UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2018 Dominik Veselý

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Implementace arkádové hry

Dominik Veselý

Bakalářská práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Dominik Veselý**
Osobní číslo:	**I14197**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Implementace arkádové hry**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce je ohraničena tématy herního průmyslu, tvorby aplikací pro stolní počítače a metodami distribuce hotových řešených prostřednictvím oficiálních i alternativních tržních kanálů.

V práci je třeba vymezit prostředí jednotlivých operačních systémů z pohledu podpory Unity3D aplikací, a popsat distribuční kanály pro jednotlivá prostředí, přičemž důraz je kladen na časové, legislativní a finanční aspekty. Dále bude v práci zapracován rozbor postupů při vývoji počítačových her.

Realizovanou případovou studií bude aplikace, která bude odpovídat současným požadavkům na moderní hru. V rámci implementace bude využito 2D nebo 3D scény, práce s modely či texturami a lineárním příběhem. Případová studie musí být vymezena prostřednictvím korektních dokumentů (např. storyline, game design document, konceptuální schémata).

Vlastní implementace hry bude v čistě 3D prostředí. Student využije assety, které sám vytvořil, nebo může využít assetů s otevřenou licencí, případně dokonce assetů vytvořených jako školních děl. Student má však povinnost u každého nepůvodního assetu uvést zdroj a odůvodnit jeho použití. Samotná hra bude rozšířením koncepce ?bomber man?, velmi populární v minulém stolení. Koncepce bude přetvořena tak, že při zachování maxima původních vlastností hry, bude umožňovat pohled první osoby (hráče) a tím změní herní styl z klasické plošinovky na 3D arkádu.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

DILLE, Flint. The ultimate guide to video game writing and design. New York: Watson-Guptill Publications, 2007. ISBN 158065066X.
PECINOVSKÝ, Rudolf. OOP: Naučte se myslet a programovat objektově. Brno: Computer Press, a.s., 2010. ISBN 978-80-251-2126-9.
KNUTH, D. E.: Umění programování - Základní algoritmy, Brno, Computer Press 2008, ISBN: 978-80-251-2025-5.
WRÓBLEWSKI, Piotr. Algoritmy: datové struktury a programovací techniky. Vyd. 1. Překlad Marek Michalek, Bogdan Kiszka. Brno: Computer Press, 2004, 351 s. ISBN 80-251-0343-9.
KEOGH, Jim; DAVIDSON, Ken. Datové struktury bez předchozích znalostí : průvodce pro samouky. Vyd 1. Brno : Computer Press, 2006. 223 s. ISBN 80-251-0689-6.

Vedoucí bakalářské práce: **Ing. Josef Brožek**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2016**
Termín odevzdání bakalářské práce: **12. května 2017**

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Zdeněk Šilar, Ph.D.
pověřený vedením katedry

V Pardubicích dne 31. března 2017

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 11. 05. 2018

Dominik Veselý

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce Ing. Josefu Brožkovi, za ochotu, trpělivost a užitečné rady při práci na mé bakalářské práci. Dále bych chtěl poděkovat své rodině a přátelům, kteří mě stále podporují během mých studií.

ANOTACE

Bakalářská práce se zabývá tvorbou počítačové hry Bomberman pro více hráčů po síti LAN. V úvodu práce je představena samotná hra a její historie. Následně je popsán herní engine Unity, ve kterém byla hra vytvořena. Nakonec je podrobně vysvětlena implementace hlavních prvků ve hře.

KLÍČOVÁ SLOVA

3D počítačová hra, hra po síti LAN, arkádová hra, hra v Unity, Bomberman

TITLE

The Arcade Implementation

ANNOTATION

The bachelor´s thesis is about creating of the computer game called Bomberman for more players through LAN network. At the beginning of my thesis is introduced the game and its history. Than is described the game engine called Unity, in which the game was created. Finally, there is mentioned the implementation of the main game elements in detail.

KEYWORDS

3D computer game, game through LAN network, arcade game, game in Unity, Bomberman

OBSAH

Úvod 14

1 Hra Bomberman 15

1.1 Historie 15

Dyna Blaster 16

1.2 Představení hry 17

2 Unity 19

2.1 Unity Editor 20

2.2 Další důležité prvky v Unity 20

Komponenta 21

Kamera 21

Collider 21

MonoBehaviour 21

Coroutine 22

Layers 23

3 Unity Networking 24

3.1 Transport Layer API 24

3.2 HLAPI 24

NetworkBehaviour 24

NetworkIdentity 25

Spawn 25

SyncVar 25

Hook metoda 26

Command metoda 26

RPC metoda 26

Cílená RPC metoda 26

Zprávy po síti 26

4 Implementace 27
4.1 Lobby 27
Vytvoření lobby 27
Připojení do lobby 27
Hráč v lobby (LobbyPlayer) 28
Tlačítko pro změnu stavu hráče 28
Tlačítko pro spuštění hry 29
Tlačítko pro vyhození hráče z lobby 29
4.2 Krabice 30
4.3 Pickup 32
Uzdravení 33
Zmražení 33
Nesmrtelnost 34
Zvýšení rychlosti 34
Zvýšení dosahu bomby 35
Spouštěcí bomba 35
4.4 Bomba 35
Položení bomby 35
Výbuch bomby 36
Základní bomba 39
Spouštěcí bomba 39
4.5 Hráč ve hře (GamePlayer) 40
Základní vlastnosti 40
Tabulka se statistikami hráčů 40
Pohyb hráče 41
Animace hráče 41
Zvuky hráče 42

Místo položení bomby 43
Inventář 44
4.6 Průběh hry 44
Inicializace hry na straně serveru 44
Inicializace hry na straně klienta 44
Před spuštěním hry 45
Před spuštěním nového kola 45
Nové kolo 46
Divácký režim 46
Konec kola 46
Konec hry 47
5 Shrnutí 48
5.1 Porovnání s titulem Bomberman 48
Společné vlastnosti 48
Rozdílné vlastnosti 48
5.2 Možné zlepšení 49
Závěr 50
Použitá literatura 51

SEZNAM ILUSTRACÍ

Obrázek 1: Bomberman (1983) 15
Obrázek 2: Super Bomberman R (2017) 15
Obrázek 3: Dyna Blaster – Hlavní menu 16
Obrázek 4: Dyna Blaster – Úvodní scéna z kampaně 16
Obrázek 5: Dyna Blaster – Úroveň z kampaně 17
Obrázek 6: Dyna Blaster – Mód pro více hráčů 17
Obrázek 7: Ukázka hlášky, která se zobrazí hráči při napojení se na zaplněný server 27
Obrázek 8: Ukázka Lobby na straně serveru 28
Obrázek 9: Ukázka Lobby na straně klienta 28
Obrázek 10: Ukázka metody, která zpracovává zprávu, když je klient vyhozen ze serveru. 29
Obrázek 11: Krabice, z které padá pickup. 30
Obrázek 12: Kolize krabice s výbuchem bomby 31
Obrázek 13: Zdrojový kód Krabice 31
Obrázek 14: Pickup s částicovým efektem 32
Obrázek 15: Ukázka částicových efektů 32
Obrázek 16: Ukázka časových pickupů v GUI 32
Obrázek 17: Ukázka pickupu zmražení 33
Obrázek 18: Ukázka barvy hráče po aplikování pickupu zmražení 33
Obrázek 19: Ukázka pickupu pro nesmrtelnost 34
Obrázek 20: Ukázka barvy hráče po aplikování pickupu nesmrtelnosti 34
Obrázek 21: Ukázkový kód metody pro položení bomby 36
Obrázek 22: Ukázka metody zjišťující náraz mezí zdí a výbuchem bomby 37
Obrázek 23: Ukázka detekce kolizí výbuchu bomby s ostatními objekty 38
Obrázek 24: Ukázka základní bomby 39
Obrázek 25: Ukázka spouštěcí bomby 39
Obrázek 26: Ukázka tabulky se statistikami hráčů ve hře 41
Obrázek 27: Ukázka postavy hráče v nečinnosti 42
Obrázek 28: Ukázka postavy hráče ve druhé fázi nečinnosti 42
Obrázek 29: Ukázka postavy hráče při úmrtí 42
Obrázek 30: Ukázka zvýrazněného místa pro položení bomby 43
Obrázek 31: Ukázka změny barvy crosshairu 43
Obrázek 32: Ukázka metody pro zrození všech hráčů na serveru 45

Obrázek 33: Ukázka odpočtu před zahájením nového kola 46
Obrázek 34: Ukázka tabulky se shrnutím na konci hry 47

SEZNAM ZKRATEK

2D	dvojdimenzionální
3D	trojdimenzionální
GUI	Graphical User Interface
LAN	Local Area Network
API	Application Programming Interface
HLAPI	High Level API
FPS	First-person shooter

ÚVOD

Cílem mé bakalářské práce je vytvořit 3D hru inspirovanou titulem Bomberman. Hra musí být pro více hráčů a komunikovat po síti LAN. Tuto hru jsem se rozhodl vytvořit v herním enginu Unity. S Unity jsem se seznámil na vysoké škole, díky volitelnému předmětu, který jsem během svého studia úspěšně absolvoval.

Už od střední školy je programování mým velkým koníčkem. Vždy mě fascinovalo a udivovalo, co pouhým napsáním kódu může vzniknout za výsledek. Ačkoliv jsem se během střední a vysoké školy věnoval spíše webovým aplikacím, vždy mě lákalo vytvořit si vlastní hru, ale nikdy jsem se k tomu neodvážil. Proto jsem se rozhodl pro toto téma, abych si mohl vyzkoušet vytvořit svou vlastní počítačovou hru.

Jelikož mě baví spíše programovací část, spolupracoval jsem se svým kolegou Janem Houžvičkou, který do hry vytvořil vizuální část, ke které patří převážně modely a animace objektů. Tím se celá hra dle mého názoru posunula o několik úrovní výše.

V první kapitole bude představen titul hry Bomberman a její historie. Následně bude představena samotná hra této bakalářská práce a porovnání rozdílů s titulem Bomberman.

Druhá kapitola uvede čtenáře do historie herního enginu Unity a vysvětlí jeho důležité části. Účelem této kapitoly je vysvětlit základní principy tohoto enginu a také čtenáři představit jeho důležité vlastnosti, jelikož budou v dalších kapitolách často zmiňovány.

Třetí kapitola se zabývá síťovou komunikací a vysvětluje dva způsoby síťové komunikace, s kterými lze v Unity pracovat. Cílem kapitoly je vysvětlit výhody a nevýhody těchto dvou způsobů a podrobněji vysvětlit způsob, kterým je tato hra vytvořena.

Čtvrtá kapitola je zaměřena na samotnou implementací jednotlivých části hry. Cílem této kapitoly je čtenáři co nejlépe vysvětlit princip fungování jednotlivých částí hry. Buď rozebíráním konkrétních části kódu nebo vysvětlením složitějších části, které nejsou vysvětleny do hloubky, aby byly pro čtenáře lépe pochopitelné.

Poslední kapitola shrnuje samotnou hru, její nedostatky a možné nápady na zlepšení.

