VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INOVATIVNÍ HERNÍ DEMO

INNOVATIVE GAME DEMO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ KAZÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Kazík Tomáš**
Program: Informační technologie
Název: **Inovativní herní demo**
Innovative Game Demo
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte metody procedurálního generování.
2. Navrhněte inovativní herní mechaniky založené na procedurálním generování.
3. Implementujte herní demo, kde demonstrujete navržené mechaniky.
4. Vytvořte video.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

1. Body 1, 2 a kostra aplikace

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 31. července 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Cílem bakalářské práce je vytvořit procedurální generátor, který vygeneruje herní mapu na základě schopností herní postavy. Práce zkoumá vlastnosti již existujících řešení – tedy her, jež obsahují procedurálně generované prostředí, anebo prostředí, k jehož zdolání jsou vyžadovány speciální schopnosti herní postavy. Následně se zabývá nezbytnými základy jak vývojového prostředí pro vývoj her, tak procedurálního generování. Na základě získaných informací se poté zaměřuje na návrh a implementaci herní postavy i procedurálního generátoru mapy.

Abstract

The aim of the Bachelor's thesis is to create a procedural generator, which is able to generate game map based on abilities of a game character. The thesis investigates features of already existing solutions – games, which contain procedurally generated environment, or the environment, which requires special abilities of a game character. Consequently, the thesis is engaged in the basics of both integrated development environment and procedural generation. Based on information acquired so far, the thesis is focused on designing and implementing the game character and the procedural map generator.

Klíčová slova

Herní demo, Procedurální generování, Generování herní mapy, Unity3D, C#

Keywords

Game demo, Procedural generation, Game map generation, Unity3D, C#

Citace

KAZÍK, Tomáš. *Inovativní herní demo*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Inovativní herní demo

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Kazík
30. července 2020

Poděkování

Chtěl bych poděkovat vedoucímu práce za podporu a čas, který mi věnoval při konzultacích a e-mailových konverzacích.

Dále bych chtěl věnovat velké díky své přítelkyni, snoubence a budoucí manželce Daniele, která mě podporovala i v těch nejtemnějších dnech, do kterých mě tato práce dostala.

Obsah

Kapitola 1

Úvod

V dnešní době, kdy herní průmysl zažívá svůj největší rozmach, začínají být na herní průmysl kladeny čím dál větší požadavky. S každým novým titulem je očekáváno, že bude blíže k reálnějšímu vyobrazení skutečného světa, a to za stejné, nebo ne moc odlišné náklady. Na společnosti vyvíjející hry je tak kladen nátlak, aby svou tvorbu zefektivnili – např. recyklováním již vytvořených herních modelů nebo automatizací procesu.

Jednou z možností automatizace procesu tvorby obsahu hry je procedurální generování obsahu hry. Jak je podrobněji popsáno v kapitole 4, způsobů, jak generovat herní obsah, je více – Perlinův šum, L-systém, základní generátor pseudonáhodných čísel a další možnosti, či kombinace jednotlivých možností mezi sebou. Za pomoci procedurálního generování lze dosáhnout automatizace tvorby libovolného obsahu hry – je-li tomu správně přizpůsoben algoritmus.

Tato práce se tedy zaměřila na možnost procedurálního generování herního světa, konkrétně herní překážkové mapy, a to na základě schopností herní postavy. Práce se nejprve zaměřuje na rozbor několika známých herních titulů, obsahujících schopnosti herní postavy a herní prostředí, které vyžaduje využití daných schopností – viz. kapitola 2. Kapitola 3 uvádí do vývojového prostředí Unity3D, ve kterém je herní demo procedurálního generování vyvinuto. V následující části práce je rozepsán návrh herní postavy, tedy možnosti, jak herní postavu vyvinout ve vývojovém prostředí Unity3D, a návrh a detailní popis jednotlivých schopností implementované herní postavy (kapitola 5). Dále je popsán návrh procedurálního generátoru herní mapy, tedy jeho kompozice z různých částí a kompozice generované herní mapy (kapitola 6), a návrh grafického rozhraní pro uživatele herního dema, tedy rozhraní pro testování a prezentaci herního dema (kapitola 7). V dalších kapitolách je popsána implementace herní postavy, tedy jejích základních pohybových a ovládacích vlastností a jejích schopností, včetně řešení problémů, se kterými se při jejich implementaci lze potýkat (kapitola 8). Následně je rozebrána implementace procedurálního generátoru herní mapy, tedy jednotlivých menších generátorů tvořících celek, průběh generování herní mapy a řešení problémů, se kterými se lze při implementaci setkat (kapitola 9). Dále je řešena implementace rozhraní pro testování a prezentaci herního dema a ukládání uživatelova nastavení jednotlivých generátorů herního dema (kapitola 10). V poslední kapitole (kapitola 11) jsou uvedeny příklady výsledných herních map vyprodukovaných procedurálním generátorem herního dema.

Kapitola 2

Design herní mapy s prvky překážek

V herním průmyslu existuje nespočet herních titulů, jež obsahují různé aspekty zdolávání překážek. Je zjevné, že ke zdolání daných překážek je vždy potřeba speciálních schopností herní postavy, nebo schopností hráče využít plný potenciál herní postavy, jež speciální schopnosti nemá. Cílem této práce je generovat mapu na základě schopností herní postavy, proto zkoumá pouze herní tituly, kde je kladen důraz na schopnosti herní postavy, ne hráče.

2.1 Team Fortress 2

Team Fortress 2 je týmová hra podžánru First-person shooter (střílečka z pohledu první osoby), kde proti sobě hrají dvě skupiny hráčů, jež mají patřičný úkol, který vede k výhře. Mezi zbraněmi, které tato hra obsahuje, je i raketomet, jehož střely při dopadu nepůsobí v určité oblasti pouze poškození, ale i explozivní sílu, která vše v blízkosti odmrští. Díky tomu vznikl tzv. Rocket jump, nebo-li technika, kdy hráč využívá explozivní síly raket pro mnohem větší a delší skoky. Navíc vlastnosti herní postavy této hry umožňují tzv. Air strafe, nebo-li měnit směr pohybu ve vzduchu.

Pro tyto vlastnosti herní postavy existuje i samostatný mód překážkových map, které nelze zdolat bez využití daných vylepšených pohybových možností. Takové mapy obsahují značně vzdálené, či vysoké, překážky na které má hráč doskočit, nebo převisy, u kterých má hráč za úkol se dostat na plošinu, nacházející se právě pod daným převisem.

U Team Fortress 2 bylo cílem sledovat právě možnosti módu překážkových map. Daný mód obsahuje několik typů map i pro další vlastnosti herní postavy, jako např. dvojitý skok, který hráči umožňuje skočit ve vzduchu ještě jednou. Nicméně středem zájmu byly mapy zaměřené na Rocket jump a Air strafe, a dané vlastnosti herní postavy samy o sobě.

Při zkoumání již zmíněného módu si lze povšimnout, že síla explozí raket se sčítá a hráč tak může doskočit ještě dál. Na druhou stranu nemá hráč sebemenší problém svůj směr pohybu okamžitě změnit další explozí. Při důkladnějším pozorování si lze povšimnout, že každá exploze udává směr a sílu, přičemž je-li směr shodný, tak se síla sčítá a je-li směr opačný, tak se původní síla ignoruje a aplikuje se nová síla exploze. Díky sčítání sil shodného směru lze dosáhnout nekonečné rychlosti, čímž hrozí, že bude přesáhnuta maximální hodnota datového typu (např. maximální hodnota datového typu integer v jazyce C# je 2 147 483 647), nicméně při takové síle by byl pohyb herní postavy tak rychlý, že by hráč nebyl schopen danou postavu ovládat – z toho lze vyvodit, že vzhledem k omezeným lidským

schopnostem hráče nelze dosáhnout větší hodnoty, než je maximální hodnota patřičného datového typu, a tím pádem při vývoji vlastností herní postavy není potřeba takovou situaci brát v potaz.

Součástí zkoumání hry Team Fortress 2 byla i další jednodušší schopnost herní postavy, a to Crouch – přikrčení se.

2.2 Ori and the Blind Forest

Ori and the Blind Forest je hra žánru platform-adventure (příběhová skákačka), kde hráč plní určité úkoly/cíle, které vedou k patřičnému výsledku – v tomto případě hráč zachraňuje les před zkázou. Hráč v průběhu hraní získává pro svou herní postavu nové schopnosti, jež jsou klíčové pro překonání patřičných překážek a pokračování ve hře.

U hry Ori and the Blind Forest bylo cílem sledovat právě schopnosti herní postavy a rozvržení mapy, která neumožňovala pokračovat ve hře v případě nevyužití patřičných schopností. Zkoumané schopnosti byly Double jump (dvojitý skok), Kuro's feather (plachtění) a Wall jump (skákání po zdech). Ve všech případech mapa obsahovala oblasti, kterých nebylo možné dosáhnout, aniž by hráč využil zmíněné schopnosti. V případě schopnosti Double jump bývaly překážkou různé propasti a oblasti, jejichž doskoková zóna byla příliš daleko a někdy i s mírným převýšením. Poté se na mapě vyskytovaly oblasti, které připomínaly komín – tyto oblasti byly určeny pro využití schopnosti Wall jump, kdy hráč měl za úkol skákat z jedné stěny na druhou, dokud se nedostal až na vrchol. Dále se zde nacházely oblasti, kde cíl, kam se měl hráč dostat, byl příliš daleko, ale hráč měl možnost skočit ze značného převýšení – v takovém případě měl hráč za úkol využít schopnost Kuro's feather. Hra obsahovala ještě mnoho dalších překážek a schopností pro jejich zdolání, ale pro účely této práce postačí již vyjmenované schopnosti.

Při detailnějším zkoumání jednotlivých schopností lze pozorovat, že herní postava je schopna při využití schopnosti Double jump skočit odlišným směrem, než kterým skočila prvním skokem. Dále je herní postava při využívání schopnosti Kuro's feather schopna upravovat směr, kterým zrovna plachtí, a doba plachtění je neomezená. U poslední zkoumané schopnosti, Wall jump, lze pozorovat, že hráč nemusí od zdi odskočit okamžitě, ale že se jí může držet a pomalu sjíždět dolů. Co lze také pozorovat, je to, že u schopnosti Wall jump může hráč lézt vzhůru i bez potřeby ode zdi, které se zrovna drží, odskočit – má možnost přímo po ní lézt.)

2.3 Dishonored

Dishonored je hra žánru action-adventure (příběhová akční hra), s příběhem osobního strážce, který se snaží zachránit císařskou dceru. V průběhu hraní má hráč možnost se naučit řadu schopností – jednou z nich je schopnost Blink (teleportace), kterou budeme u této hry zkoumat.

Schopnost Blink ve hře Dishonored se využívá tak, že hráč stiskne a drží patřičný vstup (např. tlačítko myši) a sleduje, kde se mu vyobrazí výsledná pozice – na kterou bude přemístěn, pokud se rozhodne teleportaci provést. Výsledná pozice teleportace se zobrazuje do určité maximální vzdálenosti. Poté, co je hráč spokojen s výslednou pozicí, pustí stisknutý vstup a jeho herní postava se přemístí na dříve vyobrazenou pozici. Schopnost Blink spotřebovává určitou energii, které má herní postava omezené množství. Tuto energii sice lze za pomocí patřičných předmětů doplnit, ale jejich množství je též omezené. Toto omezení ma-

ximálního počtu využití schopnosti Blink nutí hráče využívat danou schopnost obezřetně. Hráč má také možnost danou schopnost nevyužít, a to pomocí stisknutí patřičné klávesy, která ukončí zobrazování výsledné pozice teleportace a po puštění vstupu pro schopnost Blink se nic neprovede.

Po delším testování bylo pozorováno, že schopnost Blink lze využívat opakovaně po sobě bez jakékoliv prodlevy a také lze za její pomoci projít úzkými prostory – pokud je na konci úzkého prostoru dostatek místa pro herní postavu a pokud je konec úzkého prostoru blíže než maximální dosah schopnosti Blink.

Při zkoumání herní mapy lze pozorovat, že hra nabízí různá převýšení a vzdálená místa, na která se bez využití schopnosti Blink nelze dostat, neboť klasickým skokem herní postava ani zdaleka nedosáhne požadované výšky či vzdálenosti.

Kapitola 3

Unity3D

Pro účely bakalářské práce, tedy návrh a implementaci procedurálního generátoru mapy, byl zvolen engine Unity3D.

Unity3D je výkonný multiplatformní 3D engine, který je díky svému uživatelsky přívětivému prostředí jednoduchý pro začátečníky ve vývoji her, a zároveň, díky obsáhlému objemu nástrojů, výkonný pro experty. Lze v něm snadno vyvíjet hry a aplikace pro počítače, konzole i mobilní zařízení [1].

3.1 Herní engine

Tato podkapitola byla převzata z [2].

Herní engine je software, který znatelně usnadňuje vývoj videoher. Tento pojem se poprvé vyskytl v polovině devadesátých let, kde byl hojně využíván při vývoji her žánru FPS (First Person Shooter). Dobrým příkladem je společnost Id Software, která vyvinula hru Doom, jež měla od náplně hry (herní prostředí, mechanismy, speciální efekty atd.) specifickým způsobem oddělené jádro, které právě řešilo základní obecné problematiky, jako vykreslování grafiky, detekci kolizí, audio systém atd. Toto jádro bylo možné využívat opakovaně při vývoji dalších her, čímž se dosáhlo rychlejšího a levnějšího vývoje, neboť společnost se již nemusela soustředit na nízkoúrovňové aspekty hry – ty za ně řešilo právě jádro – a mohla se soustředit čistě na náplň hry. Společnost Id Software poté toto jádro i licencovala dalším firmám.

Dnešní herní enginy jsou mnohem sofistikovanější, ale i tak zůstává jejich princip stále stejný. Herní engine je znovupoužitelnou částí hry, která poskytuje základní i pokročilejší funkcionality, které mají hry, nebo aspoň většina z nich, společné. Herní engine tak tvoří nedílnou součást hry, kdy určit hranici mezi enginem a hrou samotnou je až nereálné.

3.2 Možnosti vývoje herní postavy v Unity3D

V engine Unity3D lze vyvinout herní postavu dvěma způsoby - za pomocí využití komponenty Rigidbody, nebo komponenty CharacterController. Každý ze způsobů má své výhody a nevýhody – nelze tedy říct, který z nich je za všech okolností nejlepší. Způsobů, jak řešit vývoj herní postavy, je samozřejmě více – neboť vývojář si dle své libosti může zvolit, že např. vyvine svoji herní postavu od naprostých základů. Nicméně v této bakalářské práci se zaměříme na již zmíněné dva způsoby.

Prvním způsobem vývoje herní postavy je komponenta zvaná Rigidbody. Tato komponenta obnáší komplexní fyzikální vlastnosti a chování – jako např. gravitaci. K tomu, aby bylo možné tuto komponentu plně využít, je potřeba, aby herní objekt obsahoval i komponentu zvanou Collider – tato komponenta ohraničuje objekt a detekuje kolize s ostatními objekty. Dále je potřeba, aby objekt obsahoval skript s kódem, který aplikuje různé síly na Rigidbody a způsoboval tak pohyb objektu – např. metodou AddForce.

Dalších způsobem vývoje herní postavy je komponenta zvaná CharacterController. Herní postava se na základě této komponenty sama o sobě nikterak nehýbe, neboť na ní neúčinkují žádné síly (gravitace, kolize) – jak tomu je u komponenty Rigidbody. I v tomto případě je potřeba, aby objekt obsahovat skript s kódem, který by pracoval s komponentou CharacterController, u které by předával informaci, o kolik a jakým směrem se má herní objekt přemístit. Daná komponenta přesun zařídí berouc v potaz kolize – v takovém případě lze využít např. metodu Move. V tomto případě není potřeba, aby herní objekt obsahoval komponentu Collider, neboť komponenta CharacterController je zároveň i komponentou Collider.

3.3 Tvorba mapy

Mapu hry, nebo-li herní prostředí, lze v Unity3D vytvořit mnoha způsoby. Mezi nejčastější způsoby patří předvytvořené herní prostředí v (libovolném) grafickém editoru, tvoření herního prostřední v Unity3D za využití jednotlivých předpřipravených objektů – jež byly vytvořeny v grafickém editoru – a dynamické generování a upravování mapy, kdy se využívají základní objekty, které jsou dynamicky modifikované a klonované.

3.4 Prefab

Prefab systém, jenž je součástí Unity3D, umožňuje vývojáři vytvářet, konfigurovat a ukládat herní objekt (GameObject) se všemi jeho komponentami, vlastnostmi a dalšími herními objekty, jež jsou zařazené jako potomky tohoto objektu. Prefab funguje jako předloha, kterou lze kopírovat/klonovat do herního prostředí jako nový objekt, který lze modifikovat bez ovlivnění daného prefabu – naopak modifikace prefabu se projeví u všech kopií v herním prostředí.

3.5 Singleton

Návrhový vzor Singleton je způsob, jak zajistit, že bude existovat pouze jedna globálně přístupná instance specifické třídy. V mnoha ohledech se tato třída chová jako statická, ale s určitými výhodami – podporuje implementaci rozhraní, dědění stříd a lépe se využívá jako komponenta herního objektu v Unity3D (např. při potřebě navázání referencí na jiné herní objekty).

Kapitola 4

Procedurální generování obsahu

Procedurální generování obsahu v herním průmyslu znamená automatizovanou tvorbu herního obsahu pomocí algoritmů. Jedny ze známých příkladů využití jsou hry Rogue (A.I. Design 1980) a Diablo (Blizzard 1996) pro generování Dungeonů[1], dále hry Civilization (MicroProse 1991) pro generování mapy, Borderlands (Gearbox 2009) pro generování zbraní a software SpeedTree (Interactive Data Visualization 2009) pro generování vegetace.

Přesněji řečeno, pojem Procedurální generování obsahu by se dal chápat jako algoritmická tvorba herního obsahu na základě určitého, možná nepřímého, vstupu uživatele. Tato tvorba může být nahodilá, adaptivní (vůči aktuálnímu stavu hry), ale také nemusí být ani jedno z toho [4].

Mezi nejčastěji využívané metody, na základě kterých se zmíněné algoritmy rozhodují co, kde a jak vygenerují, patří Perlinův šum, L-systém a v některých případech i základní generátor pseudonáhodných čísel.

4.1 Perlinův šum

Perlinův šum je generátor pseudonáhodných čísel, jejichž posloupnost, oproti klasickému generátoru pseudonáhodných čísel, působí více přirozeně a uspořádaně. V případě generování Perlinova šumu v dvourozměrném prostoru může výsledná křivka připomínat funkci cosinus s poměrně hladkým průběhem [5]. Oproti tomu klasický generátor pseudonáhodných hodnot působí chaoticky – ačkoliv lze též pozorovat křivku připomínající funkci cosinus.

Perlinův šum lze vyhodnocovat v libovolném počtu dimenzí, nicméně Unity3D obsahuje strukturu Mathf obsahující metodu PerlinNoise, která umožňuje pracovat právě s dvourozměrným Perlinovým šumem [6].

4.2 L-systém

L-systém (Lindenmayer-system – pojmenováno po Aristidu Lindenmayeru) umožňuje definici různých komplexních tvarů pomocí iterace. Systém využívá matematický jazyk, kde je porovnán počáteční řetězec znaků s pravidly, která jsou vyhodnocována opakovaně. Výsledek každého hodnocení se stává základem pro další iteraci vyhodnocení. Výsledek poslední iterace je využit ke generování geometrických tvarů [7].

[1] Dungeon – v herním prostředí je takto označován žánr nebo pasáž hry, kdy je prostředí situováno do podzemních katakomb, labyrintů, jeskyní nebo jiných podobných kulis [3]

L-systém se pro účely této bakalářské práce ukázal jako nevhodný. Jeho prioritní využití je generování plošných i trojrozměrných modelů stromů, keřů, travin a dalších přírodních útvarů [8]. Vzhledem k tomu, že cílem bakalářské práce je procedurální generátor mapy, jež bude překážkovou dráhou, pro L-systém se nenaskytuje žádné řádné využití.

4.3 Generátor pseudonáhodných čísel

Tato podkapitola byla převzata z [9].

Klasický deterministický počítač není schopen generovat čistě náhodná čísla, neboť se řídí přesnými pravidly. To se v praxi obchází generováním tzv. pseudonáhodných čísel. Tato čísla jsou sice počítaná předvídatelným způsobem, ale jelikož mají charakter náhodného šumu, tak jsou ve většině případů dostačující. V případě, že jsou požadována čistě náhodná čísla, je potřeba využít procesor, který obsahuje obvody, jež na základě zákonů kvantové mechaniky taková čísla umí generovat. Nicméně tyto procesory jsou pomalejší a čistě náhodná čísla nejsou běžně potřeba.

Typickou procedurou generující hodnoty v intervalu, jakožto základ generování pseudonáhodných čísel, je tzv. kongruentní generátor (jednou z alternativ je Mersenne twister), fungující na základě rovnic:

$$X_0 = Z \tag{4.1}$$

$$X_{n+1} = (A * X_n + C) \mod M \tag{4.2}$$

Jedná se o vzorec posloupnosti hodnot, kde se první hodnota posloupnosti X_0, která představuje tzv. seed, nastaví na zvolenou hodnotu Z (viz. vzorec 4.1) a každá další hodnota posloupnosti se vypočítá pomocí vzorce 4.2. Proměnné A, C a M jsou konstanty, které ovlivňují výsledek výpočtu a je potřeba je nastavit právě tak, aby se dosáhlo rovnoměrného rozložení výsledných hodnot. Konstantě M je potřeba věnovat zvláštní pozornost, neboť ta je tzv. periodou, která určuje, že nejpozději po m krocích se generovaná posloupnost začne opakovat – proto je potřeba, aby hodnota konstanty M byla vysoká.

"Je zřejmé, že generátor bude vždycky vytvářet stejnou posloupnost v závislosti na počáteční hodnotě (seedu). Pro každé spuštění aplikace tedy dostaneme vždycky stejný výsledek, což se někdy, např. při simulacích, hodí a jindy, třeba ve hrách, ne. Pro dosažení různosti posloupností se proto typicky jako seed používá hodnota založená na aktuálním čase – jelikož program spouštíme v různých časech, generovaná posloupnost bude pokaždé jiná."[9]

4.4 Procedurální generování mapy

Mezi nejčastěji procedurálně generované objekty v herním průmyslu patří herní mapy – a to ať už krajiny, terény, nebo logické rozmístění místností a podzemních chodeb.

V případě generování terénu se nejčastěji využívá Perlinův šum, který má vlastnosti cosinu, tedy vlnění se. Perlinův šum při generování terénu lze využít i více krát – např. třikrát, kdy se první vyhodnocení využije pro vytvoření základní linie terénu, pomocí druhého vyhodnocení se vygenerují velké kameny a pomocí třetího vyhodnocení menší kameny, štěrk [10]. Pokud je při generování terénu generována i krajina, lze využít i L-systém pro generování vegetace.

V případě generování místností, jejich rozmístění a generování podzemních chodeb, které místnosti propojují, se nejčastěji využívá obyčejný generátor pseudonáhodných čísel. Při

generování mapy tohoto typu je na vývojáři, jakým způsobem generování náhodných hodnot využije. Jedním z možných způsobů je randomizovat počet místností, jejich rozměry a jejich polohy, a pak je pomocí nějakého algoritmu propojit chodbami – např. pomocí algoritmu A star.

4.5 Příklad her s procedurálně generovaným obsahem

Her s obsahem, který je procedurálně generovaný, je nespočet. Tato práce se zaměřuje na jedny z nejznámějších.

4.5.1 The Elder Scrolls V: Skyrim

„The Elder Scrolls V: Skyrim je akční hra na hrdiny (RPG) s otevřeným světem vytvořená společností Bethesda Game Studios a vydaná společností Bethesda Softworks. Jedná se o páté pokračování série akčních fantasy her na hrdiny The Elder Scrolls, které následuje po předchozím dílu The Elder Scrolls IV: Oblivion. Skyrim vyšel 11. listopadu 2011 na platformy PC (Microsoft Windows), PlayStation 3 a Xbox 360.“ [11]

Chování draků, kteří se ve hře vyskytují, je procedurální – tím, že si vybírají z předdefinovaných možností chování. Taktéž některá setkání s draky jsou procedurálně generovaná a náhodná – jedná se o setkání, ve kterém drak, kterého hráč při daném setkání střetne, nemá jméno [12].

4.5.2 Descenders

Descenders je akční cyklistická hra vyvinutá Holandskou společností RageSquid a vydaná společností No More Robots. Hra byla vydána 5. května 2019 na platformy PC (Microsoft Windows) a Xbox One [13].

Svět, ve kterém hráč na svém sportovním kole sjíždí závodní tratě, je procedurálně generovaný – tzn. terén, krajina, tratě a překážky jsou procedurálně generované a hra tak dosahuje zdánlivě nekonečného množství závodních tratí.

