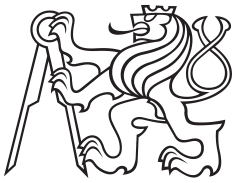
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Laboratoř Inteligentního Testování Systémů

Prototyp systému pro taktické cvičení ostrahy letiště

Bc. Filip Bursík

Vedoucí: doc. Ing. Miroslav Bureš, Ph.D.
Obor: Softwarové inženýrství
Studijní program: Otevřená informatika
Květen 2021

Poděkování

Chtěl bych poděkovat panu doc. Ing. Miroslavu Burešovi, Ph.D. za odborné vedení práce a cenné rady, které mi pomohly tuto práci zkompletovat. Dále bych chtěl poděkovat panu Martinu Čížkovskému z letiště Václava Havla v Praze za téma diplomové práce a průběžné konzultace, které byly přinosné a velice cenné pro vypracování diplomové práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 21. května 2021

Abstrakt

Inspirace pro toto téma pochází z israelské společnosti, která vymyslela software pro VR, který slouží jako trenažér pro vojáky, kteří jsou již mimo službu (ve výslužbě) nebo ze zdravotních důvodů odstoupili z armády na nějaký čas. Výcvik přes trenažér je praktičtější a to z hlediska finančního i organizačního. Mohou tímto simulovat reálné situace, které mohou nastat a software si upravit pro konkrétní potřeby, na rozdíl od cvičište, kde by byli limitováni různými faktory. Pomocí trenažéru jsou tedy schopni nasimulovat reálné podmínky a situace, které mohou v terénu nastat.

Cílem této práce bylo napodobit tento software a udělat minimální životaschopný produkt, který bude později rozšířitelný. Software má být přizůsobený pro prostředí letiště, kde taktická část je rozšířená i o operativní část a v simulátoru se budou simulovat konkrétní situace, které mohou nastat na letišti a ohrožovat bezpečnost cestujících, a zásahová jednotka bude muset tyto bezpečnostní hrozby eliminovat.

Klíčová slova: Virtuální realita, bezpečnost, letiště, výcvik, bezpečnostní hrozby, simulátor

Vedoucí: doc. Ing. Miroslav Bureš, Ph.D.
Karlovo náměstí 13,
121 35 Praha 2

Abstract

The inspiration for this topic comes from an Israeli company that invented software for the VR, which is used as a trainer for soldiers who are already off duty (retired) or left the army for some time for health reasons. Training with a trainer is more practical and from a financial and organizational point of view. In this way, they can simulate real situations that may occur and adapt the software to specific needs, where they would be limited by the various factors. Through the simulator, they are able to simulate real conditions and situations that may occur in the field.

The aim of this work was to emulate this software and make a minimal viable product that will be later extensible. The software has to be adapted to the airport environment where the tactical part is extended with the operational part and the simulator will simulate a specific situation that may occur at the airport and endanger passenger safety, where the intervention unit will have to eliminate these security threats.

Keywords: Virtual reality, security, airport, training, security threats, simulator

Title translation: System for Tactical Training of Airport Security

Obsah

1 Úvod **1**

1.1 Problematika 1

1.2 Stav poznatků o řešené problematice 2

1.3 Cíl 3

2 Analýza problému **5**

2.1 Základní pojmy 6

2.2 Herní enginy 7

2.3 Letištní budova 19

2.3.1 Fragmenty letištních budov . . 20

2.3.2 Typy senzorů 21

2.3.3 Typy útoků 22

2.3.4 Zásahové jednotky 23

2.4 Modelování 23

2.5 Virtuální realita - hardware 25

2.6 Shrnutí 27

3 Implementace a modelování **29**

3.1 Mechanismy simulace 29

3.1.1 Virtuální realita - taktická jednotka 33

3.1.2 Monitor - operátor 34

3.1.3 Virtuální realita - HUD 35

3.1.4 Denní cyklus 36

3.1.5 Letištní provoz 37

3.1.6 Cestující 38

3.1.7 Cestující 39

3.1.8 Pracovníci letiště 42

3.2 Modelování 42

3.2.1 Základní prvky modelu 43

3.3 Shrnutí 45

4 Testování **47**

4.1 Testování mechanismů 47

4.2 Statická analýza kódu 51

5 Závěr **55**

A Literatura **57**

B Screenshoty z navrhnutého letištního prostředí **59**

C Seznam použitých zkratek **65**

D Obsah přiloženého CD **67**

E Zadání práce **69**

Obrázky

1.1 Obrázek ukazující závislost produktivity na čase, pokud se kód neudržuje a netestuje [1]. 2

2.1 Ukázka koliderů v Unity. 9

2.2 Příklad GameObjectů ve scéně. . 11

2.3 Příklad komponent u GameObjectu. 12

2.4 Příklad zobrazení komponenty *Box Collider* při běhu simulace. 13

2.5 Ukázka tagů v Unity. 15

2.6 Levotočivý souřadnicový systém vs pravotočivý souřadnicový systém [2]. 16

2.7 Příklad animatoru. 17

2.8 Příklad animace a možnosti upravování animace. 17

2.9 Příklad NavMesh prostoru. 18

2.10 Příklad raycastů 19

2.11 Postup halou pro cestující, kteří odlétají. 24

2.12 Postup halou pro cestující, kteří přilétají 24

2.13 HTC Vive Pro se základními stanicemi a ovladačemi [3]. 25

2.14 Natočení VR headsetu [4]. 26

2.15 Outside-in tracking vs. Inside-out tracking [5]. 26

2.16 Příklad hardwaru simulujícího nekonečný prostor (závěs) [6]. 27

3.1 1. část UML diagramu navrhnuté architektury. 30

3.2 2. část UML diagram navrhnuté architektury. 32

3.3 Ukázka "inventáře". 33

3.4 Obrazovka, kterou vidí operátor na monitoru PC. 35

3.5 HUD pro ukončení, restart simulace nebo případnou úpravu nastavení. 36

3.6 Plán příletů a odletů - hodina 00:00 - 11:00. 37

3.7 Plán příletů a odletů - hodina 12:00 - 23:00. 37

3.8 Vysvětlivka tabulek příletů a odletů. 37

3.9 UML diagram popisující stavový automat cestujících po příletu. . . . 40

3.10 UML diagram popisující stavový automat cestujících po příletu. ... 41

3.11 UML diagram popisující stavový automat zaměstnance letiště. 42

3.12 Model letiště. 43

3.13 Model příletové haly. 44

3.14 Model odletové haly - přízemí.. 44

3.15 Model odletové haly - 1. patro. 44

3.16 Modely rukou v simulaci. 45

4.1 Ukázka Test Runneru. 49

4.2 E2E testování. 50

4.3 Ukázka z výstupu SonarQubu. . 52

4.4 SonarQube ukázka severit. 53

4.5 Docker architektura [7]. 54

B.1 Ukázka venkovního letištního prostoru. 59

B.2 Ukázka venkovního letištního prostoru. 60

B.3 Ukázka venkovního letištního prostoru. 60

B.4 Ukázka venkovního letištního prostoru. 61

B.5 Ukázka příletové haly letištního prostoru. 61

B.6 Ukázka odletové haly letištního prostoru. 62

B.7 Ukázka odletové haly letištního prostoru. 62

B.8 Ukázka odletové haly letištního prostoru. 63

B.9 Ukázka odletové haly letištního prostoru. 63

Kapitola 1

Úvod

Diplomová práce popisuje implementaci softwaru pro výcvik zásahových jednotek, kdy se v krajních situacích (jako např. útok na letiště) trénují taktické postupy spolu s operativními postupy. Taktické zasahové jednotce může být nápomocný právě zmíněný operátor, který má přehled o dění na letišti prostřednictvím kamer a senzorů. Operátorů může být více a jejich kanceláře jsou umístěné uvnitř letištní budovy.

1.1 Problematika

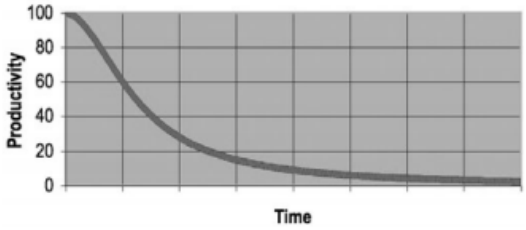
Problematika zahrnuje analýzu, která spočívá ve vybrání vhodného softwaru pro implementaci simáltoru, nastudování letištních budov, (použitých senzorů a fragmentů, které obsahuje každé letiště, případné hrozby, které mohou nastat na letišti) a poté praktickou implementaci a modelaci budovy letiště a okolí. Nezbytnou součástí vytvoření kvalitního softwaru je také samotné otestování naimplementovaných modulů a funkcionalit, kde ke kvalitnímu otestování je potřeba více testů. Je zapotřebí otestovat jednotlivé menší části kódu a i software jako celek, a vytvořit tedy testovací scénáře. Pro lepší kvalitu kódu je taktéž dobré zkontrolovat kód pomocí statické analýzy kódu.

Analýza obnáší navrhnutí potřebného modelu a vybrání vhodného softwaru pro implementaci samotné simulace a přenesení do světa VR. Je potřeba udělat analýzu všech možností a omezení konkrétních vývojových softwarů a modelovacích softwarů. Již existuje mnoho softwarů, které jsou vhodné pro

naše potřeby a tyto softwary používají např. herní studia. Implementace od samotného počátku (na zelené louce) by byla velice náročná a nereálná. Museli bychom implementovat detekci kolizí, 3D vykreslování, fyziku v simulaci a algoritmy pro hledání optimální cesty. Softwary, o kterých je řeč, už v sobě zahrnují moduly pro vykreslování 3D modelů, fyzikální chování a ostatní potřebné komponenty. Úkolem je tedy vybrat ten nejvhodnější. Také je zapotřebí udělat průzkum trhu a zvolit vhodný hardware pro testování a vývoj.

Praktická část obnáší implementaci funkcionalit (jednotlivých modulů) a naskriptování jednoduchých nepřátel, kteří budou simulovat útok na letiště. Moduly budou simulovat chování letištního provozu, cestujících, pokud útočník začne střílet (vzdávání se), nebo třeba zajišťovat chování cestujících ve frontách. Implementovat je potřeba i přenesení do světa VR, tedy zajistit sledování brýlí a případných ovladačů a provést optimalizaci pro VR.

Jak bylo již zmíněno, praktickou implementaci je potřeba i otestovat sadou vhodných testů. Je potřeba otestovat naskriptovné funkcionality a i celkové chování simulace a zvolit tedy komplexnější testy, které otestují návaznost chování jednotlivých funkcionalit. Testování je věnována značná část práce, jelikož od kvality testů se poté odráží četnost bugů a kvalita jejich opravy, efektivita psaní dalšího kódu a tedy rozšiřování simulace. Čím větší bude pokrytí testy, tím bude práce v budoucnu efektivnější a rychlejší.



Obrázek 1.1: Obrázek ukazující závislost produktivity na čase, pokud se kód neudržuje a netestuje [1].

1.2 Stav poznatků o řešené problematice

Současných softwarů umožňujících trénink výcvikových jednotek je mnoho, ale většinou jsou zpoplatněné a případné modifikace (např. úpravy pro letištní prostředí) stojí výraznou finanční sumu. Nebo společnosti software vůbec neposkytují pro veřejné účely a zdrojové kódy pro modifikace (nejsou open-source). Současně ale také můžeme říci, že téměř každá VR FPS hra je simulace reálných podmínek, tedy výcvikový trenažér s tím, že tam není

možné si upravovat prostředí a skriptovat konkrétní situace dle potřeby a požadavků.

Některé ze současných softwarů dokonce podporují použití makety pušky, které se sledují za pomoci externích zařízení a přenáší se do světa VR. Toto přináší lepší a reálnější pocit ze simulace. Jeden ze softwarů, který tyto technologie využívá je např. AsterionVR [8], který dokonce přenáší reálné překážky do VR světa a tím buduje reálnější pocit ze simulace.

Dále existuje celá řada simulátorů sloužících pro výcvik operátorů na letištních plochách. Jedním z příkladů může být RAMPVR [9], který právě tuto simulaci umožňuje. Avšak neumožňuje výcvik zásahových jednotek a simulaci případných útočníků.

1.3 Cíl

Cílem bylo vytvořit minimální životaschopný produkt (MVP), který bude v budoucnu rozšiřitelný o další moduly, funkcionality a 3D modely. Udělat stavební kámen pro budoucí diplomové práce, na kterém mohou budoucí studenti stavět a rozšiřovat výcvikový trenažér. Trenažér má obsahovat dvě části - taktickou část, která poběží ve virtuální realitě a operátora, který uvidí kamerový systém a výstup ze senzorů na monitoru počítače a bude komunikovat se zásahovou jednotkou pomocí externího softwaru. Průběh výcviku bude také možné měnit za běhu simulace, kde bude možné upravovat proměnné a tím ovlivňovat chod simulace. Lze např. měnit rychlost času, prodlevu mezi výstřelem útočníka, či upravovat příletový a odletový plán letadel.

Je nutné taktéž vytvořit vhodné prostředí pro simulaci, tedy letištní prostory a to jak vnitřní, tak venkovní. Celá simulace musí budit dojem reálného letištního prostředí a simulovat letištní provoz a dodržet základní principy fungování letištních budov.

Součástí je také vytvoření vhodného modelu, kde kostru letiště je potřeba vymodelovat a doplnit o jednoduché modely, volně dostupné na internetu. Z kostry letiště a stažených modelů je nutné vytvořit letištní prostory, na kterých se budou pohybovat letadla, cestující a zásahová jednotka. Do modelu budou vhodně umístění i pracovníci letiště.

Kapitola 2

Analýza problému

Analýza problému je nezbytná část, která se musí provést před samotnou implementací. Je potřeba proniknout do tématu a získat nezbytné informace a znalosti, které jsou poté použité v praktické implementační a modelovací části a následném otestování kódu.

Součástí analýzy, která je provedena před samotnou implementací, byla analýza současných herních enginů, modelovacích softwarů a získání základních znalostí o fungování letištních budov. Jako inspiraci můžeme použít velká herní studia jako např. Warhorse nebo 2K Czech, která však používají in-house enginy (vlastní naimplementované herní enginy) pro tvorbu her - naší simulaci můžeme zároveň nazývat hrou, avšak je koncipována jako simulace, jelikož v simulaci nejsou žádné životy, úkoly a podobně. Bylo tedy potřeba udělat analýzu trhu, které softwary případají v úvahu a podle základních bodů vybrat nejvhodnější software. Důležitými body při výběru herního enginu byly - komunita, podporované skriptovací jazyky, množství dostupných knihoven, sémantika a syntaxe skriptovacího jazyka.

Taktéž bylo nutné si udělat přehled o VR hardwaru a aktuálních trendech vývoje na tuto platformu [10]. Virtuální realita je poměrně nová technologie a směr vývoje se posunuje stále kupředu. Existuje celá řada domacích i zahraničních konferencí, kde na toto téma přednášejí odborníci a prezenzují se nově vytvořené technologie a softwary, případně frameworky pro vývojáře.

2.1 Základní pojmy

Herní engine - software implementovaný pro vytváření 3D nebo 2D her a simulací. Zajišťuje 3D vykreslování, má komponentu pro nastavení fyziky a ostatní komponenty potřebné pro vytvoření simulace nebo hry.
VR - cílem virtuální reality je přenést uživatele do fiktivního světa. To lze prostřednictvím virtuálních brýlí, které jsou složené ze dvou displejů např. AMOLED displeje, čoček a případných sluchátek. Pohybovat se ve virtuálním světe je možné prostřednictvím ovladačů nebo gamepadu a nebo chůzí v prostoru [11]. Existují i pokročilejší technologie, které umožnují nekonečný pohyb v prostoru, tedy uživatel není limitován fyzickým prostorem.
AR - AR je technologie, která do reálného světa přenáší digitální prvky. Např. 3D modely. Pro AR je potřeba kamera, která snímá okolí a na základě nějakých značek nebo významných bodů detekuje objekt, na který má vykreslit požadovaný 3D model.
Motion sickness - fyzický stav jedince, kdy vjem pohybu ve virtuálním světě nesouhlasí s pohybem v reálném světě. Např. ve virtuálním světě je možné se hýbat za pomoci joysticků, avšak v reálném světě stojíte na místě. Toto může zapříčinit motion sickness. Případně na co je také nutné si dávat pozor, je padání z výšek, které může také přivodit nevolnost. Nevhodný návrh může způsobovat velké nevolnosti a tím nemožnost používat simulátor delší dobu.
Garbage collection - způsob správy paměti. Algoritmus zvaný *garbage collector* se stará o uvolňování nevyužité paměti. Nevyužitou část paměti pozná podle toho, že na ní nevede žádný ukazatel. Šetří čas a počet řádků kódu při vývoji, ale může zpomalovat běh programu. Důvodem zpomalování programu je, že na pozadí musí běžet proces, který tuto kontrolu dělá a tím může vytěžovat procesor a paměť.
Blueprints - Vizuální skriptovací systém v herním enginu Unreal engine [12].
FPS - tedy počet snímků za sekundu. Čím nižší, tím je hra hůř optimalizována a ubírá to poté na zážitku a ovladatelnosti. Pokud je software špatně optimalizovaný může způsobovat i nevolnosti, bolení hlavy a očí.
VR Headset - brýle pro virtuální realitu. Headset může také obsahovat přídavná sluchátka. Headsetů je celá řada a liší se výrobce (HTC, Oculus, Google), ale také parametry jako např. rozlišení, obnovovací frekvence, atd. Headset může být drátový nebo také bezdrátový. Headsety se také mohou lišit podle toho, pro jakou platformu jsou určené. Máme headsety pro konzole (Xbox, Playstation, Nintendo), headsety pro PC a headsety pro mobilní zařízení.
UI - Tedy uživatelské rozhraní.
HUD - Rozhraní ukazující informace o hře/simulaci, nebo třeba menu umožňující vypnutí, restartování hry/simulace a nebo proklik do nastavení.

2.2 Herní enginy

Ve výběru herního enginu figurovaly tyto softwary:

- Unity
  - Skripty psány v jazyce C# a JavaScript.
  - Dobrá volba pro mobilní VR a AR.
  - Unity je zdarma pro výukové účely.
  - Používá garbage collection, který může způsobovat výkonnostní problémy.
  - Více dostupných knihoven (široká komunika poskytující balíčky a již hotové funkcionality volně ke stažení nebo případně za nějakou finanční sumu).
  - Cross-platform - naimplementovaný software je poté možné sestavit pro více platforem, bez nutnosti větší modifikace kódu (toto nebude potřeba, trenažér bude pouze pro Windows). Je možné taktéž vzít kód a sestavit program pro Mac a nebo Linux operační systém.
  - Obrovská komunita a fórum bohaté na různé rady při vývoji.
  - Obrovské fórum, kde lze najít nespočet způsobů řešení chyb, které mohou nastat při vývoji, případně již navrhnuté řešení konkrétního problému.
- Unreal Engine
  - Pro AAA hry.
  - Skripty psány v jazyce C++.
  - Je možné upravovat přímo zdrojové kódy herního enginu.
  - Lepší pro graficky náročné hry.
  - Herní engine je zdarma.
  - Lepší profilování než v Unity.
  - Horší pro 2D hry.
  - Silná komunita (ale slabší než v Unity).
  - Málo návodů pro VR a C++.
  - Většina návodů pro implementaci VR her/simulací je pro Blueprints vizuální skriptovací jazyk.
- CryEngine
  - Skripty psány v jazycích C#, C++ nebo LUA.
  - Slabší vývojářská komunita oproti Unreal enginu a Unity.
  - Velký nedostatek návodů pro vývoj VR her/simulací.

Jasná volba je tedy **Unity**, díky své podpoře VR a obrovské komunitě. Unity taktéž plní všechny naše potřeby (3D, podpora VR). Obsahuje i množství komponent, které nám usnadní vývoj. Komponenty jsou popsané níže.

Unity je multiplatformní herní engine, použitý pro vývoj simulace letiště. Engine je vytvořen společností Unity Technologies. Lze ho použít pro vývoj 3D i 2D her pro PC, konzole, mobily a weby. Unity engine vznikl v roce 2005. Obsahuje grafické prostředí pro tvorbu a podporuje skriptování. Unity je jedním z nejpopulárnějších herních enginů, je používán jak vývojáři, kteří mají vývoj her/simulací jako hobby, tak i profesionálními a velkými herními studii. Unity je populární hlavně kvůli skriptovacím jazykům, velmi dobrým animačním nástrojům a podpoře jak 2D, tak 3D [13].