1 HRA BOMBERMAN

Bomberman je strategická hra pro jednoho nebo více hráčů. Cílem hry je zneškodnit všechny nepřátelé a přežít jako poslední. Hlavní zbraní pro eliminaci nepřátel je bomba. Prostředí hry je tvořeno z čtvercových bloků, které jsou rozmístěny šachovnicově a můžou fungovat jako skrýš pro vyhnutí se výbuchu bomby, jelikož se o tuto zeď výbuch zastaví. Na ostatní náhodně vybraná volná místa jsou vygenerovány bloky, které jsou zničitelné bombou. Po zničení bloku je šance, že se na místě objeví bonusový prvek, který dá hráči po jeho sebrání výhodu např. větší dosah bomby, možnost pokládat více bomb za sebou nebo další jiné bonusy.

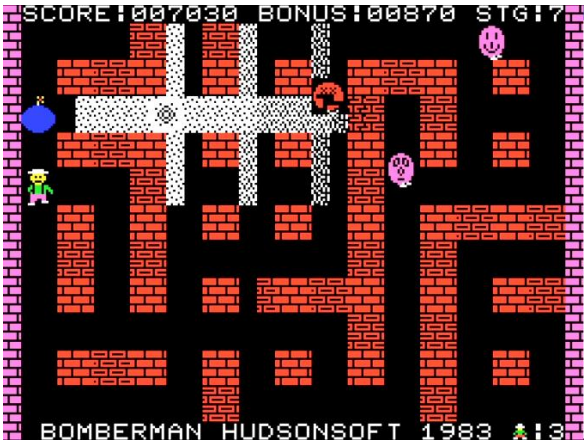
Cíl hry se liší podle módu. Pokud se hraje kampaň, tak hráč hraje sám a má za úkol zničit všechny nepřátelé a poté nalézt vchod do další úrovně. Ve druhém módu, pro více hráčů, je cílem hry eliminovat všechny ostatní protihráče a přežít jako poslední. Pro ztížení jsou v prostředí generovány nepřátelé, které jsou řízeni počítačem.

Zdroj [1].

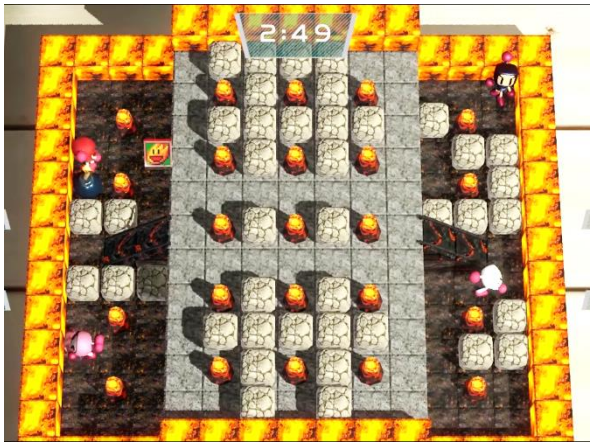
1.1 Historie

Existuje kolem 20 titulů hry Bomberman a přes 70 titulů, které jsou založeny na principu této hry. Nejstarší a také originální verze byla vydána v roce 1983 japonskou firmou *Hudson Soft* (Obrázek 1). Ovšem popularitu si získal až o dva roky novější titul se stejným názvem vydaný na platformě *Nintendo Entertainment System*. Dále se u novějších verzí měnila převážně grafika a úrovně v kampani, ale hlavní logika hry zůstala stejná až doposud. Nejnovější verze hry byla vydána v roce 2017 pod názvem *Super Bomberman R*, který obsahuje prvky 3D (Obrázek 2).

Zdroje [1], [2].



Obrázek 1: Bomberman (1983)



Obrázek 2: Super Bomberman R (2017)

Dyna Blaster

V Evropě je titul hry Bomberman znám pod názvem Dyna Blaster, jelikož byl pod tímto názvem vydáván. Pravděpodobně nejpopulárnější je verze z roku 1992 (Obrázek 3). Právě touto hrou je také hra nejvíce inspirována. Hra Dyna Blaster obsahuje 2 mody:

- kampaň,
- hra pro více hráčů.

Obrázek 3: Dyna Blaster – Hlavní menu

Obrázek 4: Dyna Blaster – Úvodní scéna z kampaně

Kampaň začíná unesením dívky, kterou má za úkol hráč zachránit (Obrázek 4). Hráč tak musí projít všech osm úrovní, které kampaň obsahuje (Obrázek 5). Každá úroveň je rozdělena do dalších podúrovní. V každé podúrovní má hráč za úkol zabít všechny nepřátele na mapě a následně nalézt portál do další podúrovně. V poslední této podúrovni čeká na hráče tzv. boss, což je nepřítel, který má vyvinutější umělou inteligenci a různé speciální schopnosti oproti normálním nepřátelům, které hráč ve hře potkává. Cílem hry je přejít všech osm úrovní a zachránit dívku, která byla unesena.

Zdroj [3].

Hru pro více hráčů můžou hrát až čtyři hráči zároveň (Obrázek 6). Hráči jsou však omezeni na jeden počítač a většinou i na jednu klávesnici, což je značně nepohodlné. Tento mód není nutné hrát se všemi čtyřmi hráči. Hráče je možné nahradit počítačem, kdy je pak tento hráč ovládán umělou inteligencí. V tomto módu je cílem hry přežít jako poslední. Hráč, který přežil jako poslední získá bod a po dosažení pěti bodů, vyhrává celou hru a hra je ukončena.

Zdroj [3].

Obrázek 5: Dyna Blaster – Úroveň z kampaně

Obrázek 6: Dyna Blaster – Mód pro více hráčů

1.2 Představení hry

Hra, o které bude tato práce, je inspirována titulem Bomberman a bude používat hlavní principy této hry. Hlavním rozdílem od originálního titulu bude zasazení hráče do 3D prostředí, které hráč uvidí z pohledu první osoby. Dále nebudou v prostředí žádní nepřátelé řízení počítačem. Všechny postavy budou ovládány živými hráči, kteří mezi sebou budou komunikovat po síti LAN.

Dalším rozdílem od původního titulu bude procentuální zdraví hráče, díky kterému bude moci hráč přežít i více zásahů bombou. V původní verzi byl hráč mrtev ihned po prvním zásahu bombou.

Hlavní zbraní v tomto titulu je bomba, která bude v této hře hlavním předmětem pro zabíjení. Základní bombu bude mít každý hráč od začátku hry a bude jí mít neomezený počet, bude

ovšem limitována tzv. cooldownem[1], který bude podrobněji vysvětlen v kapitole věnující se principu fungování bomby.

Dále bude upravena logika tzv. pickupů, což bude ve hře bonusový předmět, který dá hráči po jeho sebrání speciální schopnost. Na rozdíl od originální verze, zde budou dva druhy pickupů. Pomocí jednoho, hráč obdrží speciální schopnost nebo bude naopak protihráčům aplikována negativní schopnost. Druhým typem pickupu bude získání nové bomby do inventáře, která bude mít odlišné vlastnosti od základní bomby. Detailní probrání jednotlivých pickupů bude v samostatné kapitole o pickupech.

Cílem v této hře nebude přežít jako poslední, ale zabít co nejvíce hráčů. Hra bude rozdělena na kola a jednotlivé kolo skončí buď smrtí všech protihráčů nebo po skončení časového limitu. Při začátku nového kola budou všichni hráči obživeni a budou se moci opět pokusit zabít své protihráče. Skóre bude sčítáno za všechna kola a vítězem celé hry se stane hráč s nejvyšším dosaženým skóre za všechna tato kola.

[1] Cooldown je doba, po kterou nelze bombu použít. Viz kapitola Bomba.

2 UNITY

Unity je multiplatformní herní engine vyvíjen firmou *Unity Technologies*. Firma byla založena v Dánsku roku 2004 vývojáři Davidem Helgasonem, Nicholasem Francisem a Joachimem Antem. První verze Unity vznikla v roce 2005, která podporovala pouze jedinou platformu. Postupně se Unity vyvinulo, kdy nyní (9. 4. 2018) podporuje 27 různých platforem. Od počátku lze v Unity vyvíjet hry s podporou 2D i 3D grafiky.

Hlavním programovacím jazykem v Unity je programovací jazyk *C#*. Dříve Unity podporovalo také další dva jazyky, kterými byl programovací jazyk *Boo* a skriptovací jazyk *UnityScript* na bázi *Javascriptu*. Podpora programovacího jazyka *Boo* byla ukončena v roce 2015 s vydáním *Unity 5*. *UnityScript* byl podporován až do roku 2017, kdy se vývojáři rozhodli přestat podporovat tento jazyk, jelikož byl jazyk využíván programátory jen zřídka. Dalším důvodem byl příchod nových funkcionalit v *.NET* a *C# 6*, které nebylo možné v *UnityScript* uskutečnit.

Unity je rozděleno do čtyř licenčních verzí: Personal, Plus, Pro a Enterprise. Verze Personal je určena hlavně pro začátečníky a studenty, jelikož je zcela zdarma. Ostatní verze jsou placené a každá z těchto verzí obsahuje různé doplňky. Výhodou poslední verze Enterprise je získání přístupu ke zdrojovým kódům enginu. Cena této verze je ovšem inviduální a záleží na oboustranné domluvě mezí vývojáři a tvůrci Unity. Tuto verzi kupují velké firmy, které na tuto licenci mají rozpočet a potřebují si engine upravit podle svého.

Unity od roku 2017 změnilo svoje verzování. Nyní vždy začíná rokem vydání poté následuje tečka s číslem, které určuje číslo verze v roce, např. 2017.2. Dříve bylo verzování klasické např. 4.2.3., které je známé z většiny jiných softwarových aplikací, které používají verzování. Poslední Unity, které používalo styl starého verzování bylo Unity 5.

Při vývoji se nedoporučuje upgradovat mezi verzemi prvního řadu např. z Unity 4 na Unity 5. Jelikož mezi verzemi prvního řádu bývají radikální změny, které mění funkcionalitu celého enginu a mohlo by se stát, že by se zhroutil celý projekt. Proto je tato bakalářská práce napsaná ve verzi 5.5.6, jelikož začala být vyvíjena v Unity 5 a v té době byla tato verze Unity nejaktuálnější.

Zdroje [4], [5].

2.1 Unity Editor

Součástí Unity je také editor, který velice usnadňuje a urychluje práci při vývoji hry. Editor obsahuje mnoho oken, která si lze přizpůsobit. Nejvíce používanými okny jsou okna Hierarchy Window, Scene View, Inspector Window a Project Browser [6].

Hierarchy Window

V Hierarchy Window okně jsou zobrazeny veškeré objekty, které obsahuje aktuální scéna. Pro lepší přehlednost je možné objekty hierarchicky řadit pod sebe, od toho je také odvozen název tohoto okna.

Zdroj [6].

Scene View

Scene View je okno, které interaktivně nahlíží do scény, která je editována. Díky tomu se lze jednoduše ve scéně pohybovat a přesouvat objekty, které jsou ve scéně. Ve scéně je možné sledovat objekty v jakémkoliv směru a úhlu.

Zdroj [6].

