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 V UNITY

INNOVATIVE GAME DEMO IN UNITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAEL SCHNEIDER**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Schneider Michael**
Program: Informační technologie
Název: **Inovativní herní demo v Unity**
Innovative Game Demo in Unity
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte engine Unity a grafické efekty.
2. Navrhněte inovativní herní mechaniky a interaktivní aplikaci, kde mechaniky demonstrujete.
3. Naimplementujte navrženou aplikaci a vytvořte sadu úrovní.
4. Zhodnoťte a proměřte.
5. Vytvořte demonstrační video.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 9. května 2019

Abstrakt

Cílem této práce je vytvoření inovativního herního dema v herním enginu Unity. V práci lze najít popis nejdůležitějších systémů tohoto enginu. Zároveň je zde popsán vývoj návrhu inovativní mechaniky. Ten je realizován v podobě dema, jehož implementace je zde také rozebrána. Součástí dema je editor, který dovoluje vytvořit další úrovně. Inovativní koncept spočívá ve využití přeměn energie k úspěšnému vyřešení úrovní.

Abstract

The goal of this thesis is a development of innovative game demo in the Unity engine. In the thesis is description of the most important systems of this engine. At the same time, the development of innovative concept is described. It is implemented as a demo and implementation of it is also discussed here. Part of the demo is editor, which allows to create new levels. The innovative concept of demo is to use energy transformations to successfully solve levels.

Klíčová slova

Unity, herní engine, hra, demo, inovativní koncept, přeměna energie

Keywords

Unity, game engine, game, demo, innovative concept, conversion of energy

Citace

SCHNEIDER, Michael. *Inovativní herní demo v Unity.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Inovativní herní demo v Unity

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.

.....
Michael Schneider
13. května 2019

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Tomaši Miletovi za odborné vedení mé práce, cenné rady, podnětné připomínky a čas, kterým přispěl k vypracování této práce. Děkuji rovněž své rodině za podporu a obzvláště bratrovi Danielu Schneiderovi za rady a inspiraci.

Obsah

Kapitola 1

Úvod

Člověk jako takový je hravý už od malička. Začíná si hrát v momentě, kdy začne vnímat a objevovat svět kolem sebe. A ať už si to někdo přizná, nebo ne, různé druhy her lidstvo provázejí celým životem. Lidé hrají hry, aby se potěšili, aby měli před sebou výzvu, či jen aby si odpočinuli. Druhů her lze najít bezpočet. S vynálezem počítače vznikl nový druh a to počítačové hry. Zpočátku velmi jednoduché, ale s přibývajícím časem a také výkonem počítačů se vydávaly stále hezčí a propracovanější tituly.

Lidé si osvojili koncept, že ve hrách mohou být na chvíli někým jiným a řešit diametrálně odlišné problémy, než se kterými se denně setkávají. Hry už svojí podstatou vyvolávají emoce a lze je dělit na dvě kategorie. Hry, které se snaží apelovat na negativní emoce jako násilí, strach, úzkost a hry, které cílí na pozitivní emoce jako je radost z budování, či dosažení úspěchu vyřešením problému. Avšak těch je menšina. Proto tato práce popisuje tvorbu hry z pozitivnější kategorie.

Počítačových her se denně vydávají desítky. Herní studia platí velmi početné programátorské týmy a investují do her nemalé peníze. Proto se snaží zaujmout žánry her, které se osvědčily, čímž neriskují ztrátu. Z tohoto důvodu, když se chce prosadit menší vývojář, popřípadě menší herní studio, musí přijít na trh s inovací. Tato práce nastiňuje takovou inovativní hru.

Rozvíjí koncept přeměny energie. Energii lze nalézt všude kolem nás a v podstatě i samotná hmota je jen energie v určité podobě. Proto inovace, kterou hra přináší, spočívá v přeměnách energie, která je dostupná k tomu, abychom prošli úrovní. Jelikož se jedná o demo, je k dispozici několik ukázkových úrovní. Avšak během vývoje vznikla pořeba vytvořit nástroj na jednotlivou tvorbu urovní a proto se součástí tohoto dema stal i uživatelsky přívětivý editor. Tento editor zároveň umožňuje se podívat na hru z pohledu návrháře úrovní a vytvořit tak vlastní úrovně, čímž dává prostor k rozvoji kreativity a rozšiřuje celkovou hratelnost dema.

Mě samotného k programování přivedly hry. Už od doby, kdy jsme měli doma první počítač, jsem chtěl vytvořit vlastní hru. Zjišťoval jsem, jak počítač funguje a rozšiřoval svoje znalosti, až se mi na základní škole podařilo napsat první řádek kódu. Tehdy jsem byl náležitě hrdý, když konzole vypsala „Ahoj světe!". Od té doby jsem ušel kus cesty, ale hry programuji ve svém volném čase. Většina her, co vytvořím, skončí doslova uschovaná v mém šuplíku. Tak bylo na čase nějakou vytvořit a stát si za ní a odprezentovat ji. A není vhodnější čas než bakalářská práce.

Cílem této bakalářské práci je vytvořit hru, avšak takovou, která by mohla v dnešní době zaujmout a nebyla jen jedna z tuctu dalších. Nastínit, jak v dnešní době může vývojář začít. Následně objasnit tvorbu hry v enginu, který jsem si pro svoji hru vybral, a to konkrétně

herní engine Unity. Objasnit,jak pracovat s jednotlivými systémy enginu a jak tvořit hru od založení projektu až po první sestavení hry a spuštění.

Ve 2. kapitole je stručně shrnuta historie her, následně vysvělen herní engine, porovnání herních enginů a podrobnější informace o Unity, včetně vysvětlení základních systémů, které Unity poskytuje.

Ve 3. kapitole se nachází návrh inovativní mechaniky a vysvětlení její podstaty.

Ve 4. kapitole je shrnuta implementace inovativního dema.

Kapitola 5. obsahuje stručný náhled na optimalizace.

Kapitola 6. je následné ohlédnutí za touto prací.

Kapitola 2

Teorie

První počítačové hry, které vznikly, byly v podstatě zcela tvořeny samotným hardwarem. První až skutečně softawarově vytvořená hra se nazývala Spacewar. Vytvořil ji Steve Russell a byla to hra pro dva hráče. Její vývoj trvral 6 měsíců a ovládala se vestavěným ovladačem počítače PDP-1 (Programmed Data Processor-1). Hra obsahovala dvě vesmírné lodě, které proti sobě bojovaly v asteroidovém poli. Prototyp byl vytvořen v roce 1961 a finální verze pak v roce 1962. Jelikož počítačů PDP-1 bylo celkově prodáno 55. Hra byla nakopírována jen do několika dalších na univerzitách. [5]

První komerčně dostupnou hrou se stala až hra Computer Space [2]. Vycházela ze hry Spacewar a díky této hře vzniklo studio Atari. Toto studio vytvořilo jednu z nejznámějších her a to Pong od Allana Alcorna. Ten byl zpočátku najat na tvorbu úplně jíné hry, avšak tato hra byla natolik chytlavá, že byla nakonec vydána v roce 1972.[7]

V roce 1972 také byla vydána první herní konzole. Nesla název Magnavox Odyssey. Dodávala se s šesti hrami a dvěma ovladači, avšak tehdy ještě kvádrovými. Tato konzole byla velmi uspěšná a společně se hrou Pong odstartovaly novou éru herního průmyslu.

Od té doby se vyvíjely hry stále složitější a s hezčí grafikou. V 90. letech se pak začala přisuzovat důležitá role příběhům jednotlivých her.

Po roce 2000 je výkon počítačů dostatečný i pro vykreslování fotorealistické grafiky.

Nyní se herní průmysl nachází na přelomu, kdy se hry začínají vyvýjet pro virtuální realitu, což bude bezpochyby dalším vývojovým článkem počítačových her jako takových.

Co je to herní engine? Pojem herní engine vznikl kolem 90 let. Vznikl v souvislosti s potřebou mít možnost modifikovat hru. Konkrétně v 90. letech. Jednou z takových byla velmi populární hra Doom. Herní engine v podstatě nabízí jednotlivé systémy, které jsou zapotřebí k vytvoření hry, pro kterou vznikl. Tato skutečnost otevírá příležitost pro vývojáře modifikovat stávající hru, či v podstatě na stejném základě vytvořit zcela novou. Postupem vývoje vznikly i univerzálnější herní enginy, které se nezaměřují na jednu konkrétní hru. Slouží jako nástroje s mnoha předpřipravenými systémy pro sestavení jakékoli hry.[4]

Pokud hru vyvýjí etablované herní studio, pravděpodobně již má svuj herní engine a celý tým, který na něm pracuje. Tyto enginy jsou většinou veřejnosti nepřístupné.

Pokud tedy jde o začátek vývoje hry, je třeba se rozhodnout, zda začít s vývojem vlastního enginu, či použít nějaký z existujících. Vývoj vlastního enginu je velmi časově náročný (v řádech let) a navíc je třeba velké množství znalostí z dané problematiky.

Proto v dnešní době existuje poměrně mnoho uživatelsky přívětivých možností, jak začít s vývojem hry. Úplně nejjednodušší jsou herní engines [9] bez potřeby znalosti programovaní. Mezi takové se například řadí Gamemaker. Tento engine je velmi jednoduchý a díky tomu lze rychle vytvořit kompletní 2D hru. Je to ovšem za cenu použití předem daných

komponent, které nelze upravovat (vyjma jejich argumentů). Pro pokročilejší uživatele je tu možnost naprogramovat vlastní komponenty. Nutno podotknout, že je stále třeba plně využívat vestavěných systémů Gamemakeru, které jsou primárně uzpůsobené pro 2D prostředí. Pokud je již tvůrce mírně pokročilý, existují mnohem efektivnější a komplexnější enginy, jako jsou Unity a Unreal. Přechodem k těmto enginům lze pohodlně vyvíjet ve 3D. Samotný přechod od Gamemakeru k těmto enginům je poměrně přímočarý a intuitivní.

Tyto enginy (Unity, Unreal) jsou v dnešní době dva nejlepší herní enginy pro malé vývojáře her. Dříve bylo nahlíženo na Unreal jako engine pro vývoj AAA titulů (herní tituly s s vysokám rozpočtem na samotný vývoj i následný marketing hry, vývojařsky tým se skládá z desítek programátorů a náklady na takovouto hru jsou v milionech dolarů) a Unity zaostávalo. V poslední době se však Unity poměrně vyrovnalo Unrealu a jsou si téměř rovny. Unity vždy více cílí na jednoduchost svého rozhraní a teprve, když je potřeba, nabízí komplexnější nástroje. V Unrealu je mnohem těžší se zorientovat a klade mnohem vetší počáteční nároky na uživatele. Unity si zvolilo naopak cestu inkrementálního růstu konfidence vývojáře.

2.1 Unity

Po zvážení cílů této bakalářské práce jsem si vybral herní Engine Unity. Je to multiplatformní herní engine s podporou jak 2D tak 3D. Nabízí rychlou a efektivní práci a velmi přívětivé nástroje. Jeho další předností je vestavěný physic engine PhysX od NVIDIA, který umožnuje vysoce reálnou fyziku herních objektů. Díky vestavěným šablonám lze hned od počátku směřovat vývoj, ať již potřebujeme vysokou optimalizaci(především šablona Lightweight Render) a nebo pro vizuálně propracovanou grafiku aplikací (High Definition Render) [1]. Unity lze také rozšiřovat pomocí balíčků, ať už assetů nebo nástrojů z asetstoru[11]. Unity má poměrně rozsáhlou komunitu. Proto lze naleznout na AssetStoru mnoho výborných nástrojů, i když zpravidla ty kvalitní jsou placené. Unity je zdarma pro osobní použití nebo pro firmy, avšak jen do obratu 100 000 USD za jeden fiskální rok[12]. Pro vývoj se používá jazyk C#, avšak dříve byl zároveň plně podporován UnityScrip, který byl odvozen od JavaScriptu. UnityScript byl postupně odstraňován od roku 2017, především kvůli malému % využití [3]. Unity podporuje vývojáře vytvářením výukových serií a velmi kvalitní přehlednou dokumentací. Díky dobré komunitě vzniká nepřeberné množství video návodů.

Nyní následuje popis systémů Unity, které jsou nezbytnou součástí implementovaného dema. Samotná implementace dema je pak popsána v kapitole 4.

2.2 Šablony

Po spuštěn unity se začíná fází zakládání nového projektu. V této fázi se vybere umístění, kam se bude projekt ukládat. Zároveň je nutné správně zvolit šablonu (Template), ve které se bude v Unity tvořit. Unity v poslední době testuje možnost si na začátku vývoje vybrat, zda se bude cílit na estetičnost grafického provedení samotné hry, nebo zda se bude cílit na optimalizaci výkonu a tím slabší zařízení jako jsou mobilní telefony, tablety, či dokonce webové prostředí. Unity má proto předpřipravených 6 počátcích šablon(nastavení) enginu.

- 2D – Šablona nastavuje výchozí ortografickou projekci kamery, 2D světla a obsahuje přidané nástroje např. Sprite Packer [13]

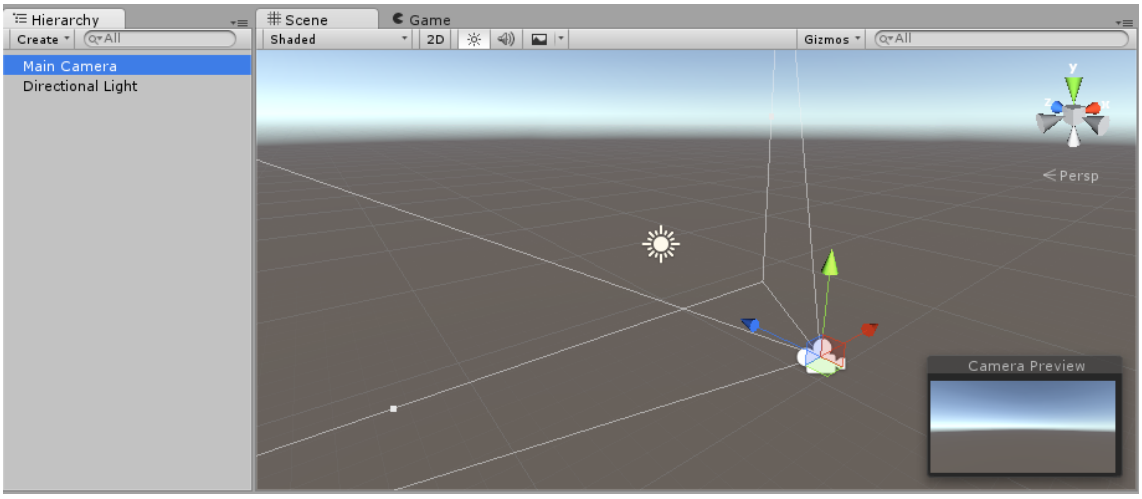
- 3D – Standardní vestavěné prostředí se standardními systémy a výchozím vykreslováním.
- 3D With Extras – Obsahuje více ukázkových příkladů, které lze použít pro rychlé fáze počátečního vývoje.
- High Definition RP – Je první z nových šablon. Tato šablona cílí na velmi výkonné platformy a obsahuje novější shader model 5.0. Má také vestavěnou skriptovatelnou redner pipeline, včetně nových možností postprocesingu. Je tak vhodnou volbou pro aplikace vyžadující velmi kvalitní grafické zpracování.
- Lightweight RP – Cílí na co nejvyšší výkon aplikace (nejmenší nároky na hardware), včetně podpory mobilních zařízení. Proto některé druhy grafických efektů nepodporuje (například využívá zapečení světla do textur) a zároveň zjednodušuje tvoření vykreslovacích řetězců, a to díky jednouchému editoru. Umožňuje tak rychlejší vykreslováni, avšak za cenu vizuální podoby.
- VR Lightweight RP – Je šablona, která cílí na vývoj pro virtuální realitu. Svým nastavením vychází ze šablony Lightweight, ale s větším důrazem na podporu a integraci vývojových SDK pro virtuální realitu.

Po vybrání optimální šablony se založí projekt a zpravidla se zkompiluje vzorový projekt pro danou šablonu (každá šablona má odlišný, aby vynikly její vlastnosti).

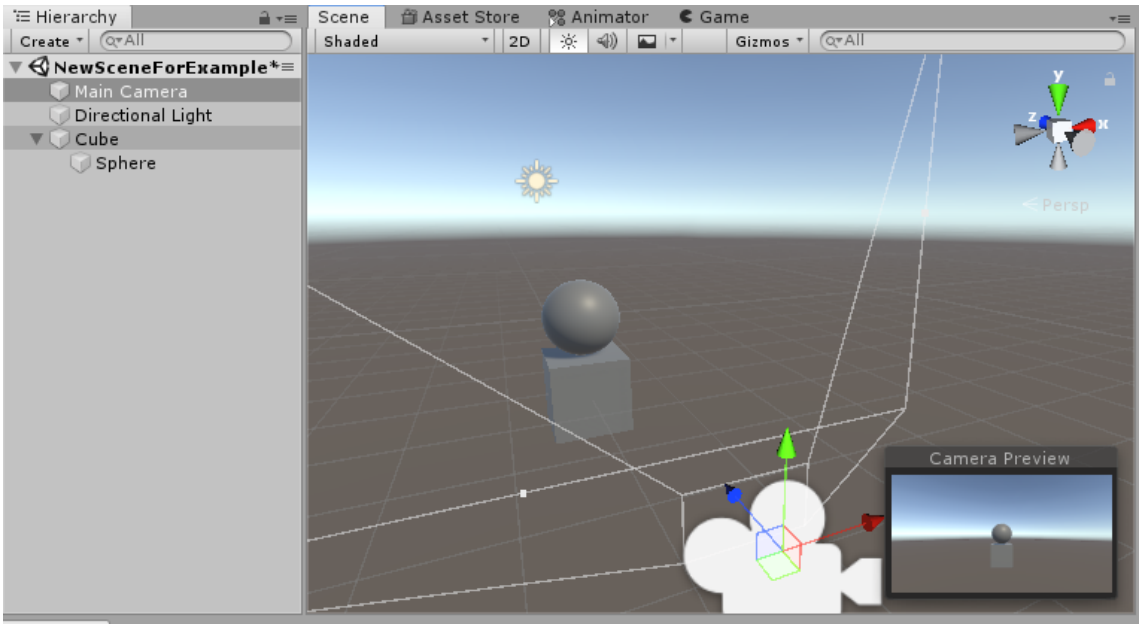
2.3 Scény

Základním prostorem pro tvorbu obsahu je scéna. Je to takzvané prostředí hry. Pro nejjednodušší představu si lze jednotlivé scény představit jako úrovně hry. Scény jako takové, jsou tedy základním stavebním kamenem celé hry. Po vytvoření scény (viz 2.1) v editoru se v ní nachází dvojce základních objektů, a to hlavní kamera(Main Camera) a světlo (Directional Light). Samotnou místnost pak lze přepínat mezi 2D a 3D režimem. Přepnutí do 2D se hodí například pro vytváření menu. Posléze lze do místnosti vkládat objekty.