Simulace je cílena primárně na platformu Windows. Pro skriptování byl použit jazyk C#, který díky své dobré sémantice je jednoduchý na naučení a syntaxticky podobný Jave (jazyk je dokonce založený na jazyce C++ firmou Microsoft). C# je objektově orientovaný překládaný jazyk, vyvinutý společností Microsoft. Jazyk C# je také možné použít pro tvorbu databázových systémů, webových aplikací a tvorbu mobilních aplikací.

Unity základní komponenty:

- **Vykreslovací engine**
  Komponenta zodpovědná za vykreslování (renderování) reálně vypadajících obrazů (buď 3D nebo 2D). Data, která má komponenta vykreslovat poskytuje grafická karta (tvar, světlo, barva a podobně). Unity taktéž poskytuje možnost proces vykreslování ovládat za pomocí skriptů.
- **Fyzikální engine**
  Fyzikální komponenta je zodpovědná za to, že se objekty ve scéně chovají podle fyzikálních zákonů. Působí na ně gravitace, pokud se střetnou s jiným objektem, tak se s nějakou určitou silou odrazí, případně s jakou a do jaké strany a podobně. Komponenta zodpovědná za fyzikální chování se jmenuje *Rigidbody*. Fyzikální engine zároveň vyhodnocuje i kolizi dvou objektů.
- **Skriptování**
  Tato komponenta je zodpovědná za logiku hry. Pomocí herních skriptů se implementuje herní logika a chování objektů. Jazyk podporovaný pro skriptování v Unity je C# a JavaScript. Skripty je taktéž možné použít pro ovlivňování ostatních komponent. Např. můžeme pomocí skriptu vypínat a zapínat gravitaci nebo měnit texturu objektu.
- **Zvukový engine**
  Zvuková komponenta je zodpovědná za načítání, a přehrávání zvukových

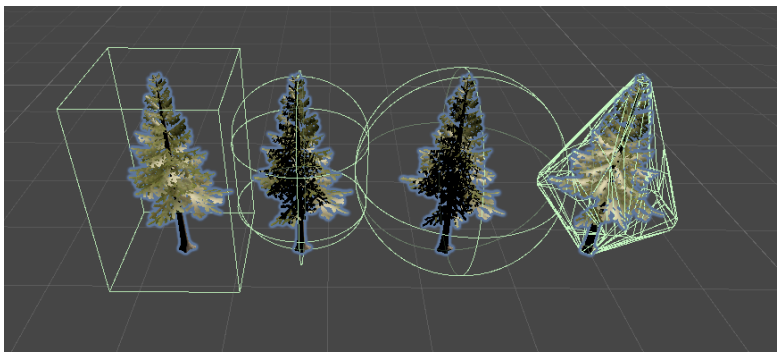
efektů či stop. Zvuk je možné různě mixovat a upravovat jeho šíření v prostoru, vytvářet smyčky, ovlivnit odrážení zvuku, nastavovat hlasitost, prioritu a podobně.

- **Další komponenty**
  Dalšími komponenty jsou např. komponenty zajišťující síťování, správu paměti, animace a hledání optimální cesty (komponenty pro animace a hledání optimální cesty jsou popsány v kapitole níže).

Nezbytnou součástí simulace jsou kolidery. Kolider detekuje kolizi s jiným koliderem a spouští událost, kterou je poté možné odchytit a podle konkrétní události jednat. Např. pokud kulka zasáhne cestujícího, tak se střetnou dva kolidery a je možné vyvolat animaci úmrtí. Kolize počítá fyzikální engine. Obecně platí, že čím jednodušší tvar, tím je kolize počítá jednodušeji a rychleji a tím méně vytěžuje výpočetní jednotku, tedy procesor.

Typy 3D koliderů v Unity:

- Box Collider - má tvar krychle nebo kvádru, nastavujeme šířku, výšku a délku.
- Sphere Collider - má tvar koule, můžeme nastavovat rádius.
- Capsule Collider - má tvar kapsule, můžeme měnit výšku a radius - tedy poloměr.
- Mesh Collider - nejkomplexnější typ kolideru, který obalí objekt koliderem, který má komplexnější tvar a přizpůsobí se tvaru objektu. Je nejsložitější na výpočet kolize a má značný vliv na výkon simulace/hry.
- Terrain Collider - používá se jako kolider pro terén, tedy jako podlaha, aby objekty, které mají nastavenou gravitaci nepropadávaly podlahou.



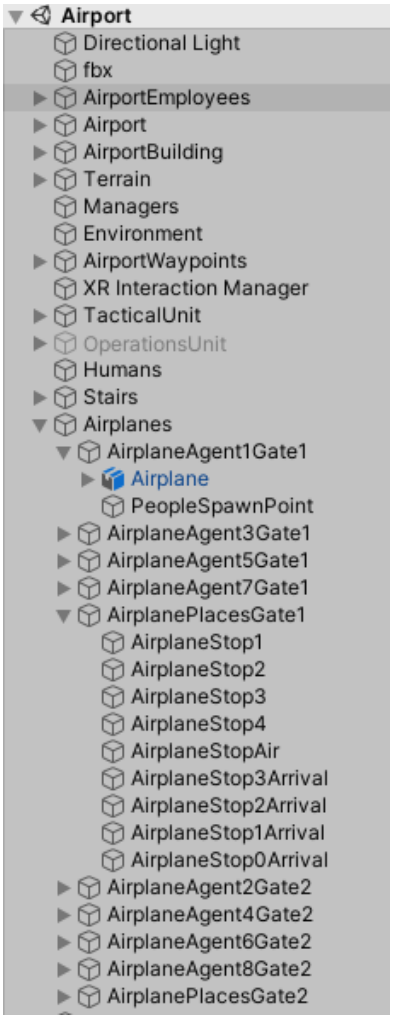
Obrázek 2.1: Ukázka koliderů v Unity.

V simulaci je používán výhradně *BoxCollider*, který je výpočetně nejjednodušší a tedy má nejmenší vliv na snížení FPS ve hře. Unity taktéž obsahuje

speciální kolidery pro 2D objekty, což v této práci není relevantní, jelikož jde o 3D simulaci, tudíž nebyly zde ani popsány. Kolidery je možné dle potřeby přizpůsobovat danému objektu - posouvat, otáček a nastavovat offset, vůči středu objektu. Kolidery jsou základním prvkem, který spouští události. Pokud přijde cestující do budovy, projde neviditelným koliderem, který má vypnuté fyzikální chování, tudíž se detekuje střet koliderů, ale nepůsobí na ně fyzikální síly a tento střet vyvolá událost příchodu cestujícího do haly. Dalším příkladem je třeba odbavovací přepážka. Detekce odchodu cestujícího z odbavovací přepážky se dělá opět pomocí neviditelného kolideru, který spustí událost opuštění odbavovací přepážky. Dalším příkladem může být, pokud si chceme u cestujících držet informaci o tom, na kterém patře se nacházejí, tak můžeme dát na první patro neviditelné kolidery bez fyzikálního chování, a pokud cestující s ním koliduje, tak se vyvolá událost a cestující si uloží stav, že přešel do prvního patra.

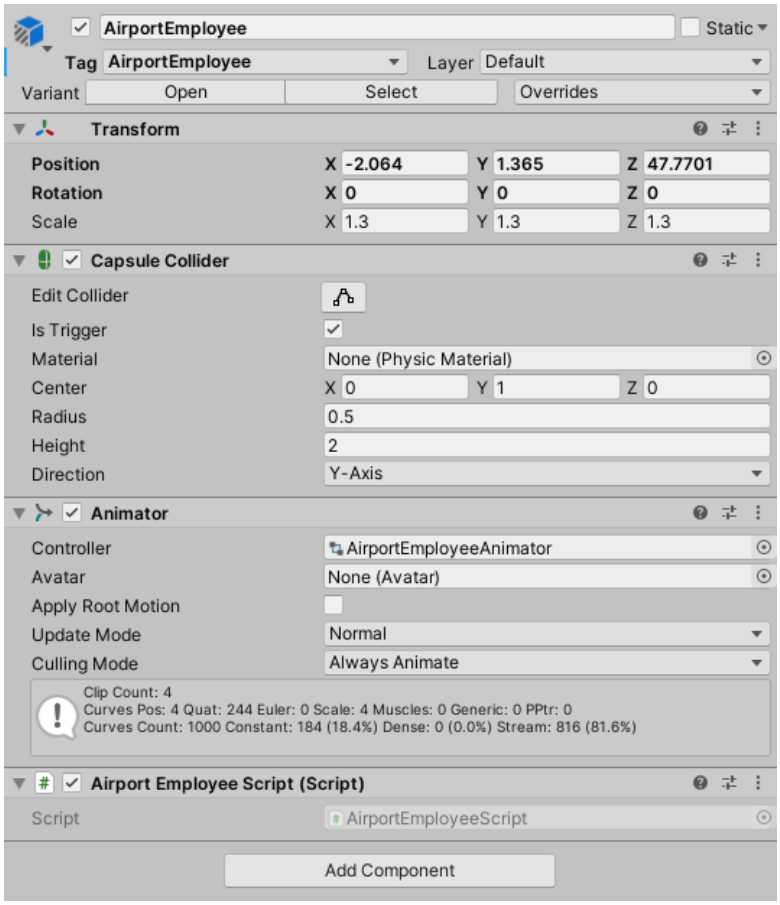
Unity se skládá ze scén. Scéna obsahuje tzv. *GameObjecty*, které jsou uspořádány v orientovaném acyklickém grafu (graf, který neobsahuje žádný cyklus). Objekty je tedy možné vnořovat do sebe. Objekty je možné vytvářet a přidávat staticky nebo dynamicky pomocí skriptů. Pro naší simulaci jsou použity jak staticky vytvářené objekty, tak dynamické objekty generované při běhu simulace. Existují ještě speciální typy objektů, které se např. při přepnutí scény nezničí a neinicializují se znovu, ale převezme se instance objektu z předchozí scény. Scéna může obsahovat libovolný počet objektů a tyto objekty mohou mít uvniř libovolný počet vnořených objektů. Počet objektů ve scéně má vliv na výkonnost simulace (počet FPS). Proto je nezbytné se snažit počet objektů ve scéně minimalizovat, a pokud např. zásahová jednotka nebo nepřítel vystřelí a vyletí kulka, tuto kulku je potřeba po určité době zničit, protože by ve scéně byla zbytečně a jen brala výkon počítače při vykreslování.

Scény je možné za pomocí skriptů přepínat a měnit dle potřeby. Rychlost přepínání scén je závislá na počtu statických objektů ve scéně, tedy čím víc statických objektů, tedy objektů které se vytvoří při zapnutí scény, tím déle bude trvat načítání scény. Při samotném vývoj je za pomocí Unity UI ve scéně možné objekty vyhledávat a tím usnadňovat práci s enginem. Vyhledávat můžeme buď pomocí jména objektu a nebo zadat jméno komponenty a zobrazí se objekty obsahující tuto komponentu. Jednotlivé objekty ve scéně jsou poté umístěné na příslušné souřadnice a mají příslušné komponenty (dle typu objektu a jeho požadovaného chování).



Obrázek 2.2: Příklad GameObjectů ve scéně.

Jednotlivé GameObjecty poté obsahují komponenty, které definují chování GameObjectu. Zde je poté možné přidat např. komponentu pro definici fyzikálního chování (Rigidbody) nebo skript, který bude popisovat a zajišťovat chování objektu. Dalším příkladem je, pokud připnu komponentu BoxCollider (která detekuje kolizi s jiným koliderem), tak ve skriptu můžeme tuto událost odchytit a vynutit případnou reakci. Skript, který je připnutý jako komponenta, může hledat komponenty na jiných objektech nebo na sobě a tyto komponenty za běhu upravovat. Např. pokud potřebuji za běhu programu vypnout komponentu, která zajišťuje kolize s jinými objekty, můžu si tuto komponentu za pomoci skriptu najít a vypnout, tedy nastavit jí jako neaktivní, nebo přímo komponentu odstranit z objektu. Další možností práce s komponentami přes skripty je upravit atributy komponenty. Měnit můžeme třeba atributy fyzikální komponenty jako jsou gravitace, nebo hmotnost objektu, měnit souřadnice objektu za běhu programu a podobně.



Obrázek 2.3: Příklad komponent u GameObjectu.

Na obrázku 2.3 je možné vidět připnuté komponenty ke GameObjectu. Jde o komponentu tzv. *Transform*, která zajišťuje pozici, rotaci a měřítko ve scéně (relativně vůči rodičovskému objektu a nebo absolutně ke středu scény - v závislosti na umístění objektu ve scéně). Dále komponentu *Capsule Collider*, která je zodpovědná za kolize s jinými kolidery. Zde je vidět i příklad toho, že kolideru můžeme měnit pozici, a to radius a výšku a měnit offset od středu objektu. Dále *Animator* jako stavový automat zajišťuje animování objektu a přechod mezi jednotlivými animacemi. A v neposlední řadě skript, zajišťující logiku a chování ve scéně. Pokud má skript věřejné proměnné, tak se promítnou zde a můžeme je měnit za běhu programu nebo před samotným spuštěním.

Při vývoji ve scéne se některé komponenty pro lepší lazení simulace promítají, ale v samotné hře jsou neviditelné. Můžeme se tedy přepnout do vývojového módu a sledovat, jak se komponenty mění a kolidery protínají. Pro lepší přehlednost ve scéně se kolidery zobrazí, až když zvolíme konkrétní objekt, je však možné zvolit všechny objekty a tím si zobrazit všechny kolidery ve scéně.

Obrázek 2.4: Příklad zobrazení komponenty *Box Collider* při běhu simulace.

Na obrázku 2.4 můžeme vidět simulaci ve vývojovém režimu při jejím běhu. Můžeme se podívat, jak je kolider umístěný a jestli koliduje s jiným koliderem jiného objektu. Pokud se přepneme do módu hry, tak tyto kolidery a souřadnicový systém již neuvidíme. Toto zobrazení výrazně ulehčuje vývoj a lazení softwaru. Kolider není jediná komponenta, která se do vývojového módu scény promítá, můžeme si promítnout např. komponentu *Nav Mesh Obstacle*, která definuje překážku při pohybu cestujících (popsána níže).

Postrannímu menu, kam je možné připínat příslušné komponenty, se také říká *Unity Inspector*. Unity Inspector slouží ke konfiguraci objektu, tedy ke konfiguraci jeho pozice a chování. Můžeme nastavovat vrstvu, ve které se objekt nachází nebo přiřadit příslušný tag a nebo upravit jméno objektu.

Skripty psané v jazyce C#, které mohou být připnuté ke GameObjectu, musejí mít následující kostru, v které musejí využívat jmenný prostor *UnityEngine* a *System.Collections*. Jmenné prostory umožňují poté používat třídy z tohoto jmenného prostoru. Dále skript musí být odvozený od třídy *MonoBehaviour* a implementovat metody *Start* a *Update*. Tyto metody jsou poté vyvolané dle životního cyklu objektu. Tato kostra se automaticky vygeneruje při vytvoření skriptu přímo přes Unity:

```
using UnityEngine;
using System.Collections;

public class MyCustomScript : MonoBehaviour {

// Use this for initialization
void Start () {

}

// Update is called once per frame
void Update () {

}
}
```

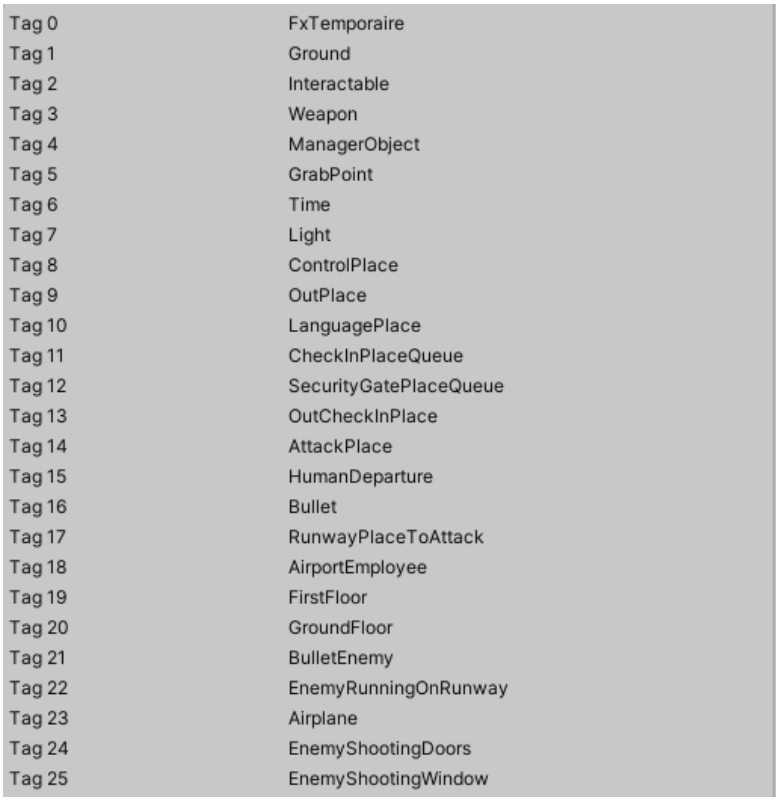
Jak je ve skriptu popsáno, tak metoda *Start* je volána při vytváření objektu ve scéně (pokud jde o dynamicky vytvořený objekt za pomocí skriptu), nebo po spuštění hry (pokud se jedná o staticky vytvořený objekt) a metoda *Update* je volána jednou za snímek. Do skriptu je možné přidat metody jako *Awake* - tato metoda se volá ještě před samotnou inicializací objektu a nebo lze přidat také metodu *FixedUpdate*.

Rozdíl mezi metodou *FixedUpdate* a *Update* je ten, že metoda *Update* se volá jednou za snímek, zatímco *FixedUpdate*, se může volat vícekrát, je volána v pravidelných intervalech, narozdíl od *Update*, který je závislý na FPS. V metodě *Start* se většinou dělá prvotní inicializace. Pokud objekt bude pracovat s jinými objekty, zde si je najde a uloží si jejich instance, nebo pokud bude pracovat s nějakou svojí komponentou, zde si jí najde a uloží pro další práci. Metoda *Update* se používá např. na odchytávání stlačení tlačítka, kdy v každém snímku kontrolujeme jeho stav, nebo zde můžeme počítat aktuální herní čas a nebo sledovat stav objektu a měnit jeho chování. Vyhledávání objektů a jejich případných komponent je nevhodné dělat v metodě *Update*, jelikož bychom zabírali výkon tím, že budeme v každém snímku prohledávat scénu s objekty a ukládat si komponenty.

Jednotlivým GameObjectum můžeme také přidávat tzn. **tag**, podle kterých je poté možné příkazem v C#

```
GameObject.FindGameObjectsWithTag("tag");
GameObject.FindGameObjectWithTag("tag");
```

hledat GameObjecty a interagovat s nimi. Můžeme si všechna letadla označit tagem *Airplane*, a potom pokud je potřeba, je možné tyto GameObjekty pomocí tagu najít a pracovat s nimi. Pokud bychom procházeli scénu přes všechny objekty, s rostoucím počtem GameObjektů ve scéně by se snižovaly FPS, protože by skripty procházely všechny GameObjekty a hledaly ty námi vyhledávané, což je časově náročné. Jednotlivé tagy se registrují do tabulky tagů. U jednotlivých GameObjektů je poté uložený tag jako integer. Pokud hledáme objekt, tak algoritmus prochází všechny objekty a porovnává integer hodnoty a pokud najde shodu, tak víme, že jsme našli objekt s požadovaným tagem. Pokud chceme získat tag z objektu, tak se nejdříve podíváme do tabulky a na základě int hodnoty vybereme příslušnou hodnotu typu string. Při hledání tagu se tedy porovnávají integer proměnná, namísto string proměnných (pokud bychom hledali GameObjekt podle jeho jména), což je výrazně rychlejší.



Obrázek 2.5: Ukázka tagů v Unity.