Inspector Window

Okno Inspector Window umožňuje upravovat vlastnosti objektu. Pomocí tohoto okna se také připínají jednotlivé komponenty a skripty na objekt. V okně jdou jednoduše upravovat vlastnosti jednotlivých komponent.

Zdroj [6].

Project Browser

Okno Project Browser zobrazuje celou strukturu projektu, respektive složku *Assets*. Veškeré soubory, které jsou použity ve hře by se měly dávat právě do této složky, aby k nim bylo možné přistupovat přímo z editoru. Mezi tyto soubory patří například scény, modely postav, textury, zvuky atd. Pro lepší přehlednost v projektu se doporučuje jednotlivé soubory uspořádat do složek, které více specifikují skupinu souborů.

Zdroj [6].

2.2 Další důležité prvky v Unity

V této kapitole budou vysvětleny prvky, které jsou v bakalářské práci značně používány.

Komponenta

Objekty v Unity se skládají z mnoha tzv. komponent, kde jednotlivé komponenty objektu dávají určitou funkcionalitu. V Unity existuje mnoho komponent, ty nejdůležitější a nejpoužívanější budou v této práci podrobněji popsány. Na objekt lze komponenty libovolně připínat či odepínat. Pouze jediné dvě komponenty nelze z objektu odepnout a těmi jsou komponenty *Transform* (3D a 2D objekty) nebo *RectTransform* (objekty v GUI). Objekt může mít pouze jednu z těchto komponent. Tyto dvě komponenty se starají o pozici, rotaci a velikost objektu ve scéně a bez nich by objekt nemohl ve scéně existovat.

Zdroj [7].

Kamera

Velice důležitou komponentou v Unity je kamera. Kamera určuje prostor, který se má hráči vykreslit na obrazovku. Lze jí definovat dosah viditelnosti, který určuje vzdálenost, do které se budou objekty ve scéně vykreslovat. Také dokáže vykreslovat pouze určitý výčet objektů a má mnoho dalších funkcí.

Zdroj [7].

Collider

Collider je komponenta, která vypočítává kolize s ostatními objekty. Existuje více druhů colliderů, které se liší svým tvarem. Hlavní collidery jsou ve tvarech krychle neboli *BoxCollider*, koule neboli *SphereCollider* a kapsule neboli *CapsuleCollider*. Každý tvar disponuje jinými parametry, které definují jeho velikost a pozici vůči objektu, ke kterému je komponenta připnuta. Není problém mít objekt ve tvaru koule a kolize počítat s tvarem krychle. Potom může nastat situace, že nastane kolize, i přestože vizuálně žádná kolize mezi objekty nenastala.

Zdroj [7].

MonoBehaviour

MonoBehaviour je základní třída, ze které dědí všechny skripty. Třídu je potřeba připnout na objekt ve scéně a tím lze objektu přiřadit chování. Třída obsahuje mnoho užitečných metod. Mezi hlavní metody patří: *Awake, Start, Update, FixedUpdate a LateUpdate*. Tyto metody jsou volány za určitých okolností.

Metoda *Awake* je zavolána, jakmile je objekt přidán do scény. Metoda *Start* je zavolaná, jakmile je objekt ve scéně poprvé aktivní, většinou je to tedy ihned po metodě *Awake*, ovšem až poté,

jakmile je provedena metoda *Awake* na všech objektech ve scéně. Většinou se tak v metodě *Awake* inicializuje objekt a v metodě *Start*, pak může k objektu přistoupit jiný objekt a díky metodě *Awake* má zajištěnou jeho inicializaci. Další velice důležitou metodou je metoda *Update*, tato metoda je poprvé zavolána po metodě *Start* a následně je volána při každém snímku. V této metodě je dobré odchytávat vstupy od uživatele. Například pokud uživatel stiskne určitou klávesu, objekt na to patřičně zareaguje.

Metody *FixedUpdate* a *LateUpdate* jsou speciální metody, které jsou volány v určitých intervalech stejně jako metoda *Update*. Metoda *FixedUpdate* je volána vždy ve stejném intervalu a používá se pro počítání fyziky. Metoda *LateUpdate* je volána vždy až po metodě *Update* a *FixedUpdate*. Metoda *LateUpdate* se nejčastěji využívá pro přesun kamery, která následuje pohyb hráče. Jelikož se logika pohybu hráče vkládá do metod *Update* a *FixedUpdate,* je zajištěno, že se kamera přesune až po samotném pohybu hráče.

Zdroje [7], [8].

Coroutine

Coroutine je metoda, která může provádět kód ve smyčce. Může se tedy chovat podobně jako metoda *Update* třídy MonoBehaviour. Výhodou této metody je pro programátora větší kontrolovatelnost. Programátor totiž může Coroutine kdykoliv zastavit nebo spustit.

Zdroj [9].

Coroutine je tedy metoda, která obsahuje část kódu opakující se v určitém intervalu. V Unity jsou předefinované třídy, které zajišťují opakovaní za určitou dobu. Těmito metodami jsou:

- WaitForEndOfFrame – metoda se spustí na konci každého snímku,
- WaitForFixedUpdate – metoda se spustí při každém snímku, při kterém se počítá fyzika,
- WaitForSeconds – metoda se spustí každou sekundu (pokud je v Unity zpomalen nebo zrychlen čas ovlivní to i tuto metodu),
- WaitForSecondsRealTime – metoda se spustí každou sekundu (není ovlivněna časem v Unity).

Zdroje [10], [11], [12], [13].

Layers

Layers neboli vrstvy jsou běžně využívány u kamer. Kameře se nastaví maska vrstev a následně kamera vykresluje pouze objekty, které spadají do vrstev z této masky. Objekt může spadat vždy pouze do jedné vrstvy [14].

Další možností je využití vrstev při hledání objektů ve scéně pomocí tzv. paprsků. Z určitého bodu a určitým směrem je vyslán neviditelný paprsek, kterému se definuje maska vrstev. Pokud paprsek narazí do objektu spadajícího do vrstvy patřící do masky vrstev, tak je paprsek zastaven a dál nepokračuje. Díky tomu lze například nalézt vzdálenost a pozici objektu od určitého místa.

Zdroje [15], [16].

3 UNITY NETWORKING

Jelikož je tato hra založena na propojení více hráčů pomocí lokální sítě. Tak v této kapitole budou vysvětleny základní principy, jak Unity pracuje a komunikuje s klienty po síti. Unity umožňuje komunikovat po síti dvěma způsoby:

- pomocí Transport Layer API,
- pomocí High Level API.

3.1 Transport Layer API

Transport Layer API je v Unity síťová komunikace na nízké úrovni. Programátor se pohybuje mezi vrstvou transportní a síťovou. Výhodou je, že je velice flexibilní a výkonnější, jelikož neobsahuje zbytečně mnoho metod, ale jenom ty nejzákladnější metody pro navázání spojení s klientem, posílání a odesílání paketů atd. Programátor má tak větší volnost pro navrhnutí vlastní komunikace po síti a má větší přehled o veškeré komunikaci po síti. Na druhou stranu značnou nevýhodou tohoto systému je náročnost při vývoji. Programátor je většinou nucen vytvořit si vlastní knihovnu pro komunikaci po síti, což může značně prodloužit vývoj hry.

Zdroje [17], [18].

3.2 HLAPI

HLAPI na rozdíl od Transport Layer API je určena hlavně pro začátečníky, kteří nemají žádné zkušenosti s vývojem her po síti. HLAPI je zaměřeno na vývoj počítačových her po síti, jelikož je to systém obalující mnoho metod a tříd, které vývoj her po síti velice usnadní. Pomocí této architektury je také vytvořena tato hra. V následujících podkapitolách budou popsány hlavní třídy a metody, které obsahuje HLAPI.

Zdroj [19].

NetworkBehaviour

NetworkBehaviour je základní třída pro skripty komunikující po síti. Základní funkcionalitu dědí z třídy MonoBehaviour a je rozšířena o metody a atributy, které jsou užitečné pro komunikaci po síti.

Skript, který je potomkem třídy NetworkBehaviour, může tedy komunikovat se serverem a s klienty pomocí speciálních metod a atributů. Mezi ty nejdůležitější patří:

- SyncVar a Hook metoda,
- Command metoda,
- RPC metoda a cílená RPC metoda,
- Zprávy po síti.

Zdroj [20].

NetworkIdentity

NetworkIdentity je komponenta, kterou musí mít každý objekt, který komunikuje po síti. Komponenta totiž udržuje síťovou identitu objektu a jednoznačně identifikuje objekt mezi klientem a serverem. Hlavním atributem této komponenty je NetworkIntacneId, který je inicializován na serveru a následně nastaven na klientech.

Například pokud je po síti odeslána metoda. Díky této komponentě dokáže koncové zařízení rozpoznat, z kterého objektu byla metoda poslána a koncové zařízení si pak tento objekt najde ve své lokální scéně a provede potřebné věci.

Zdroj [21].

Spawn

V Unity existuje metoda Instantiate, pomocí které lze vytvořit objekt. Pro vytvoření objektu po síti a zobrazení objektu všem připojeným klientům tato metoda nestačí. Proto má Unity tzv. Spawn metodu, která zajistí, že objekt vytvořený na serveru pošle všem připojeným klientům informaci o tom, že si mají na své lokální scéně vytvořit totožný objekt. Objekt musí obsahovat komponentu NetworkIdentity, aby jej bylo možné vytvořit pomocí metody Spawn.

Zdroj [22].

SyncVar

Pro zmenšení zatížení sítě je v Unity podpora tzv. synchronizačních proměnných. Tyto proměnné můžou být použity u objektu typu NetworkBehaviour. Proměnou lze měnit pouze na straně serveru, proto pokud je potřeba změnit proměnnou z klienta, musí klient zavolat command metodu, která potom tuto proměnou změní na serveru.

Ve hře jsou tyto proměnné využity např. u hráče, který si uchovává v těchto proměnných informace o aktuálních životech, rychlosti pohybu, dosahu bomby atd. Velice zjednoduší logiku

aplikace, jelikož se veškeré informace o hráči uchovávají pouze na serveru a klientům jsou odesílány pouze změny těchto proměnných pomocí tzv. hook metod.

Zdroj [23].

Hook metoda

Hook metoda je typ metody, který se přichytí k určité SyncVar proměnné. Jakmile se na serveru změní proměnná SyncVar, tak na všech klientech je zavolána tato Hook metoda.

Zdroj [23].

Command metoda

Velmi zásadním a důležitým prvkem pro komunikaci klienta se serverem je command metoda. Tato speciální metoda je zavolána na klientské straně ovšem vykonána je až na straně serveru. Metoda je často využívána pro změnu synchronizačních proměnných SyncVar.

Zdroj [24].

RPC metoda

Další důležitou metodou v komunikaci po síti je RPC metoda. Tato metoda je zavolána na straně serveru a vykonána na všech klientech, kteří jsou připojeni k serveru. Metoda může být například použita pro změnu scény na všech klientech.

Zdroj [24].

Cílená RPC metoda

Cílená RPC metoda je metoda, která se na rozdíl od klasické RPC metody posílá pouze jednomu konkrétnímu klientovi. Ve hře je tato metoda například použita při položení bomby, kdy server oznámí klientovi, že se mu má v inventáři spustit cooldown u bomby.