Objekt má vždy svoje jméno, tag, a minimálně základní komponentu Transform, která obsahuje pozici rotaci a měřítko. Pokud již objekt nemá žádnou jinou komponentu, nazývá se takzvaně prázdným objektem. Takovéto objekty primárně slouží jako organizační objekty, či objekty, vůči kterým se umísťují ostatní podřízené objekty. Hierarchie objektů tvoří strom, kdy samotná scéna je kořenový uzel a objekty pod ní mohou obsahovat další objekty.



Obrázek 2.1: Zdroj: Scenes,[13]
Takto vypadá nově vytvořená scéna, vlevo lze vidět v hiearchii objekty, které se nacházejí ve scéně a vpravo pak náhled do místnosti z editoru, včetně malého náhledu kamery. Tato místnost stanovuje základní prostor, ve kterém se hra tvoří. Obsahuje i pozadí ve formě skyboxu.

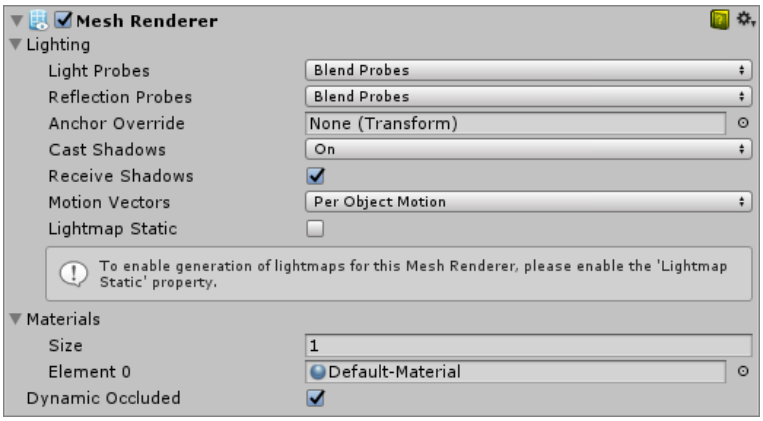


Obrázek 2.2: Na obrázku lze vidět vytvořený objekt Box a pod ním přiřazený objekt Sphere. Oba objekty lze vidět shodně jak v náhledu scény, tak i na pohledu kamery do scény.

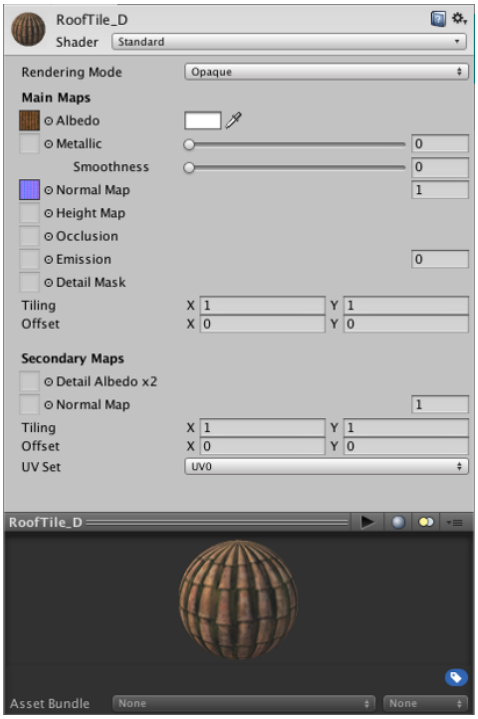
Jak lze vidět na obrázku 2.2, kamera nerozlišuje hloubku zanoření v hierarchii objektů, ale zobrazí se všechny, které obsahují komponentu Mesh Renderer 2.3.

Pro vysvětlení komponenty meshrenderer, je dobré zmínit, co je to Mesh. Mesh jako takový je geometrie objektu skládající se z polygonů. Jelikož Unity samotné neobsahuje žádný editor geometrií, je nutné geometrie vytvářet v nástrojích pro to určené. V případě této práce v Blendru.

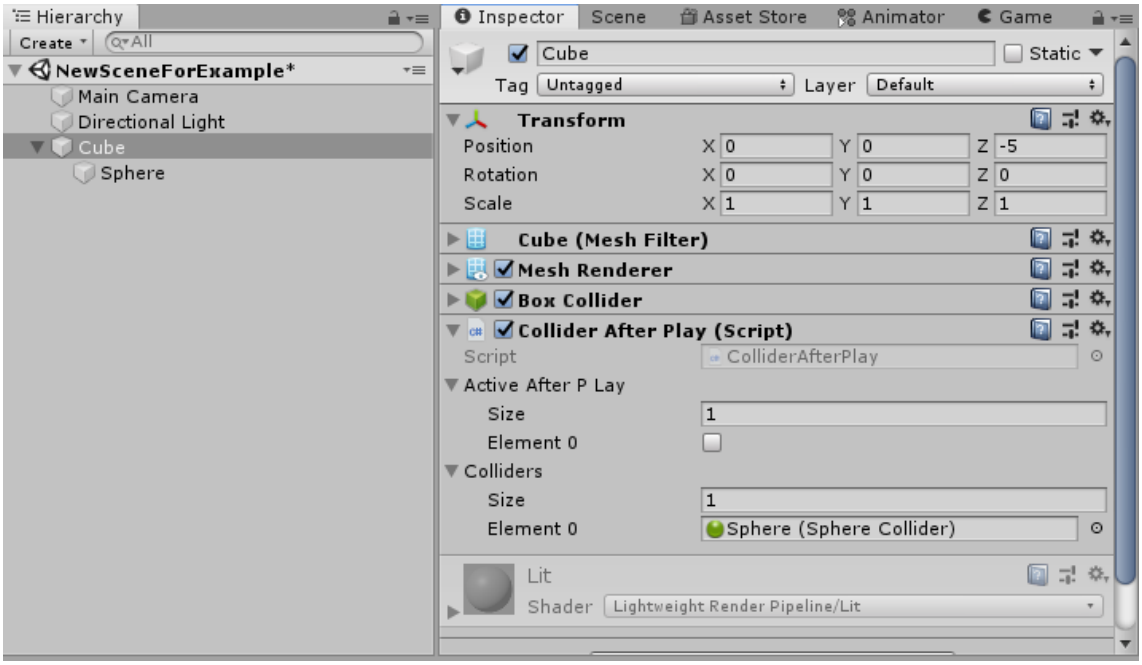
Mesh nebo-li geometrie určuje, jak bude objekt tvarován. Materiál pak určuje, jak bude objekt vypadat viz 2.4. Parametry materiálu se určují dle použitého shaderu. V projektu je nastaven standardní shader pro Lightweight šablonu. Jelikož je shader dostatečně flexibilní, nebylo ve finálním projektu potřeba vyvářet vlastní. Nicméně v jednom z prototypu, bylo třeba vytvořit například materiál černé díry a tam se již bez vlastního shaderu nelze obejít. Naštěstí Unity poskytuje s Lighweight šablonou jednoduchý nastroj shader Graph, ve kterém se skládají potřebné efekty v jednoduchém editoru s náhledem.



Obrázek 2.3: Komponenta Mesh Renderer Zdroj: [13], Mesh Renderer. Lze nastavit interakce osvětlení s daným objektem a zároveň lze objektu přidat jeden či více materiálů.



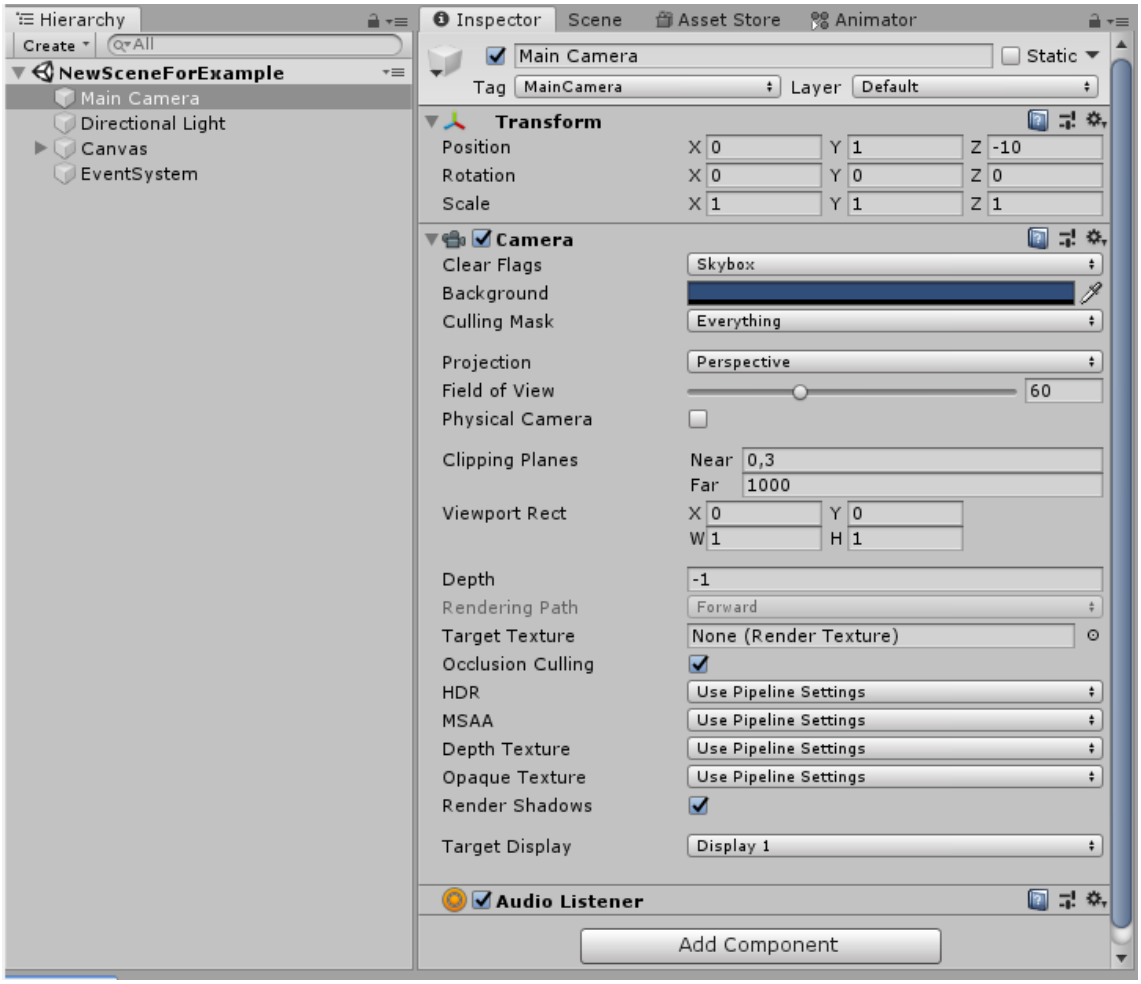
Obrázek 2.4: Materiál Zdroj: [13], Material. Jeden ze základních parametrů materiálů je shader, kterým se materiál bude vykreslovat. Materálu lze nastavit poměrně dost vlastností. Z těch nejpoužívaněších rozhodně albedo společně s normálovou mapou. Pokud je zapotřebí, aby materiál vyzařoval světlo, tak se nastavuje emise.



Obrázek 2.5: Ukázka inspektoru. Vlevo je vybrán jeden konkrétní objekt (Cube), pro který jsou v inspektoru ukázány jednotlivé komponenty.

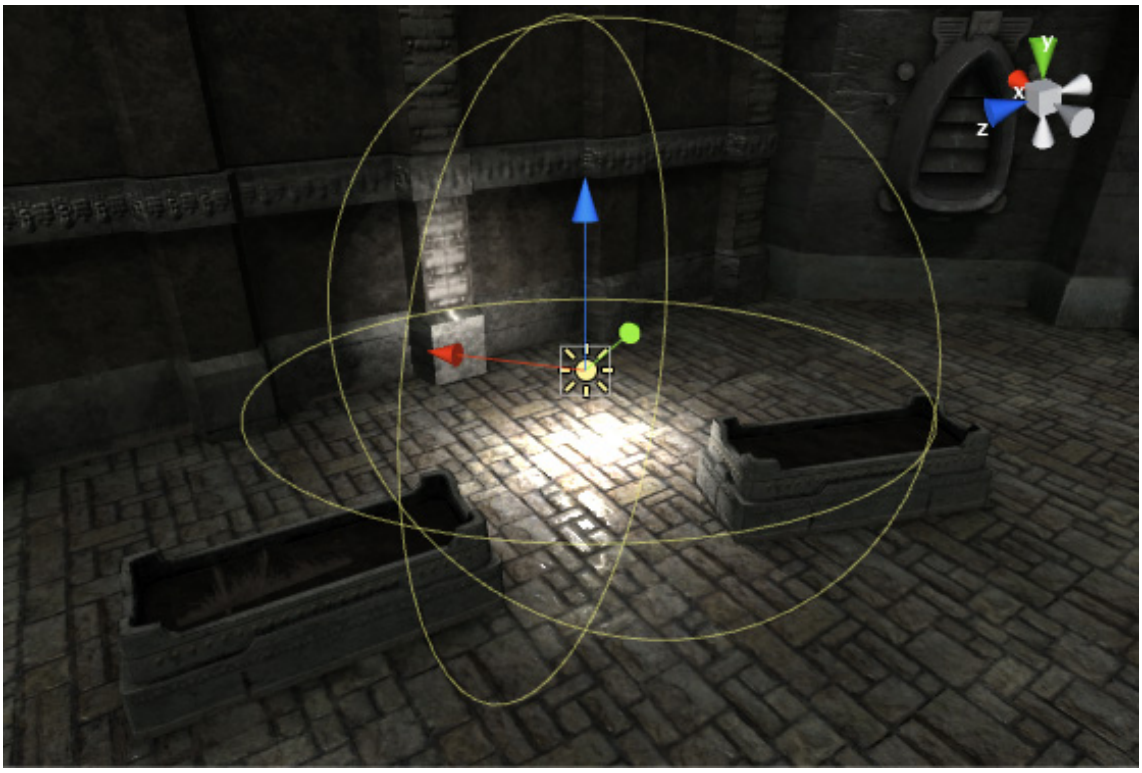
Předpřipravených komponent lze nalézt v Unity velké množství, avšak během vývoje je zapotřebí si vytvářet svoje vlastní v podobě skriptů, které se takto přiřadí k jednotlivým objektům a každý objekt s komponentou si po spuštění vytvoří vlastní instanci dané komponenty viz 2.5.

Nyní tedy ke dvěma objektům, již se scénou vygenerovaných. MainCamera je objekt, který obsahuje komponentu Camera. Tato komponenta vytváří pohled do místnosti (to, co hráč ve výsledku uvidí). Komponenta Camera nabízí i renderování do textury. Toho se využívá při tvoření minimap. Rovněž lze kameře přidat masku, pro výběr oblasti, kterou zachycuje. Viz parametry komponenty 2.6.



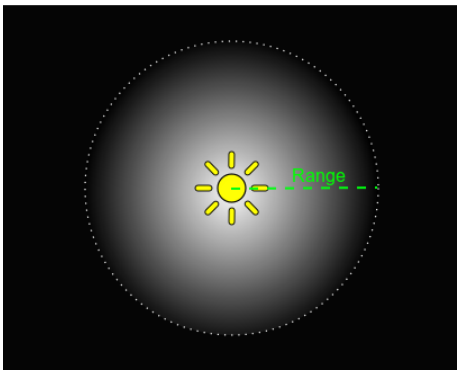
Obrázek 2.6: Komponenta Objektu Camera. Nastavuje se, zda se budou objekty zobrazovat perspektivně, či ortograficky. Následně se nastavuje zorné pole (Field of View) ve stupních a v neposlední řadě se může hodit nastavení pozadí kamery (využito při tvoření scény s menu). Pak samozřejmě ořezové roviny a jak již bylo zmíněno, lze zvolit režim vykreslení obrazu do textury.

2.4 Světla

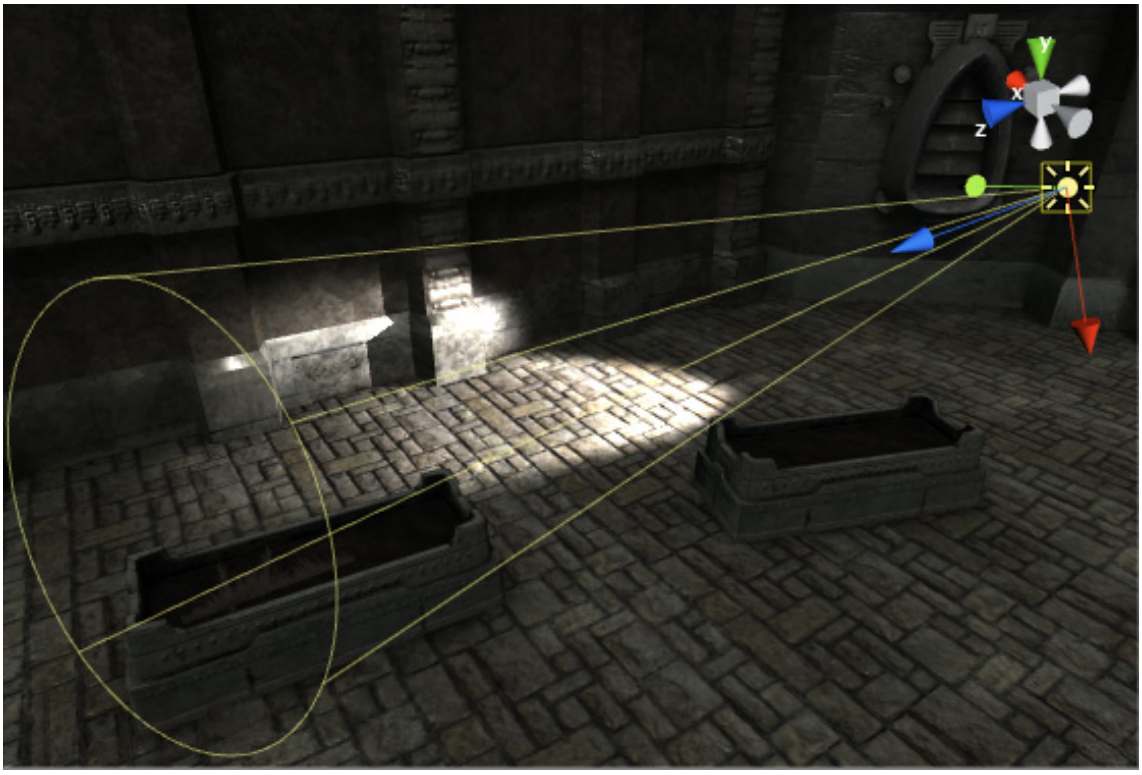


Obrázek 2.7: Bodové světlo, Zdroj:[13], Lights. Vizualizace světla ve scéně, Unity vykresluje oblast, ve které již bude nulová intenzita světla.

Bodové světlo (Point light) září z jednoho bodu do všech stran. K nastavení světla se používají dva základní parametry, a to rádius a intenzita světla viz 2.8. Rádius určuje vzdálenost, na kterou světlo dosvítí a v této vzdálenosti je světlo již nulové viz 2.7. Intenzita světla vyzařovaného do prostoru klesá exponenciálně podobně, jako světla v reálném světě. Tato světla se často používají jako standardní osvětlení, například lampy, lustry atd. Najde se však i využití například pro explozi, kdy je třeba ozářit veškerý okolní prostor.

