Dále definujeme opět proměnnou *damageValue* se stejným účelem jako dříve.

Přejdeme do grafu událostí a vytvoříme identický graf jako ve třídě statické pasti (na obrázku 14). Dále definujeme graf pro komponentu *ripBox* takto:

1. Označíme si ji na kartě komponent a vytvoříme nový uzel On Component Begin Overlap.
2. Na něj napojíme uzel Destroy Actor, který odstraní instanci nepřítele, jejíž kolizní komponenta *ripBox* byla překryta hráčem.

7.3.1 Editovatelné proměnné

Dále vytvoříme dvě další proměnné, reprezentující body, mezi nimiž bude nepřítel pochodovat – pojmenujeme je *pointA* a *pointB*. Jejich datový typ bude Actor a oproti všem dosavadním proměnným jim zatrhneme políčko **Editable**. Tato změna jim umožní být modifikovatelné pro jednotlivé instance přímo v úrovni. Tyto proměnné dále poznáme podle symbolu oka, které se u každé otevře na kartě My Blueprint.

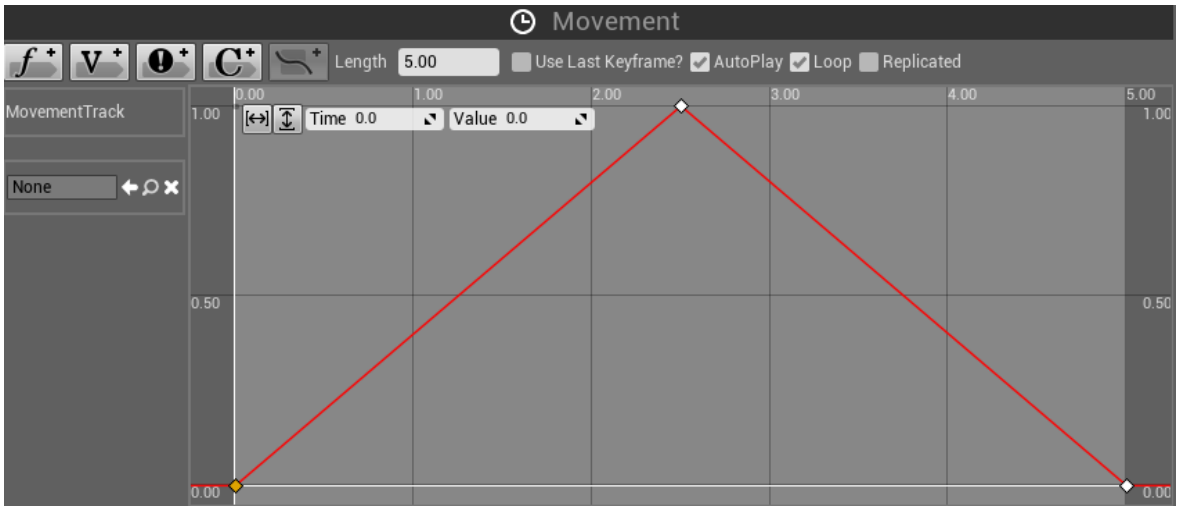
Do těchto proměnných později dosadíme skutečné objekty z úrovně, jenž budou představovat body, mezi nimiž bude nepřítel strážit.

7.3.2 Časová osa

Dalším krokem bude implementace pohybu. K těmto účelům budeme potřebovat uzel Timeline a v nabídce uzlů proto vybereme Add Timeline. Tento uzel má několik specifických vlastností – lze ho pojmenovat a dvojklikem také otevřít. Uzel se nám otevře na nové kartě, kde můžeme vytvořit novou křivku emitující různé typy dat v průběhu určitého časového intervalu. V horní nabídce vybere tlačítko se symbolem $\int^{+}$ pro vložení nové stopy typu float. Tato stopa bude generovat float data v průběhu námi definovaného časového úseku podle

křivky kterou nyní nastavíme. V horní části ještě nastavíme položku **Duration** na hodnotu 5. Tato hodnota udává, jak dlouhý bude onen časový úsek v sekundách. Tento interval se v grafu níže zobrazuje světlejší barvou. Dále zaškrtneme položky **Auto play** (funkce se sama spustí) a **Loop** (bude se neustále opakovat).

Výsledná funkce je reprezentována červenou čarou níže v tomto okně. Najedeme na ni kurzorem a stiskneme SHIFT + LEVÉ TLAČÍTKO MYŠI. Tím dojde k vytvoření nového kotevního bodu. V horní části se nám zobrazí dvě vlastnosti tohoto bodu – **Time** a **Value**, neboli čas a hodnota v tomto časovém okamžiku. Obě hodnoty nastavíme na 0. Dále vytvoříme ještě dva kotevní body. Hodnoty prvního budou 2.5 a 1 a druhého 5 a 0. Vnikne nám křivka, která má na počátku vymezeného časového úseku hodnotu nula a postupně se zvětšuje, až dosáhne hodnoty jedna. V tento moment se začne naopak zmenšovat, až nakonec skončí opět na nule.



Obrázek 15

Uzel Timeline (Zdroj: autor)

Vrátíme se zpět do grafu událostí a pomocí následujících kroků vytvoříme nový graf:

1. Vytvoříme „get uzel“ pro proměnné *PointA* i *PointB*. Na každý z nich navážeme jeden uzel Get Actor Location, který zjistí polohu tohoto objektu ve hře.
2. Vytvoříme uzel Lerp (vector). Tento uzel představuje matematickou funkci tzv. lineární interpolace [21] Jeho první dva vstupy navážeme na výstupy předchozích uzlů. Do posledního vstupu (Alpha) zapojíme float výstup uzlu Timeline. Výstupem tohoto uzlu bude měnící se vektor představující pozici mezi objekty *pointA* a *pointB* v závislosti na vstupu Alpha. Např. pokud bude Alpha 0, bude výstupem pozice *pointA*, pokud bude Alpha 1, bude to pozice *pointB* a pokud bude Alpha 0,5, bude výstupem pozice na půl cesty mezi oběma.
3. Na exekuční výstup uzlu Timeline navážeme uzel Set Actor Location, pro nastavení pozice nepřítele.

4. Vytvoříme spoj vektorových dat mezi Lerp a Set Actor Location. Uzel Lerp bude vysílat měnící se vektory reprezentující pozice v úrovni a uzel Set Actor Location podle nich vždy nastaví pozici nepřítele.

   Nepřítel je nyní již pohyblivý, ale obdobně jako u hráčovy postavy zbývá nastavit mu otáčení, podle směru, kterým se pohybuje. Na rozdíl od hráče však nemáme k dispozici jednoznačný identifikátor tohoto směru podle vstupů z klávesnice a musíme si tudíž poradit jinak.