Zdroj [24].

Zprávy po síti

Zprávy po síti jsou speciální zprávy, které lze odesílat po síti. Zprávu lze rozeslat buď ze serveru všem klientům či konkrétnímu klientovi nebo může konkrétní klient odeslat zprávu serveru. Zprávy jsou primárně využívány k jednoduchým a rychlým úkolům, kde by bylo zbytečné vytvářet kvůli tomu nový objekt a použít Command nebo RPC metody.

Zdroj [25].

4 IMPLEMENTACE

V této kapitole bude podrobně vysvětlena implementace důležitých objektů ve hře. Jelikož se jedná o objekty, které jsou synchronizovány pomocí počítačové sítě, bude v některých případech vysvětlena implementace části serverové i klientské.

4.1 Lobby

Jakmile se hráč připojí na server je ihned přesunut do lobby. Lobby je místo, na kterém se čeká na všechny hráče před spuštěním hry. Všichni hráči v lobby vidí i ostatní hráče a jejich stav (připraven/nepřipraven). Pokud jsou všichni hráči v lobby připraveni, hráč zakládající server může spustit hru.

Vytvoření lobby

Jakmile hráč založí hru je inicializováno základní nastavení serveru, ke kterému patří nastavení maximálního počtu hráčů, kteří se mohou na server napojit. Po inicializaci je vytvořen objekt LobbyPlayer, který se přiřadí hráči zakládající hru. Díky tomuto objektu může hráč ovládat server.

Připojení do lobby

Předtím než se hráč napojí na server je zkontrolováno, zda je na serveru místo. Pokud na serveru není místo je hráč ze serveru vyhozen. Po vyhození, se hráči zobrazí hláška oznamující, že je server plný (Obrázek 7). V opačném případě je hráč připojen do lobby a je mu vytvořena a přiřazena instance třídy LobbyPlayer pomocí, které může v lobby komunikovat.

Obrázek 7: Ukázka hlášky, která se zobrazí hráči při napojení se na zaplněný server

Hráč v lobby (LobbyPlayer)

LobbyPlayer je síťový objekt, který reprezentuje hráče v lobby. LobbyPlayer obsahuje metody pomocí, kterých komunikuje se serverem. Obsahuje 4 hlavní funkce:

- měnit svůj stav v lobby,
- spuštění samotné hry,
- opuštění lobby,
- a vyhození hráče z lobby.

Všechny zmíněné funkce má pouze hráč zakládající hru a ovládá je pomocí tlačítek, které jsou v lobby. Oproti tomu hráč, který se na server napojil, může pouze měnit svůj stav a opustit lobby (Obrázek 8 a Obrázek 9).

Obrázek 8: Ukázka Lobby na straně serveru

Obrázek 9: Ukázka Lobby na straně klienta

Tlačítko pro změnu stavu hráče

První tlačítko mění stav hráče. Každá instance třídy LobbyPlayer si uchovává svůj aktuální stav v lobby. Při napojení se do hry, má stav nastaven na *nepřipraven*. V lobby pak může svůj stav přepínat pomocí tlačítka na *připraven* nebo *nepřipraven*. Po kliknutí na tlačítko je tlačítko deaktivováno a je zavolána metoda *CmdSetReady*, která je poslána na server a server následně tuto metodu provede. Metoda na serveru změní hráči stav a poté je ze serveru hráči poslána metoda *TargetUpdateReadyButton*. Metoda hráči pouze lokálně aktualizuje text na tlačítku, který zobrazuje aktuální stav hráče a zároveň je znovu toto tlačítko aktivováno. Poté se metoda ukončí a pokračuje se dále v provádění metody *CmdSetReady*. Na konci metody je zavolána metoda *RefreshStartGameButtonServer*, která zkontroluje, zda už jsou všichni hráči připraveni do hry a pokud ano, tak se zakladatelovi hry zaktivuje tlačítko pro spuštění hry.

Tlačítko pro spuštění hry

Pokud je tlačítko pro spuštění hry aktivní, zakladatel hry jej může stisknout a tím spustit hru. Po kliknutí na tlačítko je nastaven statický atribut *Game.IsGameStarted* na hodnotu *true*, tento atribut zajišťuje, aby se nemohl připojit nový hráč do již spuštěné hry. Následně je na serveru zavolána statická metoda *SceneManager.ChangeSceneServer*, která nejprve všem hráčům nastaví atribut *NetworkConnection.isReady* na hodnotu *false* a následně nahraje scénu s prostředím hry.

Hra začíná, jakmile se všem hráčům načte scéna s prostředím hry. Po načtení scény se jednotlivým hráčům mění hodnota atributu *NetworkConnection.isReady* zpátky na *true*, díky tomu může server zjistit, zda se již všem hráčům načetla scéna a následně odstartovat hru.

Tlačítko pro vyhození hráče z lobby

Hráč zakládající hru má u každého hráče speciální tlačítko pro vyhození hráče ze serveru. Po kliknutí na toto tlačítko je ze serveru klientovi odeslána zpráva třídy *DisconnectMessage* pro odpojení ze serveru. Zpráva obsahuje důvod, proč byl hráč vyhozen ze serveru. Tato informace je reprezentována hodnotou z výčtového typu *DisconnectReason*. Na straně klienta je potom tato zpráva přečtena pomocí metody *Message.OnDisconnectClient* (Obrázek 10).

```
private static void OnDisconnectClient(NetworkMessage netMsg)
{
    DisconnectMessage msg = netMsg.ReadMessage<DisconnectMessage>();
    string message;

    switch (msg.reason)
    {
        case DisconnectReason.FullServer:
            message = "Server je plný.";
            break;
        case DisconnectReason.Kicked:
            message = "Byl jste vyhozen ze serveru.";
            break;
        case DisconnectReason.GameAlreadyStarted:
            message = "Nelze se připojit do již spuštěné hry.";
            break;
        default:
            message = "Byl jste odpojen ze serveru.";
            break;
    }

    if (msg.isHost)
        NetworkManager.Singleton.StopHost();
    else
        NetworkManager.Singleton.StopClient();

    AlertWindow.Show(message);
}
```

Obrázek 10: Ukázka metody, která zpracovává zprávu, když je klient vyhozen ze serveru.

Metoda *Message.OnDisconnectClient* (Obrázek 10) nejprve přetypuje zprávu z obecné třídy *NetworkMessage* na zprávu typu *DisconnectMessage*. Následně je ze zprávy získána informace, proč byl uživatel ze serveru vyhozen a podle toho je do lokální proměnné *message* uložena zpráva, která se uživateli zobrazí po dokončení této metody. Poté je na straně klienta zavolána příslušná metoda pro odpojení hráče ze serveru. Po odpojení je hráč přesunut zpátky do scény s hlavní nabídkou a je mu zobrazena hláška proč byl odpojen ze serveru.

4.2 Krabice

Krabice má ve hře velmi důležitou roli, jelikož jedině díky ní lze ve hře vygenerovat pickup.

Obrázek 11: Krabice, z které padá pickup.

Krabice je součástí prostředí, proto je její pozice a počet při každém načtení prostředí stejný. Z pohledu implementace je to velice primitivní objekt, který má za úkol po svém zničení vygenerovat pickup. Rozhodnutí, jestli bude krabice obsahovat pickup je provedeno na začátku každého kola, kdy jsou nalezeny všechny krabice v prostředí a následně je pro každou krabici volána metoda, která s určitou pravděpodobností přiřadí krabici náhodný pickup nebo krabici nepřiřadí žádný.

K zajištění správné funkčnosti je nutné, aby krabice osahovala tyto komponenty: BoxCollider, WoodBoxController a NetworkIdentity.

BoxCollider je velice důležitou komponentou, jelikož se stará o kolize s ostatními objekty a díky němu lze detekovat, zda byla krabice zasažena bombou. Zabraňuje také hráči, aby prošel skrz krabici.

Další důležitou komponentou je WoodBoxController, jelikož je potomkem třídy NetworkBehaviour je nutné, aby objekt obsahoval komponentu NetworkIdentity. Obsahuje jednu metodu, která je volána na straně serveru. Metoda kontroluje kolizi mezi krabicí a výbuchem bomby (Obrázek 12). Pokud nastane kolize, je zavolána metoda *SpawnPickUpAndDestroyServer*.

```
else if (Utils.CompareTag(other, TagEnum.WoodBox))
{
    var woodBox = other.GetComponent<WoodBoxController>();

    spawnedPickup = woodBox.SpawnPickUpAndDestroyServer();
}
```

Obrázek 12: Kolize krabice s výbuchem bomby

Metoda *SpawnPickUpAndDestroyServer* (Obrázek 13) využívá tento atribut nejprve pro zjištění, zda krabice pickup obsahuje a pokud ano, tak je zavolána statická metoda PickUpControlleru pro zrození pickupu na straně serveru. Poté je samotná krabice zničena a na místě zničení je přehrán zvuk, který se podobá zvuku při rozbití dřevěné krabice.

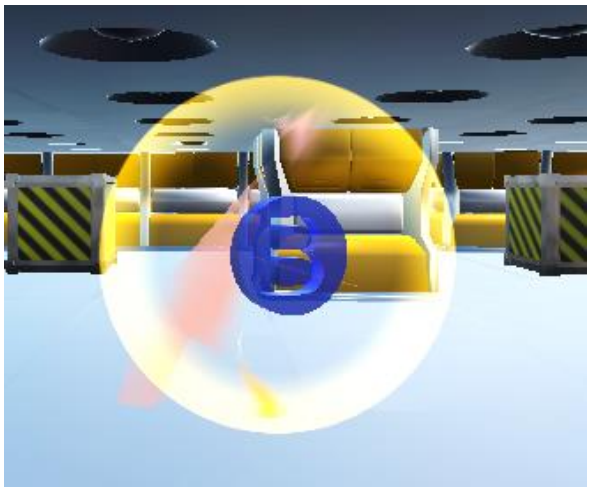
```
public class WoodBoxController : NetworkBehaviour {

    public Ability ability;

    1 reference | Dominik Veselý, 13 days ago | 1 author, 4 changes
    public PickUpController SpawnPickUpAndDestroyServer()
    {
        PickUpController pickup = null;

        if (ability != null)
        {
            pickup = PickUpController.InstantiateAndSpawnServer(ability, transform.position);
        }

        Utils.DestroyServer(gameObject);

        PlayClipMessage.SendToAll(SoundEnum.WoodBoxCrack, transform.position, 5);

        return pickup;
    }
}
```

Obrázek 13: Zdrojový kód Krabice

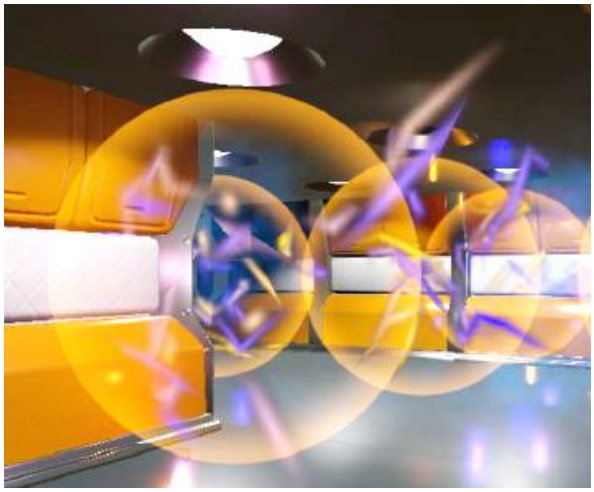
4.3 Pickup

Hra obsahuje šest různých pickupů, které budou podrobně probrány v této kapitole. Nejprve, ale bude probrána obecná implementace pickupu.