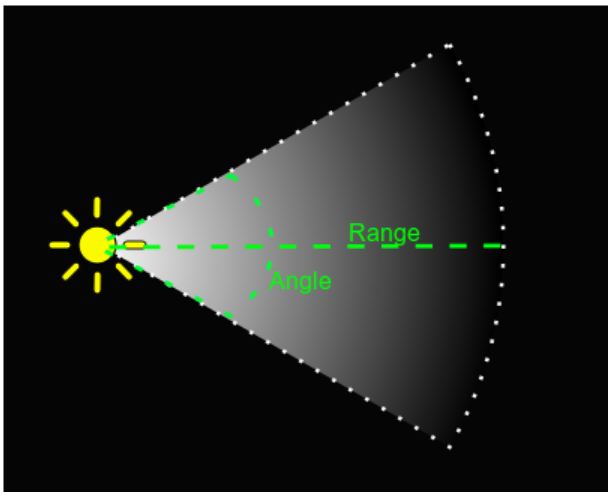


Obrázek 2.8: Parametry bodového světla vzdálenost a samotná intenzita, Zdroj:[13], Lights

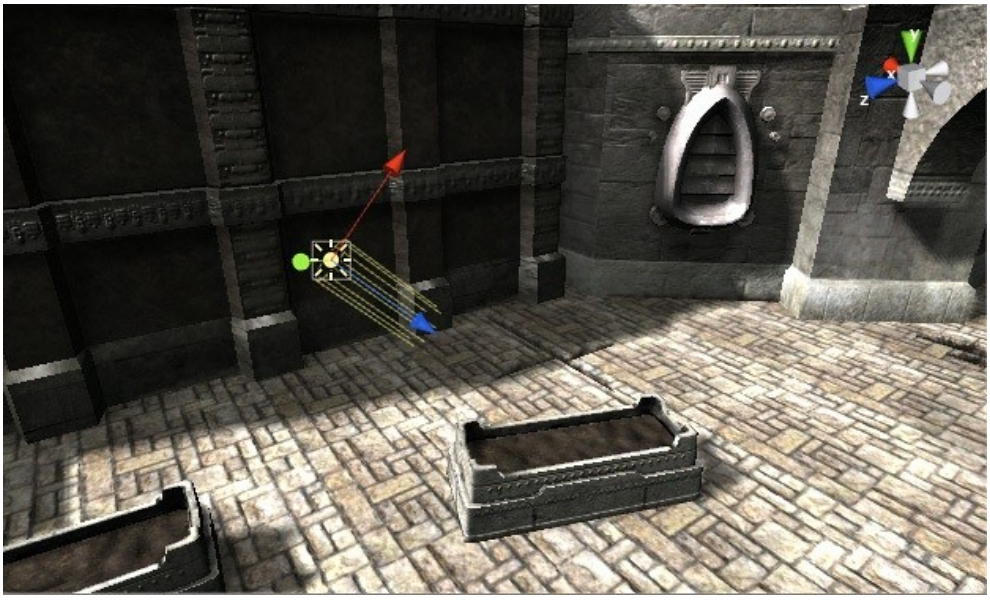


Obrázek 2.9: Vizualizace světelného kuželu ve scéně, Zdroj:[13], Lights

Světelný kužel je velmi podobné světlo světlu bodovému viz 2.9. Na rozdíl od bodového světla není všesměrové a jako parametr se zvolí úhel vyzařovaného světla (parametry 2.10). Vzniká tím kužel světla. Tento typ světla se používá nejčastěji na pracovní světla, světla automobilu a zpravidla na svítilny nejrůznějšího druhu vytvořené člověkem.

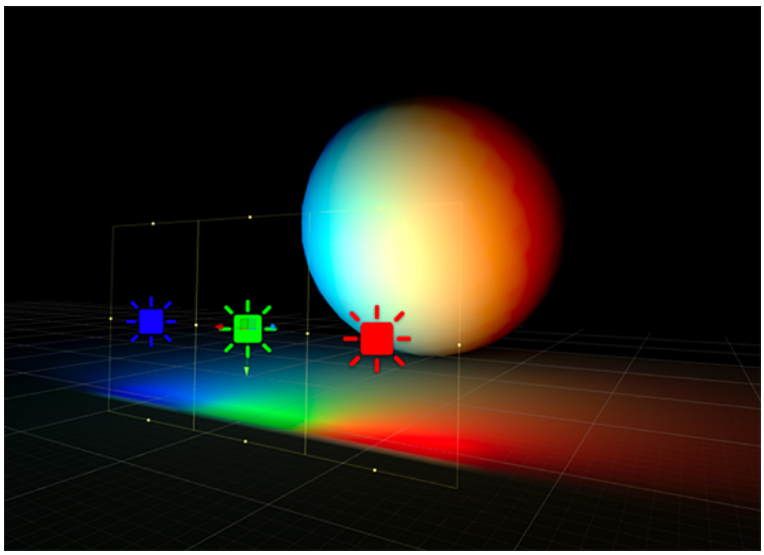


Obrázek 2.10: U světelného kužele se nastavují tři parametry intenzita, úhel a vzdálenost, Zdroj:[13], Lights



Obrázek 2.11: Stejnosměrné světlo vytvářející měkké stíny, Zdroj:[13], Lights

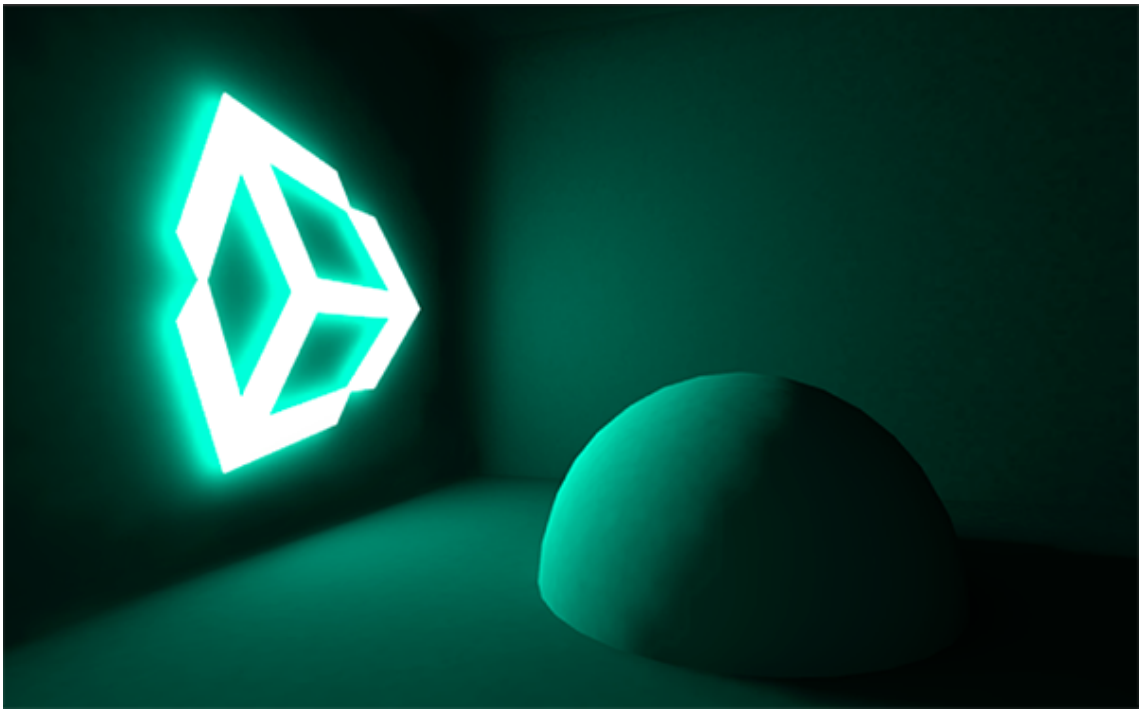
Stejnosměrné světlo se nachází ve výchozím stavu ve scéně. Jedná se o stejnosměrné světlo reprezentující velmi vzdálený zdroj světla, například slunce. Tento druh světla by měl na objektech tvořit měkké stíny viz 2.11. To je však záležitost vykreslování a také záleží na zvolené šabloně. Světlu jako parametru lze zvolit směr, ze kterého přichází. Pokud bude tímto směrem rotovat, lze tím vytvořit simulaci dne a noci. K vytvoření denního cyklu existuje v Unity nastavení skyboxu (Implicitní pozadí scén), které reflektuje právě změnu stejnosměrného světla.



Obrázek 2.12: Světlo oblasti, Zdroj:[13], Lights. Na obrázku lze vidět tři světla. Svítí rovnoměrně pouze na jednu stranu jejich oblasti (žluté ohraničení).

Světlo oblasti je vyzařováno z obdélníkové oblasti, a to pouze jedné strany takovéhoto obdélníku viz 2.12. Nelze určovat vzdálenost dosvitu, pouze lze nastavit jeho intenzitu. Na

základě intenzity světla se dopočítá dosvit. Výpočet iluminace tohoto druhu světla je značně náročný. Proto nelze toto světlo vypočítávat za běhu aplikace a je tedy nutné jej zapéct do textur.

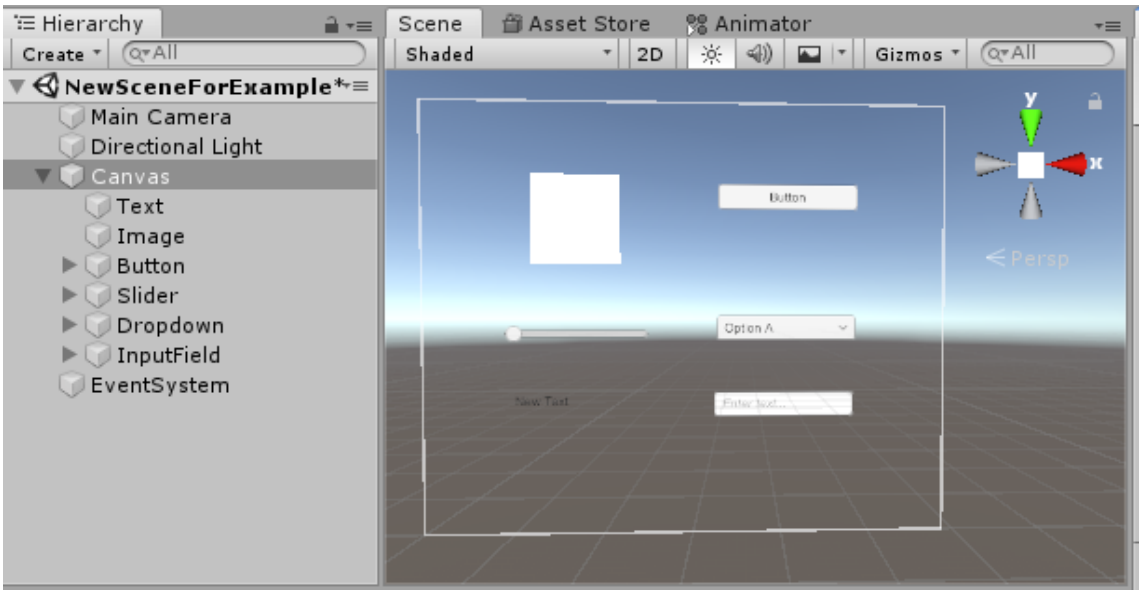


Obrázek 2.13: Zářivý materiál, Zdroj:[13], Lights Komponentě Mesh renderer lze přiřadit materiál, který má aktivovanou vlastnost Emission. Nastaví se barva a výsledný objekt následně osvětluje prostor.

Materiály vyzařující světlo se chovají podobně jako světla oblasti, avšak lze je nastavit pro libovolný materiál a tím pádem na jakýkoli tvar objektu. Objekt tedy ve scéně vyzařuje světlo viz 2.13.

Posledním typem světla, který se v Unity využívá je ambientní osvětlení. Ambientní, nebo-li všudypřítomné, je druh světla, který ovlivňuje celkovou světlelnost scény. Je připočítané jako statická hodnota při výpočtu osvětlení scény.

2.5 Plátno

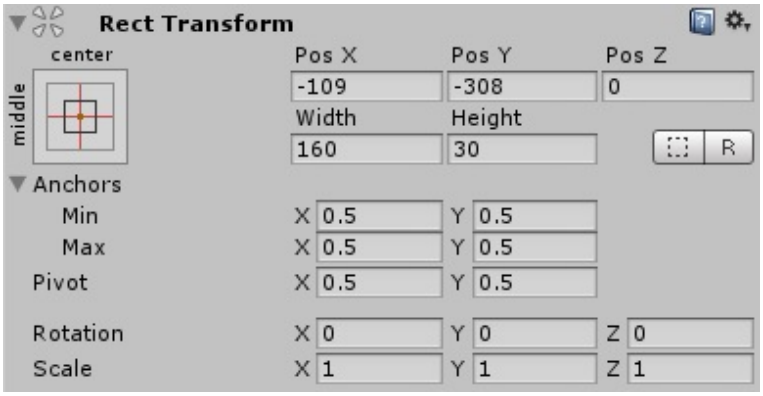


Obrázek 2.14: Na obrázku lze vidět plátno a základní UI elemnty, které Unity poskytuje. Každý UI objekt obsahuje specifickou komponentu, krerá definuje jeho chování.

Zpravidla ve hře je třeba vytvořit uživatelské rozhraní takzvané UI. Prvky UI jsou také objekty, avšak a musejí se nacházet na takzvaném plátně (Canvasu) viz 2.14. Plátno je tedy objekt, jehož potomci (objekty) jsou UI Objekty. Unity obsahuje všechny základní grafické prvky, které ve většině případů stačí. Pokud budou potřeba nějaké nestandardní UI prvky, lze si každý prvek individuálně zcela upravit, či použít nové z balíků na AssetStore. Plátno se v Unity editoru zobrazuje jako obdélník, přímo ve scéně. Lze proto upravovat umístění stejným způsobem, jako pohybujeme všemi objekty v Unity. Plátnu lze nastavit mód vykreslování, kterým určíme, zda se bude vykreslovat v rámci prostoru obrazovky (screen space) či přímo ve scéně (word space).

- Screen Space Overlay UI objekty se renderují přes objekty scény. Pokud obrazovka změní rozlišení plátno se automaticky přizpůsobí.
- Screen Space Camera Velmi podobné jak předcházející mód ale plátno je umístěno v předem definované vzdálenosti od kamery.
- World Space Plátno je umístěno ve scéně, stejně jako ostatní objekty. Většinou se toto nastavení používá pro displeje přímo ve scéně.

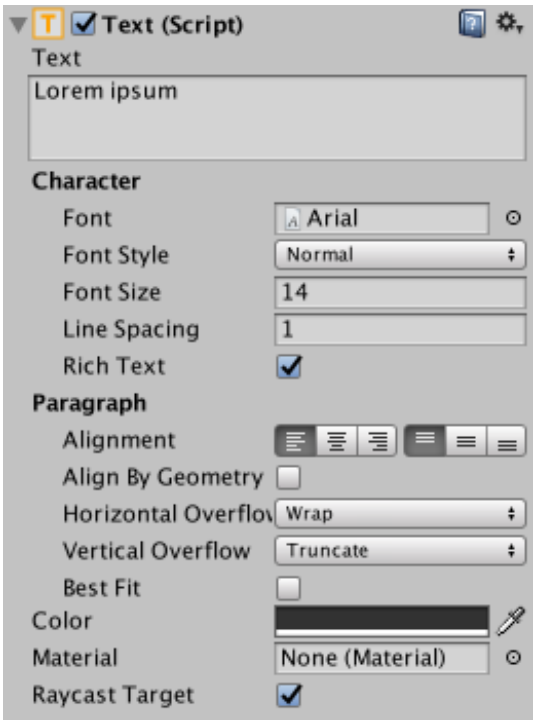
Každý UI prvek obsahuje namísto základní komponenty Transform komponentu Rect Transform 2.15. Tato komponenta zajišťuje pozicování prvku na plátně a oproti komponentě Transofrm umožňuje například relativní pozicování vůči proměnlivému plátnu.



Obrázek 2.15: Komponenta Rect Transform Zdroj: [13], Basic Layout Mezi základní nastavení Rect Transform komponenty patří to, vůči jakému bodu plátna se bude objekt pozicovat, jaká bude tato relativní pozice a samotné rozměry objektu (výška, šířka). Případně lze nastavit rotaci a měřítko.

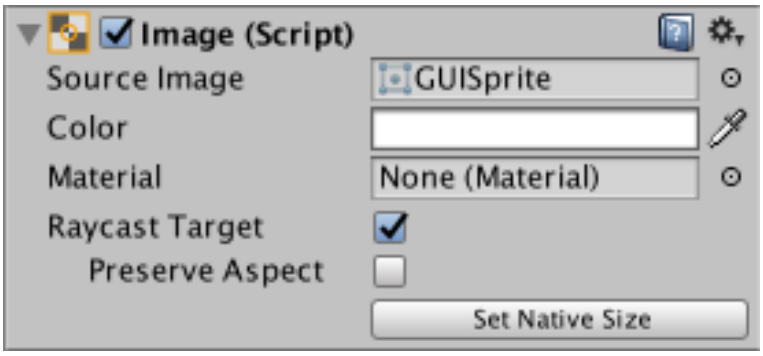
Základní vizuální objekty UI:

- Text Komponenta pro vypsaní textu. Dovoluje základní editaci písma jako je nastaveni fontu velikosti, odsazování a zarovnání odstavce, popřípadě nastavení barvy Textu viz 2.16. V dnešní době je spíše na ústupu a nahrazuje ji komponenta TextMeshPro, která se stane součástí Unity. Nyní se však musí stále ručně importovat.



Obrázek 2.16: Ukázka komponenty Text Zdroj: [13], Visual Components. V této komponentě se nastavuje text a vše k němu potřebné: zarovnání (vertikalní i horizontálni), font, velikost, zalamování, ořez.

- Image Komponenta pro dosazení obrázku viz 2.17.



Obrázek 2.17: Ukázka komponenty Image Zdroj: [13], Visual Components. Lze nastavit základní parametry, jako zdroj obrázku, barevnost včetně průhlednosti, popřípadě materiál.

- Button Komponenta typu tlačítko nabízí základní nastavení jako volbu pozadí, barevnost, barevnost při najedí na tlačítko myší a pak také přímo v této komponente lze reagovat na událost kliknutím na toto tlačítko.
- Dropdown Komponenta pro vytvoření rozbalovacího menu je velmi podobná tlačítku.
- Slider Komponenta slider vytvoří posuvník. Alternativou této komponenty je pak komponenta Scrollbar. Tyto komponenty jsou si velmi podobné a vrací normalizovanou hodnotu mezi 0 a 1 dle polohy posuvníku.
- Input Field Komponenta vstupního pole. Přijímá vstup od uživatele. Přímo v komponentě lze nastavit, jaké znaky jsou přípustné, popřípadě jaká je maximální délka očekáváného řetězce. Opět lze měnit barvu, pozadí. Zároveň objekt, na kterém je tato komponenta, má dva synovské objekty, na kterých najdeme komponentu text. Lze je využít k nastavení fontu a vzhledu písma, které uživatel bude zadávat a druhý na zadání textu, který bude v poli předvyplněn jako nápověda.