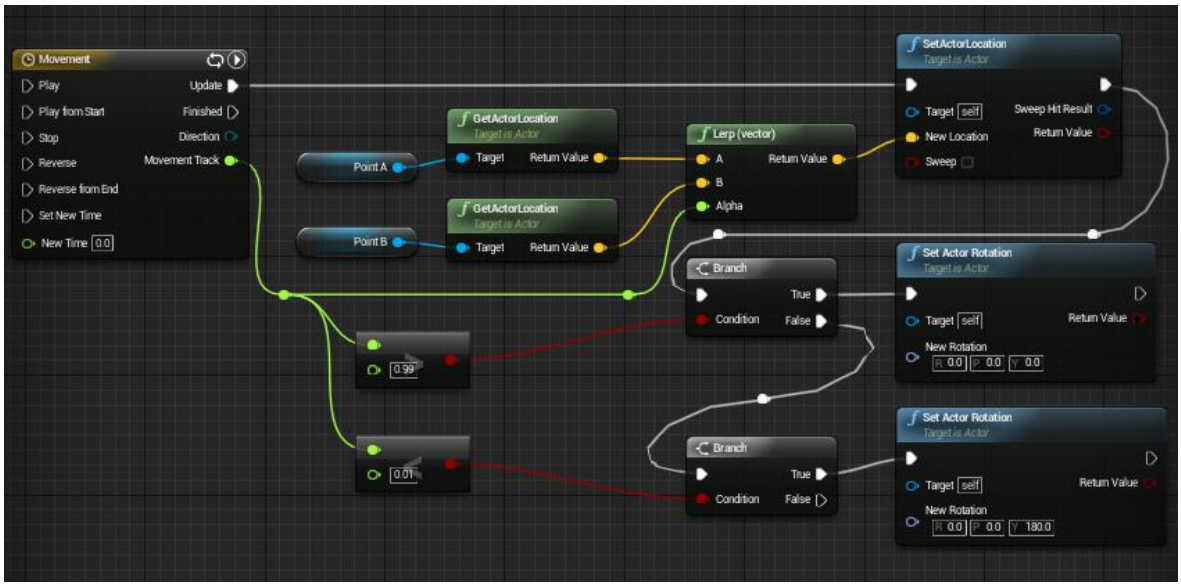
5. Nepřítele je třeba otočit, pokud dorazí do jednoho z koncových bodů. Použijeme hodnotu Alpha, jelikož víme, že pokud nabyde hodnot 1 nebo 0, nepřítel se právě nachází v jednom z těchto bodů. Na float výstup Timeline tedy napojíme porovnávací uzel „>“ a jeho druhý vstup nastavíme na hodnotu blízkou jedné např. 0.99 (technicky by mohla hodnota být rovna jedné, v UE4 to však často způsobuje problémy).

6. Na boolean výstup porovnávacího uzlu napojíme větvící uzel Branch který bude exekučně následovat uzel Set Actor Location. Na horní výstup Branch napojíme uzel Set Actor Rotation, hodnoty ponecháme.

7. Na float výstup Timeline dále napojíme uzel „<“ a jeho druhý vstup nastavíme na hodnotu blízkou nule např. 0.01 (stejný problém jako v bodě 5).

8. Na výstup navážeme nový uzel Branch a exekučně ho připojíme na předchozí uzel Branch. Podobně jako v předchozím kroku nakonec přidáme uzel Set Actor Rotation. Jeho hodnotu Y v položce New Rotation nastavíme na 180, což zaručí, že se charakter otočí o 180° po ose Y.

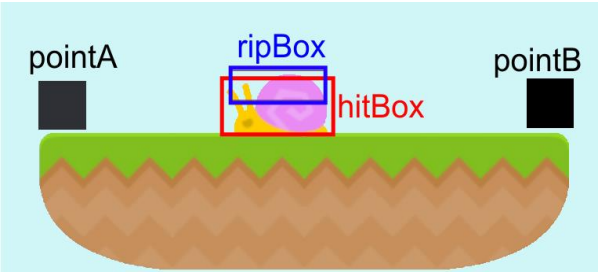
Výsledek by měl připomínat graf na obrázku 16.



Obrázek 16

Graf pro pohyb nepřítele (Zdroj: autor)

Nyní můžeme instanci této třídy přidat do hry. Aby fungovala správně, musíme jí ještě poskytnout body, mezi kterými se bude pohybovat. Označíme ji v náhledu a na kartě detailů najdeme kategorii Default. Tady můžeme pomocí tlačítka se symbolem kapátka obsadit proměnné *pointA* a *pointB* nějakými konkrétními objekty ze hry. Pokud se v úrovní vhodné objekty zatím nenachází, což je velmi pravděpodobné, můžeme vložit jednoduché tvary z palety. Těmto objektům však musíme na kartě detailů vypnout kolizi nastavením kolizního typu na No Collision. O několik řádků níže se nachází možnost **Actor Hidden In Game**, jejímž zatrhnutím dojde ke skrytí objektu během hraní hry. Protože se bude nepřítel pohybovat přímo mezi těmito tvary, musí být jejich souřadnice na ose Y rovny 0.



Obrázek 17

Nepřítel s kolizními tvary a referenčními body pohybu (Zdroj: autor)

Podobně můžeme tvořit další nepřátele. Uzel Timeline nabízí sám o sobě spoustu možností k experimentování. Můžeme např. vytvářet složitější šablony pohybu mezi více body, se zastavováním, změnami rychlosti apod.

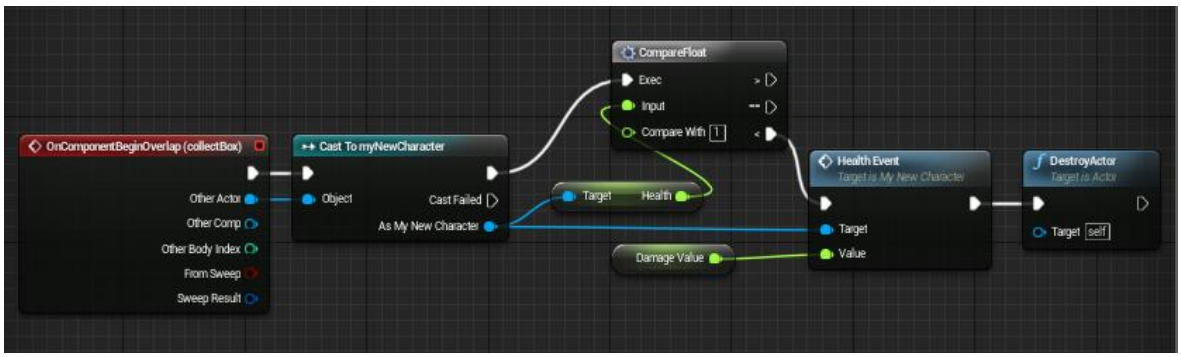
7.4 Smrt pádem

Dalším způsobem jak může hráč zemřít, je „vypadnutí“ z úrovně bez možnosti návratu. Pro tyto účely můžeme najít položku Kill Z Volume, kterou najdeme na panelu nástrojů pod **Settings** →**World Settings**. Tato hodnota definuje hladinu na ose Z. Pokud se na/pod ni hráč dostane, je jeho postava okamžitě zničena. [18] Jelikož již máme stanovený postup při jejím zničení, stačí nám nastavit tato hodnota a máme hotovo. V náhledu se hladina Kill Z zobrazuje jako tlustá červená čára.