Pokud se po rozbití krabice zrodí pickup je s jeho zrozením vytvořen i částicový efekt (Obrázek 14 a Obrázek 15), který znázorňuje zrození samotného pickupu. Spolu s efektem je také přehrán zvuk připomínající elektrický výboj.



Obrázek 14: Pickup s částicovým efektem



Obrázek 15: Ukázka částicových efektů

Po chvíli tento efekt zmizí a v prostředí už je pouze samotný pickup. Pickup je hráčem automaticky sebrán, jakmile se k němu přiblíží dostatečně blízko a nastane kolize mezi hráčem a pickupem. Pokud se tak stane, pickup je okamžitě aplikován na hráče a hráč získává schopnost tohoto pickupu.

Existují dva druhy pickupů: okamžitý a časový. Schopnost z okamžitého pickupu je hráči přidána napořád, na rozdíl od časového pickupu, kde se schopnost aplikuje na hráče pouze na omezenou dobu a poté je schopnost hráči odebrána. Aktuální časové pickupy jsou hráči zobrazovány v GUI (Obrázek 16) za pomocí ikonek a jejich částečné překrytí udává zbývající dobu pickupu.

Obrázek 16: Ukázka časových pickupů v GUI

Pickup lze také zničit pomocí bomby. Pokud totiž výbuch bomby zasáhne pickup, je nenávratně z prostředí odstraněn. Nyní budou podrobně popsány jednotlivé druhy pickupů.

Uzdravení

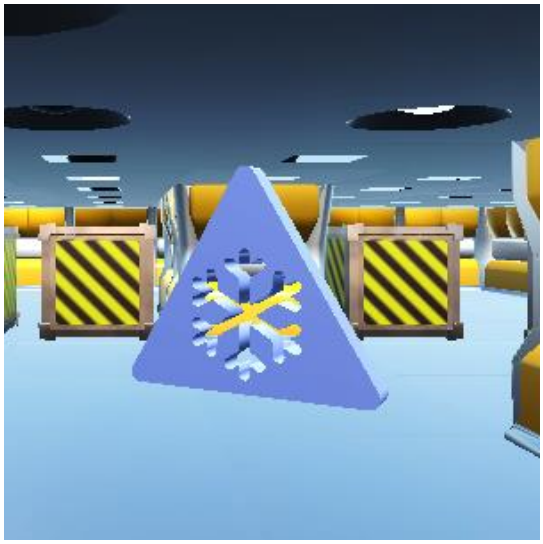
Pickup pro uzdravení je jedním ze dvou pickupů patřící mezi okamžité pickupy. Po jeho sebrání je hráči zvýšeno určité množství zdraví.

Zmražení

Pickup pro zmražení je prvním pickupem patřící do kategorie časových (Obrázek 17). Na rozdíl od ostatních z této kategorie je pickupem negativním. To znamená, že dá hráči negativní schopnost, která ho oproti ostatním hráčům znevýhodní. Proto je aplikován na ostatní živé protihráče na místo toho, aby byl aplikován na hráče, který jej sebral.

Po sebrání tohoto pickupu jsou ostatní hráči zmražení a nemůžou se po určitou dobu hýbat. Pokud je zmražený hráč zasažený bombou, tak mu je tato schopnost odebrána předčasně a může se opět pohybovat.

Obrázek 18: Ukázka barvy hráče po aplikování pickupu zmražení



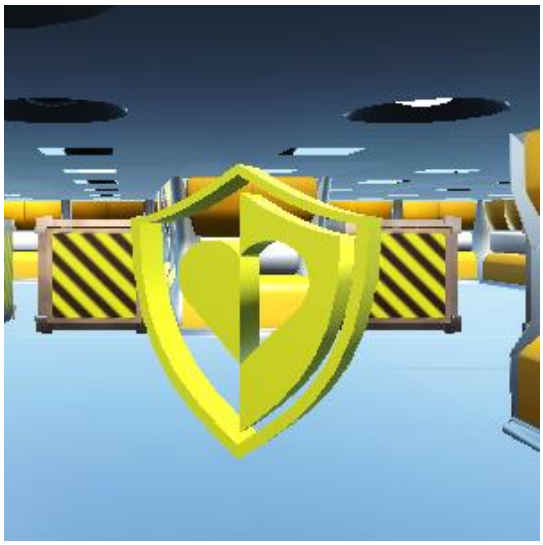
Obrázek 17: Ukázka pickupu zmražení

Zmražený hráč je zbarvený do modra (Obrázek 18), aby ostatní hráči poznali, že se nemůže pohybovat. Protože hráč nevidí vlastní postavu a nepoznal by tak, že je zmražený. Je mu do levého horního rohu v GUI přidána ikonka s vločkou překrytá modrým pozadím (Obrázek 16), které postupně mizí a znázorňuje tak dobu, kdy zmražení skončí.

Nesmrtelnost

Pickup patřící mezi pickupy časové je i pickup pro nesmrtelnost (Obrázek 19). Po jeho sebrání, získá hráč na krátkou chvíli nesmrtelnost a díky tomu je hráč imunní proti veškerému poškození, které obdrží.

Obrázek 20: Ukázka barvy hráče po aplikování pickupu nesmrtelnosti



Obrázek 19: Ukázka pickupu pro nesmrtelnost

Jakmile hráč získá nesmrtelnost, ostatní hráči vidí jeho postavu zbarvenou do červena (Obrázek 20), aby poznali, že je hráč nesmrtelný. Hráči, který sebral tento pickup, se zobrazí v levém horním rohu obrazovky ikonka štítu (Obrázek 16), který značí nesmrtelnost.

Zvýšení rychlosti

Další pickup, který patří mezi časové pickupy je pickup zvyšující rychlost pohybu. Po získání pickupu je hráči zvýšena procentuálně rychlost. Pokud hráč získá během aktuálního pickupu, další pickup tohoto typu, je pouze prodloužena doba jeho trvání a rychlost zůstává stejná. Po skončení pickupu je hráči opět vrácena původní rychlost.

Hráči, který sebral tento pickup, se zobrazí v levém horním rohu ikonka znázorňující zvýšení rychlosti a aktuální doba trvání pickupu (Obrázek 16).

Zvýšení dosahu bomby

Poslední pickup, patřící do kategorie časových pickupů, je pickup zvyšující dosah bomby. Po sebrání tohoto pickupu se hráči zvětší dosah všech bomb, které má v inventáři.

Jakmile hráč položí bombu, je bombě přiřazen dosah bomby z pickupu. Bombě je přiřazen vždy větší dosah, pokud má tedy bomba základní dosah větší, než který by získala z pickupu, tak se její dosah nezmění.

Spouštěcí bomba

Úplně posledním pickupem je spouštěcí bomba patřící do kategorie okamžitých pickupů. Po sebrání pickupu hráč obdrží celkem tři spouštěcí bomby. Tyto bomby je možné odpálit předčasně. Spouštěcí bomba bude podrobněji vysvětlena v následující kapitole věnující se právě bombám.

4.4 Bomba

Bomba patří mezi nejdůležitější předměty ve hře, díky kterému lze zneškodnit protihráče, ale i sebe. V této kapitole bude podrobněji popsána její implementace.

Položení bomby

Bombu lze pokládat pouze na podlahu. Podlaha se skládá z malých čtverců. Tyto čtverce určují místo položení bomby, jelikož je bomba vždy položena na střed tohoto čtverce. Bombu lze položit pouze na čtverec, který je volný.

Čtverec je volný, jakmile nemá žádnou kolizi s jiným objektem. Pokud je tedy na čtverci hráč, pickup nebo bomba, nelze na čtverec položit bombu.

Kontrola, zda je čtverec volný je prováděna nejprve na straně klienta, aby se předešlo zbytečnému zatěžování sítě. Pokud je čtverec volný, zavolá se na klientovi command metoda pro vytvoření bomby s parametry obsahující název, pozici a rotaci bomby. Tato command metoda je vykonána až na straně serveru.

Na straně serveru je v command metodě volána metoda *SpawnBombServer* (Obrázek 21). Metoda si nalezne v hráčově inventáři bombu podle názvu, který byl odeslán jako parametr s metodou. Pokud bomba není připravena k použití, znamená to, že má bomba stále cooldown nebo místo pro položení bomby není volné a bomba se tedy nepoloží.

Jestliže je místo volné, nic nebrání tomu, aby byla bomba položena. Položení bomby je provedeno pomocí statické metody *BombController.InstantiateAndSpawnServer*, který nejprve vytvoří bombu na straně serveru a následně bombu vytvoří na všech klientech. Poté je hráči snížen počet bomb v inventáři. Bomba je z inventáře odebrána, jestliže má položka v inventáři nulový počet bomb.

```
private void SpawnBombServer(string bombName, Vector3 bombPosition, Quaternion rotation)
{
    if (!isServer) return;

    var item = inventory[bombName];
    FloorController closestFloor;

    if (!item.IsReady || !BombController.IsPositionValidForBomb(bombPosition, out closestFloor))
    {
        TargetBombWasNotPlanted(connectionToClient);
        return;
    }

    BombController.InstantiateAndSpawnServer(item.Bomb, closestFloor.transform.position,
        closestFloor.transform.rotation, this);

    if (connectionToClient.hostId != -1)//neni host
        TargetBombWasPlanted(connectionToClient, bombName);
    else//host
        WaitForAnswerByServer = false;

    item.StartCooldownServer();
    inventory.RemoveCount(bombName);
}
```

Obrázek 21: Ukázkový kód metody pro položení bomby

Výbuch bomby

Po položení bomby je na bombě spuštěn časovač a po jeho vypršení bomba vybuchne. Bomba může také vybuchnout dříve, za předpokladu, že jí zasáhne výbuch z jiné bomby.

Během odpočítávání je z bomby slyšet zvuk pípání, který se lineárně zrychluje podle zbývajícího času na bombě. Hlasitost pípání se snižuje v závislosti na vzdálenosti hráče od bomby. Po vypršení časového limitu se výbuch bomby postupně rozšiřuje do všech čtyř směrů, než docílí své maximální vzdálenosti. Narazí-li výbuch do zdi, tak výbuch v tomto směru dále nepokračuje.