Kromě vizuálních komponent, také najdeme automatické pořadače skupin vizuálních prvků. Jedná se o systém Auto Layout, který poskytuje potřebné komponenty:

- Content Size Fitter Komponenta zajišťuje velikost dceřiných prvků objektu, ke kterému je přiřazena. Jedná se o vertikální, či horizontální předdefinované rozměry. Jde však pouze o preferované rozměry, tzn. nemusejí se uplatnit vždy, záleží na poskytnutém prostoru.
- Layout Element Touto komponentou označujeme objekty, které budou řazeny definovaným layoutem. V komponentě lze nastavit, jak pevné rozměry, tak i flexibilní, aby se objekt správně přizpůsoboval velikosti okolního prostoru.
- Horizontal Layout Group Skládá prvky vedle sebe do řádku. Lze nastavit směr, od kterého skládá. Samozřejmě se nastavuje odsazení prvku od sebe a lze i vynutit, aby prvky byly roztaženy k vyplnění maximálního prostoru rodiče.
- Vertical Layout Group Obdobně jako horizontální layout s tím, že skládá objekty pod sebe. Byl využít v menu pro uspořádaní tlačítek.

- Grid Layout Group Tento layout je kombinací předchozích dvou komponent. Byl využit pro zarovnání ikon v editoru.

2.6 Animace

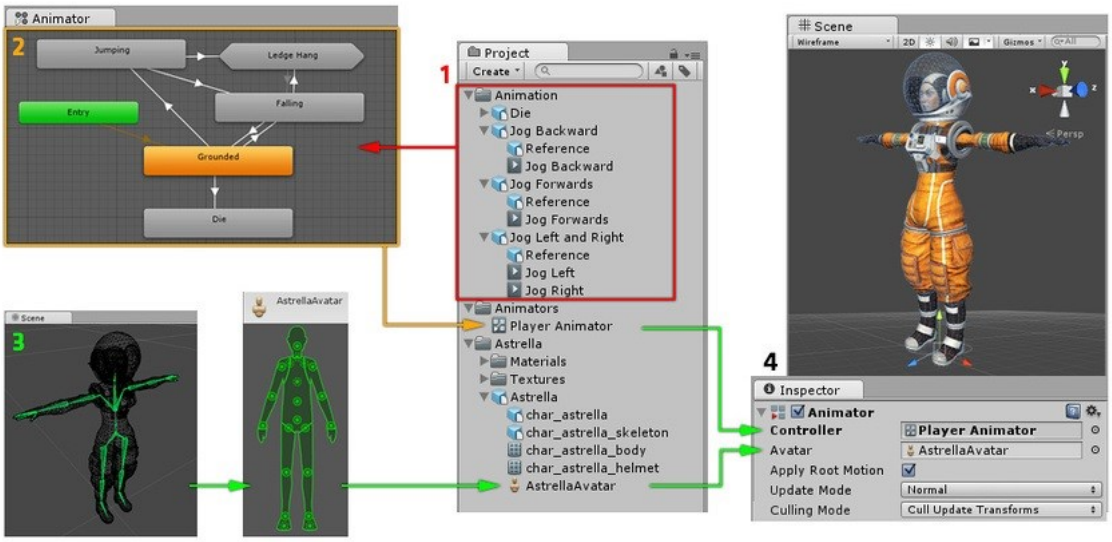
Unity si vyvinulo vlastní systém animací. Tyto animace lze uplatnit na libovolný objekt ve hře, a to včetně všech jeho komponent. Zároveň však podporuje i animační klipy z jiných zdrojů, tedy se stal velmi univerzálním.

Tento systém je založen na konceptu Animation Clips takzvaných animačních nahrávek, ve kterých lze najít změny jednotlivých objektů, jako je změna pozice, rotace, měřítka či nejrůznějších parametrů komponent.

Systém je postaven na takzvaných Animator Controllerech. Animator Contreller slouží jako stavový automat, který propojuje jednotlivé animační klipy a lze jej editovat v nástroji Animator.

Animační záznamy, kontrolér a objekt jsou spojeny komponentou Animator Component, která se připne k objektu. Lze v ní nastavit rovněž dodatečné parametry animace.

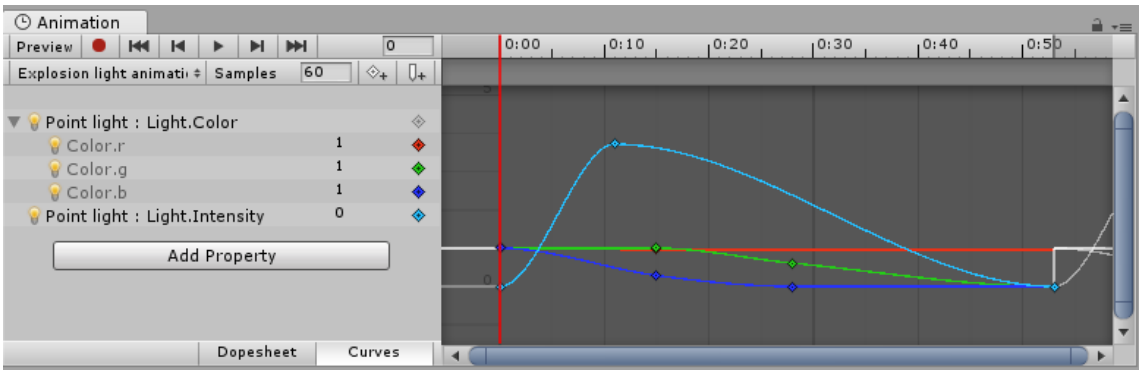
Na diagramu 2.18 lze vidět jednotlivé komponenty systému animace.



Obrázek 2.18: Diagram vývoje animace Zdroj: [13], Animation System Overview. V červeném ramečku lze vidět složku animací, která obsahuje jednotlivé klipy. Z těch se vytvoří kontrolér, což je zmíněny stavový automat, jehož jednotlivé stavy jsou animace. Následně objektům přidáme komponentu animátoru, do kterého dosadíme zmíněny kontrolér, který obsahuje reference na Animace a odkaz na avatara, vůči kterému se animace uplatňují.

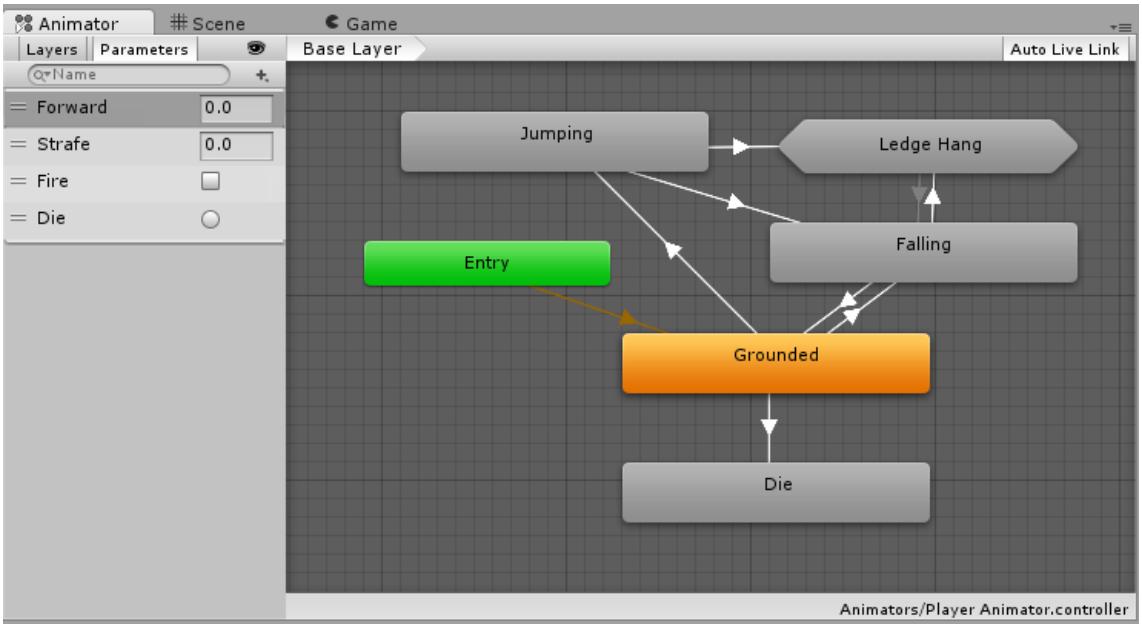
Na obrázku 2.19 lze vidět základní nastroj pro nahrávaní animačních klipů. Použiti je velmi jednoduché. Nastaví se časová osa do bodu 0:00, a stiskne se tlačítko nahrávání. Vybere se objekt, kterého se bude animace tykat a následně jsou dvě varianty, jak přidat nějakou hodnotu do animace. Stačí změnit hodnotu, co chceme animovat a automaticky se přidá jako sledovaná proměnná do klipu. Nebo přímo v nástroji lze vybrat Add Property, kde se zvolí sledovaná proměnná. Následně se přepíná do klíčových bodů animace většinou minima a maxima proměnných a v jednotlivých časech proměnné nastavujeme na

požadované hodnoty. Po dokončení všech požadovaných hodnot se ukončí nahrávaní klipu opětovným stisknutím tlačítka pro nahrávaní. Animační klip je nutno uložit a pak s ním lze teprve pracovat.



Obrázek 2.19: Vytvaření animace Zdroj: [13],Animation Clips. Vlevo lze vidět jednotlivé proměnné, které se animují. V hlavní části pak časovou osu se změnami proměnných.

Když jsou již jsou animační klipy vytvořeny, je potřeba vytvořit stavový automat na jejich správu. Takový stavový automat se nazývá Animator Controller a edituje se v nástroji Animator.



Obrázek 2.20: Tvorba stavového automatu, Zdroj: [13],The Animator Controller Asset. V levém menu nastavujeme jednotlivé proměnné, kterými se stavový automat řídí. Jednotlivé přechody, které jsou znázorněny bílými šipkami umožní přechod pouze tehdy jsou-li hodnoty parametru na požadovaných hodnotách. Jednotlivé stavy znázorňující animační klipy jsou pak šedě s výjimkou vstupního stavu (zelený) a aktivního stavu (oranžový).

Na obrázku 2.20 lze vidět výchozí stav Entry. Z něj je veden přechod do stavu, který se provede vždy po startu animace. Následuje již stav, kterému je přidělena animace. Většinou jde o stav Idle a podobě. Například na obrázku můžeme vidět stav Grounded. Lze

vidět, že ze stavu Grounded vede několik orientovaných hran(přechodů) do ostatních stavů. Každá z nich může být podmíněna, a to bud na rovnost integeru, boolu či spuštění na triger. Animaci nám tak ovládá jiný skript. Z příkladu na obrázku se jedná o skript chůze, který odesílá kontroleru jednotlivé proměnné, kterými tak ve stavovém automatu povoluje přechody animací tak, aby správně reagovaly na akci prováděnou hráčem. Stavový automat pro animace zpravidla nemá žádný cílový stav.

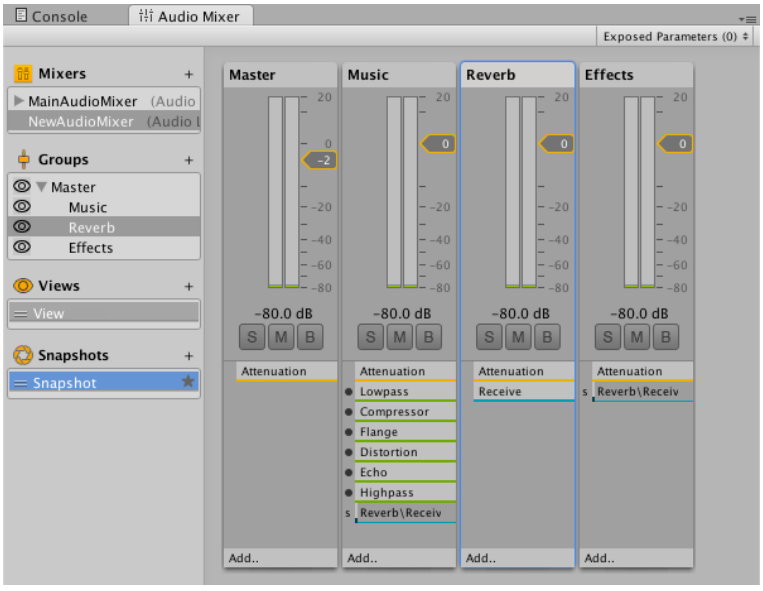
2.7 Zvuk

Unity má propracovaný systém zvuků. Dovoluje importovat nahrávky několika standardních formátech jako je:

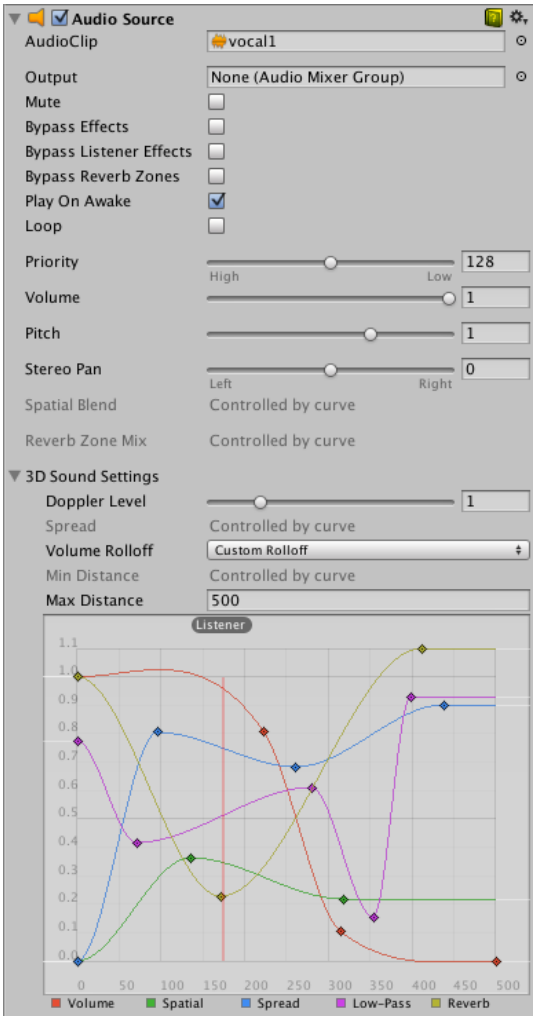
- WAV
- MP3
- Ogg
- AIFF

Z importovaných nahrávek se pak při importu do Unity vytvoří Audio Clip.

Aby zvuky působily co nejpřirozeněji, existuje v Unity komponenta AudioSource, kterou lze připnout k libovolnému objektu a zvuk se tudíž ozývá z různých zdrojů ve scéně obr. 2.22. Hráč pak obsahuje komponentu Audio Listener, která pro upřesnění nejčastěji bývá přidána přímo ke kameře. Pokud se pak ve scéně přehrává zvuk, je brána v potaz vzdálenost posluchače od objektu, jeho natočení vůči objektu, či dokonce, zda jiný bližší objekt nepřehrává zvuk. V takovém momentě dochází k překrývaní zvuků úplně stejně, jak by bylo možno očekávat v reálném světě. Pokud je vyžadován nějaký efekt zvuku, lze ho dosáhnout použitím nástroje Audio Mixer, který je součástí enginu Unity viz 2.21.



Obrázek 2.21: Audio Mixer, Zdroj: [13]. V nástroji Audio Mixer lze upravovat výsledné audio. Jak lze vidět na obrázku zvuky lze rozdělit do několika skupin a ty pak separátně upravovat, či jim přidávat jednotlivé efekty.



Obrázek 2.22: Audio Source, Zdroj: [13]. Nejdůležitější komponentou, jak již bylo zmíněno, je Audio Source. Tato komponenta přehrává zvuk z místa objektu, na kterém je umístěna. Audiosource se může na objektu nacházet vždy jeden. Má-li tedy objekt vydávat více zvuků, je nutno měnit v komponentě parametr AudioClip za požadované zvuky. Je-li tak učiněno, aktuální zvuk se přestane přehrávat a začne se přehrávat následující. Pro to existuje funkce playOneShot, která zaručuje přehrání více zvuků zároveň z jedné komponenty. Následně je důležitý parametr skupin. Zvuk se může zařadit do skupiny pro Nástroj Mixer. Tyto skupiny slouží k přidání nejrůznějších efektů. Dále komponenta obsahuje nastavení možnosti jak ztlumit zvuk, vynechat efekty, či přehrát ihned po stvoření objektu. Následně lze zvolit prioritu Zvuku. Tato hodnota slouží v momentě přehrávání více zvuků zároveň. Samozřejmě lze zvolit hlasitost i hladinu zvuku, či upravit rozložení sterea. Následně komponenta obsahuje graf, kde lze nastavit parametry zvuku v závislosti na vzdálenosti posluchače od zdroje. Zvuk také lze přepnout čistě do 2D režimu, kdy je toto nastavení eliminováno.

2.8 Kolidery a fyzický engine v Unity

Unity obsahuje systém Koliderů, což je další komponenta, kterou lze připnout k objektu. Kolider definuje tvar objektu v rámci kolizí na základě fyzického enginu. Tvar kolideru nemusí odpovídat vizuální komponentě objektu čili Mesh rendereru. Je často žádoucí, aby

kolider byl pouze tvarová abstrakce daného předmětu. Jednoduchost koliderů velmi ovlivňuje výpočetní náročnost každého snímku hry. Pokud se využijí standardní komponenty unity ve 3D režimu,lze najít 4 typy možných koliderů.

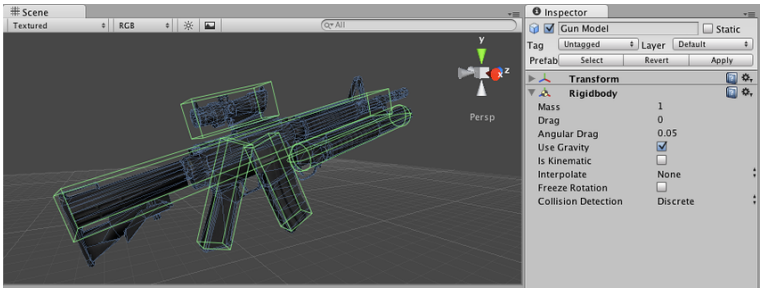
- Box Collider
- Sphere Collider
- Capsule Collider
- Mesh Collider

Box, Sphere, Capsule -jejich tvarem jsou základní primitiva. U nich se upravuje jen pozice, rotace a meřítko vůči svému rodiči (prvku v hierarchii objektu nadřazenému).

Mesh collider je kolider přesného tvaru, jako jeho geometrie a pro správnou funkčnost je nutno, aby byl mesh konvexní, popřípadě komponenta sama je schopna takový model z meshe dogenerovat. Mesh collider je velmi náročný na výpočetní výkon. Je tedy nutné dobře zvážit přínosy takovéhoto kolideru. Pokud je potřeba, jsou dvě možnosti optimalizace. První možnost je seskládat kolider z jednotlivých primitiv viz 2.23. Druhá pak nahrát pro kolider méně detailní model, který ovšem nebude mít mesh renderer a bude tak jen geometrií pro kolider.