7.5 Interaktivní předměty

Podobně jako nepřátele lze do hry vkládat další objekty jinak interagující s hráčem. Např. vytvoříme lékárničku, která hráče při sebrání vyléčí. Postup je obdobný jako u předcházejících kapitol, proto bude uveden pouze ve zkratce.

Předmětu stačí pouze jedna kolizní komponenta přesahující na všechny strany. Lékárnička také potřebuje proměnnou *damageValue*, jelikož bude v hráčově třídě iniciovat událost *healthEvent* podobně jako nepřátelé. Hodnota této proměnné musí však být záporná, jelikož má na hráčův stav životů opačný efekt. Při sebrání také chceme, aby předmět zmizel a nešel tudíž sebrat znovu.



Obrázek 18

Graf pro sebrání srdíčka (Zdroj: autor)

Výsledný graf může vypadat jako na obrázku 18. Jedinou věcí, která by snad mohla být nejasná je přítomnost uzlu Compare Float. Ten porovnává hráčův stav života, a pokud má hráč plný počet, lékárnička se nesebere.

8 Podmínky vítězství a další úrovně

Dalším krokem bude poskytnout hráči prostředky pro úspěšné dokončení úrovně. Pro každou úroveň je tedy nezbytné definovat podmínky, které musí hráč splnit a co se má stát dále, pokud je skutečně splní.

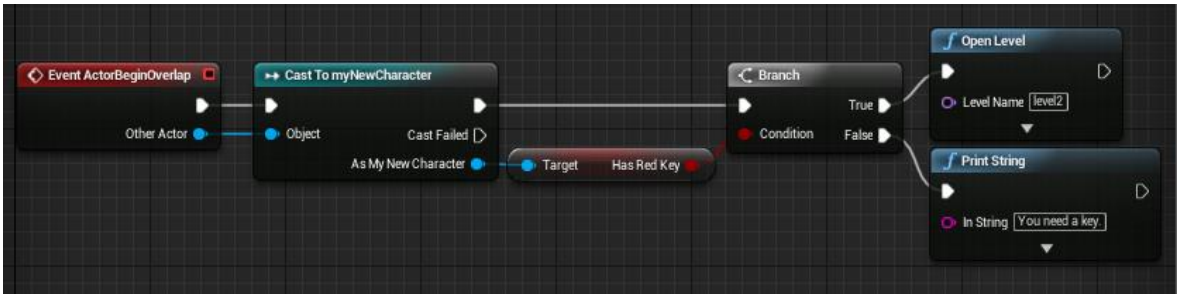
Pro naši zatím jedinou úroveň definujeme následující podmínku vítězství: Hráč v úrovni najde a získá klíč a následně se dostane na konec úrovně k zámku, který pomocí tohoto klíče otevře.

Nejprve definujeme v hráčově třídě proměnnou typu boolean, kterou pojmenujeme např. *hasKey*. Ta bude udávat, zdali má hráč klíč u sebe nebo ne. Také budeme potřebovat objekty typu Sprite s klíčem a se zámkem. Naimportujeme si je tedy a z obou vytvoříme nové třídy.

Nejdříve budeme pracovat s třídou klíče. Požadovanou funkcionalitou je, aby při překrytí klíče hráčem, došlo v hráčově třídě k nastavení proměnné *hasKey* na pravdu a zničení této instance klíče. Jako počáteční bod nového grafu využijeme událost On Actor Begin Overlap. Zbytek grafu je takřka identický s grafem lékárničky v předchozí kapitole. Jediným rozdílem je úkon, který se provede v hráčově třídě. Lékárnička inicializovala novou událost, zatímco klíč pouze změní hodnotu proměnné *hasKey*.

Pro zámek bude situace o poznání složitější, jelikož na jednu větev budeme chtít navázat otevření nové úrovně a v našem projektu se zatím nachází pouze jedna. Nejprve tedy budeme muset založit novou úroveň a vytvořit v ní alespoň základní strukturu. (Postup pro založení nové úrovně se nachází na straně 9) Nesmíme také zapomenout přidat na nějaké rozumné místo objekt Player Start a opět změnit herní režim na kartě World Settings. Mezi úrovněmi lze libovolně přecházet v hlavním menu pomocí **File → Open Level.**

Když máme další úroveň vytvořenou, můžeme se pustit do tvorby zámku. Otevřeme si jeho třídu a v jejím grafu událostí navážeme na zamlženou událost On Actor Begin Overlap nový graf. Ten se nejprve podívá do překrývající instance hráčovy třídy a zjistí hodnotu proměnné *hasKey*. Pokud se ukáže, že je pravdivá, otevře následující úroveň (uzel Open Level). V opačném případě vypíše na obrazovku zprávu, že hráč potřebuje klíč. Výsledek by měl vypadat jako na obrázku 19:



Obrázek 19

Graf třídy zámek (Zdroj: autor)

9 Zvuk

Zvuk je další nezbytnou složkou pro tvorbu her, ať už se jedná o zvuky v pozadí, interaktivní zvuky způsobené činností hráče, nebo doprovodnou hudbu. UE4 poskytuje funkce pro modelování zvuku, což je výhodou, jelikož v externím programu stačí vyprodukovat pouze „čistou“ verzi požadovaného zvuku a tu poté ji v UE upravit do požadované podoby.

Pro import je nutné mít zvukový soubor ve formátu WAV. Tento soubor je potom chápán jako jednoduchá zvuková vlna a pro její další použití je nutné z ní vytvořit soubor typu Sound Cue. K tomu stačí na importovaný zvukový soubor kliknout pravým tlačítkem myši a vybrat možnost **Create Cue**.

9.1 Zvukové efekty

Zvukovými efekty rozumíme zvuky vydávané hráčem nebo jinými herci, které jsou vydávány zejména jako reakce na určitou událost.