4.5.3 Enter the Gungeon

Enter the Gungeon je hra žánru bullet hell[2] roguelike[3] vyvinutá společností Dodge Roll a vydaná společností Devolver Digital. Hra byla vydána 5. dubna 2016 na platformy PC (Microsoft Windows, OS X, Linux) a PlayStation 4, 5. dubna 2017 na Xbox One a 14 prosince 2017 na Nintendo Switch.

Ve hře je několik pater, kterými hráč musí projít, aby hru dohrál. Každé patro má náhodný počet místností. Zatímco místnosti jsou předdefinované, jejich rozmístění v patře, a stejně tak i rozmístění nepřátel a pokladů v místnostech, je procedurálně generováno [16].

4.5.4 No Man's Sky

No Man's Sky je hra žánru exploration survival[4] vyvinutá a vydaná společností Hello Games. Hra byla vydána v srpnu 2016 na platformy PlayStation 4 a PC (Microsoft Windows) a července 2018 na Xbox One.

[2] Bullet hell – hra, ve které je hrací obrazovka často téměř celá zaplněná nepřátelskými střelami [14]
[3] Roguelike – hra podobná hře Rogue z roku 1980 [15]
[4] Exploration survival – hráč prozkoumává a přežívá v neznámém světě

Charakteristickým rysem hry No Man's Sky je to, že téměř celá galaxie je procedurálně generována na základě jediné hodnoty (Seed) – včetně hvězd, planet, flory a fauny na daných planetách a setkání s mimozemským životem [17].

Kapitola 5

Návrh herní postavy

Před samotným vygenerováním mapy, jež je založeno na schopnostech herní postavy, je třeba si danou herní postavu nejprve definovat. Součástí definice jsou jak základní vlastnosti herní postavy, tzn. pohyblivost a ovládání, tak i její schopnosti.

Práce se zaměřuje nejprve na možnosti, které Unity3D pro vývoj a ovládání postav nabízí, posléze upřesňuje, která možnost a proč byla zvolena, a následně definuje určité vylepšení na základě UX. Na závěr jsou definovány schopnosti herní postavy.

5.1 Unity3D

Jak již bylo zmíněno v předchozích kapitolách, Unity3D nabízí několik možností, jak lze implementovat herní postavu – Rigidbody, CharacterController, vlastní skripty.

Komponenta Rigidbody je založená na fyzikálních vlastnostech a umožňuje vývojáři pracovat s herní postavou pomocí fyzikálních sil. Komponenta dále umožňuje nastavit, které fyzikální síly se mají ignorovat.

CharacterController je naopak primitivnější nástroj správy herní postavy, neboť nereaguje na žádné působení sil – ani gravitaci. V případě, že vývojář požaduje, aby se herní postava někam přemístila, předá komponentě CharacterController informaci o tom, jakým směrem a jak moc se má přesunout, a komponenta daný požadavek vykoná, berouc v potaz případné kolize. V případě, že vývojář chce, aby na herní postavu působila setrvačná síla, musí ji sám průběžně přepočítávat a výsledek předávat komponentě CharacterController.

5.1.1 Rigidbody

Rigidbody je komponenta, která umožňuje správu polohy objektu pomocí fyzikální simulace – jak je popsáno dále. Důvod, proč bylo rozhodnuto, že pro účely této práce bude využito právě komponenty Rigidbody, místo např. komponenty CharacterController, je ten, že jedna z plánovaných schopností herní postavy je tzv. Rocket jump. Detailní popis techniky Rocket jump je obsažen níže v návrhu této schopnosti, nicméně základní vlastností této schopnosti je využívání explozivní síly raket pro dlouhé skoky. Právě tato explozivní síla, a hlavně její předávání herní postavě, byla hlavním důvodem pro volbu komponenty Rigidbody. Dalšími důvody volby komponenty Rigidbody bylo působení gravitační a setrvačné síly na komponentu.

Je-li objektu přidána komponenta Rigidbody, jeho pohyb začne být pod kontrolou fyzikálního engine Unity3D (část engine Unity3D, která se zaměřuje na fyzikální vlastnosti objektů). Aniž by byl přidán jakýkoliv další kód, objekt se začne okamžitě pohybovat, neboť

na něj začne účinkovat gravitační síla, a pokud bude objekt obsahovat i komponentu Collider, řešící kolize objektů, bude reagovat na veškeré kolize s ostatními objekty, jež obsahují komponentu Collider.

Rigidbody má také skriptovací API, které umožňuje aplikovat na objekt síly a ovládat ho fyzikálně realistickým způsobem – např. chování vozidla lze simulovat aplikováním sil na jeho kola. Na základě těchto informací je fyzický engine schopen simulovat další aspekty simulace pohybu auta, takže je auto schopné realisticky zrychlovat a detekovat kolize.

Využívání zmíněného API je vhodnější v metodě FixedUpdate, neboť tato metoda je vyhodnocována těsně před každou aktualizací fyziky, takže veškeré provedené změny budou zpracovány okamžitě [6].

5.1.2 CharacterController

Druhou nejčastěji používanou komponentou pro řízení pohybu herní postavy je CharacterController. Tato komponenta umožňuje jednoduchou správu pohybu herní postavy, nicméně neobnáší žádné komplexní výpočty. V případě, že vývojář požaduje, aby byl pohyb postavy komplexnější – např. aby reagovala na sílu nárazu, která ji odmrští – musí danou komplexnost naimplementovat sám.

Vzhledem k tomu, že jednou z plánovaných schopností je Rocket jump, který vyžaduje náročnější výpočty reakce na explozivní sílu a následně výpočty setrvačné síly, se komponenta CharacterController projevila jako nevhodná volba. Proto, jak již bylo zmíněno výše, bylo pro účely této bakalářské práce rozhodnuto pro komponentu Rigidbody [6].

5.2 UX - ovládání postavy

Jak bude hra vnímat vstupní signály od hráče a jak na ně bude reagovat, se liší žánr od žánru hry. Tato bakalářská práce se zaměřuje pouze na oblast pohybu herní postavy, neboť zkoumání ostatních oblastí by bylo pro tuto práci nevyužitelné, tedy zbytečné. Oblast pohybu herní postavy, na kterou se práce zaměřuje, je poměrně obecná, neboť UX úpravy, které práce aplikuje, se běžně aplikují ve všech hrách, které dané činnosti umožňují.

Mezi nejčastější problémy s pohybem herní postavy patří pocit hráče, že hra v patřičných případech nereaguje na jeho vstupní signál ke skoku. Běžnou podmínkou pro uskutečnění skoku herní postavy je, že postava musí detekovat, že stojí na zemi. Klíčový problém, proč postava nereaguje na vstupní signál ke skoku, jsou právě krajní meze, kdy postava ještě, nebo již, nedetekuje, že stojí na zemi. Jsou to situace, kdy hráč dá vstupní signál ke skoku buď chvíli před dopadem herní postavy, nebo chvíli poté, co herní postava opustí zem (např. sejde z hrany jiného objektu). Zmíněná chvíle je v řádech desítek, popřípadě nižších stovek, milisekund. Ačkoliv hráč dává signál ke skoku v nevhodnou chvíli, je potřeba takový případ podchytit a jedná-li se o krajní mez, skok herní postavy je třeba umožnit.

Umožnění skoku herní postavy i ve chvíli, kdy se nedotýká země – jedná-li se o příjem vstupního signálu ke skoku v krajních mezích – se řeší tak, že si hra po určitou dobu pamatuje, že hráč vydal vstupní signál pro skok a stejně tak si po určitou dobu pamatuje, že herní postava se dotýkala herní plochy.

5.2.1 Skok po opuštění plochy

Jedná se o situaci, kdy herní postava sejde z hrany herní plochy a hráč v daný moment vydá signál k výskoku. Z pohledu hráče se jedná o výskok v poslední moment – aby dosáhl maximální efektivity a co nejdelšího skoku – ačkoliv herní mechanismus vyhodnotil, že herní postava se na herní ploše již nenachází. Za normálních okolností, bez žádných úprav, by se žádný skok neuskutečnil a herní postava by začala padat, zatím co hráč by byl zklamán a měl by pocit, že chyba je na straně hry, neboť signál k výskoku byl z jeho pohledu vydán včas.

Jedním z možných řešení je dočasná paměť, že herní postava se nacházela na herní ploše. Přesněji řečeno se herní postava po určitou dobu poté, co z herní plochy spadla, chová, jako by se stále na herní ploše nacházela. Tento odklad změny stavu, kdy se herní postava na herní ploše již nenachází, by neměl být příliš velký. Bude-li odklad znatelný, hráč si jej pak může všimnout, neboť za takových okolností lze již poznat, že herní postava po určitou dobu poté, co sejde z herní plošiny, může stále skákat – tedy že může skákat uprostřed vzduchu. Pro příklad si lze představit odklad o jednu celou vteřinu – v takovém případě by se jednalo o velmi znatelný odklad. Odklad by měl tedy dosahovat maximálně 200 milisekund, kdy už začíná být možné si jej všimnout. Na základě testování byl pro účely této bakalářské práce zvolen odklad o 100 milisekund.

Možností, jak řešit odklad zaznamenání, že se herní postava již nenachází na herní ploše, nýbrž je ve vzduchu, je více. Nejčastějším, jakožto nejsnazším řešením je, že dokud se herní postava nachází na herní ploše, je ukládán do paměti aktuální čas – v případě, že se herní postava nachází ve vzduchu, ukládaný čas se již neaktualizuje – poté, když je přijat signál ke skoku, se zkontroluje, zda odchylka aktuálního času od onoho uloženého je ve zvolené mezi přijatelná, a v případě, že ano, provede se skok. Toto řešení je jednoduché na implementaci, ačkoliv náročnější na výpočetní výkon, neboť každou iteraci cyklu vyhodnocování aktuálního stavu hry se ukládá do paměti aktuální čas, a navíc, při přijetí signálu pro skok se pro jeho vyhodnocení porovnávají dvě proměnné datového typu float.

Pro účely této bakalářské práce bude pro odklad změny stavu herní postavy při sejití z herní plochy využíváno funkcionality, jež poskytuje Unity – jedná se o Coroutines, které umožňují práci v novém vlákně, a proto jsou ideální pro různé funkcionality obnášející časovač [6]. V případě, že herní postava sejde z herní plochy, pokusí se vydat signál, jenž nastaví proměnou, která spravuje stav herní postavy na herní ploše, na hodnotu, která vyjadřuje, že se herní postava na herní ploše již nenachází. Nicméně tento signál spustí tzv. Coroutine, jež Unity poskytuje, která se na zvolenou dobu přeruší (v tomto případě byla zvolena odmlka na 100 milisekund), a teprve pak nastaví proměnnou, spravující stav herní postavy na herní ploše, na hodnotu, jež vyjadřuje, že herní postava se na herní ploše nenachází, tedy je ve vzduchu. V tomto případě není potřeba při každém cyklu vyhodnocování stavu hry ukládat do paměti aktuální čas, pouze, změní-li se stav, se spustí tzv. Coroutine, která se po určitou dobu odmlčí, tzn. po danou dobu se nic neprovádí, a tím pádem dochází ke snížení výpočetní náročnosti, a teprve poté se změní hodnota spravující stav herní postavy na herní ploše. V případě přijetí signálu ke skoku již není potřeba porovnávat dva časy (aktuální čas s uloženým časem posledního dotyku herní postavy herního povrchu), ale stačí pouze zkontrolovat aktuální stav herní postavy vůči herní ploše – algoritmus tedy místo porovnání dvou proměnných datového typu float porovnává dvě proměnné datového typu Boolean. Toto řešení je sice náročnější na implementaci, ale výsledkem je menší výpočetní zátěž.

5.2.2 Skok před dopadem

V tomto případě se jedná o situaci, kdy hráč krátký moment před dopadem na herní plochu vydá signál ke skoku. Pokud budeme brát v potaz podobnou situaci, jaká je popsána v kapitole 5.2.1 – situace, kdy si herní postava udržuje informaci o tom, zda se dotýká herní plošiny, neboť skákat může pouze tehdy, pokud na herní plošině stojí – tak pokud hráč chvíli před tím, než herní postava dopadne na herní plochu, vydá signál ke skoku, herní mechanismus vyhodnotí, že herní postava se na herní ploše nenachází, a skok tedy neprovede. Z pohledu herního mechanismu je toto plně korektní, nicméně hráč má pocit, že se signálem ke skoku počkal dostatečně dlouho a nevydal ho brzy – je tedy frustrován, když herní postava nereaguje a nevyskočí.

Podobně jako v předchozím případě kapitoly 5.2.1 je jedním z možných řešení dočasná paměť, že hráč vydal signál ke skoku – tedy po určitou dobu poté, co hráč vydal signál, se herní mechanismus chová, jak kdyby signál ke skoku byl vydán každý cyklus vyhodnocení aktuálního stavu hry – pokud se herní postava do vypršení časového limitu ocitne na herní ploše, tak je signál k výskoku přijat a skok proveden. Tato opožděná reakce na hráčův vydaný signál ke skoku by neměla být až příliš opožděná, neboť by pak hráč nabýval pocitu, že hra nereaguje korektně, že se naopak seká, či opožďuje. Odklad by měl dosahovat maximálně opět 200 milisekund, kdy už začíná být znatelné opoždění. Na základě testování byl pro účely této bakalářské práce zvolen odklad o 100 milisekund.

Nejčastější způsob, jak se možnost opožděné reakce na signál skoku implementuje je, že herní mechanismus si zapamatuje čas, kdy byl daný signál vydán. Poté, co herní postava dopadne na herní plochu, je porovnán aktuální čas s časem, kdy byl signál vydán, a je-li to v patřičné mezi, je signál přijat a skok proveden. V takovém případě již není možné vyhodnocovat signál výskoku pouze tehdy, je-li přijat, je potřeba jej vyhodnocovat každý cyklus vyhodnocování aktuálního stavu hry – tzn. každý cyklus vyhodnocování aktuálního stavu hry se porovnává aktuální čas s uloženým časem posledního signálu skoku – a to je, oproti řešení, které bylo v této bakalářské práci zvoleno, náročnější na výkon, ačkoliv jednodušší na implementaci.

Podobně jako v předchozí kapitole 5.2.1, i v tomto případě bylo využito funkcionality, jež poskytuje Unity – Coroutines [6]. V případě, že hráč vydá signál ke skoku, spustí se zmíněná Coroutine, jež Unity poskytuje, která uloží do proměnné, že hráč vydal signál ke skoku a herní postava se teda snaží vyskočit, poté se na zvolenou dobu přeruší (v tomto případě byla zvolena odmlka na 100 milisekund) a následně obsah zmíněné proměnné změní – uloží zde, že hráč již nevydává signál ke skoku a herní postava se již nesnaží vyskočit. Pokud v průběhu odmlky se herní postava naskytla na herní ploše, je zmíněná proměnná změněna předčasně a herní postava vyskočí. V tomto případě se sice stále signál ke skoku vyhodnocuje každý cyklus vyhodnocení aktuálního stavu hry, ale již není potřeba porovnávat proměnné typu float, stačí porovnat proměnné typu Boolen – což je jednodušší, tedy méně náročné na výkon. Toto řešení je sice náročnější na implementaci, ale výsledkem je menší výpočetní zátěž.

5.3 Návrh schopností herní postavy

Vzhledem k tomu, že procedurální generátor herní mapy, jež je cílem této práce, bude danou mapu generovat na základě schopností herní postavy, tak je potřeba si dané schopnosti nejprve specifikovat a blíže popsat. Všechny navržené schopnosti jsou inspirovány známými tituly – viz. kapitola 2. Dané schopnosti byly pečlivě vybírány tak, aby za jejich pomoci bylo možné zdolávat překážky, které by bez jejich pomoci nebylo možné zdolat.

5.3.1 Air strafe - pohyb ve vzduchu

Air strafe je schopnost, která umožňuje herní postavě měnit směr pohybu ve vzduchu – např. při výskoku. Hráč tak může skákat za roh nebo obskočit celou překážku. V případě této bakalářské práce bylo rozhodnuto, že schopnost Air strafe bude řešena působením sil na herní postavu, přesněji na herní komponentu Rigidbody, jež je dodána vývojovým prostředím Unity3D. Komponenta vyřeší výpočet působení sil na herní postavu a vyhodnotí její nový směr pohybu. Sílu, kterou se na herní postavu působí pro změnu směru pohybu, bude možné upravovat v inspektoru editoru Unity3D – v okně editoru, které obsahuje detail herního objektu, jež má vývojář právě označený. Na změnu této síly bude generátor herní mapy reagovat – tzn. při dalším generování vytvoří mapu, jež bude brát v potaz aktuální sílu schopnosti Air strafe.

5.3.2 Crouch - krčení se

Crouch je schopnost, která umožňuje herní postavě přikrčit se a projít nižšími prostory. V podstatě jde o to, že rozměry herní postavy se na výšku zmenší. V různých hrách se poměr zmenšení liší. Pro účely této bakalářské práce se bude herní postava zmenšovat o polovinu své výšky. Při vývoji v herním prostřední Unity3D stačí zmenšit rozměry kořenového herního objektu, na který jsou navázané ostatní herní objekty a komponenty herní postavy – vývojové prostředí změnu rozměrů všech ostatních objektů a komponent zařídí. Poměr, o kolik se má herní postava zmenšit, bude opět přístupný skrze inspektor vývojového prostředí. V případě, že se poměr zmenšení herní postavy při skrčení změní, při příštím generování herní mapy bude změna brána v potaz a nová mapa bude přizpůsobena novému poměru.

5.3.3 Double jump - dvojitý skok

Double jump, nebo-li dvojitý skok, je schopnost, která umožňuje herní postavě vyskočit dvakrát – tedy jednou z herní plochy a podruhé ve vzduchu. Hráč může tuto schopnost využít kdykoliv a nezávisle na tom, jak dlouho, či rychle, herní postava padá, její svislý pohyb bude pozastaven a bude proveden skok, jako by stála na herní ploše. Možnost využít tuto schopnost se obnoví při každém dopadu na herní plochu.

Implementace této schopnosti je provedena pomocí čísla vyjadřujícího počet skoků, které lze provést před opětovným kontaktem s herní plochou. Důvodem je možnost rozšíření této schopnosti, neboť při navýšení počtu skoků lze dosáhnout další schopnosti – Triple jump (trojitý skok), popřípadě Multiple jump, kdy se počet skoků může lišit dle potřeby, či návrhu herního mechanismu. Toto číslo je opět přístupné skrze inspektor vývojového prostředí, nicméně pro účely této práce má jeho úprava vliv pouze na schopnost Double jump – zda je tato schopnost uplatněna, nebo ne. V případě, že je zmíněný počet skoků nastaven na více než dva, je taková skutečnost ignorována a brána, jako by hodnota byla číslo dva.

5.3.4 Floating - plachtění

Schopnost Floating slouží herní postavě k plachtění – rychlost jejího pádu se drasticky zpomalí, zatímco rychlost horizontálního pohybu zůstane nepozměněna. Za pomoci této schopnosti je herní postava schopna pomocí skoku dostat se mnohem dál, obzvláště skáče-li z převýšení. Tato schopnost je v této práci řešena tak, že při jejím aplikování je zredukována gravitační síla – tím pádem, na rozdíl od jiných herních titulů, aktuální rychlost pádu není nijak snížena, či vynulována, pouze narůstá pomaleji. Hráč, nebude-li herní postava obsahovat schopnost Air strafe, nemá možnost v průběhu plachtění měnit směr pohybu a navíc bude doba, po kterou může herní postava plachtit, omezena. Dobu plachtění je možné nastavit v inspektoru vývojového prostředí a podobně jako u předchozích schopností, jsou-li její vlastnosti pozměněny, generátor mapy to při novém generování bere v potaz.

5.3.5 Wall catch - chytání stěn

Wall catch je schopnost, která umožňuje hráči chytat se stěn. Od stěny, které se hráč zrovna drží, je možné skočit, ale také po ní lézt vzhůru. Aby hráč mohl od stěny skákat, je potřeba vynulovat počet skoků, které již využil – tak, jak jsou nulovány při dopadu na herní plochu, aby mohl znovu skákat. V případě, že herní postava bude obsahovat schopnost Double jump, po přichycení ke zdi bude schopna schopnost znovu využít. Hráč by neměl mít možnost se držet stěny po libovolnou dobu bez žádné penalizace, neboť je potřeba jej udržet v nátlaku tvořícím potřebu dokončit herní mapu co nejdříve a nejefektivněji – proto, ačkoliv se bude hráč držet stěny, bude herní postava stále pomalu padat. Navíc, aby se zamezilo nekontrolovatelnému využívání schopnosti, je herní postava schopna se zachytávat pouze na specifických stěnách, které jsou znatelně vizuálně odlišené. Jak bylo zmíněno, herní postava je schopna po dané stěně lézt vzhůru – tato skutečnost je aplikována tak, že jak je hráči umožněno skočit od stěny, je mu umožněno poskočit i na místě, pouze vertikálně, a znovu se zachytit o stěnu, tentokrát o něco výše – a tímto způsobem může hráč lézt po stěnách, k tomu určených, vzhůru. Rychlost, kterou hráč konstantně klesá, ačkoliv se drží stěny, lze nastavit v inspektoru vývojového prostředí.

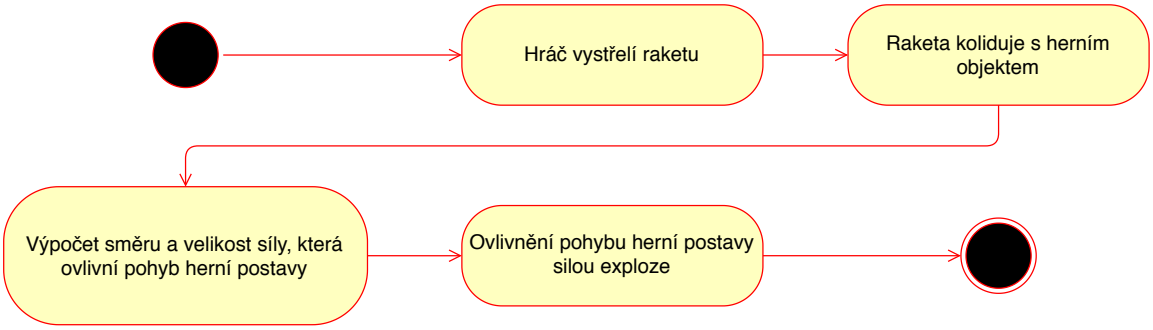
5.3.6 Teleport - teleportace

Schopnost Teleport umožňuje hráči se v mžiku přemístit z jednoho bodu do druhého. Pro tuto schopnost byla čerpána inspirace z herního titulu Dishonored. Schopnost v této práci je navrhnuta tak, že hráč si nejprve zobrazí místo, kam si přeje být teleportován, a pokud mu výsledná oblast vyhovuje, tak ji přijme a je na dané místo přemístěn. Místo, kam si hráč přeje být přemístěn, musí být viditelné z aktuální pozice herní postavy a musí být v patřičné maximální vzdálenosti, kterou lze nastavit přes inspektor ve vývojovém prostředí. Pokud hráč označí místo, které není v rámci maximální vzdálenosti, je výsledná pozice, na kterou bude hráč případně přemístěn, vyobrazena právě v oné maximální vzdálenosti, ve směru k výsledné pozici. V případě, že hráč určí jako výslednou pozici místo, kde není možné, aby se herní postava vešla, je hráči navrhnuta nejvzdálenější možná pozice – tedy nejvzdálenější od hráče, ale nejbližší k požadované pozici, v požadovaném směru bez žádného vychýlení.

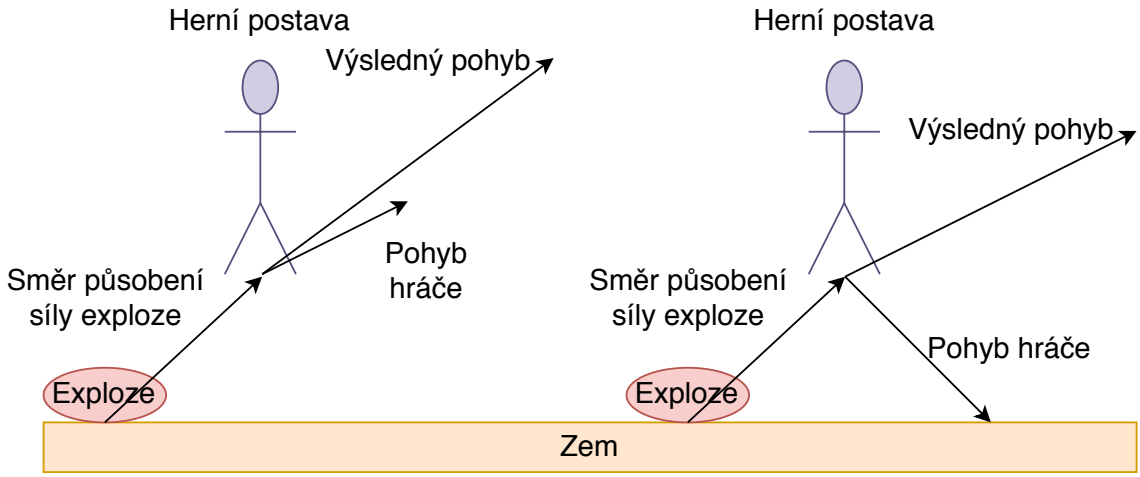
5.3.7 Rocket jump - skákání pomocí explozí raket

Rocket jump je schopnost, nebo spíše herní mechanismus, u které hráč využívá explozivní síly rakety k větším a delším skokům. Munice a kadence střelby je neomezená – hráč tedy

může střílet rakety neomezeně a tak často, jak rychle zvládne vydávat vstupní signál ke střelbě. Sekvence akcí při použití schopnosti Rocket jump znázorňuje obrázek 5.1 a velikost a směr síly působící na herní postavu od exploze znázorňuje obrázek 5.2.