Kontrolu, zda výbuch narazil do zdi, zajišťuje metoda *BombContorller.IsWallInDirection* (Obrázek 22). Pokud výbuch narazí do zdi, metoda vrátí *true*, jinak vrací *false*. Vstupními parametry metody jsou:

- direction – určuje směr výbuchu,
- maxDistance –maximální možná vzdálenost výbuchu,
- hit – výstupní parametr obsahující informace o potencionálním zásahu zdi.

```
4 references | Dominik Veselý, 23 days ago | 1 author, 1 change
private bool IsWallInDirection(Vector3 direction, float maxDistance, out RaycastHit hit)
{
    var ray = new Ray(transform.position, direction);

    return Physics.Raycast(ray, out hit, maxDistance, Layer.OnlyLayer(LayerEnum.Wall));
}
```

Obrázek 22: Ukázka metody zjišťující náraz mezí zdí a výbuchem bomby

Metoda je volána do všech směrů výbuchu. Nejprve se vytvoří paprsek, pomocí konstruktoru třídy *Ray*, která má 2 vstupní parametry. První určuje místo odkud bude paprsek vyslán a druhý určuje jeho směr. Následně je vykonána statická metoda *Physics.Raycast* s těmito vstupními parametry:

- ray – parsek určující pozici a směr jeho putování,
- hit – výstupní parametr obsahující informace o potencionálním zásahu objektu paprskem,
- maxDistance – maximální délka paprsku
- LayerMask – maska vrstev.

Protože se výbuch může zastavit pouze o zeď je posledním parametrem metoda *Layer.OnlyLayer*, která vrací masku obsahující jedinou vrstvou *Wall*. Pokud tedy paprsek při své cestě narazí do objektu patřící do vrstvy *Wall*, tak bude přerušen dřív, než dosáhne své maximální vzdálenosti. Proto musí každá zeď patřit do vrstvy *Wall*, aby se paprsek o zeď zastavil a výbuch nepokračoval dál za zeď. Výstupní parametr *hit* obsahuje informace o vzdálenosti, kterou parsek urazil. Díky této informaci lze zjistit, do jaké vzdálenosti bude výbuch v určitém směru generován.

Jednotlivé části výbuchu jsou samostatný objekt, který obsahuje komponentu *BoxCollider* pomocí, které je zajištěno detekování kolizí s ostatními objekty. Výbuch kontroluje celkem čtyři kolize:

- s hráčem – při této kolizi je hráči uděleno poškození podle bomby z které výbuch vzešel,
- s bombou – pokud výbuch zasáhne jinou bombu, tak bomba vybouchne také,
- s krabicí – krabice je po kolizi s výbuchem zničena,
- s pickupem – stejně jako krabice, je pickup zničen po kolizi s výbuchem.

Všechny tyto kolize se kontrolují na straně serveru (Obrázek 23) a klientovi jsou pouze posílány informace o zničení objektu či o aktualizaci statistik hráče.

```
0 references | Dominik Veselý, 23 days ago | 1 author, 3 changes
void OnTriggerEnter(Collider other)
{
    if (!isServer) return;

    if (Utils.CompareTag(other, TagEnum.Player))
    {
        other.GetComponent<GamePlayer>().TakeDamageServer(Bomb);
    }
    else if (Utils.CompareTag(other, TagEnum.Bomb))
    {
        other.GetComponent<BombController>().BlowUpServer();
    }
    else if (Utils.CompareTag(other, TagEnum.WoodBox))
    {
        var woodBox = other.GetComponent<WoodBoxController>();

        spawnedPickup = woodBox.SpawnPickUpAndDestroyServer();
    }
    else if (Utils.CompareTag(other, TagEnum.PickUp)
        && other.GetComponent<PickUpController>() != spawnedPickup)
    {
        Utils.DestroyServer(other.gameObject);
    }
}
```

Obrázek 23: Ukázka detekce kolizí výbuchu bomby s ostatními objekty

Základní bomba

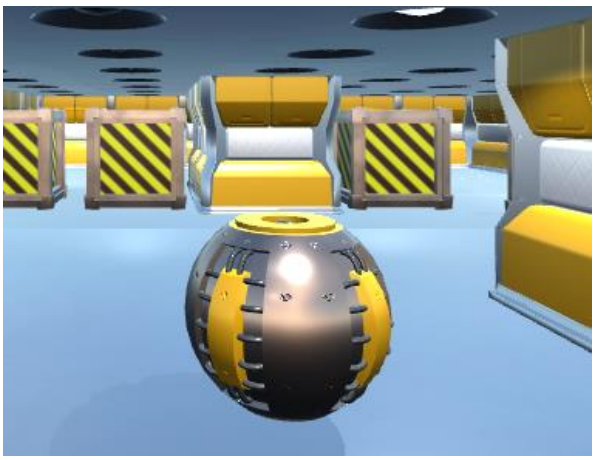
Základní bomba (Obrázek 24) je základní zbraní ve hře. Bombu má v inventáři každý hráč od začátku hry a její počet je neomezený. Hlavními vlastnostmi bomby jsou:

- poškození – určuje množství poškození, které je hráči uděleno,
- dosah bomby – maximální vzdálenost, po kterou je výbuch bomby generován,
- cooldown – doba, po kterou bombu nelze znovu použít,
- zpoždění výbuchu – určuje za jak dlouho exploduje bomba,
- trvání výbuchu – určuje, jak dlouho bude moci výbuch zasáhnout hráče,
- ikonka – obsahuje obrázek bomby, který je zobrazen v inventáři hráče.

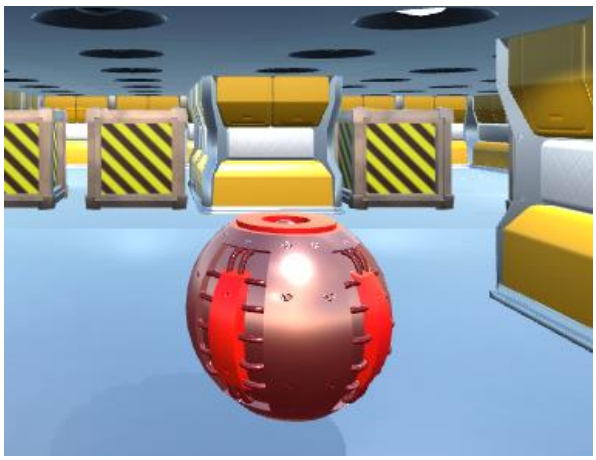
Spouštěcí bomba

Spouštěcí bomba (Obrázek 25) je speciální bomba, která obsahuje stejné vlastnosti, jako základní bomba jen s odlišnými hodnotami. Bomba například uděluje hráči větší poškození, ale na druhou stranu má větší cooldown. Cooldown na spouštěcí bombě je dvojnásobný od cooldownu základní bomby. Hráč tedy stihne položit základní bombu dvakrát během jednoho cooldownu spouštěcí bomby.

Hlavním rozdílem od základní bomby je možnost předčasného odjištění bomby. Hráč, který bombu položil má možnost vzdáleně odjistit bombu a spustit tak její výbuch dříve. Jakmile hráč stiskne tlačítko pro odjištění bomby jsou odjištěny všechny nevybuchlé spouštěcí bomby v prostředí, které tento hráč položil.



Obrázek 24: Ukázka základní bomby



Obrázek 25: Ukázka spouštěcí bomby

4.5 Hráč ve hře (GamePlayer)

Nejdůležitějším prvkem ve hře je samozřejmě samotný hráč, kterého řídí třída *GamePlayer* a bez kterého by samotná hra nemohla fungovat.

Základní vlastnosti

Hráč ve hře má tyto základní vlastnosti:

- zdraví,
- rychlost pohybu,
- globální dosah bomby,
- počet zabití,
- počet smrtí,
- skóre.

Tyto vlastnosti se při spuštění hry všem hráčům inicializují na stejné hodnoty. Následně se během hry vlastnosti mění.

Nejdůležitější vlastností hráče je zdraví. Na začátku hry má hráč maximální zdraví a při každém zásahu bombou je mu určité množství odečteno podle poškození bomby, která jej zasáhla. Pokud se hráčovo zdraví dostane na nulu, tak zemře.

Dále jsou důležitými vlastnostmi statistická data o počtu zabití, počtu smrtí a celkovém skóre hráče. Pokud hráč zabije jiného hráče je mu zvýšen počet zabití a skóre. Naopak pokud hráč zemře je mu přičtena smrt a pokud se zabil vlastní bombou je mu odečteno skóre.

Tabulka se statistikami hráčů

Během hry je možné si zobrazit tabulku se statistikami hráčů (Obrázek 26). V tabulce je možné vidět tyto statistiky:

- počet zabití,
- počet smrtí,
- skóre.

Hráči jsou v tabulce seřazeni dle skóre vzestupně. Proto je hráč s nejvyšším skóre v tabulce na prvním místě (Obrázek 26). Tabulka je aktualizována pomocí událostí, které jsou přichyceny na jednotlivé hráče. Jakmile se změní některému z hráčů hodnota v tabulce. Díky události není potřeba aktualizovat celou tabulku, ale pouze hodnotu, která byla změněna. Pokud se změní hráči skóre je vždy zavolána metoda pro nové seřazení všech hráčů v tabulce. Hráči jsou řazení sestupně podle skóre.

Obrázek 26: Ukázka tabulky se statistikami hráčů ve hře

Pohyb hráče

O pohyb hráče se stará třída *CharacterContorller*, která je součástí *Unity*. Třída obsahuje vlastní *Collider*, kterým vyhodnocuje kolize s ostatními objekty ve scéně. Pro pohyb hráče je používána metoda *Move* třídy *CharacterContorller*, která má jediný parametr udávající směr hráčova pohybu. V této metodě je řešená veškerá fyziky a využívá zmiňovaný *Collider*, pro zjištění, zda hráč při svém pohybu nenarazil do zdi či jiného neprůchozího objektu. Pokud by se tak stalo, tak by se hráčův pohyb neuskutečnil.

Animace hráče

O animace hráče se starají komponenty *Animator* a *NetworkAnimator*. Komponentě *Animator* se nastaví atributy *controller* a *avatar*. Atribut c*ontroller* obsahuje logiku animací a atribut *avatar* je figura hráče, kterou *controller* „probudí k životu“. Jelikož komponenta *Animator* umí tyto animace provádět pouze lokálně. Musí být na hráči připnuta i druhá komponenta

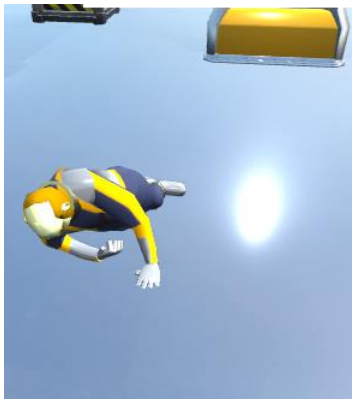
NetworkAnimator, která zajistí sdílení animací po síti. *NetworkAnimator* tedy posílá ostatním klientům přes síť informace o tom, které animace se mají u postavy provádět.

Animace se spouští za určitých situacích. Postava hráče má tři druhy animací:

- animace při nečinnosti (Obrázek 27 a Obrázek 28),
- animace chůze,
- animace smrti (Obrázek 29).

Obrázek 27: Ukázka postavy hráče v nečinnosti

Obrázek 28: Ukázka postavy hráče ve druhé fázi nečinnosti



Obrázek 29: Ukázka postavy hráče při úmrtí

První animace je spuštěna automaticky, jakmile se hráč připojí do hry.