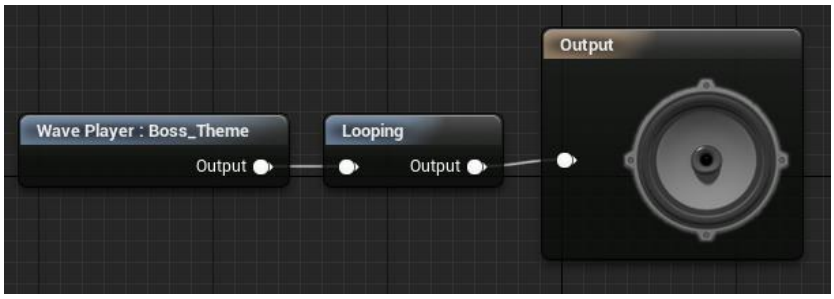
Pro tyto zvuky je většinou vhodné nastavit útlum (angl. Attenuation) [15]. Vytvoříme si proto nový soubor tohoto typu, v nabídce **Add New** v průzkumníku navigujeme takto: **Sounds** → **Sound Attenuation**. Uvnitř tohoto souboru máme možnost nastavit řadu hodnot. Pro nás budou důležité hodnoty **Radius** a **Falloff Distance**, určující vzdálenost, odkud pokud bude probíhat tlumení (položka Radius tedy udává vzdálenost, odkud bude zvuk slyšet naplno a položka Falloff Distance kam až bude zvuk slyšet tlumený. [15] Tyto hodnoty můžeme kalibrovat, až budeme lépe vědět, jak tento rozsah odpovídá vzdálenostem v našem vytvořeném světě (pokud používáme přiložené textury, obě ideální hodnoty se pohybují okolo 300).

Vytvoříme například zvuk, který bude vydávat nepřítel vytvořený dříve v kapitole 7.2. Nejprve si opět importujeme vhodný zvuk a vytvoříme nový soubor typu Sound Cue. Soubor otevřeme a přiřadíme mu výše vytvořený útlum typu Attenuation, jelikož chceme, aby byl slyšet, pouze pokud je hráč nablízku. To provedeme tak, že ho na kartě detailů, která je tentokrát netradičně vlevo, vybereme v položce **Attenuation Settings**.

Zvuk máme připraven a můžeme ho přidat nepříteli. V jeho třídě musíme nejprve vytvořit novou Audio komponentu. Zvukový soubor do ní vložíme na kartě detailů v položce **Sound** ve stejnojmenné kategorii. Nyní stačí na libovolné místo v grafu událostí navázat uzel Play. Například chceme tento zvuk přehrát, pokud nepřítel dorazí do jednoho z bodů, mezi nimiž se pohybuje. V takovém případě navážeme jeden uzel Play na každý uzel Set Actor Rotation, jelikož nepřítel se otáčí, právě když je v jednom z těchto bodů.

9.2 Hudba

Pro hru také vytvoříme doprovodnou hudbu. Importujeme si vhodný soubor a opět z něj vytvoříme soubor typu Sound Cue. Tento zvuk však bude mít jednu speciální vlastnost – bude se opakovat. Otevřeme si jej a do existujícího grafu kde uzel Wave Player přehrává zdrojovou zvukovou vlnu, a uzel Output reprezentuje finální výstup tohoto souboru, přidáme uzel Looping, aby grafu vypadal jako na následujícím obrázku. Navíc můžeme na kartě detailů zatrhnout možnost **is UI Sound**. Potom bude hudba plynule hrát i v menu a při pozastavení hry.



Obrázek 20

Sound Cue (Zdroj: autor)

Instanci tohoto zvuku přidáme do úrovně jednoduchý přetažením z karty průzkumníka. Nezáleží, kam ji umístíme, jelikož jsme tomuto zvuku nepřiřadili žádný objekt typu Attenuation a bude tak slyšet stejně nahlas napříč celou úrovní.

10 Uživatelská rozhraní

Uživatelská rozhraní (zkrat. UI z angl.. User Interface) poskytující uživateli prostředky pro komunikaci s programem. Ve hrách se zpravidla rozlišují dva typy UI. Prvním typem jsou tzv. průhledové displeje (zkrat. HUD z angl. Heads-Up Display). Tyto objekty se zobrazují během samotného hraní a obvykle obsahují aktuální informace jako počet životů nebo stav inventáře a za běhu hry se dynamicky mění.

Druhým typem jsou běžné nabídky (často označována jako menu). Při pohybu v nabídce je hra pozastavena a uživatelovy vstupy jsou omezeny na ovládání této nabídky. Těchto nabídek budeme potřebovat hned několik. Bude to hlavní menu s podnabídkami a nastavením, menu pro vítězství odkazující na další úroveň a také menu pro smrt s možností restartovat úroveň.

Uživatelská rozhraní se vytváří pomocí nástroje UMG (Unreal Motion Graphics) Designer. Ten obsahuje dvě podokna – Designer a Graph. Designer slouží k přidávání jednotlivých grafických elementů, editaci jejich atributů a celkového vzhledu, zatímco Graph je obdobou již známého grafu událostí. [3 str. 167]

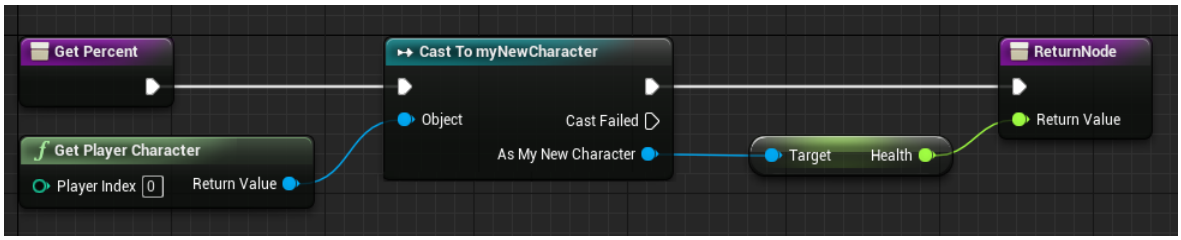
10.1 HUD

Průhledový displej bude obsahovat dvě informace – aktuální stav životů a velmi jednoduchý inventář. Obě tyto položky budou dynamicky reagovat na relevantní herní události.

Pomocí tlačítka **Add New** vytvoříme nový soubor. V kategorii **User Interface** vybereme typ **Blueprint Widget**, což je speciální soubor určený pro tvorbu uživatelských rozhraní. Stejně jako pro ostatní objekty i tentokrát vytvoříme celou třídu, jejíž instance půjde následně vkládat na hráčovu obrazovku. Nový soubor pojmenujeme a otevřeme.