Obrázek 5.1: Hráč využije raketomet, který herní postava vlastní, a vystřelí raketu zvoleným směrem. Raketa letí konstantní rychlostí, dokud nedojde ke kolizi s dalším herním objektem. Ve chvíli, kdy raketa koliduje s dalším objektem, odstraní se raketa z herního prostředí a vytvoří se exploze. Pokud je herní postava v určité maximální vzdálenosti od exploze, zjistí se směr a vzdálenost postavy od exploze. Na základě vzdálenosti a směru pozice herní postavy od exploze a aktuálního pohybu herní postavy je vypočítán vliv exploze na herní postavu a upraven její aktuální pohyb.



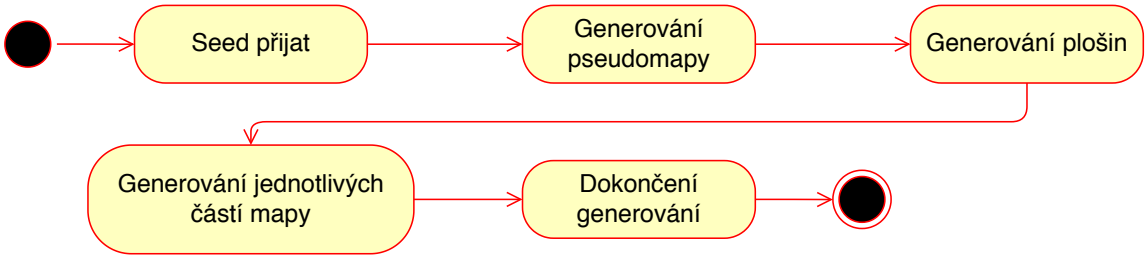
Obrázek 5.2: Tento obrázek znázorňuje výpočet výsledného pohybu herní postavy, jež byla ovlivněna explozí rakety. V případě, že byla herní postava ovlivněna explozí rakety, je vypočten směr a velikost síly působící na herní postavu, jež ovlivní pohyb postavy. Směr síly je počítán od bodu, kde exploze započala, směrem ke středu herní postavy, a velikost působící síly je nastavena na hodnotu síly exploze, redukované vzdáleností herní postavy od exploze. Síla exploze, působící na herní postavu, se poté sečte s jejím aktuálním pohybem (viz. levá část obrázku) s jednou výjimkou, a to pokud některá z hodnot vektorů pohybu herní postavy a síly exploze působící na herní postavu – jedná se o trojrozměrné vektory, neboť herní demo je implementováno v trojrozměrném prostoru – má odlišné znaménko, je daná hodnota vektoru pohybu herní postavy nastavena na 0 a dále se počítá pouze s hodnotou vektoru síly exploze působící na herní postavu (viz. pravá část obrázku).

Kapitola 6

Návrh procedurálního generátoru mapy

Procedurální generátor, jenž je výsledkem této práce, generuje herní mapu na základě herní postavy, jež má speciální schopnosti, které získá teprve v průběhu hraní – tzn. generátor vytvoří mapu, která na začátku nevyžaduje využití žádné schopnosti, neboť herní postava žádnou nemá, a poté, jak hráč postupně získává nové schopnosti, bude generátor postupně generovat části mapy, které vyžadují využití již získaných schopností.

Celý generátor bude rozdělen na páteřní generátor a dílčí generátory. Páteřní generátor vytváří pseudomapu, která určuje, jak budou jednotlivé části mapy propojeny, a poté využije dílčí generátory na generování jednotlivých částí mapy – viz. obrázek 6.1.



Obrázek 6.1: Generátor nejprve přijme seedy celého generování, a na jejich základě vygeneruje pseudomapu, která představuje, jak budou jednotlivé části mapy na sebe navazovat. Poté se vygenerují jednotlivé přechodné a rozvětvující se plošiny, jejichž účelem je propojit jednotlivé části mapy, nebo-li průchody. Následně, za využití dílčích generátorů, jsou generovány jednotlivé průchody mapy. Na závěr se mapa kompletně uzavře – tzn. díry, skrze které by mohl hráč spadnout z mapy, se zacelí – např. všechny plošiny jsou obezděny.

6.1 Páteřní generátor

Páteřní generátor je základní část celého procedurálního generátoru, která spravuje celé generování herní mapy. Páteřní generátor nejprve vytvoří pseudomapu, a posléze na jejím základě a za využití dílčích generátorů, jež jsou uložené v kontejneru dílčích generátorů, vygeneruje herní mapu.

6.1.1 Pseudomapa

Pseudomapa slouží jako předloha, na základě které se pak generuje celá herní mapa. Pseudomapa určuje patřičné závislosti mapy – např. které plošiny jednotlivé části herní mapy, nebo-li průchody, spojují, a na jakých schopnostech herní postavy jsou průchody závislé. Ve výsledku pseudomapa určuje, jak na sebe jednotlivé části herní mapy navazují.

Pro vytvoření pseudomapy si generátor nejprve zjistí, které schopnosti herní postavy mají být využity, a poté za využití generátoru náhodných hodnot vygeneruje pseudomapu s využitím zvolených schopností herní postavy. Podrobnější popis, jak je pseudomapa vytvořena, lze nalézt u obrázku 6.2.

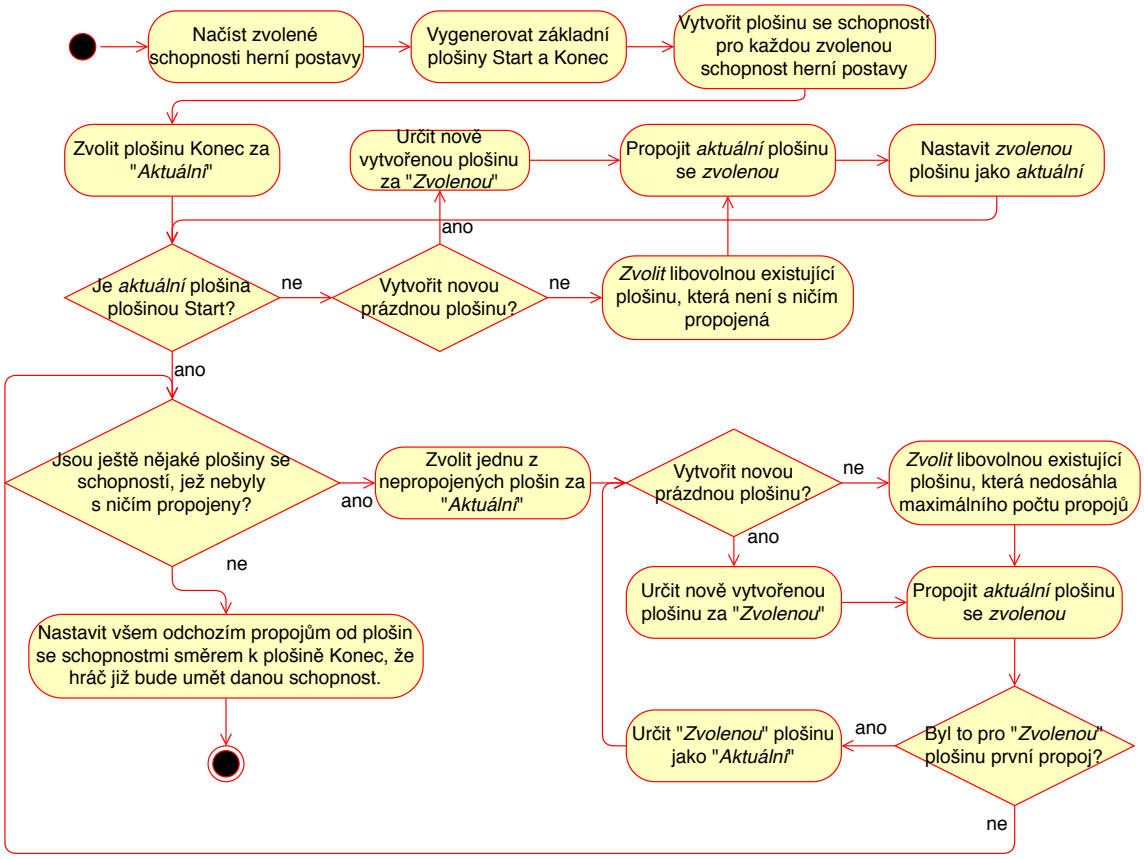
6.1.2 Kontejner dílčích generátorů

Úkolem kontejneru dílčích generátorů je sjednocení jednotlivých generátorů částí herní mapy na jedno místo. Generátory se dělí na základě schopnosti herní postavy, pro kterou jsou určeny. Pro jednu schopnost může být více generátorů, které generují odlišný typ mapy. Vzhledem k tomu, že je možné herní postavě přidávat další schopnosti a pro každou schopnost může být nespočet generátorů herní mapy, je výhodné mít dané generátory někde přehledně uspořádané a snadno přístupné.

6.1.3 Generování mapy

Jak již bylo zmíněno v kapitole 6.1, páteřní generátor nejprve vytvoří pseudomapu a na jejím základě, za využití dílčích generátorů, vygeneruje celou herní mapu. Tvorba pseudomapy je popsána v kapitole 6.1.1 a obrázku 6.2 a způsob využití dílčích generátorů je popsán v kapitolách 6.1.2 a 6.2.

Poté, co generátor připraví pseudomapu, prochází jednotlivé části herní mapy, které se v pseudomapě nachází, a jednu po druhé generuje. Nejprve je generována plošina – pokud již neexistuje – se kterou je začátek části mapy propojen. Poté je zjištěno, jaké schopnosti jsou v dané části známé. Následně je náhodně vybrána jedna ze známých schopností dané části mapy a z kontejneru dílčích generátorů je náhodně vybrán jeden z generátorů pro danou schopnost. Pak je dílčímu generátoru předána informace o pozici a směru začátku části herní mapy a generátor vygeneruje průchod, tedy část mapy obsahující překážky, pro jejichž zdolání je vyžadováno využití zvolené schopnosti. Na závěr, pokud není průchod připojen k již existující plošině, se vygeneruje nová plošina, na kterou se hráč, po projití průchodem, dostane.



Obrázek 6.2: Postup vytvoření pseudomapy. Generátor si nejprve zjistí schopnosti herní postavy, které byly zvoleny, a uloží si je do paměti. Následně generátor vygeneruje základní plošiny Start a Konec. Plošinou se rozumí část herní mapy, plocha, která spojuje jednotlivé části mapy, a v některých případech plní funkci křižovatky. Dále jsou vygenerovány plošiny pro všechny schopnosti herní postavy, které byly zvoleny. Následně je vygenerována hlavní linie herní mapy, tedy posloupnost průchodů, částí herní mapy, od plošiny Start k plošině Konec. Tato linie může být přímá, tvořící jeden průchod propojující plošinu Start přímo s plošinou Konec, nebo přes více plošin – o tom rozhodne vyhodnocení generátoru náhodných hodnot – stejně tak jako o tom, zda aktuálně tvořenou linii propojí s již existující plošinou obsahující schopnost herní postavy, nebo s prázdou plošinou, kterou čerstvě vygeneruje. Když je hotová hlavní linie, generátor se zaměří na to, aby každá plošina se schopností byla součástí mapy. Projde tedy všechny plošiny se schopnostmi, které ještě nebyly s ničím propojené, a propojí je s již existující mapou – opět buď přímo jediným průchodem, nebo sekvencí průchodů přes více plošin, které ještě nejsou s ničím propojené, nebo jsou nově vygenerované. Poté, co jsou všechny plošiny herní mapy propojené, generátor určí, v kterých částech herní mapy herní postava již zná/umí patřičné schopnosti a ve kterých ne.

6.2 Dílčí generátory mapy

Aby byl projekt rozšiřitelný, je generování mapy rozděleno na dílčí generátory. Každý dílčí generátor má určeno, pro kterou schopnost generuje mapu, tedy jeden generátor může generovat pouze pro jednu schopnost, ale pro jednu schopnost může generovat více generátorů.

Dílčí generátor generuje průchod mapou, nebo-li část herní mapy, po segmentech – počet segmentů a jejich délku je možné specifikovat skrze grafické rozhraní vývojového prostředí. Na začátku generovaného průchodu a na konci každého segmentu vygeneruje generátor checkpoint – viz. kapitola 6.2.2. Průchod je členěn na segmenty právě proto, aby do něj bylo možné umístit checkpointy, které usnadňují zdolávání herní mapy a zpříjemňují hráči požitek ze hry.

6.2.1 Randomizace jednotlivých kroků průchodu mapy

Po patřičném zkoumání možností Perlinova šumu a jeho využití bylo zjištěno, že funkcionality Perlinova šumu jsou velice užitečné při souvislém generování – např. při generování terénů herní mapy, dvourozměrných textur, trojrozměrných textur atd. Vzhledem k tomu, že při překonávání překážek herní mapy není potřeba držet se jakéhokoliv souvislého přechodu – např. postupný přechod ze skákání vlevo na skákání vpravo – ale naopak je vhodné v jeden moment s herní postavou skákat prudce vlevo a v druhý moment skákat prudce vpravo, tak není potřeba využívat tak sofistikované funkce, jako je Perlinův šum. Naopak bude vhodnější využít jednodušších funkcionalit, kterých lze dosáhnout za využití obyčejného generátoru pseudonáhodných čísel – viz. kapitola 4.3 – díky čemuž lze dosáhnout zlomového přechodu mezi opačnými hodnotami (jako např. přechod ze skoku prudce vlevo na skok prudce vpravo).

6.2.2 Checkpointy

Checkpoint je bod návratu pro případ, že hráč neuspěje při zdolávání překážky. Ve vygenerované herní mapě se nachází rovnou několik checkpointů. Checkpointy se vždy vyskytují na začátku části herní mapy, nebo-li průchodu, a pak periodicky po několika překážkách daného průchodu. Pod celým průchodem je neviditelná plošina, která při kontaktu s herní postavou přemístí hráče na poslední checkpoint, kterým hráč prošel. Checkpoint vždy pokrývá celou plochu plošiny, na které se nachází, aby nehrozilo, že by se hráč vyhnul kontaktu s checkpointem.

Kapitola 7

Návrh rozhraní pro testování a prezentaci

Vzhledem k tomu, že cílem této práce je vytvořit procedurální generátor herní mapy, nikoliv herní mapu samotnou, či ucelenou hru, je demonstrace výsledků práce náročná, neboť generátor lze ovládat pouze skrze vývojové prostředí Unity3D. Proto je potřeba vyvinout i uživatelské rozhraní, za jehož pomoci by bylo možné testovat možnosti procedurálního generátoru a prezentovat výsledky.

7.1 UI pro testovací a prezentační účely

Generování herní mapy pomocí procedurálního generátoru, jenž má být výsledkem této bakalářské práce, je závislé na mnoha aspektech. Jak je již zmíněno v zadání práce, generování je závislé na schopnostech herní postavy – proto je možné pomocí uživatelského rozhraní nastavovat veškeré vlastnosti herní postavy – od základních vlastností jako je rychlost pohybu, výška skoku apod., až po vlastnosti schopností herní postavy jako např. síla Air strafe schopnosti, síla exploze rakety při Rocket jumpu apod.

Teprve poté, co budou nastaveny veškeré vlastnosti – základní vlastnosti herní postavy, vlastnosti schopností herní postavy, dokonce i vlastnosti jednotlivých generátorů (např. délka generovaného průchodu mapy), a hlavně kořen a inkrement hlavního generátoru pseudonáhodných hodnot – teprve tehdy se bude generovat herní mapa, kterou si lze posléze prohlédnout a projít.

7.1.1 Virtuální prohlídka mapy

Poté, co je herní mapa vygenerována, má uživatel testovacího prostředí možnost si mapu prohlédnout. Uživatel má možnost se v prostředí herní mapy svobodně pohybovat a danou herní mapu si dle libosti prohlédnout. Způsob, jakým se uživatel může v testovacím prostředí pohybovat, je napodobením ovládacích možností vývojového prostředí Unity3D – viz. kapitola 10.2.

Uživatel si může prohlédnout libovolnou část herní mapy, a v případě, že mu výsledek generování nevyhovuje, může se vrátit k nastavování vlastností generování a pozměnit je dle libosti, dokud výsledek generování není vyhovující.

7.1.2 Možnost odehrání mapy

V případě, že uživateli vyhovuje výsledná procedurálně vygenerovaná mapa, může ještě přejít do herního módu, kdy má možnost herní mapu projít jakožto hráč – tzn. na začátku mapy se objeví herní postava, kterou uživatel může ovládat a projít s ní mapu až do konce. V případě, že uživatel v průběhu hraní narazí na nějaký problém či mu vygenerovaná mapa nevyhovuje, může hraní za herní postavu přerušit, upravit vlastnosti generování mapy a vygenerovat novou mapu.

7.2 Ukládání nastavení pro testovací a prezentační účely

Vzhledem k tomu, že dílčích generátorů, jejichž vlastnosti se před generováním mapy nastavují, může být nekonečné množství, a samotné nastavování vlastností herní postavy a jejich schopností je komplexní, obnáší výsledek práce i automatické ukládání nastavených hodnot. Tzn. kdykoliv uživatel pozmění jakoukoliv vlastnost kteréhokoliv generátoru (i toho páteřního), nebo vlastnost herní postavy, či některé z jejich schopností, je tato změna automaticky uložena a zapamatována, dokud nebude znovu pozměněna. Tyto uložené hodnoty jsou automaticky načteny při příštím spuštění aplikace – uživatel si nemusí pamatovat, jaké hodnoty využil posledně.

Kapitola 8

Implementace herní postavy

Tato kapitola se zaměřuje na vývoj herní postavy a její schopnosti. Nejprve probírá její základní vlastnosti, tedy pohyb, základní skok, kolize, gravitaci, ale také její ovládání, tedy reagování na patřičné vstupy apod. Poté, co je popsána tvorba základních vlastností herní postavy, se kapitola zaměří na jednotlivé schopnosti herní postavy, jejich vlastnosti a implementaci.

8.1 Základní vlastnosti herní postavy

Jako herní objekt prezentující herní postavu byl zvolen jeden ze základních geometrických útvarů dodaných vývojovým prostředím Unity3D, a to kapsule. Tento herní objekt, dodaný vývojovým prostředím Unity3D, již implicitně obsahuje kolizní kapsuli o stejném tvaru – není tedy třeba řešit její tvorbu. Co je ale potřeba dodat jako základní výbavu je komponenta Rigidbody – popis co je zač komponenta Rigidbody, jaké jsou další možnosti a proč byla zvolena zrovna komponenta Rigidbody je v kapitole 5.1.1. V tento moment, právě díky komponentě Rigidbody, je již možné, aby se herní postava hýbala – působí na ni totiž gravitační síla a po spuštění hry začne herní postava padat.

8.1.1 Základní ovládání herní postavy

Teď když má herní postava základní fyzikální vlastnosti – díky kolizní kapsuli může kolidovat a díky komponentě Rigidbody na ní působí síly jako např. gravitace – je na čase naimplementovat základní ovládání herní postavy. Mezi toto základní ovládání spadá možnost rozhlížet se po okolí, možnost pohybu a nakonec i možnost skoku.

Implementace rozhledu herní postavy

Aby se mohl hráč rozhlížet v herním prostředí, je třeba, aby se v tomto prostředí nacházela kamera. Při vytvoření nového herního prostředí Unity3D implicitně vytvoří prostředí, jež obsahuje statickou kameru – pokud si vývojář tuto kameru smazal, nebo ji využívá k jiným účelům, musí si do herního prostředí přidat novou kameru. Aby nebyla kamera statická, ale dynamicky měnila pozici na základě pozice herní postavy, je potřeba tuto aktualizaci zařídit – toho lze dosáhnout buď vlastním skriptem, nebo tím, že se kamera nastaví jako potomek patřičného herního objektu. Vzhledem k tomu, že herní demo, jež má být výsledkem této práce, bude formou pohledu z první osoby, lze využít možnosti, kdy se herní kamera nastaví jako potomek herní postavy, a její výchozí pozice se nastaví na vrchní část

kapsule, jež prezentuje herní postavu – zhruba v oblasti, kde by se nacházela hlava herní postavy. Je-li kamera nastavena jako potomek herní postavy, je pozice a rotace kamery aktualizovaná též, kdykoliv se pozice či rotace herní postavy pozmění. Vzhledem k tomu, že kamera reaguje na rotaci postavy, je rozhlížení pomocí rotování kolem vertikální osy již řešeno implementací pohybu herní postavy. Nicméně ještě je potřeba implementovat možnost rotace pohledu kolem horizontální osy – pro zaznamenání, jestli a jak moc má být pohled rotován, jsou snímány vstupní hodnoty pohybu uživatelovy myši – tento pohyb je násoben zadanou citlivostí myši, a výsledek se vyhodnotí jako počet stupňů, o kolik má být kamera rotována.

Pro dosažení maximální intuitivnosti ovládání kamery herní postavy je potřeba úhel, o kolik lze pohled kamery rotovat kolem horizontální osy, omezit. Jak pro vstupní hodnoty znázorňující záporné hodnoty, tedy rotování kamery kolem horizontální osy směrem dolů, tak i pro kladné vstupní hodnoty, tedy hodnoty znázorňující rotování kolem horizontální osy vzhůru, je potřeba určit maximum – v obou případech se jedná o ohraničení 90 stupňů, celkově tedy 180 stupňů. Herní postava se tak bude moci podívat pod sebe a nad sebe, ale nebude hrozit, že by se pohled kamery přetočil.

Implementace pohybu herní postavy

Ačkoliv má herní postava komponentu Rigidbody, která zařizuje spolehlivou simulaci na základě působení sil, je vzhledem k typu hry ovládání herní postavy skrze působení sil nevhodné – vyjma gravitační síly. Ovládání herní postavy skrze působení sil by působilo prodlevu při změně směru pohybu, a to je pro zdolávání překážkové mapy nevhodné. Vzhledem k daným okolnostem je pohyb herní postavy řešen tak, že jsou komponentě Rigidbody přímo modifikovány směr a rychlost pohybu herní postavy. Kdyby byl pohyb herní postavy řešen přes působení sil na komponentu Rigidbody, komponenta by si výsledný směr a rychlost pohybu vypočítala sama. Když vývojář pracuje přímo se samotným směrem a rychlostí pohybu, bez působení sil, má správu pohybu herní postavy plně ve svých rukou – což je v žánru hry tohoto dema žádané.

Co se týče rotace herní postavy kolem vertikální osy, implementovaný kód zjišťuje vstupní hodnoty pohybu uživatelovy myši, tyto hodnoty násobí nastavenou sensitivitou a výslednou hodnotou vyjadřuje počet stupňů, o kolik má být herní postava rotována kolem vertikální osy. Herní postava je rotována pomocí komponenty Transform, již vývojové prostředí Unity3D implicitně dodává každému hernímu objektu. Implementace zjistí aktuální rotaci herní postavy a inkrementuje ji o vypočtenou hodnotu. Tímto způsobem je hráči umožněno se rozhlížet kolem vertikální osy – neboť je kamera nastavena jako potomek herní postavy, tzn. když je rotováno herní postavou, je rotováno i všemi jejími potomky.

Implementace skoku herní postavy

V případě skoku herní postavy je tomu podobně jako při klasickém pohybu – pouze se nastaví vertikální směr a rychlost pohybu herní postavy podle nastavené síly skoku a o zbytek se postarají výpočty komponenty Rigidbody na základě působení gravitační síly.

Při testování chování herní postavy na základě zmíněné implementace se ukázalo, že výchozí působení gravitační síly může působit nepřirozeně. Proto byla vypnuta vlastnost Use Gravity komponenty Rigidbody, neboť gravitační síla, jež působila na herní postavu, budí dojem, že není dostatečně silná. Namísto toho byla herní postavě přidána komponenta Constant Force, která konstantně na komponentu Rigidbody působí silou, jež byla nastavena skrze inspektor vývojového prostředí Unity3D. Zvolená hodnota, jež byla komponentě

Constant Force nastavena, je -30 (tedy vektor, kde jediná nenulová osa je osa Y s hodnotou -30), tedy zhruba trojnásobek síly, jež je aplikována klasickou gravitací komponenty Rigidbody.

8.2 Schopnosti herní postavy

Poté, co byla dokončena implementace základních vlastností herní postavy, tedy její základní pohyb a skok, je načase začít s implementací schopností herní postavy – tedy speciálních vlastností, které umožňují překonávat speciální překážky, které se základními pohybovými vlastnostmi herní postavy, tedy bez speciálních schopností, nelze překonat.