Pokud se dá hráč do pohybu je spuštěna animace pro chůzi, která animuje kroky hráče a při stálém pohybu je tato animace opakována ve smyčce, dokud se hráč nezastaví. Rychlost animace je také ovlivněna aktuální rychlostí hráče, pokud se tedy hráči zvýší rychlost pohybu, zrychlí se i rychlost animace chůze. Po zastavení hráče je spuštěna opět první animace.

Poslední animací je animace smrti. Tato animace, jak již název napovídá, je spouštěna, jakmile hráč zemře.

Zvuky hráče

Na hráči je připnuta komponenta *AudioSource*, pomocí které lze přehrávat libovolné zvuky. Komponenta obsahuje mnoho možností, kterými se nastavuju např. hlasitost zvuku, doba přehrávání či opakující se smyčka atd. Ovšem nejdůležitějším atributem komponenty je *AudioClip,* který obsahuje konkrétní zvuk, který bude komponentou přehráván.

Skript, tak mění pouze zmiňovaný atribut *AudioClip* a ostatní atributy se nemění. Hráč může vydávat tři různé zvuky:

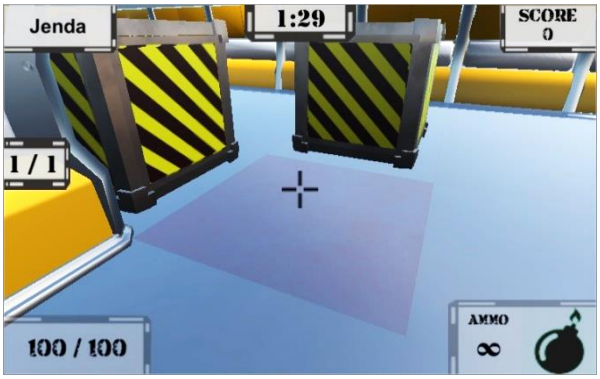
- při pohybu je vydáván zvuk chůze,
- pokud je hráč zasažen bombou, přehraje se zvuk připomínající heknutí,
- při smrtí je přehrán zvuk, který znázorňuje poslední výkřik před smrtí.

Všechny tyto zvuky jsou přehrávány za určité situace a nikdy se nemůže stát, že by se přehrávaly dva zvuky zároveň. Zvuky byly použity ze stránek freesound.org a od kolegy Jana Houžvičky.

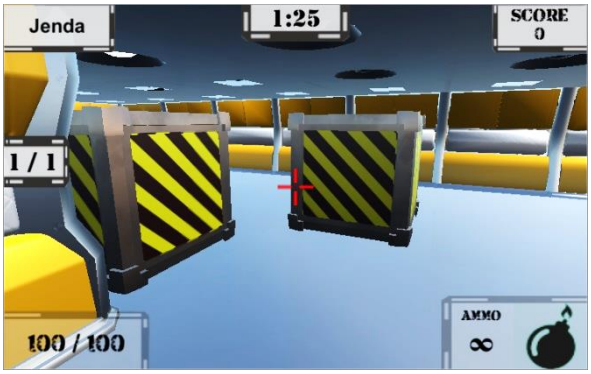
Místo položení bomby

Pro lepší rozpoznání, jestli je místo pro položení bomby obsazené nebo volné, byla přidána funkce, která dané místo zvýrazní červeně (Obrázek 30), pokud na toto místo lze položit bombu.

Další nápovědou pro hráče může být také tzv. crosshair[2], který je uprostřed obrazovky (Obrázek 31). Crosshair jednak ukazuje na místo, kam hráč míří, ale také mění barvu podle toho, zda může bombu položit nebo ne.



Obrázek 30: Ukázka zvýrazněného místa pro položení bomby



Obrázek 31: Ukázka změny barvy crosshairu

Pro určení místa položení bomby je použita statická metoda *Physics.Raycast*. Tato metoda vyšle imaginární paprsek z prostředka obrazovky ve směru, kam hráč v danou chvíli míří. Paprsek se zastaví, jakmile narazí do země nebo pokračuje do „nekonečna“. Pokud paprsek narazil do země metoda vrátí hodnotu *true* do výstupního parametru je uložena souřadnice nárazu se zemí. Tímto lze určit potencionální místo pro položení bomby.

[2] Crosshair je často využíván v FPS hrách, kdy hráči pomáhá s mířením se zbraní. Většinou se vyskytuje uprostřed obrazovky.

Inventář

Každý hráč má inventář obsahující bomby, které má k dispozici. Jednotlivé položky v inventáři obsahují číslo, které udává aktuální počet bomb v inventáři. Na lokálním zařízení má inventář pouze lokální hráč a ostatní hráči inventář nemají, jelikož je to zbytečné.

Naopak na serveru mají inventář všichni hráči, jelikož na serveru běží veškerá logika o rozhodování, zda může hráč bombu položit. Bez znalosti inventáře by server nevěděl, jaké bomby hráč vlastní a ani by nemohl přiřadit bombě cooldown po jejím použití.

Hráč může v inventáři libovolně listovat mezi bombami. Po použití bomby je u inventáře snížen počet u konkrétní bomby. Pokud se počet dostane na nulu je bomba z inventáře odstraněna.

4.6 Průběh hry

V této kapitole budou probrány jednotlivé situace, které mohou v průběhu hry nastat. Převážně tyto situace řeší třída *GameController* a o divácký režim se stará třída *SpectateController*.

Inicializace hry na straně serveru

Při načtení scény s prostředím se na straně serveru vytvoří prázdný seznam hráčů, který se bude postupně plnit hráči. Tento seznam si server uchovává, aby měl rychlejší přístup k hráčům, kteří jsou připojeni do hry.

Následně jsou inicializovány pomocné proměnné pro zjišťování, zda bylo kolo ukončeno (*isEndedRoundServer*) nebo zda právě teď běží (*isRoundRunningServer*). Také je vytvořeno pole, které bude zaznamenávat časy jednotlivých kol. Nakonec jsou na serveru přiřazeny události, které jsou v průběhu hry spouštěny.

Inicializace hry na straně klienta

Na straně klienta je nastaven v GUI aktuální čas kola a aktuální číslo kola ve hře. Tyto informace jsou poslány ze serveru. Ke klientovi jsou přiřazeny také události, které jsou v průběhu hry za určitých okolností spouštěny.

Jakmile se hráči načte scéna s prostředím je přidán do scény i objekt *GamePlayer*, který bude moci klient ovládat. Zároveň je tento objekt přidán do seznamu hráčů, který byl inicializován na serveru. Přidáním hráče do seznamu je pro server znamení, že je daný hráč připraven na spuštění hry.

Před spuštěním hry

Server čeká až se načte všem klientům scéna a provede se inicializace hry. Jakmile jsou všichni hráči připraveni, může být spuštěno první kolo hry. Jelikož hra je rozdělena na 5 hracích kol.

Před spuštěním nového kola

Před začátkem kola je změněna hodnota pomocné proměnné *isEndedRoundServer*. Tato proměnná kontroluje, zda bylo již ukončeno předchozí kolo a tím se tak předešlo nečekané situaci, kdy by se spouštělo nové kolo, přestože ještě předchozí neskončilo. U prvního kola tato kontrola projde vždy, jelikož je to první kolo.

Dále je nastaven čas kola a změněno číslo aktuálního kola, pak je spuštěna událost *OnInitRound*, která obsahuje metodu *RespawnAllPlayersServer* (Obrázek 32) pro zrození všech hráčů a jejich inicializaci pro nové kolo.

```
private void RespawnAllPlayersServer()
{
    var spawns = SpawnController.GetAllSpawns().ToList();
    //var players = GamePlayer.GetAlivePlayers();

    if (spawns.Count < players.Count())
        throw new InvalidOperationException(
            string.Format("Neexistuje dostatek pozic ({0}) pro respawn všech hráčů ({1}).",
            spawns.Count, players.Count()));

    foreach (var player in players)
    {
        var spawnIndex = UnityEngine.Random.Range(0, spawns.Count - 1);
        var spawn = spawns[spawnIndex];

        LeaveSpectatorModeServer(player);
        player.CanMoveAndPlantServer(false);
        player.RespawnServer(spawn.transform);
        spawns.RemoveAt(spawnIndex);
    }
}
```

Obrázek 32: Ukázka metody pro zrození všech hráčů na serveru

Nakonec je spuštěna *Coroutine*, která spustí odpočet pro začátek nového kola a po dokončení tohoto odpočtu je spuštěno nové kolo (Obrázek 33).

Obrázek 33: Ukázka odpočtu před zahájením nového kola

Nové kolo

Při spouštění nového kola je všem hráčům povolen pohyb a pokládání bomb na mapě a poté je zahájen odpočet kola. Spuštění kola je oznámeno všem klientům pomocí *RPC* metody *RpcNewRound*, která na všech klientech zavolá událost *OnNewRound*.

Divácký režim

Pokud hráč zemře a kolo není ukončeno je přemístěn do diváckého režimu. V tomto režimu může sledovat ostatní živé hráče a může mezi nimi přepínat. Divák má také aktualizované GUI podle toho, kterého hráče právě sleduje. V GUI se aktualizuje jméno hráče, skóre a aktuální životy hráče. Inventář sledujícího hráče divák nevidí.

Pokud hráč zemře nebo se odpojí během toho, co ho sleduje jiný hráč, tak se divákovi zobrazí další živý hráč v pořadí. Jestliže se hráč odpojí a na mapě není žádný živý hráč, je divák přesunut zpátky na vlastní mrtvou postavu, než začne nové kolo.

Konec kola

Konec kola nastane buď, že skončí odpočet kola nebo pokud na mapě zůstane pouze jeden živý hráč. Pokud nastane první možnost a bude na mapě více živých hráčů. Bude kolo ukončeno a statistiky živých hráčů budou nezměněny.

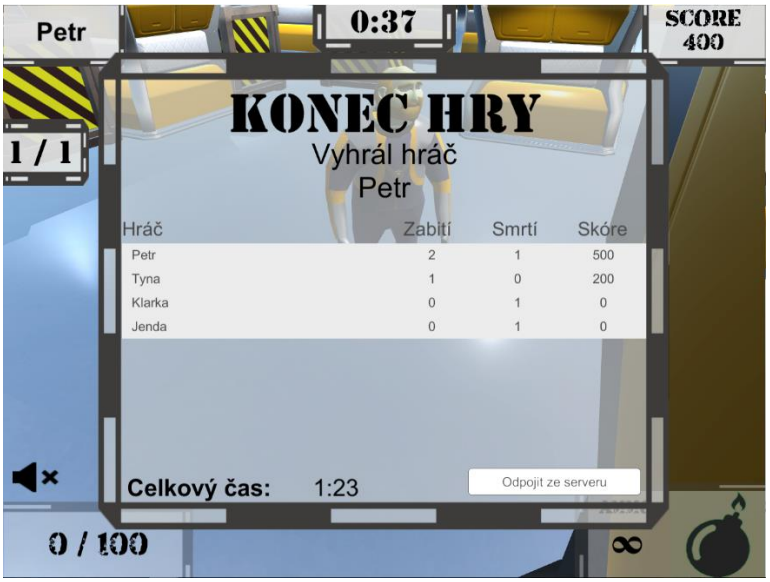
Po skončení kola je změněna hodnota proměnné *isEndedRoundServer*, která zabraňuje ukončení již ukončeného kola. Do pole obsahující jednotlivé časy kol je zaznamenán uplynulý

čas kola. Následně je zavolána *RPC* metoda *RpcEndRound*, která na všech klientech vyvolá událost *OnEndRound* a oznámí tím klientům skončení kola.

