ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra počítačů

Využití umělé inteligence v simulátoru pro nácvik taktických a operačních postupů ostrahy letiště

Artificial intelligence in the simulator for training tactical and operational procedures of airport security

Diplomová práce

Studijní program: Otevřená informatika

Studijní obor: Softwarové inženýrství

Vedoucí práce: Ing. Karel Frajták, Ph.D.

Bc. Vladimír Glingar

Praha 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Glingar** Jméno: **Vladimír** Osobní číslo: **457873**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Využití umělé inteligence v simulátoru pro nácvik taktických a operačních postupů ostrahy letiště

Název diplomové práce anglicky:

Artificial intelligence in the simulator for training tactical and operational procedures of airport security

Pokyny pro vypracování:

Rozšiřte stávající simulátor pro nácvik taktických a operačních postupů ostrahy letiště vytvořený jako DP Ing. Filipa Bursíka o komplexnější chování útočníků, cestujících a pracovníků letiště. V chování útočníků využijte postupy a algoritmy umělé inteligence a učení se z předchozích běhů simulací. Chování útočníků vhodně parametrizujte. Výsledný kód otestujte sadou vhodných testů a celý postup zdokumentujte.

Seznam doporučené literatury:

Chowdhary, K. R. (2020). Fundamentals of artificial intelligence. Springer Nature.
Russell S., Norwig P. (2009) Artificial Intelligence – A Modern Approach (3rd Edition). Prentice Hall.
Parisi T. (2015) Learning Virtual Reality: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile. O'reilly Media

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Karel Frajták, Ph.D. laboratoř inteligentního testování systémů FEL

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

Datum zadání diplomové práce: **09.02.2022** Termín odevzdání diplomové práce: __________

Platnost zadání diplomové práce: **30.09.2023**

Ing. Karel Frajták, 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

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Prohlášení

„Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.“

Praha, 9. 2. 2022

Vladimír Glingar

Poděkování

Rád bych poděkoval panu Ing. Karlu Frajtákovi, Ph.D. za vedení práce a cenné rady a připomínky. Poděkování rovněž patří panu doc. Ing. Miroslavu Burešovi, Ph.D. za vedení v rámci samostatného projektu, který této diplomové práci předcházel. Zároveň chci poděkovat rodičům za podporu.

Abstrakt

Předmětem této diplomové práce je rozšíření simulátoru pro nácvik taktických a operačních postupů ostrahy letiště, vytvořený Ing. Filipem Bursíkem, o komplexnější chování jednotlivých postav vyskytujících se v simulaci s využitím umělé inteligence a učení se z předchozích běhů simulací. Práce se skládá z pěti kapitol. První dvě kapitoly se zabývají umělou inteligencí a problematikou práce, včetně analýzy dodaného kódu. Třetí kapitola popisuje implementované funkcionality a ve čtvrté se věnuji testování simulátoru. Stručné shrnutí provedených úprav je uvedeno v kapitole páté.

Klíčová slova

Umělá inteligence, inteligentní agent, simulátor, virtuální realita, Unity, strojové učení

Abstract

The subject of this diploma thesis is the extension of the simulator for training tactical and operational procedures of airport security, created by Ing. Filip Bursík, on the more complex behavior of individual characters occurring in the simulation with the use of artificial intelligence and learning from previous runs of simulations. The thesis consists of five chapters. The first two chapters deal with artificial intelligence and issues, including the analysis of the supplied code. The third chapter describes the implemented functionalities and the fourth deals with testing the simulator. A brief summary of the adjustments made is given in chapter five.

Key words

Artificial Intelligence, Intelligent Agent, Simulator, Virtual Reality, Unity, Machine Learning

Obsah

Obsah **9**
1 Úvod **11**
1.1 Problematika a cíl práce 12
2 Analýza problematiky **13**
2.1 Definice technických pojmů 13
2.2 Popis simulátoru 13
2.3 Analýza dodaného kódu 14
2.4 Umělá inteligence 15
2.4.1 Různé pohledy na umělou inteligenci 16
2.4.2 Oblasti využití umělé inteligence 18
2.4.3 Umělá inteligence v herním průmyslu 19
2.5 Inteligentní Agent 21
3 Vývoj a implementace **23**
3.1 Vyhledávání potenciálních cílů 23
3.2 Držení cílů „pod kontrolou“ 27
3.3 Enemy sanity 28
3.4 Chování cestujících a zaměstnanců 29
3.5 Správce statistických dat 29
3.6 Nahrání, ukládání a resetování paměti 31
3.7 Výběr optimálního místa 32
3.8 Escape procedura 33
3.9 Identifikace útočníka cestujícími 36
3.10 Trigger konce hry 37
3.11 Oživení pracovníků letiště 38
3.12 Výpočet herního skóre 38
3.13 Chování útočníka na ranveji 39
3.14 Shrnutí 40
4 Testování **43**
4.1 Testování identifikace cílů 44
4.2 Testování escape procedury 45
4.3 Další testy a opravy chyb 46
4.4 Testování po dokončení vývoje 47
4.4.1 Testování bez zásahu uživatele 47

4.4.2 Testování se zásahem uživatele (obranné jednotky) 49
4.5 Shrnutí 51
5 Závěr 53
6 Zdroje / použitá literatura 57
7 Seznam příloh 61
8 Přílohy 63

1 Úvod

V současné době existuje nemálo počítačových her zajišťujících simulaci různých situací. V posledních letech se na trhu pro běžné uživatele objevila nová technologie - virtuální realita. Tato nová technologie povznesla o úroveň výše realističnost tehdy populárních 3D her (emulujících 3D prostor na obrazovku počítače). Uživatel je přenesen do virtuálního prostoru, simulujícího dané prostředí, a může interagovat s objekty podobně jako v reálném životě. Je nutné si však uvědomit, že v dnešní době veškeré nové technologie před uvedením pro běžné uživatele jsou testovány a použity v armádním sektoru. Jako veřejně známý příklad může sloužit např. simulátor AsterionVR [1] využívaný pro trénink speciálních složek. Rovněž je technologie virtuální reality (VR) dnes využívána v kombinaci s drony. Toto umožňuje přenos obrazu uživateli s VR brýlemi a ten může pomocí nich dron ovládat.

Dále pak stojí za zmínku simulátor RAMPVR [2], sloužící pro trénink pozemních jednotek letiště (vlastníkem je IATA – International Air Transport Association).

Prototyp podobného simulátoru letiště vytvořil ve své diplomové práci Ing. Filip Bursík [3]. Simulátor by měl sloužit pro nácvik taktických a operačních postupů ostrahy letiště. Ing. Filip Bursík uvádí, že téma jeho diplomové práce bylo inspirované jednou izraelskou společností, která vytvořila software pro VR, používaný jako trenažér pro vojáky ve výslužbě.

Cílem jeho diplomové práce bylo vytvořit minimální životaschopný produkt, který bude možné později rozšířit. Měl vzniknout prototyp systému pro taktické cvičení ostrahy letiště v různých situacích ohrožujících bezpečnost cestujících, během nichž zásahová jednotka musí tyto bezpečnostní hrozby eliminovat.

Ing. Filipu Bursíkovi se podařilo vytvořit projekt, který simuluje provoz na letišti a také modeluje chování 3 různých typů útočníků, letištního personálu a cestujících.

Projekt byl navržen tak, že jednotka měla pouze jeden pokus na zvládnutí simulace a personál letiště a cestující se při zahájení útoku okamžitě vzdávali. Zároveň byla v diplomové práci navrhnuta vhodná architektura, umožňující pokračování a rozšiřování projektu. Ing. Bursík píše: „*Při rozšíření je 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*“ [3, str. 56].

1.1 Problematika a cíl práce

Ing. Filip Bursík v rámci své diplomové práce vytvořil prototyp simulátoru pro nácvik taktických a operačních postupů pracovníků ostrahy letiště.

Cílem mé práce je rozšířit a upravit simulátor za použití umělé inteligence a strojového učení tak, aby bylo docíleno větší realističnosti chování jednotlivých postav v simulátoru.

Stejně jako původní diplomová práce bude vývoj pokračovat v aplikaci Unity (verze 2019.4.23f1), ale s tím rozdílem, že simulátor bude vyvíjen pro VR headset „*Oculus Rift S*“ [4] oproti headsetu „*HTC Vive*“ [5]. Vzhledem k tomu, že aplikace je spouštěna na platformě **SteamVR** [6], která je společná jak pro **Oculus**, tak pro **HTC**, neměly by zde být vážnější problémy kromě konfigurace ovladačů.

2 Analýza problematiky

Než se budu věnovat problematice rozšíření simulátoru navrženého v diplomové práci Ing. Bursíka (dále: původní koncept simulátoru), je nezbytné popsat výchozí stav simulátoru a definovat některé technické pojmy.

2.1 Definice technických pojmů

Níže uvádím seznam některých technických pojmů, používaných při tvorbě počítačových her (a simulací), které jsou použity dále v práci.

- Collider – neviditelný objekt, který pokrývá postavu a snímá kolize s ostatními objekty
- Mesh – kolekce bodů, hran a stěn definující objekt
- NPC – non-playable character – postava řízena počítačem (nehratelná postava)
- spawnutí – proces vytvoření postavy a její umístění do simulace
- spawnpoint – výchozí místo postavy
- VR – virtuální realita.

2.2 Popis simulátoru

Simulátor zachycuje komplexní simulaci provozu na letišti, tj.:

- proces příletu a odletu letadel
- proces odbavení cestujícího od příchodu na letiště až po nastoupení do „prstu“ k letadlu
- proces příletu cestujících a jejich odchod z letiště.

Celkově zde máme 6 typů postav (NPC):

- Cestující – odlétající
- Cestující – přilétající
- Obsluha letiště (personál)
- Nepřítel – útočící z galerie
- Nepřítel – útočící ze střešního okna
- Nepřítel – útočící na ranvej.

Chování cestujících i útočníků je řízeno stavovým automatem. Pro odlétající cestující vyjadřují jednotlivé stavy proces odbavení cestujícího. Stejně tak pro přilétající cestující. Pro útočníka vyjadřují jednotlivé stavy proces útoku.

V simulátoru jsou aktuálně definovány tři scénáře útoku, přičemž každý má vlastní typ útočníka (a každý z těchto útočníků má vlastní skript):

- Útok z galerie veřejného prostoru (scénář 1)
- Narušení integrity pláště budovy (scénář 2)
- Vniknutí na ranvej (scénář 3).

Ve scénáři 1 se mezi odlétající cestující spawnuje útočník z galerie. Tj. objeví se před budovou letiště, poté jde spolu s ostatními cestujícími do letištní haly na zadané místo útoku. Jakmile tam dojde, vytasí zbraň a útok začíná. Útočník následně vybere náhodný cíl a ten po náhodném čase zastřelí.

Scénář 2 je podobný předchozímu scénáři, ale s tím rozdílem, že útočník proskočí stropním oknem, dojde na místo útoku a poté spustí útok.

Scénář 3 je specifický v tom, že se neodehrává v letištní budově a útočník nemá zbraň. Jedná se o případ vniknutí útočníka na ranvej, kde může ohrozit provoz letadel.

Jednotlivé procesy, scénáře útoku a NPC jsou řízeny různými komponentami simulátoru (alias manažery), které mezi sebou v rámci definované architektury komunikují. Například útočník je spawnován pomocí komponenty **SpawnScript**, která komunikuje s komponentou **AttackManager**, jenž dále útočníka řídí v průběhu jeho útoku.

2.3 Analýza dodaného kódu

Vzhledem k zadání práce bylo nezbytné využít dodaný kód simulátoru a upravit ho s použitím postupů a algoritmů umělé inteligence.

Prvním krokem bylo vizuální seznámení se s fungováním simulátoru, jehož účelem bylo zároveň identifikovat nedostatky původního chování postav. Byly identifikovány následující problémy:

- V případě útoku se všichni cestující a personál letiště vzdají, nehledě na svou pozici vůči útočníkovi. Přitom z logiky věci mohou být mimo bezprostřední ohrožení ze strany útočníka.
- Útočník se zaměřuje pouze na jeden cíl, ostatní potenciální cíle ignoruje. Z hlediska chování útočníku v realitě, je pro něj žádoucí držet všechny potenciální cíle ve stálém ohrožení, aby žádný z cílů neutekl (popř. nespustil poplach).
- Útočník v rámci výběrů potenciálních cílů úplně ignoruje personál letiště, což je rovněž nerealistické.
- Množina potenciálních cílů útočníka na ranvej neobsahuje zajímavá místa (např. řídící budovy). Pokud by se jednalo o atentátníka nebo sabotéra, měl by se zaměřit na konkrétní místa, aby vyřadil letiště z provozu.
- Jednotka má pouze jeden pokus na zvládnutí simulace. Úmrtím některé z postav letiště simulace končí. Toto je nežádoucí z pohledu příp. sběru dat a učení se. Simulace jsou příliš krátké a nelze odhadnout nebezpečnost útočníka či rizikovost místa útoku.

Pro detailní analýzu nedostatkům, identifikovaných v předchozím kroku, bylo nezbytné je nalézt v kódu a poté rozhodnout o způsobu jejich odstranění. Vyřešením těchto problémů bude dosaženo větší komplexity chování jednotlivých postav a zároveň i větší realističnosti, což je jeden ze zadaných cílů práce.