8.2.1 Air strafe

Air strafe je schopnost, která pomocí působení sil na herní postavu umožňuje ovlivňování pohybu herní postavy ve vzduchu. Hráč tak může skákat za roh, nebo jakkoliv jinak měnit směr letu vzduchem.

Při implementaci této schopnosti je potřeba brát v potaz, že pohyb herní postavy nesmí přesáhnout patřičnou maximální rychlost, a zároveň, pokud se herní postava již pohybuje rychlostí nad povolené maximum, nesmí schopnost Air strafe při působení silou stejným směrem, jako je aktuální pohyb herní postavy, pohyb herní postavy zpomalovat – viz. algoritmus 1.

Ačkoliv se jedná o působení sil na herní postavu, stále nebude využito metody AddForce komponenty Rigidbody. Při využití metody AddForce je náročnější udržovat pohyb herní postavy v mezích určené maximální rychlosti. Pro snazší práci je síla schopnosti Air strafe aplikována přímo na rychlost pohybu postavy, již též spravuje komponenta Rigidbody – té nebude předávána síla, čili se nebude volat metoda AddForce apod., ale pracuje se přímo s výslednou rychlostí pohybu herní postavy.

8.2.2 Crouch

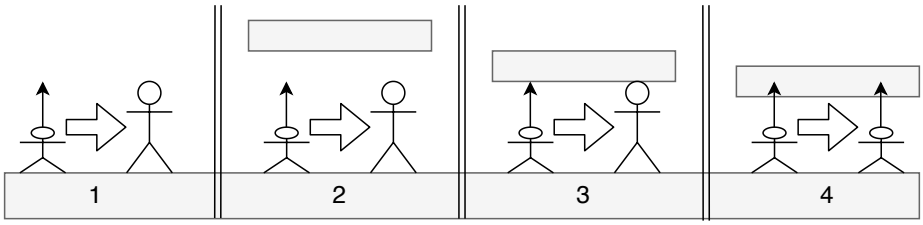
Při implementaci schopnosti Crouch (přikrčení se), tedy snížení výšky herní postavy, je potřeba brát v potaz, že rozměry herní postavy se mění souměrně od středu herní postavy. V tom případě lze očekávat, že při zmenšení herní postavy se bude herní postava nacházet ve vzduchu a při zvětšení herní postavy se bude herní postava nacházet částečně v herní plošině, na které původně stála. Stejně tak je potřeba brát v potaz, že účel schopnosti je možnost průchodu nižšími prostory, kterými by za normálních okolností nebylo možné projít – v případě, že hráč chce, aby herní postava nabyla své původní výšky, musí být nejprve zkontrolováno, zda se herní postava do aktuálního herního prostředí ve své původní výšce vejde. Všechny tyto případy jsou blíže popsány v obrázku 8.1 a algoritmu 2.

Algorithm 1: Metoda pro výpočet působení síly schopnosti Air strafe na aktuální pohyb herní postavy – GetAirStrafeForce. Vstupními parametry jsou trojrozměrné vektory aktualniPohyb (reprezentující aktuálního pohybu herní postavy) a vstupniPohyb (reprezentující sílu, jež by měla ovlivnit pohyb herní postavy). V případě, že některá z os vektoru vstupniPohyb má shodné znaménko, ale zároveň menší absolutní hodnotu, jako ta samá osa vektoru aktualniPohyb, je hodnota dané osy vektoru vstupniPohyb nastavena na 0. Poté je vektor vstupniPohyb vynásoben sílou schopnosti Air strafe – AirStrafePower. Hodnota proměnné AirStrafePower může být mnohonásobně větší než maximální rychlost pohybu herní postavy – to proto, že jiné schopnosti herní postavy umožňují přesáhnout maximální rychlost pohybu herní postavy, a síla AirStrafePower naopak umožňuje dané přesáhnutí maxima rychle snížit. Poté je aktualniPohyb modifikován silou vstupniPohyb a zkontrolován, zda nepřesáhl maximální rychlost pohybu herní postavy – popřípadě původní rychlost herní postavy na začátku metody, pokud rychlost již přesáhla maximum.

```
1  function GetAirStrafeForce(aktualniPohyb, vstupniPohyb)
2      if směr vstupniPohyb == směr aktualniPohyb
3      and rychlost vstupniPohyb < rychlost aktualniPohyb then
4          vstupniPohyb = 0
5      vstupniPohyb *= AirStrafePower
6      aktualniPohyb += vstupniPohyb
7      if původní rychlost aktualniPohyb > maximální rychlost pohybu then
8          velikost vektoru aktualniPohyb ohraničena původní rychlostí aktualniPohyb
9      else
10         velikost vektoru aktualniPohyb ohraničena maximální ryhlostí pohybu
```

Algorithm 2: Metoda DoCrouch řeší působení schopnosti Crouch. Jediným parametrem této metody je příznak aplikovatSchopnost, který určuje, zda má být funkcionalita schopnosti aplikována, nebo naopak negována. V případě, že má být schopnost aplikována, nejprve se zkontroluje, zda již nebyla aplikována. Pokud schopnost ještě aplikována nebyla, tak se aplikuje – výška postavy se zredukuje na předurčenou hodnotu CrouchHeightReduction schopnosti Crouch a vertikální pozice herní postavy se sníží tak, aby herní postava stále přiléhala k povrchu herní plochy. V případě, že má být schopnost negována, nejprve se zkontroluje, zda má herní postava dostatek prostoru pro nabytí své původní výšky – pokud ne, tak je tato funkce vyhodnocována od znovu při vykreslování následujícího snímku obrazovky. Pokud má herní postava dostatek prostoru pro nabytí své původní výšky, zkontroluje se, zda schopnost byla aplikována, a pokud ano, herní postava nabyde své původní výšky a vertikální pozice je navýšena tak, aby herní postava přilnula k hernímu povrchu.

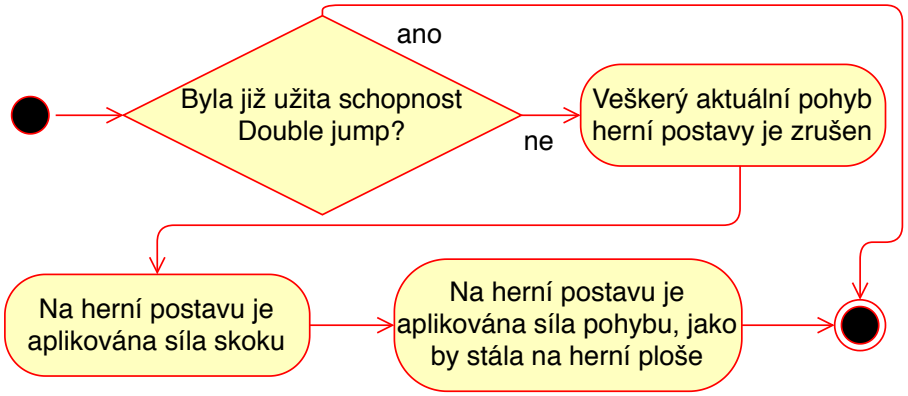
```
1  function DoCrouch(aplikovatSchopnost)
2      if aplikovatSchopnost then
3          if schopnost nebyla již aplikována then
4              výška postavy *= CrouchHeightReduction
5              vertikální pozice herní postavy -=
6                  (původní výška postavy / 2) - (aktuální výška postavy / 2)
7      else
8          if nad herní postavou není dostatek místa then
9              při vykreslování dalšího snímku znovu zavolat funkci DoCrouch
10         else
11             if schopnost byla aplikována then
12                 vertikální pozice herní postavy +=
13                     (původní výška postavy / 2) - (aktuální výška postavy / 2)
14                 výška postavy /= CrouchHeightReduction
```



Obrázek 8.1: Tento obrázek znázorňuje postup vyhodnocování, zda herní postava může nabýt zpět své původní výšky po využití schopnosti Crouch. V případě, že má herní postava po využití schopnosti Crouch nabýt zpět svou původní výšku, nejprve vyšle přímo nad sebe paprsek, jehož délka je rovna výšce, o kterou byla herní postava snížena. Tento paprsek je znázorněn šipkou směřující od levé menší postavičky ve všech čtyřech případech znázorněných na obrázku. Pokud tento paprsek neprotne žádný herní objekt, nedetekuje tedy žádný herní objekt v dané vzdálenosti nad herní postavou, herní postava nabyde svou původní výšku – jak znázorňují první tři případy vykreslené na obrázku, kdy v případě 1 se nad herní postavou nenachází žádný objekt, v případě 2 lze již objekt pozorovat, ale je příliš vzdálený a v případě 3 se objekt nachází právě ve výšce herní postavy, ale herní postava se zde stále vleze a kontrolní paprsek tedy daný objekt nedetekoval. V případě 4 lze vidět, že herní objekt nad herní postavou je již příliš nízko a kontrolní paprsek, jenž znázorňuje výšku, kterou by herní postava nabyla, daný herní objekt již protíná, tedy detekuje – proto herní postava nenabyla své původní výšky a pokračuje v kontrolní detekci, dokud nenajde vhodné místo, kde by mohla nabýt své původní výšky.

8.2.3 Double jump

Schopnost Double jump, nebo-li dvojitý skok, umožňuje hráči vyskočit během pohybu vzduchem. Ve chvíli kdy hráč tuto schopnost uplatní, je směr jeho pohybu okamžitě změněn – nezávisle na předchozích okolnostech. Postup vyhodnocení schopnosti lze pozorovat na obrázku 8.2.



Obrázek 8.2: V případě, že se herní postava nachází ve vzduchu a hráč vydá signál k výskoku, nejprve se zkontroluje, zda v průběhu letu vzduchem schopnost Double jump nebyla již použita – jinak by se nejednalo o Dvojitý skok, ale o trojitý atd. Pokud schopnost byla již použita, herní postava na signál skoku nijak nereaguje, v opačném případě je nejprve zrušen veškerý aktuální pohyb herní postavy (jako by stála na herní ploše), poté je na herní postavu aplikována síla klasického skoku a nakonec je ještě aplikována na postavu síla klasického pohybu – díky čemuž může hráč okamžitě změnit směr pohybu herní postavy uprostřed vzduchu, neboť v daný moment se postava zachová, jako by stála na herní ploše.

8.2.4 Floating

Schopnost Floating, nebo-li plachtění, slouží k překonání propastí a jiných delších překážek, které je potřeba překonat bez kontaktu s herním povrchem. Schopnost je řešena redukcí gravitační síly, tedy redukcí síly osy Y komponenty Constant Force, a tím pádem zpomalením růstu rychlosti pádu. Tato redukce je aplikována teprve tehdy, kdy postava padá, neboť kdyby byla aplikována dříve, tak by bylo možné s herní postavou dosáhnout vyšších pozic – a to není chtěné (viz. algoritmus 3).

8.2.5 Wall catch

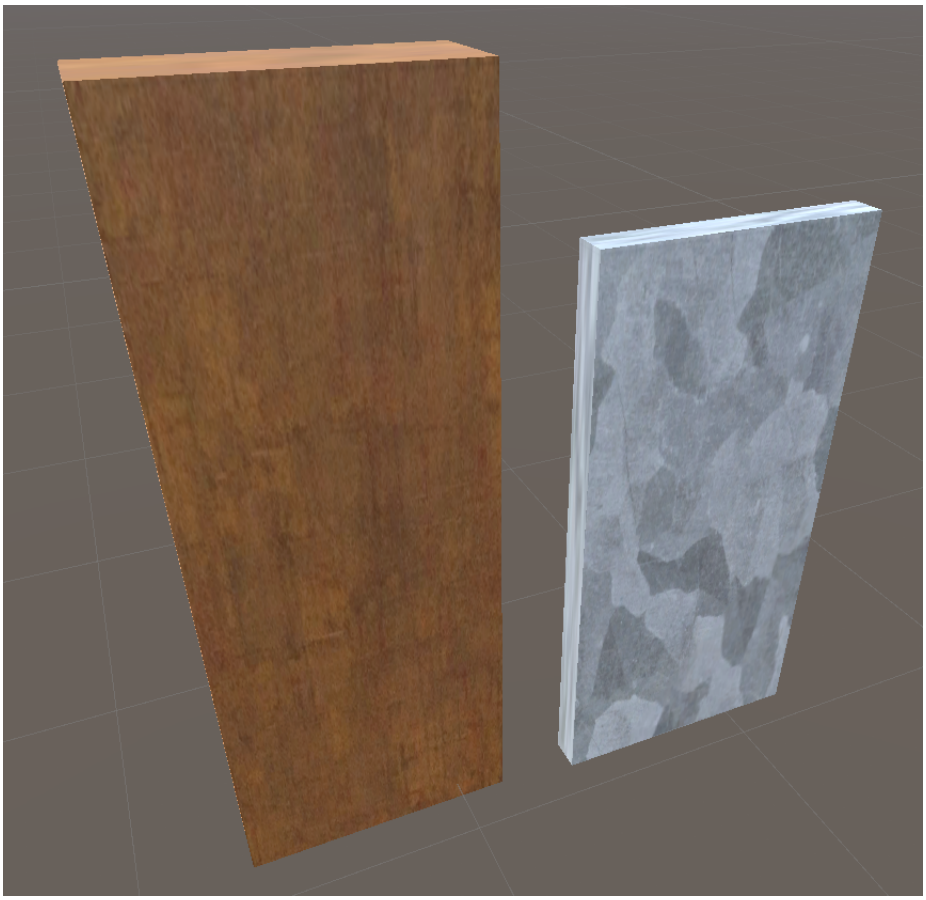
Wall catch je schopnost, která umožňuje hráči přichytávat se na zdi. Tato schopnost není omezena početně, ani časově, herní postava při držení se stěny pouze postupně klesá – tím by mohla být schopnost Wall catch zneužívána. Aby se zabránilo zneužívání schopnosti, je možné schopnost využít pouze u specifických zdí, které jsou vizuálně odlišné od jiných zdí viz. obrázek 8.3. Algoritmus 4 popisuje postup vyhodnocení schopnosti.

Algorithm 3: Má-li být aplikována schopnost DoFloating, nejprve se zkontroluje, zda za daný pád/skok nebyla schopnost již aplikována a zda herní postava právě padá/klesá. V případě, že jsou všechny podmínky splněny, je síla komponenty Constant Force herní postavy, na kterou komponenta působí pouze v ose Y. Poté se vyhodnocování metody na dobu, jež je uložena v proměnné FloatingTime, odmlčí a po odmlce redukci pádu rekurzivním voláním metody zruší. V případě, že aplikování schopnosti Float má být zrušeno, je komponentě Constant Force navrácena její původní síla – kdy její původní síla je uložena v proměnné OriginalFallingSpeed.

```
1 function DoFloating(aplikovatSchopnost)
2     if aplikovatSchopnost then
3         if schopnost nebyla již aplikována and herní postava klesá (padá) then
4             ConstanteForce.y *= FloatingFallReduction
5             Wait(FloatingTime)
6             DoFloating(false)
7     else
8         ConstantForce.y = OriginalFallingSpeed
```

Algorithm 4: V průběhu hraní je průběžně detekováno, zda se v bezprostřední blízkosti herní postavy nachází speciální zeď pro schopnost Wall catch. Pokud se v blízkosti herní postavy taková speciální zeď nachází a uživatel vydá signál pro využití schopnosti Wall catch a herní postava již padá (nestoupá), je rychlost pádu, nebo-li vertikální pohyb herní postavy, nastavena na konstantní hodnotu WallCatchFallSpeed. V případě, že některá z podmínek není splněna, např. se v blízkosti herní postavy nenachází požadovaná speciální zeď, schopnost se neaplikuje, tím pádem se neaktualizuje konstantní rychlost pádu a herní postava začne klasicky nabírat rychlost pádu díky působení gravitační síly.

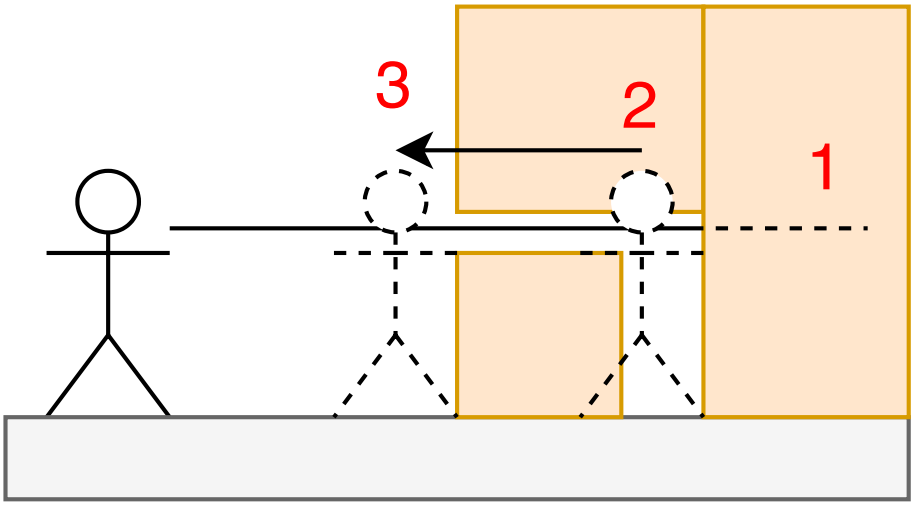
```
1 function CheckWallCatch()
2     if aktivován(vstupní signál schopnosti)
3         and poblíž(WallCatch zeď) and herní postava padá then
4         VertikalniPohybPostavy = WallCatchFallSpeed
```



Obrázek 8.3: Vizuální odlišení zdí. Hnědá zeď vlevo prezentuje klasickou zeď. Šedá zeď vpravo prezentuje speciální zeď, za kterou se herní postava za pomoci schopnosti Wall catch může zachytávat.

8.2.6 Teleportace

Schopnost Teleportace je implementována tak, aby si hráč nejprve mohl zkontrolovat výslednou pozici herní postavy po využití schopnosti Teleportace. Tuto pozici pak může buď přijmout, nebo odmítnout. Postup vyhodnocení výsledné pozice je detailněji popsán na obrázku 8.4. Dále, aby nebylo možné schopnost libovolně zneužívat, má schopnost omezený počet využití. Bylo zamýšleno, aby tento počet využití schopnosti byl při vstupu do nové sekce mapy obnoven, nebo aby bylo možné jej navyšovat – např. sbíráním speciálních herních objektů.



Obrázek 8.4: obrázek popisuje průběh využití schopnosti Teleport. Krok č. 1 je vyslat paprsek o určené maximální délce ve směru, ve kterém se herní postava právě dívá. Tímto paprskem je hledán výsledný bod teleportace. V případě, že paprsek protnul jiný herní objekt, bod, kde paprsek poprvé protnul jiný herní objekt, se stává výsledným bodem. V případě, že paprsek neprotnul žádný jiný herní objekt, je výsledným bodem koncový bod paprsku. Krok č. 2 znázorňuje projekci herní postavy ve výsledném bodě a kontrolu, zda je možné zde herní postavu umístit. V případě, že se v dané lokalitě nenachází dostatek místa pro herní postavu, je její projekce postupně přibližována ke skutečné herní postavě, dokud není nalezeno vhodné místo, kde by se herní postava mohla vlézt – znázorněno krokem č. 3. Poté, co je nalezeno vhodné místo pro herní postavu, je tato projekce hráči vyobrazena a v případě, že hráč tuto lokalitu přijme, je herní postava na tuto lokalitu přemístěna.

8.2.7 Rocket jump

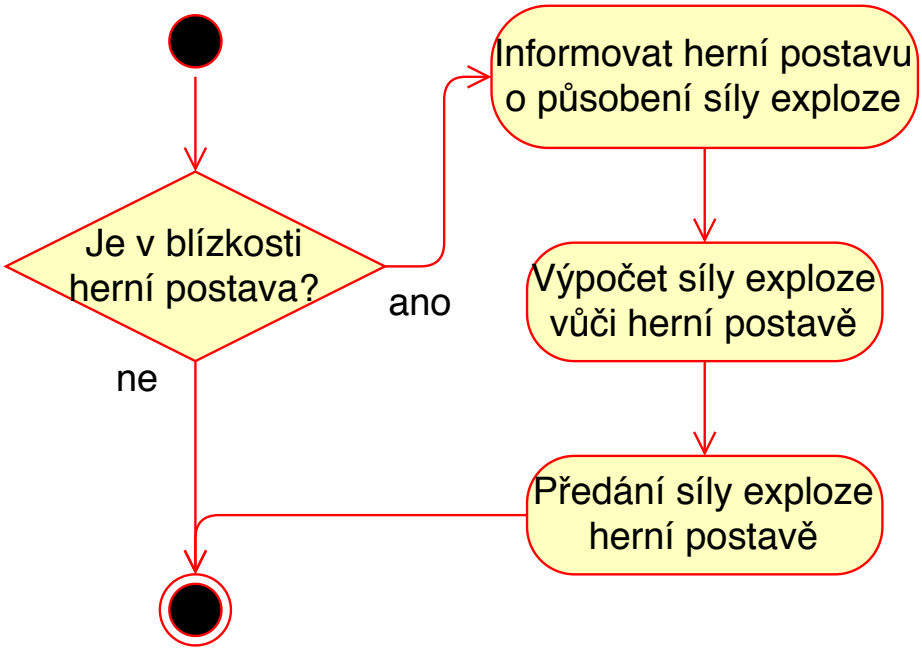
Implementace schopnosti Rocket jump, nebo-li skákání za využití explozivní síly raket, je rozdělena na více částí – těmi jsou střelba raket, let a exploze raket a simulace vlivu exploze rakety na herní postavu.

Pro lepší přehled hráče byl herní postavě přidán kvádr prezentující raketomet – za jeho pomoci by se měl hráč při míření lépe orientovat. Poté co hráč vyvolá signál střelby, vyletí z daného kvádru ve směru, ve kterém právě kvádr směřuje, herní objekt reprezentující raketu. Aby hráč nemohl schopnost Rocket jump neomezeně zneužívat, je počet raket, které hráč může vystřelit, omezený. Bylo zamýšleno, aby tento počet využití schopnosti byl při vstupu do nové sekce mapy obnoven, nebo aby bylo možné jej navyšovat – např. sbíráním speciálních herních objektů.

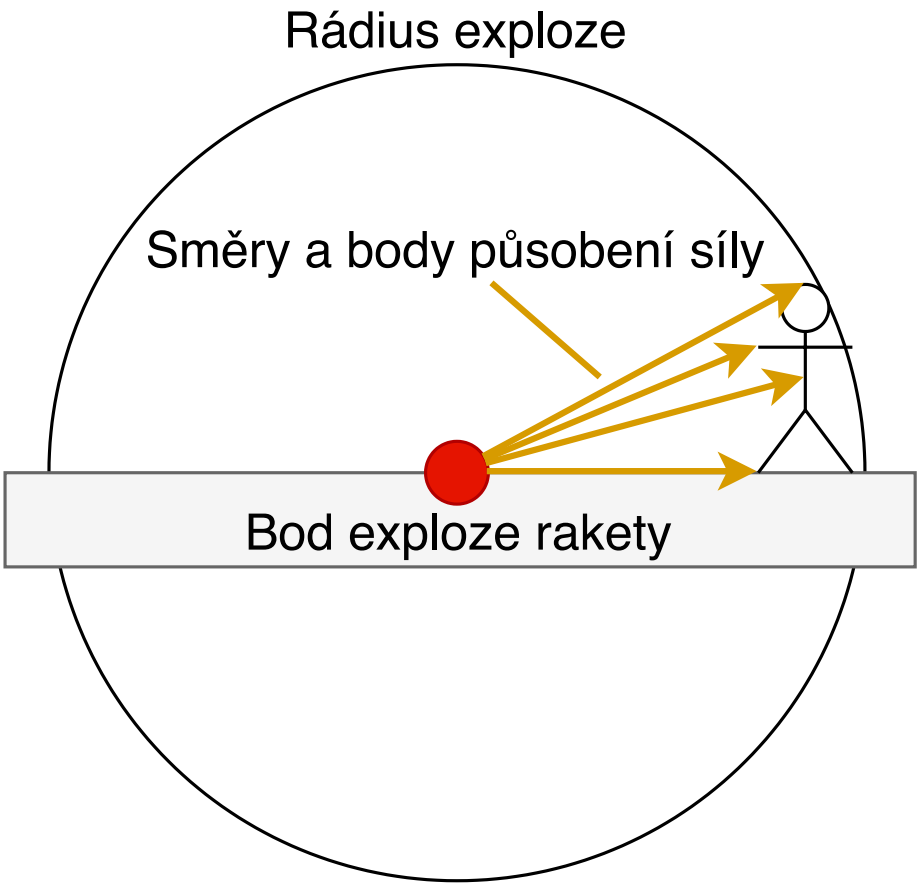
Rychlost letu rakety je konstantní a směr přímočarý – to aby mohl hráč snáze odhadovat čas a místo dopadu rakety na herní plochu, a tím pádem lépe plánovat využití schopnosti. Poté, co nastane kolize rakety s jiným herním objektem, raketa okamžitě exploduje, zjistí všechny herní objekty v nastaveném rádiu, a pokud mezi nimi najde herní postavu, tak jí předá patřičnou explozivní sílu.