V pravém horním rohu nového okna se nachází přepínač mezi režimy Designer a Graph. Začneme v režimu Designer. Pro životy potřebujeme panel, který se bude plnit podle hráčova stavu života. Tuto funkcionalitu poskytuje element typu **Progress Bar**. Najdeme ho na panelu paleta (Palette) vlevo a přetáhneme na kartu Hierarchy hned pod ním. Tím se přidá i do náhledu, kde ho lze libovolně přemisťovat a škálovat (obdélník vyznačený přerušovanou čarou představuje obrazovku).

Na kartě detailů v kategorii Style vložíme do položky **Background** texturu nebo barvu reprezentující prázdný počet života a do položky **Fill Image** naopak texturu nebo barvu pro plný počet. Níže, v kategorii Progress se nachází položka **Percent** určující plnost tohoto panelu. Jelikož chceme tuto hodnotu navázat na hráčův počet životů, zvolíme tlačítko **Bind** a následně **Create Binding**. Automaticky jsme přesměrováni do okna pro tvorbu nové funkce, jejímž float výstupem bude právě procento zaplnění panelu [3 str. 172] [16]. Graf funkce je na obrázku 21.



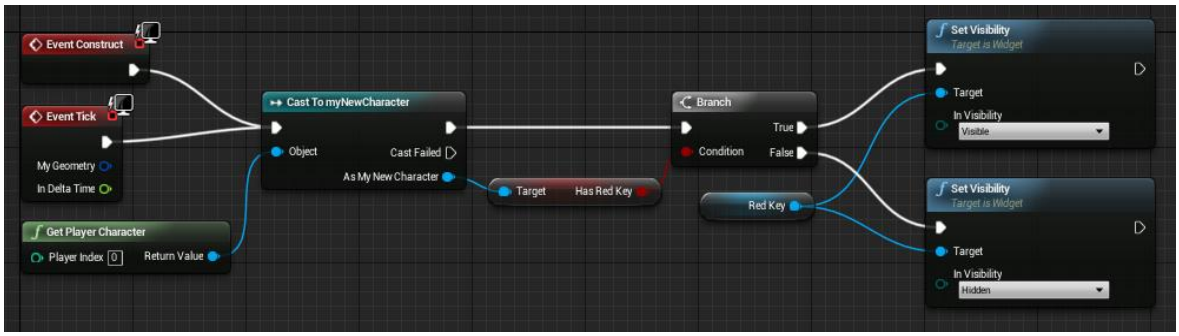
Obrázek 21

Funkce pro získání hráčova stavu života (Zdroj: autor)

Pozn. Uzel Get Player Character slouží k vybrání postavy definované jako defaultní v herním režimu.

Nyní přistoupíme k tvorbě inventáře. Bude se jednat pouze o indikátor, zobrazující, zdali má hráč klíč nebo ne. Vrátíme se tedy do okna Designer, na paletě najdeme položku **Image** a přetáhneme na kartu Hierarchy. Na kartě detailů následně vložíme v sekci Appearance do položky **Image** příslušnou texturu.

Nyní můžeme přepnout do okna Graph. Chceme vytvořit graf, který bude sledovat hráčovu postavu a zjišťovat, jestli má klíč (proměnná *hasKey*). Toto bude probíhat po celou dobu hraní, proto bude graf navazovat na událost Event Tick. Na základě obdržené hodnoty proměnné *hasKey* potom graf nastaví viditelnost elementu s klíčem pomocí uzlu Set Visibility. Na to slouží konkrétně položka Visibility uvnitř tohoto uzlu. Hodnota Hidden skryje element a naopak hodnota Visible ji zviditelní. Výsledný graf je na obrázku 22.



Obrázek 22

Nastavení zobrazování klíče (Zdroj: autor)

Funkcionalitu průhledového displeje již máme, potřebujeme ho však odněkud inicializovat. Otevřeme si proto hráčovu třídu a za graf navazující na událost Event Begin Play navážeme uzel Create Widget. Tento uzel slouží pro vytvoření nové instance jedné z tříd typu Blueprint Widget. Uvnitř uzlu nastavíme položku Class na název nedávno vytvořené třídy. Nakonec na oba výstupy tohoto uzlu napojíme uzel Add to Viewport. který tuto nově vytvořenou instanci přidá na hráčovu obrazovku.

10.2 Nabídky

Dalším krokem bude vytvoření klasických herních menu. Jelikož se některé tyto nabídky budou vzájemně referovat, vytvoříme si nejprve nové třídy pro každou z nich. Podobně jako v předcházející kapitole budou typu **Blueprint Widget**. Pro ovládání nabídek se v UE4 defaultně využívá myš, proto budeme muset také manipulovat se vstupy.

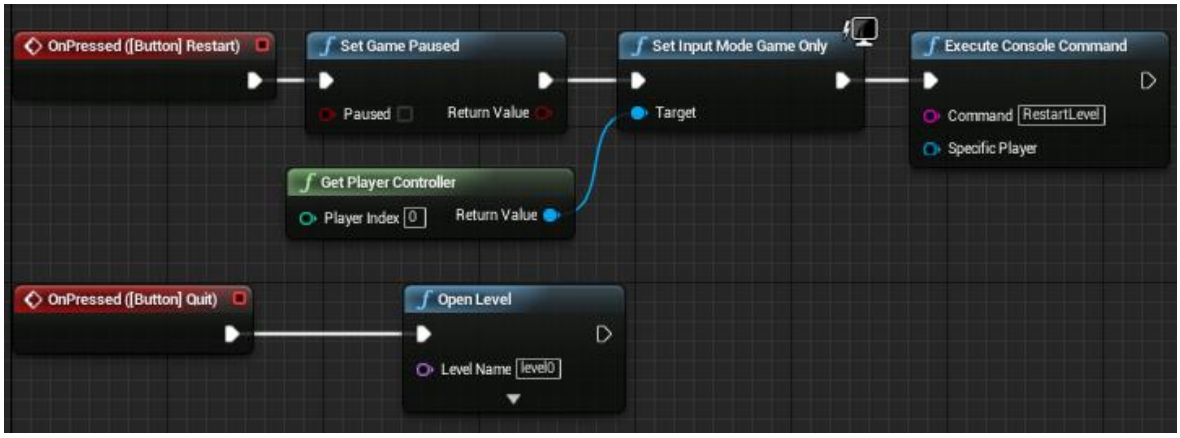
Než začneme, založíme další novou úroveň. Tato úroveň bude složit pouze pro hlavní menu a nemusíme do ní tedy vkládat žádné objekty. Nastavíme ji ale jako startovní úroveň po spuštění hry (postup je na straně 9).

Posmrtné menu

Jako první vytvoříme nabídku zobrazující se po zničení hráčovy postavy. Bude obsahovat tří položky – informaci že hráč zemřel, tlačítko pro restartování úrovně a tlačítko pro návrat do hlavní nabídky.