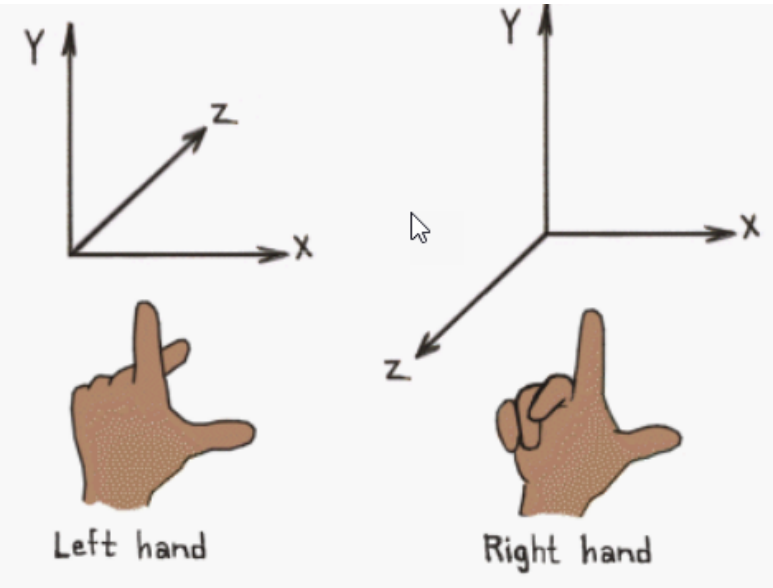
Poté pomocí příkazu:

```
gameObject.GetComponent<ComponentType>();
```

můžeme z vyhledaných GameObjektu vyselektovat jen určité komponenty

a s těmi poté pracovat - upravovat její proměnné, tím upravovat její nastavení, nebo vynutit některou událost, odstranit komponentu nebo jí zneaktivnit. GameObject může mít pouze jeden tag, není tedy možné přiřadit jednomu GameObjectu tag *Airplane* a zároveň *Weapon*.

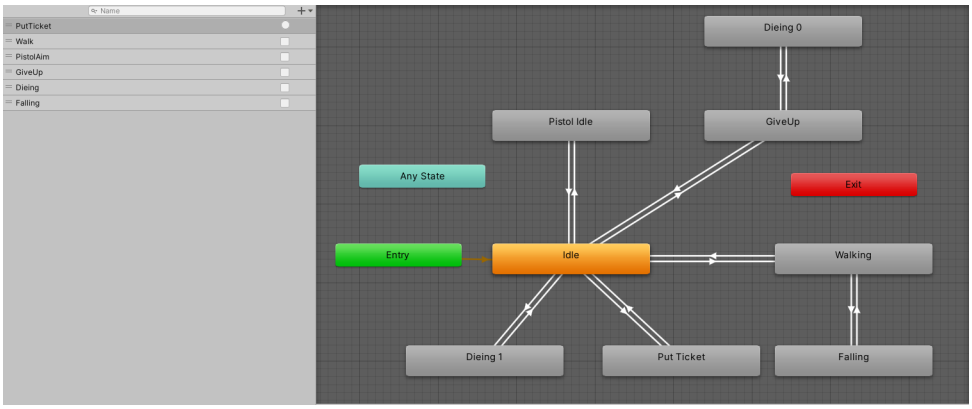
Komponenta zvaná *Transform* definuje pozici, rotaci a měřítko ve scéně. Může definovat rotaci, měřítko a pozici relativně vůči rodičovskému GameObjectu a nebo absolutně a tedy ke středu scény. Unity využívá levotočivý kartezský souřadnicový systém - LHS.



Obrázek 2.6: Levotočivý souřadnicový systém vs pravotočivý souřadnicový systém [2].

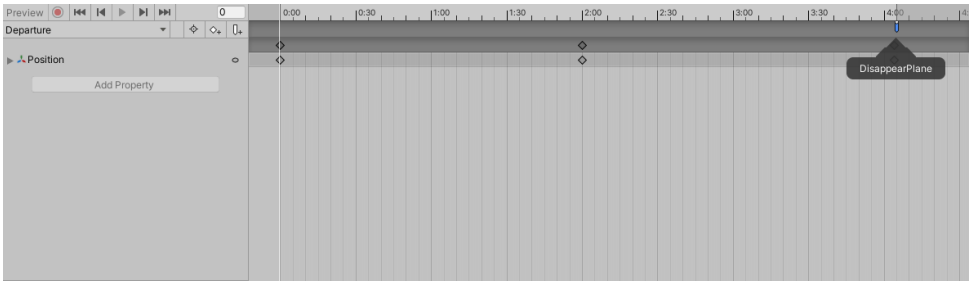
Pro animace jednotlivých postav (zaměstnanec letiště, cestující a případně přílety a odlety letadel), je použita komponenta *Animator*, která je nativně součástí v Unity. Animator se chová jako stavový automat, kdy se animace mění na základě proměnných. Můžeme definovat i přechody animací, tedy jak na sebe animace budou navazovat, případně zda se přepnou okamžitě. Stavy animatoru můžeme měnit za pomoci různých proměnných. Můžeme spouštět tzn. *Trigger*, který přepne animator z jednoho stavu do druhého. Můžeme mít proměnnou která nabývá hodnoty pravda, nepravda a na základě toho se přepíná stav nebo můžeme na základě *int* a *float* proměnných měnit stav podle hodnot, kterých nabývají. Toto se hodí pro animace chůze a běhu. Můžeme říct, že pokud se objekt pohybuje rychlostí 1, tak běží a jakmile spadne pod hranici 0,5, tak objekt jde a přepne se tedy animace z běhu na chůzi. Animatory jsou poskládané z animací vytvořených webovou aplikací *Mixamo* [14]. Jde o humanoidní animace pro zaměstnance letiště a cestující. Animace pro letadla jsou animace ručně vytvořené, které mění pouze pozici na x,y,z osách jednotlivých letadel. Animace mohou měnit i rotaci a měřítko objektu. Na vytvořené animace je poté možné připnout událost. Pokud tedy

animace začně, můžeme vyvolat událost a ve skriptu odchytávat a nebo pokud skončí, tak odchytit událost ve skriptu.



Obrázek 2.7: Příklad animatoru.

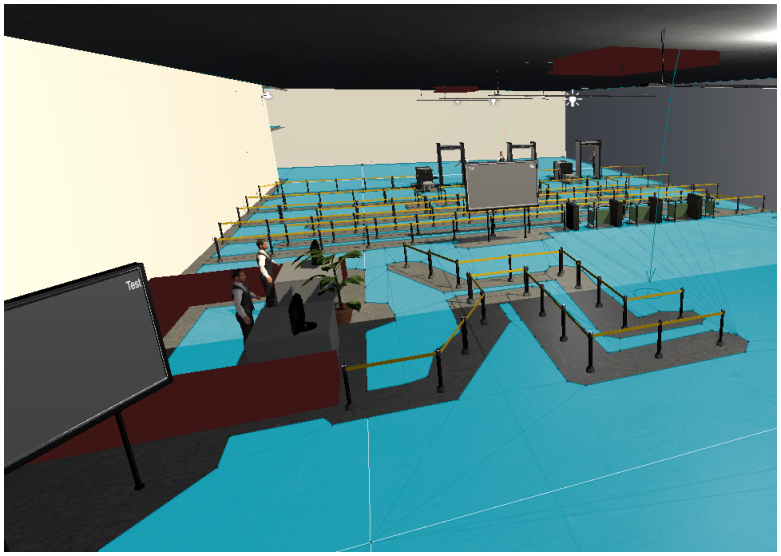
Na obrázku 2.7 lze vidět, že *Animátor* má výchozí stav, ve kterém po vytvoření objektu začíná (oranžový stav) a poté jsou definované přechody, které jsou znázorněny šipkami. Animátor začíná ve stavu *Entry* a přechází do výchozího stavu, který může být pouze jeden. V levé postranním menu poté můžeme vidět definované proměnné, které lze za pomocí skriptu měnit a tím zajistit přechody mezi animacemi.



Obrázek 2.8: Příklad animace a možnosti upravování animace.

Na obrázku 2.8 můžeme vidět příklad animace, kde můžeme nastavit pozici v určitý čas, např. 2 sekundy po startu animace a Unity se již postará o kontinuální přechod mezi pozicemi. Takto můžeme měnit jak pozici, tak i rotaci nebo měřítko. V konkrétním případu animace je ukázaný i spouštěč, který zavolá konkrétní metodu na odchycení události.

Pro navigaci objektů byla použita nativní komponenta Unity **NavMesh**. NavMesh je jednoduchá komponenta, kde se definují jednotlivé objekty podle toho, do jaké oblasti patří (v našem případě jsou jen dvě - walkable, not walkable). Takto je potřeba definovat všechny objekty ve scéně a následně se vytvoří *Mesh*.



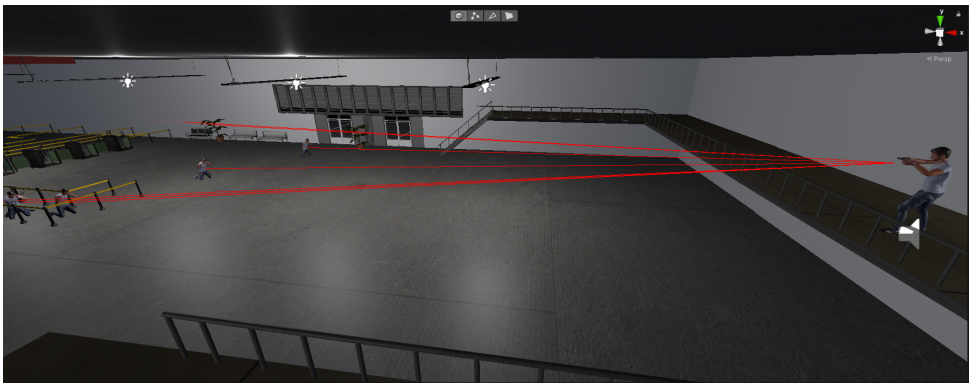
Obrázek 2.9: Příklad NavMesh prostoru.

Jak jde vidět na příkladu na obrázku 2.9, tak se vytvořil *Mesh*, tedy prostor, po kterém se může pohybovat tzv. *Nav Mesh Agent*. *Nav Mesh Agent* má definované základní atributy, tedy výšku, šířku, rychlost a prioritu (je možné také definovat výšku kroku, pokud např. jde po schodech a spoustu dalších atributů, které v našem případě nebude potřebné konfigurovat). Dle daných parametrů, se poté vytvoří *NavMesh* pro agenta. Poté se agentovi nastavuje pouze cíl a on sám volí nejkratší cestu. Cestu volí v rámci modrého pole, které lze vidět na obrázku 2.9. Překážky můžeme agentovi generovat staticky (nastavíme objekt jako not walkable) a nebo dynamicky. Pro dynamické překážky slouží komponenta *Nav Mesh Obstacle*, která vytvoří překážku, přes kterou agent nepřejde.

Na hledání nejkratší cesty v meshy používá Unity algoritmus *A**. Mesh se skládá z konvexní polygonů. Unity si ukládá hranice jednotlivých polygonů a také jejich sousedy. Poté procházíme jednotlivé polygony od počátečního po koncový (lze si představit jako graf) a za pomocí *A star* algoritmu Unity hledá nejkratší cestu [15]. A* můžeme přirovnat k prohledávání do šířky (BFS) nebo prohledávání do hloubky (DFS), s tím, že A* používá hladový princip. Na základě heuristiky určí další uzel kudy se má vydat (podle heuristiky vyhodnotí nejlepší uzel) a takto postupuje dokud se nedostane do cíle. Jako heuristiku je možné si představit např. vzdálenost, kterou již algoritmus prošel + cesta vzdušnou čarou do cíle přes všechny vrcholy, a jako další vrchol se tedy bere ten, u kterého vyjde hodnota nejmenší.

Pro lepší a pohodlnější debugování kódu je možné použít tzn. *Raycasty*. Raycasty umožňují promítnout do scény paprsek, který můžeme chápat jako vektor, který má počátek, směr a délku. Tímto detekujeme, že zásahová

jednotka míří na útočníka. V simulaci se tyto raycasty nepromítnou, můžeme je vidět pouze ve vývojovém módu scény. Skripty umožňují vypisovat na standartní výstup do konzole pro debugování hodnot proměnných a stavů.



Obrázek 2.10: Příklad raycastů

Poslední částí Unity je optimalizace. Pro optimální chod simulace (počet FPS je přibližně 30-60) je nutné dbát na jistá pravidla při implementaci. Je nutné vše, co je možné dát pryč z metod *Update*, aby se v každém snímku neprováděly zbytečné operace. Dále je nezbytné volit správný typ koliderů, tedy nevyužívat Mesh Collidery, pokud to není nutné. Pro teoretické nastudování bylo čerpáno z knih *Pro Unity Game Development with C#* [16], *Learning C# Programming with Unity 3D* [17] a *Learn Unity by Creating a 3D Multi-Level Platformer Game* [18].

2.3 Letištní budova

Letištní budova je kocipována dle vlastního návrhu. Do úvahy jsem vzal i možnost modelovat část letiště v Praze, nebo jiné mezinárodní letiště, ale bohužel není veřejný detailní plán pro žádné letiště nebo detailní fotografie, podle kterých by bylo možné v návrhu postupovat. Dalším rizikem, které by zde vznikalo, by byla možnost zneužití konkrétních plánů letiště. Bylo tedy třeba postupovat dle vlastní fantazie a vytvořit si návrh letiště. Na letištích je řada fragmentů (částí budovy), které se opakují a jsou na všech světových letištích. Tyto prvky však mohou být různě zakomponovány do celkové budovy letiště. Konkrétně se jedná o tyto prvky:

2.3.1 Fragmenty letištních budov

- Terminálová hala přílety/odlety

  Hala je velký vnitřní prostor s větší koncentrací osob. Možné jsou zde ochozy, vysoké stropy a odbavovací přepážky (check-in). V odletové hale se může také nacházet bezpečnostní kontrola, dále řada obchodů, a služeb - toto není z hlediska bezpečnosti tak důležité v našem návrhu. Do letištní haly se mohou dostat i osoby, které nikam necestují a jdou pouze do obchodu nebo narušovat bezpečnost letiště.

- Letištní galerie

  Galerie je ochoz vyvýšený nad úroveň patra v prostorách letištních hal. Galerie je přístupná všem lidem ve veřejném prostoru, kde se pohybují nezkontrolovaní lidé. V modelu byla zahrnuta galerie v terminálové odletové hale. Galerie se však může nacházet i v příletové hale nebo kdekoliv v prostorách letištní budovy.

- Letištní prsty

  Letištní prsty jsou koridory s odbočkami do jednotlivých gatů. Do tohoto prostoru je možné se dostat po odbavení a bezpečnostní kontrole. Letištní prsty mohou a nemusí obsahovat obchody a jiné služby poskytované cestujícím.

- Letištní brány

  Místa případné bezpečnostní kontroly (pokud se jedná o decentralizovanou letištní kontrolu) a odbavovacích přepážek. Z letištní brány cestující postupuje do nástupních mostů a do letadla. Letištní brány mohou a nemusí obsahovat bezpečnostní kontrolu. Záleží, zda jde o terminál s centralizovanou, či decentralizovanou kontrolou.

- Celniště

  Místa pro vyzvednutí zavazadel, určené k celním úkonům. Není potřeba je v modelu řešit. Zde není riziko nějakého útoku, jediné riziko které tu hrozí, je nebezpečný náklad v zavazadlech.

- Odbavovací stanoviště bezpečnostní kontroly

  Odbavovací stanoviště jsou dvou typů:

  - Centralizovaná kontrola
    Kontrola probíhá na jednom centrálním místě, kde poté cestující postupují na příslušné brány, dle svého letu. Nevýhoda je, že se koncentruje velké množství osob na jednom místě, kde se provádí kontrola. Ale zmenšují se tím prostory, kde jsou nezkontrolovaní cestující.
  - Decentralizovaná kontrola
    Kontrola probíhá na příslušných letištních branách nebo prstech. Kontrola se může lišit podle cílové destinace. Výhodou je, že se

netvoří taková koncentrace osob jako u centralizované kontroly, avšak zvětšují se prostory, kde se pohybují nezkontrolovaní cestující.

- Odbavovací stanoviště ostrahy letiště

  Jde o kontrolu stejnou jako u cestujících, zde se však jedná o kontrolu ostrahy letiště.

- Nástupní mosty

  Prostor připojený k letadlu, kudy se cestující dostanou do letadla. Může být včetně "chobotu"nebo případně rotundy. Rotunda představuje bezpečnostní riziko, pokud cestující vnikne neoprávněně na venkovní prostory letiště, a z nich může proniknout do letadla a ohrožovat cestující.

- Letadlo

  Zde už jsou odbavení, zkontrolovaní cestující, kteří čekají na vzlet na svých místech. Mohou být ohrožení vniknutím útočníka na vzletovou plochu a nebo přes rotundu přímo do letadla.

- Kolektory a produktovody

  Podzemní šachty, vstupy často s žebříky do hloubky 3 a více metrů, šachty v délce stovek metrů, špatné světelné podmínky, stísněné prostory, někde víceúrovňové. V modelu tato část nebyla zahrnuta. Vniknutí do těchto prostorů je nepravděpodobné.

- Kancelář

  Běžný kancelářský prostor, kde sedí administrativní pracovníci a operátoři, kteří hlídají bezpečnost. V simulaci je tento prostor namodelován a představuje prostor, kde se nacházejí monitory, které ukazují výstup ze senzorů a přímí přenos bezpečnostních kamer.

2.3.2 Typy senzorů

Na letišti je celá řada senzorů, které snímají pohyb na venkovní i vnitřní ploše. Přístup k výstupu ze senzorů mají bezpečnostní operátoři, kteří jsou upozorněni na případné nebezpečí, které musejí řešit. Např. dát vědět zásahové jednotce, aby hrozbu prověřila, nebo obvolat příslušné jednotky, jako třeba hasiče a policii nebo záchrannou službu. Na základě senzorů musejí být schopni správně vyhodnotit situaci a riziko a vědět, jak správně postupovat dále. Informace od bezpečnostního operátora je klíčová pro zásahovou jednotku.

- Pohybové senzory

  Pohybové senzory snímají pohyb přes pomyslnou hranici. Pokud někdo přejde přes prostor, kde není přístup povolen, operátor uvidí výstrahu a

jedná. Pohybovými senzory není pokryt celý prostor, jen vybrané části letiště.

- Kamerový systém

  Kamerami jsou pokryté celé letištní prostory. Jak vnitřní, tak venkovní. Operátor má přístup k záznamům a k živému přenosu, kde může sledovat dění a informovat zásahovou jednotku o případné hrozbě.

- Senzory pro neoprávněné přístupy

  Některé dveře na budově jsou osazeny senzory, které snímají vstupy. Pokud pracovník projde dveřmi a ověří se kartou, operátor toto uvidí na obrazovce. Pokud se pracovník neúspěšně ověří, uvidí toto operátor také. Pokud se osoba, které není přístup určen, pokusí přes dveře dostat, operátor si všimne upozornění a může o tom informovat a jednat dále. Ne všechny dveře v budově jsou osazeny tímto senzorem. Např. toalety není potřeba monitorovat. Dveře, kam běžní cestující nemají přístup, jako např. do kanceláří, jsou osazeny tímto senzorem.

- Senzory snímající integritu pláště

  Tyto senzory snímají, zda nebyl narušen pomyslný plášť budovy, např. zda není rozbité okno.

2.3.3 Typy útoků

- Narušení integrity pláště

  Útočník naruší integritu pláště budovy (rozbije okno a zeď) a vnikne neoprávněně do budovy. Narušení integrity pláště může být provedeno více způsoby. Např. výbuchem bomby nebo účtoník manuálně rozbije okno.

- Útok z galerie veřejného prostoru

  Útočník přijde ozbrojený do veřejného prostoru na galerii a ohrožuje cestující zbraní. Po určité době začne střílet, a to jak na cestující, tak na zásahovou jednotku. Útočník může jednat různými způsoby, zde nelze uvést jeden přesný příklad. Útočník může zbraní jen ohrožovat, a nebo střílet, může držet rukojmí nebo také nemusí. Taktéž může použít různé zbraně, jako např. nůž nebo pistoli.

- Vniknutí na runway

  Útočník přeleze plot a vnikne na venkovní plochu, kde ohrožuje provoz na ploše. Zde hrozí srážka s letadlem a tím ohrožení cestujících na palubě nebo může letištní prostory vykrádat. Zde opět nelze jednoznačně určit, jak se bude útočník chovat.