Dalším cílem je implementace učení se z předchozích simulací, které se v původním konceptu simulátoru nevyskytuje.

2.4 Umělá inteligence

Vzhledem k tomu, že tématem práce je využití umělé inteligence v určité oblasti, je nezbytné se nejprve věnovat teorii umělé inteligence.

2.4.1 Různé pohledy na umělou inteligenci

Samotný pojem „umělá inteligence" je relativně nový. Přesto myšlenky, které vedly ke vzniku tohoto pojmu, lze datovat až do 17. století, kdy dlouhou dobu před vznikem počítačů byla kladena otázka „Mohou-li stroje myslet?" [7, str. 15]. Rok 1956 je považován za položení základního kamene vědního oboru „umělá inteligence", když na konferenci, organizované Johnem McCarthym, byl poprvé definován název a obsah tohoto oboru.

Termín „umělá inteligence" (angl. artificial intelligence, zkráceně AI) má vícero definicí. Jedna z možných definic pochází z roku 1967 a patří významnému americkému vědci Marvinu Lee Minskému, který byl spoluzakladatelem laboratoře umělé inteligence na univerzitě MIT. Jeho definice se zakládá na „Turingově testu" [1] a zní: „*Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení úkolu užívat takového postupu, který – kdyby ho dělal člověk – bychom považovali za projev jeho inteligence*" (překlad a citace z [7, str. 17]).

Další možná definice, uvedená v roce 1990 v knize „Artificial Intelligence", je od Richové a Knighta. Volně přeloženo by se dalo říci, že z jejich pohledu cílem oboru je neustále zlepšovat a vylepšovat počítače tak, aby byly schopné řešit úlohy, které jsou dnes řešené člověkem lépe a efektivněji [2] [8, str. 3].

Autor knihy „Artificial Intelligence" Richard Urwin uvádí následující definici umělé inteligence: „*Umělá inteligence je počítačový program, který nehledě na své umístění (samostatné či vestavěné) vykazuje vnější znaky inteligence*" [3] (vlastní překlad [10, str. 9]).

Urwin dále rozlišuje následující tři typy umělé inteligence: silný, slabý a pragmatický. Silný typ (*strong AI*) předpokládá, že je možné vytvořit stroj myslící stejně jako člověk. Slabý typ (*weak AI*) považuje za dostačující, pokud se stroj chová dostatečně inteligentně. Pragmatický typ (*pragmatic AI*) předpokládá, že místo snahy vytvořit stroj s inteligencí srovnatelnou s lidskou, by se mělo zaměřit na jiné bytosti, než je člověk, a pokusit se např. emulovat chování zvířat [10, str. 7-8].

---

[1] Turingův test, vyvinutý v roce 1950 a pojmenovaný po svém autoru Alanu Turingovi, je vědecký pokus mající za cíl zjistit, zda počítač dokáže napodobit lidské myšlení [9].

[2] „*AI is the study of how to make computers do things, which, at the moment, people do better*" [8, str. 3].

[3] „*It is a computer program, whether standing alone in a date centre or a PC or embodied in a device such as a robot, which displays the outward signs of being intelligent*" [10, str. 9].

Další pohled na problematiku umělé inteligence nalezneme v knize Russella a Norviga „*Artificial Intelligence A Modern Approach*“ [11]. Autoři uvádí 8 různých definic umělé inteligence a rozdělují je do čtyř skupin (dle pohledů na umělou inteligenci): uvažující jako lidé (*Thinking Humanly*), chovající se jako lidé (*Acting Humanly*), uvažující racionálně (*Thinking Rationally*) a chovající se racionálně (*Acting Rationally*).

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense." (Haugeland, 1985)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985)
"[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998)
"The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"AI ... is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figure 1.1 Some definitions of artificial intelligence, organized into four categories.

Obrázek 2.1: Definice umělé inteligence dle kategorií [11, str. 2]

Umělá inteligence, spadající do kategorie *Thinking Humanly*, je spojená s kognitivními schopnostmi lidského mozku a předpokládá schopnost rozhodování, řešení problémů a získávání zkušeností z vlastních chyb. Problémem však je, že dnešní věda prozatím nebyla schopna jednoznačně definovat, jak probíhá v lidském mozku proces myšlení.

Základem problematiky kategorie *Acting Humanly* je již zmíněný Turingův test. Snahou je přiblížit umělou inteligenci té lidské včetně všech kladů a záporů. Za zmínku rovněž stojí, že pod tuto problematiku rovněž spadá i opačné odlišení člověka a stroje, využívané v ověřovacích metodách „CAPTCHA“.

Kategorie *Thinking Rationally* se oproti kategorii *Thinking Humanly* zabývá obecnou logikou. Pod tuto kategorii spadá např. „rezoluční metoda“ [12], využívaná v logickém programování pro automatické dokazování tvrzení.

Do poslední kategorie *Acting Rationally* řadíme stroje, které přemýšlí a konají racionálně (tj. např. z dostupných strategií vyberou tu nejlepší na základě definované metriky). Tato oblast bude dále klíčová pro mou práci, protože do této kategorie spadá i problematika tzv. inteligentních agentů (viz kapitola 2.5).

2.4.2 Oblasti využití umělé inteligence

V předchozí podkapitole byly popsány pohledy různých autorů na téma umělé inteligence. Oblastí, ve kterých se dnes umělá inteligence využívá, je velké množství, přesto považuji za vhodné některé alespoň stručně zmínit [13].

1) **Robotika** je obor zabývající se vývojem a zdokonalováním robotů.
2) **Plánování** je oblast zabývající se algoritmy určenými k tvorbě časových rozvrhů a k optimalizaci (např. výrobních) plánů.
3) **Strojové učení** (angl. machine learning) je podoblast umělé inteligence, zabývající se využitím statistických dat a algoritmů pro tvorbu modelů a předpovídání dat.
4) **Zpracování přirozeného jazyka** – oblast zabývající se počítačovým zpracováním lidského jazyka a jeho interpretací.
5) **Rozpoznávání řeči** se zabývá převodem mluveného slova do textu.
6) **Expertní systémy** jsou specifické programy, které z definované báze znalostí pomocí řídícího mechanismu utváří závěry.
7) **Počítačové vidění** je obor zabývající se porozuměním a získáním informací z obrazových dat.

2.4.3 Umělá inteligence v herním průmyslu

Mezi další oblast využití umělé inteligence by se dal zařadit herní průmysl. Ruku v ruce s vývojem technologií se rozvíjely i počítačové hry, mj. v důsledku nároků kladených herní komunitou.

Vzhledem k tomu, že v dnešní době si herní studia střeží svůj kód, je velmi obtížné určit, zda a jak je umělá inteligence ve hrách využita. Je nutné si však uvědomit, že v rámci počítačových her není většinou využívaná umělá inteligence v akademickém slova smyslu, ale „umělá inteligence z herního pohledu“.

Jeden z předních vývojářů a průkopníků v oblasti herní umělé inteligence Alex J. Champandard ve své knize *AI Game Development: Synthetic Creatures With Learning and Reactive Behaviors* odděluje umělou inteligenci z akademického pohledu od herní umělé inteligence následujícím způsobem: „*Z pohledu her je pro NPC dostačující vykazovat omezenou formu inteligence, pokud je jejich chování věrohodné. Tudíž klíčovým faktorem je finální výsledek. Z vnějšího pohledu nelze využití umělé inteligence ve hrách ověřit, protože realistických výsledků se dá dosáhnout i s využitím standardních softwarových technik (např. scriptingu)*“[4] (vlastní překlad [14, str. 7]). Z tohoto vyplývá, že cílem počítačové umělé inteligence je vytvoření „iluze inteligence“.

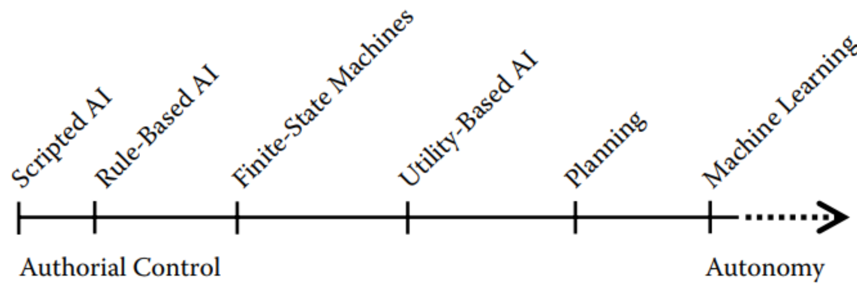
O iluzi inteligence v počítačových hrách se zmiňuje rovněž Kevin Dill, který píše: „*Termín herní umělá inteligence využíváme k popisu umělé inteligence, která je zaměřena na vytvoření iluze inteligence a na vytvoření konkrétního zážitku pro diváka, spíše než na vytvoření skutečné inteligence, jaká se vyskytuje u lidí*“[5] (vlastní překlad [15, str. 4]).

Kevin Dill dále tuto problematiku vystihuje obrázkem, který popisuje rozdíly mezi jednotlivými technikami umělé inteligence [15, str. 5 - 6].

---

[4] „*From an external point of view, NPCs need only to display a certain level of intelligence. This is one key realization; computer game AI requires the result. It doesn't really matter how NPC intelligence is achieved, as long as the creatures in the game appear believable. So AI technology is not justifiable from this outsider's point of view, because standard software engineering techniques could be used equally well to craft the illusion of intelligence (for instance, scripting)*“ [14, str. 7].

[5] „*We use the term game AI to describe AI, which is focused on creating the appearance of intelligence, and on creating a particular experience for the viewer, rather than being focused on creating true intelligence as it exists in human beings*“ [15, str. 4].



Obrázek 2.2: Vybrané techniky umělé inteligence [15, str. 6]

Jak bylo zmíněno výše, rozhodujícím faktorem dle Dilla je finální výsledek, resp. zážitek ze hry. Aby bylo možné dosáhnout zážitku dle očekávání autorského týmu, je potřeba zachovat určitou formu kontroly nad chováním hry (na obrázku 2.2 popsána jako „Authorial Control“).

Nejvíce pod kontrolou autora je dle obrázku tzv. *Scripted AI* („skriptovaná počítačová umělá inteligence“). Tento název však pouze označuje, že postava vykonává sekvenci příkazů v přesně daném pořadí a z akademického významu je diskutabilní, zda se o umělou inteligenci jedná, či nikoliv.

Dalším uvedeným typem je *Rule-Based AI* („umělá inteligence definována pravidly“). Takto definovaná umělá inteligence funguje na principu pevně definovaných pravidel (tj. *if-then* logika).

Komplexnějším přístupem jsou *Finite-State Machines* („konečné stavové automaty“). Oproti předchozím přístupům umožňuje složitější interpretaci chování NPC, kdy v každém stavu je chování různé. Přechod mezi jednotlivými stavy zajišťuje přechodová funkce.

Více autonomní technikou je tzv. „Utility-Based umělá inteligence“, fungující na principu ohodnocení jednotlivých možností a volbou „nejlepší“ varianty. Tuto techniku využívají tzv. „inteligentní agenti“, jejichž problematika je popsána v následující kapitole. Problematika výběru optimální možnosti je rovněž velmi blízká úlohám lineárního programování, které se zabývají hledáním minima či maxima kriteriální funkce pro danou množinu přípustných řešení.

Zbylé dvě metody jsou z pohledu herní umělé inteligence spíše teoretickým konceptem, než že by byly často aplikovány. Problematika plánování typově spadá mezi

optimalizační úlohy a určité typy těchto úloh lze opět řešit pomocí lineárního programování. Strojové učení jako celek spadá již do akademické umělé inteligence a v počítačových hrách je málokdy využito.

Jak dále uvádí Kevin Dill, není pevně definováno, která z technik je nejlepší. Vždy záleží na konkrétním zaměření hry. Jako jeden z příkladů uvádí hru *World of Warcraft*, která i přes to, že je v ní chování NPC primárně založeno na skriptované umělé inteligenci, má mnoho fanoušků a proto i nadále vznikají nové data-disky (verze hry).

Jedním z druhů počítačových her jsou simulátory. Za zmínku stojí česká hra *ArmA,* označovaná autory za taktickou střílečku z vojenského prostředí [16]. Jako ukázku aktuálních možností propojení virtuální reality, simulace a nepochybně i technik umělé inteligence považuji známou hru *Half-Life: Alyx*, kombinující střílečku a dobrodružnou hru, zasazenou do fiktivního post-apokalyptického světa a obohacenou o nespočet logických úloh a rébusů [17].