Po nasazení komponenty kolideru na objekt, lze rozlišovat kolidery na dva typy, a to statické a dynamické. Záleží však na tom, zda objekt obsahuje komponentu Rigidbody. Tato komponenta dává enginu informace o fyzikálních vlastnostech objektu, jako je hmotnost, odpor pohybu v rámci prostředí. Objektu také lze přiřadit fyzický materiál s vlastnostmi. Dokonce lze u objektu nastavit působení gravitace a její sílu. Pokud tedy objekt obsahuje komponentu rigidbody, stává se kolider dynamický (předpoklad že se bude po scéně pohybovat). Pokud objekt komponentu rigidbody nemá, stává se z něj kolider statický. Takovéto objekty si můžeme představit jako stěny, podlahy, ulice a další objekty vytvářející základní prostředí, po kterém se hráč pohybuje. Při spuštění hry, pokud dynamický kolider zasáhne statický kolider, tak se zavolají události OnColliderEnter, OnCollisionStay během probíhající kolize a OnColliderExit po přerušení kolize. Koliderům však lze nastavit ještě vlastnost Trigger. Trigger se stává průchozím a při kolizi s jiným objektem, volá funkce OnTriggerEnter (Collider), OnTriggerStay, OnTriggerExit. Tento kolider se používá ke spuštění událostí ve hře.

Díky přítomnosti koliderů ve scéně, lze používat další systém enginu unity Raycast. Jedná se o vyslání paprsku z určitého bodu scénou, určitým směrem, na určitou vzdálenost. Paprsek si pak poznamenává, jaké kolidery po cestě protnul. Tento systém lze využít k mnoha věcem. Například zjištění, jaký objekt se nachází přímo před hráčem, na jaký objekt chtěl hráč myší klinout, či na čem hráč ve virtuálním prostoru stojí, nebo na zjištění vzdálenosti od podlahy a určení, zda je hráč ve vzduchu (skok pád a pod). Kolidery, které raycast zasahuje, lze také rozdělit do několika vrstev. Při vysílaní paprsku Raycast pak lze upřesnit, zda se má aplikovat jen nad určitou vrstvou koliderů, a tím značně zoptimalizovat použití raycastu.



Obrázek 2.23: Zdroj: [13], Rigidbody. Na obrázku lze vidět zeleně označené geometrické primitiva. Jsou to jednotlivé kolidery, které společně tvoří fyzický model objektu. Tato optimalizace se hodí u objektu, kde hráč není schopen poznat přesnou hranici objektu.

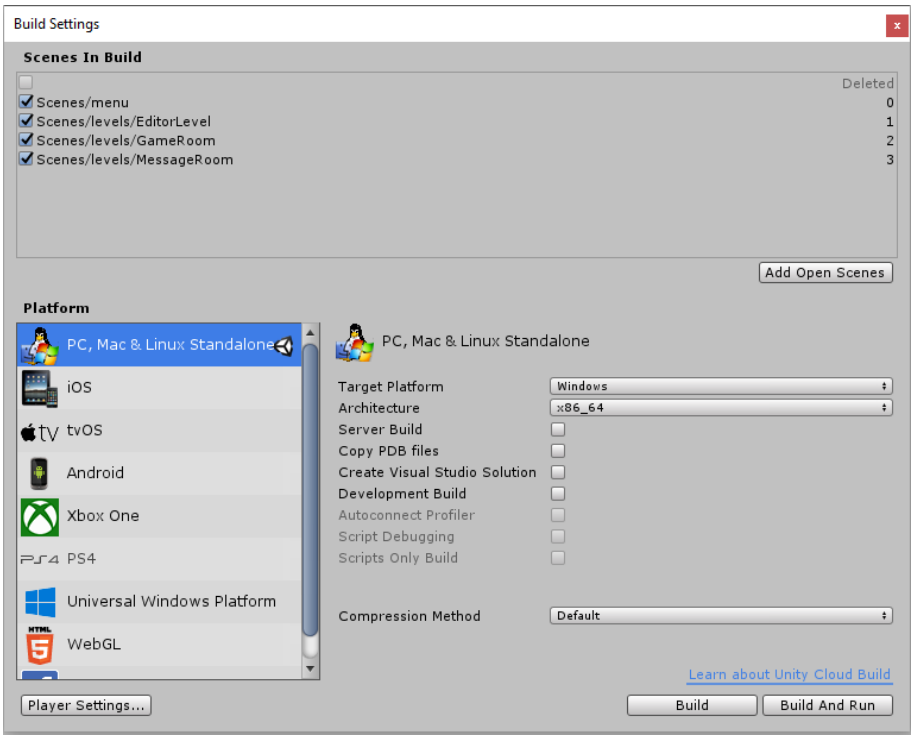
2.9 Balíky a rozšiřitelnost unity.

Jak již bylo zmíněno v úvodu, Unity obsahuje možnost rozšiřitelnosti pomocí balíků. Tyto balíky přidávají funkcionality přímo do herního enginu. Ty nejzdařilejší se dokonce i časem stávají součásti herního enginu, pokud jsou vyhodnoceny za velmi užitečné. Některé balíky se bohužel také odstraňuji z nejrůznějších důvodu kompatibility, či kvůli novému směru šablon použitelných v Unity.

Protože Unity dříve obsahovalo již v základu balík standardních asetů, byl jsem z minulosti již s ním dobře obeznámen. Proto i teď, když už je pouze jako volitelný, jsem se rozhodl jej přidat do projektu. Balík standartních asetů, který je zdarma k použití, obsahuje několik předchystaných modelů a asetů.

Balík obsahuje základní 2D i 3D prefaby, jako například hráče z různých pohledů (FPS, pohled třetí osoby) Také obsahuje hned několik variant modelů s různým zastoupením komponent a nejrůznější komponenty, které jsou často využívány ve hrách. Tyto asety jsou velkým zdrojem inspirací . Zvlášť při počátku vývoje lze velmi rychle dosáhnout prvního prototypu a následně jej upravovat.

2.10 Sestavení



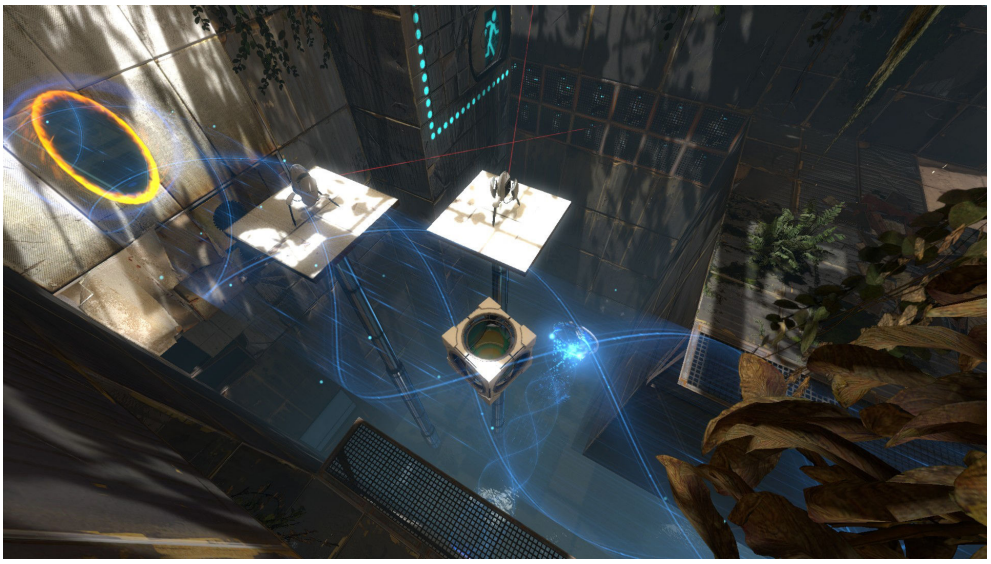
Obrázek 2.24: Build Settings. Unity, jako multiplatformní engine, nabízí podporu velkého množství operačních systémů. Když se otevře build menu, což je součást enginu Unity, je nutné vybrat scény, které má výsledná hra obsahovat. Během vývoje zpravidla vytvoříme více scén (místností), ať už jako testovací, či s konfigurací, která nakonec byla vylepšena a bylo vhodné místnosti nahradit. Výchozí nastavení Unity je vývoj hry pro Windows. V build menu lze toto změnit. Je ale potřeba mít samotnou instalaci rozšířenou o podporu platforem, pro které se sestavují hry. Platformy se již volí během instalace samotného enginu. Není však problém si je i doinstalovat později. Poté, co se vybere platforma, jedná-li se o jinou, než aktuální cílovou, je potřeba skripty znovu zkompilovat a ověřit jejich správnost vůči platformě. Několik málo knihovních funkcí je totiž v Unity pouze pro použití na určitých platformách. Proto, když takové funkce jsou použity, je nutné vytvořit alternativní skripty pro ostatní platformy. Stejné problémy se řeší, pokud je cíleno například na mobilní telefony a tablety. To je třeba vyřešit a namapovat například ovládací prvky přímo na display, jako nerušnější joysticky, či namapovat samotný dotyk na kurzor myši. U dotyku je nutné taky kontrolovat stisknutí a je to více komplexnější, než samotné události myší. Poté, když jsou nutné záležitosti závislé na platformě vyřešeny, lze zaškrtnout, zda se jedná o vývojářskou, či finální verzi a takto již lze vytvořit finální sestavení. Tady bych chtěl podotknout, že napříč několika posledními verzemi se v Unity objevila chyba, kdy překlad skriptů při přepnutí platforem selže. Zatím se to nepodařilo odstranit, ale stačí vždy Unity restartovat. Pokud by se taková chyba někomu objevila, je to nejjednodušší způsob, jak se s ní vypořádat. Pokud se tak neučiní, tak se může v sestavení objevit náhodné chování. V mém případě to byla naprostá nefunkčnost systému raycast.

Kapitola 3

Návrh

3.1 Inovativní mechanika

Kvalitních her s inovativním konceptem není mnoho. Hned zpočátku lze uvést asi největšího zástupce, a tím je série her Portal 3.1. Tato hra přinesla úplně nový náhled na způsob, jak překonávat levely a přirozeně podněcuje kreativní způsoby řešení jednotlivých úrovní. Pro hráče to byla velká změna, neboť hra obsahuje velmi chytlavý a promyšlený příběh. V tomto typu her je tvorba příběhu za hrou často problematická. Studio Valve, které tuto sérii vydává (stále se čeká na pokračovaní, které nelze říct, zda bude...), si dalo záležet na tom, aby vše bylo dotaženo k dokonalosti. Hraje se hned s několika koncepty z fyziky. Díky originálním mechanikám se z této hry stal světový fenomén. Po jejím boku se samozřejmě nachází několik dalších her, které představily zajímavé inovativní koncepty. Jde například o The Talos Principle, nebo The Witness.



Obrázek 3.1: Hra Portal 2 Zdroj: [14]. Hra vydaná společností Valve v roce 2011. Díky inovativním mechanikám a zpracování má hra velkou aktivní komunitu i po mnoha letech. Hra obsahuje rovněž editor, který umožňuje komunitě vytvářet nové úrovně. Tedy i dnes stále přibývá herní obsah.

Jak již bylo zmíněno, bylo zapotřebí vymyslet inovativní prvek. Nastala tedy fáze experimentování. Od nápadů s ohýbáním času a prostoru, až po změny trajektorií planet za pomoci černých děr. V této fázi vzniklo několik drobných prototypů, které ověřovaly, zda by mělo význam tyto nápady rozvíjet, nebo byly již v minulosti dostatečně zpracovány, tudíž jsou neinovativní.

V momentě, kdy se nápad pohyboval kolem tématiky vesmíru, bylo dobré se na vše podívat trochu s odstupem. Počátek vesmíru - Velký třesk. Při něm došlo k obrovskému uvolnění energie, která od té doby ve vesmíru nepřibývá. Naopak aktuální výzkumy naznačují, že se časem promění v temné prázdno . Proto je dobré si uvědomit, že vše kolem nás je pouze energie. Veškerá hmota, co známe, je tvořena energií, která vznikla při velkém třesku . Tady myšlenka na inovativní mechaniku začíná. Lidstvo zná několik druhů energie, které mají společnou vlastnost a to takovou, že lze jeden druh energie přeměnit na druhý. Toto téma se stalo hlavní myšlenkou mé hry. Hra je demonstrace jen části takových přeměn energie. Proto jsem vytvořil základ, do kterého lze přidávat další a další přeměny.

Jak mechanik s přeměnou energie přibývalo, bylo zapotřebí vytvořit nástroj pro jejich správu, aby bylo možno jednodušeji vytvářet herní úrovně. Zde došlo k další inspiraci,kdy ve hře Portal se také nachází editor. Dlouho po vydaní hry aktivní komunita dokáže vytvářet nové výzvy a hra se tak stala teoreticky nesmrtelnou a poskytující další obsah,i když vývojáři dávno pracují na jiných projektech. Proto jsem nástroj pro vytváření úrovně upravil na samotný editor tak, aby byl uživatelsky přívětivý a tím výrazně rozšířil možnosti tohoto dema. V podstatě poskytuji nástroje, jak si hru doslova sestavit podle svých představ. Samozřejmě za použití právě inovativního konceptu přeměny energie.

V momentě, kdy byl ujasněn celkový koncept, bylo na řadě vymyslet, jak tento koncept znázornit. V úvodu byl návrh, že hráč bude ovládat osobu z prvního pohledu, a ta bude muset jíst, aby měnila jídlo na energii. Jelikož je ale lidské tělo příliš složité, bylo nutné provést abstrakci. Abstrakce je však podobá více robotovi, bylo pak třeba změnit hlavní postavu z člověka na robota. To zároveň výrazně rozšířilo možnosti aplikovatelných přeměn energie na hráče.

Robotovi lze dodat energii více způsoby. Například použitím nalezených baterek, či generátorů energie, dokonce i z nabíječky fungující na principu piezoelektrického jevu. Tímto jevem se přemění energie potencionální na kinetickou a následně, po dopadu na piezoelektrickou podložku, na energii elektrickou, která je schopná dobít robota, který ji opět změní v pohyb (kinetickou energii). Samozřejmě každá přeměna má určité ztráty v podobě tepelné energie apod. Proto nelze měnit do nekonečna. Z toho důvodu je cílem hráče vyjít z místnosti, ve které se nachází, za použití přeměn energie, bez kterých se hráč neobejde.

Přehled jednotlivých přeměn energie, které nalezneme v demu:

- Pohyb hráče

  Hráč je vtělen do robota. Ten potřebuje ke svému pohybu energii. V podstatě hráč mění elektrickou energii v kinetickou. Je však nutno vzít v úvahu, že čerpání této energie je nerovnoměrné a odvíjí se od skutečnosti, zda se robot pohybuje do kopce, či na rovině, nebo z kopce. Je zřejmé, že do kopce spotřebuje více energie, než na rovině. Avšak při pohybu z kopce se situace liší dle typu robota. Pokud by byl robot s koly, je schopen při sjezdu rekuperovat část energie zpět. V tomto případě je ale výhodnější, použít humanoida, který díky své konstrukci napodobuje lidské tělo, a tím umožnuje hráči se lepé vžít do robota, za kterého hraje. Proto při pohybu z kopce dochází ke spotřebovávání energie, která je spotřebována k jednotlivým krokům a zaroveň brždění pohybu.

- Externí zdorje energie

  S cílem prodloužit vzdálenost, do které se hráč může dostat, bylo do dema zapotřebí přidat zdroje externí energie jako baterky, či generátor. Baterky lze aplikovat přímo a dobít si tak 20% energie z každé. Pokud bude hráč chtít využít generátor, musí se do něj prvně nalézt palivo. Samozřejmě tady se nachází přímá úměra. Čím více paliva je nalezeno, tím více energie lze získat.

- Piezoelektrický jev

  Jeden ze zajímavých způsobů, jak přeměnit kinetickou energii na elektrickou energii, je piezoelektricky jev. Tento jev je specifický pro krystaly, které když se deformují, generují elektrické napětí. Pro potřeby dema je vytvořena plošina, na kterou bude možnost skočit. Skokem na plošinu se stlačují krystaly k sobě, čímž generují elektřinu. Tato plošina zároveň vyrobenou energií dobije hráče. I zde platí, že množství dobité energie záleží na tom, z jaké výšky a jakou rychlostí hráč na plošinu dopadne. Díky této mechanice lze využít potenciál více vertikálně členitých úrovní.

- Přemísťování objektů

  V úrovni, jako jedna z dalších překážek, se nacházejí boxy. Tyto boxy lze zvednout a přemístit. Avšak přesun těchto boxů spotřebovává značné množství energie. Hráč se proto musí rozhodnout, zda je výhodnější v daný moment boxy přemístit, či využít raději energii na jejich obejití jinou cestou.

- Elektrická energie

  Elektrická energie je jedna z nejběžnějších, se kterou se lze setkat. Ve hře je možno elektrickou energii generovat dvěma způsoby. Alternátorem, nebo generátorem. Elektrická energie se poté vede stožáry až ke spotřebičům, jako je nabíječka, výtah, elektrické dveře. Spotřebič elektřinu spotřebovává a dle druhu spotřebiče umožňuje hráči provést akce. Nabíječka uloženou energii nabije zpět hráči, dveře se otevřou a je možno jimi projít. Výtah po připojení začne jezdit sám, není již potřeba manuálního ovládání, což vede na úsporu energie. Využití elektrické energie rovněž prohlubuje komplexnost úrovní.

- Mechanická energie

  Pokud se nevyužije elektrické energie a je potřeba se výtahem dostat do vyššího patra, nezbude nic jiného, než využít přeměny energie hráče na mechanickou energii. Použije se kladkostroj a vyjede se do požadované výšky. Tato možnost samozřejmě spotřebuje značné množství energie.

3.2 Editor

Editor se během vývoje stal součástí dema a podněcuje hráče k vytvoření vlastních výzev pro ostatní hráče. Prostor v editoru je rozdělen do voxelové mřížky. Je tak snadnější editace jak prostoru, tak zarovnání objektů mezi sebou. Zároveň voxelová mapa umožnuje optimalizace, které mají pozitivní dopad na výkon.

- Schopnost upravit místnost(nejen rozměry krychle).

- Vložit do místnosti klíčivé komponenty jako je hráč a cíl.

- Vložit mechaniky s přeměnou energie.
- Propojit v případě potřeby jednotlivé komponenty mechaniky (Elektrické vedení).
- Místnost otestovat.
- Místnost uložit.
- Načíst předchozí vytvořenou místnost .

Po načtení editoru se před hráčem nachází prázdná místnost. Nejsou v ní žádné elementy a hráč je ve výchozím stavu v editačním módu. Je to z toho důvodu, aby hráč mohl ihned přizpůsobit rozložení místnosti svým představám. Módy jsou celkově v editoru čtyři. A to:

- Editační mód
- Mód práce s objekty
- Mód propojování elektrického vedení
- Herní mód

V tomto editačním módu lze myší vybrat oblast voxelů, kterou lze posunovat. Kolem oblasti se objeví rám, za který lze výběr uchopit a posouvat podle potřeby. Směr, po které ose se bude posunovat, se rozhodne na základě prvního tahu myší promítnutého do 3D prostoru. Proto velmi záleží na natočení kamery vůči výběru. Pokud se vyberou voxely pouze jedné stěny, posune se daná stěna. Pokud se vybere však i protější stěna, posune se celá oblast. Takto se dají posouvat kompletní místnosti, pokud je potřeba.