V designovém okně nejprve vytvoříme element **Vertical Box**, do kterého lze uspořádávat další elementy pod sebe. Toho dosáhneme tak, že na kartě Hierarchy přetáhneme elementy přímo na něj. Vložíme tedy element **Text** a dvě tlačítka (elementy typu **Button**). Oběma tlačítkům přiřadíme ještě jednoho následníka typu **Text**. K experimentování je na kartě detailů připravena spousta možností jak všechny tyto elementy řadit, obarvovat, umisťovat apod. Pro textové elementy lze také na řádku **Text** nastavit vlastní text, který budou zobrazovat.

Pro každé tlačítko nyní definujeme události při stisknutí. Tyto možnosti se nachází úplně dole na jejich kartě detailů. Konkrétně využijeme událost On Clicked a proto zvolíme příslušné zelené tlačítko se symbolem „+". Tím se v grafu událostí vytvoří nová událost, která bude inicializována, pokud dojde ke stisknutí tohoto tlačítka. Když jsou události pro obě tlačítka v grafu událostí, můžeme začít s definicí zbytku funkcionality. Grafy obou tlačítek jsou na obrázku 23.



Obrázek 23

Funkce tlačítek v posmrtném menu (Zdroj: autor)

Pro tlačítko navigující do hlavního menu stačí pouze navázat uzel Open Level s názvem naposledy vytvořené úrovně.

Restartovací tlačítko bude o poznání obtížnější, jelikož musí obsahovat následující funkce: zrušit pozastavení hry, nastavit hráčovy vstupy na herní a restartovat úroveň. K pozastavení/spuštění hry slouží uzel Set Game Paused. O jakou z těchto dvou akcí půjde, určuje jeho boolean vstup. Uzel Set Input Mode Game Only povolí hráčovy vstupy ovládající hru a naopak zakáže vstupy ovládající menu. [15] Jak už víme, uzel pro restartování úrovně neexistuje a využijeme proto uzel Execute Console Command.

Posledním krokem bude dát odněkud pokyn k vytvoření instance této nabídky. Nejjednodušší je to z hráčovy třídy, kde nahradíme stávající graf navazující na událost zničení postavy vytvořením tohoto menu, omezením hráčových vstupů pouze na ovládání této nabídky a pozastavením hry. Postup bude následující:

1. Novou instanci vytvoříme pomocí uzlu Create Widget a na oba výstupy navážeme uzel Add to Viewport.
2. Hráčovy vstupy omezíme pomocí uzlu Set Input UI Only (referujícím uzel Get Player Controller).
3. Hru pozastavíme uzlem Set Game Paused, ve kterém zatrhneme vstup Paused.

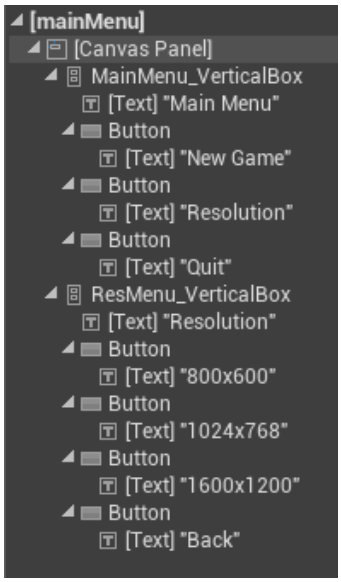
Hlavní menu

Hlavní menu bude fungovat na zcela stejných principech, avšak bude se skládat z více vrstev. Základní vrstva bude obsahovat tlačítka pro spuštění nové hry, ukončení hry a otevření podnabídky s nastavením rozlišení. Nabídka rozlišení bude obsahovat tlačítka pro nastavení hodnot rozlišení a tlačítko pro návrat do základní vrstvy. Hlavní menu samozřejmě může obsahovat spoustu dalších funkcí, nicméně pro demonstraci budou tyto bohatě stačit.

Postupujeme stejně jako u předchozího menu. Nejprve vytvoříme element **Vertical Box**, vložíme do něj tlačítka a na každé z nich vložíme další textový element. Když jsme hotoví, můžeme pomocí symbolu oka vpravo na panelu Hierarchy skrýt tuto část nabídky, aby se nepletla při další práci.

Dále vytvoříme další element **Vertical Box** pro nabídku rozlišení a na kartě detailů mu nastavíme položku **Visibility** na hodnotu **Hidden**. To znamená, že tento element bude defaultně skrytý. Následně ho opět naplníme tlačítky.

Výsledná hierarchie by měla připomínat tu na obrázku 24. [15]



Obrázek 24
Celková hierarchie hlavního menu (Zdroj: autor)

Nyní můžeme konečně přistoupit k programování jednotlivých tlačítek. Začneme od shora. Pro založení nové hry navážeme na událost On Clicked příslušného tlačítka uzel Open Level, do kterého napíšeme název první úrovně. Další a nejobtížnější bude tlačítko pro přechod do nabídky rozlišení, po jehož stisknutí musí dojít ke skrytí aktuální nabídky a zobrazení nové. Pro oba tyto kroky navážeme jeden uzel Set Visibility. První z nich skryje vertikální box základní vrstvy, zatímco druhý zviditelní box vrstvy s rozlišením. Pro tlačítko ukončení hry použijeme jednoduchý uzel Quit Game.

Zbývají tlačítka pro nastavení rozlišení. Pro ně stačí navázat již známý uzel Execute Console Command, jehož příkaz bude „*r.setRes + hodnota rozlišení*“ např. pro nastavení rozlišení na hodnotu 1600×900 to bude „*r.setRes 1600x900*“. [15]

Hlavní menu je nyní hotové a funkční. Zbývá pouze nastavit, aby úroveň hlavního menu toto hlavní menu zobrazovala. K tomu využijeme speciální třídu Level Blueprint, což je třída samotné úrovně. [15] Tuto třídu však nelze jen tak vytvořit, neboť se vytváří pouze automaticky při založení nové úrovně. Najdeme ji na panelu nástrojů pod **Blueprints → Open Level Blueprint**. Uvnitř této třídy navážeme na událost Event Begin Play naprosto stejnou sekvenci uzlů jako při vytváření instance posmrtného menu v předchozí kapitole.

Vítězné menu

Poslední je menu pro přechod do další úrovně, které se zobrazí po úspěšném dokončení aktuální úrovně. Tato funkcionalita se již ve hře nachází, v současné chvíli ji obsahuje třída zámek. Stačí tedy vytvořit nový objekt Blueprint Widget s jediným tlačítkem, jehož funkcí bude otevřít další úroveň. Nakonec zbývá změnit konec grafu ve třídě zámek, aby místo nové úrovně zastavil hru a otevřel tuto nabídku.