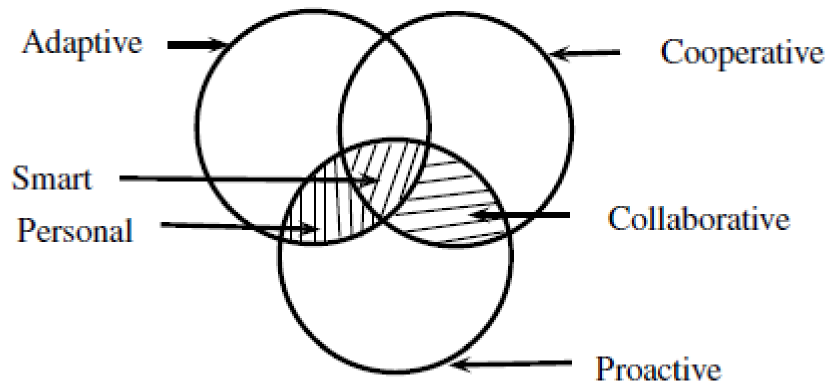
2.5 Inteligentní Agent

V původním simulátoru je několik typů postav, přičemž každý typ vykonává svou činnost, do jisté míry nezávisle na ostatních. Z pohledu využití umělé inteligence v tomto simulátoru se mi jevilo nejvhodnější použít koncept inteligentních agentů.

Russel a Norvig definují „racionálního agenta“ takto: „*Pro každou sekvenci vnímání by měl racionální agent na základě vestavěných znalostí a znalostí ze sekvence vnímání zvolit akci, která předpokládá maximalizaci účelové (výkonnostní) funkce*“[6] (vlastní překlad) [11, str. 37].

Podobný koncept inteligentního agenta uvádí i K. R. Chowdhary ve své knize *Fundamentals of Artificial Intelligence*, kde rozlišuje vícero typů inteligentních agentů na základě jejich vlastností a uvádí je na následujícím obrázku [18, str. 474].

[6] „*For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has*“ [11, str. 37].



Obrázek 2.3: Typy inteligentních agentů dle funkčnosti [18, str. 474]

Na Vennově diagramu vidíme 3 základní typy agentů z pohledu Chowdharyho. Adaptivní (*adaptive*) agent je schopný se učit z předchozích zkušeností a na základě nich mění své chování. Proaktivní (*proactive*) agent dokáže konat bez nutnosti vstupu z vnějšku (např. interakce od jiného agenta nebo příkazu uživatele). Kooperativní (*cooperative*) agent je schopen komunikovat s ostatními agenty a měnit chování na základě této komunikace. Kombinuje-li agent tyto tři vlastnosti, považuje ho Chowdhary za chytrého (*smart*) [18, str. 473 - 474].

Cílem mnou provedených úprav bude snaha přiblížit se tomuto modelu tak, aby útočníka bylo možné považovat za chytrého agenta dle definice uvedené výše. Tj. agent by se měl být schopen učit ze získaných dat, některé jeho akce by měly být realizované bez vnějšího podnětu a dále by agent měl být schopen komunikovat s ostatními agenty a případně upravovat chování dle získaných informací.

3 Vývoj a implementace

Základní komponentou C# Unity skriptu jsou funkce *Start* a *Update*. Funkce *Start* je volaná při inicializaci objektu, kdežto funkce *Update* je volána každý frame. Dále je zde funkce *Awake*, která je spuštěna, jakmile je skript nahrán do Unity prostředí. Tj. je spuštěna ještě před funkcí *Start*. Funkce *Start* se oproti funkci *Awake* nespustí, pokud je komponenta zakázaná (disabled) [19].

V podkapitolách níže se budu detailně věnovat úpravám, které byly provedeny. Pro přehlednost jsou úpravy rozdělené do několika částí dle funkcionalit. Zároveň byla snaha reflektovat pořadí, ve kterém byly jednotlivé úpravy provedeny.

3.1 Vyhledávání potenciálních cílů

Jak bylo již krátce zmíněno v kapitole 2.2, chování postav je řízeno stavovým automatem. Po spawnutí útočníka, které zajištuje **PeopleSpawnScript**, je tímto skriptem volána funkce *StartAttack*, dávající příkazy k útoku.

Jakmile jsou komponentě NavMeshAgent[7] předány cílové souřadnice, je na pozadí vypočtena optimální trasa a útočník se vydá na místo. Po příchodu na místo určení jsou provedeny přípravy k útoku a mimo jiné výběr cílů.

V původním konceptu simulátoru zajišťovala výběr cílů funkce **AttackManagera** *GetTarget*, která si interně do seznamu uložila cíle na dostřel a poté z něj vybrala jeden náhodný cíl. Pokud byla identifikována na dostřel obranná jednotka, pak byla preferenčně vybrána ona.

Logika výběru byla mnou kompletně přepsána. Jak jsem již zmínil v kapitole 2.3, pro útočníka není žádoucí zaměřit se na jeden z cílů a ostatní ignorovat.

Proto byla v rámci **AttackManageru** vytvořena nová funkce *GetReachableTargets*, která pro daného útočníka vrátí seznam dostupných cílů, se

[7] NavMeshAgent je vestavěná funkcionalita Unity zajišťující výpočet a provedení optimální trasy mezi dvěma body [20]. Zajímavostí je, že je při tom využíván algoritmus A*, který je velmi populární v oblasti umělé inteligence [21].

kterými může útočník dále pracovat. Nová funkce bere rovněž v potaz personál letiště jako legitimní cíle.

```
public List<GameObject> GetReachableTargets(GameObject currentObject){
    List<GameObject> possibleTargets = new List<GameObject>();
    Vector3 fromPosition = currentObject.transform.position;

    //najdi vsechny utocitelne zamestnance
    GameObject[] objects = GameObject.FindGameObjectsWithTag("AirportEmployee");
    foreach (GameObject employeeObject in objects) {···
    }

    //najdi vsechny utocitelne lidi
    GameObject[] objects2 = GameObject.FindGameObjectsWithTag("HumanDeparture");
    foreach (GameObject humanDepartingObject in objects2) {···
    }

    return possibleTargets;
}
```

Obrázek 3.1: Ukázka kódu pro výběr cílů

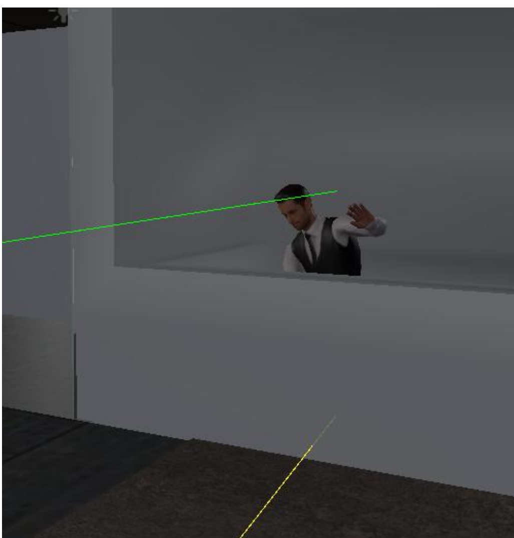
Na obrázku výše je zobrazena část kódu zajišťující výběr cílů. Vzhledem k tomu, že pro každou postavu je zjišťováno, zda není mrtvá (tj. ve stavu *Dead*), bylo nezbytné oddělit vyhledávání cílů z řad personálu letiště od vyhledávání cílů z řad cestujících. Je tomu tak z důvodu rozdílných skriptů.

Pro každou z postav je následně proveden test dosažitelnosti subjektu. Ten spočívá v tom, že je pomocí vestavěné Unity funkce *Physics.Raycast* vržen paprsek. Funkce obdrží výchozí bod a směr paprsku a pomocí klíčového slova „out“ vrátí *RayCastHit*, jenž v sobě obsahuje odkaz na první objekt, na který paprsek narazil. Analýzou tagu získaného objektu ověří, zda se opravdu jedná o cestujícího nebo o personál, a pokud ano, pak to znamená, že subjekt je na dostřel a lze ho přidat na seznam potenciálních cílů.

V rámci testování této funkce byly odhaleny dva nedostatky. Ukázalo se, že souřadnice cílového objektu, které jsou klíčové pro výpočet směru paprsku, vracejí souřadnice nohou příslušné postavy (resp. „spodku“ postavy). Pokud by se postava nacházela za nějakou nízkou překážkou (např. pracovník obsluhy za přepážkou nebo cestující za zábradlím), pak by funkce vyhodnotila, že subjekt není na dostřel, i přestože by tomu tak nebylo.

Potenciálním řešením problému, které mne napadlo, bylo zaměřit hlavu postavy a vůči hlavě vrhat paprsek. Identifikovat umístění hlavy (resp. výšky hlavy) se ukázalo jako netriviální problém. Problematikou určení rozměru postavy se zabýval již někdo dříve a k tomuto tématu jsem nalezl odkaz na Unity fóru [22]. Bohužel se mi nepodařilo využít rady, uvedené na fóru, a proto jsem se nakonec uchýlil k řešení ve stylu KISS („Keep it simple, stupid“). Po několika pokusných měřeních jsem určil výšku postav ručně. Hodnota je uložena v konstantě *char_height* a byla určena na 1.7f. Pokud by se v budoucnosti měnila výška postav jako takových, bylo by vhodné tuto konstantu přepočítat a změnit.

Zaměření potenciálního cíle tedy probíhá vůči hlavě postavy a spodku postavy. Pro eliminaci případného chybného zaměření se pro cestující dále provádí zaměření vůči střední výšce postavy, pokud předchozí zaměření nejsou úspěšná.



Obrázek 3.2: Ukázka zaměření cíle v Unity – ke spodku a k hlavě

Na dalším obrázku je zobrazena finální část kódu, prováděná v části vyhledávání cílů z řad personálu letiště. Pro cestující funkce funguje analogicky.

```
//najdi vsechny utocitelne zamestnance
GameObject[] objects = GameObject.FindGameObjectsWithTag("AirportEmployee");
foreach (GameObject employeeObject in objects) {
    if(employeeObject.GetComponent<AirportEmployeeScript>().IsDead())continue;
    RaycastHit hit;

    bool found = false;
    Vector3 direction = employeeObject.transform.position - fromPosition;
    if(Physics.Raycast(currentObject.transform.position, direction, out hit)){
        if (hit.collider.gameObject.tag == "AirportEmployee"){
            Debug.DrawRay(currentObject.transform.position, direction, Color.green, 10.0f);
            possibleTargets.Add(employeeObject);
            found = true;
        }
        else{
            Debug.DrawRay(currentObject.transform.position, direction, Color.yellow, 10.0f);
        }
    }
    if(!found){
        //zkus zamerit hlavu
        Vector3 direction2 = new Vector3(employeeObject.transform.position.x,
                                         employeeObject.transform.position.y + char_height,
                                         employeeObject.transform.position.z)
                                         - fromPosition;
        if(Physics.Raycast(currentObject.transform.position, direction2, out hit)){
            if (hit.collider.gameObject.tag == "AirportEmployee"){
                Debug.DrawRay(currentObject.transform.position, direction2, Color.green, 10.0f);
                possibleTargets.Add(employeeObject);
                found = true;
            }
            else{
                Debug.DrawRay(currentObject.transform.position, direction2, Color.yellow, 10.0f);
            }
        }
    }
}
```

Obrázek 3.3: Detail kódu pro výběr cílů – zaměření cíle

Dalším identifikovaným nedostatkem bylo chybné učení dostupnosti postav, nacházejících se za útočníkem. Z pohledu útočníka by tyto postavy měly být brány jako dostupné cíle, protože nic nebrání útočníkovi se otočit na daném místě v libovolném úhlu. Pokud však zavolám funkci *Raycast* z pistole na objekt, který se nachází za útočníkem, bude první kolizní objekt samotná postava útočníka, a proto, i když může být subjekt na dostřel, tak by nebyl zahrnut do seznamu.

Možností řešení tohoto problému mne napadlo více. Jednou z možností by bylo, aby se útočník několikrát okolo sebe „rozhlédl“ (a postupně se otočil o 360°), v každé výseči dohledal subjekty na dostřel a poté provedl jejich agregaci přes všechny výseče. Toto je však zbytečně komplikované, proto se mi po další analýze podařilo nalézt jednodušší řešení. Jelikož problém spočívá v tom, že první, na co paprsek narazí, je

postava útočníka, zaměřil jsem se na nalezení způsobu, jak dočasně vypnout tento collider.

Ve finálním řešení identifikace cílů je pro daného útočníka nejprve collider deaktivován pomocí příkazu *gameObject.GetComponent<BoxCollider>().enabled = false*. Následně je pomocí funkce *GetReachableTargets* získán seznam potenciálních cílů a poté je collider opět zapnut pomoci nastavení hodnoty příkazu na „true“.

Po provedení všech operací, zmíněných výše, má útočník k dispozici seznam dostupných cílů a přepne se do stavu *WaitingForAttack*.

3.2 Držení cílů „pod kontrolou“

Po úpravě výběru cílů útočníka bylo nezbytné upravit jeho chování vůči jednotlivým cílům tak, aby útočník měl dané cíle vizuálně pod kontrolou. Toto chování by v reálné situaci mělo reflektovat případ, kdy útočník potřebuje kontinuálně ohrožovat cíle, aby nedošlo k případné vzpouře nebo k přivolání policie či jiné pomoci. V rámci původního konceptu simulátoru útočník míří na daný objekt, získaný v rámci předchozího kroku. Následně snižuje *attackCounter* a jakmile zjistí, že je *attackCounter* menší nebo roven nule, pak vystřelí kulku v aktuálně zaměřeném směru a tak zastřelí daný objekt.