Při předávání explozivní síly herní postavě a simulování jejího vlivu je potřeba brát v potaz skutečnost, že pořadí vyhodnocování jednotlivých skriptů je náhodné – tzn. jednou může být vyhodnocen nejprve skript rakety a poté skript herní postavy, a podruhé naopak. V případě, že by exploze rakety účinkovala na herní postavu, jež stojí na herní ploše, a pořadí vyhodnocování skriptů by bylo zrovna takové, že se nejprve vyhodnotí exploze rakety a poté pohyb herní postavy, exploze rakety sice předá herní postavě explozivní sílu, jež má ovlivnit její pohyb, ale následné vyhodnocení pohybu herní postavy vyhodnotí, že postava stojí na herní ploše, a tím pádem její pohyb bude záviset pouze na vstupních signálech pohybu, které zadá hráč – tzn. pokud hráč nevyvolá žádný signál pohybu, tak ačkoliv měla být na herní postavu aplikována síla exploze rakety, herní postava se nikterak nepohne. Z toho důvodu je při explozi rakety nejprve herní postavě předána informace, že na ni účinkuje síla exploze, což zapříčiní, že herní postava na krátký moment bude ignorovat herní plochu pod sebou a bude tedy simulovat, že se právě nachází ve vzduchu. Poté, co je herní postavě předána informace o působení explozivní síly, jí je předána i síla exploze – viz. obrázek 8.5.

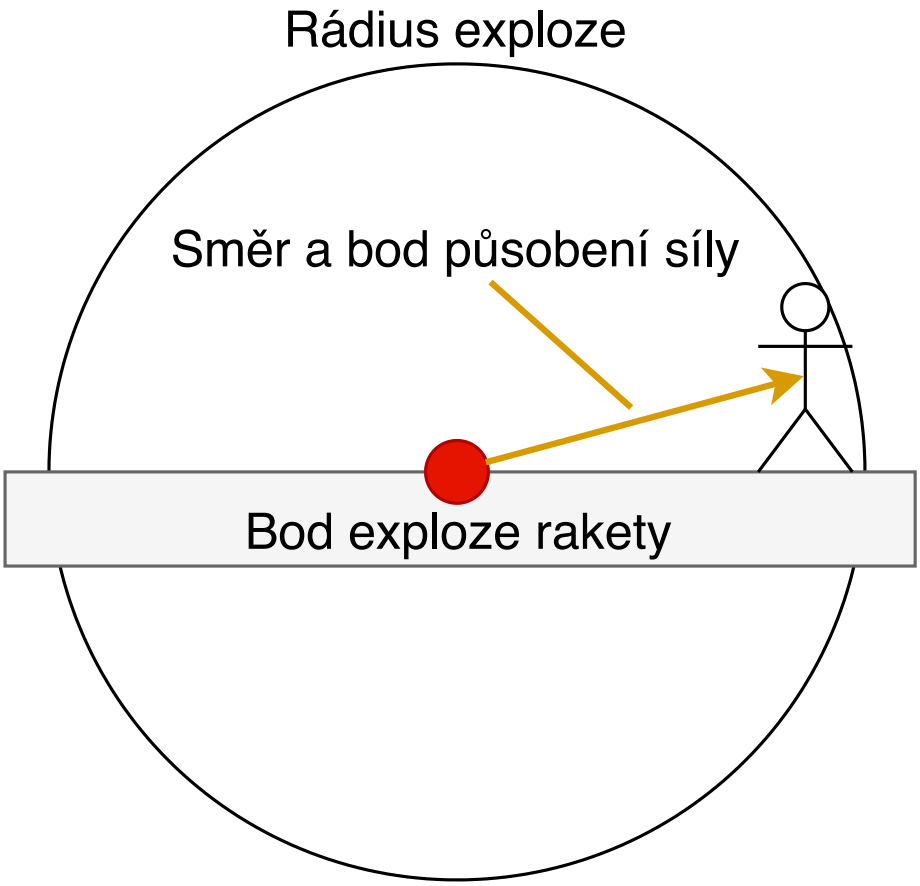
Při vývoji schopnosti se naskytlo několik možností, jak sílu exploze aplikovat na herní postavu – nejlepšími možnostmi byla metoda AddExplosionForce komponenty Rigidbody (viz. Unity manuál [6]) a vlastní výpočet vlivu exploze na herní postavu. Nejprve bylo vyzkoušeno využití metody AddExplosionForce, neboť tato metoda vypočítá sílu, směr a bod působení exploze na herní postavu podle fyzikálních zákonů (viz. obrázek 8.6) – nicméně výsledky této metody by se mohly hráči jevit nepřirozené, ačkoliv by se řídily fyzikálními zákony. Využití metody AddExplosionForce by bylo vhodnější v případě, že by herní postava měla více fyzikálních vlastnosti. Nicméně v průběhu implementování byly různé vlastnosti herní postavě odebrány – např. jí byly zakázány všechny rotace kromě rotace kolem vertikální osy. Proto bylo nakonec přistoupeno k vlastní implementaci působení síly exploze na herní postavu. Tato implementace počítá směr a sílu působení exploze na herní postavu pouze vůči jejímu středu, jak je znázorněno na obrázku 8.7.



Obrázek 8.5: obrázek popisuje vyhodnocení exploze poté, co raketa koliduje s jiným herním objektem. Poté, co je exploze vyhodnocena, je raketa odstraněna z herního dema.



Obrázek 8.6: obrázek popisuje vyhodnocení působení síly exploze na herní postavu pomocí metody AddExplosionForce komponenty Rigidbody, již obsahuje herní postava. V tomto případě síla působí na celou plochu herní postavy a výsledný pohyb postavy by byl převážně horizontální.



Obrázek 8.7: Vliv exploze rakety na herní postavu je simulován tak, že se vypočítá směr a vzdálenost pouze mezi bodem exploze a středem herní postavy. Poté je v daném směru právě na střed herní postavy aplikována síla exploze, která s narůstající vzdáleností herní postavy od bodu exploze slábne.

Kapitola 9

Implementace procedurálního generátoru

Implementace procedurálního generátoru je rozdělena do více částí, a to na generátor náhodných hodnot, páteřní generátor, pomocné generátory a dílčí generátory mapy. Jednotlivými částmi procedurálního generátoru se zabývají následující kapitoly.

9.1 Generátor náhodných hodnot

Generátor náhodných hodnot, nebo spíše pseudonáhodných hodnot, je pouhá nadstavba generátoru Random, jenž dodává Unity3D [6]. Účelem této nadstavby je správa seedu generátoru a jeho inkrementu, která umožňuje stejný výsledek opětovného generování se stejnými vstupními hodnotami. Seed a inkrement generátoru je možné nastavit skrze uživatelské rozhraní.

Generátor náhodných hodnot obsahuje metodu ResetSeed, která umožňuje obnovit hodnoty seedu a inkrementu na hodnoty nastavené uživatelem, a metodu Evaluate, která nastaví hodnotu seedu Random generátoru na aktuální hodnotu uloženou v implementaci nadstavby, tuto hodnotu inkrementuje pro příští použití o inkrement uložený v implementaci nadstavby a navrátí náhodně vygenerovanou hodnotu metody Range třídy Random.

9.2 Pomocné generátory

Pomocné generátory jsou generátory různých herních objektů, které jsou využívány dílčími generátory herní mapy. Jejich cílem je sjednotit obecné generování objektů na jedno místo.

9.2.1 Checkpoint generátor

Herní mapa obsahuje dva typy checkpointů – pouhou plochu checkpointu a celou místnost checkpointu. Pouhou plochou checkpointu se rozumí zóna, která snímá vstup herní postavy a bývá využívána primárně na začátku sekce herní mapy, neboť tyto začátky se u různých sekcí mohou lišit. Místností checkpointu se rozumí herní plocha, strop a pouhá plocha checkpointu obehnaná zdí, která je vygenerována pomocí generátorů zdí – viz. kapitola 9.2.2.

Při tvorbě checkpointu i místnosti je vždy specifikována šířka, délka, výška a hloubka checkpointu a jsou vytvořeny klony podlahy, stropu a zóny checkpointu, jež jsou specifi-

kovány v nastavení generátoru, a jejich rozměry jsou upraveny podle hodnot předaných parametry metod.

9.2.2 Generátor zdí

Při generování zdí chodeb, jejichž rozměry a tvary se při každém generování mohou lišit – na základě vstupních hodnot generování, či prostředí generovaného pro schopnost – je vhodné dynamicky generovat nový herní objekt originálního tvaru, nikoliv klonovat předdefinované objekty.

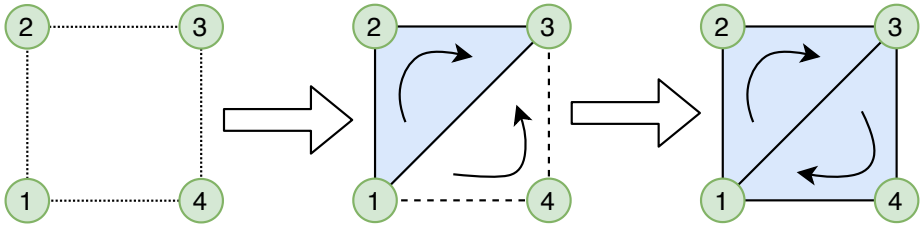
Pro dynamické generování herního objektu je potřeba vytvořit nový herní objekt a přidat mu potřebné komponenty pro vykreslování objektu v herním prostoru – Mesh Renderer a Mesh Filter, ačkoliv při tvorbě herního objektu, a jeho tvaru, je potřeba pracovat pouze s komponentou Mesh Filter, neboť komponentě Mesh Renderer stačí pouze předat materiál herního objektu, specifikující jeho barvu a další vizuální vlastnosti.

Chceme-li definovat vlastní objekt pomocí vlastní definované Meshe – která je později předána komponentě Mesh Filter pro zpracování – musíme brát v potaz, že mesh je vykreslována pomocí trojúhelníků. Trojúhelníky meshe, které jsou vykreslované, jsou definovány třemi po sobě jdoucími body. Tyto trojúhelníky jsou vykreslovány pouze tehdy, je-li definované pořadí bodů v aktuálním úhlu pohledu ve směru hodinových ručiček – v opačném případě trojúhelník vykreslen není – viz. obrázek 9.1. Proto, aby generovaná zeď byla vidět z vnějšku i zevnitř, je potřeba definovat každý trojúhelník dvakrát – jednou je pořadí bodů ve směru hodinových ručiček, podruhé v opačném směru.

Proto byl vytvořen generátor zdí, který umožňuje specifikovat bod, kde se má nacházet další část chodby, a rozměry chodby, a o zbytek – rozmístění a propojení jednotlivých bodů meshe – se postará právě generátor zdí.

Zeď, která je pomocí generátorů zdí generována, obsahuje také komponentu Mesh Collider, tedy komponentu, která vyhodnocuje případné kolize herního objektu s ostatními herními objekty – tato komponenta tedy zajišťuje, aby hráč nemohl projít skrze dynamicky generované zdi. Nicméně je potřeba této komponentě předat mesh, na základě které vyhodnotí tvar kolizních zón – tzn. poté, co je dokončeno generování celé chodby, všech zdí, tedy generování celé meshe, je tato mesh předána komponentě Mesh Collider.

Dále herní objekt prezentující podlahu generované chodby obsahuje komponentu Void, jejíž jediný účel je detekovat kolizi s herní postavou, a v případě, že se tak stane, tak herní postavu přemístit na poslední známý checkpoint.



Obrázek 9.1: Obrázek popisuje, jak správně definovat mesh. Mesh bývá definována trojúhelníky, které jsou definovány třemi po sobě jdoucími body, kde každý z nich znázorňuje bod v trojrozměrném prostoru. Na obrázku jsou znázorněny 4 body v prostoru, které je potřeba spojit do dvou trojúhelníku tak, aby se vykreslily z aktuálního úhlu pohledu. Je-li pořadí bodů trojúhelníku z aktuálního úhlu pohledu ve směru hodinových ručiček (body 1, 2, 3, či body 1, 3, 4), je trojúhelník vykreslen, v opačném případě (body 1, 4, 3) nikoliv – jak znázorňuje prostřední obrázek, kde modrá plocha trojúhelníku znázorňuje vyobrazený trojúhelník a bílá plocha s vynechávanou čarou znázorňující hrany trojúhelníku, který aktuálně není vidět.

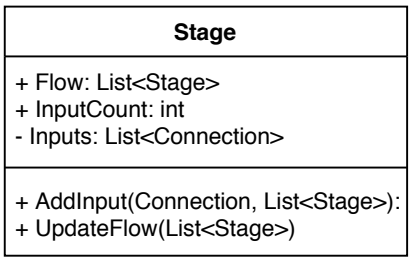
9.3 Páteřní generátor

Úkolem páteřního generátoru je správa celého generování herní mapy. Páteřní generátor tedy nejprve vygeneruje pseudomapu, kterou za pomocí pomocných a dílčích generátorů realizuje. Ve výsledku páteřní generátor propojuje výsledky dílčích generátorů realizujících vygenerovaný návrh herní mapy.

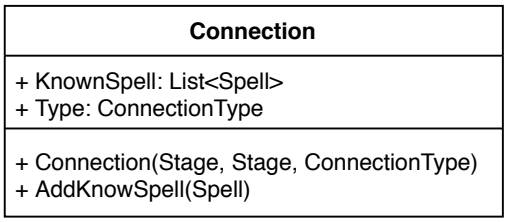
9.3.1 Pseudomapa

Pseudomapa je základ celého generování herní mapy. Pseudomapa určuje, jak budou na sebe jednotlivé části herní mapy navazovat a jak budou na sobě závislé. Podrobný popis algoritmu a tvorby pseudomapy se lze dočíst v kapitole 6.1.1 a u obrázku 6.2.

Při implementaci generování pseudomapy byly vytvořeny dvě pomocné třídy, a to třída Stage (viz. obrázek 9.2) a třída Connection (viz. obrázek 9.3). Tyto třídy jsou využity pro prezentaci propojení a závislosti jednotlivých částí herní mapy.



Obrázek 9.2: Obrázek popisuje pomocnou třídu Stage, která představuje herní plochu, jež může být začátkem herní mapy, koncem herní mapy, plochou, kde se herní postava naučí novou schopnost, nebo přechodnou plochou. Třída Stage obsahuje seznam ploch, nebo-li Stage, které následují po dané Stage, nebo-li ploch, kterých z dané Stage lze dosáhnout. Dále třída Stage obsahuje vlastnost InputCount, která je čítačem ploch, které jsou přímo spojené s danou Stage – kdy maximální počet jsou 4 – a seznam samotných propojů s danými plochami, vlastnost Inputs. Třída Stage také obsahuje metodu AddInput, která propojuje danou Stage se specifikovaným propojem a aktualizuje list Flow – viz. algoritmus 6 – a metodu UpdateFlow, která rekurzivním voláním předává aktualizaci listu Flow – viz. algoritmus 5.



Obrázek 9.3: Obrázek popisuje pomocnou třídu Connection, která představuje propojení dvou herních ploch, nebo-li Stage. V pozdější fázi, při generování herní mapy pomocí dílčích generátorů, slouží tato třída ke specifikování schopností herní postavy, pro které by daná část herní mapy měla být vygenerovaná jako překážková dráha. Třída Connection obsahuje seznam schopností, které herní postava při průchodu tímto propojem již zná, a typ propoje, který určuje speciální vlastnosti propoje. Dále třída Connection obsahuje metodu Connection, která propojuje dvě Stage do jednoho Connection (viz. algoritmus 7), a metodu AddKnowSpell, která přidává specifickou schopnost do seznamu schopností, které herní postava na daném propoji již zná – viz. algoritmus 8.

Algorithm 5: Algoritmus popisuje řešení rekurzivního předávání známých ploch (Stage), tedy tvoření seznamu oblastí herní mapy, kterých lze dosáhnout při pokračování ve zdolávání herní mapy. Daný seznam je potřeba pro budoucí předávání již známých schopností, tedy informace o tom, že herní postava by v dané oblasti specifické schopnosti již měla umět.

```
1 function UpdateFlow(novyFlow)
2     AktualniFlow += novyFlow
3     AktualniFlow.Distinct()
4     listPropoju = GLOBAL.VsechnyPropoje
5     listPropoju = listPropoju.Vyber(propoje vedoucí od Startu k této Stage)
6     listPropoju.VstupniStage.UpdateFlow(AktualniFlow);
```

Algorithm 6: Algoritmus popisuje, jak je řešené připojení Stage k propoji, který propojuje dvě Stage, a počáteční spuštění rekurzivního předávání již známých ploch, tedy Stage, voláním funkce UpdateFlow – viz. algoritmus 5. V případě, že Stage nemá již zabrané všechny vstupní hrany, které jsou maximálně 4, přijme připojení k danému propoji a spustí rekurzivní předávání známých ploch – viz. algoritmus 5.

```
1 function AddInput(propoj, pripojenyFlow)
2     if VolneVstupniHrany > 0 then
3         VolneVstupniHrany -= 1
4         Vstupy.Pridat(propoj)
5         UpdateFlow(pripojenyFlow)
6         return true
7     return false
```

Algorithm 7: Algoritmus popisuje, jak je za využití funkce AddInput, viz. algoritmus 6, v průběhu propojování dvou herních ploch, nebo-li Stage, předáván seznam již známých ploch. V případě, že se nejedná o zpětný propoj (viz. dále), je předán vstupní ploše celý seznam známých ploch výstupní plochy včetně výstupní plochy samotné. Výstupní ploše žádný seznam předáván není, neboť předávání je pouze jednosměrné – výstupní ploše je pouze oznámeno, že byla připojena. V případě zpětného propoje, tedy když je vedlejší větev průchodu připojena zpět k rozvětvení, se seznam známých ploch nepředává, neboť by došlo k zacyklení.

```
1 constructor Connection(od, do, typ)
2     flow = empty
3     if typ != ZpetnyPropoj then
4         flow = do.Flow + do
5     od.AddInput(this, flow)
6     do.AddInput(this, emptyList)
```

Algorithm 8: Algoritmus popisuje předávání informace o tom, ve které části herní mapy herní postava již danou schopnost umí. Schopnost je předávána pouze tehdy, nejedná-li se o zpětný propoj z konce větve na rozvětvení, a je předávána ve směru, ve kterém se nenachází samotná plošina, na které se herní postava danou schopnost teprve naučí. Metoda je poprvé volána z plochy, kde se herní postava učí danou schopnost, a poté je rekurzivně volána až do cíle herní mapy.

```
1 function AddKnowSpell(schopnost)
2     if this != ZpetnyPropoj and nasledujiciVetev.Neobsahuje(schopnost) then
3         ZnameSchopnosti += schopnost
4         listPropoju = GLOBAL.VsechnyPropoje
5         listPropoju = listPropoju.Vyber(propoje vedoucí od tohoto dále)
6         listPropoju.AddKnowSpell(schopnost)
```

9.3.2 Realizace mapy

Před tím, než je generována pseudomapa a herní mapa samotná, jsou nejprve smazány všechny herní objekty předchozího generování herní mapy, herní postavě jsou odebrány všechny schopnosti, které umí, generátor náhodných hodnot je obnoven do počátečního stavu zadaných hodnot seedu a inkrementu a herní postava je přemístěna na pozici budoucí první herní plochy – plochy Start.

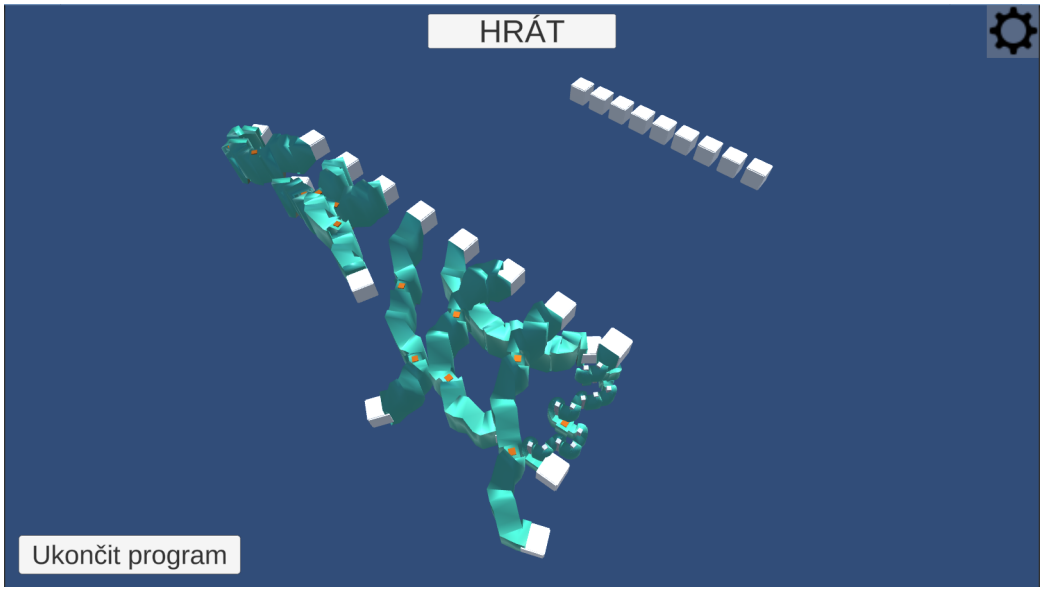
Poté, co je vygenerována pseudomapa, jsou postupně vedle sebe vygenerované všechny herní plochy, Stage, specifikované pseudomapou. Důvod, proč jsou nejprve generovány Stage a proč jsou generovány vedle sebe, je popsán v kapitole 9.3.3. Zároveň je každá Stage kontrolována, zda se nejedná o plochu, kde se herní postava naučí novou schopnost – v případě, že tomu tak je, je na plochu přidána zóna, po jejímž kontaktu s herní postavou je herní postavě přidána daná schopnost.

Když jsou všechny Stage vygenerované, začne páteřní generátor na základě pseudomapy postupně generovat všechny propoje herních ploch, tedy průchody mapou – části map – jež mají představovat překážkovou dráhu pro hráče. Páteřní generátor cyklicky projde veškeré propoje specifikované pseudomapou a za pomocí pomocných a dílčích generátorů vygeneruje část herní mapy – viz. algoritmus 9.

Na závěr jsou všechny Stage uzavřeny – každá Stage má 4 hrany a každá hrana, která není ucelená portálem na průchod, je zacelena zdí – to aby ze Stage byla uzavřená místnost ve tvaru krychle. Do poslední Stage, prezentující konec mapy, je umístěna zpráva o dosažení konce mapy.

Algorithm 9: Algoritmus popisuje úkol páteřního generátoru při generování části herní mapy – průchodu, překážkové dráhy. Páteřní generátor nejprve vygeneruje vstupní místnost průchodu, do které, za využití generátoru checkpointů – viz. kapitola 9.2.1 – umístí i první checkpoint celého průchodu. Následně vstupní místnost propojí se Stage, což mezi nimi vytvoří propojující portál – Output gate. Tento portál je pouze jednosměrný, neboť není potřeba, aby se hráč jakkoliv vracel, neboť průchod mapou je pouze jednosměrný. Poté je, na základě schopností, které bude herní postava již umět, vybrán dílčí generátor, který vygeneruje celý průchod. Na závěr je vygenerována výstupní místnost průchodu, která se propojí se Stage, což mezi nimi opět vytvoří propojující portál – Input gate – který je opět jednosměrný.

```
1  function GenerateFlow(connection)
2      flowInput = generateFlowInput(actualPosition)
3      CheckpointGenerator.GenerateCheckpoint(actualPosition)
4      connection.From.AddOutputGate(flowInput)
5      WallsGenerator.Init(actualPosition)
6      generator = GeneratorsContainer.Get(connection.KnownSpells)
7      generator.GenerateMap(actualPosition)
8      flowOutput = generateFlowOutput(actualPosition)
9      connection.To.AddInputGate(flowOutput)
10     WallsGenerator.Finish(actualPosition)
```



Obrázek 9.4: Obrázek vyobrazuje vygenerovanou herní mapu – v pravé vrchní části se nachází veškeré Stage a vprostřed, což je znázorněno zelenými objekty, jsou jednotlivé průchody herní mapou, které propojují Stage.