V momentě, kdy je rozložení místnosti alespoň nahrubo vytvořeno, začínají se vkládat objekty. Ty se vybírají z levého dlaždicového menu. Editor se sám přepne do módu práce s objekty po výběru. Vhodnou praktikou je nejdříve umístit stěžejní objekty, tedy hráče a cíl v podobě dveří.

Následně jsou vkládány překážky, potažmo mechaniky, které při vhodném zkombinování umožní hráči úroveň projít. Fantazii se meze nekladou. Každý pokládaný objekt musí ležet na podlaze. Proto je zapotřebí přizpůsobovat rozměry místnosti právě zmíněným objektům. Některé objekty lze výškově upravit, např. výška výtahu se nastavuje kolečkem. Objekty se mohou přesouvat také hromadně. Pokud je zapotřebí vybrat kvádrovou oblast, jsou dvě moznosti. Buď se vybere jako první podlahový voxel a následně nějaký výše položený, či se vyberou pouze podlahové a následně kolečkem se provede trojrozměrný výběr.

Pokud se využije mechaniky elektrického proudu, bude potřeba ji správně propojit (od generátoru, až po spotřebič). Je nutné se přepnout do módu propojování, stisknutím klávesy „C“ (z anglického slova connect). Vždy se označí jako první objekt, který se bude propojovat. Ten se zvýrazní. Jako druhý se pak vybere objekt, ke kterému se připojuje. Ten rovněž změní označení. Mezi nimi se poté zobrazí grafické zobrazení spoje. Tak se zapojí celá kaskáda objektů. Zpět do editačního módu se lze dostat opětovným stisknutím klávesy „C“.

Následně, když jsou mechaniky hotové, je nutno místnost vyzkoušet, zda daná kombinace mechanik funguje podle představ. Proto, stisknutím klávesy „M“, se editor přepne do herního módu. Po dokončení, či zmačknutí klávesy „Escape“ se navrátí k rozpracované úrovni.

Během procesu je možno místnost průběžně ukládat. Uložení se provede stisknutím ikony diskety v pravém horním rohu. Vyskočí okno, do kterého se zadá název úrovně. Zde

je vhodné připomenout, že pokud již takové jméno úrovně existuje, bude úroveň přepsána. Pokud je potřeba pokračovat v editaci místnosti předešle uložené, stačí stisknout ikonu složky a rozpracovaná místnost bude načtena. Po ukončení práce a před vypnutím editoru je důležité vytvořenou místnost uložit, jinak budou změny zahozeny.

Kapitola 4

Implementace

V této kapitole je popsána implementace dema. Jak již bylo zmíněno, implementačním jazykem je C# a pracuje se v enginu Unity ve verzi 2018.3.6f1.

Implementované demo bylo pojmenováno EnergyGalaxies a je vytvářeno v šabloně Lightweight RP. Je to z toho důvodu, že koncept celé hry vyžaduj co největší možnost optimalizace a vzhledem k tvorbě ve voxelovém systému se nepředpokládají rozsáhlé reálně vypadající scenérie.

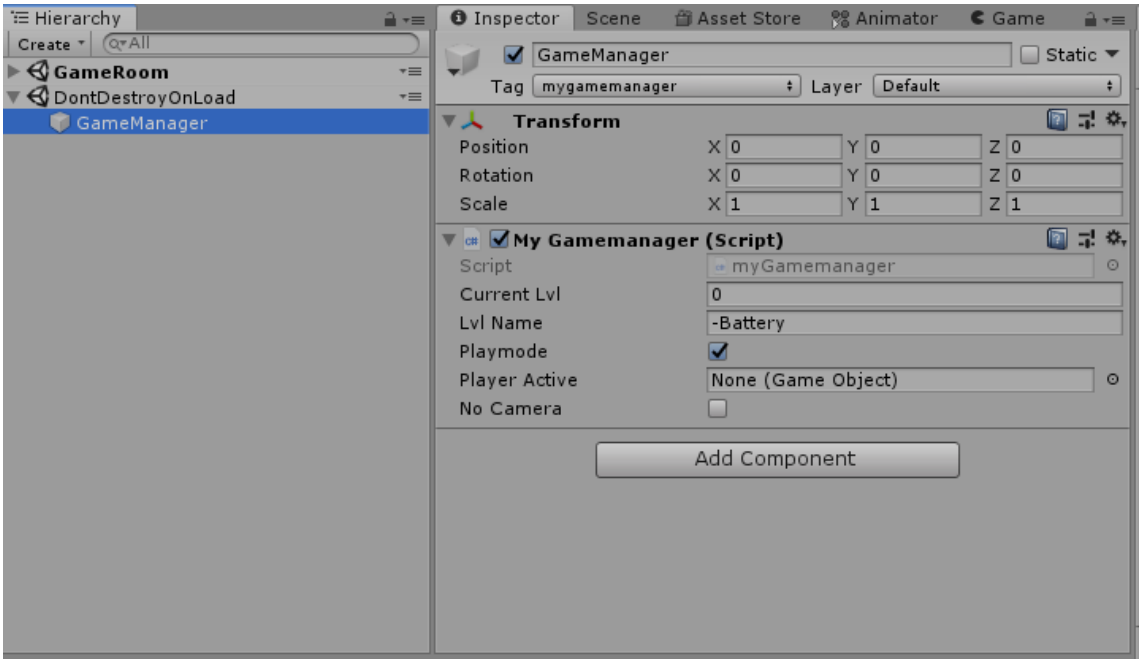
Dalším krokem v tvorbě hry je vytvořit scény. Scény se ve hře nacházejí celkem čtyři.

- Scéna Menu
- Scéna Editoru
- Scéna Herní místnosti
- Scéna Oznamovací místnosti

Do místností je potřeba dosadit základní objekty a vytvořit modely jednotlivých mechanik. Modely objektů pro tuto hru byly vytvářeny v programu Blender. Jedná se o program pro tvorbu 3D modelů. Je zdarma a zcela dostačující pro relativně jednoduché geometrie, které implementovaná hra obsahuje. Modely poté byly importovány do Unity, kde se z nich po dosazení nutných komponent vytvořily Prefaby (herní objekt s předdefinovanou strukturou komponent).

Jelikož se jednotlivé úrovně odehrávají v nespecifikovaných prostorech, bylo zapotřebí zvolit světla, která se budou vyskytovat ve hře. Vzhledem k tomu že se snažím zachovat optimalizovanou hru, je nejefektivnější použít ambientní osvětlení. Avšak je potřeba mu lehce upravit barvu, konkrétně do modra, aby místnosti působily pro lidské oko přívětivěji. Na jednotlivých objektech je pak použit materiál, který emituje světlo a dotváří tak atmosféru daných scén.

Než se implementují zvuky do Unity, je potřeba je vytvořit. V případě této práce byly zvuky vytvářeny za použití profesionálního programu FL Studio. Bylo potřeba vytvořit zvuky pro jednotlivé položky menu, a to jak na najetí myší, tak i zvlášť pro samotné kliknutí na tlačítko. Každá mechanika ve scéně pak má svůj zvuk v závislosti na typu mechaniky. U hráče lze nalézt trojici zvuků, a to zvuk pro chůzi, skok a dopad.



Obrázek 4.1: Vlastnost DoNotDestroy, vytvoří po načtení jiné scény scénu paralelní, vé které uchovává objekty, jejichž komponenty tuto vlastnost obsahovaly. Objekty jsou sice ve virtuální scéně, ale jsou jim umožněny všechny interakce v rámci zrovna načtené scény

4.1 Správa scén

Po vytvoření jednotlivých scén je potřeba vytvořit řízení celé hry. Tuto úlohu přebírá správce scén. Objekt, který má za úkol řídit průběh hry přepínáním příslušných místností. Je vytvořen v první scéně a obsahuje komponentu (skript) myGameManager. Tento správce hry je implementován jako singleton a s parametrem DoNotDestroyOnload [10], který zajišťuje, že při přecházení mezi místnostmi se tento prvek dostane do paralelní místnosti a zachová si tak všechny hodnoty.

Na obrázku 4.1 si lze povšimnout, že hierarchie obsahuje dva kořenové uzly, každý pro strom své scény. Scéna DontDestroyOnLoad běží paralelně se scénami dema a zachovává tak instance objektů v ní se nacházejících. Avšak chování takto vyčleněných objektů je nezměněno a mohou interagovat s právě načtenou scénou, jako by byly její součástí, i když ve skutečnosti jsou spuštěné ve scéně paralelní.

Na samotné přepínání scén je v Unity implementován systém správy scén SceneManagaer. Tento sytém poskytuje řadu funkcí, díky kterým lze odkazovat na místnosti a to dokonce trojím způsobem . Buď názvem scény, jejím build indexem (což je pořadí v jakém jsou scény vkládány do samotného sestavení viz. 2.24) a nebo strukturou místnosti.

Tyto hodnoty jsou pak klíčové k řízení průběhu hry. Správce hry musí znát dosavadní průběh hry, aby mohl jednoznačně určit, jaká místnost bude následovat, či zda bude muset zařídit specifické inicializace.

Následuje popis implementované logiky pro přepínání jednotlivých místností viz diagram 4.2. Po spuštění hra začíná v menu. Menu lze považovat jako rozcestník celé aplikace. Pokud v menu klikneme na tlačítka Example levels, či Select level, správce hry nepřepíná celou místnost, pouze zařídí deaktivace a aktivace příslušných objektů na plátně. Pokud je v menu kliknuto na jakýkoliv název úrovně, ať už z kategorie ukázkových (Example)

úrovní, či vytvořených úrovní přímo uživatelem, tak správce hry načte herní místnost a volá jednotlivé inicializace. Nejdříve je potřeba inicializovat samotný level ze souboru. Tedy předá informaci, z jakého souboru se načítá a zařídí inicializaci. Poté deaktivuje všechny inicializační objekty a aktivuje postavu hráče v místnosti. V případě, že dojde ke zjištění, že místnost neobsahuje hráče, načte se oznamovací místnost se zprávou o chybějící kameře. Z té je pak hráč přepnut zpět do menu. Pokud místnost úspěšné dokončí a splní všechny požadavky (například napájení dveří elektřinou a následné projití), načte se oznamovací místnost s pozitivní zprávou o výhře a spustí se časovač . Po časovači s ukázkovým levelem se načte následující místnost v pořadí a hráč pokračuje dál. Pokud to je místnost utvořena hráčem, vrací se do menu. Pokud hráč v dané úrovni umře, načte oznamovací místnost oznámení o neúspěchu a po vypršení časovače načte aktuální hranou místnost. Pokud se vsak hraje ve zkušebním módu z editoru, vrátí se po smrti do editoru.

Dále také obsluhuje vstupy od uživatele jako je stisknutí klavesy „Escape“, či stisknutí klávesy „M“ v editoru. Klávesa „Escape“ umožní vždy opustit místnost o úroveň výše a klávesa „M“ slouží k přepnutí se do testovacího módu z editoru a zpět.

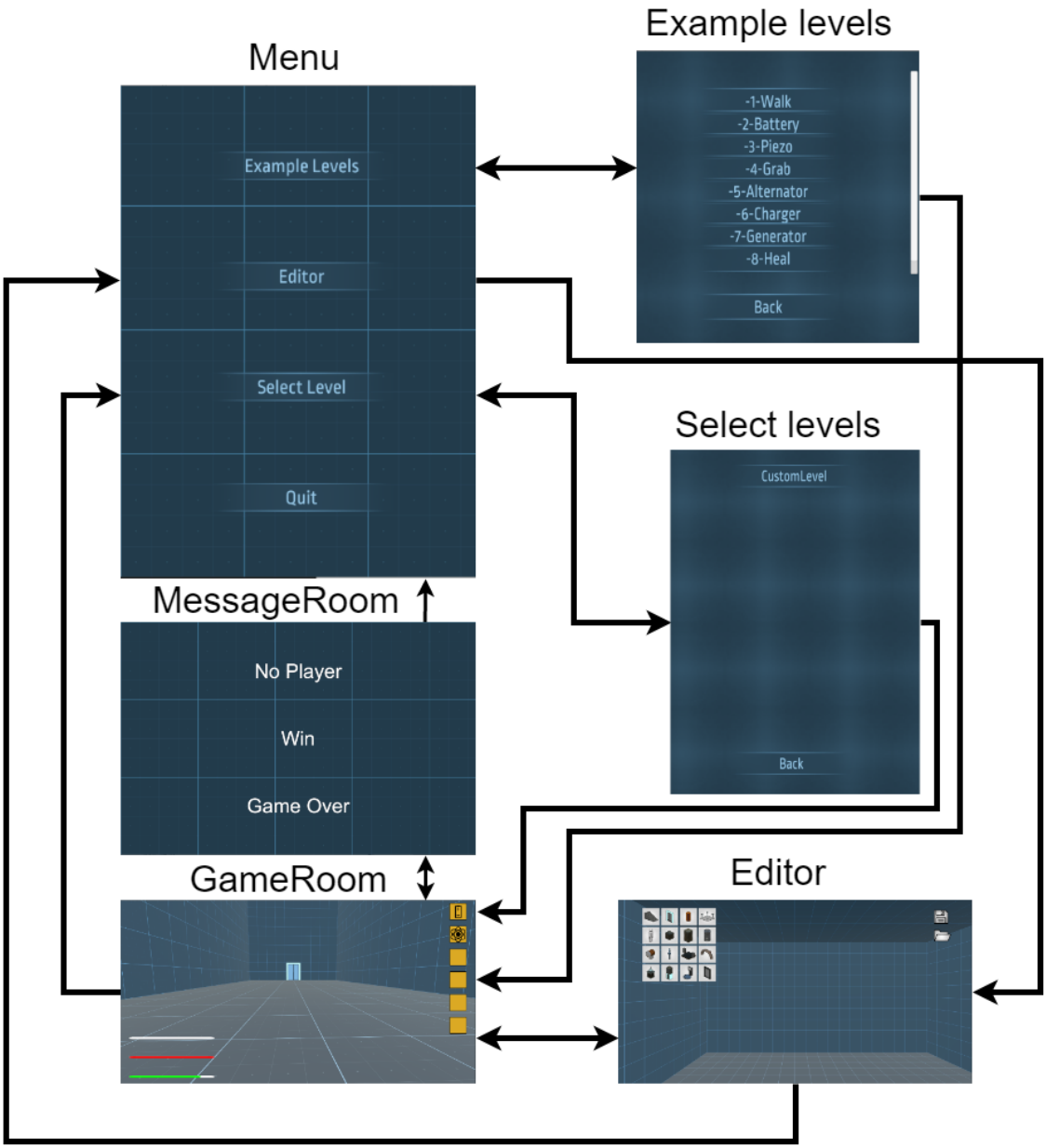
4.2 GUI

Implementováno pomocí plátna v Screen Space - Overlay módu. Plátno a jeho komponenty se přizpůsobují obrazovce vůči referenčnímu rozlišení, které je 1920x1080. GUI se vždy přizpůsobuje dle šířky. Proto, máme-li obrazovku vyšší, než širší, ovládací prvky zůstávají na středu s tím, že se adekvátně zmenší, aby mohly dodržet referenční pozicování. Rozdílné GUI jsou implementovány pro každou scénu. GUI editoru lze vidět na obr. 4.3 a GUI herní místnosti pak na obr. 4.4

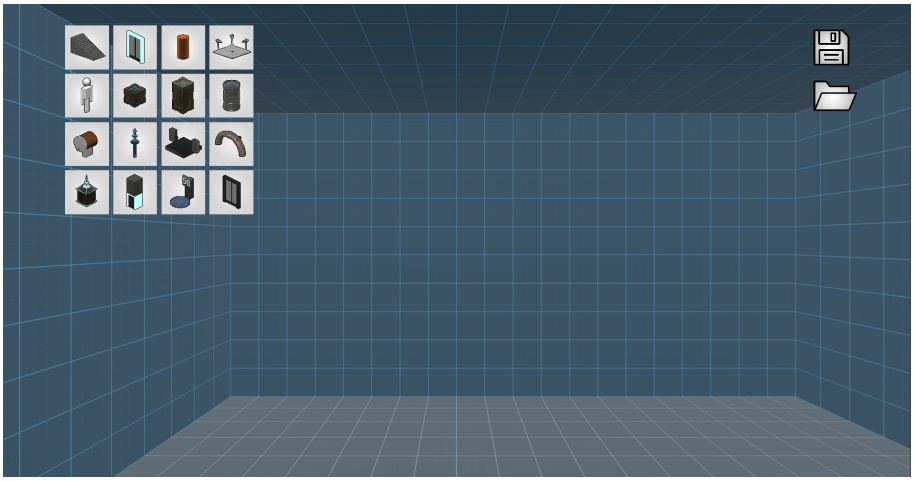
Hned po zapnutí hry se zobrazí místnost s hlavní nabídkou. Tato místnost je tvořena čistě UI prvky na plátně. Najdeme zde 4 tlačítka Example Levels, Editro Select Level a Quit. Tlačítko Example levels skryje tyto tlačítka a místo nich zobrazí podmenu s výběrem testovacích úrovní. Obdobně je řešeno tlačítko Select level. Zmiňuji to především proto, že mají shodná chování a volají skript FileExpLoad, který prohledá složky s uloženými místnostmi a dle nich vygeneruje příslušná tlačítka pro jejich spuštěni. Tato tlačítka jsou generovaná z prefabu. Tomuto prefabu stačí pak jen měnit text, který se použije jako parametr při načítaní místnosti. Pokud neobjeví žádnou místnost, vygeneruje podobný prefab tlačítku, jen již neklikatelný.

Následně tu nejdeme tlačítko pro spuštěni editoru. Toto tlačítko zavolá správce hry, který se postará o zbytek viz. 4.1 a tím nás přenese do editoru.

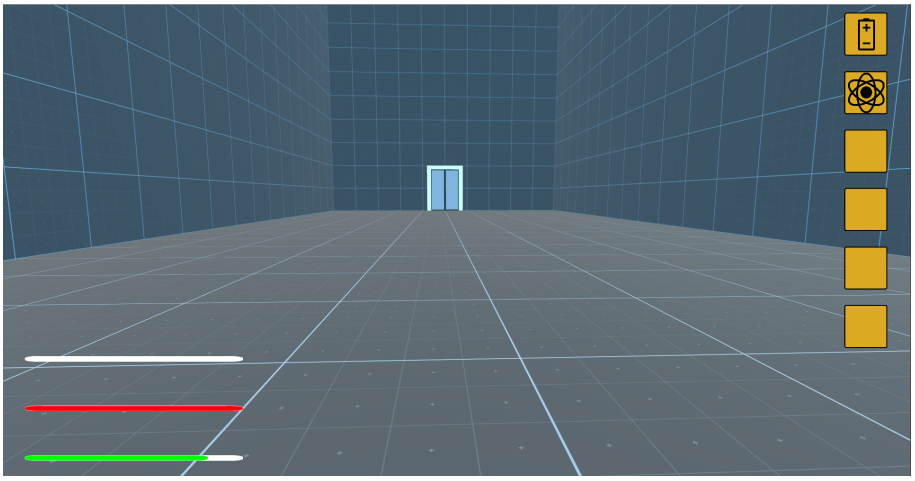
Poslední tlačítko pro opuštěni hry pak volá Application.Quit(); jedna se o knihovní funkce Unity [8].