Upravený kód zaměřuje jednotlivé potenciální cíle dle jejich pořadí v seznamu. Přepnutí zaměření na dalšího útočníka v pořadí je provedeno každých *x* volání funkce *Update*, kde hodnota *x* je daná konstantou *fps*, definovanou v rámci skriptu útočníka. Momentálně je hodnota *fps* nastavena na 30.

Finální kód je uveden na obrázku 3.4.

```
if(fps_counter ==fps){//vyber target
    if(target_index==target_size)target_index = 0; //reset na zacatek pole

    target = ppl_in_sight[target_index];
    target_index++;

    if (target != null){
        gameObject.transform.LookAt(target.transform.position);
    }
    fps_counter =0;
}
else{
    fps_counter++;
}
```

Obrázek 3.4: Ukázka kódu – držení cílů „pod kontrolou“

3.3 Enemy sanity

Aby nebylo chování nepřátel jednotvárné, je na úrovni **AttackManagera** vytvořena proměnná *enemy_sanity*, která mění chování útočníka. Tato proměnná je aktualizována v závislosti na úspěšnosti obranné jednotky ve zneškodňování hrozeb. Pokud je obrana úspěšná, je proměnná zvýšena o hodnotu nastavenou v *increment_sanity_val*. V případě selhání dojde ke snížení o stejnou hodnotu. Lze rovněž nastavit minimální a maximální rozsah této proměnné, což je řízeno konstantami *enemy_sanity_min* a *enemy_sanity_max*.

Hodnota sanity upravuje scénáře útočníků následujícím způsobem. Momentálně jsou implementovány 2 variace scénářů, které vylepšují původní scénář.

Pokud je *enemy_sanity* větší nebo rovna 75 (dle aktuální konfigurace 5 vítězných her), dochází k úpravě scénáře. Pokud útočník identifikuje, že má na dostřel obrannou jednotku[8], pak místo přepínání pohledů na jednotlivé oběti se útočník zaměří pouze na ní. Tento scénář byl inspirován případnou výměnou rukojmí.

---

[8] Dohledávání obranné jednotky funguje analogicky jako dohledávání potenciálních cílů (viz kap. 3.1), slouží k tomu však jiná implementovaná funkce v rámci **AttackManageru** – *GetUnit*, která pro daného útočníka vrátí *GameObject* obranné jednotky, pokud je na dostřel, anebo hodnotu *null*, pokud není.

Dalším scénářem je „zrada“. Pokud je *enemy_sanity* větší nebo rovna 90, je nezbytné útočníka zastřelit. Namíříme-li v tomto případě na útočníka zbraň s cílem, aby se vzdal, vyprovokuje ho to k agresivitě a před vzdáním se ještě vystřelí.

3.4 Chování cestujících a zaměstnanců

V původním konceptu simulátoru je nastaveno, že jakmile začne útok, tak se všichni cestující a personál letiště vzdají (tj. zastaví své chování a provedou animaci vzdání se). Tato akce byla pevně naprogramována jako součást procedury spuštění útoku (funkce *Attacking* v rámci **AttackManageru**). Jelikož můžeme chápat letiště jako větší celek, nedává smysl, aby se například obsluha v hale na vydávání zavazadel vzdala, pokud probíhá útok ve vedlejší budově.

Po provedených úpravách se nyní vzdají pouze ti cestující nebo personál, kteří jsou na dostřel útočníka. Tato úprava byla provedena již v rámci samostatného projektu. Původní funkce *Attacking* byla nahrazena novou funkcí *GoAttacking*, která na rozdíl od původní funkce obdrží seznam potenciálních cílů, získaný pomocí funkcionality, popsané v kapitole 3.1. Původní funkce byla však v kódu ponechána pro případné budoucí využití.

V rámci projektu však nebylo dořešeno chování ostatních postav mimo dostřel, které se pro jednoduchost zastavily až do ukončení útoku (zneškodnění hrozby). V diplomové práci je tato nedokonalost odstraněna a chování postav bylo dále upraveno. Bližší detaily k této nové funkcionalitě jsou uvedeny v kapitole 3.9.

3.5 Správce statistických dat

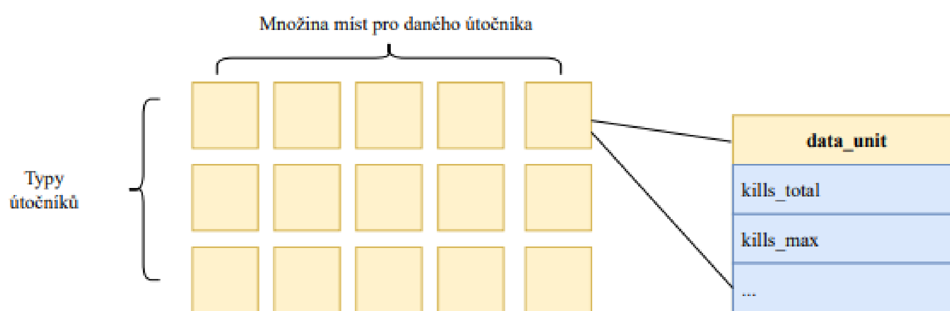
Správce statistických dat je nejdůležitější částí simulátoru ve věci rozhodování a učení se. Slouží k ukládání statistických dat, která dále vstupují do rozhodování. V rámci simulátoru je pojmenován jako **IntelManager** a jeho konfigurace je uložená pod objektem **Managers**.

Veškerá data jsou uzavřena ve třídě *data_unit*. **IntelManager** obsahuje dvourozměrné pole těchto tříd. Cílem bylo mít pro každou dvojici typu útočníka a možného místa útoku jeden prvek *data_unit*, obsahující statistická data pro tuto kombinaci.

Konkrétně jsou ukládány následující údaje (pro každé místo a typ útočníka). V závorce za údajem je uveden interní název v rámci kódu.

- počet zabití celkem (*kills_total*)
- maximální počet zabití jedním útočníkem (*kills_max*)
- počet pokusů o útok (*attack_tries*)
- počet pokusů o útěk (*escape_tries*)
- počet úspěšných útěků (*escape_success*)
- počet úmrtí útočníka v době útoku (*attack_dies*)
- počet „vzdání se" útočníka v době útoku (*attack_submits*)
- počet úmrtí útočníka při útěku (*escape_dies*)
- počet „vzdání se" útočníka při útěku (*escape_submits*)

Pro lepší pochopení struktury paměti je na obrázku níže zobrazen detail v grafické podobě.



Obrázek 3.5: Struktura paměti simulátoru (vlastní zpracování)

Vkládání dat do jednotlivých *data_unit* probíhá prostřednictvím níže definovaných funkcí:

log_attack_attempt (*attacker_type_id, place_id*)

Funkce ověří, zda vstupní argumenty *attacker_type_id* a *place_id* odpovídají daným intervalům, a poté inkrementuje hodnotu *attack_tries* v příslušné *data_unit*.

log_kill (*attacker_type_id, place_id*)

Funkce ověří, zda vstupní argumenty *attacker_type_id* a *place_id* odpovídají daným intervalům, a poté inkrementuje hodnotu *kills_total* v příslušné *data_unit*.

log_escape_attempt *(attacker_type_id, place_id)*

Funkce ověří, zda vstupní argumenty *attacker_type_id* a *place_id* odpovídají daným intervalům, a poté inkrementuje hodnotu *escape_tries* v příslušné *data_unit*.

log_ending_type*(attacker_type_id, place_id, ending_type, ending_state_type, kill_iteration)*

Funkce ověří vstupní argumenty *attacker_type_id*, *place_id* a *ending_type* a poté inkrementuje příslušnou hodnotu v příslušné *data_unit*. *Ending_type* = 0 slouží k zalogování konce útočníka ve stádiu útoku. *Ending_type* = 1 slouží k zalogování konce útočníka při útěku. Pokud se útočník vzdal, voláme funkci s *ending_state_type* = 0, při zabití útočníka s *ending_state_type* = 1. Zároveň na základě hodnoty *kill_iteration*, udávající počet zabití útočníka, aktualizuje hodnoty *kills_max* a zároveň aktualizuje maximální počet zabití přes všechna místa (interní proměnná **IntelManageru** – *max_kill_max*).

log_escape_success*(attacker_type_id, place_id, kill_iteration)*

Funkce se volá v případě úspěšného útěku útočníka. Stejně jako funkce *log_ending_type* rovněž aktualizuje hodnotu „*kills_max*“ a příp. i maximální počet zabití přes všechna místa.

3.6 Nahrání, ukládání a resetování paměti

Součástí implementace paměti, definované v kapitole 3.5, bylo naprogramováno případné uložení paměti pro zachování informací i po ukončení nebo restartování

simulátoru. Paměť je uložena do souboru definovaného relativní cestou, která je zapsaná v proměnné *file_path*. Součástí definice této cesty je i název souboru. Hodnotu této proměnné lze upravit pomocí **Unity Inspectoru**. V aktuálním nastavení je paměť ukládána do kořenového adresáře projektu v souboru *memory.txt*.

Samotné uložení paměti se provádí ručně prostřednictvím tlačítka „Save Memory“ v Game-Over místnosti. Pokud bychom chtěli vrátit paměť do výchozího stavu, lze tak učinit tamtéž pomocí tlačítka „Reset memory“, případně stačí smazat paměťový soubor.

Nahrávání paměti je provedeno automaticky v rámci *Awake* funkce **IntelManageru**. Tím je zajištěno, že paměť je načtena při každém spuštění simulace. Zároveň funkce kontroluje mezní situaci, pokud by soubor neexistoval. V tomto případě se nestane nic a paměť bude inicializována prázdná.

3.7 Výběr optimálního místa

Poté, co máme k dispozici údaje v paměti, bylo možné využít tyto údaje k úpravě chování útočníků a tím docílit věrohodnější simulace. Jako jednou z dalších úprav bylo implementováno rozhodování o volbě místa útoku na základě dat z paměti. Nejprve však bylo potřeba se zamyslet, na základě jakých kritérií se bude útočník rozhodovat.

Po delší úvaze jsem vyhodnotil dvě kritéria úspěchu útočníka. Cílem útočníka (se zbraní) je jednak <u>zlikvidovat co nejvíce cestujících</u> a zároveň <u>úspěšně uniknout chycení ostrahou letiště</u>. Z údajů, dostupných v paměti pro konkrétní místo útoku, se tato kritéria dají vystihnout následujícím vzorcem:

$$k * \frac{počet\ úspěšných\ útěků\ z\ místa}{počet\ útoků\ na\ dané\ místo} + l * \frac{maximální\ počet\ zabitých\ na\ tomto\ místě}{maximální\ počet\ zabitých\ přes\ všechna\ místa}$$

kde koeficienty k a l umožňují nastavení preferencí toho či onoho kritéria.

Je třeba poznamenat, že proměnné ve jmenovateli mohou nabývat hodnoty 0, pro které není standardní dělení definováno. V těchto případech je zlomek zvlášť ohodnocen na 0.

Je třeba rovněž vzít v potaz, pokud by útočník alespoň jednou zabil na daném místě nebo z místa alespoň jednou úspěšně utekl, pak bude hodnota prvního či druhého zlomku nenulová. To by mělo za důsledek preferenci daného místa před místy, ze kterých se nikdy neútočilo, nehledě na fakt, že dané místo má nízkou úspěšnost úniku nebo nízký počet zabitých. Proto je v kódu implementována doplňující podmínka:

Vyjde-li hodnota prvního zlomku menší nebo rovna hodnotě *koeff_reduction_escape*, pak bude hodnota celého místa 0. Stejně tak pro druhý zlomek, kde je podmínka řízena proměnnou *koeff_reduction_kill*. Hodnoty těchto dvou řídících proměnných jsou definovány v **IntelManageru** a momentálně jsou obě nastaveny na 0.2. Změnit je lze v kódu nebo prostřednictvím **Unity Inspectoru** (na objektu **Managers** – sekce **IntelManager**).

3.8 Escape procedura

V původním konceptu simulátoru byla implementována logika, že simulace je ukončena vždy při zabití některého z cestujících. Toto nastavení nepovažuji za příliš vhodné při výcviku uživatelů, protože je de facto tolerována pouze jedna chyba. Zároveň se tím znemožňuje efektivní sběr informací do paměťové komponenty simulátoru.

Nově byl simulátor upraven tak, že simulace končí pouze v případě úmrtí obranné jednotky nebo zabitím cestujícího obrannou jednotkou. Tato úprava otevřela nové možnosti chování útočníků, které by dle definice původního konceptu nebyly možné. Při pouhém odstranění volání funkce *game_over* by totiž útočník stále útočil na náhodné cíle, dokud by nebyl zlikvidován. Proto byla vytvořena nová funkcionalita, která byla pracovně označena jako „escape procedura“. Ta spočívá v tom, že při každém

zabití a v některých dalších případech (viz dále) je volána funkce **AttackManagera** *decide_to_escape*, která volá funkci *decide_memory_escape* ve třídě **IntelManager**.

Tato funkce na základě informací z paměti a údajů, poskytnutých útočníkem, rozhodne, zda bude procedura aktivována či nikoliv.