- Další typy útoků

Systém je navržený tak, aby byl rozšířitelný o další typy útoků. Útoků může být mnoho a mohou být různě variabilní. Útočník může cestující ohrožovat různými zbraněmi a to buď puškou, pistolí, nožem a nebo dokonce bombou. Útočník může a nemusí být agresivní a může zasáhnout na různých místech letiště. Tyto útoky se budou odehrávat náhodně během dne/noci a úkolem zásahové jednotky bude odzbrojit útočníka a zpacifikovat ho.

Důležitým aspektem je, nasimulovat nepředvídatelnost útočníka, aby operativní i zásahová jednotka, byly schopny správně vyhodnotit situaci a podle toho správně zasáhnout. Není možné k útočníkovi s puškou přistupovat jako k útočníkovi, který neoprávněně vnikne na venkovní prostory letiště a je neozbrojený. Tuto variabilitu útočníků lze do jisté míry napodobit zavedením prvku náhodnosti v jednání. Např. útočník nestřílí hned, ale začně střílet po určité době a s určitou prodlevou. Cíl si také útočník vybírá náhodně a to buď zásahovou jednotku nebo cestujícího čekajícího na odlet.

2.3.4 Zásahové jednotky

Zásahové jednotky se pohybují v jednočlenných skupinách. Jsou vybaveny zbraní. Zásahových jednotek může být více druhů. Po letišti se mohou pohybovat policisté a nebo ostraha letiště. V našem případě budeme brát v úvahu pouze ostrahu letiště s dvěma puškami a neomezeným množstvím nábojů. Na reálném letišti mají zásahové jednotky vstupní karty, které umožňují se chodbami, které nejsou pro běžné cestující přístupné, pohybovat budovou a usnadňovat tak průchod letištěm. Pohybují se po vnitřní i venkovní ploše letiště. Na venkovní ploše si musejí dávat pozor, aby neohrozily provoz letadel na ploše.

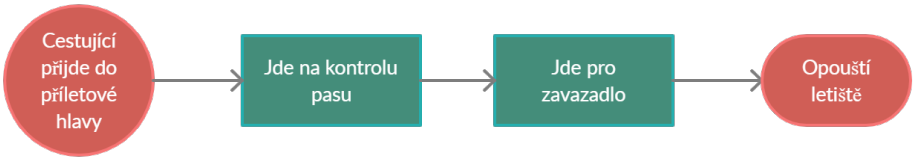
2.4 Modelování

Jako modelovací systém jsem vybíral mezi Blender, Maya a Cinema 3D. Blender je díky své jednoduchosti a nespočtu návodů jedním z nejpoužívanějších. Při modelování bylo potřeba vzít v úvahu poznatky získané při analýze letištní budovy a jejího okolí, tedy nezbytné fragmenty budov a jejich rozmístění ve výsledném modelu. Některé fragmenty je možné vynechat, protože nehrají významnou roli v bezpečnosti letiště. Cestující, který přichází na odlet, musí dodržet následující postup letištěm:

Obrázek 2.11: Postup halou pro cestující, kteří odlétají.

Toto je zjednodušený postup. V reálné situaci, cestující nemusí na odbavení zavazadel, pokud žádné nemá a palubní lístek má online, případně jde na kontrolu až u brány, pokud jde o decentralizovanou kontrolu. V popisu je brána v potaz centralizovaná kontrola. Může také jít na galerii a rozhlížet se, nebo jít nakupovat a nebo se posadit a čekat na svůj odlet. Možností je mnoho a chování do jisté míry i nepředvídatelné.

Cestující, kteří přilétají, jdou ven z příletové haly následujícím způsobem:



Obrázek 2.12: Postup halou pro cestující, kteří přilétají

Opět je to zjednodušený model. Cestující nemusí opouštět letiště a mohou přestupovat a kontrolou prochází podle toho z, jaké země přilétají. Toto jsou však zjednodušené modely, dle kterých byla navrhnuta výsledná příletová a odletová hala a tedy celé letiště.

Venkovní prostory jsou taktéž zjednodušené. Je potřeba jedna řídící věž, stačí nám jedna vzletová plocha a několik hangárů. Celý venkovní prostor je oplocený a hlídaný senzory.

Pro účely diplomové práce, jsou postačující základní modely postupu cestujících, není potřeba brát v úvahu i to, že cestující může jít buď na bránu nebo si sednout na lavičku nebo jít nakupovat.

2.5 Virtuální realita - hardware

Pro testovací a implementační účely byl použit HTC Vive Pro. Který používá dva AMOLED displeje (AMOLED patří do rodiny OLED displejů) s vysokým rozlišením (2880 x 1600 pixelů) na jedno oko, vykreslovací frekvencí 90 Hz a dvěma ovladači pro interakci s objekty ve světě VR a pohyb. Pro sledování brýlí a ovladačů jsou použity dvě základní stanice, které na základě infračervených paprsků snímají pohyb ovladačů a headsetu (brýlí). Headset obsahuje přídavná sluchátka. Headset je možné používat ve dvou módech - tzn. *Standing-only play area* a *Room-scale play area*. Pro standing-only mód se kalibruje pozice headsetu vůči zemi a headset se používá ve stoje bez chůze. Pro room-scale mód se musí nakalibrovat i herní prostor. Tedy vymezit hranice, které pokud překročíte, tak headset na to upozorní. Při nastavování snímané plochy je nutné dodržet pravidla pro konkrétní headset, která se mohou lišit podle výrobce headsetu. Pravidly je myšlena např. vzdálenost a umístění sledovacích stanic a podobně.

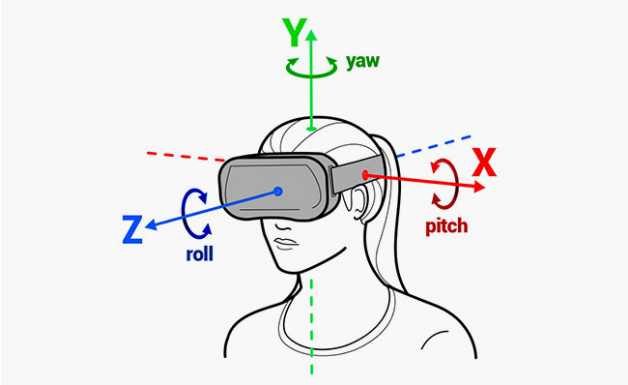
Pro sledování pohybu a umístění brýlí a ovladačů v prostoru, je potřeba mít dvě základní stanice, které na základě infračerveného světla detekují aktuální pozici VR headsetu a ovladačů.

Obrázek 2.13: HTC Vive Pro se základními stanicemi a ovladačemi [3].

Dalšími typy VR headsetů jsou např. Oculus Rift nebo Valve Index. Tyto dva typy VR headsetu se však liší od HTC Vive headsetů. Mají rozdílné obnovovací frekvence a rozlišení, ale největší rozdíl je ve sledování pozice headsetu. HTC Vive a Valve Index používá tzv. *Outside-in tracking* a Oculus Rift používá tzv. *Inside-out tracking*. Pro *Inside-out tracking* je na headsetu umístěna kamera, která detekuje změnu pozice VR headsetu vůči okolnímu prostředí. Tuto změnu může detekovat za použití konkrétních obrazců nebo znaků (čtverců, obdélníků a podobných tvarů, které kamera rozezná a určí změnu pozice) - nevýhoda tohoto použití je, že jakmile kamera nesnímá obrazce, tak neví, na základě čeho má určit pozici. Změnu je ale možné detekovat i bez použití obrazců a to např. tak, že kamera si určí významné

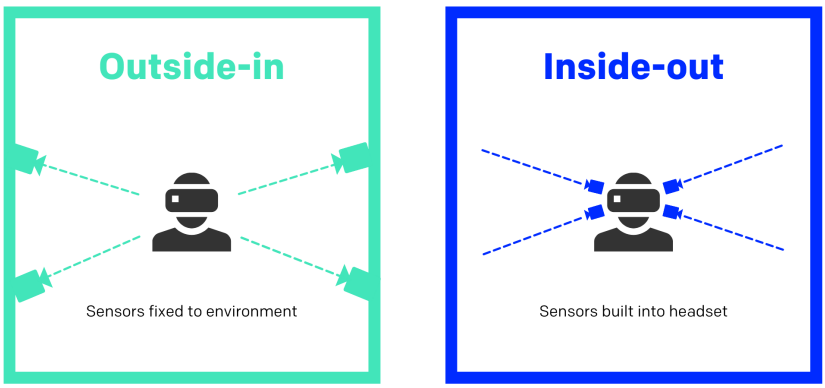
body a vůči nim poté počítá změnu pozice headsetu. Ke zpřesnění polohy headsetu může systém používat i dodatečné senzory, jako např. gyroskop a nebo akcelerometr.

Gyroskop určuje naklonění a natočení headsetu a akceletometr měří zrychlení posuvné a rotační. Kombinací těchto dvou senzorů je možné určit přesné natočení headsetu. Naklonění headsetu můžeme měřit pomocí natočení headsetu vůči ose x, y, z. Tyto úhly se nazývají yaw, roll, pitch.



Obrázek 2.14: Natočení VR headsetu [4].

U *Outside-in tracking* pro detekci VR headsetu v prostoru jsou zapotřebí externí zařízení, kterými jsou u HTC Vive a Valve Index základní stanice, které se v prostoru umístí naproti sobě. Výhoda tohoto řešení je, že sledování headsetu je přesnější (je taktéž možné přidat více externích zařízení pro sledování headsetu, můžeme mít třeba 10 zařízení pro sledování pozice a rozmístit je různě po prostoru).



Obrázek 2.15: Outside-in tracking vs. Inside-out tracking [5].

Dalším možným vylepším a zlepšením požitku ze simulace může být tzv. nekonečný prostor. Nekonečných prostorů a firem, které se touto technologií zabývají je mnoho. Můžeme použít běžecké pásy pro VR, můžeme použít závěsy. Tyto technologie nám umožní odstranit limitaci fyzickým prostorem. Nebude potřeba řešit kontinuální pohyb (který za použití ovladačů způsobuje motion sickness) a bude možné zcela zanedbat pohyb teleportem (který je nepraktický pro účely taktického výcviku). Příkladem závěsu je pohybová platforma KatVR Kat Walk, která je cenově dostupná a umožnuje pohyb v celém rozsahu, tedy 360°.

Obrázek 2.16: Příklad hardwaru simulujícího nekonečný prostor (závěs) [6].

Při implementaci v Unity poté existují knihovny, které umožňují snímání pohybu přenášet jako vstup do Unity, a další práce s tímto vstupem může poté probíhat tak, že budeme hlídat tento vstup a na základě změny budeme hýbat s VR postavou.

2.6 Shrnutí

Dle provedené analýzy byl jako engine pro tvorbu softwaru zvolen herní engine Unity, kvůli komunitě a podpoře VR. Software pro modelování byl zvolen Blender kvůli své jednoduchosti a možnostem, jimiž disponuje.

Pro samotnou implementaci byly zvoleny tři typy útoků - útočník přijde na

galerii a začně střílet. Útočník naruší integritu pláště budovy a začne střílet a třetí útok, kdy útočník vnikne do neveřejného prostoru letiště.

Výraznou část času jsem strávil modelováním budovy letiště. Kostra letiště byla stažená a poté byla rozšířena na větší a komplexnější model, který obsahuje více prostorů, odbavovací brány, bezpečnostní brány a brány pro průchod do letadla. Do modelu letištní budovy byly přidány jednoduché modely laviček, květin, odbavovacích přepážek a bezpečnostních přepážek, přepážka pro kontrolu po příletu, pásové dopravníky pro zavazadla a podobně.

Kapitola 3

Implementace a modelování

Simulace, jak jsem již zmínil, je implemetována v herním eginu Unity za pomoci jazyka C#. A také za pomoci Asset Storu (obchod součástí Unity), kde Unity komunita nabízí balíčky zpoplatněné i balíčky zdarma pro vývoj, které je možné si naimportovat a použít. Např. pro VR byl použit balíček XR, který výrazně zjednodušuje implementaci pro VR hry/simulace. Nabízí skripty, které za nás zajistí možnost vzít objekt nebo pohyb po mapě nebo ukládat věci do tzv. inventáře. Simulace je koncipována tak, že útok může přijít kdykoliv a zásahová jednotka netuší kdy. Může to vydedukovat např. z toho, že ví, kdy odlétají a přilétají letadla, tedy v jakou dobu, se bude na letišti hromadit nejvíce lidí.

Důležitým prvkem v bezpečnosti byla galerie, která je zahrnutá do modelu letiště. Ta dává útočníkovi jistou výhodu, a to tu, že má přehled o dění na letišti, zatímco zasahující jednotka ne.

3.1 Mechanismy simulace

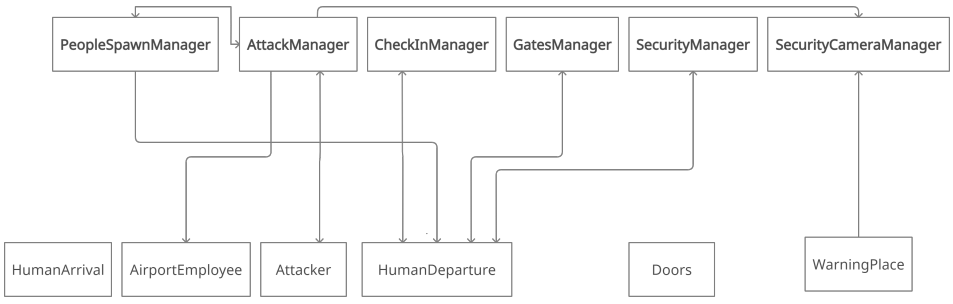
V simulaci je implmenetovaná řada mechanismů, které fungují nezávisle na sobě (pouze spolu komunikují) a ve výsledku dávají funkční simulaci letištního provozu a letiště jako celku a případného útoku na letišti.

Architektura

Celá simulace je poskládaná z tzv. manažerů a skriptů připnutých k jednotlivým objektům - dále už jen jako "objekty". Skripty popisují a implementují chování modelů v konkrétních situacích, které mohou nastat. Herní mechanismy pro chování cestujících, kteří se chystají na odlet nebo přílet, zaměstnanců letiště a letadel, jsou naimplementovány jako stavové automaty. Stavové automaty jsou popsány v bodech níže pomocí UML diagramu. Manažeři se starají o jednotlivé modely a skripty tak, aby mohly mezi sebou komunikovat. Např. *AttackManager* zajišťuje chod útoku. Tzn. ví o tom, že již útok probíhá, tedy nevytvoří dalšího útočníka a ví i o tom, že již byl útočník zpacifikován a může vytvořit dalšího. Celá architektura je navrhnutá tak, aby bylo možné simulaci rozšiřovat o další manažery a naskriptované objekty, jako např. další útočníky a chování jednotlivých objektů. Je možné jednoduše naskriptovat dalšího útočníka, či pracovníka letiště nebo jednoduše přidat chování příslušných jednotek.

UML stavový diagram popisuje graficky stavy a jednotlivé přechody mezi nimi. Každý objekt, který se chová jako stavový automat, má konečnou množinu stavů a jasně definované přechody mezi stavy. Můžeme tedy jednotlivé objekty popsat UML stavovým diagramem. Pro znázornění architektury, tedy jak spolu komunikují manažeři a objekty, je taktéž možné použít UML diagram, kde si všechny manažery a objekty nakreslíme do boxu a propojíme spolu šipkou boxy, které spolu komunikují jedním, druhým nebo obouma směry.

UML Diagram popisující komunikaci objektů a jednotlivých manažerů:



Obrázek 3.1: 1. část UML diagramu navrhnuté architektury.

Diagram popisuje objekty, kterých může být ve scéně více instancí (např. cestující, který se chystá na odlet, tak každý jeden cestující je jeden objekt), zatímco manažeři zajišťují komunikaci mezi objekty a mezi manažery, mají ve scéně jen jednu instaci. Manageři komunikují buď jednosměrně nebo obousměrně s objekty a ostatními manažery.

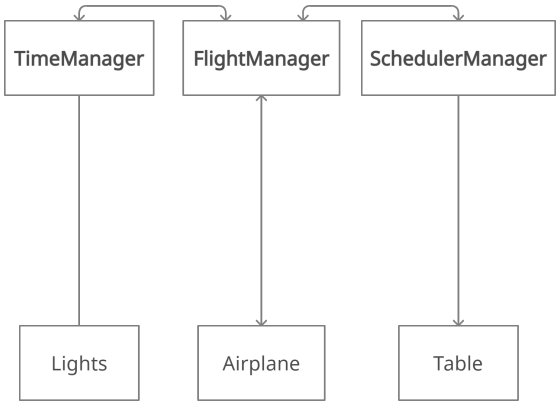
Manažeři:

PeopleSpawnManager - manažer zodpovědný za vytváření objektů *HumanDeparture* a *Attacker*. Lidé se vytvářejí 2h před odletem letadla a vytváří se 30 - 50 cestujících s náhodným časovým rozestupem. S určitou pravděpodobností je také možné, že se vytvoří útočník. Tyto konstanty je možné měnit přímo ve skriptu manažera nebo v Unity Inspectoru.
AttackManager - tento manažer se stará o samotné útoky. Ví o případném útoku a komunikuje s *PeopleSpawnManager*. Ví o možných instancích útočníků a o jejich chování. Pokud *PeopleSpawnManager* vytváří útočníka, na instanci se ptá právě *AttackManager*, který si drží jejich seznam a náhodně vybírá.
CheckInManager - stará se o fronty na odbavovací přepážce. Cestující nejdříve přijde do fronty, kde čeká, až mu přijde pokyn od *CheckInManagera*, že některá z přepážek je volná a případně místo přepážky, která je volná. Jakmile cestující opustí přepážku, tak o tom informuje právě *CheckInManagera* a ten si stav přepážky uloží a případně pošle dalšímu cestujícímu informaci o volné přepážce.
GateManager - *GateManager* ví o letadlech, které jsou právě v bráně a čekají na odlet. Poskytuje tuto informaci cestujícím, kteří na základě toho buď jdou do příslušné volné brány a nebo chodí náhodně po prvním patře letiště.
SecurityManager - tento manažer se chová obdobně jako **CheckInManager**. Tedy spravuje frontu, která se tvoří na bezpečnostní kontrole. Ví o obsazených přepážkách bezpečnostní kontroly, a pokud se místo uvolní, objekt *HumanDeparture* o tom informuje *SecurityManagera*, který si tento stav uloží a případně pošle na cestující ve frontě.
SecurityCameraManager - Tento manažer spravuje bezpečnostní kamery a senzory. Jakmile cestující projde přes pohybový senzor, tak o tom senzor informuje *SecurityCameraManagera* a ten už vykonává příslušnou akci. Tedy ukáže operátorovi na obrazovce upozornění. Stará se také o to, aby bylo upozornění z obrazovky staženo po skončení útoku. Tuto informaci dostane od *AttackManagera*.

Objekty:

HumanArrival - objekt nepotřebuje žádný manažer pro své fungování. Jeho fungování je jednoduché. Cestující vystoupí z letadla, jde na kontrolu a z ní buď pro zavazadlo a nebo ven z letiště a objekt se zničí (aby zbytečně nezabíral výkon při vykreslování). Jestli půjde pro zavazadlo k některému z dopravníků, nebo ven z letiště, se rozhodne náhodně.
AirportEmployee - objekt komunikuje pouze s *AttackManagerem*, a to pouze, aby mu *AttackManager* předal informaci o tom, že jeden z cestujících

útočí a tedy aby se objekt mohl příslušně zachovat (panikařit v našem případě).
Attacker - Dává informaci *AttackManagerovi* o tom, že začíná útok, aby manažer mohl informovat o této skutečnosti ostatní objekty. Taktéž informuje *AttackManagera* o tom, že byl zničen nebo zatčen, a tedy žádný útoku neprobíhá. Manažer poté pošle informaci objektům **HumanDeparture** a **AirportEmployee** a při komunikaci s **PeopleSpawnManagerem** posílá informaci, že žádný útok právě neprobíhá a může se další vytvořit.
HumanDeparture - Tento objekt komunikuje s *PeopleSpawnManagerem*, *CheckInManagerem*, *SecurityManagerem* a *GateManagerem*.
Doors - Tento objekt nekomunikuje se žádným z manažerů. Pouze hlídá kolizi, bez aplikování fyziky, objektu s jinými objekty (za pomoci koliderů, popsaných v kapitole 2.2) a případně otevírá a zavírá dveře.
WarningPlace - Objekt simulující pohybový senzor. Při kolizi s jiným objektem vyvolá událost do *SecurityCameraManager* a ten podle toho jedná.