Obrázek 4.2: Na diagramu je znázorněno přepínání scén řízené správcem hry. Demo začíná v Menu. Součástí Menu je také Example leveles a Select levels. Jedná se pouze o jiný obsah plátna. Obě obsahují tlačítko zpět, které zobrazí původní menu. Example levels obsahuje ukázkové urovně a Select level úrovně vytvořené a uložené v editoru. Kliknutí na tlačítko úrovně vede na načtení úrovně v herní scéně (GameRoom). Pokud nastane výhra, či místnost neobsahuje hráče, přepíná se do oznamovací scény. Z té pak následně do Menu. Pokud hráč prohrál, znovu se načítá herní scéna a úroveň se opakuje. Klávesa „Esc“, pak kdykoli ukončí hru a načte Menu. Z Menu lze spustit Editor. V editoru lze vytvářenou místnost testovat v herní místnosti. Po dokončení editace, lze editor ukončit a tím se načte opět Menu.



Obrázek 4.3: Editor GUI V editoru v levém horním rohu najdeme grid layout zaplněn ikonami (komponenta Image) s jednotlivými vkladatelnými objekty do scény. Vpravo pak ikony pro uloženi a načtení. Po kliknutí na tyto ikony se aktivuje okno, které je ve výchozím stavu skryto. Okno pro uložení se skládá z objektu s komponentou Input Field a dvěma tlačítky. Jedno pro uložení a jedno pro opětovné skrytí okna. Okno načtení se pak skládá z jednotlivých her nalezených v adresáři pro uložení hry a každá naleznutá úroveň má svoje vygenerované tlačítko. Opět se zde nachází tlačítko zpět. Přepne-li se hráč do režimu Connect, zobrazí se rámeček kolem obrazovky. Ten je tvořen obrázkem s nastavenou maskou a zrušením reakce na raycast, aby šlo skrz něj „klikat“.



Obrázek 4.4: GUI hráče / herní scény. Samozřejmě hráč potřebuje svoje UI rozhraní, tedy spolu s místnosti se již načte předpřipravené plátno, které obsahuje prvky hráče. Toto plátno lze rozdělit na dvě části. Vlevo se ukazují všechny potřebné ukazatele pro hráče, například energie, život či ostatní ukazatele. Vpravo se pak nachází hráčův inventář, který lze ovládat klávesami 1 až 6, jelikož inventář obsahuje pouze 6 míst. Součástí plátna je také obrázek, který je ve výchozím stavu neviditelný. Jedná se o animovaný obrázek, kterému se upravuje alpha kanál, což má za následek efekt tepu, pokud hráči dochází energie. Tato animace je ovládána ze skriptu energie, který se nachází přímo na hráči.

4.3 Editor

Editor se stal jednou z nejvýznamnějších komponent celé práce. Díky tomu je také jednoznačně nejsložitější. Scéna editoru zpočátku obsahuje jen několik málo objektů, mezi které patří:

- Plátno
- Editor - Objekt, na kterém se nachází komponenta EditorRoom, což je největší komponenta(skript) nacházející se v této práci.
- Camera Objekt s komponentou kamery a využívající komponentu Simple Camera Controller ze standardních assetů.
- Fileload Objekt obsahující komponentu File Exp Load(skript), který nalezne a generuje uložené hry v rámci příslušného menu.

Zbytek objektů pro funkčnost celého konceptu je generováván. Po načtení se proto generuje základní místnost, která se bude upravovat. Lze však načíst nějakou z již rozpracovaných místností, ale o tom více v systému uložení 4.7.

4.4 Prostředí

Celý prostor scény rozdělíme na jednotlivé krychle - voxely. Stanovíme si rozměr každé krychle na 2x2x2 jednotky rozměru scény. Je tak proto, protože výchozí prefab hráče ze standardních assetů má na výšku 1,8 jednotek, tedy rozměr 2 jednotky je ideální, aby se hráč vešel do každého voxelu. Dle této mřížky, je pak vytvořeno měřítko jednotlivých vkládaných objektů a 1 voxel se tak stává základní jednotkou této hry. Aby se dalo ve skriptech počítat pouze s kladnými čísly, celá scéna byla přesunuta do I. Kvadrantu. Tímž se zaručí pouze kladné hodnoty souřadnic a zároveň je oříznuta tak, že první voxel, tedy jeho roh, začíná na souřadnici 0, 0, 0 a nejpravější roh a pak souřadnicí 200, 200, 200. Jedná se tedy o krychli o 1 000 000 voxelů. Tak velký prostor se ukázal jako více než dostatečný pro libovolné úrovně. Každý voxel je v Unity reprezentován svým objektem. Aby však byla hra optimalizovaná a nenacházelo se ve scéně mnoho objektů najednou, vytváří se a zase ničí s editací scény. Ponechávají se jen ty, které tvoří hranici mezi vnitřním prostorem a vnějškem. Tak lze udržet počet opravdu nutných objektů na minimu. Tyto voxely se zapisují do prostorové bitmapy. Tato bitmapa obsahuje 0 pro voxely neobsahující vnitřní prostor místnosti a 1 pro vnitřní prostor. K této bitmapě má přístup každý voxel a díky tomu zná svoje okolí, i když objekty, které jej reprezentují, zrovna nemusí existovat. Pokud aktualní voxel zjistí, že je vnitřním prostorem a jakýkoli z 6 okolo je vnější prostor, vygeneruje si vnitřní voxel zeď s vnějším. Vnější zeď je instanciována z prefabu a každý voxel si udržuje na danou zeď referenci, aby ji v případě potřeby mohl smazat. K tomu dochází v momentě, kdy se předchozí vnější voxel změní na vnitřní voxel. Takto se samotný prostor udržuje uzavřen a není možné, aby někde zůstala „prázdná díra“ do vnějšího prostoru. Pokud se pak provádí editace místnosti, vždy se volají jen nejbližší objekty reprezentující voxely, aby ověřily, že se prostředí kolem nich nezměnilo. Potom není potřeba dogenerovat další zeď.

4.5 Editace objektu v editoru

Aby bylo vidět do místnosti, bylo, třeba vyřešit problém přivracených a odvracených stěn. Implicitně v Unity sice odvracené objekty lze prohlédnout skrz, nicméně z logiky věci je raycast stejné zachytí a zároveň se nabízí optimalizace, což vedlo k vytvořeni systému skrývaní stěn. Tento systém zkoumá pozici kamery vůči stěnám, nacházejícím se ve scéně, a propočítává úhel, kterým je stěna k němu natočena. Stěny, které jsou odvrácené deaktivuje, avšak musí na ně udržovat referenci, aby je bylo možno znovu aktivovat. Pokud by se reference na takový objekt ztratila, nelze jej znovu zapnout, ani zničit a zbytečně spotřebovává prostředky počítače. Systém je řízen také proměnnou pro jeho dočasné pozastavení. To je potřeba tehdy, pokud se bude přepínat kamera, nebo ověřovat možnost usazení objektu ve scéně.

Jednou z prvních interakcí hráče v editoru bude výběr oblasti. Tento výběr je umožněn, pokud se skript editoru nachází v prvním, neboli editačním módu. Po stisknutí tlačítka myši se vyšle raycast z bodu, kterým je poloha myši v závislosti na kameře. Pokud tento raycast projde koliderem, zdi se označí a slouží jako výchozí bod tahu výběru. Každý následující snímek pak probíha kontrola, zda hráč stále drží myš, a tím zvětšuje výběr. V každém následujícím tahu se vysílá raycast z pozice myši a kontroluje se, zda neprotnul jiný kolider typu zeď. Pokud ano, na základě nové souřadnice vygeneruje krychli, která pojme obě tyto zdi a zároveň tím přirozeně pokryje i další kolidery. Tyto kolize jsou zaznamenané a zdi se přidají také do výběru. Tak se postupně vytváří výběr a přidávají se do něj zdi v okolí. Poté, až hráč pustí tlačítko myši, přidají se všechny herní objekty do vyběru, který se nachází ve vybrané krychli. Zároveň se degenerují okraje krychle, které se stávají interaktivními a v dalším kroku editace lze za tyto okraje prostor posouvat.

Pokud je zapotřebí zvolit nějaké předměty nad výběrem, lze výběr rozšířit do výšky za pomoci kolečka. Zde si systém pamatuje výchozí souřadnici, ze které se kolečkem začalo vybírat a nikdy nelze přetéct do záporného výběru. Tento alternativní mód výběru byl implementován primárně kvůli možnosti vybrat skupinu objektů, a ty hromadně přemístit (když se v jejich okolí nenachází žádná jiná zeď).

V momentě, kdy je označen výběr a je potřeba jím posouvat, se najede myší na rám označovací krychle. Po najetí se změní kurzor, který naznačuje, že za danou oblast se dá táhnout. Pokud se pak stiskne tlačítko myši, vyšle se raycast a systém si zapamatuje souřadnici, kde došlo ke kolizi. Vypočítá se vzdálenost myši od kolize. Následně se čeká 0.15 sekundy a znovu se promítne pozice myši do prostoru a porovná s uloženou vzdálenosti a určí se, která ze tří souřadnic se změnila nejvíce. Na základě tohoto směru se vygeneruje grafická reprezentace osy, po které se bude táhnout. Následně se počítá delta pozice myši promítnuté do prostoru vůči pozici zdi. Pokud překročíme deltu velikosti voxelu, dojde k přesunu po dané ose o jeden voxel. To se provede tak, že se všechny objekty, vyjma zdí, odeberou z vnitřních struktur systému. Objekty předmětů jsou indexovány jejich pozicí. Přepíše se tedy jejich pozice a znovu se přidají do vnitřních struktur skriptu, až po dokončení přesunu. Co se týče zdí, úpravy se provádějí pouze v bitmapě, dle toho, jaké zdi obsahuje výběr. Zároveň se voxelům, které jsou v okolí změny, zasílá zpráva o změně, a ty na to reagují aktualizací (odstraní, či přidají stěny tak, aby zůstal vnitřní prostor uzavřen).

Je třeba si připomenout, co je prefab. Prefab je předpřipravený podstrom objektu s jedním kořenovým uzlem, který je pojmenován jménem prefabu. Samozřejmě všechny objekty mohou obsahovat libovolné komponenty z enginu. Aby se takový prefab mohl instanciovat po vytvořeni scény, je nutné jej přesunout do složky Resources, která je právě výchozí složkou a přibaluje se do výsledného sestavení. Z ostatních složek instanciovat možné není.

Pro možnost vytvoření objektu v implementovaném editoru musí prefab splňovat určíte podmínky. Kořenový objekt prefabu, musí obsahovat komponentu Default val, která v sobě udržuje informace jako jsou rozměry objektu přepočteného na voxely, kolik místa na podlaze potřebuje a také odkazy na komponenty typu Mesh renderer, aby bylo možné vyznačit změnou materiálu dobu, kdy je pouze umisťován, ale není ještě součástí místnosti. Prefab také může obsahovat pouze jeden aktivní kolider. Tento kolider se deaktivuje během umisťovaní, aby bylo možno posílat raycast skrz tento objekt. Po dosazení opět hlavní kolider aktivuje, aby se dal objekt opakovaně vybrat. Potřebuje-li nějaký objekt mít pak ve herní scéně více koliderů, je nutné aplikovat na kořenový objekt další komponentu s názvem Collider after play. Té se předají reference pouze v rámci daného prefabu na ostatní kolidery. Zvolí se, zda kolidery mají být v herní místnosti aktivní, nebo zda je potřeba je vypnout. Tento skript si hlídá, v jaké scéně se nachází, díky vestavěným funkcím sceneManageru.

V momentě, kdy v editoru je kliknuto na ikonu objektu, skript editoru se automaticky přepne do režimu přidáváni objektu. Ikona volá funkci pravě v tomto editoru a předává svůj název. Tento název je shodný s názvem prefabu, který je přidáván do scény. Tento prefab, se nejprve vytvoří na souřadnici 0,0,0. Zároveň po instanciaci prefabu ve scéně je v každém dalším snímku vysílán raycast a zasáhne-li podlahu, je právě vytvořený prefab přesunut na pozici této podlahy. Hráč pak stisknutím tlačítka myši může objekt položit. Proběhne zde však kontrola, zda se prefab vejde do umístění, kam hráč požaduje a zároveň, zda všechny jeho části stojí na podlaze. Pokud ne, objekt není položen a hráč stále zůstává v módu přidávání objektu. Hráč může ukončit mód úspěšným položením prefabu na místo, které splňuje požadavky na uložení nebo stisknutím klávesy Delete, čímž je prefab znovu vymazán a editor se opět přepíná do módu editace.

V rámci přidávání objektů bylo nutno ohlídat dvě krajní situace. Jednak, když se hráč pokusí přidat druhého hráče. Skript editoru si vždy pamatuje referenci na aktuálního hráče a vymaže předchozího hráče po úspěšném vytvoření nového.

Další odlišností v přidávání je přidávání objektu výtahu. U tohoto prefabu lze ovlivnit jeho výšku kolečkem a tím se mu generují jednotlivé kolejnice. Po voxelech se ukládají do skriptu lokální kopie prefabu, který je vytvářen.

4.6 Propojení elektrifikovaných objektů

Mód propojení elektrické energie vznikl z přirozené potřeby propojit vedeni od generátoru až ke spotřebiči. Každý objekt, který má interagovat s tímto režimem, musí obsahovat komponentu Electricity. Ta v sobě obsahuje základní funkce pro distribuci elektrické energie a zároveň si vždy pamatuje následníka, kterému elektrickou energii předává. Tento režim má za úkol doplnit následnost objektů, a tím objekty propojit. Pro přepnutí do tohoto režimu se použije klávesa „C“ a režim je odlišen rámečkem. Kliknutím na objekt, ten zčervená, čímž je rozlišeno, že je aktivní. Dalším kliknutím se zvolí objekt, který je následující v kaskádě a přebarví se na oranžovou barvu. Opětovným kliknutím na červeně označený objekt je označení zrušeno. Zpět z tohoto režimu se lze přepnout opětovným stisknutím klávesy „C“.

4.7 Systém uložení

Řešení ukládacích systémů lze najít v Unity několik. Ukládat lze pouze serializovatelná data. Kvůli tomu nelze ukládat pro unity nativní vektory, či dokonce celé GameObjekty . Je nutné tedy vytvořit konvertory do právě serializovatelných proměnných. Poté, kdy

jsou vytvořeny sterilizované proměnné, jsou k dispozici tři druhy uložení. Unity podporuje zapsat jednoduché hodnoty přímo do registru, může se tak jednat o hodnoty typu skóre, či dosažená úroveň. Pokud je nutné ukládat informací více, lze použít dvě metody. Buď jde o uložení do takzvané persistentní paměti. Toto umístění je specifické pro každý operační systém, avšak Unity má k dispozici knihovní funkci persistent path, která potřebnou cestu vrací.

Další možná metoda je uložení do libovolné složky. Zde nastává ovšem často problém s oprávněními dané aplikace a je tedy dobrou praktikou ukládat pouze do složek, které jsou podsložkami složky s umístěním samotné hry. Poté již lze vytvořit soubor a data se do něj uloží v binárním kódu díky formátovači. Tak jsou ochráněna proti cílenému přepisovaní a tím možnosti změny jejich obsahu za ziskem výhody ve hře. Při opačném postupu, jsou tyto uložené hry otevřeny z daného umístění. Formátovačem se rozkódují a data se přiřadí potřebným proměnným, či se generují potřebné objekty.

V mém demu využívám na ukládání celkově dva skripty. První, Level data, obsahuje strukturu dat, které se budou ukládat. Díky tomu, že hra ukládá objekty místnosti na základě pozice, stačí jen šest proměnných typu float, do kterých se převede vektor pozice a vektor rotace. Následně je potřeba si uložit bitmapu, která je již v serializovatelném formátu. Poté je potřeba uložit kaskádu elektrického zapojení, což se docílí uložením pozice následujícího objektu kaskády zapojení.

Druhý skript, SaveSystem, se stará o správu souborů v rámci systému, na kterém hra běží a zároveň o zakódování dat při ukládání a následně jejich rozkódování při načítání.

4.8 Herní místnost

Po načtení samotné scény jsou zároveň načteny podstromy objektů, které byly do dané místnosti umístěny a uloženy. Aby bylo možno vytvářet vlastní scény se svou vlastní strukturou objektu, byl zapotřebí vytvořit systém ukládaní a načtení objektů. O samotném uložení je vice napsáno v kapitole 4.7. Dále se bude vycházet z toho, že již existuje soubor reprezentující uloženou scénu z editoru. Herní místnost po načtení obsahuje pouze základní kameru a objekt, na který je připevněn script editoru, vzhledem k tomu, že nelze předem určit tvar místnosti v souboru, Script Editoru má v sobě zakomponované funkce pro generovaní objektů. Tyto funkce využívají systému Unity, kdy lze instanciovat prefaby, neboli předpřipravené stromové struktury objektu, do již načtené scény. Tudíž je čten soubor s uloženými daty a poté se dosadí bitmapa prostoru pro daný level. Dále se dogenerují do místnosti jen potřebné voxely a ty rozhodují o vytvoření jednotlivých scén na základě svého okolí. Následně se pak instancují ostatní objekty. Je však třeba nezapomínat na hráče. Poté, co se nahraje herní místnost, manažer hry zabezpečuje načtení místnosti podle předaného jména úrovně. Při načtení objektu je především nutné ohlídat, zda se v místnosti nachází hráč, viz. herní manager. Pokud se v místnosti hráč nachází, lze vstoupit obrazem do místnosti. Je třeba poukázat na to, že pokud se v místnosti nenachází cíl v podobě dveří, nelze místnost úspěšné dokončit. Základní logiku úrovně musí zajistit designér dané úrovně.

Jedním z již předchystaných assetů je FirstPerson3DCharacter. Tento předpřipravený model se skládá ze skriptů pro chůzi a kamery, která má na sobě skript animace, jež imituje chůzi. Také zároveň obsahuje ovládání pohledu pomocí myši. Tento model jsem použil jako výchozí, pouze jsem ho lehce modifikoval skripty. Zároveň jsem pod kameru přiřadil prázdný objekt rukou, který slouží při potřebě hýbat s objekty. Ke kořenovému objektu jsem přidal komponenty – skripty Energy, Walking a Inventory.

Script Energy v sobě obsahuje správu, již zmíněné energie hráče, stará se o aktualizování stavů ukazatelů na plátně a zároveň poskytuje rozhraní pro aktualizovaní hodnot energie od ostatních skriptů.

Script Walking sleduje pozici hráče a adekvátně k tomu odebírá energii. Skript musí ohlídat, kdy hráč skáče, či padá. Pokud hráč padá, musí si pamatovat z jaké výšky spadl, aby mohl spočítat, zda bude hráč zraněn. Rovněž se tento údaj použije u piezoelektrické plošiny. Všechny změny energií se poté posílají formou zpráv skriptu Energy, který je přes své rozhraní zpracovává. Skript také spouští, či zastavuje zvuk chůze hráče a spouští zvuk doskoku při dopadu.