Podmínky jsou nastavené následovně a jsou vyhodnocovány v pořadí uvedeném níže:

a) Pokud je počet útoků z daného místa roven nule nebo maximální počet zabitých na daném místě roven nule, pak je útěk vždy zamítnut. Takové nastavení zajišťuje, že minimálně při prvním pokusu o útok (tj. při prázdné paměti) nebude escape procedura útočníkem provedena. Díky tomu bude útočník motivován zabít co nejvíce osob a simulátor získá vhodnou hodnotu maximálního počtu zabití na konkrétním místě. Tato hodnota je dále využita v logice chování útočníka.

b) Pokud je pravděpodobnost útěku, vyjádřená podílem počtu úspěšných útěků vůči celkovému počtu útoků na dané místo, vetší nebo rovna hodnotě proměnné *koeff_escape_decision*, pak je útěk zamítnut. Tato podmínka reflektuje následující scénář: má-li útočník zkušenosti, že z daného místa s vysokou pravděpodobností úspěšně uteče, pak se mu nevyplatí utíkat a v koncovém důsledku se bude snažit o maximalizaci zabitých osob.

c) Pokud nebude výsledek funkce rozhodnut podmínkami, uvedenými výše, pak je rozhodnutí o útěku útočníka definováno následující nerovnicí:

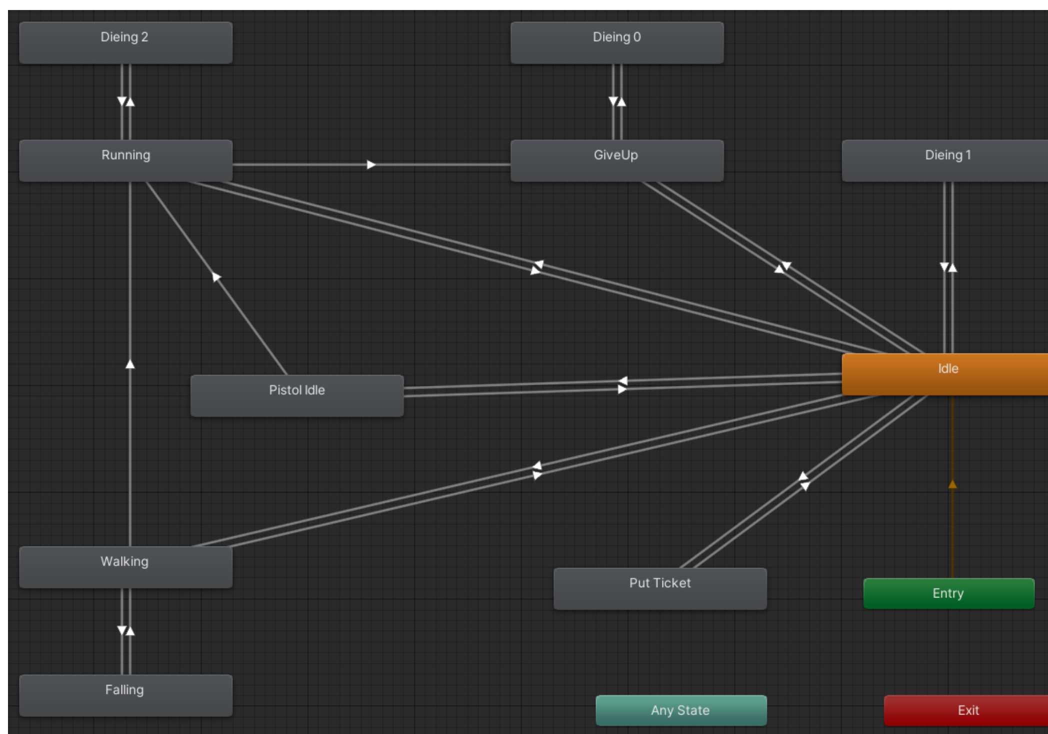
$$\frac{aktuální\ počet\ zabitých\ útočníkem}{max.počet\ zabitých\ na\ tomto\ místě} > \frac{počet\ úspěšných\ útěků\ z\ místa}{počet\ útoků\ na\ místo}$$

Je-li tato nerovnice splněna, pak je podán pokyn útočníkovi k útěku. V ostatních případech je útěk zamítnut.

Princip útěku útočníka je následující. Na letišti nově bylo definováno místo *EscapePoint*, což je cílové místo útěku útočníka. Pro útočníky s pistolí (útočník z galerie a útočník z okna) je toto místo identifikováno pomocí tagu *EscapePoint*. Pro útočníka na ranvej je cílovým místem útěku výchozí „spawnpoint". Z důvodu, že ranvejový útočník využívá trhliny v plotě pro vniknutí na letiště, jedná se z jeho pohledu o jedinou únikovou cestu, která je mu známa (názorně na obrázku A.1 v příloze A).

Z vizuálního hlediska byla importována nová animace pro jednotlivé útočníky zobrazující „rychlý běh". Podobně jako Ing. Bursík jsem využil knihovnu *Mixamo* [23], nabízející již hotové animace, které se dají přizpůsobit pro konkrétní účel.

Animační schéma po provedených úpravách vypadá dle obrázku níže.



Obrázek 3.6: Schéma přechodu animací pro útočníky a cestující

Všimněme si, že do nového stavu *Running* lze přejít ze stavu *Walking*, *Pistol Idle* nebo *Idle*. Tento přechod je řízen stejnojmennou animační proměnnou *Running*.

Mimo rozdílné animace je při spuštění *escape procedury* také zvýšena hodnota *agent.speed*, proměnné omezující maximální rychlost pohybu útočníka. Útočník se tak pohybuje rychleji, než při běžné chůzi.

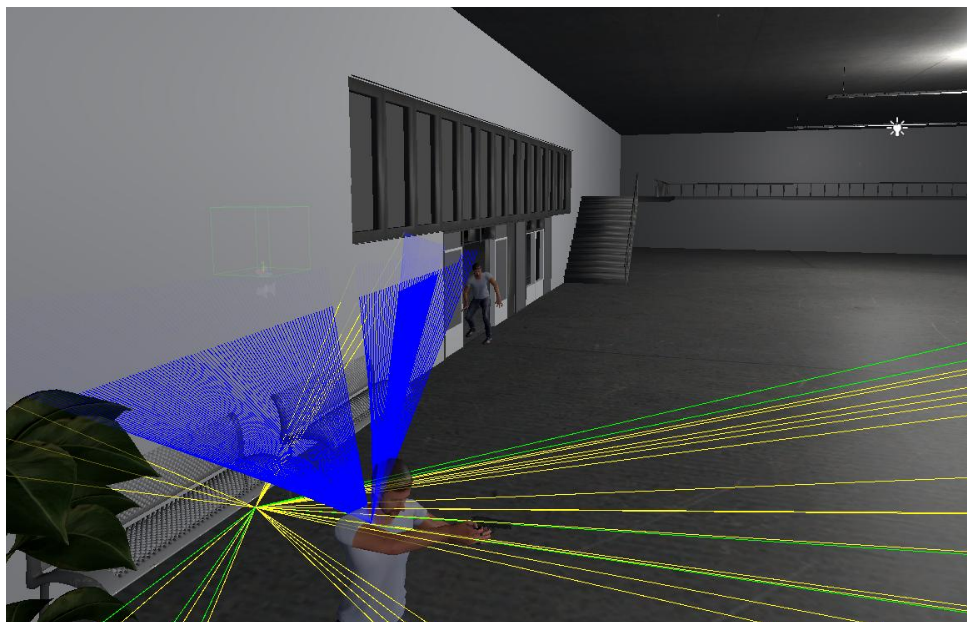
Podaří-li se útočníkovi dostat na místo útěku dříve, než bude zastřelen nebo zastaven obrannou jednotkou, pak útok končí ve prospěch útočníka.

3.9 Identifikace útočníka cestujícími

Jak již bylo zmíněno v předchozích kapitolách, velkou výhodu přináší to, že v upraveném konceptu je simulátor uchopen jako jeden celek. Díky tomu mohou cestující komunikovat s **AttackManagerem** a na základě toho simulovat reakci, např. zahlédnutí útočníka. V původní implementaci se všichni cestující a personál letiště vzdali, jakmile začal útok, nehledě na svou pozici vůči útočníkovi. V důsledku provedených úprav bylo chování cestujících a personálu změněno tak, že se vzdají pouze ti cestující a personál, kteří jsou v přímém dosahu (dostřelu) útočníka. Funkcionalita pro identifikací cílů je však volána pouze před začátkem samotného útoku nebo po zastřelení nějaké oběti (v rámci aktualizace seznamu cílů) a tudíž nepokrývá cestující, přicházející do kontaktu s útočníkem v mezičase.

Tento problém byl vyřešen následujícím způsobem. V rámci **AttackManageru** byla vytvořena nová funkce *check_attacker_nearby*, která na vstupu obdrží *GameObject* postavy, volající tuto funkci (tj. cestující) a dotazuje se, zda má na dohled aktivního útočníka. Pokud ano, je vrácen *GameObject* příslušného útočníka nebo hodnota *null*, není-li cestující ohrožen. Z technického hlediska byl v rámci třídy **AttackManager** implementován seznam aktuálních útočníků, do kterého je útočník přidán v případě spuštění útoku a odebrán při ukončení útoku. Pro všechny útočníky na seznamu je vyhodnocováno, zda je přímá viditelnost mezi danou postavou a daným útočníkem a zároveň je vykreslen v rámci aplikace Unity modrý paprsek, indikující spojnici těchto postav. V původním konceptu simulátoru může na letiště útočit maximálně jeden útočník. Tuto podmínku jsem neměnil, ale pokud by v budoucnu byla zrušena, pak by to na funkci *check_attacker_nearby* nemělo mít dopad díky využití seznamu.

Pro účely testování je tato funkčnost reprezentována modrými paprsky v aplikaci Unity (viz obrázek níže).



Obrázek 3.7: Modré paprsky - vizualizace identifikace útočníka cestujícími (Unity)

3.10 Trigger konce hry

Dle původního konceptu je zastřelení postavy důsledkem konce simulace, což je nastaveno ve skriptu **HumanDeparture** ve funkci *OnTriggerEnter*, zajišťující chování v případě kolize s objekty. Z důvodů, uvedených v kapitole 2.3, byla tato funkcionalita zrušena.

Konec simulace je nyní iniciován v následujících případech:

- V případě úmrtí obranné jednotky.
- V případě zabití nevinného pracovníka či cestujícího obrannou jednotkou.
- Pomocí tlačítka v menu – Restart nebo Quit.

Při těchto úpravách se mi podařilo identifikovat neočekávané chování, kdy po konci hry (tj. po přesunu uživatele do *GameOver* místnosti) nedojde k zastavení hry jako takové. Tudíž pokud útočník nebyl zneškodněn, pokračuje i nadále v útoku a střílí dále

na cíle. Toto chování je nežádoucí, protože může mít vliv na hodnoty *enemy_sanity* a rovněž na informace v paměti.

Tento problém byl eliminován implementací nové funkce **AttackManageru** *stop_the_game*, která iniciuje přechod stavového automatu všech postav letiště do nového stavu *Stop*. Díky tomu dojde k zastavení provozu na letišti v případě konce hry.

3.11 Oživení pracovníků letiště

V rámci provedených testů (viz kap. 4) byl identifikován nový problém, způsobený provedenými úpravami simulátoru. Tím, že útočníci útočí nově i na personál letiště (viz kap. 3.1) a zabití útočníkem neukončuje simulaci (viz kap. 3.10), bylo nutné dodatečně definovat chování pracovníků letiště v případě jejich úmrtí.

Možných variant řešení tohoto problému bylo několik. Teoreticky by stačilo implementovat zmizení (despawn) postavy ve stavu *Dead* po uplynutí daného času. Toto chování by ale mělo dopad na realističnost simulace, protože by cestující byli odbavováni u prázdné přepážky.

Implementovaným řešením, které sice není tolik realistické, ale řeší potenciální problém, uvedený výše, je doplnění přechodového stavu z animace *Dead* do stavu *Idle*. Toto „oživení“ pracovníků letiště je provedeno jako součást procedury ukončení útoku.

3.12 Výpočet herního skóre

Dalším důsledkem úpravy kritérií pro konec simulace je chybějící zpětná vazba pro účastníka simulace, dávající přehled úspěšnosti při zneškodňování hrozeb. Využitím dat z paměti je vypočteno a zobrazeno jednak skóre daného účastníka simulace a zároveň agregované statistiky útoků.

Skóre je definováno následujícím vzorcem

$$Skóre = k_a * (k_s * U_{vzd} + k_d * U_{die}) + k_e * \left(k_s * U_{e_vzd} + k_d * U_{e_die}\right)$$

$$- (k_{kill} * U_{kill} + k_{esc} * U_{esc})$$

Na první pohled se může výpočet zdát nepřehledným, ale není tomu tak.

Hodnoty k_x jsou jednotlivé koeficienty pro úpravu preferencí té či oné kvality. Nastavení těchto koeficientů je opět možné přes **Unity Inspector**. V rámci iniciálního nastavení je preferováno zneškodnění útočníka při útoku vzdáním se.

Hodnoty U_x a odpovídají sbíraným údajů v paměti:

- U_{vzd} odpovídá počtu vzdání se útočníka v průběhu útoku
- U_{die} odpovídá počtu zabití útočníka v průběhu <u>útoku</u>.
- Analogicky podobné proměnné s prefixem „$_{e_}$“ indikují počty vzdání se / zabití v průběhu <u>útěku</u>.
- U_{kill} odpovídá celkovému počtu zabitých cílů
- U_{esc} je celkový počet úspěšných útoků.