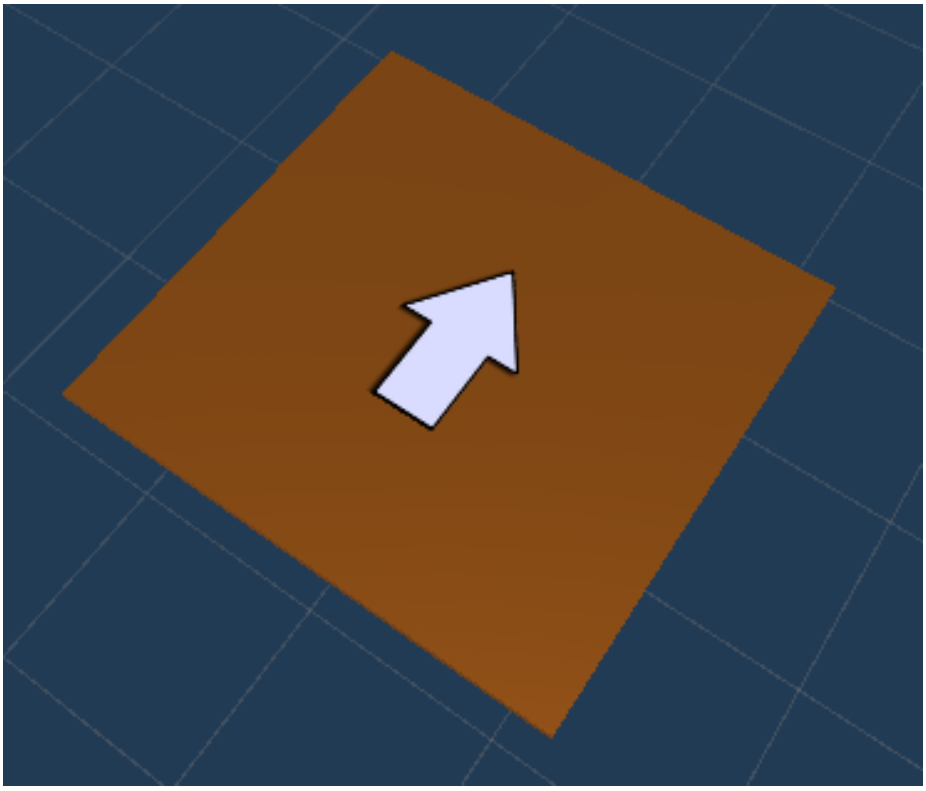
9.3.3 Řešení kolize jednotlivých částí map

Kdyby jednotlivé části map, průchody, byly napojené přímo na Stage, musel by být průchod celou mapou bez žádného větvení – neboť dosáhnout toho, aby vedlejší větev opět navazovala na hlavní větev průchodu mapou, by bylo velice náročné. Proto se mezi Stage a průchody prochází skrze portály, které hráče mezi nimi přemisťují. Jelikož není potřeba navazovat průchody přímo na Stage a obráceně, neboť hráč se mezi nimi stejně pohybuje pomocí portálů, jsou všechny stage generovány vedle sebe a všechny průchody o kousek dále od Stage jsou vedle sebe generovány též – za účelem lepší přehlednosti viz. obrázek 9.4.

Dále je v moment, kdy hráč prochází portálem Output gate, a je přemisťován na začátek specifického průchodu, daný průchod odhalen – pokud již nebyl – a všechny ostatní průchody, na kterých se hráč právě nenachází, jsou skryty. Účelem toho je zabránit kolizím, které by mohly průchody mezi sebou zapříčinit.

9.3.4 UX - orientace na mapě

Za určitých okolností si nemusí být hráč jist, kde se na herní mapě nachází a kterým směrem má průchodem pokračovat, aby dosáhl cíle. Za těchto okolností byly na herní objekty v průchodech přidány šipky, které naznačují správný směr průchodu – viz. obrázek 9.5. Pokud se tedy hráč vydá ve směru těchto šipek, dosáhne konce průchodu a bude moci pokračovat na další Stage.



Obrázek 9.5: Herní objekt, jež je součástí průchodu, se šipkou znázorňující korektní směr průchodu.

9.4 Dílčí generátory mapy

Vzhledem k tomu, že herní postava může mít nespočet schopností, a pro každou schopnost herní postavy může být opět nespočet druhů překážkových drah, není vhodné implementovat generování celé herní mapy do jednoho velkého generátoru, neboť s přibývajícími schopnostmi a překážkovými drahami by bylo potřeba tento generátor modifikovat. Namísto toho je generátor rozdělen na dílčí generátory, z nichž každý musí implementovat rozhraní dílčího generátoru – viz. kapitola 9.4.1. Díky tomu mohou být všechny dílčí generátory shromážděny společně v jednom kontejneru – viz. kapitola 9.4.2. S novou příchozí schopností herní postavy a novou překážkovou mapou stačí implementovat nový dílčí generátor, který bude implementovat zmíněné rozhraní a bude umístěn do zmíněného kontejneru.

9.4.1 Abstraktní generátor

Abstraktní generátor je implementován jako abstraktní třída s jedinou metodou, a to GenerateMapFlow. Abstraktní generátor byl původně plánován jako rozhraní, nicméně vývojové prostředí Unity3D neumožňuje zobrazovat rozhraní v inspektoru editoru. Vzhledem k tomu, že kontejner dílčích generátorů je navrhnut tak, že jednotlivé dílčí generátory jsou do něj vkládány skrze inspektor editoru, bylo nutné od rozhraní dílčích generátorů odstoupit a vytvořit místo něj zmíněnou abstraktní třídu, kterou dílčí generátory dědí.

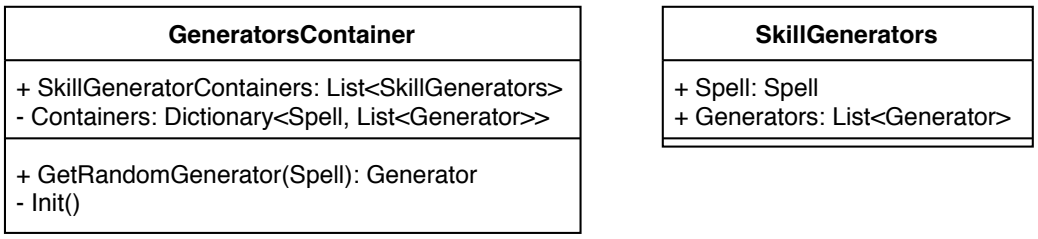
Metoda GenerateMapFlow generuje část herní mapy, překážkovou dráhu, počínaje od pozice předané parametrem. Mapa, která je dílčím generátorem generována, je specifická

pro určitou schopnost herní postavy, či dokonce pro specifické využití dané schopnosti. Poté, co dílčí generátor vygeneruje specifickou část herní mapy, předá zpět modifikovanou pozici, která tentokrát odkazuje na konec vygenerované části herní mapy.

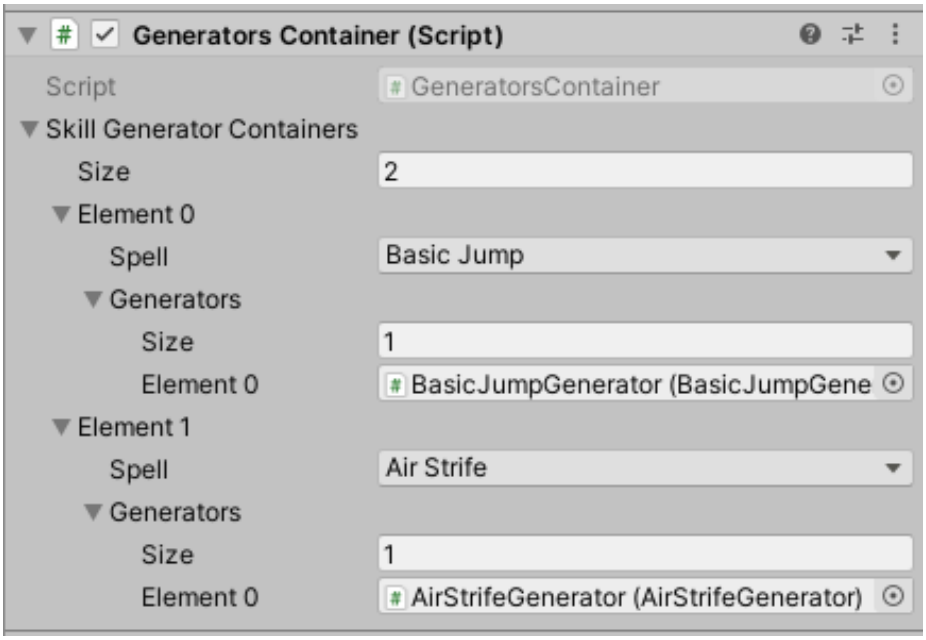
9.4.2 Kontejner dílčích generátorů

Kontejner dílčích generátorů slouží ke sjednocení všech dílčích generátorů částí herní mapy na jedno místo s možností náhodného výběru generátoru na základě použité schopnosti herní postavy.

Při implementaci je vhodné využít kolekci klíčů a hodnot Dictionary, kde schopnost herní postavy představuje klíč a seznam dílčích generátorů pro danou schopnost představuje hodnotu. Nicméně, vývojové prostředí Unity3D neumožňuje zobrazování Dictionary v inspektoru editoru, natož editaci – přidávání a odebírání dílčích generátorů do kolekce. Tuto skutečnost je potřeba obejít, a proto je implementována pomocná struktura SkillGenerators, která obsahuje dvě vlastnosti – seznam dílčích generátorů a schopnost, pro kterou jsou ony dílčí generátory určeny (viz. obrázek 9.6). Následně je v kontejneru dílčích generátoru vytvořen seznam pro SkillGenerators, který je již plně přístupný a funkční v inspektoru editoru Unity3D – viz. obrázek 9.7. Při inicializaci aplikace je seznam SkillGenerators, do kterého jsou v průběhu vývoje přiřazeny dílčí generátory, transformován do zmíněné požadované kolekce Dictionary, se kterou je posléze pracováno.



Obrázek 9.6: Obrázek vyobrazuje třídu GeneratorsContainer, reprezentující kontejner dílčích generátorů částí herní mapy, a strukturu SkillGenerators, využívanou třídou GeneratorsContainer. Využití struktury SkillGenerators, logika zdánlivě duplicitních vlastností SkillGeneratorContainers a Containers a metoda Init jsou vysvětleny v kapitole 9.4.2. Metoda GetRandomGenerator třídy GeneratorsContainer slouží k náhodnému výběru jednoho z dílčích generátorů pro specifickou schopnost herní postavy specifikovanou parametrem metody.

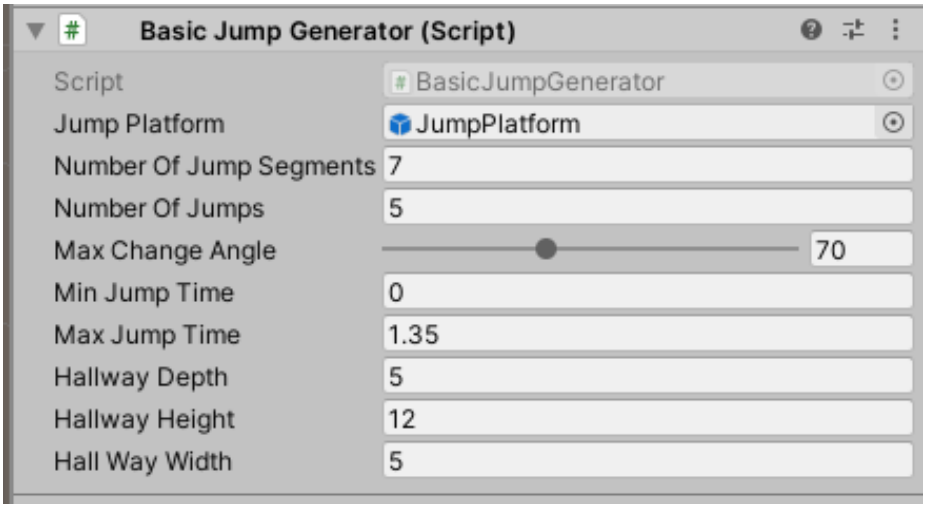


Obrázek 9.7: Vyobrazení třídy GeneratorsContainer jako komponenty herního objektu viditelné v inspektoru editoru Unity3D. Komponenta má již specifikován jeden dílčí generátor části herní mapy pro základní schopnost herní postavy Basic Jump, tedy obyčejné skákání, a jeden dílčí generátor pro schopnost Air Strafe.

9.4.3 Generátor základního skoku

Generátor základního skoku je základní dílčí generátor, který je dostupný od samého začátku generování herní mapy, neboť k překonání překážkové dráhy, kterou generuje, není zapotřebí žádné speciální schopnosti herní postavy.

Část herní mapy, překážková dráha, je v rámci implementace zděděné abstraktní metody GenerateMapFlow abstraktní třídy Generator, generována na základě parametrů specifikovaných skrze inspektor editoru Unity3D – viz. obrázek 9.8. Implementace nejprve vypočítá konečnou pozici herní postavy po výskoku (viz. obrázek 9.9 a algoritmus 10), posléze, nejedná-li se o poslední skok patřičného segmentu, na dané místo vygeneruje dopadovou plošinu – viz. obrázek 9.5 – a následně je vygenerována část zdí průchodu pomocí Generátoru zdí – viz. kapitola 9.2.2. Poté, co je celé generování dokončeno, vrací metoda společně s pozicí konce průchodu veškeré vygenerované herní objekty jako výsledek.



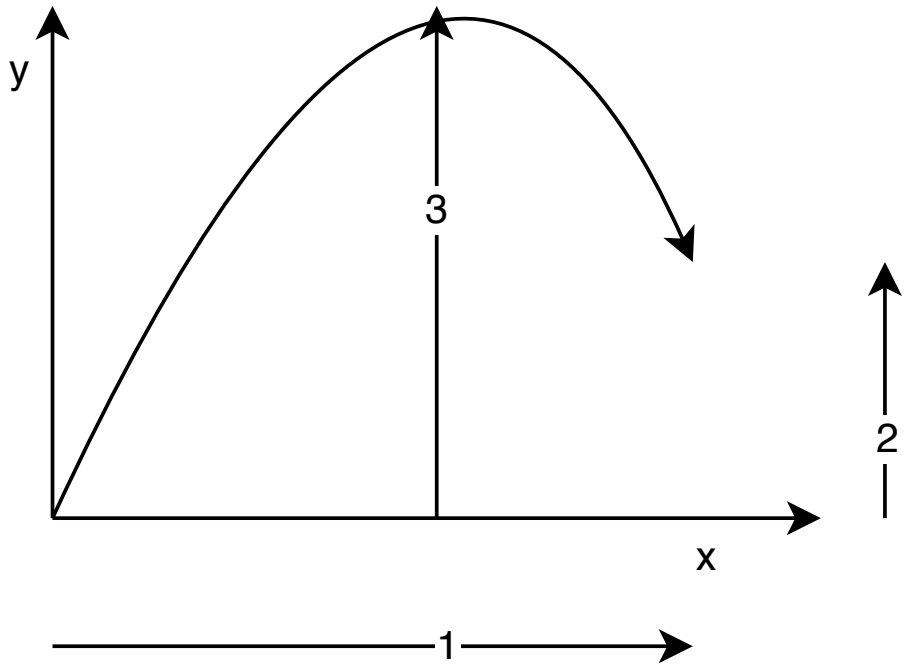
Obrázek 9.8: Vyobrazení rozhraní komponenty Generátoru základního skoku (viz. kapitola 9.4.3) v inspektoru editoru Unity3D. Rozhraní umožňuje editovat parametry generování části herní mapy – konkrétně počet segmentů, kdy po každém segmentu je generován checkpoint (viz. kapitola 9.2.1), počet skoků na segment, maximální úhel změny směru pohybu na každý skok, minimální a maximální dobu trvání skoku a parametry rozměrů chodby průchodu mapou.

$$V_\mathrm{f} = V_\mathrm{i} + at \tag{9.1}$$

$$d = V_\mathrm{i}t + \frac{at^2}{2} \tag{9.2}$$

Kinematické rovnice, využívané při generování pozice dopadové plošiny pro herní postavu, jež využila patřičné schopnosti.

- d – vzdálenost
- t – čas
- V_i – počáteční rychlost
- V_f – konečná rychlost
- a – zrychlení



Obrázek 9.9: Výpočet výsledné pozice herní postavy po základním skoku. Nejprve je potřeba si určit, jak dlouho se má herní postava nacházet ve vzduchu – minimální doba skoku *3* je doba, jakou trvá, než herní postava dosáhne maximální výšky. Minimální doba skoku *3* se vypočítá pomocí vzorce 9.1, kdy konečná rychlost V_f je 0, počáteční rychlost V_i je síla skoku, zrychlení a je gravitace a výsledná proměnná čas t je doba skoku, než herní postava dosáhne maximální výšky. Zadaná doba skoku herní postavy je zaměněna za zmíněnou minimální dobu v případě, že zadaná doba daného minima nedosáhla. Poté, co se konečně definuje doba, kterou herní postava stráví ve vzduchu, se pomocí vzorce 9.2 vypočítá vzdálenost *1* a výška *2* výsledné pozice herní postavy. V případě vzdálenosti lze ignorovat druhou polovinu vzorce, neboť zrychlení a je nulové, a výsledná vzdálenost d se vypočítá na základě rychlosti pohybu herní postavy V_i a doby t, kterou herní postava stráví ve vzduchu. Výška výsledné pozice se vypočítá na základě síly skoku V_i, doby skoku t a gravitačního zrychlení a.

Algorithm 10: Popis generování výsledné pozice po základním skoku. Algoritmus pomocí Generátoru náhodných hodnot – viz. kapitola 9.1 – nejprve vygeneruje náhodné hodnoty od 0 do 1 včetně, kde první náhodnou hodnotu upraví do rozsahu -1 až 1 včetně a vynásobí tím proměnnou Maximálního úhlu změny směru pohybu při skoku – tím získá úhel změny směru pro aktuálně generovaný skok. Druhou náhodnou hodnotou je vynásoben rozdíl Maximální a Minimální doby skoku, k čemuž je přičtena právě Minimální doba skoku a výsledkem je doba trvání aktuálně generovaného skoku. Doba skoku je předána funkci GetPositionOverTime – viz. algoritmus 11 – která vygeneruje výslednou pozici právě generovaného skoku. Výsledná pozice a rotace jsou navráceným výsledkem této funkce.

```
1  function GenerateJumpPosition()
2      tmpRndVal = RandomGenerator.Evaluate()
3      jumpAngle = (tmpRndVal - 0.5f) * 2 * MaxChangeAngle
4
5      tmpRndVal = RandomGenerator.Evaluate()
6      jumpTime = ((MaxJT - MinJT) * tmpRndVal) + MinJT
7      jumpPosition = GetPositionOverTime(jumpTime)
8
9      return (jumpPosition, jumpAngle)
```

Algorithm 11: Algoritmus popisuje výpočet výsledné pozice herní postavy po určitém čase od aplikování základního skoku. Nejprve je vypočítána doba potřebná k dosažení maximální výšky výskoku, k jejímuž výpočtu je aplikován kinematický vzorec 9.1. V případě, že zadaný čas, pro který má být generována výsledná pozice, je nižší, než doba potřebná k dosáhnutí maximální výšky, je tento zadaný čas navýšen právě na danou vypočtenou dobu – pokud by herní postava narazila na cílovou plošinu dříve, než by dosáhla maximální výšky, nemohla by dané plošiny dosáhnout. Poté, pomocí kinematického vzorce 9.2, je vypočtena vzdálenost, kterou herní postava při výskoku po specifikované době urazí, a výsledná výška, ve které se herní postava po specifikované době bude nacházet. Tyto výsledné hodnoty jsou posléze navráceny.

```
1  function GetPositionOverTime(time)
2      xVi = Player.MovementSpeed
3      yVi = Player.JumpPower
4      yA = Player.FallingSpeed
5
6      tUp = -yVi / yA
7      t = time >= tUp ? time : tUp
8
9      result.x = xVi * t
10     result.y = (yVi * t) + (yA * Mathf.Pow(t, 2) / 2)
11
12     return result;
```

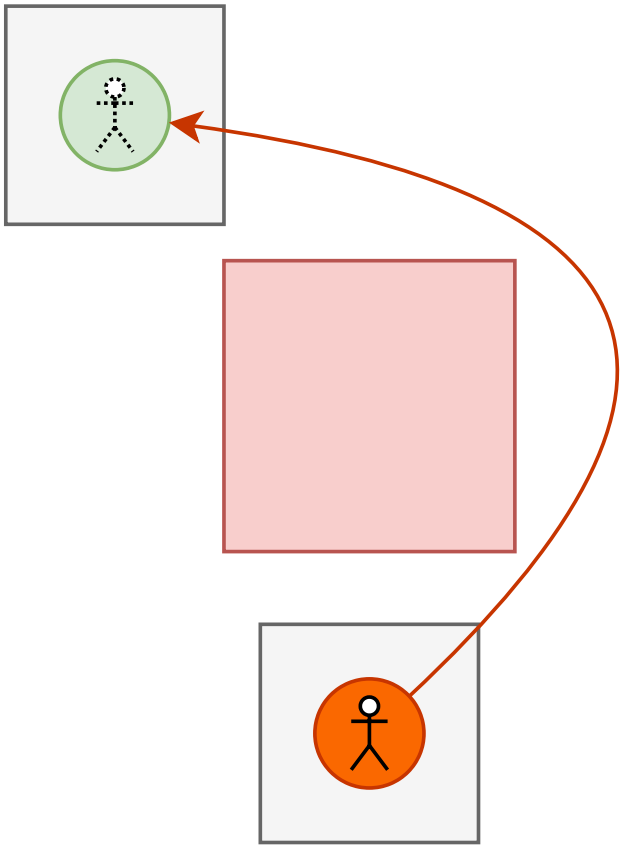
9.4.4 Generátor schopnosti Air strafe

Dílčí generátor herní mapy pro schopnost Air strafe – viz. kapitola 8.2.1 – generuje část herní mapy, průchod, kde hráč za využití schopnosti Air strafe má za úkol překonávat překážky tohoto průchodu tak, že v průběhu výskoku, za využití schopnosti, upravuje směr svého pohybu do křivky tak, že obskočí danou překážku kolem a dopadne např. o patro níže – viz. obrázek 9.10. Generátor je tedy implementován tak, že na základě parametrů, zadaných skrze inspektor editoru Unity3D – viz. obrázek 9.11 – je generována část herní mapy s překážkami, které lze překonávat pouze krouživým skokem, popř. následujícím pádem. Hráč má za úkol pomocí schopnosti Air strafe měnit směr letu po výskoku tak, aby kroužil kolem herního objektu, představujíc překážku, dokud nedopadne na cílovou herní plošinu.

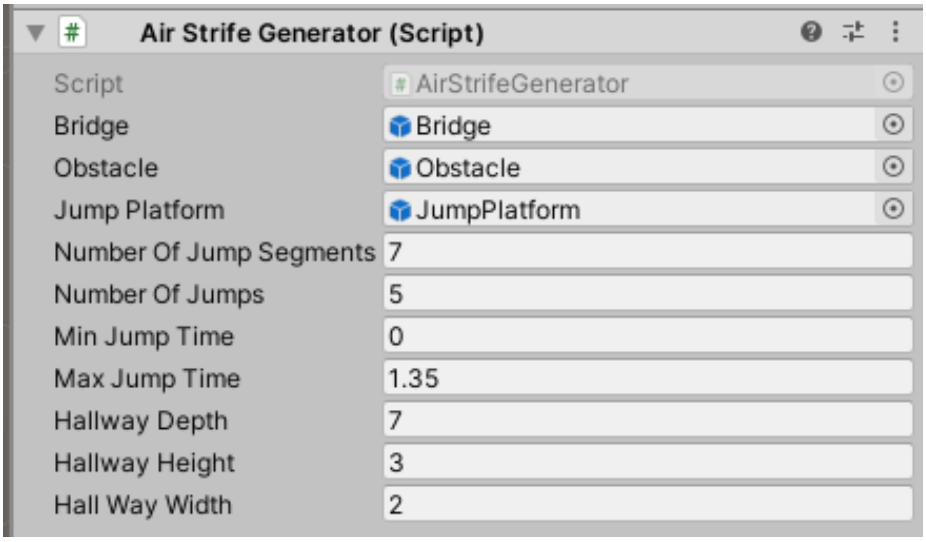
Aby se zamezilo kolizím mezi jednotlivými překážkami průchodu, je každá překážka, každý skok za využití schopnosti Air strafe, oddělena krátkou chodbou. Toto opatření zamezí protínání zdí jednotlivých skoků kolem herní překážky, neboť v případě, že by uživatel nastavil širší chodbu pro průchod částí herní mapy, tyto širší chodby by se navzájem protínaly. Díky chodbě, která jednotlivé překážky/skoky, od sebe odděluje, je tomuto protnutí zamezeno.

Směr překážkové dráhy se postupným krouživým skákáním kolem překážek může snadno stočit zpět k překážkové dráze a opět způsobit kolizi. Aby se zamezilo této hrozbě neustálého postupného otáčení, je zavedena podmínka, že celkový součet úhlů změny směru trati mezi začátkem skoku a koncem skoku, nesmí spadat mimo interval -80, až +80 stupňů. Proto se vždy před generováním překážky nejprve zjistí místa, kde lze umístit dopadovou plošinu a stále udržet úhel směru pokračování průchodu ve zmíněném intervalu – viz. obrázek 9.12.