Obrázek 3.2: 2. část UML diagram navrhnuté architektury.

Manažeři:

TimeManager - je zodpovědný za 24h cykly v simulaci. Tedy počítá aktuální čas a tuto hodnotu si drží pro případnou komunikaci s ostatními manažery.
FlightManager - manažer se stará o přílety a odlety. Komunikuje s *TimeManagerem* a podle času spíná události nad objekty *Airplane*.
SchedulerManager - stará se pouze o zobrazování příslušných odletů a příletů. Ptá se *FlightManagera* na objekty *Airplane* a jejich přílety a odlety, jejichž hodnoty si *FlightManager* drží a poskytuje ostatním manažerům.

Objekty:

Lights - světla jsou zodpovědná za osvícení vnitřní a venkovní plochy. Spínají se na základě události z *TimeManagera*, který spíná událost zapnutí

a vypnutí světel dle aktuálního času.
Airplane - samotné letadlo, které dostává příkazy, aby odletělo nebo přiletělo od *FlightManager*. Je zodpovědné za vytváření *HumanArrival* objektů.
Table - jednoduché tabule, které ukazují v seznamu, sestupně podle času, přílety a odlety, které se stahují z *SchedulerManagera*, a ten si tuto hodnotu stahuje od *FlightManagera*.

3.1.1 Virtuální realita - taktická jednotka

Celé cvičení za zásahovou jednotku se odehrává ve VR. Zásahová jednotka si nasadí VR headset a může se po letišti, a okolí letiště, pohybovat pomocí ovládačů. Má k dispozici dvě zbraně:

- Automatickou pušku - zásahová jednotka drží spoušť a zbraň střílí sama s určitým rozestupem.
- Pistoli - zásahová jednotka musí mačkat spoušť a střílet po jedné kulce.

Zbraně je možné brát, pokládat a ukládat. Ukládat je možné do "inventáře", kde po levém a pravém boku je místo vždy na jednu zbraň. Toto simuluje připnutí zbraně k opasku a tedy pokud zásahová jednotka vloží zbraň do "inventáře", zbraň se pohybuje s ním a může si jí kdykoliv vzít zpět do ruky. Nebo je možné zbraň zahodit nebo položit a později vzít. Zbraně mají neomezené množství nábojů a není potřeba je přebíjet.

Obrázek 3.3: Ukázka "inventáře".

Na obrázku 3.3 lze vidět "inventář", kde na jedné straně je připnutá automatická puška a druhou pušku drží zásahová jednotka v ruce a zelená koule indikuje, že zbraň je možné připnout na pásek, na volné místo.

Pohybovat se je možné pomocí ovladačů kontinuálním pohybem, který může způsobovat motion sickness a nebo se přepnout do teleportačního módu v HUD (více o HUD v kapitole 3.1.3). Teleport funguje tak, že jednotka namíří na místo, kam se chce teleportovat a teleportuje se, pokud na dané místo je možné se dostat. Toto je poněkud nepraktické pro účely simulace, jelikož to může zásahové jednotce ulehčit práci při zásahu, může se teleportovat za překážku a tam se poté krýt a střílet bez nutnosti se kontinuálně přesunout a tím riskovat zásah. Teleport je možné použít (stejně jako kontinuální pohyb) po celé ploše letiště, vyjma ploch, kam by se reálně nebylo možné dostat. Teleportování nezpůsobuje motion sickness. Pohybovat se je možné po venkovní i vnitřní ploše. Co se týče venkovních prostorů, tak zde je potřeba si dávat pozor na letadla připravující se na vzlet a přílet, která mohou ohrozit zásahovou jednotku a zásahová jednotka může tento provoz také ohrozit.

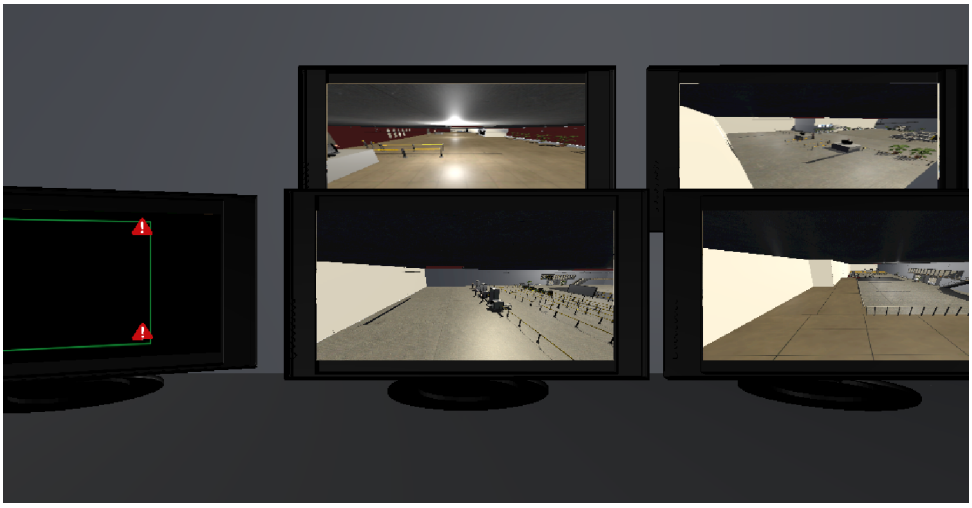
Zásahová jednotka nemá přehled o dění na letišti, nemá k dispozici výstup ze senzorů a kamerového systému. Pohybuje se po ploše letiště, a to buď venkovní nebo vnitřní. Informace o útoku přichází buď od operátora, který má k dispozici kamerový systém, nebo je možné zaslechnout střelbu (při výstřelu je přehrán zvuk střelby). Na reálných letištích jsou i senzory, které snímají neoprávněný vstup, tedy pokud se někdo dostane přes dveře, kam nemá oprávnění. V implementaci ten senzor nebyl zahrnut, jelikož služební chodby nebyly v modelu namodelovány, tudíž zde není potřeba hlídat neoprávněné prolomení dveří.

Zásahová jednotka si taktéž musí dát pozor. Útočníci střílí nejenom na cestující, ale mohou také vystřelit na zásahovou jednotku a tím končí simulace. Zásahová jednotka se tedy musí krýt a postupovat takticky, aby zamezil útoku na cestující a svou osobu. Nemá informaci o tom, zda útočník pouze zbraní ohrožuje, nebo jestli se po náhodné době rozhodne vystřelit. Riziko je potřeba posoudit dle chování útočníka. Útočník může a nemusí být ozbrojený na základě toho je potřeba jednat.

3.1.2 Monitor - operátor

Po celém namodelovaném letišti jsou implementovány kamery, jejichž obraz se promítá do monitorů, které vidí operátor. V Unity je možné výstup GameObjektu, který má komponentu *Camera*, přesměrovat do *Render Texture*.

Tuto texturu je poté možné obalit na GameObjekt. Toto nám zajistí, že se výstup kamer promítá přímo do obrazovek operátora a real-time vidí dění na letišti.



Obrázek 3.4: Obrazovka, kterou vidí operátor na monitoru PC.

Je možné volit kvalitu a rozlišení textury a kamery, a tím simulovat různé druhy kamer (různé kvality a rozlišení). Kamery by měly být po celé letištní ploše, v implementaci bylo však zvoleno pokrytí pouze vnitřních prostorů a venkovní prostory jsou pokryté pohybovými senzory.

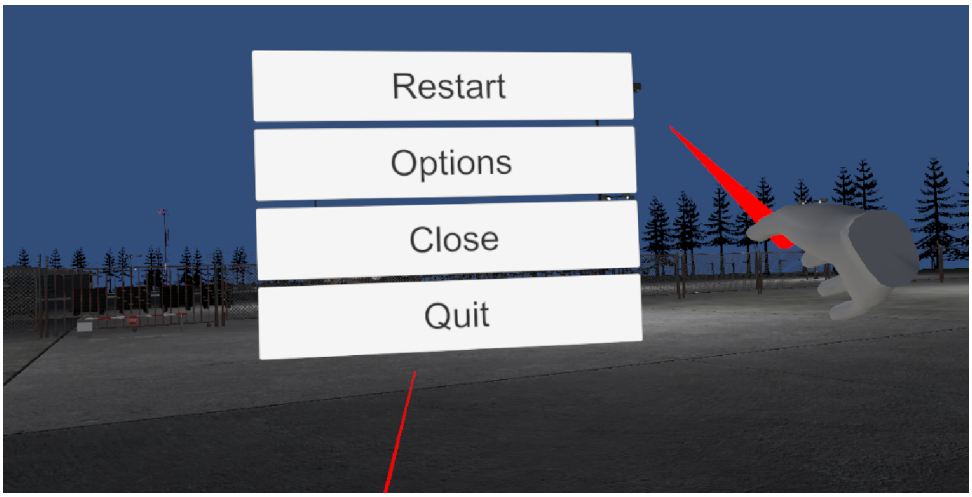
Operátor má přístup k celému kamerovému systému a vidí jednotlivé cestující a sleduje jejich chování. Vidí také pohybové senzory, které mohou zaznamenat vniknutí do venkovního prostoru letiště. Vidí prostory jak příletové, tak odletové haly a výstupy pohybových senzorů ve venkovním prostoru.

Pohybové senzory jsou implementovány za pomocí *Box Collideru* a jejich atributu *IsTrigger*. Tento atribut zajištuje, že pokud se objekty srazí, tak na ně nebude aplikována fyzika a tedy v simulaci se tváří jako kdyby nedošlo ke kolizi, ale skripty kolizi zachytí a vyvolají patřičnou událost - v tomto případě alarm u operátora.

3.1.3 Virtuální realita - HUD

Slouží pro ukončení výcviku či restartování výcviku a nebo případné úpravy nastavení. HUD může také vyskočit, pokud se výcvik nepodaří, např. zásahovou jednotku někdo zastřelí nebo zastřelí nevinnou osobu, a nebo zásahovou jednotku srazí letadlo, při přistávání nebo vzletu. HUD je možné ovládat za

pomoci ovladačů a raycastů (čar v prostoru). Je zde i možnost nastavení, která umožňuje výcvikovou jednotku ve VR přepnout do módu teleport (pohybuje se po mapě teleportováním) a nebo může využít kontinuální pohyb, který lépe simuluje reálné podmínky.



Obrázek 3.5: HUD pro ukončení, restart simulace nebo případnou úpravu nastavení.

Jak je na obrázku 3.5 znázorněno, HUD se ovládá za pomoci raycastů (červený raycast indikuje, že není možná žádná akce a bílý naopak, že možná je), kde výcviková jednotka může vybrat jednu z možností a proklikávat se tak celým HUD menu (jít do nastavení, ukončit simulaci, začít simulaci znovu a nebo schovat HUD).

3.1.4 Denní cyklus

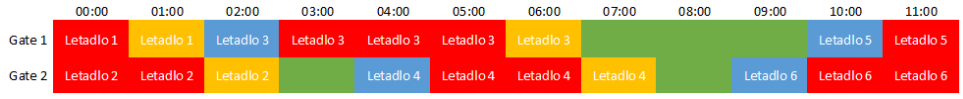
Celý simulátor běží v 24 hodinovém cyklu, kde se střídá den a noc. Na základě denního cyklu se mění i osvětlení, které simuluje východ a západ slunce. Tento cyklus je možné urychlit dle potřeby v postranním menu v Unity Inspectoru, kde se dají měnit proměnné skriptů. Od toho se poté odvíjí jednotlivé události. S denním cyklem se také počítá aktuální čas a podle aktuálního času se řídí ostatní manažeři. Např. *PeopleSpawnManager* zajišťuje, aby se lidé začínali objevovat 2 hodiny před odletem, nebo *FlightManager*, který na základě časové hodnoty připravuje letadla na odlet a přílet.

Na denní cyklus se vážou objekty *Lights*, které se vypínají a zapínají dle konkrétních hodin. V noci zajišťují osvětlení venkovních i vnitřních prostorů a ve dne, kdy to není potřeba, tak *TimeManager* vygeneruje událost vypnutí

světel a pošle jí všem objektům *Light*. Naopak v noci vygeneruje událost pro zapnutí a všem objektům, představujících světla, odešle. Časové hodnoty lze opět upravovat v Unity Inspectoru a měnit hodiny událostí zapnutí a vypnutí světel.

3.1.5 Letištní provoz

Letiště má dvě brány, kde se střídají letadla dle letového řádu. Letový řád je možné vidět na tabulích v simulaci, kde se zásahová jednotka může podívat na přílety a odlety. Podle letového řádu letadlo v danou hodinu odjede na vzletovou plochu, kde poté vzlétne do vzduchu a zmizí a čeká na příkaz od **FlightManagera** na přílet zpět. V určitou hodinu se vrátí zpět na příletovou plochu a zajede (pomocí komponenty Nav Mesh Agent) do příslušné brány, k chobotu a o této informaci informuje **GateManagera**. O přílety a odlety se stará **FlightManager**, který v pravidelných intervalech provádí kontrolu aktuálního času a podle toho zařídí přílet/odlet letadla. Pohyb letadel po letištní ploše je implementován pomocí *NavMeshAgent* (NavMesh komponenta popsána v kapitole 2.2), kde na letištní ploše je vygenerovaná *Nav Mesh*, kde agent má jasně dané cíle, po kterých se postupně pohybuje a končí u brány modelu letiště.



Obrázek 3.6: Plán příletů a odletů - hodina 00:00 - 11:00.

Obrázek 3.7: Plán příletů a odletů - hodina 12:00 - 23:00.

Obrázek 3.8: Vysvětlivka tabulek příletů a odletů.

V plánu je důležité brát v potaz dobu, v průběhu níž letadlo vzlétá a zase přistává. Aktuálně je čas zrychlený 10x, což je ovšem možné změnit v Unity Inspectoru. Není možné, aby letadlo z brány 1 a 2 vzlétalo ve stejnou hodinu. Avšak je možné, aby letadlo ve stejnou hodinu vzlétlo a jiné současně přistávalo, jelikož než letadlo dojede na vzletovou plochu, tak to druhé stihne přistát. Je taktéž možné nastavit odlet i v určitou minutu, které **TimeManager** počítá a hodnotu si ukládá. Letadlo po příletu vytváří cestující, kteří jdou do příletové haly.

3.1.6 Cestující

Cestující se začínají objevovat dvě hodiny před odletem (je možné měnit v Unity Inspectoru) a nebo po příletu letadla. Cestující při příletu nepředstavují takové bezpečnostní riziko jako cestující, kteří se připravují na odlet a ještě neprošli bezpečnostní kontrolou. O útok se stará **AttackManager**, který s určitou pravděpodobností (která jde měnit opět v Unity Inspectoru) vytvoří útočníka, pokud útok již neprobíhá, a který vykonává jeden ze tří náhodných útoků:

1. Narušení integrity pláště
   Útočník se objeví na střeše a přes okno vnikne do budovy, kde vytáhne zbraň a po náhodné době mu **AttackManager** vybere cíl a na ten vystřelí. Poté opět chvíli čeká a střílí. **AttackManager** může vybrat jako terč i zásahovou jednotku a tím ukončit a zneúspěšnit celou simulaci. Doba, po které útočník vystřelí, je náhodná a zásahová jednotka tedy neví, jak se bude útočník chovat a není možné předpovědět zcela jistě chování. Musí postupovat dle standartních postupů tak, aby neohrozila sama sebe a cestující.
2. Útok z galerie veřejného prostoru
   Útočník přijde bežným vchodem a dojde na galerii a tváří se jako běžný cestující. Na galerii získá dostatečný přehled a výhodu nad zásahovou jednotkou. Po náhodné době mu **AttackManager** vybere cíl a na ten vystřelí. Poté opět chvíli čeká a střílí (stejné chování jako u útočníka, který naruší integritu pláště). **AttackManager** může vybrat jako terč i zásahovou jednotku a tím ukončit a zneúspěšnit celou simulaci.
3. Vniknutí na runway
   Útočník projde dírou v plotě a pohybuje se po venkovní ploše letiště. Cílem zásahové jednotky je tohoto útočníka zadržet. Útočník není ozbrojený a nepředstavuje tak velké bezpečnostní riziko. Útočník nijak neohrožuje cestující, ale ohrožuje provoz na letišti. Může vniknout na vzletovou a přistávací plochu a tím vytváří bezpečností riziko.

Tyto útoky se odehrávají náhodně během dne nebo noci a úkolem zásahové jednotky je odzbrojit útočníka a zpacifikovat ho, tedy buď zastřelit, nebo namířit na něj zbraní a útočník se vzdá. Důležitým faktorem je i to, že útočník může zaútočit i na zásahovou jednotku a tím tak ukončit celý výcvik.

Zde je prostor pro přidání dalších naskriptovaných nepřátel a **AttackManager** se již postará o jejich zařazení do výcvikové simulace. Např. nepřátelé, kteří odpálí bombu nebo způsobí požár. Mohou být i chytří útočníci, kteří

způsobují problémy na bezpečnostní kontrole a nebo se tváří jako neškodní a poté vytáhnou nůž. Dále mohou být útočníci, kteří zajmou rukojmí.

Cestující po příletu a před odletem se chovají jako stavový automat. Podle stavu se rozhodují co budou dělat a kam půjdou. Posloupnost akcí, které vedou k odbavení cestujícího může narušit útok a případný výstřel na cestujícího. Proto je potřeba před přechodem do dalšího stavu zkontrolovat, zda neprobíhá útok a tedy cestujícího přepnout do příslušného stavu a přesvědčit se, zda cestující není zasažen kulkou, buď od útočníka nebo zásahové jednotky.

3.1.7 Cestující

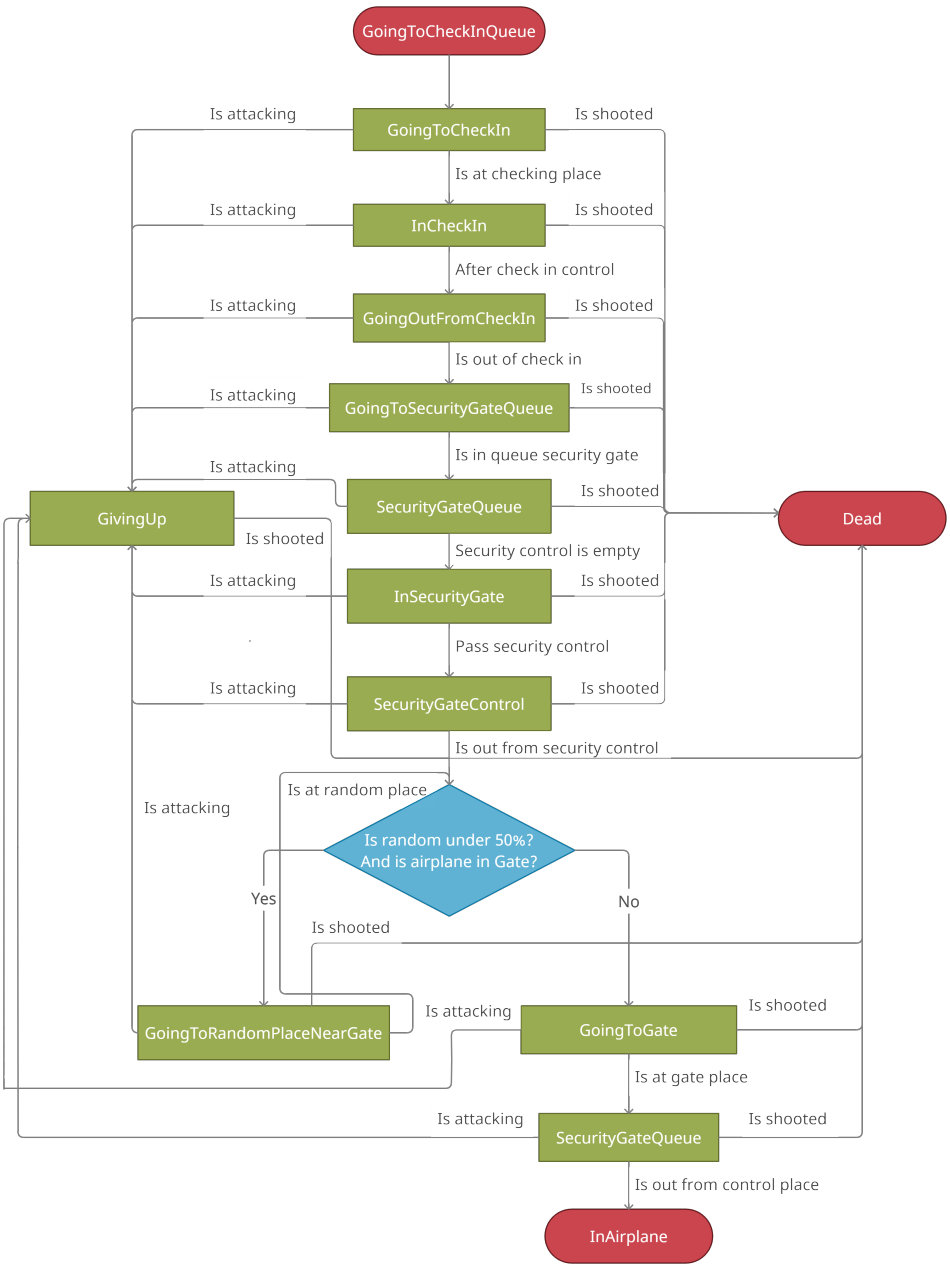
Cestující čekající na odlet:

Cestující čekající na odlet se objeví dvě hodiny před odletem a jejich vytvoření zajišťuje *PeopleSpawnManager*. Jako první jdou k odbavovací přepážce (check-in), kde se tvoří případná fronta, o kterou se stará **CheckInManager**. Odbavovací přepážky jsou dvě. Jakmile jsou obě plné, lidé čekají ve frontě. Jakmile se přepážka uvolní, jde první člověk ve frontě dále a fronta se posune. Jakmile projde odbavovací přepážkou, tak jde k bezpečnostní bráně. A informuje **CheckInManagera** o uvolnění místa u přepážky.