Jak bylo zmíněno výše, součástí skóre na Game-Over obrazovce jsou základní statistiky, agregované přes jednotlivé scénáře útoku. Mezi zobrazované hodnoty patří:

- Celkový počet útoků
- Celkový počet útěků
- Celkový počet zabitých cílů
- Celkový počet úmrtí daného útočníka
- Celkový počet vzdání-se útočníka.

3.13 Chování útočníka na ranveji

Vzhledem k tomu, že útočník ze scénáře 3 není ozbrojen, bylo potřeba stanovit kvalitativní cíle pro zhodnocení útoku. Pro tohoto útočníka se kvalita útoku odvíjí od počtu „navštívených míst“. Pokud útočník dané místo navštíví, dá se předpokládat, že toto místo mohlo být sabotováno, ohroženo bombou nebo jiným rizikem.

Útok pro tento typ útočníka začíná vniknutím na letiště, resp. návštěvou kontrolního bodu, nacházejícího se poblíž děr v plotě, kterými se útočník na letiště dostává. V prostoru letiště, kde se útočník z původního konceptu pohyboval, bylo

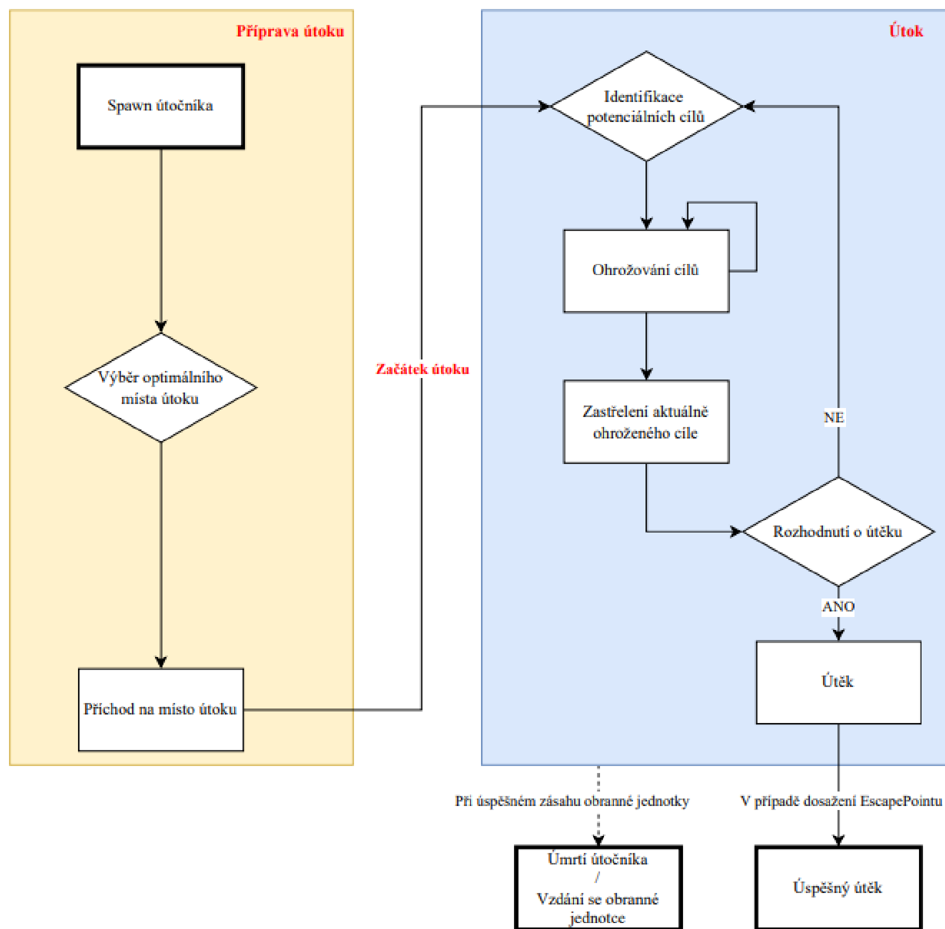
vybráno několik zajímavých objektů (řídící věže a hangáry), které by mohly být vhodným cílem útoku. Útočník tato místa navštěvuje v náhodném pořadí a návštěvu zaznamenává jako zabití cestujícího. Díky tomu může bez dalších úprav použít funkci *decide_to_escape*, na základě které se útočník rozhoduje, zda uteče či nikoliv.

Místem útěku pro tohoto útočníka je jeho spawnpoint, protože útočník při vniknutí na letiště zná pouze tuto cestu do prostoru letiště. Zároveň byla upravena rychlost útočníka. Jelikož se pohybuje v prostoru letiště, který je veřejnosti nepřístupný, musí se pohybovat rychleji, aby nebyl spatřen a chycen.

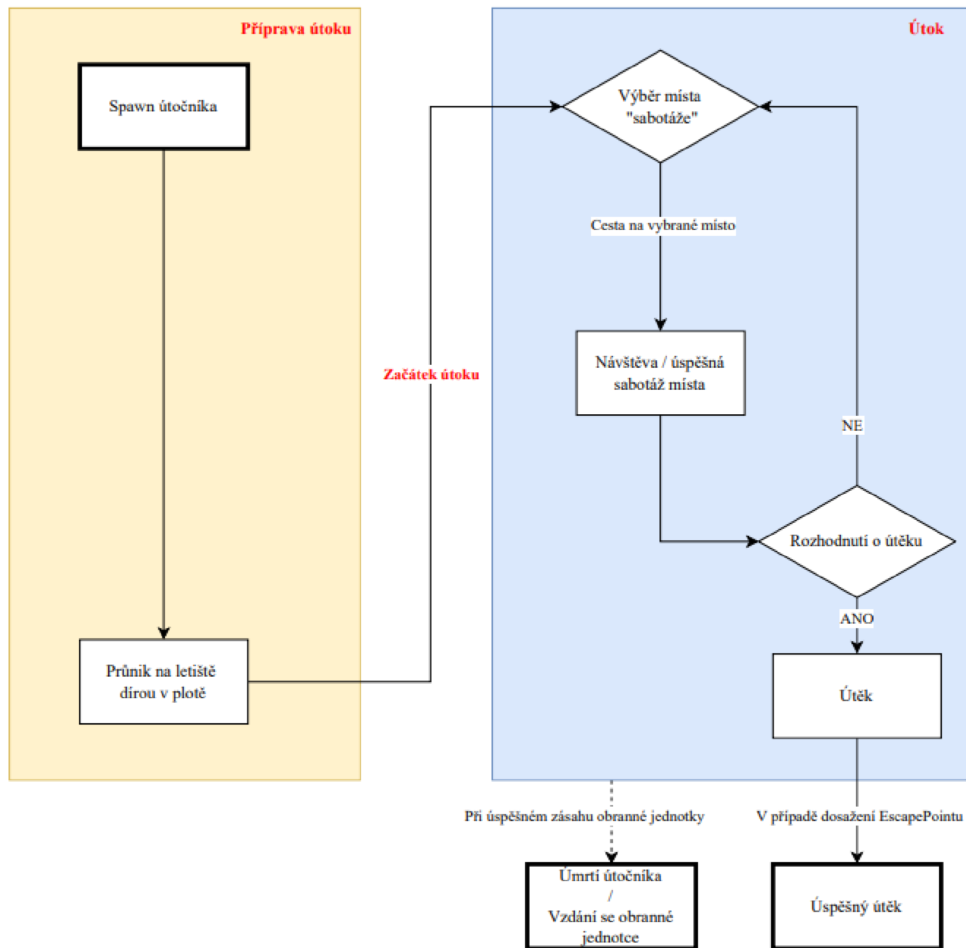
3.14 Shrnutí

V této kapitole byly uvedeny nově implementované funkcionality, které by měly přispět k větší realističnosti simulace a zároveň implementovat využití umělé inteligence a paměti simulátoru. Jako základ pro pochopení problematiky vývoje aplikací pro virtuální realitu jsem využil informací z kapitoly 3 knihy *Learning Virtual Reality* od Tonyho Parisiho [24]. V rámci implementace jsem primárně využil materiály z manuálu Unity [25]. Dále při úpravách, souvisejících s animacemi, mi byla nápomocna kniha *Unity UI Cookbook* [26]. Kniha *C# 5.0 Programmer's Reference* [27] byla použita jako reference při práci se souborovým systémem.

Po provedených úpravách je schéma útoku vyobrazeno na obrázcích 3.8 a 3.9.



Obrázek 3.8: Zjednodušené schéma chování útočníků se zbraní (tj. scénář 1 a 2)



Obrázek 3.9: Zjednodušené schéma chování útočníka na ranvej (tj. scénář 3)

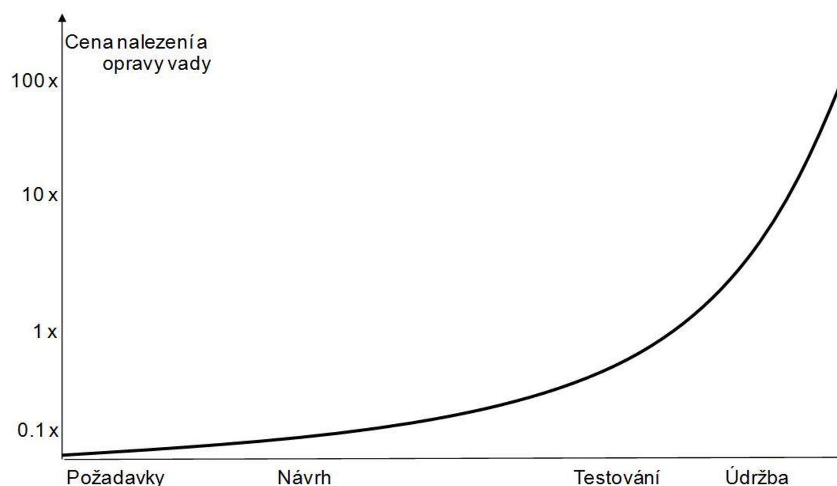
Statistická data, sbíraná pomocí komponenty **IntelManager**, jsou použita pro *výběr optimálního místa útoku* ve scénáři 1 a 2 (obr 3.8)[9]. Data jsou rovněž využita při *rozhodnutí o útěku* jak ve scénářích 1 a 2 (obr 3.8), tak ve scénáři 3 (obr 3.9).

[9] Scénáře 1 a 2 byly zjednodušeny do jednoho schématu z důvodu, že signifikantním rozdílem těchto scénářů je primárně „entrée“ na místo útoku. Velkou část procesu však mají shodnou.

4 Testování

Řádné otestování aplikace je nezbytnou součástí vývoje. Na toto téma je napsáno mnoho odborných publikací a článků, zabývajících se vhodnými postupy a typy testování [28].

Motivací pro důsledné testování může být několik, například cena. Obrázek níže zobrazuje násobek ceny za nalezení a opravy vady v různých fázích vývoje a po dokončení vývoje.



Obrázek 4.1: Graf nárůstu ceny opravy vady při a po vývoji [29]

Testování bylo prováděno primárně prostřednictvím manuálních end-to-end testů (E2E). Po provedení jednotlivých úprav (viz kapitola 3) byl simulátor vždy spuštěn a bylo sledováno chování simulace a obsah logů.

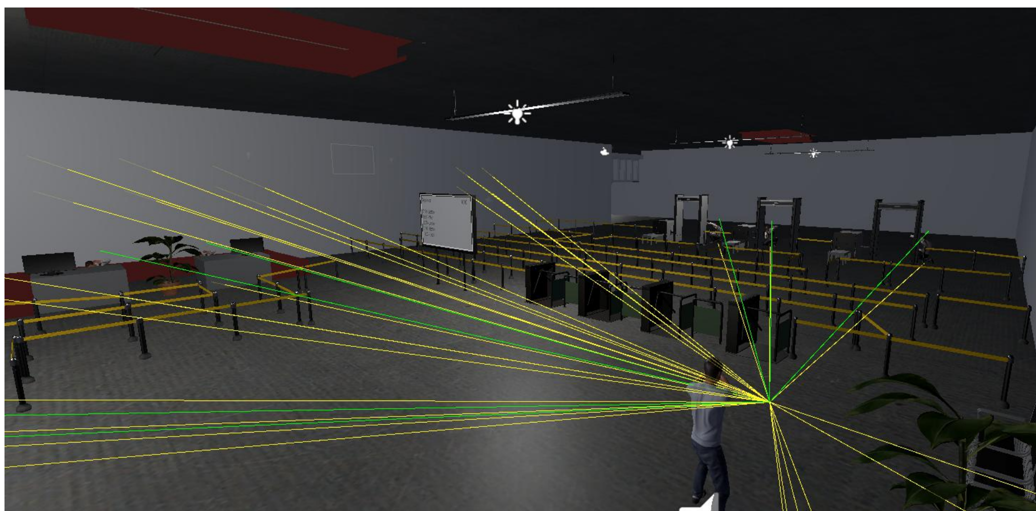
Pro účely testování byl použit vlastní VR headset „*Oculus Rift S*“ [4]. Vzhledem k tomu, že je simulátor postaven na platformě **SteamVR** [5] od Valve Corporation, zde by neměl být problém s portabilitou mezi zařízeními podporující **SteamVR**. Jediným potenciálním rizikem je konfigurace ovladačů tak, aby byla jednotlivá tlačítka a joysticky správně „namapovaná“. V tomto případě však nebylo potřeba nic upravovat, defaultní nastavení fungovalo bez větších potíží. Byl identifikován pouze drobný technický problém: pokud není současně s aplikací **SteamVR** spuštěn i klient aplikace

Steam, pak se v simulátoru nezobrazí ruce a nelze simulátor používat. Pro správné chování simulátoru je tedy nezbytné mít nainstalovaného a spuštěného klienta aplikace **Steam**. Aplikace **SteamVR** je spuštěna automaticky při spuštění simulace.