Nakonec je pomocí podmínky zjišťováno, zda bude nějaké kolo následovat nebo to bylo poslední kolo ve hře. Pokud se nejednalo o poslední kolo ve hře je spuštěna *Coroutine*, která po menší pauze spustí opět nové kolo. Jednalo-li se o finální kolo ve hře, tak je hra ukončena.

Konec hry

Při skončení hry je všem hráčům zobrazena tabulka (Obrázek 34) obsahující jméno vítězného hráče, statistiky všech hráčů a celkový čas hry. Tabulka také obsahuje tlačítko pro návrat do hlavní menu nabídky. Po kliknutí na tlačítko je hráč odpojen ze serveru.



Obrázek 34: Ukázka tabulky se shrnutím na konci hry

5 SHRNUTÍ

Tato kapitola bude shrnovat celkový výsledek z praktické části. Jak výsledná hra vypadá, jaké nedostatky obsahuje a následně postupy, jak tyto nedostatky odstranit.

5.1 Porovnání s titulem Bomberman

Výsledkem práce je hra pro více hráčů inspirována titulem Bomberman. Z tohoto titulu si hra vzala jen základní hrací prvky, ale celková hra se značně liší od historického titulu. V této podkapitole budou shrnuty společné a rozdílné vlastnosti oproti staršímu titulu. Pro upřesnění je hra porovnávána se hrou *Dyna Blaster* z roku 1992.

Společné vlastnosti

Stejně jako v originálním titulu Bomberman je hlavní zbraní bomba. Bomba je pokládána na zem poskládaná z čtverečků. Výbuch bomby se rozšiřuje do čtyř směru po jednotlivých čtvercích na podlaze.

V prostředí mapy jsou také rozmístěny krabice, které lze rozbít a může z nich vypadnout pickup. Tento pickup dá hráči speciální schopnost nebo je naopak na ostatní protihráče aplikována negativní schopnost.

Hra je rozdělena na kola. Kolo skončí buď po skončení časového limitu nebo jakmile zůstane na mapě poslední živý hráč. Každé kolo hráč získává body za zabití protihráčů či vyhrání samotného kola. Po skončení posledního kola je vyhodnocena hra a hráč s nejvyšším skóre vítězí.

Rozdílné vlastnosti

Hlavním rozdílem je možnost hrát hru s více hráči přes lokální síť. Starší titul sice bylo také možné hrát s více hráči, ale hráči byli nuceni hrát hru na jednom přístroji, kde byl pohled shora na celou mapu. To znamená, že hráči viděli sebe, ale i své protihráče po celou dobu svého hraní. Toto aktuální hra eliminuje, jelikož je hra v trojrozměrném prostředí a každý hráč hraje na vlastním přístroji mající pohled z první osoby.

Dalším rozdílem oproti staršímu titulu je zdraví. Ve staré hře byl hráč po zásahu bombou ihned mrtev. V této práci byla snaha toto změnit. Ve hře má tak každý hráč zdraví procentuálně a může přežít více zásahů bombou. Díky tomu se prodloužila hrací dobu kola, ale také to přidalo na zábavě, jelikož má hráč díky procentuálnímu zdraví vetší šanci na přežití.

Posledním velkým rozdílem je absence umělé inteligence. Jediným nepřítelem ve hře jsou ostatní protihráči, které ovládá živý hráč. Tím se liší starší titul, protože ten měl v prostředí nepřátelé ovládané umělou inteligencí, ale také bylo možné přidat do hry hráče ovládaného počítačem.

5.2 Možné zlepšení

V budoucnu by mohla být hra obohacena například nepřáteli, kteří by měli umělou inteligenci. Také by prostředí mohlo obsahovat různé pasti, které by byly rozmístěné po mapě. Tím by se hra více oživila a pro hráče by byla hra více zábavnější.

Další věc, která by mohla projít úpravou je vyhodnocování skóre. Nyní totiž hra obsahuje málo faktorů, za které může hráč obdržet skóre. Tím dochází na konci hry k časté shodě skóre mezi hráči a systém pak nemůže rozhodnout o vítězi. Řešením by tak mohlo být přidat více faktorů, za které by bylo možné získat skóre nebo při shodě skóre zohledňovat další statistická data hráče. Například zdraví, počet zabití nebo počet smrtí hráče.

Hra nyní obsahuje pouze jednu mapu, proto se nabízí vytvořit více map, které by se lišily rozměry, strukturou uliček a rozmístěním krabic.

Poslední zlepšení by se týkalo inventáře, který by se rozšířil o více druhů bomb. Aktuálně hra obsahuje pouze dva druhy bomb. Dalším typem bomby by mohla být například mrazící bomba, která by zmrazila všechny zasažené hráče nebo bomba jejíž výbuch by se nerozšiřoval postupně jak je tomu u aktuálních bomb, ale výbuch bomby by se vygeneroval okamžitě v celém jejím rozsahu výbuchu. Také by se mohly u bomb pozměnit pouze parametry, které určují množství poškození, vzdálenost výbuchu, délku výbuchu atd.

ZÁVĚR

V rámci bakalářské práce se mi podařilo vytvořit plnohodnotnou 3D hru pro více hráčů po síti LAN. Značnou inspiraci jsem si vzal z titulu Bomberman, podle kterého také výsledná hra nese název. Ovšem výsledná funkcionalita hry se značně liší od prvních vydaných titulů hry.

Teoretická část měla za úkol čtenáři představit herní engine Unity a jeho základní funkce pro komunikaci po síti. Následně byla podrobně popsána implementace hlavních předmětů a funkcionalit ve hře. V poslední kapitole byly předvedeny praktické ukázky ze hry.

Praktická část mi zabrala nejvíce času, jelikož se věnovala samotnému návrhu hry a následnému vytvoření hry. Díky této práci jsem si mohl vyzkoušet všechny fáze vývoje hry od návrhu až po její vytvoření a testování. Rozšířil jsem si obzory, jak v samotném herním enginu Unity, tak i v programovacím jazyku C# a více jsem nahlédl pod pokličku, jak funguje komunikace po síti v počítačových hrách.

Na hře bych chtěl pracovat nadále a doladit chyby, na které nezbyl čas během bakalářské práce. Podle mého názoru je hlavním nedostatkem nedostatečná optimalizace síťové komunikace. Během vývoje jsem se sice snažil, aby se po síti odesílalo, co nejméně dat a snažil jsem se přenést většinu výkonu na zařízení koncového klienta, ale myslím si, že je stále co zlepšovat. Do hry bych chtěl také vytvořit více úrovní, aby byla hra více dynamická a hráče tak brzo neomrzela. V nových úrovních by byli nepřátelé ovládáni umělou inteligencí a byly by přidány různé pasti a nástrahy na hráče. Také plánuji rozšířit inventář o více typů bomb, které by opět hru více zpestřily.

S výslednou práci jsem nadmíru spokojen, i přestože jsem na začátku měl velké pochybnosti, zda takto rozsáhlou práci zvládnu v dostatečné kvalitě dokončit.

POUŽITÁ LITERATURA

[1] Wikipedia, the free encyclopedia. Bomberman. *Wikipedia, The Free Encyclopedia.* [online] 2018. [cit. 2018-02-25]. Dostupné z: https://en.wikipedia.org/wiki/Bomberman.

[2] Wikipedia, the free encyclopedia. Bomberman (1983 video game). *Wikipedia, The Free Encyclopedia.* [online] 2018. [cit. 2018-02-25].
Dostupné z: https://en.wikipedia.org/wiki/Bomberman_(1983_video_game).

[3] Wikipedia, the free encyclopedia. Bomberman (1990 video game). *Wikipedia, The Free Encyclopedia.* [online] 2018. [cit. 2018-02-25].
Dostupné z: https://en.wikipedia.org/w/index.php?title=Bomberman_(1990_video_game)&oldid=836180886.

[4] Wikipedia, the free encyclopedia. Unity (game engine). *Wikipedia, The Free Encyclopedia.* [online] 2018. [cit. 2018-02-25].
Dostupné z: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=832515206.

[5] Fine, Richard. UnityScript's long ride off into the sunset. *Unity Blog.* [online] 2017. [cit. 2018-02-25]. Dostupné z: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/.

[6] TRISTEM, Ben a Mike GEIG. *Unity Game Development in 24 Hours, Sams Teach Yourself.* 2nd edition. Indianapolis, United States: Pearson Education (US), 2015. ISBN 0672337517.

[7] Joseph, Hocking. *Unity in action: multiplatform game development in C#.* Shelter Island, NY : Manning Publications Co., 2015. ISBN 978-1617292323.

[8] Technologies, Unity. Unity – Manual: MonoBehaviour. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/ScriptReference/MonoBehaviour.html.

[9] Unity Technologies. Unity – Manual: Coroutines. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/Coroutines.html.

[10] Unity Technologies. Unity – Manual: WaitForEndOfFrame. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/ScriptReference/WaitForEndOfFrame.html.

[11] Unity Technologies. Unity – Manual: WaitForFixedUpdate. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/ScriptReference/WaitForFixedUpdate.html.

[12] Unity Technologies. Unity – Manual: WaitForSeconds. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/ScriptReference/WaitForSeconds.html.

[13] Unity Technologies. Unity – Manual: WaitForSecondsRealtime. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/ScriptReference/WaitForSecondsRealtime.html.

[14] MURRAY, Jeff. *C# Game Programming Cookbook for Unity 3D*. Natick, United States: Taylor & Francis, 2014. ISBN 1466581409.

[15] Unity Technologies. Unity – Manual: Layers. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/Layers.html.

[16] Unity Technologies. Unity – Manual: Tags and Layers. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/class-TagManager.html.

[17] Unity Technologies. Unity – Manual: Using the Transport Layer API. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/UNetUsingTransport.html.

[18] Abramychev, Alexey. All about the Unity networking transport layer. *Unity Blog.* [online] 2014. [cit. 2018-04-09] Dostupné z: https://blogs.unity3d.com/2014/06/11/all-about-the-unity-networking-transport-layer/.

[19] Unity Technologies. Unity – Manual: The High Level API. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/UNetUsingHLAPI.html.

[20] Unity Technologies. Unity – Manual: NetworkBehaviour. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/class-NetworkBehaviour.html.

[21] Unity Technologies. Unity – Manual: NetworkIdentity. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/class-NetworkIdentity.html.

[22] Unity Technologies. Unity – Manual: Object Spawning. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/UNetSpawning.html.

[23] Unity Technologies. Unity – Manual: State Synchronization. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/UNetStateSync.html.

[24] Unity Technologies. Unity – Manual: Remote Actions. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/UNetActions.html.

[25] Unity Technologies. Unity – Manual: Network Messages. *Unity Documentation.* [online] 2017. [cit. 2018-04-09].
Dostupné z: https://docs.unity3d.com/550/Documentation/Manual/UNetMessages.html.