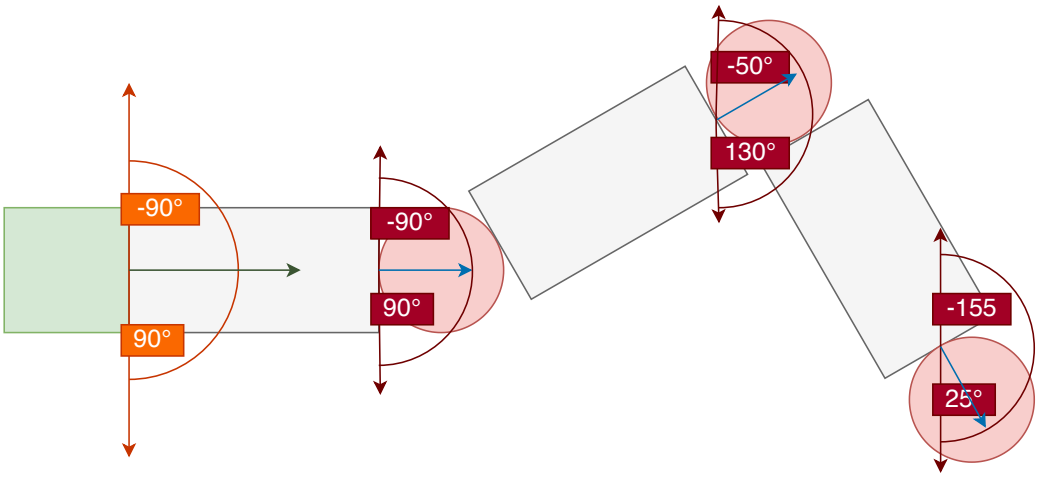
Výsledná pozice dopadové plošiny překážky je generována za pomoci simulace skoku a aplikace schopnosti Air strafe, jež je umožněna skriptem spravující ovládání, pohyb a schopnosti herní postavy. Implementace generátoru nejprve využije implementaci skriptu herní postavy, která na základě zadaných parametrů – směr pohybu, rychlost a délka – vyhodnotí výslednou pozici herní postavy. Poté implementace generátoru normalizuje výslednou pozici tak, aby splňovala podmínku výsledného směru průchodu – viz. obrázek 9.12 – a na závěr je vygenerována výsledná dopadová plošina.



Obrázek 9.10: Obrázek znázorňuje vzor, podle kterého jsou generovány překážky dílčím generátorem herní mapy pro schopnost Air strafe – viz. kapitola 9.4.4. Vyobrazený šedý čtverec s panáčkem z plné čáry v oranžovém kolečku znázorňuje počáteční pozici, ze které má hráč za úkol s herní postavou vyskočit, za pomocí schopnosti Air strafe zakřivit dráhu letu herní postavy a dopadnou za překážkou, znázorněnou červeným čtvercem, na dopadovou pozici, kterou znázorňuje opět šedý čtverec, tentokrát s panáčkem z tečkované čáry v zeleném kolečku. Oranžová křivka znázorňuje dráhu letu ovlivněnou schopností herní postavy Air strafe.



Obrázek 9.11: Vyobrazení rozhraní komponenty dílčího generátoru herní mapy pro schopnost herní postavy Air strafe (viz. kapitola 9.4.4) v inspektoru editoru Unity3D. Rozhraní umožňuje editovat parametry generované části herní mapy – konkrétně počet skoků na jeden segment, po které je generován Checkpoint (viz. kapitola 9.2.1), počet segmentů na část herní mapy, minimální a maximální doba skoku za využití schopnosti Air strafe a rozměry chodby tvořící průchod části herní mapy.



Obrázek 9.12: Obrázek znázorňuje metody řešení kolize části herní mapy generované generátorem pro schopnost herní postavy Air strafe – viz. kapitola 9.4.4. Zelený čtverec znázorňuje začátek průchodu části herní mapy, zelená šipka počáteční směr průchodu a oranžové šipky s půlkruhem a specifikovanými stupni úhlů znázorňují maximální úhly, o kolik se může celkový směr průchodu odchýlit od počátečního směru. Šedý obdélník znázorňuje chodbu, která odděluje jednotlivé překážky, skoky, od sebe, aby jednotlivé překážky mezi sebou netvořily kolizi. Červené kruhy znázorňují překážky, které má hráč za úkol za pomoci schopnosti Air strafe (viz. kapitola 8.2.1) zdolat. Modrá šipka znázorňuje aktuální směr průchodu a červené šipky s půlkruhem a specifikovanými stupni úhlů znázorňují maximální úhly, o kolik se může aktuální směr průchodu změnit tak, aby stále splňoval podmínku maximální odchylky celkového směru průchodu od počátečního směru.

Kapitola 10

Implementace rozhraní pro testování a prezentaci

Vzhledem k tomu, že je procedurální generátor nástrojem pro generování herní mapy případné výsledné hry, je jeho nastavení dostupné skrze rozhraní inspektoru editoru Unity3D. Nicméně, toto rozhraní je dostupné pouze vývojářům, externím uživatelům herního dema nikoliv. Proto je vytvořeno grafické rozhraní, skrze které lze nastavit potřebné vlastnosti herní postavy a všech generátorů herní mapy.

10.1 Grafické uživatelské rozhraní

Aby uživatel měl možnost nastavit všechny potřebné parametry ke generování herní mapy, je implementováno stránkovací rozhraní, ve kterém lze na jednotlivých stránkách nastavit parametry herní postavy, globální parametry všech generátorů a jednotlivé dílčí generátory – viz. algoritmus 12. Toto stránkování nastavení automaticky načítá data, jež byla nastavena posledně, a v případě změny nová data opět ukládá, aby mohla být znovu automaticky načtena – viz. kapitola 10.3.

Dále uživatelské rozhraní umožňuje si vygenerovanou herní mapu prohlédnout – za pomocí ovládacích prvků volného pohybu, viz. kapitola 10.2. Poté, co je uživatel s vygenerovanou mapou spokojen, se dokonce může přepnout do herního módu a danou mapu si projít jako hráč ovládající herní postavu.

Uživatelské rozhraní navíc umožňuje vstup na herní mapu, kde byly vyvíjeny a testovány všechny schopnosti herní postavy. Vzhledem k tomu, že herní demo neobsahuje generátory herní mapy pro všechny schopnosti herní postavy – obsahuje pouze generátor pro základní skok – viz. kapitola 9.4.3 – a generátor pro schopnost Air strafe viz. kapitola 9.4.4 – je možné si zbylé schopnosti herní postavy otestovat alespoň ve vývojové mapě.

Algorithm 12: Algoritmus popisuje řešení stránkování jednotlivými grafickými rozhraními pro nastavení dílčích generátorů, vlastností herní postavy a globálních vlastností generátorů. V případě, že se má načíst následující stránka, zkontroluje se, zda není již načtena poslední stránka – pokud ano, je spuštěno generování herní mapy – v opačném případě je aktuální stránka deaktivována a následující stránka nastavení je naopak aktivována. V případě, že má být načtena předchozí stránka, je chování algoritmu obdobné. Je-li první stránka již načtena, je uživatel přesměrován na úvodní stránku herního dema – v opačném případě je aktuální stránka deaktivována a předcházející stránka aktivována.

```
1  function ChangePage(nextPage)
2      if nextPage then
3          if ActualPage == (Pages.Count - 1) then
4              StartMapGeneration()
5          else
6              Pages[ActualPage++].SetActive(false)
7              Pages[ActualPage].SetActive(true)
8      else
9          if ActualPage == 0 then
10             LoadMainMenu()
11         else
12             Pages[ActualPage–].SetActive(false)
13             Pages[ActualPage].SetActive(true)
```

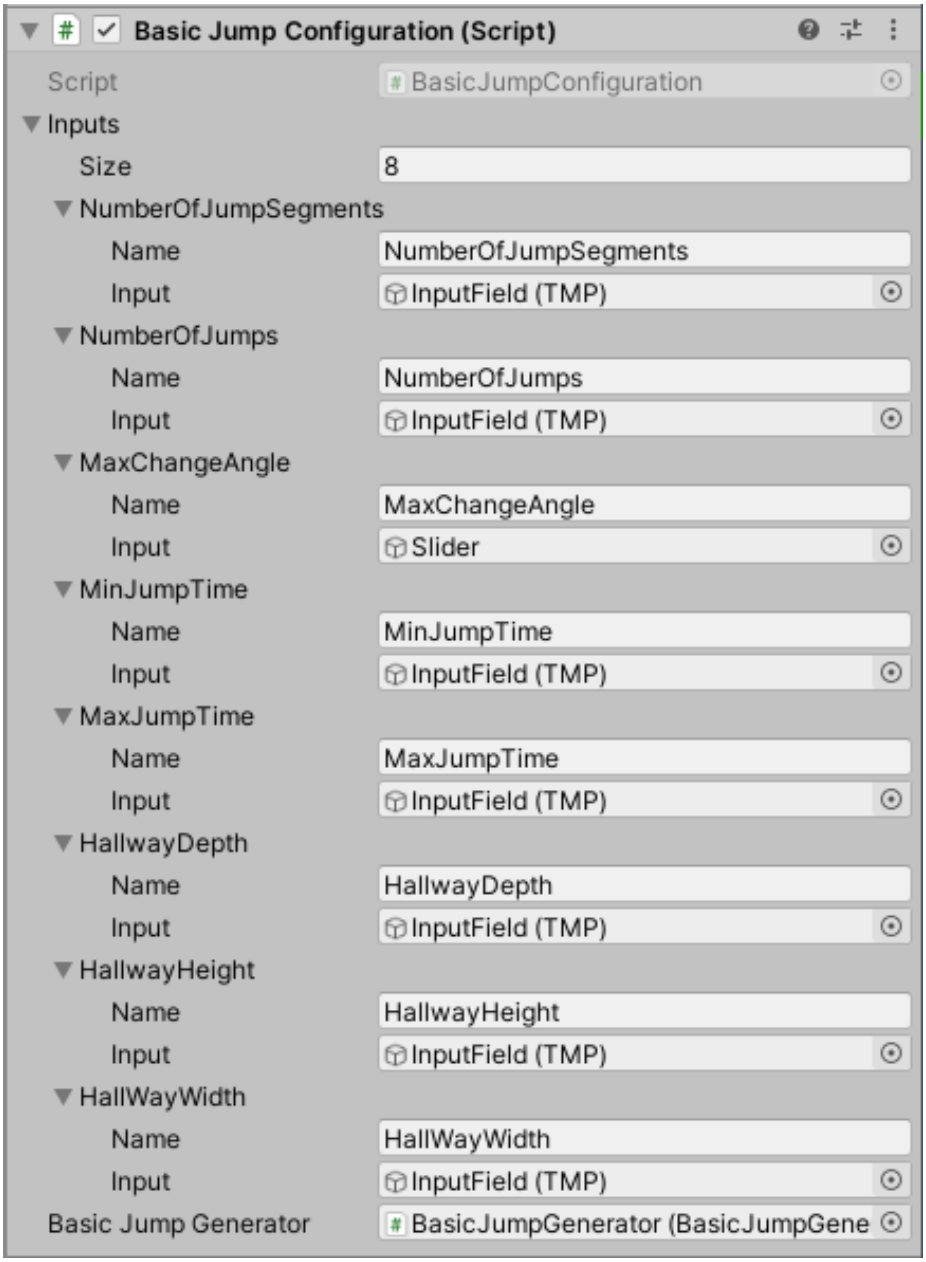
10.2 Ovládání herního dema

Před nastavením herní postavy a jednotlivých generátorů herní mapy je potřeba nejprve zvolit, které schopnosti herní postavy budou při průchodu herní mapou a při samotném generování mapy použity. Pro zvolení těchto schopností je potřeba přejít z úvodní obrazovky na Generátor herní mapy, vybrat požadované schopnosti herní postavy a pokračovat ke generování mapy. Následně se uživatel nachází na stránkovacím rozhraní, kde na jednotlivých stránkách lze nastavit patřičné parametry generování herní mapy – viz. kapitola 10.1. Poté, co uživatel projde všechny stránky nastavení jednotlivých parametrů generování, se vyskytne v herním prostředí, kde lze vidět vygenerovanou mapu. V daném prostředí se lze, při držení pravého tlačítka myši, pohybovat pomoci kláves W, A, S, D, Q a E a pohybem myši. Pro přizpůsobení vlastností volného pohybu prostorem lze kliknout na znak nastavení v pravém horním rohu obrazovky a upravit rychlost a citlivost ovládání. V případě, že je vygenerovaná mapa nevyhovující, lze se pomocí klávesy ESC vrátit zpět k nastavení jednotlivých parametrů generování herní mapy. Pokud je herní mapa vyhovující, lze ji otestovat jako hráč hrající za herní postavu, a to pomocí tlačítka Hrát. Herní postava se ovládá pomocí tlačítek W, A, S a D pro pohyb, mezerníkem pro skok a myší pro rozhlížení se po okolí. Po stisku klávesy ESC lze v Nastavení upravit citlivost myši pro rozhlížení a pomocí tlačítka Ukončit hraní ukončit testování herní mapy a vrátit se do prohlížení herního prostoru.

10.3 Ukládání nastavení prostředí

Herní demo automaticky ukládá a načítá nastavení vlastností herní postavy a všech generátorů herní mapy, díky čemuž není potřeba při znovuspuštění herního dema znovu nastavovat jednotlivé parametry – ty se načtou z konfiguračních souborů.

Automatické ukládání vlastností je řešeno pomocí abstraktní obecné třídy, které je při dědění jako argument předávána třída objektu, jehož vlastnosti jsou požadovány k uložení. Abstraktní třída obsahuje list objektů, které specifikují názvy vlastností herního objektu a pole pro zadávání hodnot těchto vlastností – viz. obrázek 10.1. Dědící třída dále specifikuje název souboru (ve kterém bude konfigurace objektu uložena), samotný herní objekt (jehož vlastnosti mají být uloženy), a v případě potřeby i způsob práce s vlastnostmi herního objektu (např. u herní postavy, kde je potřeba předávat hodnoty vlastností rakety pro schopnost Rocket jump – viz. kapitola 8.2.7 – právě hernímu objektu prezentující onu raketu). Samotná abstraktní třída pak obsahuje mechanismy, které při změně hodnoty pole ve zmíněném listu objektů automaticky aktualizují hodnoty herního objektu, ke kterému konfigurátor náleží, a uloží je do konfiguračního souboru. Stejně tak abstraktní třída při spuštění herního dema automaticky načítá data z konfiguračního souboru a aktualizuje údaje jak v přiděleném objektu, tak v přidělených polích pro zadávání hodnot vlastností objektu.



Obrázek 10.1: Vyobrazení rozhraní komponenty konfigurátoru dílčího generátoru základního skoku herní postavy. Rozhraní obsahuje list propojů vlastností daného generátoru s poli pro zadávání hodnot zmíněných vlastností. Specifikování propoje je učiněno tak, že je určen herní objekt obsahující hodnotu nastavení a specifikován název vlastnosti generátoru, se kterým má být herní objekt propojen. Skrze rozhraní se konfigurátoru předává i samotný dílčí generátor – aby mu mohly být nastaveny patřičné hodnoty.

Kapitola 11

Příklady generování herní mapy

Tato kapitola se zaměřuje na výsledek této práce a demonstruje názorné příklady generování herní mapy, možnosti výsledek generování modifikovat a slabosti generátoru, nebo-li chyby, na které lze narazit.

Obrázek 11.1 znázorňuje plné využití procedurálního generátoru, kdy se mapa skládá z několika částí. Pokud by uživatel vznesl požadavek na větší mapu, lze počet částí herní mapy ještě více navýšit větším poměrem tzv. prázdných plošin, nebo lze jednotlivé části herní mapy prodloužit navýšením počtu segmentů, ze kterých se skládají, či navýšením počtu překážek/skoků na jeden segment. Velikost herní mapy lze ovlivnit i počtem využitých schopností herní postavy – čím více schopností je využito, tím větší herní mapa je.

Obrázek 11.2 vyobrazuje generování části herní mapy pro základní skok herní postavy. Na základě parametrů nastavených uživatelem může nastat situace, kdy nebude možné, aby se hráč se svou herní postavou dostal ze začátku části herní mapy na konec, neboť herní mapa bude vygenerována chybně. Tato situace může nastat při nevhodně nastavených parametrech generování herní mapy, jako např. nastavení dlouhotrvajících skoků a nízkého stropu chodby průchodu, což zapříčiní, že nízký strop chodby znemožní dosáhnutí cílové plošiny při překonávání překážky. Parametry generátorů nejsou nijak omezeny, aby uživatel, popř. herní vývojář, nebyl nijak omezen při využívání procedurálního generátoru – veškerá zodpovědnost nad vhodností užitých parametrů generování spadá na uživatele generátoru.

Obrázek 11.3 znázorňuje generování více částí herní mapy pro základní skok herní postavy. Na obrázku je vyobrazena jedna z chyb generování, se kterou se lze setkat. Jedna z částí herní mapy se v jeden moment neustále otáčí stejným směrem bez změny výšky, což způsobuje, že daný průchod zasahuje sám do sebe a znemožňuje hráči průchod. Pro nápravu této chyby je potřeba změnit parametry generování – primárně seed a inkrement generování.

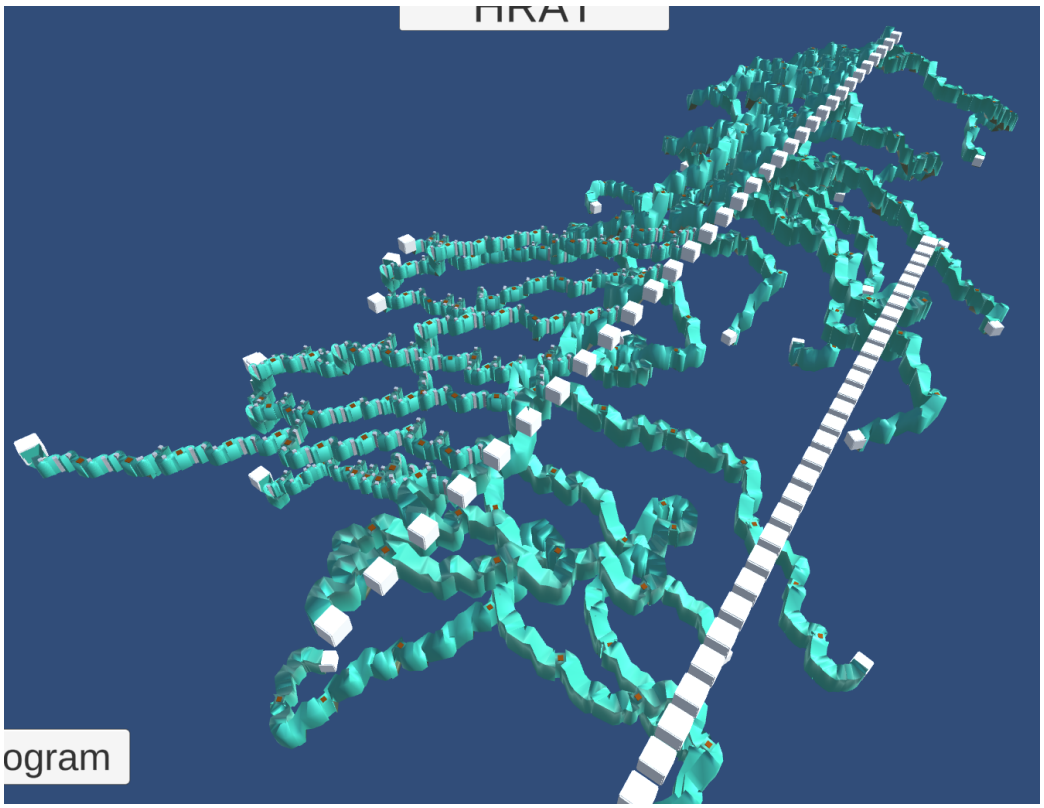
Na obrázku 11.4 je vyobrazeno generování části herní mapy pro schopnost herní postavy Air strafe. Generování herní mapy tohoto typu mělo velkou pravděpodobnost, že by herní mapa zasahovala sama do sebe – proto bylo implementováno omezení, které zapříčiňuje, že se herní mapa nemůže stočit zpět, ale může pouze směřovat vpřed.

Obrázek 11.5 znázorňuje kombinaci různých typů částí herní mapy při generování herní mapy jako celku. Obrázek vyobrazuje, jak jednotlivé části herní mapy navzájem mezi sebou kolidují, nicméně tento problém je řešen zpřístupněním pouze té části herní mapy, na které se právě hráč se svou herní postavou právě nachází.

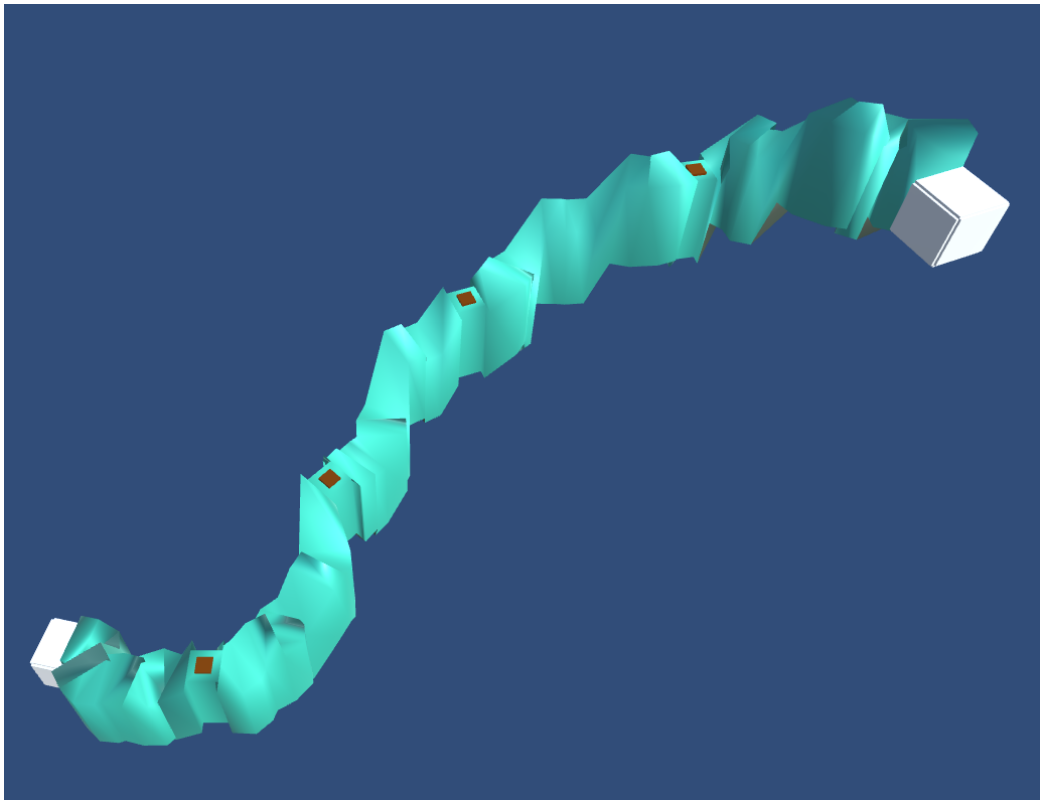
Jak již bylo zmíněno, procedurální generátor obnáší omezení, kdy uživatel generátoru přebírá plnou zodpovědnost nad zadanými parametry generování. Uživatel se tak musí vyhnout nevhodnému nastavení jako např. nastavení inkrementu generátoru na 0, kdy vý-

sledek generátoru náhodných hodnot bude vždy stejný (viz. obrázek 11.6). Další rizikový parametr je nastavení úhlu, o kolik maximálně se může změnit směr průchodu při překonávání překážky – tento úhel je možné nastavit tak, že dopadové plošiny jsou generovány mimo možnosti herní postavy (viz. obrázek 11.7). Dále je možnost nastavit větší délku skoku a zároveň nízký strop průchodu, což zapříčiní, že strop přilehne ke kraji plošiny, kde začíná překážka a znemožní jakýkoliv další průchod (viz. obrázek 11.8).