4.1 Testování identifikace cílů

Pro účely testování je v rámci funkce *GetReachableTargets* volána funkcionalita Unity *Debug.Drawray*, která umožňuje vykreslit paprsek, podobný paprsku pro hledání kolizí. Pokud je paprsek zelený, znamená to, že cíl byl označen jako dostupný, žlutý paprsek znamená opak. Tímto lze zkontrolovat, zda funkcionalita funguje dle očekávání. Díky těmto paprskům bylo možné včas identifikovat chyby při dohledávání cílů (viz kapitola 3.1).

Na obrázku níže je vidět vizualizaci této funkce.



Obrázek 4.2: Ukázka paprsků *Debug.Drawray* (Unity)

Je důležité poznamenat, že pokud chceme v průběhu testování simulace v aplikaci Unity tyto paprsky vidět, je nezbytné mít aktivní *gizmos*, což provedeme pomocí tlačítka v aplikaci Unity (v pravé části okna se simulací).

Na obrázku níže je tlačítko zvýrazněno bílou šipkou.

Obrázek 4.3: Zobrazení *gizmos* (Unity)

Při běhu simulace jsou do logu vypisovány některé klíčové informace (např. počet obětí na dostřel, zda byla obranná jednotka zachycena útočníkem atp.), což při testování pomáhá určit, v jakém stavu se simulace nachází a tím pádem ověřit, že se chová dle očekávání.

4.2 Testování escape procedury

V průběhu testování funkcionality *escape procedury* byla identifikována zajímavá chyba. Původní *EscapePoint* pro útočníky na letištní halu byl umístěn za halou na vydávání zavazadel. V tomto případě však docházelo ke kuriózní situaci, kdy útočník po spuštění *escape procedury* seběhl dolů z galerie, poté vyrazil čelem ke stěně směrem k únikovému bodu a proběhl stěnou. Původní předpoklad důvodu této chyby spočíval v tom, že tag pro *EscapePoint* byl špatně nastaven, resp. byl použit jinde a proto cestující utíkal jakoby do zdi. Po zevrubné analýze se ukázalo, že důvod této chyby je mnohem komplexnější.

Model letiště má ve zdi mezi odletovou a příletovou halou určitou nedokonalost, spočívající v tom, že část stěny nepřiléhá ke druhé a mezi nimi se nachází malá mezera, odhalitelná pouze ze správného úhlu pohledu. Z tohoto důvodu, podle mého názoru, vestavěný Unity algoritmus, počítající optimální cestu k danému bodu, určí cestu špatně.

Pokusy o eliminaci této chyby byly následující.

Nejprve jsem se snažil upravit model v aplikacích pro opravu *meshe*. Při importování modelu do aplikace **Netfabb** [30] jsem však zjistil, že takových nedokonalostí je na modelu více a nebylo by v mých silách je opravit, jelikož nemám tolik zkušeností s editacemi modelu.

Druhým pokusem bylo vložit na místo další stěnu, která by problémový otvor zakryla. Zde jsem však bohužel asi nebyl úspěšný v konfiguraci, jelikož Unity algoritmus stále projektoval optimální cestu skrze danou zeď i přesto, že se útočník o novou zeď již zastavil.

Nakonec byl problém vyřešen nejjednodušší metodou (tzv. workaroundem) a to přesunem únikového místa na opačnou stranu budovy od vchodu. Tímto se podařilo tuto chybu, i když dočasně, eliminovat. Bylo by však vhodné, pokud bude projekt simulátoru letiště dále vyvíjen, provést opravu modelu.

4.3 Další testy a opravy chyb

Při příležitosti testování aplikace se povedlo opravit několik varování, které program generoval v rámci původního konceptu.

Jednalo se o varování na neexistující skripty u kamer a jezdících schodů. Chyba byla opravena odebráním skriptu.

Dále zde bylo varování na více aktivních **AudioListeners** ve scéně, způsobena tím, že při tvorbě kamery je komponenta **AudioListener** defaultně zapnutá. Jejich vypnutí varování eliminovalo.

Další varování upozorňovalo na zastaralost funkce *Application.LoadLevel*, volané při restartu simulátoru. Dle doporučení ve varování byla funkce nahrazena ekvivalentním voláním funkce *SceneManager.LoadScene*. Následným testem bylo ověřeno, že tato úprava nemá dopad na fungování aplikace.

V rámci původního prototypu simulátoru byl vytvořen kamerový velín, který zároveň signalizuje případné narušení prostoru dráhy útočníkem. Do této místnosti ve hře nedá fyzicky dostat, což je pochopitelné, protože má sloužit operátorovi simulátoru k navigaci uživatele, nacházejícího se ve VR. Jelikož je v aplikaci zajištěno, že je

spawnován maximálně jeden útočník, jedná-li se o útočníka útočícího na ranvej, pak nedojde k útoku na halu a nelze testovat chování útočníků v hale.

Za účelem pohodlnějšího testování i pro případné budoucí využití jsem do scény přidal další kameru, která zabírá kamerový systém velínu a výstup z této kamery jsem vyvedl na *Display2* (Unity nabízí celkem 8 obrazovek, na které lze souběžně vysílat). Vhodným nastavením *Target Eye* v detailech kamery lze provozovat aplikaci ve VR a zároveň přijímat „stream“ z kamerové místnosti pro operátora (uživatele u počítače).

4.4 Testování po dokončení vývoje

Poté, co byly do aplikace implementovány veškeré funkcionality, zmíněné v předchozích kapitolách, bylo nezbytné otestovat funkčnost simulátoru jako celku. Toto bylo provedeno ve dvou kolech testů.

4.4.1 Testování bez zásahu uživatele

V prvním kole byla testována aplikace samotná, bez zásahu obranné jednotky (uživatele) simulátoru. Protože se útočníci vytvářejí náhodně, byl simulátor spouštěn / restartován tolikrát, než byly otestovány všechny scénáře níže.

➢ **Test útěku – útočník z galerie**

Pro tento test funkce decide_to_escape v attack manageru vrací vždy true.

- Útočník přijde na náhodně vybrané místo z množiny míst (paměť je prázdná).
- Vybere všechny cíle na dostřel.
- Postupně střídá zaměření.
- Až dojde čas, provede zastřelení aktuálně zaměřeného cíle.
- Vyhodnotí, zda má utéct (ano).
- Bude utíkat na místo *EscapePoint* (a přitom bude přehrána animace běhu).
- Na místě útěku útočník zmizí a útok je ukončen (a v logu uveden jako úspěšný útěk).

- ➢ **Test běžného procesu bez zásahu – útočník z galerie:**

- Útočník přijde na náhodně vybrané místo (*paměť je prázdná*).
- Vybere všechny cíle na dostřel.
- Až dojde čas, provede zastřelení aktuálně zaměřeného cíle.
- (Opakuje vybrání cílů + zastřelení, dokud žádný cíl nezbude).
- Vyhodnotí, zda má utéct (*ano z důvodu, že nezbyly žádné cíle*).
- Bude utíkat na místo *EscapePoint* (a přitom bude přehrána animace běhu).
- Na místě útěku zmizí a útok je ukončen (a zalogován jako úspěšný útěk).

Identické testy byly provedeny pro útočníka z okna.

- ➢ **Test útěku – útočník na ranvej**

Pro tento test funkce decide_to_escape v attack manageru vrací vždy true.

- Útočník přijde na náhodně vybrané místo (*z množiny míst*).
- Zaznamená návštěvu místa (tj. úspěšnou sabotáž).
- Vyhodnotí, zda má utéct (*ano*).
- Bude utíkat na místo svého spawnpointu (útočník „zná“ místo, odkud na letiště vnikl) – animace utíkání.
- Na místě spawnpointu zmizí a útok je ukončen (a v logu uveden jako úspěšný útěk).

- ➢ **Test běžného procesu bez zásahu – útočník na ranvej:**

- Útočník přijde na náhodně vybrané místo (z množiny míst).
- Zaznamená návštěvu místa (tj. úspěšnou sabotáž).
- Navštěvuje další místa ze seznamu, dokud nenavštíví všechny.
- Vyhodnotí, zda má utéct (ano, sabotoval všechna místa).

- Bude utíkat na místo svého spawnpointu (útočník „zná“ místo, odkud na letiště vnikl) – animace utíkání.
- Na místě spawnpointu zmizí a útok je ukončen (a zapsán jako úspěšný útěk).

4.4.2 Testování se zásahem uživatele (obranné jednotky)

Druhé kolo testů mělo prověřit fungování a interakci zásahové jednotky s útočníky a prověřit funkčnost upravených tlačítek a procesů.

Následující testy byly provedeny pro všechny scénáře útoku, tj. pro všechny tři typy útočníků (s výjimkou *testu znovuoživení personálu letiště*).

➢ **Test zadržení útočníka vzdáním se v průběhu útoku**

- Útočník provede standardní proces až po ohrožování cestujících zbraní.
- Na útočníka je z dostatečné vzdálenosti namířena zbraň.
- Útočník se vzdá (animace *GiveUp*) a zmizí.
- Útok je ukončen.

➢ **Test zadržení útočníka zastřelením v průběhu útoku**

- Útočník provede standardní proces až po ohrožování cestujících zbraní.
- Na útočníka je proveden výstřel.
- Útočník umírá (animace *Dieing*) a zmizí.
- Útok je ukončen.

➢ **Test zadržení útočníka vzdáním se v průběhu útěku**

Pro tento test funkce decide_to_escape v attack manageru vrací vždy true.

- Útočník provede standardní proces až po spuštění útoku.
- Po uplynutí času útoku zastřelí vybraný cíl (příp. dle scénáře 3 navštíví vybrané místo).
- Začne utíkat (animace *Running*).

- Na útočníka je z dostatečné vzdálenosti namířena zbraň.
- Útočník se vzdá (animace *GiveUp*) a zmizí.
- Útok je ukončen.

➢ **Test zadržení útočníka zastřelením v průběhu útěku**

Pro tento test funkce decide_to_escape v attack manageru vrací vždy true.

- Útočník provede standardní proces až po spuštění útoku.
- Po uplynutí času útoku zastřelí vybraný cíl (příp. dle scénáře 3 navštíví vybrané místo).
- Začne utíkat (animace *Running*).
- Na útočníka je z dostatečné vzdálenosti namířena zbraň.
- Útočník se vzdá (animace *GiveUp*) a zmizí.
- Útok je ukončen.

➢ **Test znovuoživení personálu letiště**

- Útočník provede standardní proces až po spuštění útoku.
- Po uplynutí času útoku zastřelí vybraný cíl (čekání, dokud není zastřelen personál).
- Zneškodnění útočníka obrannou jednotkou (vzdání se / zastřelení).
- Útok je ukončen.
- Mrtvý personál je oživen.

Testy uvedené výše pomohly odhalit několik chyb, které v kódu byly. Jako příklad uvádím, že v případě úspěšného úniku nebyl útok řádně ukončen a nedošlo ke spawnutí dalšího útočníka.

Další identifikovanou chybou bylo opakování volání funkce *GotMe*, která se vykonává při chycení útočníka obrannou jednotkou a tím pádem i vícečetné volání funkce **AttackManageru** *StopAttack*. Tato chyba byla opravena kontrolou stavu útočníka uvnitř problematické funkce. Podobný problém se vyskytoval i v případě

zastřelení útočníka (*OnTriggerEnter*), kde docházelo (mohlo dojít) k vícečetnému volání funkce a dokonce i k souběžnému zavolání funkce *GotMe*. I toto nežádoucí chování bylo eliminováno kontrolou stavu subjektu.

V rámci konceptu původního simulátoru bylo identifikováno nečekané zamrzání aplikace při vzdávání se / úmrtí útočníka. Po provedených úpravách tento problém nebyl identifikován, lze tedy předpokládat, že byl vyřešen úpravami v kódu.

Další testy se zaměřily na ukládání paměti.