Script inventory dbá o správu hráčova inventáře a zároveň se stará o uloženi aktuálního stavu inventáře. Také obsluhuje hráčovu interakci v podobě užití itemu po stisknuti kláves 1 - 6, čímž se vybírá předmět, který hráč použije. Skript samozřejmě při každé změně aktualizuje plátno a překresluje příslušné pole.

Další úpravu tohoto modelu lze nalézt na kameře, kde se nachází komponenta se skriptem PlayerRaycast. Tento skript vysílá raycast z pozice kamery a pokud protne kolider jiného objektu vyvolá událost. Skript na tuto událost reaguje a vyhodnocuje, jaký objekt byl takto zasažen, jinými slovy, na který objekt se hráč zrovna dívá. Zároveň se počítá vzdálenost od objektu a pokud je hráč dostatečně blízko a stiskne klávesu „E“, skript reaguje dle mechaniky daného objektu.

Jak již bylo zmíněno, mechaniky lze třídit do několika kategorií. Dále se nachází výčet jednotlivých implementací mechanik podle pořadí předmětu, jak se nacházejí v editoru.

- První takovouto mechanikou je cíl, který je reprezentován dveřmi. Jedná se spíše o pasivní mechaniku, která je obsloužena bez jakéhokoli nového skriptu, neboť se jedná pouze o zprávu správci hry o tom, že byla dokončena úroveň. Tato zpráva se zasílá ze skriptu PlayerRacast na kameře hráče.

- Následující mechanika je baterie, se kterou se váže systém inventáře, který již zde byl popsán. Po kolizi s baterií se přidá předmět do inventáře a v případě, že se použije, dobije energii pouhým voláním přidaní energie ve skriptu energy.

- Následující mechanikou je Piezoelektrický efekt, který je ztvárněn doskokovou plošinou. Implementace je obsažena, již ve skriptu walking, který si zároveň hlídá, po jakých objektech chodí. To je umožněno pomocí raycastu směrovaného přímo z prostředí hráče rovně dolů a ignorováním hráčova vlastního kolideru. První zásah kolideru je proto podlaha. Pokud je podlaha piezoelektrická podložka, přidává se energie na základě rozdílu výšky z místa skoku a výšky plošiny. Skript Walking pak volá skript energy s hodnotou energie na dobití.

- Následně lze najít mechaniky překážek. Jsou implementované v podobě velké a malé krabice, které lze přemisťovat. Jejich zvednutí stojí hráče energii. Tuto vlastnost zajišťuje script CarryUp, který objekt při interakci skrze rscript raycast player nadzvedne o 0,2 jednotky do výšky a pak periodicky odebírá energii. Objekt však kopíruje polohu objektu rukou a pokud narazí do jakéhokoli jiného kolideru automaticky se přestává objekt přesunovat a vrátí se zpět na zem. Samozřejmě menší krabice ubírá při zvednutí méně energie, neboť objekty mají vlastnost Rigidbody. To je jedna z komponent fyzického modelu unity, v které lze nastavit i simulovanou váhu objektu, včetně působení gravitace. Gravitace se pro přenos objektu vypíná a hráč s ním může tedy pohybovat libovolně.

- Další je sud, který obsahuje palivo do generátoru elektřiny. Tato mechanika funguje na základě označování objektu různými tagy, což je označovací systém přímo v enginu Unity. Pokud raycast script narazí právě na takový tag,. hráči se do inventáře přidá toto palivo nalezené v sudu Aby se výběr paliva nemohl opakovat, změní se tag sudu na nedefinován. Tím se zabrání další možnosti opakovaného výběru. Skript na tento objekt následně nereaguje. Díky skriptu inventory palivo z inventáře lze použít a zaplní jeden z dalších barů.

- Další mechanikou je využití elektrické energie v místnosti a také způsobu, jak ji vést mezi předměty. Nejprve je potřeba se zaměřit, jak elektřinu vyrobit. Hra obsahuje dva způsoby - manuální a pomocí nalezeného paliva. Manuální je reprezentován pomocí alternátoru. Pokud k němu hráč přijde, opět se pomocí raycastu určí, že jde o alternátor. Stisknutím tlačítka E se spouští animace otáčení klikou a zároveň i script Generate enregy (komponenta na alternátoru), který volá metodu Generate a tím v pravidelných časových intervalech vytváří energii. To samozřejmě spotřebovává hráčovu energii. Jelikož jde vždy o výměnu, volá se proto na hráči odečtení energie. Zároveň se kontaktuje další objekt v kaskádě zapojení. Zpravidla je k dispozici přenosová anténa, která má také malou zásobu energie. Ta působí v podstatě jako kondenzátor a díky němu lze pozorovat efekt zpoždění při nabíhání a vypínání vedení.

  Automatický způsob výroby elektřiny je znázorněn elektrickám generatorem, do kterého se musí vložit palivo. To se odečte ze skriptu energy a na základě odečteného množství se postupně přeměňuje na energii. Volá obdobnou funkci Generate jako alternátor a opět napájí další objekt v kaskádě zapojení.

  O kaskádu zapojení se stará script Elektricity, který musí být součástí všech objektů, které používají elektrickou energii.

  Mezi spotřebiče patří výtah, či elektrické dveře. Výtah po zapojení na elektrickou energii jezdí sám a není již vyžadován ruční pohon a dveře umožní hráči přejit do další úrovně. Také lze připojit nabíječku, která dokáže předat vyrobenou elektrickou energii hráči.

- Výtahy byly implementovány, aby poskytly více způsobů, jak využít vertikální prostor úrovní. Bez elektřiny je výtah v čistě manuálním módu a lze jej ovládat klikou, podobně jako alternátor. Aby bylo možné vůbec posouvat hráče společně s výtahem, bylo nutné vytvořit systém změny rodičovského objektu hráče. Systém změní rodiče hráče v hierarchii objektů na výtah. Tím je svázána výška hráče s polohou výtahu. Naopak při opuštění prostoru výtahu se musí rodič hráče zpět přiradit na místnost, aby se veškerá hráčova poloha nevztahovala k výtahu. Na to slouží oblast triger kolideru, takový kolider nezabraňuje vstoupit jinému kolideru do jeho oblasti, avšak vyvolává událost. V události se pak ošetří změna rodiče hráče a pak se pouze znovu přes skript raycastu hráče kontroluje, zda hráč otáčí pákou výtahu. V ten moment se odečítá energie a výtah se posouvá směrem nahoru. Pokud hráč 3 vteřiny s výtahem nepohne, výtah sám samovolně sjíždí dolů. Je to řešeno spuštěním časovače v neaktivní době a pokud hráč pohne s výtahem, časovač se resetuje.

- Další mechanikou jsou nebezpečné zóny. V implementovaném demu je reprezentována v podobě horké vodní páry. Ve hře se nachází potrubí, odkud pára periodicky uniká. Tato oblast má okolo sebe opět kolider typu triger. Pokud se vněm nachází hráč a

pára zrovna uniká, tak se mu odečítá život opět voláním skriptu energy s příslušnými hodnotami.

- Poslední mechanikou jsou jednosměrné dveře, které byly přidány pro tvorbu komplexnějších levelů a fungují opět na kolideru typu triger. Pokud k nim hráč vstoupí ze strany, kde se kolider nachází, dveře se otevírají až do úplného otevření. Když z kolideru vystoupí, dveře se začnou uzavírat. Je třeba ohlídat krajní stavy dveří, aby nedocházelo k nežádoucím událostem. Toto chovaní zabezpečuje skript SmallDoors.
- Cílem úrovně je projít finálními dveřmi. Existují dva druhy dveří. Klasické, ty jsou hráči otevřeny stále a dveře elektrické, kterým je třeba dodat elektrickou energii, aby bylo možno projit.

4.9 Oznamovací scéna

Obrázek 4.5: Na obrázku je zobrazena oznamovací scéna, kde lze vidět jednotlivé zprávy. Hrači se vždy zobrazí pouze jedna z nich, záleží jaka hodnota je nastavena při inicializaci místnosti. Automatický Layout zajistí vycentrování zvoleného textu.

Tato scéna byla vytvořena za účelem oznamovaní herního stavu. Jedná se v podstatě o 2D scénu, kde je pouze text, který značí herní událost 4.5. V implementovaném demu jsou takové události tři.

- V místnosti chybí kamera.
- Hráč místnost vyhrál.
- Hráč místnost prohrál.

Po načtení scény se zároveň ve správci hry spouští časovač na 3 vteřiny, po které je z místnosti přepnuto na základě předchozího průběhu hry. Pokud hráč umřel, tak se herní místnost zpravidla restartuje. Pokud byl v editoru, tak se vrátí do editoru. Pokud vyhrál a byla to úroveň spuštěná z vytvořených úrovní, bude hráč vrácen do menu, jinak obdobně vrácen do editoru.

Kapitola 5

Testování a optimalizace

Během vývoje vznikalo velké množství prototypů. Každý se používal k testování nově přidaných funkcionalit a ověření správného směru vývoje celé aplikace. Optimalizací během vývoje bylo provedeno nespočet. Od jednoduchých oprav chyb až po přepsání celých systémů. Velmi často byla i užitečná zpětná vazba, pokud jde o rozhraní. Jelikož hry se mohou hrát v jakémkoli věku a s jakýmkoli vzděláním, dával jsem hru otestovat všem, kdo projevili zájem. Tak jsem nasbíral velmi cenné informace o tom, co ve hře funguje a co ne a je třeba změnit.

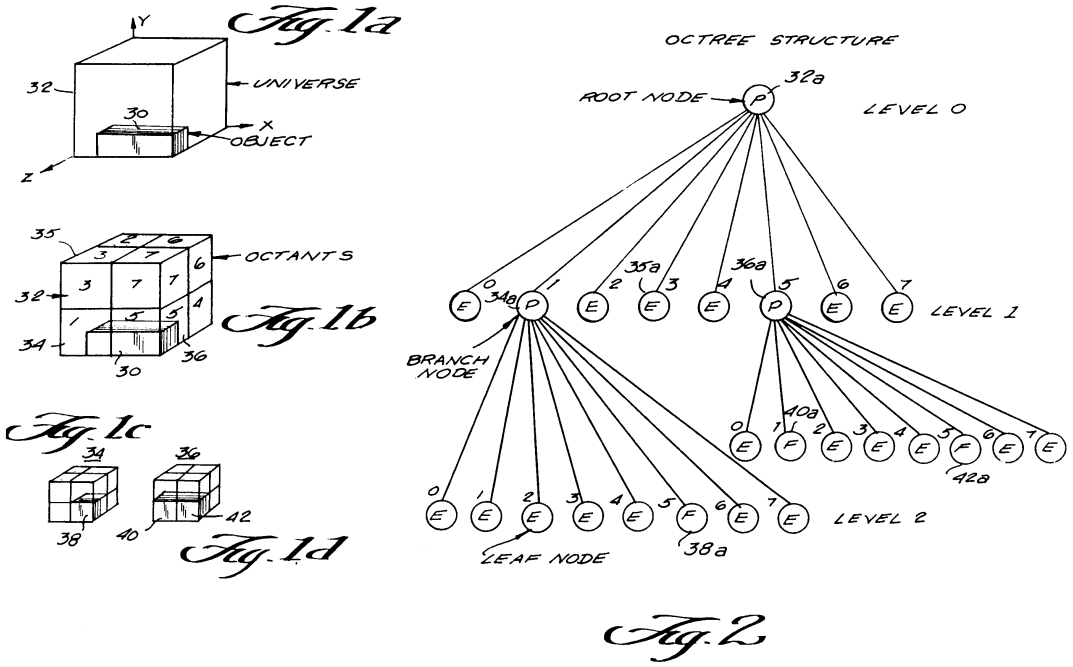
Jak aplikace procházela mnoha fázemi vývoje, jeden ze zásadních problémů, který se vyskytoval, byly dlouhé načítací časy. V jednom případě jsem naměřil dokonce více jak dvě minuty, než se načetla požadovaná scéna. Pro takovéto demo to bylo naprosto nepřijatelné a narušovalo to celkový zážitek. Když jsem zkoumal, co takový nepřípustný čas způsobuje, došel jsem k zajímavému zjištění, a to, že příčina je jedna z mých dřívějších optimalizací. Tato optimalizace spočívala v rozdělení scény a prostoru k editaci datovou strukturou Octree viz 5.1. Tato struktura byla ideální pro binární rozdělení prostoru. Krychle buď obsahovala prázdný prostor, nebo vnitřek místnosti. Prostor vždy dokázala popsat zcela, neboť nejmenší úrovní detailu byl jeden voxel a ten musel být vždy buď vnitřní, nebo vnější prostor místnosti.

Stromová struktura sice umožňovala rychlý přístup k objektům, avšak při editaci a tvorbě voxelů v místnosti bylo třeba voxely spojovat a zase rozdělovat. Tedy se Octree musel vždy znovu přepočítat. A právě tady spočíval kámen úrazu. V určitých oblastech je Octree efektivní, ale v Unity funkce vytvoření objektu a zničení objektu patří k nejnáročnějším. Jak se Octree měnil v závislosti na editaci, docházelo k masivnímu zamrznutí celé aplikace. Místo Octree bylo tedy nutno spravovat voxely jinak. Tak se použil způsob sledování oblastí změn. V praxi to znamená, že se voxely generují jen tam, kde jsou nezbytné. Všude jinde jsou buď jen deaktivované, či je nebylo potřeba vůbec tvořit. Tato optimalizace zkrátila čas načítání na pár sekund a dokonce uspořila i paměť,neboť není zapotřebí uchovávat všechny voxely.

Z herních mechanik se pak měnily například způsoby přidávání, či editace objektů. Velké zlepšení také nastalo přidáním možnosti vytvářet více stejných objektů, pokud se drží klávesa shift. Vzhledem k množství interakce hry s člověkem je velice jisté, že se dříve nebo později v demu objeví chyby. Je to jednoduše způsobené tím, že se dojde k nějakému z případů, se kterým se nepočítalo, či nějaká oprava způsobila novou chybu. U her toto bývá zásadní problém. Proto jsou hry pravidelně aktualizovány i dlouho po dokončení vývoje.

Mohl bych zde uvést příklad hry Fallout 76, která je i více jak půl roku po vydání hrou s obrovským počtem chyb, přičemž se jedná o AAA titul od herního studia Bethesda.

https://www.overleaf.com/project/5c6cf7f2fd19f402fe03f66c



Obrázek 5.1: Octree Zdroj: [6], Jedná se o Stromovou strukturu, která popisuje objekt ve 3D prostoru. Struktura vychází vždy z krychle, která je okolo popisovaného předmětu. Tato krychle se rozdělí na 8 stejných dílů. Díly, které obsahují popisovaný objekt a nejsou zcela zaplněny tímto objektem, se dělí znovu, až na požadovanou úroveň detailu. Postupným dělením vznikají jednotlivé úrovně stromu. Výsledný popis objektu je pak soustavou všech krychlí, které byly zcela zaplněny.

Kapitola 6

Závěr

Počítačové hry změnily svět. Pro mě osobně se staly motivací k samotnému programování. Být schopen naprogramovat hru, byl odmalička můj sen a dnes, když se ohlédnu, jsem si již tento sen splnil. Cílem bakalářské práce bylo prostudovat Unity a vytvořit inovativní herní demo. Během testování jsem obdržel pozitivní reakce na hru. Stejně tak jsem obdržel i řadu nápadů, co by se mělo změnit, či přidat. Pokud bych se měl zaměřit na to, co by se v krátkém časovém horizontu dalo zlepšit, tak například doladit některé méně intuitivní věci v editoru. Pokud bych se díval na demo z dlouhodobého hlediska, tak bych z něj chtěl udělat plnohodnotnou hru. Rozhodně přidat příběh a pokračovat v přidávání mechanik. Následovalo by přidání hry pro více hráčů a v poslední variantě by vznikla verze pro virtuální realitu. Jak již bylo zmíněno v práci, stojíme na přelomu nové herní éry, a to éry virtuální reality. Když bych měl odhadnout ze zkušeností z jiných her a pokusů, co jsem ve svém volném čase implementoval na dokončení tohoto dema, na plnohodnotnou hru ve virtuální realitě bych potřeboval minimálně další dva roky. Když nad tím přemýšlím, stálo by za úvahu, zda nepokračovat v této práci na magisterském studiu. Upřímně, tvorbu dema jsem si užil. Vytvořit herní demo a nové mechaniky není snadné. Samotný editor, který je součástí dema, jsem zpočátku odhadoval na více jak třičtvrtě roku vývoje. Jsem spokojený se stavem, v jakém se demo nachází a rozhodně počítám s pokračováním vývoje, ať už jako školní projekt, či ve svém volném čase.

Literatura

[1] Burke, N.: How to get the most out of the new Unity Project Templates in 2018.1. 2018, [Online; navštíveno 10.04.2019].
URL https://blogs.unity3d.com/2018/04/26/how-to-get-the-most-out-of-the-new-unity-project-templates-in-2018-1/

[2] Edwards, B.: *Computer Space and the Dawn of the Arcade Video Game*. 2011, [Online; navštíveno 11.04.2019].
URL https://www.technologizer.com/2011/12/11/computer-space-and-the-dawn-of-the-arcade-video-game/

[3] Fine, R.: UnityScript's long ride off into the sunset. 2017, [Online; navštíveno 10.04.2019].
URL https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/

[4] Gregory, J.: *Game Engine Architecture*. A K Peters/CRC Press, jun 2009, ISBN 9781568814131.

[5] Kent, S. L.: *The Ultimate History of Video Games: From Pong to Pokemon–the Story Behind the Craze That Touched Our Lives and Changed the World*. Rocklin, CA, USA: Prima Communications, Inc., 2001, ISBN 0761536434.

[6] Meagher, D. J.: High-speed image generation of complex solid objects using octree encoding. Září 15 1987, uS Patent 4,694,404.

[7] Modany, A.: *Pong, Atari, and the origins of the home video game*. 2012, [Online; navštíveno 11.04.2019].
URL https://americanhistory.si.edu/blog/2012/04/pong-atari-and-the-origins-of-the-home-video-game.html

[8] Unity Technologies: *Application.Quit*. 2019, [Online; navštíveno 12.04.2019].
URL https://docs.unity3d.com/ScriptReference/Application.Quit.html

[9] Unity Technologies: *Game engines how do they work?* 2019, [Online; navštíveno 5.04.2019].
URL https://unity3d.com/what-is-a-game-engine

[10] Unity Technologies: *Object.DontDestroyOnLoad*. 2019, [Online; navštíveno 10.04.2019].
URL
https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html

[11] Unity Technologies: *Unity Asset Store - The Best Assets for Game Making*. 2019, [Online; navštíveno 6.04.2019].
URL https://assetstore.unity.com

[12] Unity Technologies: *Unity Software Additional Terms*. 2019, [Online; navštíveno 10.04.2019].
URL https://unity3d.com/legal/terms-of-service/software

[13] Unity Technologies: *Unity User Manual*. 2019, [Online; navštíveno 5.04.2019].
URL https://docs.unity3d.com/Manual/

[14] VALVE: *Portal 2*. 2019, [Online; navštíveno 17.04.2019].
URL https://store.steampowered.com/app/620/Portal_2/?l=czech