Cestující postupují k bezpečnostní bráně, kde se opět tvoří fronta o kterou se již stará **SecurityManager**. Ten má přehled o všech třech přepážkách, jejich obsazenosti a posílá události jednotlivým cestujícím, kteří čekají ve frontě. Jakmile projde cestující bezpečnostní kontrolou, informuje o tom **SecurityManagera** a **SecurityManagera** volné místo přiřadí případně dalšímu cestujícímu ve frontě.

Dále cestující postupuje k jednotlivým branám, kde se o cestující stará **GateManager**. Ten ví, zda je v 1. nebo 2. bráně letadlo a pokud ano, umožní cestujícím nastoupit. Pokud ne, zajišťuje náhodný pohyb cestujících po ploše první patra.

Chování cestujících lze popsat UML diagramem:



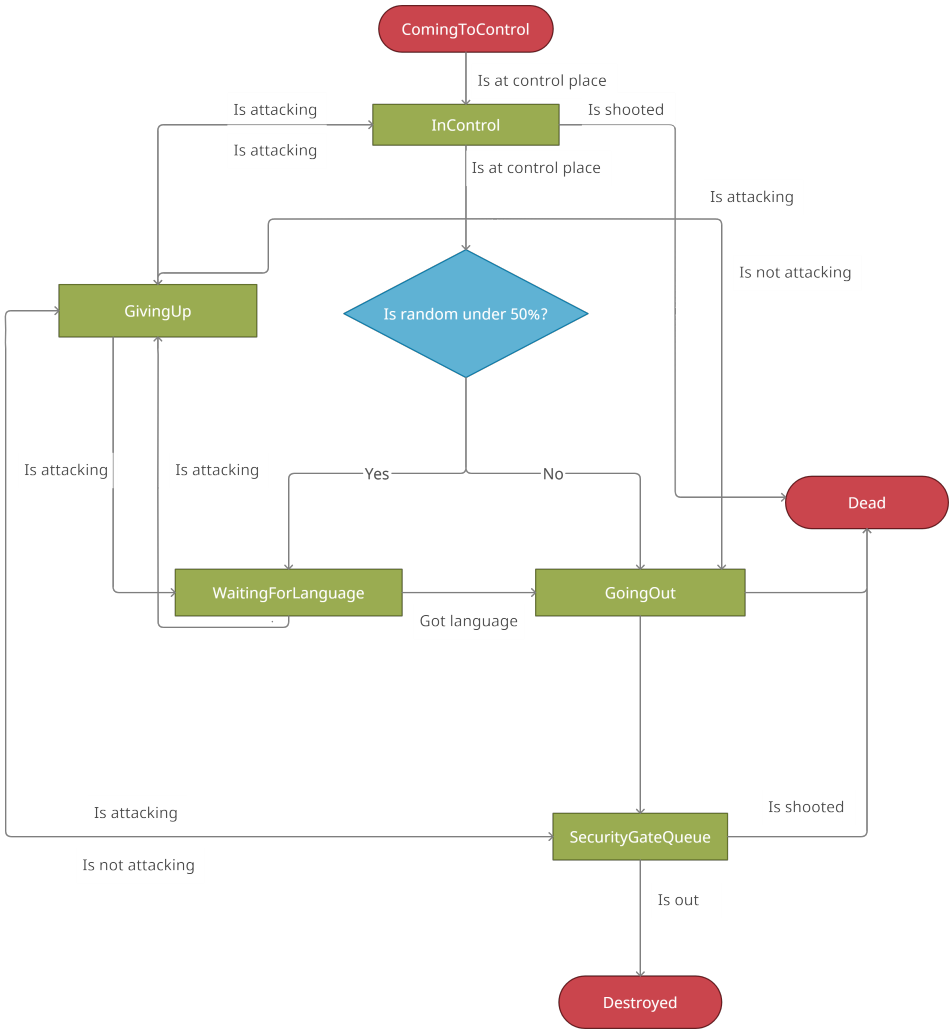
Obrázek 3.9: UML diagram popisující stavový automat cestujících po příletu.

V každém přechodu se kontroluje, zda není objekt zasažen kulkou. Pokud ano, tak přejde do konečného stavu. Tzv. zásahová jednotka je taktéž může zastřelit.

Cestující, kteří právě přiletěli:

Cestující, kteří se objeví po přistání letadla a přistavení k chobotu. Cestující jdou na kontrolu, vyzvednout zavazadlo nebo hned opustí letiště (s 50% pravděpodobností opustí letiště a s 50% pravděpodobností si nejprve půjdou pro zavazadlo). Na tyto cestující neútočí nepřátelé v odletové hale, ani se mezi nimi neobjevují útočníci. Tito cestující nepředstavují vysoké riziko.

Jejich chování lze popsat UML diagramem:



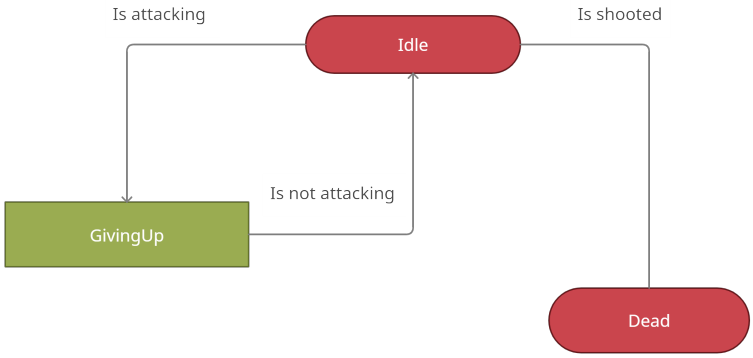
Obrázek 3.10: UML diagram popisující stavový automat cestujících po příletu.

V každém přechodu se kontroluje, zda není objekt zasažen kulkou. Pokud ano, tak přejde do konečného stavu. Tzv. zásahová jednotka je taktéž může zastřelit.

3.1.8 Pracovníci letiště

Pracovníky letiště představují lidé na odbavovacích přepážkách, bezpečnostních prohlídkách, kontrolách při příletu, branách do letadla a nebo jsou jimi prodavači v obchodech. Tyto zaměstnance může zásahová jednotka také zastřelit, ale tím končí simulace, jelikož byl zastřelen nevinný člověk.

V případě útoku se chovají obdobně jako cestující, kteří odlétají, tedy pokud začne útok, model se začne animovat a vzdávat se. Pokud zaměstnance zasáhne kulka, tak zemře. Chování zaměstnance je implementováno stavovým automatem a lze ho popsat stavovým diagramem:



Obrázek 3.11: UML diagram popisující stavový automat zaměstnance letiště.

V každém přechodu se kontroluje, zda není objekt zasažen kulkou. Pokud ano, tak přejde do konečného stavu. Taktéž je nutné kontrolovat, zda neprobíhá útok a přepnout zaměstnance letiště do příslušného stavu.

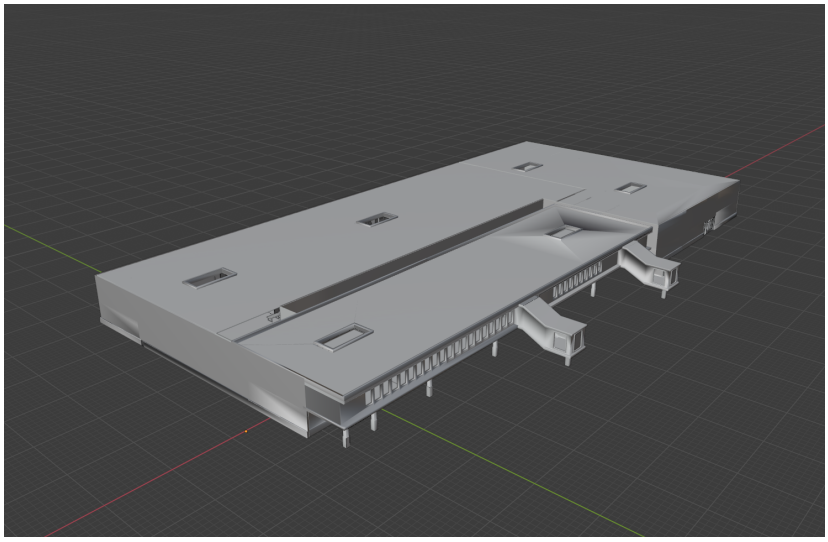
3.2 Modelování

Model letiště a příslušných prvků letiště je modelován, jak jsem již zmínil, v programu Blender. Model je smyšlený a nepředstavuje žádné reálné letiště. Letiště je modelováno dle vlastních představ, přičemž jsou dodržovány určité základní prvky letišť, které jsem konzultoval s pracovníky Letiště Václava Havla v Praze. Tyto fragmenty jsou pro většinu letišť stejné a veřejně známé nebo dohledatelné pro veřejnost.

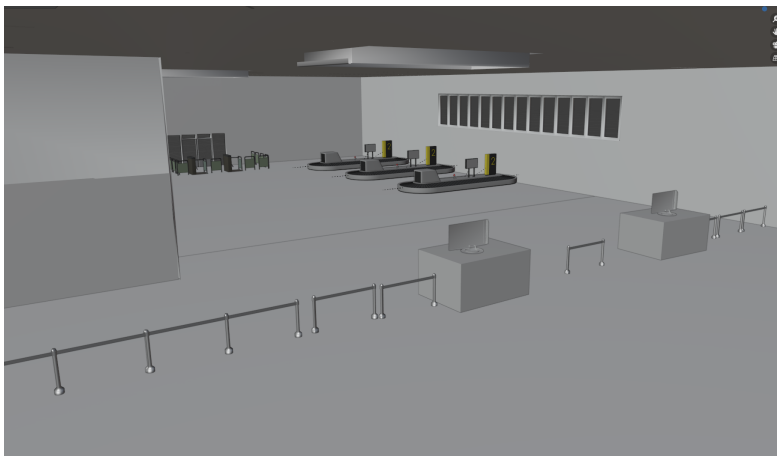
3.2.1 Základní prvky modelu

Důležitým faktorem při modelování bylo dodržet základní průchod cestujících letištěm, tedy přijde neodbavený a nezkontrolovaný cestující, který představuje vysoké riziko útoku, poté jde případně na check-in, tedy odbavovací přepážku (pokud nemá letenku online a nebo pokud má zavazadlo na odbavení), na bezpečnostní kontrolu a poté na příslušnou bránu nebo se volně pohybuje po ploše 1. patra letiště, kde jsou již cestující zkontrolovaní a odbavení. Důležitým aspektem v bezpečnosti je galerie, kde může útočník získat jistou výhodu nad zásahovou jednotkou. Inspirace byla čerpána z letiště Václava Havla v Praze, kde se galerie nachází ve veřejném prostoru příletové haly, hned u vstupu do budovy a je dostupná pro všechny nezkontrolované cestující. Z této galerie je poté možné se případně dostat do dalších prostorů (mohou být např. kanceláře), ale toto nebylo v modelu zahrnuto, jelikož to nehraje významnou roli při výcviku zásahové jednotky.

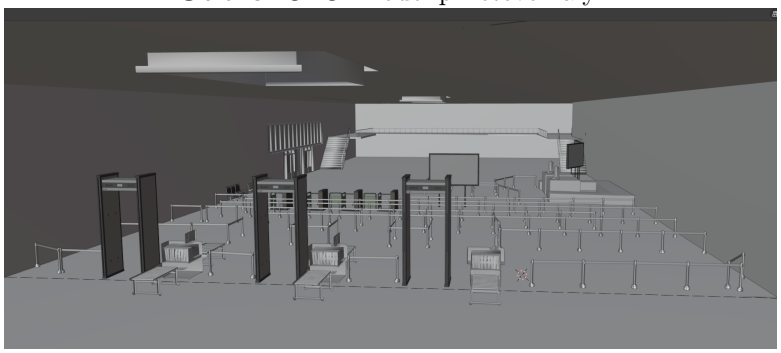
Další prvek je příletová hala, která je oddělená od odletové haly. Haly jsou v rámci stejné budovy a mohou být průchůzí s tím, že u průchodu bývá bezpečnostní pracovník, který hlídá průchozí cestu. U cestujících, kteří přilétají, je také určitý postup letištěm, který je potřeba dodržet - cestující projde kontrolou (zde záleží na tom, z které země přilétá. Pokud v rámci EU, tak kontrola nebývá, ale v modelu byla zahrnuta), jde si pro zavazadlo a projde přepážkami, které zabrání vniku neoprávněných osob z venku do příletové haly.



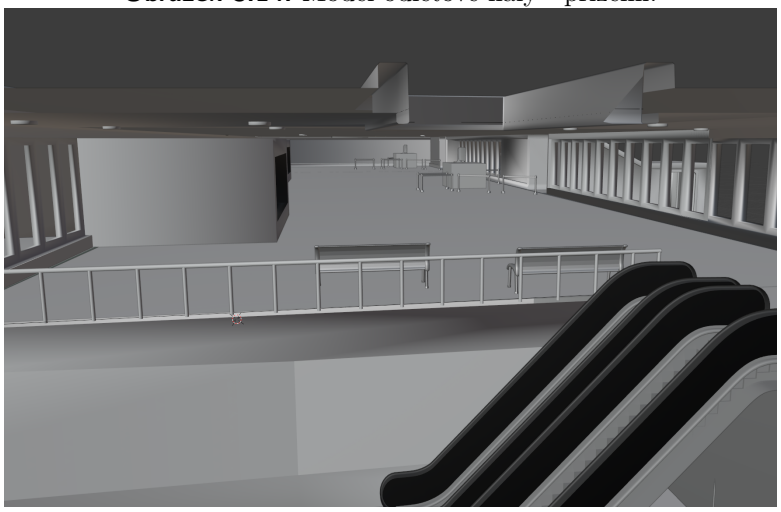
Obrázek 3.12: Model letiště.



Obrázek 3.13: Model příletové haly.



Obrázek 3.14: Model odletové haly - přízemí.



Obrázek 3.15: Model odletové haly - 1. patro.

Výsledný model je poskládaný ze stažených menších modelů (schody, výtah, lavičky, bezpečnostní prvky, monitory, květiny a přepážky). Tyto modely jsou dostupné v databázích, které jsou přístupné přes web *TurboSquid* [19], *Free3D* [20], *CGTrader* [21] nebo *Unity Asset Store* [22]. Kostra letiště je ručně modelovaná. Výsledný model je obalen jednoduchými texturami. Celý model je poté vyexportován a spolu s materiály importován do Unity, kde jsou

navázány příslušné mechanismy simulace. Venkovní prostory jsou poskládané z již hotových modelů, volně dostupných ke stažení z výše zmíněných stránek.

Modelování spočívá v přesouvání, roztahování, otáčení stěn, hran a vrcholů. Vrcholy, hrany a stěny je taktéž možné duplikovat. Pomocí těchto úkonů je možné poskládat stěny, strop, galerii, okna a dveře. Na výsledný model se posléze nabalí textury.

Pro větší věrohodnost simulace, jsou místo ovladačů použity modely rukou:

Obrázek 3.16: Modely rukou v simulaci.

Na levé ruce je model hodinek, který ukazuje aktuální čas pro lepší orientaci zásahové jednotky.

3.3 Shrnutí

Při implementaci jsem se potýkal s problémy a to převážně při práci s VR. Zvolený headset pro testování byl HTC Vive, který v některých případech není kompatibilní s XR balíčkem použitým pro usnadnění implementace a bylo potřeba najít různé kličky. Co se týče samotného modelování, zde bylo nejtěžší vytvořit si představu a navrhnout tak smyšlené letiště, které ale bude splňovat určité předpisy. Modelovací část práce zabrala nejvíce času, jelikož bylo potřeba udělat podrobnou analýzu toho, jak má vlastně letiště vypadat a poté si navrhnout vlastní. Pro lepší představu při modelování byla také domluvena prohlídka letiště Václava Havla v Praze s panem Martinem Čížkovským, který tímto značně přispěl do mé diplomové práce a předal mi

potřebné informace.

Příslušné manažery a objekty je možné ovlivňovat proměnnými v Unity Inspectoru a tím tedy řídit a optimalizovat průběh simulace. Je možné měnit proměnné jako třeba rychlost času, pravděpodobnost vytvoření útočníka, rychlost střílení ze zbraní a podobně.

K modelaci UML diagramů byl použit nástroj *Creately*.

Kapitola 4

Testování

Testování je nezbytnou součástí vytvoření kvalitního softwaru. Co se týče testování VR mechanismů, tak byl použit HTC Vive PRO jako testovací hardware. Více o HTC Vive PRO je možné se dočíst v kapitole 2.5. Důvod volby tohoto konkrétního VR headsetu byla jeho přesnost sledování a podpora Steam VR. Co se týče testování, tak zde bylo zvoleno více druhů testování pro větší pokrytí. Testy nám dávají lepší informaci o spolehlivosti softwaru a snižují riziko a množství chyb. Chyby je poté daleko snažší objevit a opravit. U simulace je toto nutností, jelikož některé situace mohou nastat jen zřídka (kvůli zanesení náhodnosti) a tudíž může být chyba objevena až později v "produkci"nebo neobjevena vůbec [23], [24].

4.1 Testování mechanismů

Typy testů použitých v pro otestávní Unity simulace:

- Unit testy

  Unity testy testují menší části kódu, ideálně samostatné metody. Pro běh těchto testů není potřeba *Game Mode*, ale běží v *Edit Modu* a nepotřebují pro své spuštění zapnutí simulace. Unit testy jsou použity pro jednotlivé metody, které se dají samostatně testovat. Např. pomocné třídy a jejich statické metody, v naší simulaci konkrétně metody pro formátování času nebo metody pro výběr náhodného prvku.

- Integrační testy

  Integrační testy testují komunikaci mezi manažery ve scéně. Obecně integrační testy slouží k otestování komunikace např. mezi moduly softwaru, pro otestování komunikace s hardwarem a nebo třeba databází. Integrační testy běží v tzv. *Game Modu*, to znamená, že pro běh testů se zapne simulace a spustí se v ní testy.

- Manuální E2E testy

  Tyto testy slouží pro otestování aplikace od samotného startu až do konce. K tomuto slouží testovací scénáře. Pod tímto je možné si představit, že simulaci spouštíme stále dokola a snažíme se vytvářet unikátní situace a v těchto situacích testovat chování softwaru jako celku. Testování probíhá manuálně, to znamená, že simulaci testuje přímo fyzická osoba a testovací scénáře má předem předpřipravené. Existují také automatické E2E testy, které v tomto případě nejsou použity. Rozdíl mezi manuálními a automatizovanými E2E testy je, že automatizované testy si vytvoří SUT (model testovácího systému) a vygeneruje si testy, které na SUT vyzkouší a vyhodnotí. Tento druh testů není možné použít v Unity, jelikož jde o komplexní simulaci a systém pro generování automatizovaných E2E testů, by bylo složité, možná téměř nereálné navrhnout a implementovat.