- **Test načtení paměti při spuštění – neexistující soubor**
- **Test načtení paměti při spuštění – existující soubor, špatná verze (selhání načtení)**
- **Test načtení paměti při spuštění – existující soubor (úspěšné načtení)**
- **Test uložení paměti – neexistující soubor**
- **Test uložení paměti – soubor již existuje (přepsání souboru)**
- **Test resetu paměti – neexistující soubor**
- **Test resetu paměti – existující soubor (soubor je smazán)**
- **Komplexní test resetu paměti**

- Reset paměti (neúspěšný)
- Uložení paměti
- Reset paměti (úspěšný)
- Reset paměti (neúspěšný)

4.5 Shrnutí

V průběhu testování byla uskutečněna řada testů. Jednotlivé testy ověřily fungování nově implementovaných vlastností simulátoru. Zároveň bylo provedeno testování fungování simulátoru jako celku, nejprve bez zásahu uživatele a poté včetně zásahu. Všechny identifikované chyby v rámci testů (s výjimkou identifikované nedokonalosti modelu letiště) byly vyřešeny.

5 Závěr

Cílem mé práce bylo rozšířit a upravit simulátor za použití umělé inteligence a učení se z předchozích simulací tak, aby bylo docíleno větší realističnosti chování jednotlivých postav v simulátoru.

Klíčovou provedenou úpravou pro splnění zadání byla implementace paměťového modulu, který zaznamenává vybrané statistické údaje o jednotlivých útocích v závislosti na typu útočníka a místě útoku. Tyto údaje jsou dále využity při výběru místa útoku a rovněž při rozhodování, zda se má útočník pokusit o útěk či nikoliv.

Co se týče chování jednotlivých typů postav, zde došlo k rozsáhlým úpravám celého procesu.

Z pohledu útočníka byl kompletně předělán výběr cílů. Původní koncept simulátoru počítal s tím, že útočníku je přiřazen právě jeden živý cíl, dokud není zastřelen. Po implementaci úprav dostává útočník seznam dostupných cílů, se kterými dále pracuje. Z dalších úprav bych vystihl výše zmíněné nové funkcionality, využívající data z předchozích simulací. Výběr optimálního místa útoku pro útočníky na halu kombinuje mnou definovaná kritéria – maximalizovat pravděpodobnost úspěšného útěku a rovněž zabít co nejvíce postav. Implementovaná možnost útěku útočníka, doprovázená animací běhu, dle mého názoru přispívá ke zkvalitnění simulace a zároveň díky nastaveným kritériím dokáže postupně zvyšovat obtížnost simulace v závislosti na úspěšnosti ve zdolávání hrozeb.

Z pohledu cestujícího dle původního konceptu došlo v případě útoku ke „vzdání se“ všech cestujících nehledě na jejich vzdálenost a umístění vůči útočníkovi. Tento proces se podařilo vylepšit jednak díky úpravám při výběru cílů útočníka, zmíněných výše, a dále díky nové funkcionalitě, která dynamicky v rámci procesu chování cestujícího identifikuje případnou hrozbu v podobě útočníka a provede příslušnou reakci. Zde je nutno poznamenat, že cestující, kteří identifikují útočníka v rámci svého procesu, stejně tak cestující, kteří jsou ve chvíli začátku útoku v přímém ohrožení útočníkem, se

vzdají. V případě, že by byla definována vhodná logika, bylo by možné toto chování dále vylepšit o situaci, kdy se cestující pokusí o útěk. Vzhledem k tomu, že útočníci a cestující sdílejí stejný model postavy, je příslušná animace útěku již implementována, pouze by bylo třeba doplnit potřebné přechodové stavy v animačním schématu. Z pohledu osoby nacházející se v simulaci toto však nemá velký přínos (kromě vizuálního), protože by se jednalo o soupeření simulace proti simulaci a zároveň by tím útočník přicházel o potenciální cíle, které by mohl ohrozit a tím prodloužit simulaci.

Z pohledu personálu letiště zde nebylo mnoho prostoru pro úpravy. Až na pracovníky u detekčních rámů jsou veškeří zaměstnanci umístěni v kioskách nebo za přepážkami, ze kterých není cesta úniku. Nicméně v souvislosti s provedenými úpravami pro cestující, byla provedena identická úprava v chování personálu, kdy se vzdají pouze ti, kteří jsou v ohrožení (na dostřel) útočníka. Pro personál bylo dále doimplementováno automatické oživení po skončení útoku. Toto bylo učiněno primárně z důvodu, že by nebylo z mého pohledu žádoucí, aby personál, zajištující provoz na přepážkách, nebyl přítomen. Z pohledu fungování reálného letiště by se dalo předpokládat, že letiště má dostatek personálu pro případ nečekané situace.

Je nezbytné poznamenat, že v rámci vývoje bylo nutné v souvislosti s provedenými úpravami provést řadu dalších změn. Například pro efektivnější sběr dat byla upravena kritéria skončení simulace.

Dále v rámci testování, které je detailně popsáno v kapitole 4, jsem se zaměřil i na otestování simulátoru jako celku a rovněž na eliminaci jak identifikovaných chyb, tak varování, které se v rámci testů objevily.

V závěru kapitoly 2 byla definována kritéria chytrého agenta. Vzhledem k tomu, že se útočník učí ze získaných statistických dat, pak splňuje kritéria adaptivního agenta. Dále jsou jeho některé akce (např. rozhodnutí o útěku) řízeny na základě jeho znalostí, tudíž by se dal označit i za proaktivního agenta. Protože v rámci útoku předává informaci cílům o tom, že je na ně útočeno, a rovněž přijímá informaci, že má někoho nově na dostřel, dal by se (dle mého názoru) považovat útočník z části i za

kooperativního agenta. Splněním těchto tří kritérií lze agenta dle definice Chowdharyho považovat za chytrého.

Na závěr bych rád zmínil přínos tohoto projektu nejen z pohledu zadavatele, ale i z osobní roviny. Jednalo se o můj první projekt vývoje aplikace pro virtuální realitu, obohacený o nutnost porozumět kódu, který napsal někdo jiný, což považuji za obtížnější, než vyvíjet libovolný software „na zelené louce“, kdy má programátor větší volnost v rozhodování.

6 Zdroje / použitá literatura

[1] *AsterionVR* [online]. © 2012 – 2022 ASTERION VR. [vid 12. 5. 2022]. Dostupné z: https://asterionvr.com/

[2] *Rampvr virtual reality traing tool* [online]. © 2022 International Air Transport Association (IATA). [vid. 12. 5. 2022]. Dostupné z: https://www.iata.org/en/training/pages/rampvr/

[3] BURSÍK, F. *Prototyp systému pro taktické cvičení ostrahy letiště.* Praha, 2021 Diplomová práce. ČVUT v Praze, Fakulta elektrotechnická, Katedra počítačů.

[4] *Oculus Rift S* [online]. © Facebook Technologies LLC. [vid. 12. 5. 2022]. Dostupné z: https://web.archive.org/web/20200622231954/https://www.oculus.com/rift-s/

[5] *Vive* [online]. © 2011-2022 HTC Corporation. [vid. 12. 5. 2022]. Dostupné z: https://www.vive.com/us/

[6] *SteamVR* [online]. © 2022 Valve Corporation. [vid 12. 5. 2022]. Dostupné z: https://www.steamvr.com/cs/

[7] MAŘÍK, V., O. ŠTĚPÁNKOVÁ, J. LAŽANSKÝ a kol. *Umělá inteligence (1).* Praha: Academia, 1993. ISBN 80-200-0496-3.

[8] RICH, E., K. KNIGHT, S. B NAIR. *Artificial Intelligence (Third Edition).* New Delhi: Tata McGraw-Hill, 2009. ISBN 978-0-07-008770-5.

[9] *Základní průvodce AI* [online]. Oxford Internet Institute in partnership with Google. [vid. 12. 5. 2022]. Dostupné z: https://atozofai.withgoogle.com/intl/cs/turing-test/

[10] URWIN, R. *Artificial Intelligence.* London: Arcturus Publishing, 2016. ISBN 978-1-78-428869-3

[11] RUSSELL S., P. NORVIG. *Artificial Intelligence A Modern Approach (Third Edition).* New Jersey: Prentice Hall, 2009. ISBN 978-0-13-604259-4.

[12] ROBINSON, J. A. A Machine-Oriented Logic Based on the Resolution Principle. In: *Journal of the ACM*, 1965, 1, 23-41. ISSN: 0004-5411. Dostupné z: https://doi.org/10.1145/321250.321253

[13] KUMAR, C. Artificial Intelligence: Definition, Types, Examples, Technologies. In: *Medium* [online]. Chetan Kumar, 2018. [vid. 12. 5. 2022]. Dostupné z:

https://chethankumargn.medium.com/artificial-intelligence-definition-types-examples-technologies-962ea75c7b9b

[14] CHAMPANDARD, A. J. *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. San Francisco: New Riders, 2003.
ISBN 1-5927-3004-3

[15] DILL K. What Is Game AI? In: RABIN S. (ed.). *Game AI Pro: Collected Wisdom of Game AI Professionals*. Boca Raton: CRC Press, 2014, pp. 3-9.
ISBN 978-1-4665-6596-8

[16] *Arma III* [online]. © 2022 BOHEMIA INTERACTIVE a.s. [vid 12. 5. 2022]. Dostupné z https://arma3.com/

[17] *Half-Life: Alyx* [online]. © 2022 Valve Corporation [vid 12. 5. 2022]. Dostupné z https://www.half-life.com/cs/alyx

[18] CHOWDHARY, K. R. *Fundamentals of artificial intelligence.* New Delhi: Springer Nature, 2020. ISBN 978-81-322-3970-3.

[19] FRENCH, J. L. Start vs Awake in Unity. In: *Game Dev Beginner* [online]. John Leonard French, 2020. [vid. 12. 5. 2022]. Dostupné z: https://gamedevbeginner.com/start-vs-awake-in-unity/

[20] *NavMeshAgent* [online]. © 2021 Unity Technologies. [vid. 12. 5. 2022]. Dostupné z: https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html

[21] *Inner workings of the navigation system* [online]. © 2021 Unity Technologies. [vid. 12. 5. 2022]. Dostupné z: https://docs.unity3d.com/Manual/nav-InnerWorkings.html

[22] *Unity Answers* [online]. © 2020 Unity Technologies. [vid. 12. 5. 2022]. Dostupné z: https://answers.unity.com/questions/120387/measuring-the-length-of-objects.html

[23] *Mixamo* [online]. © 2022 Adobe Systems Incorporated. [vid. 12. 5. 2022] Dostupné z: https://www.mixamo.com/

[24] PARISI, T. *Learning Virtual Reality: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile*. Sebastopol (California): O'Reilly Media, 2015. ISBN 063-6-920-03846-7.

[25] *Unity Documentation* [online]. © 2021 Unity Technologies. [vid. 12. 5. 2022]. Dostupné z: https://docs.unity3d.com/Manual/index.html

[26] SAPIO, F. *Unity UI Cookbook.* Birmingham: Packt Publishing, 2015.

ISBN 978-1- 78588-582-2

[27] STEPHENS, R. *C# 5.0 Programmer's Reference.* Indianapolis: John Wiley & Sons, 2014. ISBN 978-1-118-84728-2.

[28] PATTON, R. *Testování softwaru.* Praha: Computer Press, 2002. ISBN 80-7226-636-5

[29] Cena opravy vady. In: *systemonline.cz* [online]. © 2001 - 2022 CCB spol. s r.o. [vid. 12. 5. 2022]. Dostupné z https://www.systemonline.cz/erp/testovani-v-procesu-implementace-informacniho-systemu.htm

[30] *Netfabb* [online]. © 2020 Autodesk, Inc. [vid. 12. 5. 2022]. Dostupné z https://www.autodesk.cz/products/netfabb/

7 Seznam příloh

A. Mapa letiště

B. Obsah přiložené SD karty

8 Přílohy

A. Mapa letiště

Obrázek A.1: Mapa letiště s popisky

Žlutý bod – počáteční místo hry – startovní bod obranné jednotky.

Fialový bod – *escapePoint* pro útočníky na halu.

Zelená linka – servisní vstup ostrahy na letištní plochu.

Červené body – spawnpointy útočníků na ranvej.

Modré body – cíle útočníků na ranvej.

Hnědé linky – díry v plotě – místa průniku útočníku na ranvej na letištní plochu.

B. Obsah přiložené SD karty

```
+- simulator.zip
        +- simulator-master/              -- složka zdrojového kódu DP
                +- Assets/
                        ...
                        +- Scripts/       -- složka se skripty
                        ...
                +- Library/
                +- LocalPackages/
                +- Logs/
                +- obj/
                +- Packages/
                +- ProjectSettings/
                +- Properties/
                +- UserSettings/
        +- Build                          -- složka sestaveného simulátoru
                +- AirportVR_Data/
                +- MonoBleedingEdge/
                +- AirportVR.exe          -- spustitelný simulátor
        +- readme.txt                     -- pokyny ke spuštění simulátoru
        +- ukazka.mkv                     -- videosoubor s ukázkou běhu simulátoru
+- DP_glingvla.pdf                        -- text práce v PDF
```

Poznámka: velká část složek jsou technické složky vývojářského studia. Klíčovou složkou je *simulator-master/Assets/Scripts*, kde jsou uloženy jednotlivé skripty, řídící jednotlivé postavy a zajištující fungování simulátoru. Pro přehlednost jsou v hierarchii výše uvedeny pouze složky v prvních dvou úrovních.

Složka *Build* obsahuje sestavený projekt včetně spustitelného *exe* souboru. Bližší detaily ke spuštění jsou uvedeny v souboru *readme.txt*