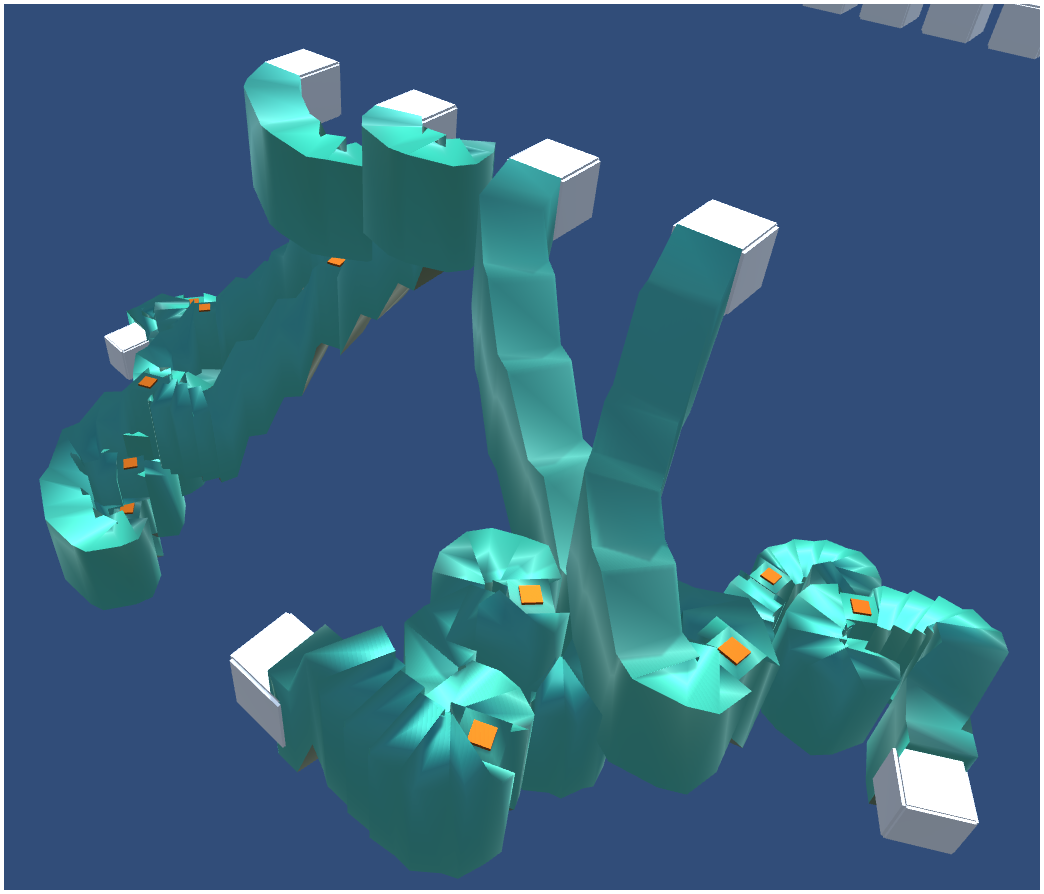
Možností, jak narazit na omezení procedurálního generátoru je více, nicméně všechny omezení lze řešit změnou parametrů generování – ať už změnou vlastností generování (délky skoku, rozměrů mapy atd.), nebo samotného seedu a inkrementu generátoru, a vygenerovat úplně novou herní mapu.



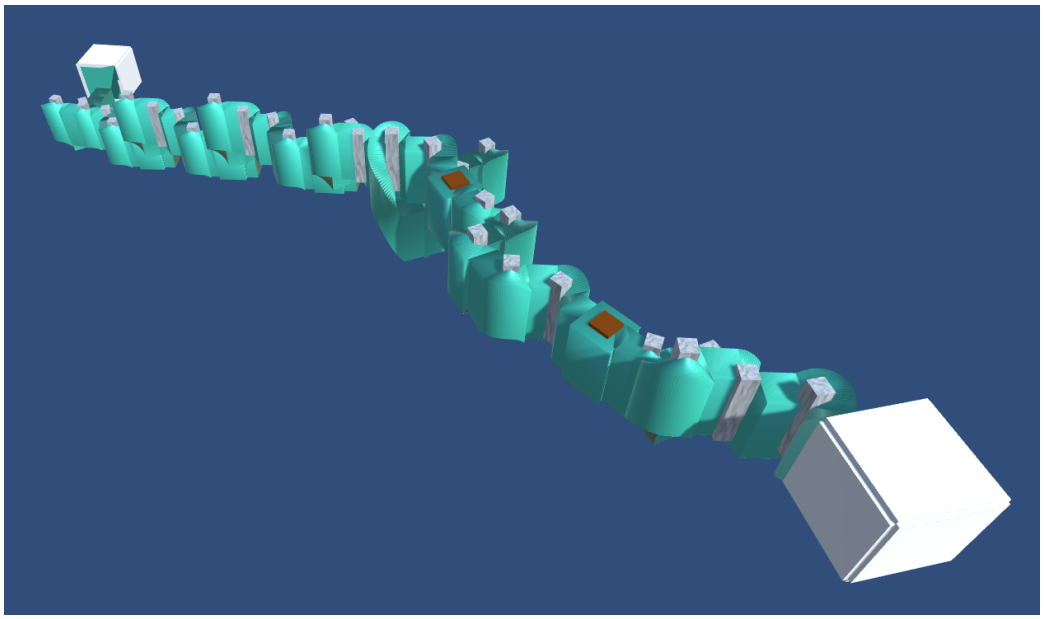
Obrázek 11.1: Rozsáhlé generování herní mapy. Herní mapa obsahuje vyšší poměr tzv. prázdných plošin a využívá základní skok herní postavy a schopnost Air strafe.



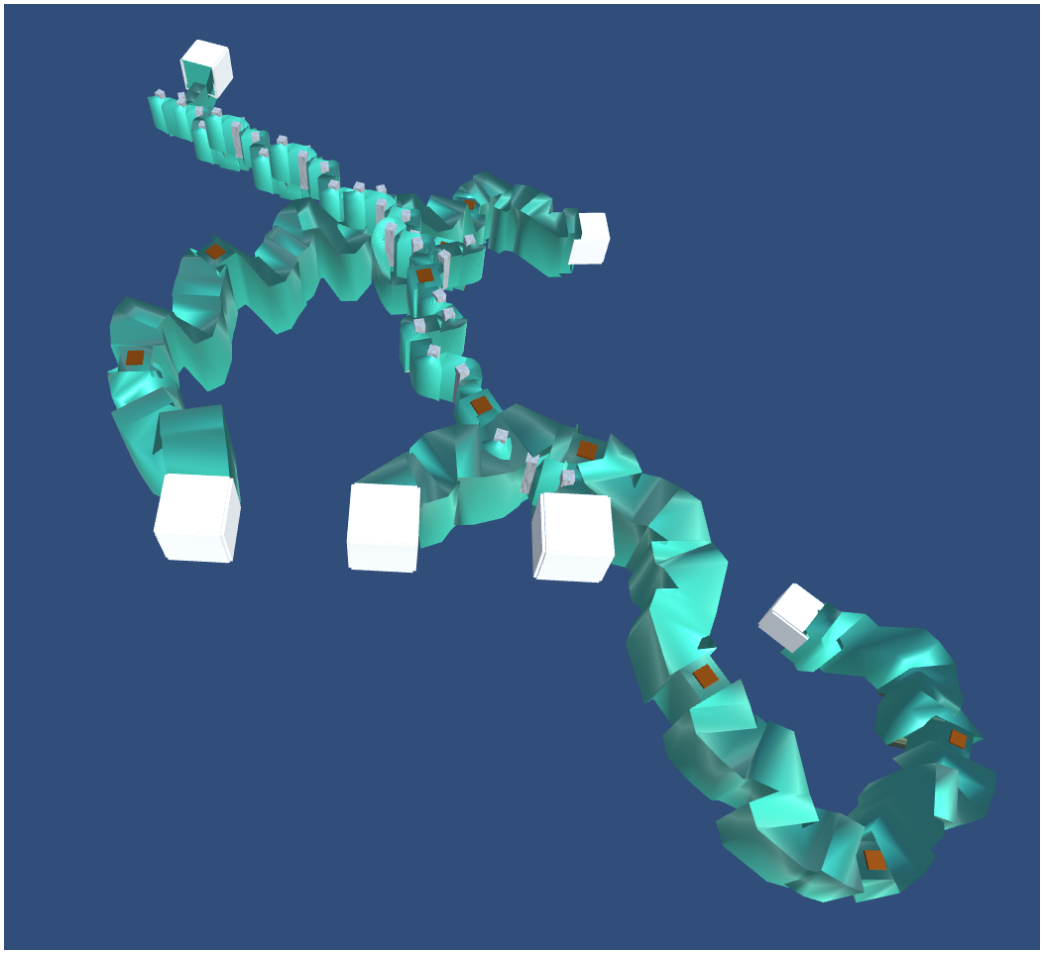
Obrázek 11.2: Výsledek generování základního skoku o 5 segmentech po 7 skocích na jeden segment. V případě, že by tvar mapy byl nevyhovující, stačí pozměnit seed nebo inkrement generátorů.



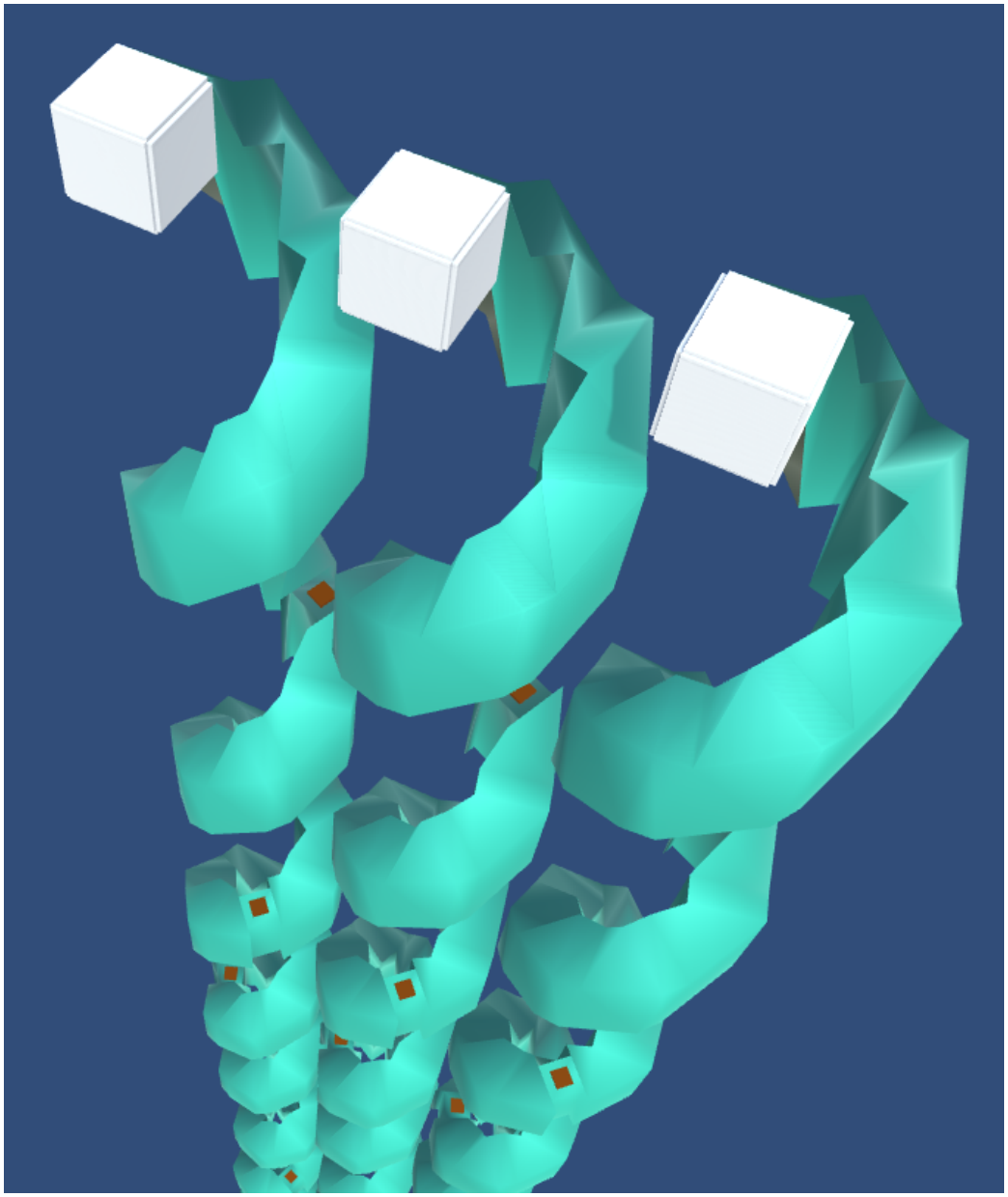
Obrázek 11.3: Výsledek generování herní mapy založené čistě na základním skoku herní postavy. Oproti obrázku 11.2 byla navýšena pravděpodobnost prázdných plošin – tedy pravděpodobnost, že mapa bude delší s více překážkami.



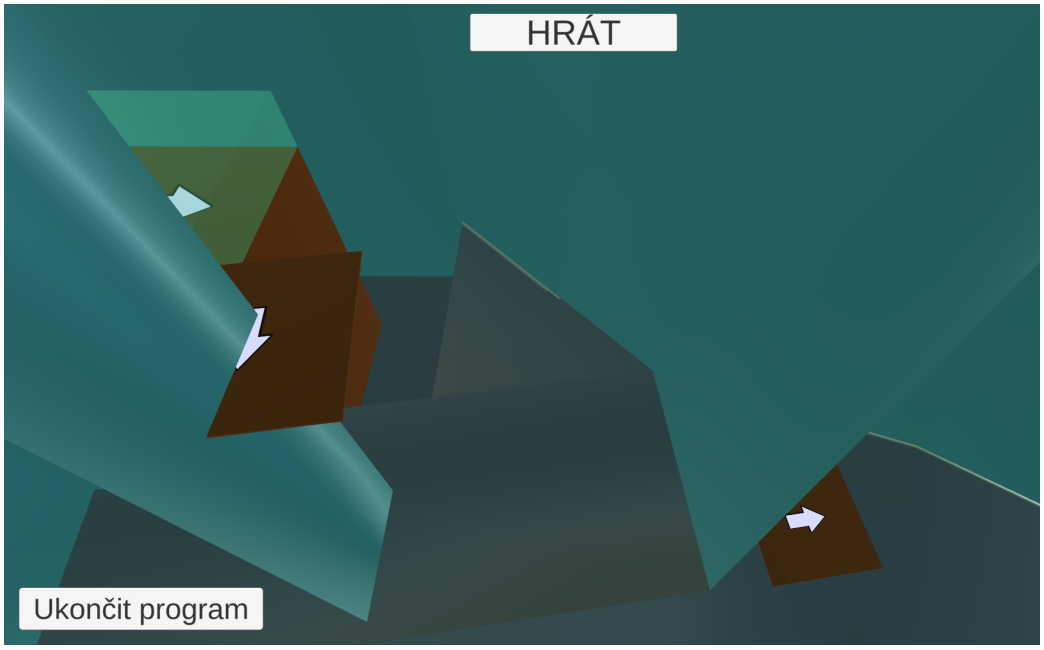
Obrázek 11.4: Příklad generování herní mapy pro schopnost herní postavy Air strafe. Tato část herní mapy se skládá ze 7 segmentů o 5 překážkách na každý segment. Na obrázku lze pozorovat, že generování herní mapy pro schopnost Air strafe neumožňuje generovat směr průchodu, který by se odkláněl od počátečního směru o více jak 80 stupňů.



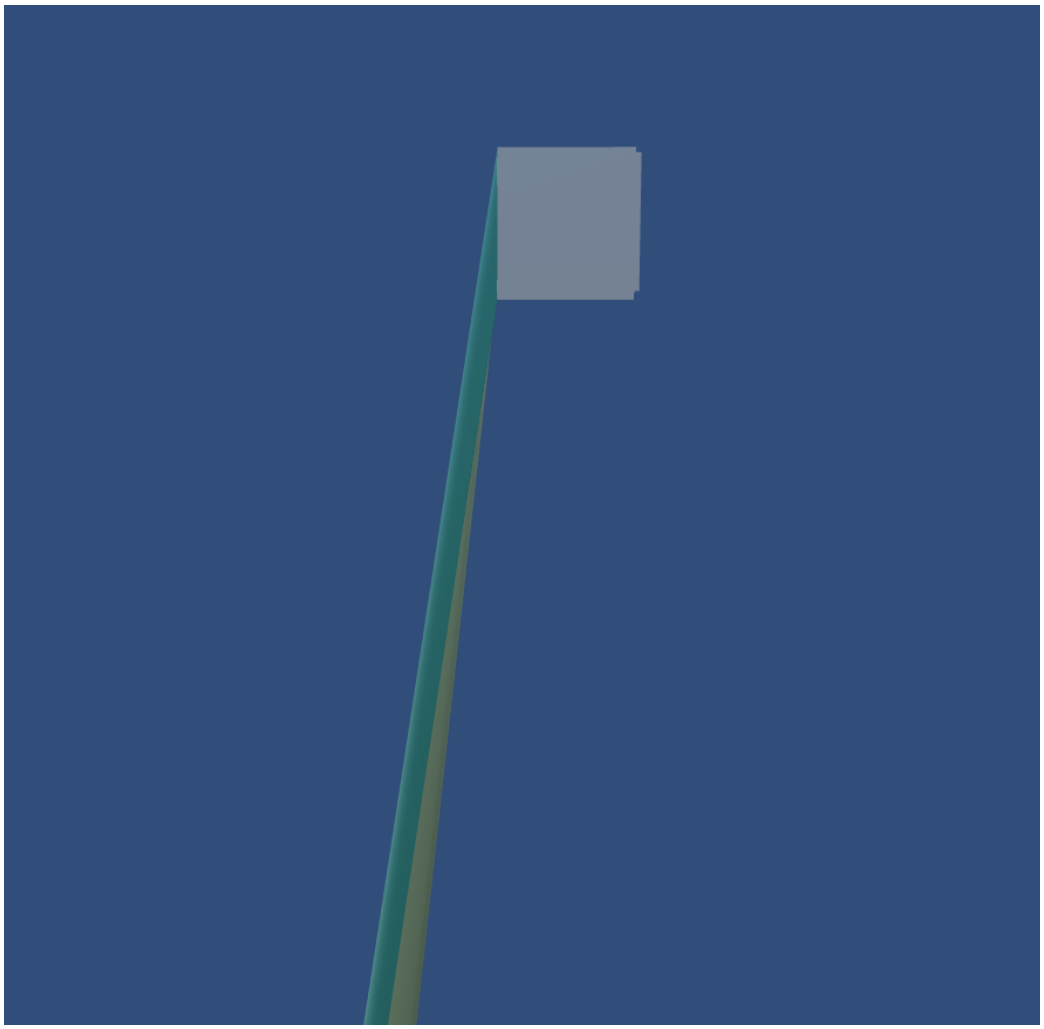
Obrázek 11.5: Názorná ukázka generování herní mapy základního skoku a schopnosti herní postavy Air strafe zároveň. Jednotlivé průchody jsou navzájem propletené a kolidují spolu, nicméně tento problém je řešen zakrýváním všech průchodů, na kterých se hráč právě nenachází – viz. kapitola 9.3.3.



Obrázek 11.6: Herní mapa vygenerovaná při nastavení inkrementu generátorů na hodnotu 0.



Obrázek 11.7: Část herní mapy při nastavení příliš velkého úhlu zatočení při skoku. Dopadové plošiny se pak mohou naskytovat v generované zdi, nebo dokonce za rohem, kam se hráč se svou herní postavou nemá jak dostat.



Obrázek 11.8: Část herní mapy při nastavení příliš dlouho trvajícího skoku s nízkým stropem průchodu.

Kapitola 12

Závěr

Cílem práce bylo navrhnout a implementovat procedurální generátor herní mapy, generující mapu na základě schopností herní postavy. Součástí zadání byl i vývoj herního dema, skrze které je možné procedurální generování mapy prezentovat, a vývoj herní postavy se schopnostmi, na základě kterých je herní mapa generována.

V první fázi vývoje byla vyvinuta herní postava a sedm speciálních schopností určených ke zdolávání překážek herní mapy. Následně byl navrhnut a implementován systém procedurálního generátoru, který je rozdělen na páteřní generátor, jenž se stará o celé generování herní mapy, a množinu dílčích generátorů, které se zaměřují na generování jednotlivých částí herní mapy. V poslední fázi bylo vyvinuto grafické uživatelské rozhraní, které umožňuje demonstraci funkcionality procedurálního generátoru herní mapy – tzn. umožňuje uživateli nastavit parametry jednotlivých generátorů herní mapy a ovlivňovat tak výsledek generování herní mapy, kterou si lze po jejím vygenerování prohlédnout, či dokonce otestovat jako hráč hrající za herní postavu.

Vzhledem k množství schopností herní postavy a jejich komplexnosti nebylo časově uskutečnitelné navrhnout a implementovat dílčí generátory, generující herní mapu, pro všechny schopnosti herní postavy. Proto je výsledný procedurální generátor schopen generovat herní mapu pouze pro základní skok herní postavy a schopnost herní postavy Air strafe. Každopádně aby bylo možné demonstrovat všechny implementované schopnosti herní postavy, v grafickém uživatelském rozhraní je přidána možnost přejít do testovací mapy, kde jsou přístupné všechny schopnosti herní postavy.

Procedurální generátor herní mapy je dále rozšiřitelný, což je umožněno jeho rozdělením na páteřní generátor a dílčí generátory. Pro přidání nového generátoru herní mapy stačí, aby daný generátor dědil abstraktní třídu dílčích generátorů a byl přidán do kontejneru dílčích generátorů pod specifickou schopnost herní postavy, či do skupiny určené pro základní skok herní postavy. Pro případné rozšíření zásoby schopností herní postavy je potřeba modifikovat skript herní postavy, neboť schopnosti herní postavy nejsou implementovány externě – tak jak je tomu u dílčích generátorů herní mapy. Další možností, jak by bylo možné tuto práci rozšířit, je zkoumat generování mapy a hledat možnosti, jak ještě více omezit pravděpodobnost, že by při generování herní mapy došlo ke kolizi a mapa by musela být vygenerována znovu, ale s jinými vstupními parametry.

Literatura

[1] ZAMOJC, Ian. Introduction to Unity3D. *Envatotuts+* [online]. 2012 [cit. 2020-07-28]. URL https://code.tutsplus.com/tutorials/introduction-to-unity3d--mobile-10752

[2] KUMA. Co je to herní engine. *České mody* [online]. 2016 [cit. 2020-07-28]. URL https://www.ceskemody.cz/clanky.php?clanek=56

[3] Dungeon. *IT SLOVNÍK* [online]. [cit. 2020-07-28]. URL https://it-slovnik.cz/pojem/dungeon

[4] TOGELIUS, Julian, Emil KASTBJERG, David SCHEDL a Georgios N. YANNAKAKIS. *What is Procedural Content Generation? Mario on the borderline* [online]. Copenhagen, Denmark, 2011 [cit. 2020-07-28]. URL https://course.ccs.neu.edu/cs5150f13/readings/togelius_what.pdf

[5] Perlin noise. *Khan Academy* [online]. [cit. 2020-07-28]. URL https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise

[6] Unity User Manual (2019.4 LTS). *Unity Technologies* [online]. 2020 [cit. 2020-07-28]. URL https://docs.unity3d.com/Manual/index.html

[7] L-System geometry node. *Houdini* [online]. [cit. 2020-07-28]. URL https://www.sidefx.com/docs/houdini/nodes/sop/lsystem.html

[8] TIŠNOVSKÝ, Pavel. L-systémy: přírodní objekty i umělé artefakty. *Root.cz* [online]. 2006 [cit. 2020-07-28]. URL https://www.root.cz/clanky/l-systemy-prirodni-objekty-i-umele-artefakty/#k01

[9] Generování (pseudo)náhodných čísel. *ITnetwork.cz* [online]. [cit. 2020-07-28]. URL https://www.itnetwork.cz/navrh/algoritmy/algoritmy-matematicke/algoritmus-generovani-pseudo-nahodnych-cisel

[10] LAGUE, Sebastian. Procedural Landmass Generation (E01: Introduction). *YouTube* [online]. 2016 [cit. 2020-07-28]. URL https://youtu.be/wbpMiKiSKm8

[11] The Elder Scrolls V: Skyrim. *Wikipedia* [online]. [cit. 2020-07-28]. URL https://cs.wikipedia.org/wiki/The_Elder_Scrolls_V:_Skyrim

[12] SUTTON, Matt. How much of the content in The Elder Scrolls V: Skyrim was procedurally generated? *Quora* [online]. 2011 [cit. 2020-07-28].
URL https://www.quora.com/How-much-of-the-content-in-The-Elder-Scrolls-V-Skyrim-was-procedurally-generated

[13] Descenders. *Wikipedia* [online]. [cit. 2020-07-28].
URL https://en.wikipedia.org/wiki/Descenders

[14] Shoot 'em up. *Wikipedia* [online]. [cit. 2020-07-29].
URL https://en.wikipedia.org/wiki/Shoot_%27em_up#Bullet_hell

[15] Roguelike. *Wikipedia* [online]. [cit. 2020-07-29].
URL https://en.wikipedia.org/wiki/Roguelike

[16] Enter the Gungeon. *Wikipedia* [online]. [cit. 2020-07-28].
URL https://en.wikipedia.org/wiki/Enter_the_Gungeon

[17] No Man's Sky. *Wikipedia* [online]. [cit. 2020-07-28].
URL https://en.wikipedia.org/wiki/No_Man%27s_Sky

Příloha A

Obsah přiloženého paměťového média

Přiložené CD obsahuje:

- tento dokument v PDF formátu
- zdrojový kód tohoto dokumentu v LaTeX
- složku se zdrojovým kódem herního dema
- složku se spustitelnou aplikací herního dema, určené pro Windows 64bit
- demonstrační video