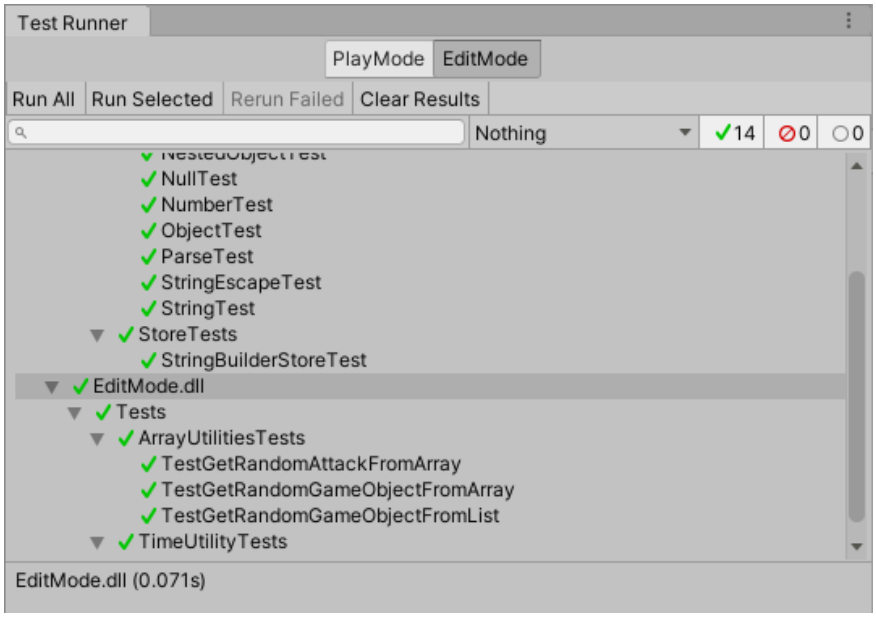
Testování jednotlivých mechanismů probíhalo přímo v Unity, kde architektura je navržená tak, že se dají mechanismy testovat jednotlivě. Např. pokud je *SchedulerManager* závislý na datech z *FlightManageru*, jsou vytvořené metody/třídy, které tzv. mockují výstup z tohoto dotazu a vrací smyšlená data. Toto umožňuje efektivní testování jednotlivých mechanismů.

Mockovací objekt, je objekt, který nějakým způsobem napodobuje chování skutečného objektu. Pokud voláme metody mockovacího objektu, tak by se měly z venku tvářit jako metody skutečného objektu a mohou vracet smyšlená data pro účely testování. Tudíž, aniž bychom museli vytvářet pro testování jednotlivé manažery (závislosti testovaného manažeru), můžeme vytvořit. tzv *Mockovací objekt*, který se chová jako reálný objekt, ale ve skutečnosti je to objekt, který plní jen funkčnost závislosti a tváří se jako plnohodnotný manažer. Tento přístup nám umožní efektivně testovat, odlazovat a rozšiřovat funkcionality jednotlivých manažerů, které jsou zakomponované do celkové simulace.

Byly vytvořeny jednotlivé prototypy, např. pro vytváření lidí se udělala samostatná scéna, kde je prototyp pouze tohoto mechanismu. Manažer komunikuje s manažerem zodpovědným za vytváření útoků, manažerem zodpovědným za správu času a manažerem zodpovědným za rozvrh příletů a odletů letadel. Pro prototypování byly tito tři manažeři namockováni, tedy byly vytvořeny fiktivní metody, které vracejí smyšlená data. Samotná scéna tedy

slouží pouze pro otestování *PeopleSpawnManager* manažeru, a tedy toho, že skript vytváří cestující s nějakým časovým rozestupem a případné útočníky. Pokud manažera budeme upravovat, testování probíhá v této scéně a změny se taktéž promítnou do scény výsledné simulace.

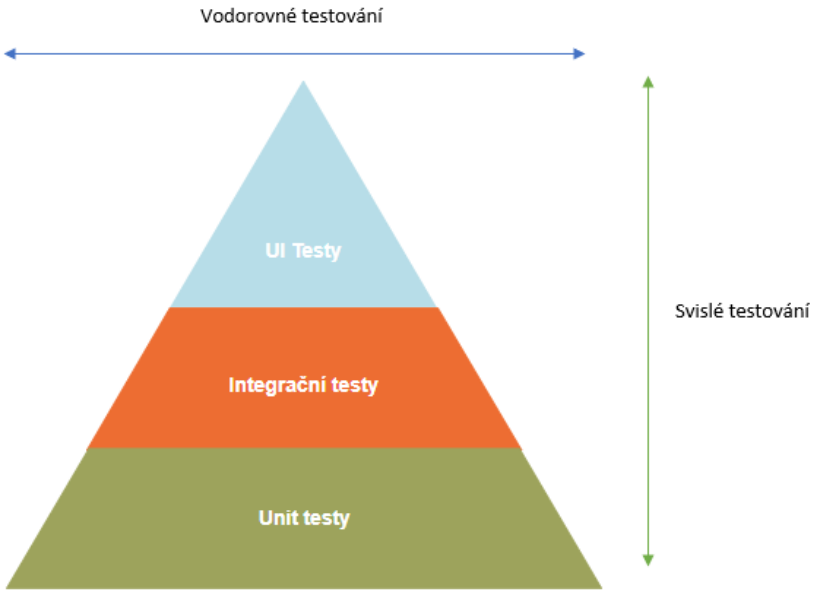
Manažerům a objektům pomáhají tzv. *Utility*. Tyto utility slouží pro jednoduché operace, např. pro formátování času nebo pro vybrání náhodného objektu. Jednotlivé pomocné třídy (Utility) se poté testují za pomocí Unity testů a frameworku *Unity Test Framework*. Framework zajistí běh skriptů, aniž by se musela zapínat hra a tím otestovat jednotlivé metody pomocných tříd. Testování probíhá přes Unity komponentu *Test Runner*.



Obrázek 4.1: Ukázka Test Runneru.

Tyto testy běží v tzv. *Edit Modu*. Ideální pro Unity testování, kde testy nepotřebují tzv. *Game Mode* a přístup k objektům v tomto módu (testy beží bez zapnutí simulace). Zde se dají otestovat např. funkce pro formátování času nebo náhodný výběr prvku z pole. Testy je taktéž možné zapnout v *Game Modu*, kde můžeme testovat chování objektů ve scéně pokud simulace běží. Tímto způsobem lze poté vytvořit integrační testy, které budou testovat komunikaci mezi objekty ve scéně.

Testování lze ukázat na pyramidě. E2E testování se dělá podle vodorovné a svislé osy. Tato pyramida se však pro Unity mírně liší:



Obrázek 4.2: E2E testování.

Vodorovné E2E testování znamená, že zkoušíme software jako celek. Tedy zvolíme vhodné testovací scénáře a ty zkoušíme, buď manuálně nebo automatickými testy (v našem případě pouze manuální E2E testy). Svislé testování znamená, že pro každou vrstvu máme napsané testy, které by měly pokrývat co největší část kódu a ty testujeme. Tedy začneme např. Unity testy, kterých by mělo být nejvíce a postupujeme k integračním testům a jako poslední jsou UI testy (které v tomto softwaru nejsou potřeba, jelikož simulace obsahuje pouze HUD pro nastavení, restartování nebo vypnutí simulace a potom UI, které oznamuje konec simulace).

Příklad testovacích scénářů pro E2E testování:

- Zastřelení

  Po startu simulace čeká zásahová jednotka na informaci o tom, že útočník ohrožuje cestující v příletové hale, jde do příletové haly a zastřelí útočníka. Tímto by měla simulace pokračovat a neměl by se přerušit provoz letiště. Cestující se vrátí zpět do původního stavu (jestli šli k odbavovací přepážce nebo na bezpečnostní kontrolu, tak se do tohoto stavu vrátí a podobně).

  Testovací scénář obsahuje tři kombinace útočníků. Může být útočník s puškou, který přišel dveřmi odletové haly, útočník který vniknul na venkovní prostory letiště a útočník, který přišel střešním oknem.
- Zadržení

Po startu simulace čeká zásahová jednotka na informaci o tom, že útočník ohrožuje cestující v příletové hale, jde do příletové haly a zamíří zbraň na útočníka a útočník se vzdá. Tímto by měla simulace pokračovat a neměl by se přerušit provoz letiště. Cestující se vrátí zpět do původního stavu (jestli šli k odbavovací přepážce nebo na bezpečnostní kontrolu, tak se do tohoto stavu vrátí a podobně).

Testovací scénář obsahuje tři kombinace útočníků. Může být útočník s puškou, který přišel dveřmi odletové haly, útočník který vniknul na venkovní prostory letiště a útočník, který přišel střešním oknem.

- Bez zásahu

  Zásahová jednotka se volně pohybuje po ploše, bez jakékoliv akce. Zde jsou tři typy scénářů. Přijde útočník z okna nebo útočník ze dveří a zastřelí zásahovou jednotku a je tedy konec simulace. Pokud přijde útočník na venkovní letištní plochu a není ozbrojený, simulace nekonční ani po určité době, ale neobjevují se noví útočníci. Pokud útočník zastřelí nevinného člověka, simulace taktéž končí.

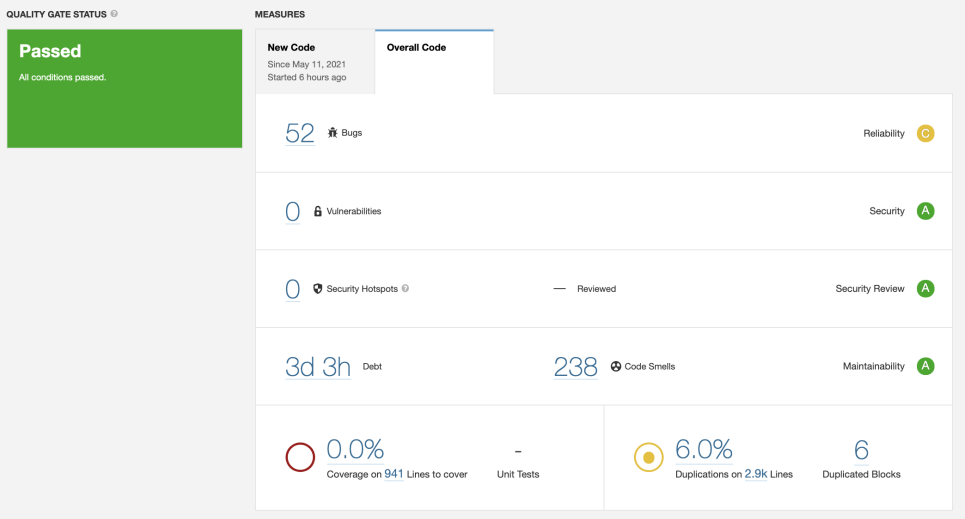
4.2 Statická analýza kódu

Statická analýza kódu spočítá ve vyhodnocení kódu bez nutnosti jeho komplikace nebo spuštění programu. Napsaný zdrojový kód se porovná vůči definovaným pravidlům, která jsou buď předem definovaná v některém nástroji a nebo je potřeba si tato pravidla ručně sepsat. Statická analýza kódu pomůže objevit potencionální bugy nebo chyby v kódu. Zvyšuje přehlednost a čitelnost kódu. Statická analýza se dá dělat např. pomocí nástroje SonarQube, který již má v sobě definovaná pravidla pro C# a my jsme schopni si tuto množinu mírně modifikovat a to tak, že zneaktivníme některá pravidla a některá necháme aktivní (pro naší simulaci byl kód analyzován vůči všem pravidlům pro jazyk C#, které nástroj SonarQube poskytuje). Existují i alternativy jako *Checkmarx* nebo *Codacy* a podobně, každopádně SonarQube byl zvolen kvůli své jednoduchosti a přehlednosti.

SonarQube je nástroj pro vyhodnocení kvality kódu, který má již v sobě zabudovadou sadu pravidel pro různé jazyky jako např. Java, C, C++, Swift, C# a podobně. Tyto sady lze různě upravovat (aktivovat a deaktivovat různá pravidla) a na základě toho poté skenovat zdrojové kódy a sledovat výstup a podle toho vylepšovat svůj kód. SonarQube umožňuje udělat i rozdíl oproti předchozí analýze, kde ukáže kolik přibylo chyb, zranitelností, pachů v kódu.

SonarQube umožňuje i nastavení tzv. *QualityGates* a *QualityProfiles*. Toto

slouží pro vyhodnocení, zda výstup z konkrétní analýzy zdrojových kódů prošel určitými kriterii jako např. pokrytí testů je větší než 80% a podobně.

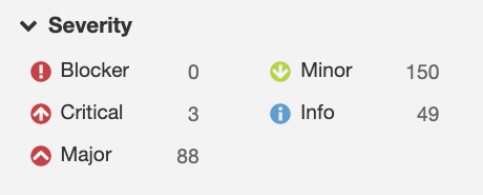


Obrázek 4.3: Ukázka z výstupu SonarQubu.

Výstup ze SonarQubu, který nás zajímá a v našem konkrétním případě, který je nápomocný při vylepšení kódu je:

- Počet bugů - vyhodnotí počet bugů v kódu.
- Vulnerabilities - zranitelnosti v kódu (vztahuje se především na webové aplikace, které jsou dostupné zvenčí prostřednictvím internetu).
- Pachy v kódu (code smells) - potencionální bugy nebo chyby v kódu, nebo části kódu, které jsou špatně čitelné a musejí být přepsány. Nepředstavují přímo chyby v kódu, ale jde o části kódu, které mohou v budoucnosti potencionálně způsobovat chyby.
- Technický dluh - počet dnů a hodin potřebných k opravení všech chyb, duplicit nebo nedodělaných částí kódů.

Bugy a pachy v kódu jsou dále rozdělené dle severity - tedy dle jejich závažnosti. Závažnost bugu nebo code smellu určuje pravidlo, které vyhodnotní část kódu jako code smell nebo bug.



Obrázek 4.4: SonarQube ukázka severit.

SonarQube vyhodnotí dle pravidel severitu, podle které je možné filtrovat a vyfiltrovat si přímo místo v kódu, kde se chyba nachází. Jak je ukázáno na obrázku 4.5, tak severity máme:

- Blokační
- Kritické
- Důležité
- Méně důležité
- Informační

Analýza kódu se provádí za pomocí SonarScanneru a tří příkazů a přímo v adresáři s projektem:

```
SonarQube.Scanner.MSBuild.exe begin /k:"AirportVR" ^
 /d:sonar.host.url="http://192.168.0.102:9000" ^
 /d:sonar.login="7e0671aea960f044b3d8a7b20f4fd4197786a42a" ^
 /d:sonar.exclusions=Assets\ImportedAssets\**, ^
   Assets\XR\**,Assets\Maquette\**, ^
   Assets\ModularAirportKit\**
```

Tento příkaz předpřipraví projekt před jeho analýzou, vytvoří konfigurační soubory. Je to takový preprocesor.

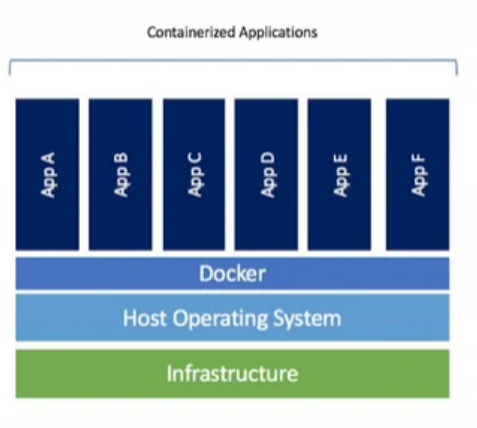
```
MSBuild.exe AirportDP.sln /t:Rebuild
```

Tento příkaz sestaví projekt a udělá analýzu podle konfigurace vytvořené předchozím příkazem.

```
SonarQube.Scanner.MSBuild.exe end ^
/d:sonar.login="7e0671aea960f044b3d8a7b20f4fd4197786a42a"
```

Poslední příkaz vezme výstup z analýzy a nahraje ho na SonarQube, který běží buď lokálně nebo na IP adrese definované v prvním příkazu. Pro testovací účely a analýzu kódu napsaného pro tuto práci byl použit SonarQube, který bežel na počítači v lokální v síti jako kontejner v Dockeru. Nástroj pro analýzu kódu je přiložený v odevzdaných souborech.

Docker je software, který umožňuje izolaci aplikací do tzv. kontejnerů. Tzn., že si můžeme stáhnout obraz již nějakého hotového softwaru a vytvořit si kontejner, ve kterým obraz poběží. Výhodou je jednotné prostředí a to, že není nutné pro rozběhnutí softwaru instalovat závislosti, definovat prostředí a proměnné nebo si software sestavovat. Můžeme si jednoduše stáhnout obraz z veřejného repozitáře, který zabalíme do kontejneru a ten pustíme a vystavíme na námi určený port. Tímto zajistíme, že aplikace bude dostupná i z venku, což je v našem případě pro SonarQube nezbytné. Rozhraní dockeru pro Mac, Linux a Windows je jednotné.



Obrázek 4.5: Docker architektura [7].

Kapitola 5

Závěr

Nejprve bylo potřeba udělat teoretickou analýzu a nasbírat potřebné teoretické znalosti. Poté bylo potřeba udělat jednoduchý model letiště a zakomponovat ho do vnějších prostorů. Pro ulehčení práce, byly použity již hotové modely pro poskládání venkovních prostorů, pro letadlo, cestující, a pro základní kostru modelu letiště. Základní kostra letiště, ale nebyla dostačující, tak byla rozšířena v modelovacím softwaru Blender, kde bylo možné si vytvořit vlastní model a zbavit se tedy případných omezení, které by se objevily s již hotovým modelem.

Modelování výsledného letiště zabralo značnou část celé práce. Bylo potřeba se s modelovacím systémem seznámit a osahat si jeho základní ovládání. Zkusit si nejdříve vytvořit jednoduché modely, jako např. dveře, zdi a okna a poté zapojit fantazii a vytvořit celou kostru letiště, kde nebyla použita žádná reálná předloha.

Práce s Unity byla celkem přímočará. Unity je jeden z přívětivějších herních enginů a snadný na pochopení a hlavně bohatý na návody a postupy, a ochotnou komunitu, která ráda poskytne své znalosti na vyřešení problému. Výraznou část implementace zabralo navrhnutí vhodné architektury, která bude umožňovat rozšířitelnost softwaru. Simulátor je rozšířitelný i o další modely a jinou kostru modelu leštiště. Naimplementované moduly je možné připnout na jiný model a chování zůstane zachované.

Ve výsledku z celého projektu vznikl komplexní simulátor, který simuluje provoz na letišti a náhodně vytváří útočníky, kteří mají jistou míru náhodného chování také. Projekt je brán jako simulace, to znamená, že jednotka má pouze

jeden pokus na zvládnutí simulace (oproti hře, kde jednotka má určitý počet životů, může mít smyšlené schopnosti a podobně). Byla navrhnuta vhodná architektura, která umožňuje celkem jednoduché pokračování a rozšiřování projektu.

V návrhu simulátoru byl kladen velký důraz na rozšířitelnost a možnost vzniku budoucích diplomových prací, jak již bylo zmíněno. Software má navrženou a popsanou architekturu, ve které je možné přidávat další manažery a objekty a tím rozšířit celou simulaci o další typy útoků, rozšířit chování cestujících a nebo chování pracovníků letiště. Situací, které mohou nastat je nespočet a vždy je možné naskriptovat nějakou unikátní situaci (útočníka) nebo sáhnout po umělé inteligenci, která bude řešit chování útočníka. Jednotlivé moduly nejsou závislé na letištním prostředí a všechny mechanismy jsou navržené tak, že je možné vyměnit objekty a celé prostředí a zasadit vše do jiného prostředí. Tedy, je zde prostor o rozšíření letiště, namodelování reálného letiště, podle reálných plánů, nebo jen upravit letištní plochu, pokud to bude vyžadovat rozš íření simulace. Při rozšíření je taktéž nutné klást důraz na nezbytnou část a to na testování a udělat si jednoduché prototypy, napsat unit testy, integrační testy a vymyslet testovací scénáře pro E2E testování.