11 Export výsledné aplikace

Když jsme spokojeni se stavem naší hry, můžeme přistoupit k poslednímu kroku, kterým je osamostatnění a export. Abychom toho dosáhli, vybereme na hlavní nabídce **File → Package Project**. Zde máme na výběr z několika platforem, mějte však na paměti, že jsme definovali pouze klávesnicové vstupy. Jednu z platforem vybereme a zvolíme také cílový adresář.

Následně se v dolní části obrazovky zobrazí okno zobrazující průběh exportu. Tento proces může být u komplikovanějších her s mnoha importovanými soubory značně zdlouhavý, a proto běží na pozadí a umožňuje pokračovat v práci v editoru. Výsledkem bude adresář obsahující všechny potřebné soubory včetně jednoho spustitelného, kterým lze hru spustit. [3 str. 226]

Spuštění

Samostatnou hru najdeme v adresáři, jenž jsme nastavily při exportu, ve Windows to bude například *název projektu*.exe v adresáři *cílový adresář při exportu*\WindowsNoEditor.

Překvapující může být výsledná velikost projektu. Protože výsledkem kompilace z jazyka Blueprint je bajt kód virtuálního stroje UnrealScipt, je tento virtuální stroj ke každém projektu nutně „přibalen“. Sám o sobě zabírá téměř 200MB, což u menších projektů představuje většinu výsledné velikosti.

Pozn. Častým problémem je automatické blokování nové aplikace antivirovým programem, jelikož se jedná o neregistrovanou a tím pádem i potencionálně škodlivou aplikaci. Pokud se tedy po spuštění aplikace nic neděje, je třeba tuto ochranu vypnout nebo vytvořit výjimku.

12 Závěr

Cílem této práce bylo seznámit čtenáře s frameworkem Unreal Engine 4, rozhraním Unreal Editoru a jazykem Blueprint. Nejprve byl popsán samotný koncept herních frameworků, a současná situace na trhu. V dalších kapitolách byl popsán celý proces vývoje jednoduché dvourozměrné hry. Podařilo se vytvořit samostatnou funkční aplikaci a demonstrovat na ní jednotlivé oblasti vývoje jako vytváření úrovní, tvorbu zvukových efektů, nebo práci s animacemi. Důkladně byly rozebrány zejména pracovní postupy jazyka Blueprint, včetně tvorby uživatelských rozhraní.

Všechny cíle stanovené v úvodu považuji za splněné, ani tak bohužel nedošlo na všechny možnosti, které tento framework nabízí. Nicméně poskytnuté informace příp. uvedené informační zdroje by měli čtenáři umožnit pokračovat v samostatné práci v UE4, zejména pokud zůstane u dvourozměrného konceptu.

Za sebe mohu říct, že pro mě práce v UE4 byla příjemným překvapením. Moje předchozí zkušenosti s tímto softwarem byly zanedbatelné a teprve u této příležitosti jsem se s ním skutečně seznámil. Vývojové prostředí, zejména tvorba grafů je oproti psaní kódu mnohem uživatelsky přívětivější a po osvojení základních principů je implementace velmi zábavná. Velké množství nejrůznějších nastavení se může zpočátku zdát nepřekonatelné, vše je však celkem logicky uspořádané a dostatečná orientace lze získat poměrně rychle.

Přínosy práce

- Analýza současné situace nástrojů pro podporu tvorby her.
- Popis vybraných nástrojů UE4 relevantních pro dvourozměrné projekty.
- Kompletní návod pro tvorbu dvourozměrné hry v UE4.

Budoucnost ukázkové aplikace

Vyvíjená hra má především demonstrovat základní funkce UE4 a v tomto ohledu ji shledávám dostačující, nejedná se však o titul určený k distribuci, Pro tyto účely by hra potřebovala značně rozšířit, zejména počet úrovní, jelikož v této podobě zabere i nezkušenému hráči maximálně dvě hodiny k dokončení. Po rozsáhlejším rozšíření tohoto typu, by bylo také vhodné implementovat systém pro ukládání hráčova postupu, nebo možnost začít hru od jiné úrovně.

Terminologický slovník

Termín	Význam
Herní framework	Aplikace určená pro vývoj her. [2]
Úroveň	Části herního světa, vyžadující splnění nějakého úkolu, nebo dosáhnutí určitého bodu.
Objekt	Jeden konkrétní jedinec obsažený v herním světě.
Sprite	Dvourozměrný grafický objekt. [8]
Sprite-sheet	Textura obsahující v sobě několik objektů typu Sprite.
Blueprint	Objektově orientovaný programovací jazyk využívající k implementaci grafického rozhraní.
Datový typ	Obor hodnot, kterých smí proměnná nabývat.
Třída	Základní druh objektového datového typu. Představuje skupinu objektů, které nesou stejné vlastnosti.
Komponenta	Specifická funkcionalita, kterou lze přidat třídě. [15]
Uzel	Základní stavební jednotka grafu, reprezentující určitou funkci. Obsahuje vstupy a výstupy.
Událost	Reakce na okolnosti, ke kterým dochází během hry. Události slouží jako iniciátory grafů.
Uzlový graf	Množina uzlů a spojů definujících reakci třídy na událost, na kterou graf navazuje.
Casting	Forma komunikace mezi třídami v jazyce Blueprint. [15]
Kolize	Ke kolizi dochází při střetu kolizních tvarů dvou objektů ve hře. Kolizní odezvy jsou reakcemi na kolizní události. [17]
Uživatelské rozhraní (UI)	Rozhraní poskytující uživateli grafické prostředky komunikace s programem. [19]
Průhledový displej (HUD)	Speciální typ UI zobrazující aktuální informace (např. spočet životů) v průběhu hraní.

Seznam použitých zdrojů

[1] KUDLER, Amanda. Timeline: Video Games, Part I: Early Years. *Infoplease.* [Online] Sandbox Networks, Inc., publikováno jako Infoplease, © 2000–2015. [cit. 2015-12-18] Dostupné z: http://www.infoplease.com/spot/gamestimeline1.html.

[2] ENGER, Michael. Game Engines: How do they work? *Giant Bomb.* [Online] 20. 6. 2013. [cit. 2015-12-18] Dostupné z: http://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/.

[3] SATHEESH, PV. *Unreal Engine Essentials.* Birmingham : Packt Publishing, 2016. ISBN 978-1784391966.

[4] SEWELL, Brenden. *Blueprints Visual Scripting for Unreal Engine.* Birmingham : Packt Publishing, 2015. ISBN 978-1785286018.

[5] VALCASARA, Nicola. *Unreal Engine Game Development Blueprints.* Birmingham : Packt Publishing, 2015. ISBN 978-1784397777.

[6] SHAH, Ryan. *Master the Art of Unreal Engine 4 - Blueprints: Book #1 of 3.* místo neznámé : CreateSpace Independent Publishing Platform, 2014. ISBN 978-1500213107.

[7] Unreal Engine 4 Tutorials - 002 - 2D Sprite Animations. *Youtube.* [Online] 4. 3. 2014. [cit. 2016-02-03] Dostupné z: http://youtu.be/9FKG7fFWKOA. Kanál uživatele Muhammad A. Moniem.

[8] 2D Sidescroller with Blueprints. *Youtube.* [Online] 12. 9. 2014. [cit. 2016-02-03] Dostupné z: https://www.youtube.com/playlist?list=PLZlv_N0_O1gauJh60307mE_67jqK42twB. Kanál uživatele Unreal Engine.

[9] JASANI, Tejas. The top 10 engines that can help you make your game. *Venture Beat.* [Online] 20. 8 2015. [cit. 2015-12-29] Dostupné z: http://venturebeat.com/2014/08/20/the-top-10-engines-that-can-help-you-make-your-game/.

[10] EPIC GAMES INC. About Unreal Engine 4. *Unreal Engine.* [Online] © 2004-2016. [cit. 2016-01-15] Dostupné z: https://www.unrealengine.com/unreal-engine-4.

[11] THOMSEN, Mike. History of Unreal Engine. *IGN.* [Online] 23. 2. 2010. [cit. 2016-01-17] Dostupné z: http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine.

[12] UNITY TECHNOLOGIES. Unity, Game Engine, tools and multiplatform. *Unity, Game Engine.* [Online] © 2016. [cit. 2016-01-17] Dostupné z: https://unity3d.com/unity.

[13] CRYTEK GmbH. CRYENGINE | CryTtek. *CryEngine.* [Online] ©2016. [cit. 2016-01-18] Dostupné z: https://www.cryengine.com/get-cryengine.

[14] EPIC GAMES INC. Recommended Hardware. *Unreal Engine Community Wiki.* [Online] © 2004-2015. [cit. 2016-01-24] https://wiki.unrealengine.com/Recommended_Hardware.

[15] EPIC GAMES INC. Unreal Engine 4 Documentation. *Unreal Engine 4 Documentation.* [Online] © 2004-2016. [cit. 2016-01-05.] Dostupné z: https://docs.unrealengine.com/latest/INT/index.html.

[16] UE4 Tutorial - Health Bar + 2 Ways to Take Damage. *Youtube.* [Online] 16. 3. 2015. [cit. 2016-01-17] Dostupné z: http://youtu.be/5NuLsycksvU. Kanál uživatele Ben Ormstad.

[17] COE, Katy. 2D Platform Games Part 1: Collision Detection for Dummies. *Wordpress.* [Online] 18. 1. 2013. [cit. 2016-02-29] Dostupné z: https://katyscode.wordpress.com/2013/01/18/2d-platform-games-collision-detection-for-dummies/.

[18] WTF Is? Volume - Kill Z in Unreal Engine 4. *Youtube.* [Online] 3. 12. 2015. [cit. 2016-03-04.] Dostupné z: http://youtu.be/9_jIPZv5VM8. Kanál uživatele Mathew Wadstein.

[19] BEAL, Vangie. What Is User Iterface (UI)? Webopedia definition. *Webopedia.* [Online] © 2016. [cit. 2016-03-20] Dostupné z: http://www.webopedia.com/TERM/U/user_interface.html.

[20] Creating a 2D Side-Scroller. *Youtube.* [Online] 30. 4. 2015. [cit. 2016-02-03] Dostupné z: http://youtu.be/cCl1DHhIYeY. Kanál uživatele Unreal Engine.

[21] LERP - Linear Interpolation. *Youtube.* [Online] 2. 9. 2013. [cit. 2016-03-28] Dostupné z: http://youtu.be/0MHkgPqc-P4. Kanál uživatele Jamie King.

[22] YOYO GAMES LTD. Features | YoYo Games. *Make 2D Games Wit Game Maker | Yoyo Games.* [Online] ©2013-016. [cit. 2016-04-25] Dostupné z: http://www.yoyogames.com/gamemaker/features.

Seznam obrázků a tabulek

Seznam obrázků

Obrázek 1 Domovské oko editoru s defaultním rozložením (Zdroj: autor) 8
Obrázek 2 Rozdíl mezi perspektivní a ortografickou projekcí (Zdroj: script-tutorials.com) 10
Obrázek 3 Sprite-sheet (Zdroj: autor) 11
Obrázek 4 Příklad vhodné zdrojové textury (Zdroj: opengameart.org) 12
Obrázek 5 Část úrovně vytvořené pomocí dlaždic s přidanými předměty (Zdroj: autor) 14
Obrázek 6 Mapování vstupů (Zdroj: autor) 16
Obrázek 7 Dokončený graf pohybu (Zdroj: autor) 22
Obrázek 8 Nastavení herního režimu (Zdroj: autor) 23
Obrázek 9 Výčet Animací (Zdroj: autor) 24
Obrázek 10 Hotový graf funkce Animation (Zdroj: autor) 27
Obrázek 11 Graf pro animace v grafu událostí (Zdroj: autor) 29
Obrázek 12 Systém životů (Zdroj: autor) 32
Obrázek 13 Postup při zničení postavy (Zdroj: autor) 32
Obrázek 14 Kolizní odezva a komunikace mezi třídami (Zdroj: autor) 34
Obrázek 15 Uzel Timeline (Zdroj: autor) 36
Obrázek 16 Graf pro pohyb nepřítele (Zdroj: autor) 37
Obrázek 17 Nepřítel s kolizními tvary a referenčními body pohybu (Zdroj: autor) 38
Obrázek 18 Graf pro sebrání srdíčka (Zdroj: autor) 39
Obrázek 19 Graf třídy zámek (Zdroj: autor) 40
Obrázek 20 Sound Cue (Zdroj: autor) 42
Obrázek 21 Funkce pro získání hráčova stavu života (Zdroj: autor) 44
Obrázek 22 Nastavení zobrazování klíče (Zdroj: autor) 44
Obrázek 23 Funkce tlačítek v posmrtném menu (Zdroj: autor) 46
Obrázek 24 Celková hierarchie hlavního menu (Zdroj: autor) 47

Seznam tabulek

Tabulka 1: Typy uzlů 20
Tabulka 2: Datové typy 25

Příloha A: Další zdroje

Elektronická verze této práce, výsledná aplikace, zdrojové kódy a použité textury a zvuky jsou dostupné na adrese: http://bit.ly/26Khenp.