Výrazně mou diplomovou práci ovlivnil i COVID-19 a situace a nařízení s ním spojená, která mi znesnadňovala plánování a průběh osobních konzultací, ať již s lidmi na katedře nebo mimo ni. Přesto jsem, i přes tyto překážky, věnoval své diplomové práci značné úsilí, snahu a čas, abych ji mohl odevzdat s pocitem, že možná jednou přinese užitek nejen budoucím studentům, ale i bezpečnostním složkám operujícím na potenciálně rizikových místech s velkou koncentrací lidí.

Příloha A

Literatura

[1] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, 2008.

[2] OREILLY, "Understanding left- or right-handed coordinate systems," 12.05.2021. [Online]. Dostupné z: https://www.oreilly.com/library/view/learn-arcore-/9781788830409/03e5338d-02f1-4461-a57a-ef46a976f96b.xhtml

[3] Mojevirtualnirealita, "Zprovoznění htc vive během několika minut," 12.5.2021. [Online]. Dostupné z: https://mojevirtualnirealita.cz/zprovozneni-htc-vive-snadno-a-rychle/

[4] Veative, "Why is gyroscope important for virtual reality?" 10.5.2021. [Online]. Dostupné z: https://www.veative.com/blog/gyroscope-important-virtual-reality/

[5] T. Wild, "Getting started with vr for your architecture and design team in 2021," 10.5.2021. [Online]. Dostupné z: https://thewild.com/blog/architect-getting-started-with-vr

[6] Patro, "Katvr pohybová platforma kat walk mini," 1.5.2021. [Online]. Dostupné z: https://www.patro.cz/katvr-pohybova-platforma-kat-walk-mini/

[7] S. Studios, "What is a docker container? definition," 17.5.2021. [Online]. Dostupné z: https://www.sdxcentral.com/cloud/containers/definitions/what-is-docker-container/

[8] AsterionVR, "Asterion vr® technology blends vr and reality in a mixed reality," 12.04.2021. [Online]. Dostupné z: https://vreducation.cz/virtualni-realita-historie-a-soucasnost/

[9] I. A. T. Association, "Rampvr virtual reality training tool," 12.5.2021. [Online]. Dostupné z: https://www.iata.org/en/training/pages/rampvr/#tab-1

[10] T. Parisi, *Learning Virtual Reality: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile.* O'Reilly Media, Incorporated, 2015, 2015.

[11] V. Motion, "Virtuální realita – historie a současnost," 2.4.2021. [Online]. Dostupné z: https://vreducation.cz/virtualni-realita-historie-a-soucasnost/

[12] U. Engine, "Blueprint visual scripting," 15.04.2021. [Online]. Dostupné z: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/index.html

[13] J. Petty, "What is unity 3d and what is it used for?" 29.04.2021. [Online]. Dostupné z: https://conceptartempire.com/what-is-unity/

[14] Mixamo, "Mixamo - get animated," 15.4.2021. [Online]. Dostupné z: https://www.mixamo.com/#/

[15] U. Technologies, "Inner workings of the navigation system," 19.4.2021. [Online]. Dostupné z: https://docs.unity3d.com/Manual/nav-InnerWorkings.html

[16] A. Thorn, *Pro Unity Game Development with C#.* Apress, 2014.

[17] A. Okita, *Learning C# Programming with Unity 3D.* CRC Press, 2014.

[18] P. F. Navarro, *Learn Unity by Creating a 3D Multi-Level Platformer Game.* Zenva, 2018.

[19] TurboSquid, "Turbosquid: 3d models for professionals," 18.5.2021. [Online]. Dostupné z: https://www.turbosquid.com/

[20] Free3D, "3d models for free," 18.5.2021. [Online]. Dostupné z: https://free3d.com/

[21] CGTrader, "Cgtrader - 3d models for vr / ar and cg projects," 18.5.2021. [Online]. Dostupné z: https://www.cgtrader.com/

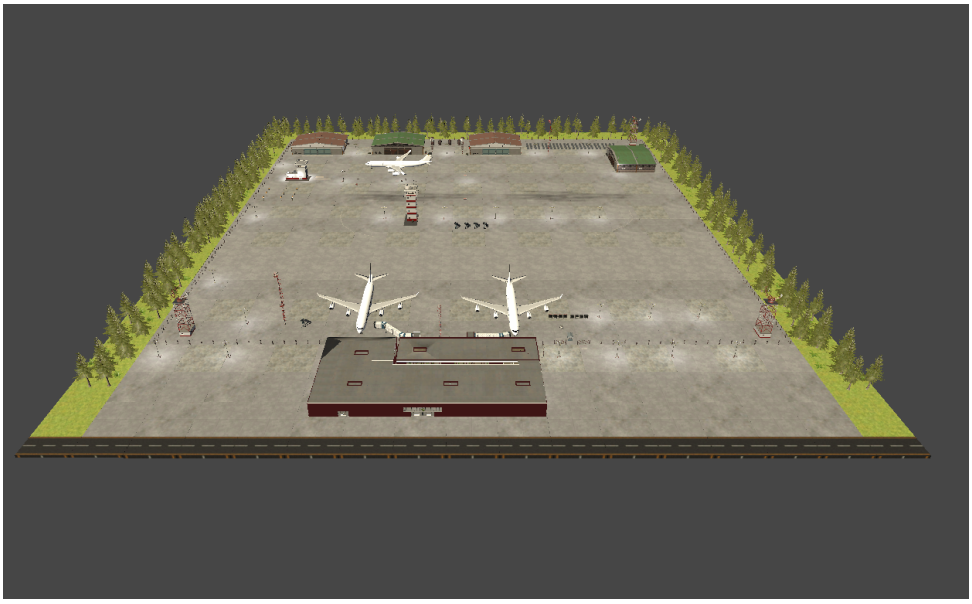
[22] Unity3D, "Unity asset store - the best assets for game making," 18.5.2021. [Online]. Dostupné z: https://assetstore.unity.com/

[23] P. Ammann and J. Offutt, *Introduction to software testing.* Cambridge University Press, 2016.

[24] M. Bureš, M. Renda, M. Doležel *et al.*, *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Grada Publishing as, 2016.

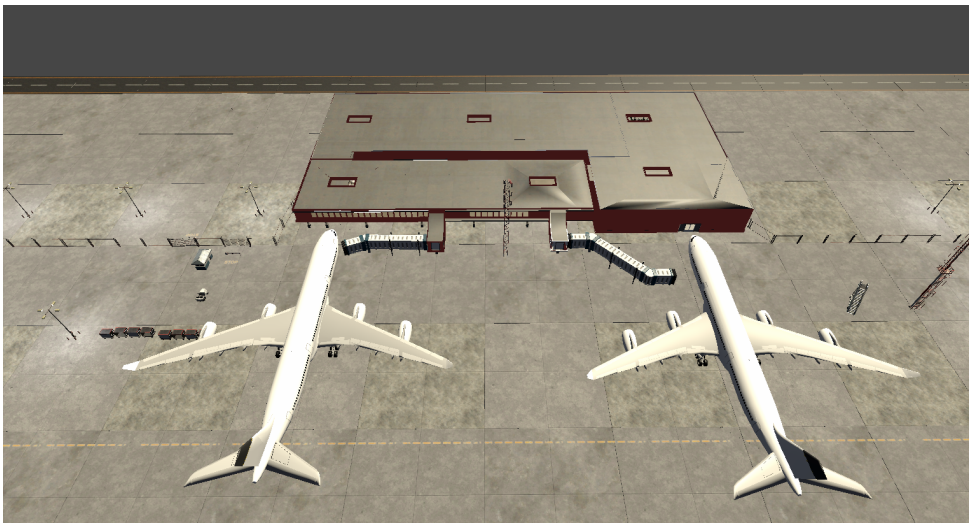
Příloha B

Screenshoty z navrhnutého letištního prostředí

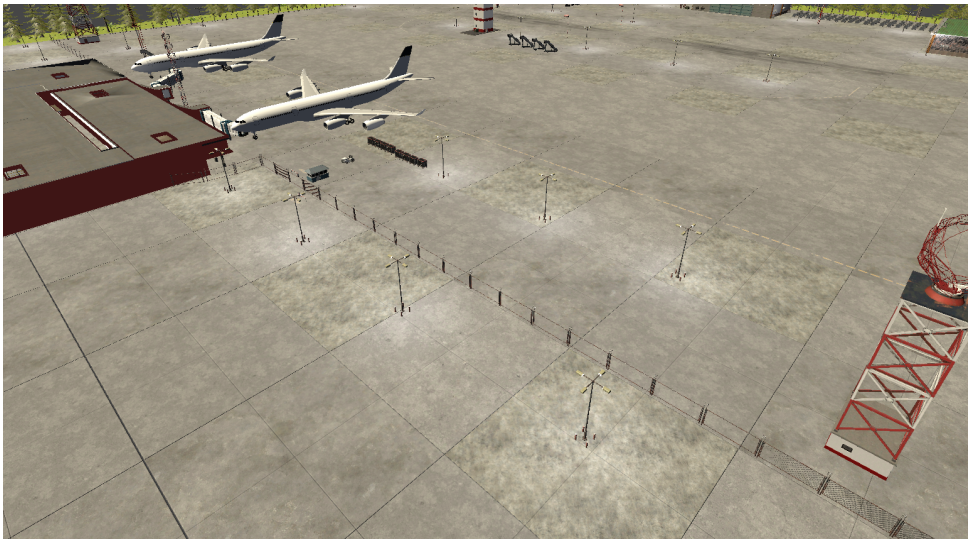


Obrázek B.1: Ukázka venkovního letištního prostoru.

Obrázek B.2: Ukázka venkovního letištního prostoru.

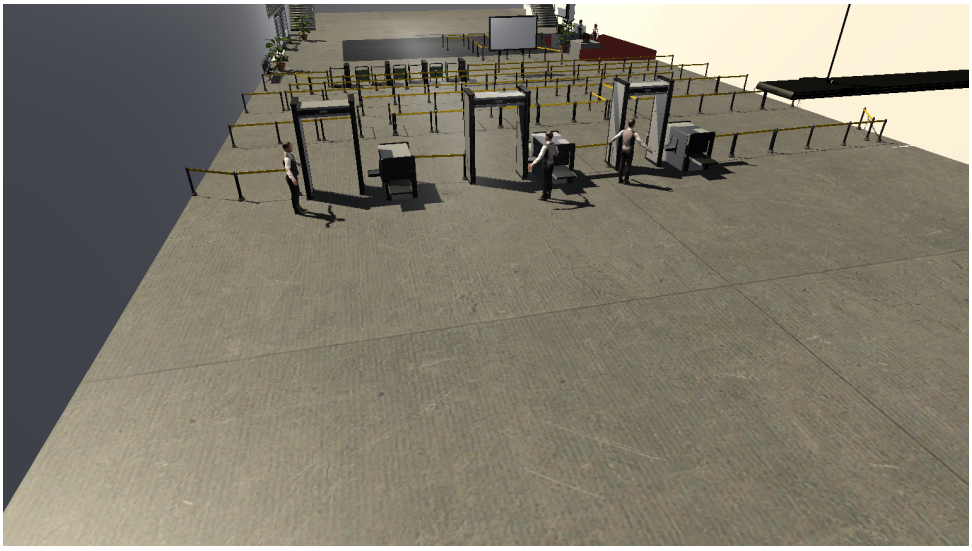


Obrázek B.3: Ukázka venkovního letištního prostoru.

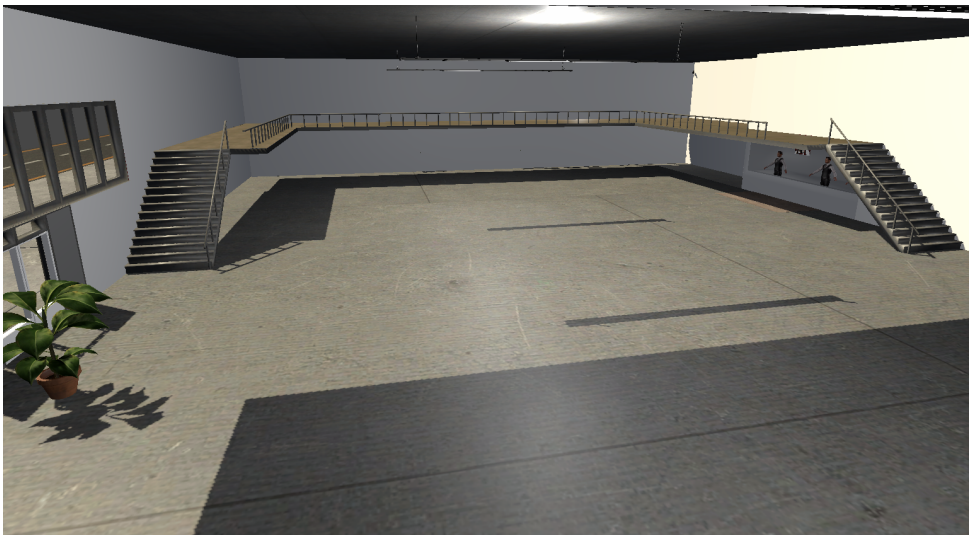


Obrázek B.4: Ukázka venkovního letištního prostoru.

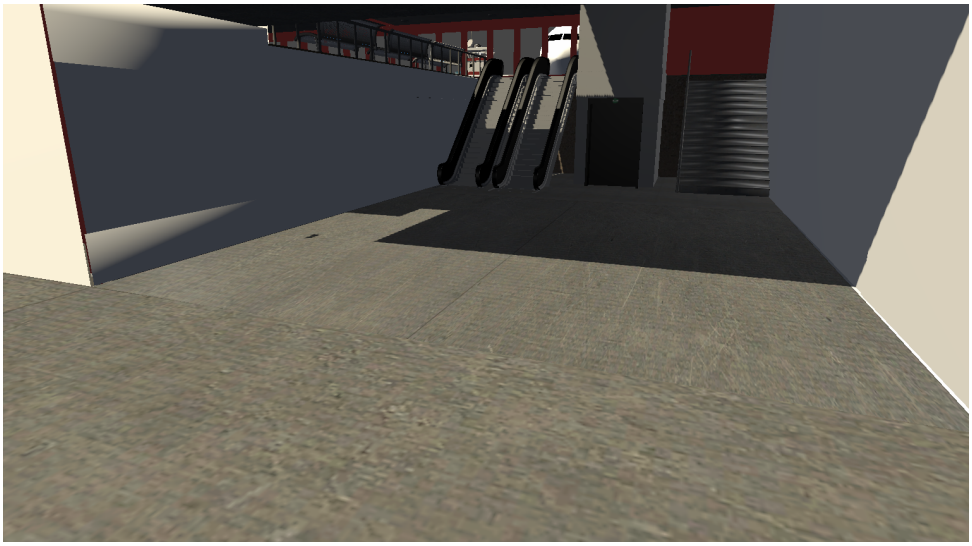
Obrázek B.5: Ukázka příletové haly letištního prostoru.



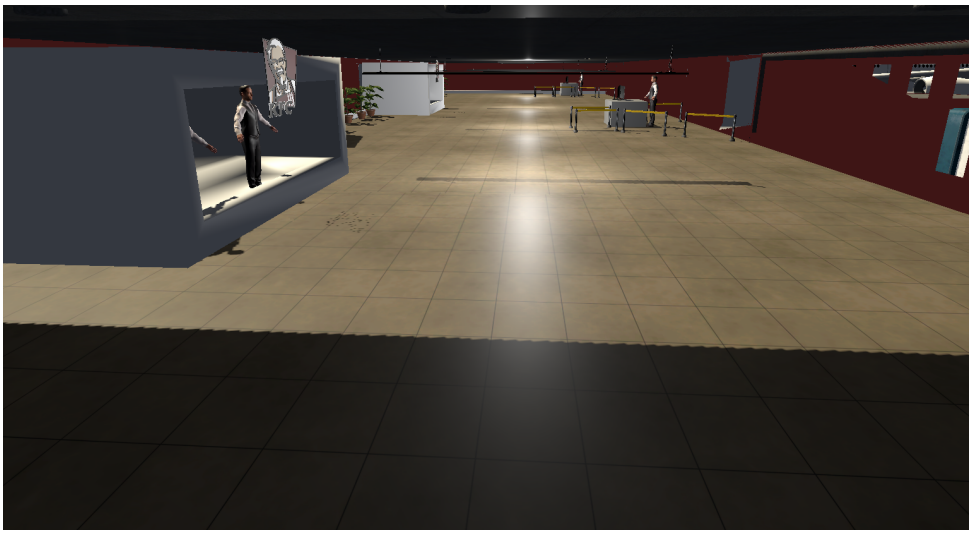
Obrázek B.6: Ukázka odletové haly letištního prostoru.



Obrázek B.7: Ukázka odletové haly letištního prostoru.



Obrázek B.8: Ukázka odletové haly letištního prostoru.



Obrázek B.9: Ukázka odletové haly letištního prostoru.

Příloha C

Seznam použitých zkratek

VR - Virtuální realita
FPS - Frames per second (počet snímků za sekundu) nebo také first-person shooter (střílečka z pohledu první osoby)
MVP - Minimal variable product (minimální životaschopný produkt)
UI - User interface (uživatelské rozhraní)
AR - Augmentová realita nebo také rozšířená realita
HUD - Head-Up display (průhledový displej)
LHS - Left-Handed coordinate system (levotočivý souřadnicový systém)
DFS - Depth-first search (prohledávání do hloubky)
BFS - Breadth-first search (prohledávání do šířky)
UML - Unified Modeling Language
E2E - End-to-end testy
SUT - System under test (model testovacího systému)

Příloha D

Obsah přiloženého CD

```
DP/
-- AirportModel/    ..............  Vytvořený model letiště
-- Documents/       ..............  Adresář, kde jsou umístěné všechny dokumenty
    -- Screenshots/  ..............  Adresář se screenshoty na tweety
    -- Thesis/
       -- Images/
       -- Manual/
-- Project/         ..............  Adresář samotného projektu
    -- UserSettings/
    -- Properties/
    -- ProjectSettings/
    -- Packages/
    -- obj/
    -- Logs/
    -- LocalPackages/
    -- Library/
    -- Assets/        ..............  Adresář s vytvořeným nebo importovaným obsahem
    -- .sonarqube/    ..............  Adresář, kam si SonarQube ukládá vše potřebné
-- Scripts/          ..............  Adresář s užitečnými skripty
-- Tools/            ..............  Adresář s nástroji
    -- SonarScanner/  ..............  Adresář s nástrojem pro statickou analýzu kódu
-- Build/            ..............  Adresář s výsledným softwarem ke spuštění
```

Jednotlivé adresáře a podadresáře v *Project/* není potřeba vysvětlovat, jelikož si některé z nich generuje přímo Unity a pro nás jsou důležité pouze adresáře a soubory v adresáří *Assets/*.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bursík** Jméno: **Filip** Osobní číslo: **457002**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Prototyp systému pro taktické cvičení ostrahy letiště

Název diplomové práce anglicky:

System for Tactical Training of Airport Security

Pokyny pro vypracování:

Navrhněte a implementujte prototyp systému pro taktické cvičení ostrahy letiště pomocí virtuální reality.
Systém umožní simlaci různých scénářů narušení bezpečnosti provozu v prostorech letiště, například teroristický útok, požár nebo rozsáhlejší porucha.
Účastníci cvičení budou mít k dispozici vhodný technický prostředek, kde se budou technikou virtuální reality zobrazovat vybrané simulované prostory letiště a objekty simulující scénář dané situace narušení bezpečnosti.
Systém umožní jednotlivé scénáře vytvořit pomocí vhodného konfiguračního rozhraní nebo pomocí skriptu.
V případě zbývající kapacity systém rozšiřte o část podporující operativní aspekt cvičení, při kterém může velící důstojník sledovat průběh cvičení a zasahovat do jeho průběhu.
Systém otestujte pomocí sady vhodných testů založených na vytváření různých scénářů cvičení.

Seznam doporučené literatury:

Parisi, T. (2015). Learning virtual reality: developing immersive experiences and applications for desktop, web, and mobile. " O'Reilly Media, Inc.".

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Miroslav Bureš, Ph.D., laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2021** Termín odevzdání diplomové práce: **21.05.2021**

Platnost zadání diplomové práce: **30.09.2022**

doc. Ing. Miroslav Bureš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta