TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Hra pre experimentálne posúdenie kognitívnych funkcií vo virtuálnej realite: herná logika a dizajn

Diplomová práca

2019 **Bc. Peter Vasiľ**

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Hra pre experimentálne posúdenie kognitívnych funkcií vo virtuálnej realite: herná logika a dizajn

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko, PhD.

2019 Košice **Bc. Peter Vasiľ**

Abstrakt

Táto diplomová práca sa zaoberá vytvorením hry, ktorá bude prostriedkom pre realizáciu experimentu, ktorý realizuje Technická univerzita v Košiaciach v spolupráci s vedcami zo Slovenskej akadémie vied a Univerzity Komenského v Bratislave. Experiment sa bude zameriavať a bude testovať vplyv dlhodobejšieho hrania hry v prostredí virtuálnej reality na kognitívne schopnosti človeka. Hra je implementovaná a prispôsobená pre prostredie virtuálno-realitnej jaskyne (CAVE), ktorá sa nachádza v laboratóriu LIRKIS na Technickej univerzite v Košiciach. Moja časť práce pokrýva návrh a implementovanie celej hernej logiky spolu s jej audiovizuálnou stránkou.

Kľúčové slová

Virtuálna realita, kognitívne funkcie, simulácia v prostredí virtuálnej reality, kognitívna veda, 3D priestor

Abstract

This diploma thesis deals with the creation of a game, which will be used in an experiment, prepared by the Technical University in Košice in cooperation with scientists from the Slovak Academy of Sciences and Comenius University in Bratislava. The experiment will focus on and test the impact of longer-term a virtual reality experience on human cognitive abilities. The game is implemented and adapted for the virtual-reality environment, especially for virtual reality cave, which is located in the laboratory LIRKIS at the Technical University in Košice. This work covers the design and implementation of the entire game logic along and its audio-visual part.

Keywords

Virtual reality, cognitive functions, human in the loop simulation, simulation in virtual reality environment, cognitive science, 3D space, visuospatial functions

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

Hra pre experimentálne posúdenie kognitívnych funkcií vo virtuálnej realite: herná logika a dizajn

Game for Experimental Assessment of Cognitive Functions in Virtual Reality: Game Logic and Design

Študent: **Bc. Peter Vasiľ**

Školiteľ: **Ing. Štefan Korečko, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Oboznámiť sa so softvérovým a hardvérovým vybavením virtuálno-realitnej jaskyne LIRKIS CAVE.
2. Analyzovať požiadavky na hru sformulované expertmi z oblasti kognitívnej vedy.
3. Pri analýze sa zamerať na požiadavky na priebeh a audiovizuálnu stránku hry a ich realizovateľnosť v LIRKIS CAVE.
4. Na základe analýzy pre LIRKIS CAVE navrhnúť a implementovať jadro hry, zahŕňajúce základnú hernú logiku a audiovizuálnu stránku.
5. Pri návrhu a implementácii spolupracovať s riešiteľom súvisiacej záverečnej práce.
6. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 26.04.2019

Dátum zadania diplomovej práce: 31.10.2018

doc. Ing. Jaroslav Porubän, PhD.
vedúci garantujúceho pracoviska

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -10-

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 26. apríla 2019 .....

vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce za jeho čas, odborné vedenie a rady počas riešenia mojej záverečnej práce. Ďalej by som sa rád poďakoval aj ľuďom z laboratória LIRKIS, za ich rady, pomoc a podporu pri testovaní výsledku tejto práce. Na záver patrí veľká vďaka celej mojej rodine, ktorá ma počas celého môjho štúdia podporovala.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Dôvodov, prečo som si ako tému svojej práce vybral práve prácu v oblasti virtuálnej reality je viacero. Ide o technológiu, ktorá sa v posledných rokoch dostáva čoraz viac a viac do popredia, či už v oblasti vedy a techniky, tak aj v oblasti počítačových hier, medicíny ako aj iných, bežných odvetviach, ktoré sú blízke takmer všetkým ľuďom. Veľký praktický význam má virtuálna realita práve v spojení s rôznymi inými vednými obormi.

V tejto práci je skĺbená práve oblasť medicíny, konkrétne oblasť kognitívnej vedy, a informatiky. Pôjde tu o experiment, v ktorom sa budú skúmať a podrobne analyzovať kognitívne schopnosti človeka, presnejšie ich dlhodobé zmeny, vplyvom zážitku vo virtuálnej realite, ktorý má formu hrania počítačovej hry. Počítačová hra, ktorá bude implementovaná práve v prostredí virtuálnej reality, má slúžiť ako prostriedok na správnu stimuláciu kognitívnych schopností, či už sa jedná o pamäťové schopnosti, schopnosti rýchlo rozmýšľať a riešiť problémy alebo o pozornosť človeka. Tento experiment bude realizovaný v spolupráci s ľuďmi zo Slovenskej akadémie vied v Bratislave, ktorí sú odborníkmi v obore kognitívnej vedy, a takisto s odborníkmi z Katedry počítačov a informatiky na Technickej univerzite v Košiciach.

Samotný experiment bude pozostávať z prieskumu vzorky účastníkov, ideálne v rovnomernom rozložení mužov a žien. Experiment bude pre každého účastníka začínať vyplnením krátkeho vstupného formuláru a EEG meraním. Pri vstupnom formulári pôjde o zistenie základných informácii o účastníkovi – zistenie demografických údajov, či už má, prípadne aké veľké má skúsenosti s podobnými experimentami, aké skúsenosti má v oblasti hrania počítačových hier, v oblasti virtuálnej, zmiešanej, prípade rozšírenej reality a podobne. EEG meranie slúži ako prostriedok pri vykonávaní behaviorálneho testu, konkrétne Change Detection Task (CDT). Tieto dva základné testy by mali zistiť a určiť tak vhodnú počiatočnú úroveň účastníka v hre a zaručiť tým fakt, že daná obtiažnosť hry nebude pre konkrétneho účastníka príliš jednoduchá, prípadne príliš zložitá, čo by mohlo mať negatívny dopad na výsledky celého experimentu.

Výhodou a prínosom takéhoto experimentu je práve rozšírenie poznatkov v oblasti kognitívnej vedy, a to práve v oblasti výskumu vplyvu rôznorodých faktorov z okolitého sveta (efekt sa má dosiahnuť práve použitím virtuálno-realitnej technológie) na kognitívne schopnosti človeka. Takisto je to spôsob, akým rozšíriť vnímanie virtuálnej reality ako technológie a ukázať, akými rôznymi spôsobmi môže byť prínosom aj v iných oboroch.

1. Formulácia úlohy a cieľ práce

Cieľom tejto diplomovej práce je navrhnutie a implementovanie hry, ktorá bude slúžiť ako prostriedok pre experimentálne posúdenie kognitívnych funkcií človeka v prostredí virtuálnej reality. Pri implementácii hry je potrebné sa zamerať na 2 hlavné celky. Prvým je herná logika a dizajn a druhým je získanie potrebných štatistík z hry. Táto práca sa bude venovať prvej časti, a to dizajnu a implementácii hernej logiky. Druhej časti hry sa bude venovať kolega Bc. Dominik Trojčák vo svojej diplomovej práci s názvom Hra pre experimentálne posúdenie kognitívnych funkcií vo virtuálnej realite – konfigurácia hry a zber údajov.

Pre úspešný návrh a vhodnú implementáciu hry je potrebné splniť niekoľko krokov. Na začiatku je potrebné sa oboznámiť so softvérovým a hardvérovým vybavením virtuálno-realitnej jaskyne LIRKIS CAVE, pre ktorú bude hra vytvorená a prispôsobená.

Ďalším krokom je správna analýza požiadaviek na hru, ktoré sú sformulované expertmi z oblasti kognitívnej vedy. Experiment, ktorý je plánovaný, je práve v spolupráci s nimi, preto by hra mala čo najlepšie pokryť všetky požiadavky, ktoré vyžadujú. V mojej časti práce je kľúčové zamerať sa pri analýze hlavne na požiadavky týkajúce sa priebehu a audiovizuálnej stránky hry. Pri analýze je nevyhnutné správne posúdiť realizovateľnosť týchto požiadaviek v prostredí LIRKIS CAVE, prípadne sa zamyslieť nad obmedzeniami, ktoré by mohli nastať počas implementácie.

Jedným z najdôležitejších krokov je na základe predošlej analýzy navrhnúť a implementovať jadro hry, zahŕňajúce celú hernú logiku a audiovizuálnu stránku. Pri návrhu a implementácii je potrebné spolupracovať s kolegom Bc. Dominikom Trojčákom, ktorého úlohou je riešenie konfigurácie a získavanie štatistík z danej hry.

Na záver je potrebné vypracovať všetku potrebnú dokumentáciu ku implementovanému systému.

2. Analýza stavu problematiky

Táto kapitola sa zaoberá detailnou analýzou stavu problematiky. Jej úlohou je oboznámiť nás s cieľami experimentu a cieľami danej hry, s jej špecifikáciou a požiadavkami na hru. Ciele hry sú rozdelené ako z pohľadu používateľa, tak aj z pohľadu celého experimentu. Ďalej je v tejto kapitole popísané ako sa bude daný experiment realizovať, aká je cieľová skupina účastníkov experimentu a ako bude daný experiment prebiehať. Z pohľadu implementácie je v tejto kapitole analyzovaný systém, ktorý sa pre vykonávanie experimentu použije. Rovnako sú v tejto kapitole analyzované aj jednotlivé požiadavky na hru z pohľadu implementácie, aké problémy by mohli vzniknúť a čo by mohlo byť najvhodnejším riešením pri samotnej implementácii konkrétnej požiadavky.

2.1. Význam a cieľ experimentu

Ako sa uvádza v článku Assessment and training of visuospatial cognitive functions in virtual reality: proposal and perspective, vizuálno-špecifické funkcie hrajú kľúčovú úlohu v kognitívnom poznaní človeka, čo v priebehu posledných rokov vyvolalo veľa výskumov zameraných na ich hodnotenie, posudzovanie a trénovanie. Zaujímavé je, že aj keď nám naše vizuálno-špecifické možnosti umožňujú pochopiť a odvodiť vzťahy 3D objektov v priestore, tieto 3D aspekty vizuálno-špecifické spracovania sú často v laboratórnych testoch zanedbávané a namiesto toho sa bežne používajú 2D návrhy. S cieľom zvýšiť platnosť takýchto testov je navrhnutý experiment na vyhodnotenie kapacity 3D virtuálneho priestoru s účelom stimulovania kognitívnych funkcií človeka. Experiment zahŕňa testovanie kognitívnych funkcií, EEG merania a úlohy stimulujúce kognitívne funkcie u človeka vo virtuálnom 3D prostredí vytvorenom pomocou CAVE systému, s cieľom posúdiť potenciálny účinok hry pred a po jej dlhodobejšom hraní. [1]

2.2. Opis a cieľ hry

V tejto kapitole je stručne opísané, o čom má byť daná hra a čo je jej cieľom a zmyslom z pohľadu účastníka, ktorý bude danú hru hrať, tak aj z pohľadu samotného experimentu.

Samotná hra má slúžiť ako prostriedok pre experiment, ktorý sa má zameriavať na posúdenie kognitívnych funkcií človeka v prostredí virtuálnej reality. Tento experiment by mal zatažiť priestorovú pamať a priestorové vnímanie z pohľadu človeka, taktiež by sa mal zamerať na vnímanie priestorových vzťahov medzi objektami a následne vyhodnotiť dlhodobejší vplyv hrania takejto špeciálne vytvorenej virtuálno-realitnej hry na kognitívne schopnosti človeka.

Navrhnutá hra sa nazýva Tower Defense a bola vyvinutá špeciálne na účely tohoto experimentu. V hre sa vyskytuje samotný hráč, ktorý bude danú hru ovládať a je takisto subjektom experimentu. Jeho úlohou bude zostreľovať nepriateľské drony, ktoré budú nalietavať proti nemu.

Čo však do hry vnáša zmysel je to, že drony nalietavajúce proti hráčovi sú rozdelené medzi takzvané *cieľe* – drony, na ktoré by sa mal hráč zamerať a zostreľovať ich a *distraktory* – drony, ktoré sú v scéne iba ako prostriedok na zmätenie a zaťaženie pracovnej pamäte daného hráča.

2.3. Rozdelenie cieľov hry

Ciele hry sa dajú v tomto prípade chápať, resp. sa na nich dá pozerať z dvoch uhlov pohľadu. Prvým je pohľad zo strany experimentu, to znamená, aký je zmysel hry pre samotný experiment a o čo ide v experimente. Druhým pohľadom je pohľad na cieľ hry zo strany samotného hráča, ktorý bude hrať danú hru. V druhom prípade ide teda skôr o opis jeho možností v hre, čo by sa mal snažiť dosiahnuť a akým spôsobom by to mal dosiahnuť.

2.3.1. Ciele hry z pohľadu experimentu

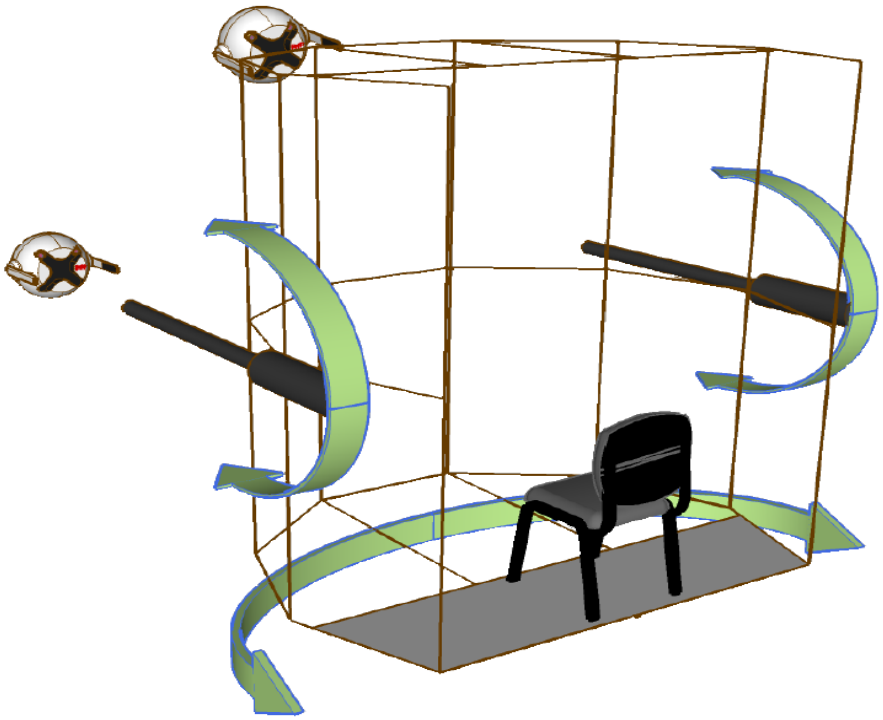
Cieľom experimentu je zistiť vplyv dlhodobejšieho hrania hry na rôzne kognitívne schopnosti človeka, ako napríklad priestorová pamäť, vzťahy medzi objektami v 3D priestore, rozpoznávanie objektov v priestore a podobne.

2.3.2. Ciele hry z pohľadu hráča

Cieľom hráča v hre bude zostreliť čo najviac nepriateľských dronov (cieľov) pomocou špeciálnych zbraní, ktoré bude vedieť ovládať a z ktorých bude vedieť strieľať. Na splnenie tohto cieľa je hráčovi k dispozícii nasledujúca funkcionalita:

- Hráčovi je umožnená streľba pomocou zbraní, ktoré ovláda ovládačom.
- Hráč sa môže pomocou ovládača natáčať smerom doprava a doľava, keďže sa vo virtuálnom svete nachádza na otáčavej platforme. Uhol otočenia je ale limitovaný konfiguráciou.
- Hráč môže pomocou ovládača nakláňať zbrane pripevnené na bokoch jeho kabíny smerom nahor resp. nadol, takisto s obmedzeným uhlom otočenia daným konfiguráciou.

Na obrázku nižšie (Obr. 1) je nakreslená schéma ako bude daná hra vyzerať v prostredí CAVE.



Obr. 1 Konceptuálny návrh hry Tower Defense

Zdroj: [1]

2.4. Špecifikácia experimentu

V tejto kapitole je bližšie popísaná špecifikácia celého experimentu, ktorá vychádza z opisu poskytnutého ľudmi zo Slovenskej akadémie vied a Univerzity Komenského v Bratislave, ktorí sú spolurealizátormi experimentu. V tejto časti budú bližšie opísaní účastníci experimentu – cieľová skupina, príprava na experiment a samotný priebeh plánovaného experimentu.

2.4.1. Účastníci (experimentálna skupina)

Pri výbere účastníkov je dôležité vedieť najmä na akú vekovú skupinu by sme sa mali pri experimente zamerať, aké by malo byť percentuálne zastúpenie jednotlivých pohlaví, a aká veľká by mala byť experimentálna skupina, aby sme dostatočne vylúčili náhodné javy a rôzne odchýlky pri experimente. Na základe toho boli pre experiment zvolené nasledujúce parametre:

- *Počet účastníkov* – minimálny počet 15, optimálny počet 20 ľudí.
- *Vek účastníkov* – 21 až 24 rokov (pravdepodobne študenti 3. - 4. ročníka na KPI na TUKE).
- *Zastúpenie pohlaví* – ideálne rovnaký počet mužov a žien (môže byť viac mužov).

2.4.2. Príprava na experiment

Tým, že každý účastník je jedinečná bytosť, je potrebné pre každého z nich na začiatku zistiť vhodný počiatočný level pre experiment (CDA/T) a hru. To sa dosiahne vyplnením vstupného formuláru konkrétnym účastníkom a absolvovaním behaviorálneho testu CDT (Change Detection Task), čo sa zopakuje 3 krát v priebehu 1 týždňa.

- *Vstupný formulár* – zisťovali by sa základné demografické údaje, frekvencia hrania hier, prípadne užívateľská skúsenosť s technológiou virtuálnej, zmiešanej alebo rozšírenej reality + inštruktáž (briefing). Odhad trvania tejto časti je približne 10 - 15 minút na jedného účastníka.
- *Behaviorálny test* – tento test sa bude realizovať v koordinácii s odborníkmi zo SAV. Odhad trvania tejto časti je približne 30 minút na jedného účastníka.
- *Výstupný formulár* – zisťovali by sa zmeny vnímané používateľom a spätná väzba v podobe návrhov a možných zlepšení v hre a v priebehu celého experimentu. Odhad trvania tejto časti je približne 10 - 15 minút na jedného účastníka.

2.4.3. Priebeh experimentu

Nižšie sú popísané kroky experimentu v presnom poradí, v akom musia byť realizované.

1. Vstupný formulár + EEG meranie (CDA/T)
2. 5 tréningových sedení
3. EEG meranie
4. 5 tréningových sedení
5. EEG meranie + výstupný formulár

Všetky tréningové sedenia sú rovnako dlhé. Každé pozostáva z troch hracích úsekov po 8 minút s krátkymi prestávkami (2 minúty). Spolu je teda odhadovaná doba trvania jedného sedenia 30 minút.

2.5. Špecifikácia požiadaviek na hru

Nižšie je popísaná špecifikácia požiadaviek na hru od zadávateľov, ktorými sú ľudia zo Slovenskej akadémie vied v Bratislave, ktorí sú spolurealizátormi experimentu. Požiadavky sú popísané v jednotlivých bodoch nižšie:

- Hra pozostáva z niekoľkých úrovní (levelov) s narastajúcou zložitosťou. Každý level predstavuje vopred stanovený počet behov (trials) = “náletov”. Každý level je daný svojou zložitosťou zhluku – bod 3. (Požiadavka hry 1, ďalej len PH1)

- V rámci jedného behu sa blížia objekty v menšom počte, pričom vytvárajú priestorový zhluk, ktorý sa v čase nerozptýli (objekty majú podobný smer pohybu a rovnakú rýchlosť). Zhluk sa vynorí v ľubovoľnej (zadnej) časti priestoru a lineárne sa pohybuje smerom k rovine hráča, no nemusí smerovať priamo na hráča. Všetky objekty sa musia objaviť v rovnakom čase. (PH2)

- Veľkosť a zloženie zhluku je dané úrovňou hry, ale trochu sa mení aj v rámci úrovne. Označenie zhluku je #T/#D, kde # = počet, T = cieľ, D = distraktor (napr. 3/2 znamená 3 cieľové objekty a 2 distraktory). Každá úroveň hry má stanovenú strednú hodnotu, pričom jednotlivé inštancie v rámci hry danej úrovne sa môžu trochu líšiť o hodnotu +/- 1 kus (čiže mierne sa mení počet cieľov a/alebo počet distraktorov v daných epizódach). Vo vstupnej konfigurácii pre jednotlivé levely bude uvedená hodnota #T/#D, charakterizujúca daný level, parametrom bude tiež variabilita okolo strednej hodnoty. (PH3)

- Objekty v rámci zhluku majú rovnakú farbu, líšia sa len tvarom (uvažujeme N rôznych 3D tvarov polygonálneho typu, nech N = 5), pričom v rámci každého behu je jeden typ objektu cieľom, ostatných N - 1 typov sú distraktory. Všetky typy objektov majú približne rovnakú tvarovú zložitosť a mali byť vzájomne zhruba rovnako nepodobné (odlíšiteľné). (PH4)

- Typ cieľového objektu sa mení v každom behu a indikuje sa na obrazovke (napr. nejakou značkou na cieľovom objekte) v čase, keď sa zhluk vynorí, aby si to hráč mohol všimnúť. (PH5)

- Počas behu raz (počas existencie zhluku) nastane situácia, že na obrazovke nastane “výpadok prúdu”. Vtedy časť obrazovky veľmi stmavne (okrem podlahy veže) na krátku dobu (600 až 900 ms). Objekty však pokračujú v pohybe a po výpadku hráč musí reagovať ako predtým (cieľové objekty už budú o čosi bližšie). Výpadok prúdu má za cieľ zaťažiť vizuálnu pracovnú pamäť. (PH6)

- Po výpadku môžu nastať dva prípady: jeden z cieľových objektov sa trochu natočí alebo nenatočí. V oboch prípadoch by hráč mal cieľový objekt rozpoznať. Zmena by nemala byť výrazná (napr. natočenie v nejakej rovine o malý uhol). (PH7)

- Cieľom hráča je zostreľovať cieľové objekty a (podľa možnosti úplne) ignorovať distraktory, ktoré zbytočne zaťažujú vizuálnu pracovnú pamäť. (PH8)

- Kabína sa nemusí hýbať, postačí ak subjekt bude pohybovať očami, preto by malo stačiť renderovanie scény na displejoch vpredu. (PH9)

- Hra sa ovláda pomocou joysticka: pozícia cieľa (2D) a streľba na cieľ (gombíkom). (PH10)

- Hra by mala byť schopná poskytnúť report pre dané tréningové sedenie (počet behov = počet zhlukov), koľko mali cieľov a koľko distraktorov, koľko a v akom čase po obnovení "výpadku prúdu" ich zostrelil + prípadne možnosť pridať ID subjektu a samozrejme čas sedenia. (PH11)

2.6. Systémová analýza

V tejto kapitole je detailnejšie popísané prostredie, v ktorom sa bude daný expertiment realizovať resp. pre ktoré bude hra implementovaná a prispôsobená.

2.6.1. Opis a možnosti prostredia

Virtuálno-realitný systém vybraný pre tento experiment je virtuálno-realtiná jaskyňa, konkrétne LIRKIS CAVE, ktorý sa nachádza na Technickej univerzite v Košiciach a je súčasťou laboratória LIRKIS. Ide o kompaktné prenosné prostredie virtuálnej reality so zobrazovacou plochou 2,5 x 2,5 x 3 metre. Jeho vizuálny výstup je vykreslený na dvadsiatich 55-palcových stereoskopických LCD paneloch. 14 z týchto panelov je umiestnených vertikálne pozdĺž 7 strán dekagónu. Vďaka týmto vlastnostiam poskytuje CAVE 250-stupňový panoramatický priestor. Zvyšných 6 panelov je umiestnených horizontálne na strope (3 panely) resp. podlahe (3 panely). CAVE podporuje širokú škálu vstupných zariadení pre používateľské ovládanie. Tieto vstupné zariadenia zahŕňajú širokú škálu, od bežných zariadení ako myš, klávesnica, Joystick resp. Gamepad, až po špeciálne ako sú MYO a OptiTrack pre zachytávanie pohybov používateľov. Renderovanie virtuálnych scén ako aj interakcia používateľa sú zabezpečené pomocou klastra zloženého zo 7 počítačov, ktoré sú vybavené grafickými kartami Nvidia Quadro.

Obr. 2 CAVE v laboratóriu LIRKIS na Technickej univerzite v Košiciach

Zdroj: [1]

2.6.2. Opis a možnosti enginu SuperEngine

SuperEngine je vizualizačný engine od slovenskej firmy Slovakia Supercomputers. Podporuje skriptovanie v jazyku Ruby, ktorým sa dá ovládať scéna bežiaca v CAVE prostredí. Najväčšou výhodou tohto engine je to, že už je vyskúšaný a overený v laboratóriu LIRKIS a je momentálne jediným plnohodnotným a funkčným riešením pre prácu s LIRKIS CAVE. Takisto sú celkom dobre preskúmané jeho možnosti, teda približne vieme, čo s jeho podporou môžeme, a čo nemôžeme v CAVE docieliť.

Medzi veľké výhody SuperEngine patrí taktiež jeho plná integrácia a funkčnosť so zariadením OptiTrack, ktoré je súčasťou hardvéru v laboratóriu LIRKIS.

Jeho veľkou nevýhodou je však veľmi slabá podpora, čo sa týka rozhrania pre programovanie aplikácii (API), ktoré nám v podstate ponúka iba základné možnosti pre vyobrazovanie objektov v 3D scéne a manipuláciu s nimi.

Medzi základné možnosti, čo SuperEngine API ponúka patria:

- Umiestnenie 3D objektu do scény, nastavenie jeho súradníc
- Rotácia objektu v scéne
- Binárne riešenie viditeľnosti objektu vloženého do scény – objekt je možné zobraziť alebo skryť
- Funkcia pre lineárnu interpoláciu
- Umiestnenie kamery do scény a pohyb kamerou v scéne

Ďalšou nevýhodou je, že architektúra nieje vhode prispôsobená pre distribuované riešenie pomocou klastra, keďže daný skript je replikovaný na všetkych počítačoch v klastri, a tým pádom výpočty vykonávané v skripte zaťažujú každý jeden počítač v klastri.

Ďalšou, avšak nie úplne kritickou, nevýhodou je verzia Ruby, ktorá je momentálne podporovaná. Je to verzia 1.6, ktorá je už dosť zastaralá (najnovšia stabilná verzia Ruby je momentálne 2.6.3) a SDK pre túto verziu neobsahuje veľa dobre použiteľných možností a jazykových konštrukcií ako tomu je v novších verziách Ruby.

2.7. Analýza požiadaviek na hru

V nasledujúcich podkapitolách sú jednotlivo analyzované všetky požiadavky na hru od zadávateľa, rovnako je analyzovaný možný spôsob implementácie jednotlivých požiadaviek, vzhľadom na softvérové a hardvérové možnosti poskytnuté v prostredí virtuálno-realitnej jaskyne v laboratóriu LIRKIS, ktoré bolo zvolené pre tento experiment.

2.7.1. Rozdelenie hry do herných celkov – scenár hry

Do úvahy pripadá viacero možností štrukturovania danej hry na nejaké logické celky v priebehu hrania. Nižšie sú uvedené 2 príklady potencionálneho scenáru hry:

1. Rozdelenie hry na levely

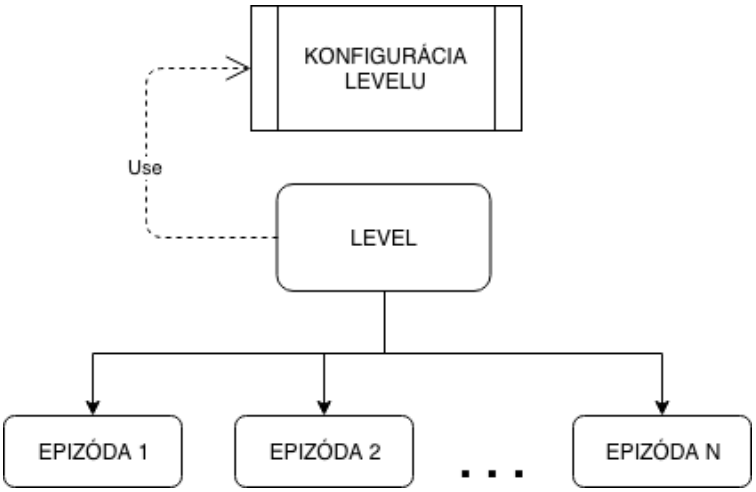
Prvý typ delenia by predstavoval rozdelenie hry na levely, kde level je jedna hra, ktorá má konfiguráciu definovanú daným levelom, kedy hra beží a hráč má možnosť sa aktívne zapájať do priebehu hry. Každý level by sa od seba líšil rôznosťou konfigurácie, ktorá by bola pre jednotlivé levely vopred definovaná.

2. Jemnejšie rozdelenie hry na levely a epizódy

Takéto rozdelenie je ďalšou z možností ako štrukturovať hru. Je podobný predošlému rozdeleniu, avšak levely sú ešte rozdelené na viacero logických celkov, ktoré sa nazývajú epizódy.

To znamená, že level pozostáva z jednej alebo viacerých epizód. Konfigurácia je na úrovni levelov, to znamená, že jednotlivé epizódy v leveli sú generované na základe konfigurácie daného levelu. Jednotlivé levely sa rovnako ako v predošlom prípade od seba budú líšiť rôznosťou konfigurácie, čím sa docieli riešiteľnosť zložitosti daných levelov.

Z analýzy rozdelenia hry do logických herných celkov a zároveň z analýzy požiadaviek od zadávateľa vyplýva, že výhodnejším riešením bude štrukturovanie na levely a epizódy. To je zobrazené na obrázku 3. Výhodou takéhoto riešenia je, že jednotlivé epizódy môžu predstavovať osobitné nálety (PH1), resp. vlny cieľov a distraktorov, ktoré sa začnú pohybovať smerom k hráčovi – každá nová epizóda bude predstavovať nový nálet. Konfigurácia bude pre zjednodušenie umožnená na úrovni levelu, bolo by nepraktické a zdĺhavé konfigurovať každú jednu epizódu v danom leveli, keďže pôjde o rádovo desiatky epizód v rámci každého levelu.



Obr. 3 Znázornenie rozdelenia hry na levely a epizódy

2.7.2. Analýza entít vyskytujúcich sa v hre

Podľa PH3 sa v hre majú vyskytovať objekty typu cieľ a distraktor. V tomto prípade sa však jedná o rovnakú entitu s jednou rozdielnou vlastnosťou resp. s jedným rozdielnym príznakom – typ *cieľ* je potrebné zostreliť, naopak typ *distraktor* sa zostreľovať nemá. Inak sú tieto objekty úplne zhodné, tým pádom môžu predstavovať rovnakú entitu, ďalej označovanú ako *dron*.

Pre každý dron je potrebné spĺňať nasledujúce požiadavky:

- Potrebuje sa pohybovať.
- Potrebuje mať priradený konkrétny vizuálny (grafický) model.
- Potrebuje mať príznak, ktorým sa budú rozlišovať ciele od distraktorov.

Ďalšou entitou, ktorá vyplýva z PH8 je samotný hráč. Ten však nepotrebuje byť nijako vizuálne reprezentovaný, keďže sa fyzicky nachádza v prostredí CAVE. V každej hre existuje práve jeden hráč.

Hráč by mal mať schopnosť vykonávať všetky nasledujúce akcie:

- otáčanie sa smerom do strán – doprava resp. doľava,
- zameriavanie dronov a
- streľba na drony.

2.7.3. Riešenie náročnosti hry

Ako bolo spomenuté vyššie, náročnosť danej hry je definovaná konfiguráciou daného levelu. Táto podkapitola sa bude venovať analýze jednotlivých možností a vlastností hry, ktoré môžu vplývať na náročnosť danej hry a mierou ich ovplyvnenia. Priebežným testovaním rôznych experimentálnych pokusov počas tvorby hry bolo zistené, že niekedy aj malá zmena istých vlastností hry môže prudko ovplyvniť jej náročnosť. Naopak, niekedy na prvý pohľad veľká zmena môže náročnosť hry ovplyvniť iba minimálne, respektíve môže ovplyvniť náročnosť hry v nesprávnom smere z pohľadu daného experimentu. Typickým príkladom ovplyvnenia náročnosti hry v nesprávnom smere je sťaženie ovládania hry, napríklad nevhodnou citlivosťou ovládača, čo však nieje cieľom experimentu, tým pádom by takéto zmeny náročnosti hry mali byť čo najviac eliminované. Je teda potrebné analyzovať, ktoré vlastnosti hry je vhodné meniť, do akej veľkej miery ich je vhodné modifikovať, a ktoré vlastnosti by mali ostať nemenné.

Predbežným testovaním základného prototypu hry sa zistilo, ktoré vlastnosti najviac ovplyvňujú náročnosť hry a do akej veľkej miery. Po konzultáciách s odborníkmi, ktorí s nami spolurealizujú experiment, bola dohodnutá a vytvorená množina parametrov, ktoré budú konfigurovateľné, majú značný vplyv na náročnosť hry, a takisto sú zmysluplné a opodstatnené pre účely daného experimentu.

Záverom analýzy náročnosti hry je nasledovná skupina meniteľných vlastností hry:

- Dĺžka epizódy
- Počet cieľových objektov
- Počet distraktor objektov
- Vertikálny a horizontálny rozptyl dronov vo virtuálnej scéne
- Pravdepodobnosť náhodne nastávajúcej udalosti v hre
- Dĺžka zobrazenia indikátora odlišujúceho ciele od distraktorov
- Možnosť zobrazenia/skrytia indikátora označujúceho stred virtuálnej scény

2.7.4. Pohyb dronov

Pre splnenie PH2 je potrebné zabezpečiť, aby sa dané drony vedeli automaticky pohybovať smerom k hráčovi. Najprv bolo analyzované, aké pohyby by mal vedieť dron vykonávať. Z PH2 a po konzultáciach vyplynulo, že je postačujúci pohyb po 1 úsečke, to znamená iba lineárny pohyb. Keďže SuperEngine nám neponúka žiadne API, ktoré by umožňovalo nejaký automatický pohyb objektu, bolo potrebné sa pozrieť po iných možnostiach. Lepším riešením ako to implementovať od začiatku bolo použiť objekt zo SuperEngine nazvaný *LinearInterpolator*. Ide o triedu, ktorej poskytneme vstupné parametre v tvare poľa, a to nasledovne:

$$[kľúč_1, súradnica\ X_1, súradnica\ Y_1, súradnica\ Z_1, kľúč_2, súradnica\ X_2, súradnica\ Y_2, súradnica\ Z_2, ..., kľúč_N, súradnica\ X_N, súradnica\ Y_N, súradnica\ Z_N]$$

Následne nad takýmto objektom vieme zavolať metódu `getValue(<kľúč1, kľúčN>)`, ktorá nám vráti súradnice X, Y a Z nachádzajúce sa na krivke vytvorenej týmto lineárnym interpolátorom pre konkrétne zadanú hodnotu kľúča.

Príklad

Chceme vytvoriť lineárnu trajektóriu medzi bodmi A, B a C, ktorých súradnice sú nasledovné (pre lepšie znázornenie odignorujeme súradnicu Z a nastavíme jej hodnotu pre všetky body na 0):

$$A[0, 0, 0], B[1, 1, 0]\ a\ C[2, 2, 0]$$

Následne potrebujeme zašpecifikovať pole, ktoré bude vstupom pre interpolátor:

$$INPUT = [\mathbf{key_A}, A_X, A_Y, A_Z, \mathbf{key_B}, B_X, B_Y, B_Z, \mathbf{key_C}, C_X, C_Y, C_Z] = [\mathbf{0}, 0, 0, 0, \mathbf{1}, 1, 1, 0, \mathbf{3}, 2, 2, 0]$$

Vidíme, že ako kľúče boli zvolené hodnoty 0, 1 a 3 (hodnoty zvýraznené hrubším písmom). Tieto hodnoty medzi sebou predstavujú akési relatívne rozostupy medzi jednotlivými bodmi. Neide však o reálne vzdialenosti medzi bodmi, tie sú špecifikované súradnicami, ide o vzdialenosti medzi kľúčmi, čo sa prejaví pri dopytovaní súradnice od interpolátora na základe kľúča. Keby sme od objektu interpolátora vypýtali hodnotu pre kľúč 0, vráti nám súradnice bodu A, keďže tam interpolátor začína. Vidíme, že bodu A a bodu B sme priradili kľúče 0 a 1, to znamená, že ich vzdialenosť z pohľadu kľúčov je:

$$key_B - key_A = 1 - 0 = 1$$

Ak chceme súradnicu, ktorá leží presne v strede medzi bodmi A a B, potrebujeme od interpolátora vypýtať hodnotu pre kľúč, ktorý sa vyráta nasledovne:

$$key = key_A + 0,5 = 0 + 0,5 = 0,5$$

Avšak vidíme, že medzi kľúčom B a kľúčom C je vzdialenosť 2. To znamená, že ak chceme presnú súradnicu medzi bodmi B a C, potrebujeme od interpolátora vypýtať hodnotu s pre kľúč vyrátaný nasledovne:

$$key = key_B + (key_C - key_B) / 2 = 1 + (3 - 1) / 2 = 2$$

V našom prípade, by teda každému dronu mohol byť priradený 1 interpolátor, z ktorého by si dron iba pýtal ďalšie body na úsečke a následne by prenastavil svoju pozíciu na dané body, čo by zabezpečilo jeho pohyb po danej úsečke.

Okrem samotného pohybu, bolo potrebné vyriešiť ešte rýchlosť pohybu daných objektov. Tu vieme opäť využiť možnosti lineárneho interpolátora zo SuperEngine, pretože vracia hodnoty na základe kľúča. Čiže, ak by sme 2 dronom priradili rovnaký interpolátor, ale jeden dron si bude v čase inkrementovať hodnotu kľúča, pre ktorý si pýta z interpolátora súradnicu napríklad o hodnotu 1, kdežto druhý dron o hodnotu napríklad 2, tak druhý dron sa bude pohybovať 2 krát rýchlejšie, ako prvý dron. Hodnoty, o ktoré sa kľúč v čase inkrementuje, v tomto príklade sú to teda hodnoty 1 a 2, predstavujú teda akúsi relatívnu rýchlosť dronov.

2.7.5. Analýza vytvárania trajektórií

Pri tejto problematike bolo potrebné vyriešiť hlavne dve základné otázky. Prvou je, akým spôsobom sa budú trajektórie vytvárať a meniť. Druhou základnou otázkou je, ako presne by mali byť trajektórie situované v priestore.

Pre vytváranie trajektórií je niekoľko možností:

1. Plne manuálne vytváranie trajektórií

Znamená to manuálne vypísanie bodov do trajektórií a ich prispôsobenie a správne umiestnenie do scény. Veľkou nevýhodou pri tomto riešení je prácnosť zmien. Napríklad, pri testovaní v CAVE sa zistilo, že daná trajektória je príliš krátka alebo je príliš nízko. Potom bolo potrebné ručne zmeniť všetky trajektórie – išlo však o veľmi zdĺhavý a nepraktický proces. Výhodou takéhoto prístupu avšak bolo, že vopred zadané a fixné trajektórie umožňovali úplnú kontrolu nad ich definíciou.

2. Manuálne vytváranie trajektórií s dynamickými variabilnými zložkami

Ide o pridanie variabilných zložiek do manuálnych definícií trajektórií, ktorými by sme vedeli dynamicky ovplyvňovať isté vlastnosti daných trajektórií, napríklad výšku umiestnenia alebo rozptyl trajektórií medzi sebou, prípadne dĺžku trajektórií. Trajektórie teda mali vo svojej definícii aj variabilnú časť, ktorá bola konfigurovateľná. Tento spôsob mal výhodu v tom, že keď všetky trajektórie boli nízko, dali sa jednoduchou zmenou jedného interného parametru posunúť vyššie, resp. sa dal iným parametrom nastaviť roztpyl trajektórií v priestore a podobne. Nevýhodou však bolo, že zmeny boli uniformné a vždy boli aplikované na všetky trajektórie, to znamená, že keď bola nízko iba polovica trajektórií, zase bolo potrebné to zmeniť manuálne pre polovicu trajektórií.

Prvé 2 možnosti majú ale jedno veľké obmedzenie. Je časovo náročné manuálne vytvoriť obrovské množstvo trajektórií, aby sme zabezpečili to, že trajektórie sa v rámci epizód budú od seba líšiť (hráč by si mal myslieť, že sú náhodne generované). Náhodné a dynamické generovanie počas behu hry by avšak bolo vo všeobecnosti problém, pretože by bolo potrebné riešiť kolízie, keďže náhodne generované trajektórie by sa mohli navzájom pretínať, a to by výrazne ovplyvnilo výpočtový výkon a mohlo by to spôsobiť nepríjemnosti pri renderovaní scény (zasekávanie scény, nízke FPS a podobne).

3. Systém na generovanie trajektórií

Ide o automatický systém na generovanie trajektórií, ktorý to vygeneruje automaticky na základe istých vstupných parametrov.

Bolo potrebné zobrať do úvahy nasledovné požiadavky:

- Pre zabezpečenie dojmu náhodnosti pre hráča je potrebné, aby generátor vedel vygenerovať veľké množstvo trajektórií, z ktorých sa následne bude vyberať istá sada trajektórií pre danú epizódu v hre.
- Generátor by mal byť navrhnutý tak, že nevygeneruje také trajektórie, medzi ktorými sú kolízie.
- V generátore by sa mali dať konfigurovať veci ako rozptyl medzi trajektóriami, výška trajektórií, dĺžka trajektórií a iné, podobne ako to bolo v druhej možnosti.

Analýzou týchto troch možností je posledná možnosť najideálnejšou na to, aby boli splnené a pokryté všetky požiadavky pre trajektórie.

2.7.6. Riešenie viditeľnosti objektov

V hre je potrebné zabezpečiť, aby drony pribúdali do scény, pri začatí epizódy sa teda zobrazili a na konci epizódy, resp. pri ich zostrele, zo scény zmizli. Tým pádom je potrebné vyriešiť viditeľnosť objektov v hre. Rovnako pri začatí epizódy sa má na istý čas nad dronami zobraziť indikátor, ktorým hráč rozlíši cieľe od distraktorov, ktorý avšak má po istom čase zmiznúť. Ďalšou situáciou, kde treba riešiť viditeľnosť je náhodne nastávajúca udalosť, nazývaná tiež výpadok prúdu, kedy na istý čas zmiznú všetky drony zo scény, stanú sa neviditeľné.

SuperEngine možnosť riešenia viditeľnosti ponúka, a to priamo vlastnosťou nad objektom modelu, pomocou ktorej sa dá nastaviť, či daný objekt je viditeľný alebo nieje viditeľný v scéne. Táto vlastnosť nad objektom typu model sa nazýva *visibility* a v prípade, že je nastavená na hodnotu 0, tak objekt nieje viditeľný v scéne, ak je nastavená na hodnotu 1, objekt je viditeľný.

SuperEngine nám teda ponúka jednoduchú možnosť ako zobraziť resp. skryť objekt v scéne. Nevýhodou takéhoto prístupu je veľký a pre ľudské oko citeľný vizuálny skok medzi zobrazením a skrytím, keďže je to vykonané okamžite, bez hociakej animácie alebo iného vizuálneho efektu. To pôsobí pre ľudské oko a vnímanie dosť neprirodzene a ruší to dobrý dojem z hry. Bohužiaľ, SuperEngine nepodporuje prácu s animáciami, preto pri požadovaní tejto funkcionality by to bolo potrebné doimplementovať.

2.7.7. Analýza modelov pre drony

Drony v hre, či už ide o cieľe alebo distraktory, potrebujú byť vizuálne reprezentované. V SuperEngine je to riešené tak, že objektu v scéne vieme priradiť istý grafický model. SuperEngine podporuje viacero formátov, v ktorých môžu byť dané modely vyexportované. Asi najpoužívanejšími sú modely vo formáte *bin*, tie je však potrebné konvertovať cez špeciálny program, preto je celý proces trochu zdĺhavý. Iným podporovaným a osvedčeným formátom je formát *3DS*, ktorého výhoda je to, že sa dá priamo vyexportovať z väčšiny modelovacích nástrojov akými sú napríklad Blender, 3ds Max a podobne. Takisto je takýto grafický model v scéne zobrazený aj s textúrou, ktorá mu je priradená v modelovacom nástroji.

Pri návrhu vhodných modelov je potrebné zvážiť a zanalyzovať zopár vlastností:

1. Tvary modelov

Čo sa týka tvarov modelov, v 3D prostredí si treba dať pozor na jeden dôležitý fakt, ktorý treba zvážiť aj pri návrhu tvarov modelov. Je veľký rozdiel, z akého uhla sa na objekt pozeráme. Tým pádom, aby boli objekty jasne odlíšiteľné, čo je v tejto hre kritické, nemali by existovať 2 rôzne

modely, ktoré sa budú líšiť iba v 1 časti, ktorá bude napríklad iba na pravej strane modelu. Dôvodom je, že ak budú takéto 2 objekty ,,nevhodne“ zrotované, môže sa stať, že ich nebude možné v danom čase a aktuálnej polohe odlíšiť. Preto by tvary modelov mali byť čím viac symetrické a čím lepšie navzájom rozlíšiteľné z rôznych uhlov pohľadu, čo minimalizuje resp. eliminuje tento problém.

V PH4 je uvedené, že grafické modely by mali mať jednoduché tvary, tým pádom je vhodné zvažovať použitie a namodelovanie primitívnych objektov ako sú guľa, kocka, kužeľ a podobne.

2. Farba modelov

Farba modelov by podľa požiadavky PH4 mala byť rovnaká pre všetky modely. Po konzultáciach s odborníkmi z Bratislavy sme sa zhodli na žltej farbe, ktorá bola dobre viditeľná v CAVE prostredí, bola dobre rozlíšiteľná, a taktiež pre hrajúci subjekt nepôsobila rušivo. Pri testovaní prototypu boli v prostredí CAVE napríklad tmavé farby ťažšie rozlíšiteľné, v čom zohrala významnú rolu aj použitá tmavá vesmírna scéna.

3. Celkový potrebný počet modelov

Pri návrhu celkového počtu modelov bolo potrebné zvážiť, koľko súčasne maximálne vyobrazených dronov počas 1 epizódy bude v scéne. Počet sa odhadol na približne 15 dronov v jednom nálete pri najťažších obtiažnostiach, tým pádom 7 rôznych modelov by malo byť dostačujúcich, keďže dané modely sa v rámci epizódy môžu aj opakovať.

2.7.8. Ovládanie hry

Aktuálne je v CAVE viacero možností ako sa dá interagovať so scénou. Od myši, cez klávesnicu, pákový ovládač (Joystick) alebo Gamepad, až po zariadenia ako Myo a OptiTrack. Keďže hra by sa mala čím viac podobať realite, najvhodnejším riešením nám prišlo použiť práve pákový ovládač (Joystick).

Pri ovládaní pákovým ovládačom je jednou z kritických častí práve to, akým spôsobom sa budú prenášať dáta z pákového ovládača. V CAVE už je pripravená aplikácia, ktorá slúži na čítanie dát z pákového ovládača a následne ich zapisuje do príslušných súborov, ktorých názvy a umiestnenie je konfigurovateľné priamo v tejto aplikácii. Napríklad pri stlačení tlačidla X na pákovom ovládači sa do súboru X.txt zapíše hodnota 1, ak tlačidlo nieje stlačené, zapíše sa tam hodnota 0. Výhodou použitia práve pákového ovládača je to, že pri pohyboch do strán sa do dedikovaných súborov nezapisujú takéto diskrétne hodnoty ako pri tlačidlách v podobe 1 alebo 0, ale spojité hodnoty v istom intervale. Napríklad -60000 v súbore HorizontalAxis.txt by znamenalo, že pákovým ovládačom sme pohli úplne doľava, 60000 by znamenalo opak, že sme ním pohli úplne doprava. Ak

by však išlo o jemnejšie pohyby, hodnoty by sa pohybovali niekde v rozmedzí intervalu -60000 a 60000. Na základe týchto hodnôt tým pádom vieme odhanúť aj intenzitu pohybu, či sa jedná o prudký alebo iba jemný pohyb do strany, prípadne nahor alebo nadol.

Tým, že tieto hodnoty sú zapisované do príslušných súborov, je možné ich následne čítať a na základe prečítaných hodnôt vykonávať v scéne príslušne namapované akcie.

Hru by bolo možné ovládať aj klávesnicou, avšak najväčšou nevýhodou je slabé vtiahnutie používateľa do deja a do virtuálnej scény.

2.7.9. Mierenie a streľba v hre

PH8 hovorí o tom, čo je cieľom hráča. Jeho úlohou je zostreľovanie objektov, ideálne by mal zostreľovať iba objekty typu cieľ. Avšak to, aké objekty budú reálne zostreľované, je na samotnom používateľovi. Z hľadiska implementácie je tu ale potrebné vyriešiť viacero problémov:

- Mierenie (ovládanie pohybov zbrane a ukazovateľ mierenia)
- Streľba (ovládanie streľby)
- Vyhodnotenie výsledkov streľby (či daný objekt, dron, bol alebo nebol zasiahnutý)

1. Mierenie

Hráčovi bude umožnené mieriť otáčaním sa doľava a doprava a zároveň natáčaním zbrane smerom nahor a nadol. Je nutné ale spomenúť, že pri pohybe doľava a doprava bude dochádzať k rotácii kamery, to z dôvodu, že ukazovateľ mierenia by mal byť vždy vycentrovaný na stredných obrazovkách, ktoré sa nachádzajú v CAVE pred používateľom. Pri pohyboch smerom nahor a nadol bude dochádzať k otáčaniu zbrane a v tomto prípade sa bude pohybovať iba samotný ukazovateľ mierenia a rotácia kamery ostane fixovaná. Dôvodom je neprirodzenosť rotácie kamery nahor a nadol, keďže v prostredí CAVE používateľ cíti, že nieje fyzicky nakláňaný napríklad pri pohybe hore, čo by mohlo spôsobiť nedôveryhodnosť a slabšie vžitie hráča do hry, prípadne by to uňho mohlo vyvolať pocity nevoľnosti. Pri analýze vizuálneho zobrazenia ukazovateľa mierenia sa ako najvhodnejšia možnosť osvedčila červená gulička zobrazená na predných obrazovkách.

Pri mierení bolo takisto potrebné vyriešiť otázku, aký rozhľad bude mať daný používateľ. Riešením je špecifikácia uhla rozhľadu, ktorý je zahrnutý ako parameter v internej konfigurácii, to znamená priamo v skripte. Hovorí, o akú hodnotu v uhloch je možné sa maximálne odchýliť (zrotovať) doprava prípadne doľava od počiatočnej polohy. Pri snahe prekročiť túto rotáciu je používateľ blokovaný a kamera sa ďalej nerotuje. Podobná technika sa osvedčila aj pri pohyboch

nahor a nadol, avšak keďže pri týchto pohyboch sa nejedná o rotáciu kamery, je pri dosiahnutí hraničnej rotácie zastavený pohyb ukazovateľa mierenia.

Ďalšou úlohou pri mierení bol spôsob indikovania, keď daný objekt bol zameraný. V takomto prípade sa okolo zameraného objektu zobrazí žltý indikátor.

2. Streľba

Je umožnená tlačidlom, či už na pákovom ovládači alebo klávesnici (v závislosti od typu ovládača) a po streľbe sa zobrazí indikácia v podobe žltého lúča. Problém, ktorý bolo potrebné riešiť súvisel s možným podvádzaním v hre, a to s neprerušenou streľbou, resp. so streľbou s veľmi vysokou frekvenciou. To je možné vyriešiť pridaním herného prvku dobíjanie zbrane, ktoré by trvalo istý čas. Tento čas by mohol byť konfigurovateľný napríklad v internej konfigurácii vo vnútri skriptu. Po každom výstrele by sa tak začal tento čas odpočítavať a počas tejto doby by hráčovi nebolo umožnené vystreliť.

3. Vyhodnotenie výsledkov streľby

Keďže SuperEngine neponúka niečo ako indikovanie, či dané 2 objekty sú alebo niesú v kolízii, bolo potrebné to doimplementovať. Bolo potrebné vymyslieť metódu, ktorá určí, či sa daný objekt nachádza v okolí priamky – v tomto prípade by priamka predstavovala akúsi pomyslenú dráhu strely, ktorá by mala pretínať aktuálnu pozíciu hráča a aktuálnu pozíciu ukazovateľa mierenia. Návrh metódy ako to správne určiť bol o čosi zložitejší, keďže bolo potrebné vyriešiť kolíziu v 3D priestore. Keďže sa jedná o zistenie kolízie priamky a objektu, nebolo to možné riešiť ako kolíziu 2 gulí, ktorá by bola jednoduchšia, avšak v tomto prípade veľmi nepresná. Návrh algoritmu a riešenia je teda nasledovný:

- Celá virtuálna scéna je kolmo premietnutá do dvoch 2D rovín – do roviny XY (pohľad zhora) a do roviny YZ (pohľad zľava).
- Daný pozorovaný objekt (cieľ alebo distraktor) je reprezentovaný jedným bodom (jeho stred, ktorý je závislý na tom, ako bol model namodelovaný a vyexportovaný. Ten by mal byť namodelovaný a vyexportovaný tak, aby sa približný stred modelu nachádzal na súradnici [0, 0, 0]).
- Je zadaný počiatočný bod priamky a uhly horizontálneho a vertikálneho otočenia.
- Následne je samostatne vyhodnotené, či sa bod (stred modelu) nachádza v okolí priamky pre oba priemety. Povolené okolie by bolo konfigurovateľné a mohol by to byť napríklad interný parameter, ktorý by bol istým spôsobom správne prispôsobený modelom – vo všeobecnosti je závislý na veľkosti modelu v scéne.

- Ak je splnené, že daný bod sa nachádza pri oboch kolmým priemetoch v okolí priamky, ide o kolíziu daného objektu reprezentovaného bodom s danou priamkou.

Vyhodnocovací algoritmus by tak iteroval všetkými objektami a ak by zistil, že daný objekt je v kolízii s priamkou, označil by ho príslušným žltým indikátorom a ukončil iterovanie. V algoritme je zahrnuté aj riešenie vzdialenosti objektu, ak by bol objekt príliš ďaleko, za hranicou povoleného dostrelu, bol by ignorovaný a k vyhodnocovaniu kolízie by v takomto prípade ani nedochádzalo. Výpočet vzájomnej vzdialenosť 2 objektov v 3D priestore je realizovaný pomocou Pytagorovej vety pre 3D priestor.

Ak by bol v čase streľby nejaký objekt v kolízii s priamkou (pomyselná priamka pretínajúca bod, na ktorom sa aktuálne nachádza používateľ a bod, kde sa nachádza ukazovateľ mierenia), tak zmizne zo scény, čo by znamenalo, že je zostrelený.

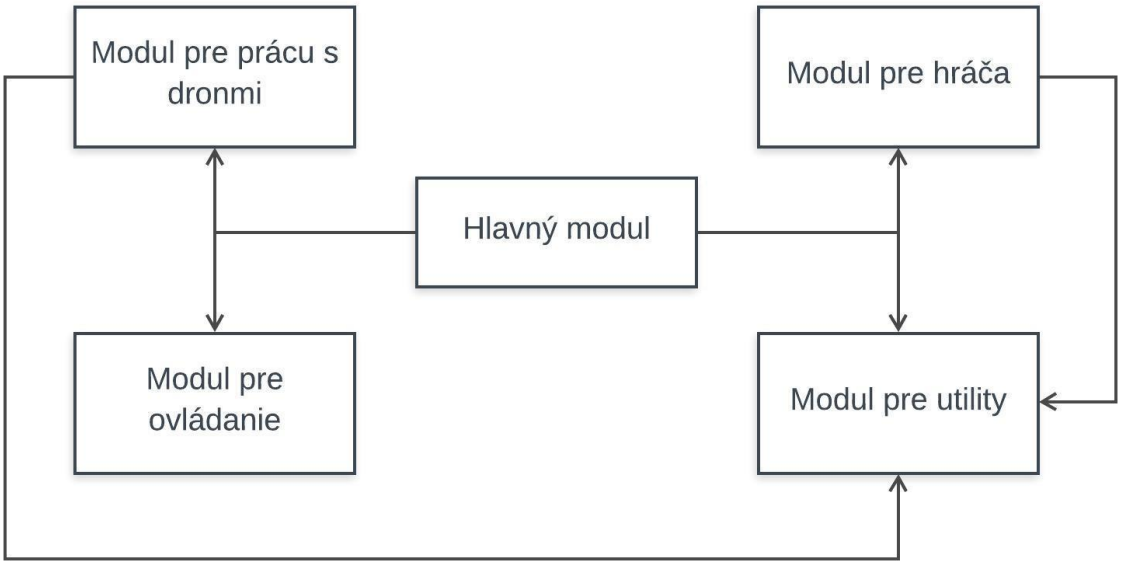
2.7.10. Náhodne vyskytujúca sa udalosť - ,,výpadok prúdu“

V PH6 zadávateľ požaduje takzvaný ,,výpadok prúdu“. Pri tejto požiadavke bolo potrebné vyriešiť, kedy má daný výpadok prúdu nastať, v akom časovom intervale a na akú dlhú dobu. Tieto hodnoty by mali byť konfigurovateľné zadávateľom, ktorý bude vedieť zvoliť, od ktorého do ktorého času výpadok môže nastať. Zadávať by to mohol napríklad v percentuálnej podobe, čo by znamenalo, že výpadok môže nastať napr. v 20 – 60 % epizódy. Dĺžka trvania tohoto výpadku by mohla byť takisto konfigurovateľná zadávateľom.

Pointou tejto udalosti je zmiasť a trochu zaskočiť používateľa hrajúceho hru. To by sa malo docieliť tým, že na istý čas zmizne zo scény všetko okrem pozadia a platformy (podložky).

3. Návrh hry

V tejto kapitole je na základe predošlej analýzy vypracovaný podrobný návrh hry. Na začiatku je navrhnutý scenár hry – aké životné cykly sa budú v hre odohrávať. Ďalej je konkrétne určené, aké časti bude pri implementácii potrebné namodelovať. Na obrázku nižšie je zobrazená vysoko-úrovňová architektúra systému, ktorý bude zhotovený na báze modulov, kde každý modul bude zodpovedný za riešenie istej časti a konkrétneho problému v hre. Na obrázku 4 je zobrazený základný návrh modulov hry spolu s vyznačenými asiociáciami medzi jednotlivými modulmi.



Obr. 4 Základný návrh modulov hry

3.1. Scenár hry

Na základe vykonanej analýzy prišlo ako najvhodnejšie riešenie pre scenár hry jeho rozdelenie do dvoch herných celkov, a to levelov a epizód.

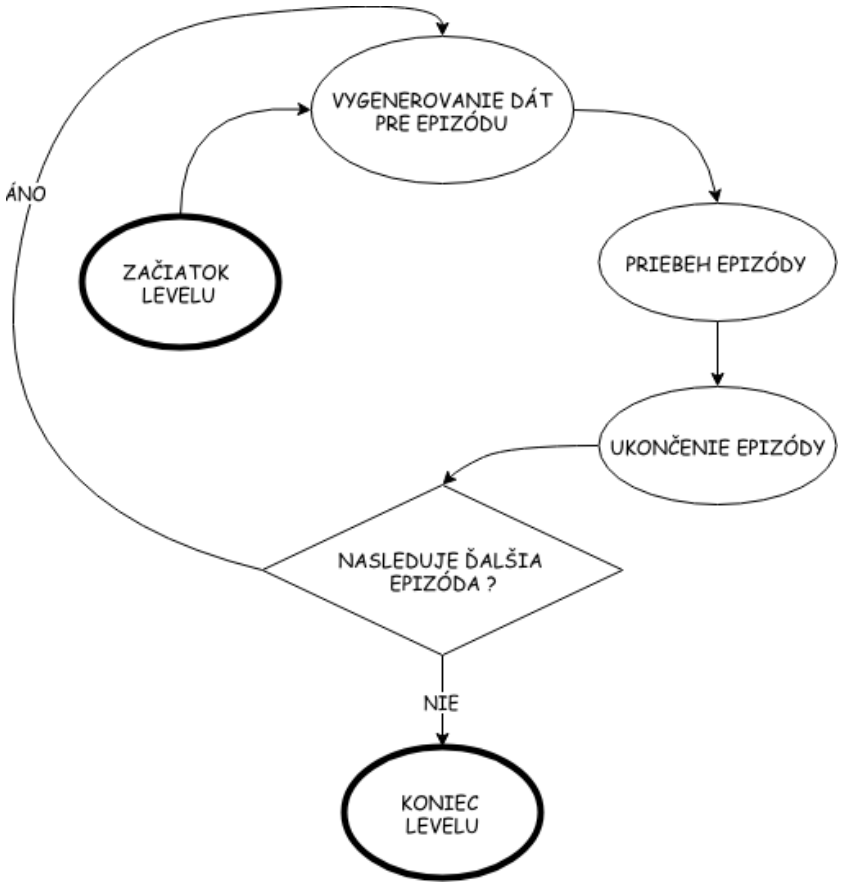
Level je logická časť hry, ktorá je samostatne konfigurovateľná a pozostáva z jednej alebo viacerých epizód. Epizódy sú časti levelu, ktoré budú predstavovať nálety dronov proti hráčovi.

Celá herná slučka tak bude predstavovať jeden level, v ktorom sa budú odohrávať jednotlivé epizódy. Životný cyklus hernej slučky je nasledovný:

1. Začiatok levelu
2. Vygenerovanie dát pre epizódu
3. Priebeh epizódy
4. Ukončenie epizódy

5. Čakanie na ďalšiu epizódu
6. Návrat do kroku 2
7. Koniec levelu

Na obrázku 5 je vidieť graficky znázornený priebeh životného cyklu jedného levelu.



Obr. 5 Znázornenie jednotlivých fáz životného cyklu hry

3.2. Opis jednotlivých fáz životného cyklu

V tejto kapitole sú detailne popísané jednotlivé fázy životného cyklu levelu, počínajúc začiatkom levelu (fáza 1) a končiac koncom levelu (fáza 7).

3.2.1. Začiatok levelu

Prvým krokom je získanie vstupnej konfigurácie pre daný level. Z analýzy Dominika Trojčáka v jeho práci s názvom Hra pre experimentálne posúdenie kognitívnych funkcií vo virtuálnej realite – konfigurácia hry a zber údajov, vyplynuli nasledujúce vstupné parametre potrebné pre konfiguráciu levelu:

- Dĺžka trvania levelu

- Počet epizód
- Dĺžka epizódy
- Počet objektov typu cieľ
- Odchýlka od počtu objektov typu cieľ
- Počet objektov typu distraktor
- Odchýlka od počtu objektov typu distraktor
- Percentuálny rozsah v epizóde, v ktorom môže nastať udalosť ,,výpadku prúdu“
- Pravdepodobnosť nastatia udalosti ,,výpadok prúdu“
- Možnosť zapnutia alebo vypnutia zobrazenia orientačného bodu v strede scény
- Dĺžka zobrazenia počiatočných indikátorov slúžiacich na rozlíšenie typu dronov

Na základe vstupnej konfigurácie sa pri začatí levelu vytvorí sada dronov. Táto sada dronov potrebuje byť dostatočne veľká, aby obsahovala dostatočný počet dronov pre daný level. Maximálny počet dronov v danom leveli je teda daný ako:

počet objektov typu cieľ + počet objektov typu distraktor + odýchlka od počtu objektov typu cieľ + odchýlka od počtu objektov typu distraktor

3.2.2. Vygenerovanie dát pre epizódu

V tejto fáze ide o určenie modelov pre objekty typu cieľ, to znamená, ktoré grafické modely môžu v danej epizóde reprezentovať cieľové objekty. Ostatné objekty môžu reprezentovať distraktorov.

- Vyrátanie počtu objektov typu cieľ v danej epizóde:

POČET CIEĽOVÝCH OBJEKTOV = POČET OBJEKTOV TYPU CIEĽ PRE DANÝ LEVEL ± ODCHÝLKA OD POČTU OBJEKTOV TYPU CIEĽ

- Vyrátanie počtu objektov typu distraktor v danej epizóde:

POČET DISTRAKTOR OBJEKTOV = POČET OBJEKTOV TYPU DISTRAKTOR PRE DANÝ LEVEL ± ODCHÝLKA OD POČTU OBJEKTOV TYPU DISTRAKTOR

- Získanie potrebného počtu cieľových a distraktor dronov zo sady dronov, ich poupravenie na konkrétne typy, keďže drony v sade dronov sú generické.
- Priradenie voľných trajektórií daným dronom, aby každý mal jedinečnú trajektóriu. Uloženie do poľa dronov vystúpujúcich v danej epizóde.
- Vyrátanie udalosti ,,výpadok prúdu“ – či má na základe pravdepodobnosti nastať, a ak áno, v ktorom čase má nastať a v ktorom skončiť.

- Nastavenie počiatočného a koncového času epizódy.
- Vyrátanie začiatočného a koncového času zobrazenia indikátorov nad dronmi.

3.2.3. Priebeh epizódy

Počas priebehu epizódy každý dron, ktorý sa nachádza v poli dronov pre danú epizódu vykonáva svoju metódu pre aktualizáciu stavu. V tejto metóde drona sa rieši jeho pohyb. Objekt hráča takisto vykonáva svoju metódu pre aktualizáciu stavu.

V priebehu epizódy sa na základe vopred predvypočítaných dát, ktoré sa vyrátali v predošlej fáze, vykonávajú aj jednotlivé akcie:

- Zobrazenie indikátorov
- Udalosť ,,výpadok prúdu“

3.2.4. Ukončenie epizódy

V tejto fáze sa vykonávajú nasledovné akcie:

- Zresetovanie trajektórií do pôvodného stavu.
- Zresetovanie predvypočítaných hodnôt pre epizódu, ako sú časy pre nastanie udalosti a ďalšie.
- Zresetovanie dronov, ktorý boli aktívni v predošlej epizóde do pôvodného stavu.

3.2.5. Čakanie na nasledujúcu epizódu

V tejto fáze sa vie hráč pohybovať, avšak nieje aktívny žiaden nálet. Plynie čakacia doba medzi epizódami, kedy sa hráč vie pripraviť na ďalší nálet.

3.2.6. Ukončenie levelu

V tejto časti dochádza k uloženiu štatistík z hry a k ukončeniu samotnej hry.

3.3. Návrh entít

Ako vyplýva z analýzy, je potrebné navrhnúť 2 entity - prvú pre drony a druhú pre hráča. Entita pre drony bude reprezentovať objekty typu cieľ a distraktor. Entita pre hráča bude reprezentovať hráča hrajúceho danú hru – používateľa.

3.3.1. Entita dron

Je reprezentovaná triedou *Drone*, ktorá je ale skomponovaná z viacerých podtried. Samotný dron sa teda skladá z jeho dizajnu, ktorý rieši jeho vizuálnu stránku, z jeho špecifikácie, ktorá rieši jeho vlastnosti a schopnosti ako rýchlosť a podobne, z trajektórie, po ktorej sa dron bude pohybovať. Ďalšie vlastnosti sú naviazane priamo na generickú triedu Drone, ako napríklad typ, čiže či sa jedná o typ target alebo distraktor. Na základe toho vieme rozlíšiť, či hráč zostrelil správneho alebo nesprávneho drona. Ďalšia vlastnosť je samotná grafická reprezentácia jeho indikátora, ktorá je zelenej farby, ak ide o distraktora a červenej farby ak ide o objekt typu cieľ, ktorý je potrebné zostreliť.

Akcie nad entitou dron:

- Vie sa reinicializovať, čiže sa vie zresetovať do predvoleného stavu.
- Vie sa pohybovať danou rýchlosťou po svojej trajektórii, ktorá mu je priradená.

3.3.2. Entita hráč

Entita hráč je skomponovaná z viacerých prvkov, ktoré súvisia s hráčom resp. sú priamo ovládané a ovplyvňované hráčom. Je teda zložená z:

- Platformy, na ktorej sa hráč nachádza a ktorá sa vie otáčať.
- Zbrane, ktorú hráč vlastní a je schopný z nej strieľať.
- Z ukazovateľa mierenia, ktorý sa na základe ovládania vie pohybovať v scéne a je ním možné zamieravať dronov.
- Z indikátora nabitia zbrane, ktorý sa nachádza priamo nad ukazovateľom mierenia a je zobrazený v prípade, že je zbraň nabitá a pripravená na streľbu.

Akcie nad entitou hráča:

- Otáčanie sa pomocou platformy smerom doprava a doľava
- Natáčanie zbrane smerom hore a dole
- Streľba zo zbrane
- Dobíjanie zbrane – táto akcia je automatická

3.4. Návrh ovládacích prvkov

Z analýzy vyplýva, že je postačujúce mať ovládací prvok v podobe pákového ovládača. Pre lepšiu testovateľnosť boli však navrhnuté 3 možné typy ovládania, a to ovládanie pomocou klávesnice, pákového ovládača a taktiež pomocou Gamepadu.

Akcie, ktoré je potrebné riadiť a ovládať sú nasledovné:

- Otáčanie doľava
- Otáčanie doprava
- Natáčanie zbrane smerom hore
- Natáčanie zbrane smerom dole
- Streľba

Navrhnuté sú 3 triedy pre jednotlivé typy vstupných zariadení:

- Klávesnica – `KeyboardController`
- Pákový ovládač (Joystick) – `JoystickController`
- Gamepad – `GamepadController`

Medzi jednotlivými vstupnými zariadeniami sa dá prepínať pomocou interného konfiguračného parametru.

3.4.1. Ovládanie pomocou klávesnice

Mapovanie tlačidiel pre jednotlivé akcie je nasledovné a je zobrazené v nasledujúcej tabuľke.

KLÁVESA	AKCIA
A	Otáčanie doľava
D	Otáčanie doprava
W	Natáčanie zbrane smerom hore
S	Natáčanie zbrane smerom dole
Šípka hore	Streľba

Tab. 1 Mapovanie ovládania pre klávesnicu

3.4.2. Ovládanie pomocou pákového ovládača

Mapovanie tlačidiel pre jednotlivé akcie je nasledovné a je zobrazené v nasledujúcej tabuľke.

POHYB PÁKOVÝM OVLÁDAČOM	TLAČIDLO	AKCIA
Vľavo	-	Otáčanie doľava
Vpravo	-	Otáčanie doprava
Od seba	-	Natáčanie zbrane smerom hore
K sebe	-	Natáčanie zbrane smerom dole
	HORE	Streľba

Tab. 2 Mapovanie ovládania pre pákový ovládač

Pre tento typ ovládača je špecifická ešte jedna vlastnosť, a to, že pri otáčaní do strán resp. pri mierení smerom nahor alebo nadol sa zohľadňuje ešte aj intenzita vychýlenia páky. V praxi to znamená, že čím viac je páka vychýlená napr. vľavo zo základnej polohy, tým rýchlejšie sa daný hráč otáča smerom vľavo. Rovnaká logika platí aj pre ostatné smery vychýlenia páky. Sila vychýlenia pre otáčanie vľavo resp. vpravo je vyrátavaná nasledujúcim vzťahom.

```
powerX = (($axisX - 32767).abs / 32767) * ($joystickPowerX / 100)
```

Význam premenných použitých vo vzorci:

- **`$axisX`** – hodnota vychýlenia v smere osi X. Prečítaná zo súboru filetextX.rb.
- **`$joystickPowerX`** – citlivosť pákového ovládača pri otáčaní vľavo resp. vpravo.
- **`powerX`** – výsledná hodnota označujúca intenzitu otáčania. Čím väčšia je táto hodnota, tým rýchlejšie bude otáčanie.

Inak povedané, rýchlosť otáčania je priamo úmerná intenzite vychýlenia pákového ovládača z jeho základnej vertikálnej polohy.

Podobný princíp je aplikovaný pri mierení smerom nahor resp. nadol, akurát sa namiesto vychýlenia od vertikálnej osi kontroluje intenzita vychýlenia pákového ovládača od jeho základnej horizontálnej osi.

3.4.3. Ovládanie pomocou Gamepadu

Mapovanie tlačidiel pre jednotlivé akcie je nasledovné a je zobrazené v nasledujúcej tabuľke.

POHYB ĽAVÝM ANALÓGOM	TLAČIDLO	AKCIA
Vľavo	-	Otáčanie doľava
Vpravo	-	Otáčanie doprava
Hore	-	Natáčanie zbrane smerom hore
Dole	-	Natáčanie zbrane smerom dole
	R2	Streľba

Tab. 3 Mapovanie ovládania pre Gamepad

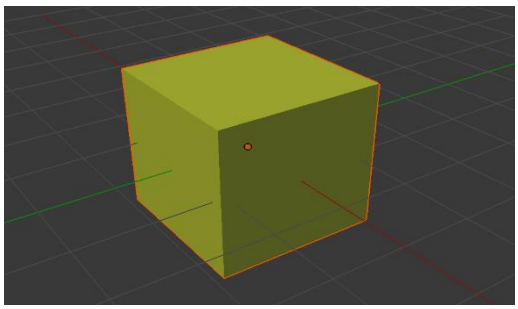
3.5. Návrh grafických modelov reprezentujúcich drony

Zo záveru analýzy vyplýva, že grafické modely pre drony by mali mať jednoduchý a čím viac symetrický tvar. Zároveň by mali byť žltej farby. Je potrebné navrhnúť minimálne 7 rôznych grafických reprezentácií pre drony.

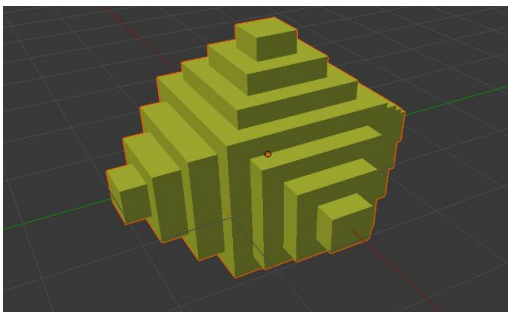
Sú navrhnuté nasledujúce grafické reprezentácie:

- Kocka
- Navrstvené hranoly
- Valec s toroidom
- Horizontálny obojstranný kužeľ
- Vertikálny obojstranný kužeľ
- Hviezdica s toroidom
- Sférický dron s výstupkami

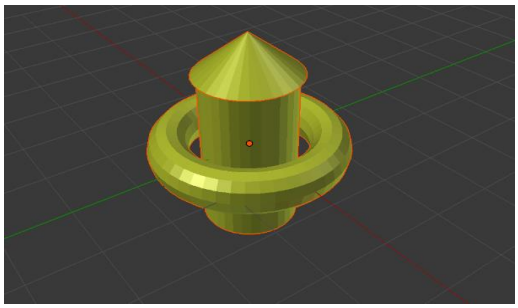
Jednotlivé grafické modely sú znázornené na nasledujúcich obrázkoch:



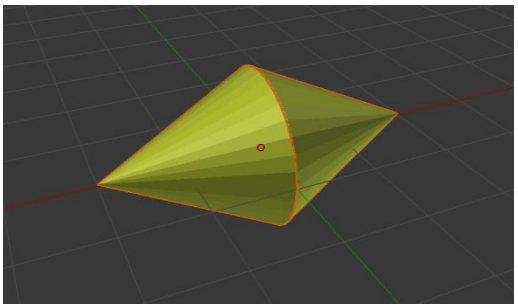
Obr. 6 Model dronu – Kocka



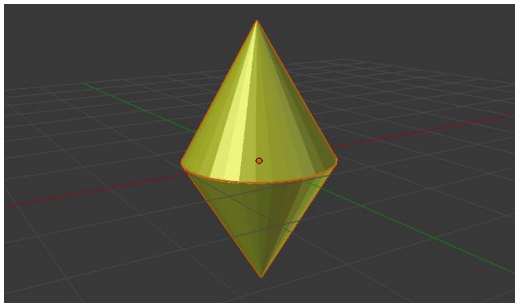
Obr. 7 Model dronu – Navrstvené hranoly



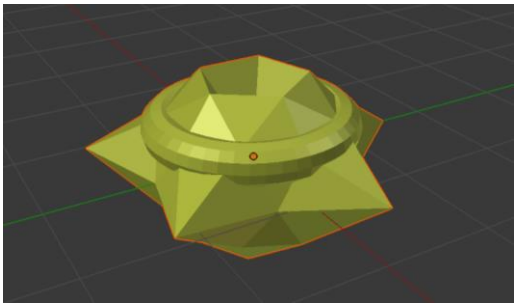
Obr. 8 Model dronu – Valec s toroidom



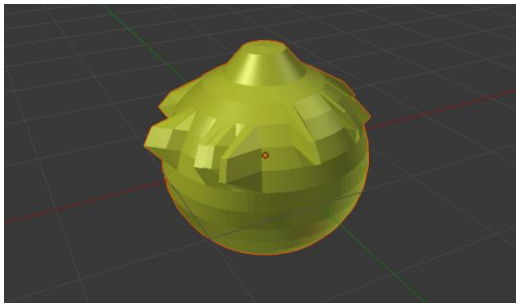
Obr. 9 Model dronu – Horizontálny obojstranný kužeľ



Obr. 10 Model dronu – Vertikálny obojstranný kužeľ



Obr. 11 Model dronu – Hviezdica s toroidom



Obr. 12 Model dronu – Sférický dron s výstupkami

3.6. Návrh zvukového modulu

Táto podkapitola sa zaoberá návrhom zvukového modulu. Bolo potrebné navrhnúť zvuky pre jednotlivé akcie odohrávajúce sa v hre:

- Streľba
- Výbuch drona po jeho zostrelení
- Hudba v pozadí

Samotný zvukový modul sa bude starať o načítanie potrebných zvukov pri začatí hry. Následne bude možné naviazať prehrávanie jednotlivých zvukov na príslušné akcie odohrávajúce sa v hre. Zvuky sú uložené vo formáte OGG, ktorý je jedným z formátov podporovaných SuperEnginom.

3.7. Návrh modulu pre riešenie priesečníkov

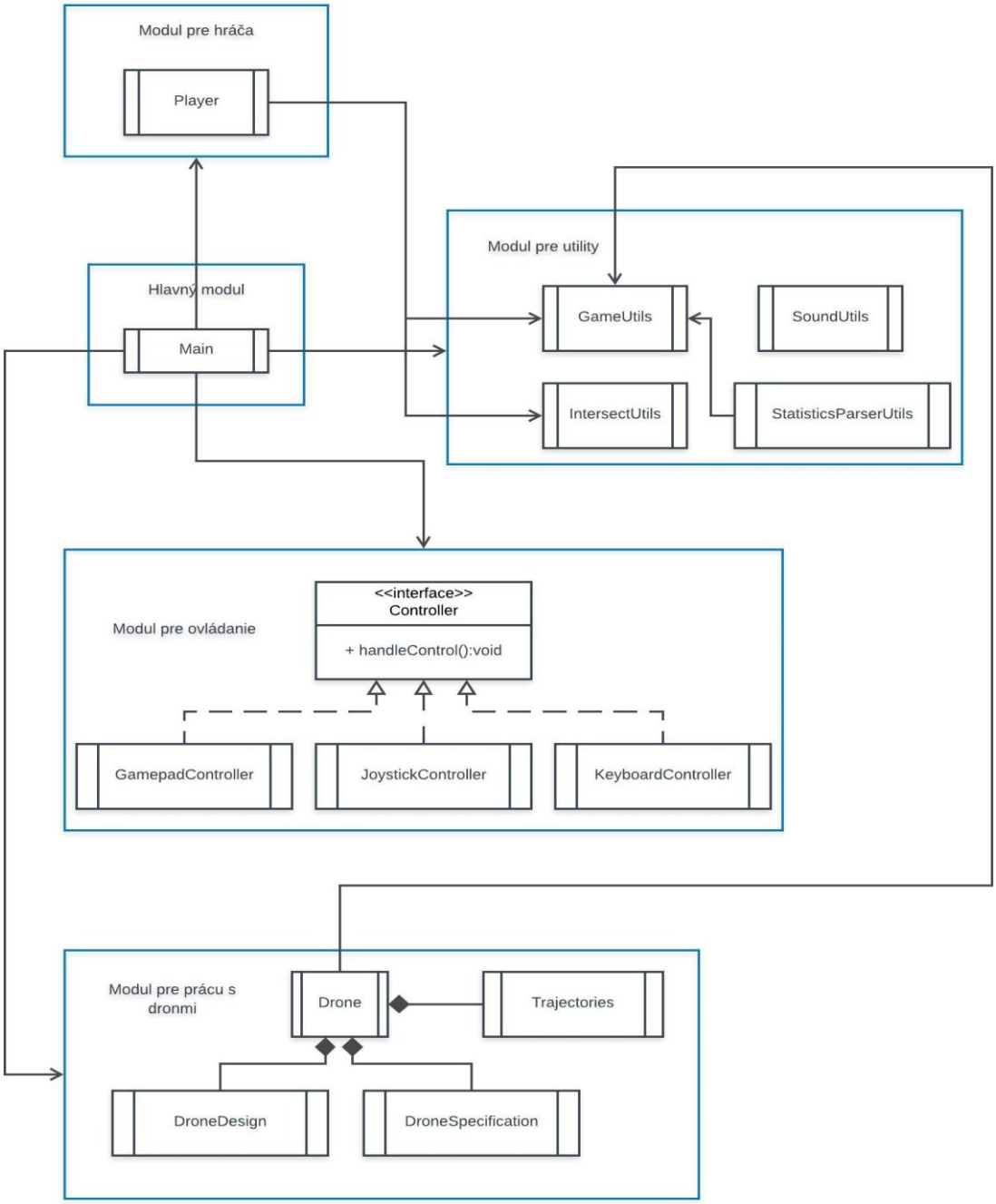
Tento modul bude slúžiť na mierenie a vyhodnocovanie výsledkov streľby. Je navrhnutá trieda *IntersectUtils*, ktorá bude mať 1 základnú metódu *intersect* s potrebnými argumentami, a ktorá vráti hodnotu *true* v prípade, že objekt sa nachádza v blízkosti úsečky, ktorá je zadaná na vstupe. V prípade, že objekt sa nenachádza v blízkosti tejto úsečky, metóda vráti hodnotu *false*. Samotné vyhodnocovanie, či nejaký objekt v 3D priestore je v blízkej oblasti pomyslenej priamky je možné vyhodnotiť na základe dvoch kolmých priemetov scény do 2D rovín. Konkrétne pôjde o kolmé priemety do roviny XY, čo predstavuje kolmý priemet v smere zhora nadol, a do roviny YZ, čo predstavuje kolmý priemet v smere zľava doprava. Na základe týchto priemetov sa osobitne vyhodnotí, či sa bod náchadza v okolí priamky, najprv pre prvý a následne pre druhý priemet. Na to bude slúžiť pomocná metóda, ktorá vyhodnotí, či sa bod nachádza v okolí priamky v 2D rovine. Na to, aby bolo možné povedať, že sa bod nachádza v okolí priamky v 3D priestore je potrebné splniť nasledujúce tvrdenia:

1. Bod sa nachádza v okolí priamky pri kolmom priemete scény do roviny XY
2. Bod sa nachádza v okolí priamky pri kolmom priemete scény do roviny YZ

Ak jedno z týchto tvrdení nieje pravdivé, je možné povedať, že bod sa nenachádza v okolí priamky v 3D priestore.

4. Implementácia hry

Implementácia hry je vytvorená podľa vyššie špecifikovaného návrhu, kde celý systém je založený na báze modulov. Každý modul obsahuje práve jednu alebo viacero tried, ktoré spadajú pod jeho kompetenciu. Na obrázku 13 je zobrazený diagram tried spolu s vyznačením, do akého modulu dané triedy spadajú.



Obr. 13 Diagram tried hry so znázornením príslušných modulov

4.1. Moduly

Hra je logicky rozčlenená do celkov, čomu zodpovedá adresárová štruktúra projektu. Každý podpriečinok predstavuje akúsi samostatnú logickú časť, modul, ktorý rieši špecifický problém. V hre boli vytvorené nasledovné moduly:

- Ovládače – controllers
- Drony – drones
- Utility – utils
- Hráč – player
- Hlavný modul - main

V nasledujúcich podkapitolách sú detailne popísané všetky z týchto modulov – ich význam a aj triedy, z ktorých tieto moduly pozostávajú a sú v hre implementované.

4.1.1. Ovládače

V tomto module sú implementované jednotlivé ovládače, ktoré slúžia používateľovi ako prostriedok na ovládanie hry v prostredí CAVE. Momentálne je implementovaná podpora pre 3 typy ovládačov, a to pre klávesnicu, pákový ovládač a Gamepad. Všetky 3 ovládače implementujú rovnaké rozhranie `Controller` s jednou bezparametrickou metódou `handleControl`, vďaka čomu je možné ich v hre jednoducho prepínať pomocou konfigurácie. Pre každý z týchto ovládačov je vytvorená samostatná trieda, v ktorej je implementovaná logika pre daný ovládač. V nasledujúcich podkapitolách bude konkrétnejšie popísaná implementácia pre jednotlivé typy ovládačov.

4.1.1.1. Ovládač pre Gamepad

Logika pre tento ovládač je implementovaná v triede `GamepadController`, ktorá sa nachádza v súbore `gamepadController.rb`. Spôsob, akým sa vyhodnocujú dáta prichádzajúce z ovládača je nasledovný. Dáta z ovládača sa čítajú pomocou externej utility, ktorá už je dostupná v prostredí CAVE a zapisujú sa na disk do presne špecifikovaných súborov. Pre každé tlačidlo, ktoré sa nachádza na gamepade je dedikovaný osobitný, názvom špecifický súbor na disku. Pri analógových tlačidlách je dedikovaný separátny súbor pre každý zo 4 smerov pohybu. Napríklad pre pohyb vľavo ľavým analógovým tlačidlom je vyhradený súbor `filetextX.rb`, do ktorého sa zapisuje hodnota vychýlenia. V nasledujúcej tabuľke (Tab. 4) sú zobrazené všetky používané mapovacie súbory pre gamepad.

TLAČIDLO alebo POHYB	SÚBOR
Ľavý analóg – vľavo	filetextX.rb
Ľavý analóg – vpravo	filetextX.rb
Ľavý analóg – hore	filetextY.rb
Ľavý analóg – dole	filetextY.rb
Tlačidlo R2	btt6.rb

Tab. 4 Mapovanie výstupov do špecifických súborov pre ovládač pre Gamepad

Následne sa vstupné dáta, ktoré sa prečítajú z príslušných súborov vyhodnocujú, a na základe toho sa vykonávajú príslušné akcie v hre. To či sa daná akcia v hre vykoná alebo nevykoná je zobrazené v pravdivostnej tabuľke nižšie (Tab. 5).

AKCIA	SÚBOR	PODMIENKA PRE VYKONANIE AKCIE
Otočenie vľavo	filetextX.rb	hodnota > -1 && hodnota < 30000
Otočenie vpravo	filetextX.rb	hodnota > 45000 && hodnota < 80000
Mierenie nahor	filetextY.rb	hodnota > 45000 && hodnota < 80000
Mierenie nadol	filetextY.rb	hodnota > -1 && hodnota < 30000
Streľba	btt6.rb	hodnota > 0

Tab. 5 Vyhodnotenie vykonania akcií v hre pre ovládač pre Gamepad

4.1.1.2. Ovládač pre pákový ovládač

Logika pre tento ovládač je implementovaná v triede `JoystickController`, ktorá sa nachádza v súbore `joystickController.rb`. Dáta z ovládača sú načítavané pomocou externej utility, podobne ako pri Gamepad ovládači. V nasledujúcej tabuľke sú zobrazené všetky používané mapovacie súbory pre pákový ovládač.

TLAČIDLO alebo POHYB	SÚBOR
Pohyb joystickom vľavo	filetextX.rb
Pohyb joystickom vpravo	filetextX.rb
Pohyb joystickom od seba	filetextY.rb
Pohyb joystickom k sebe	filetextY.rb
Zadné tlačidlo	btt1.rb

Tab. 6 Mapovanie výstupov do špecifických súborov pre ovládač pre pákový ovládač

Rovnako ako pri ovládači pre Gamepad, je to, či sa daná akcia v hre vykoná alebo nevykoná zobrazené v pravdivostnej tabuľke nižšie.

AKCIA	SÚBOR	PODMIENKA PRE VYKONANIE AKCIE
Otočenie vľavo	filetextX.rb	hodnota < 27000
Otočenie vpravo	filetextX.rb	hodnota > 38000
Mierenie nahor	filetextY.rb	hodnota < 27000
Mierenie nadol	filetextY.rb	hodnota > 38000
Streľba	btt1.rb	hodnota > 0

Tab. 7 Vyhodnotenie vykonania akcií v hre pre ovládač pre pákový ovládač

4.1.1.3. Ovládač pre klávesnicu

Logika pre tento ovládač je implementovaná v triede `KeyboardController`, ktorá sa nachádza v súbore `keyboardController.rb`. Pre čítanie dát z klávesnice má podporu priamo SuperEngine. Referenciu pre vstupnú klávesnicu si vieme získať nasledujúcim spôsobom.

```
$dev_keyboard = ENGINE::GetInputDevice("Keyboard0")
```

Následne volaním metódy `GetButtonData` nad týmto objektom získame pole, v ktorom hodnota na konkrétnom indexe reprezentuje stav pre konkrétne tlačidlo klávesnice. Táto hodnota môže byť *0*, v prípade, že tlačidlo nebolo v čase volania metódy `GetButtonData` stlačené alebo *1*, ak tlačidlo stlačené bolo. V nasledujúcej tabuľke sú vypísané indexy v danom poli pre konkrétene klávesy použité v hre.

KLÁVESA	INDEX
W	17
S	31
A	30
D	32
Šípka hore	72

Tab. 8 Indexy v poli pre jednotlivé klávesy

Rovnako ako pri predošlých ovládačoch je nižšie uvedená pravdivostná tabuľka na zistenie toho, či sa daná akcia v hre vykoná alebo nevykoná.

AKCIA	KLÁVESA	PODMIENKA PRE VYKONANIE AKCIE
Otočenie vľavo	A	$dev_keyboard[30] = 1
Otočenie vpravo	D	$dev_keyboard[32] = 1
Mierenie nahor	W	$dev_keyboard[17] = 1
Mierenie nadol	S	$dev_keyboard[31] = 1
Streľba	Šípka hore	$dev_keyboard[72] = 1

Tab. 9 Vyhodnotenie vykonania akcií v hre pre ovládač pre klávesnicu

4.1.2. Drony

V tomto moduli sú implementované triedy potrebné pre prácu s dronmi. Samotná entita dron je rozčlenená resp. pozostáva z 3 častí. Základná logika je implementovaná v triede `Drone`. Požiadavky pre design, čize pre grafickú reprezentáciu drona sú imlpementované v triede `DroneDesign`. Vlastnosti drona, akými je napríklad rýchlosť sú oddelené a definované v samostatnej triede `DroneSpecification`. V tomto module sa taktiež nachádza trieda `DroneModelPool`, ktorá slúži ako továreň pre predvytvorené modely dronov. Všetky triedy tohoto modulu sú detailnejšie popísané v podkapitolách nižšie.

4.1.2.1. Grafický dizajn drona

Ako je spomenuté vyššie, ide o vizuálnu stránku drona, ktorá je implementovaná v triede `DroneDesign`, ktorá sa nachádza v súbore `droneDesign.rb`. Konštruktor obsahuje 4 povinné parametre:

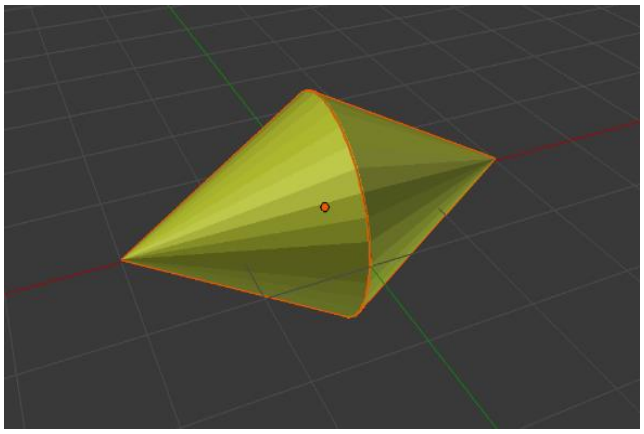
- **`baseDroneModel`** – základný grafický model, ktorým je dron reprezentovaný. Ide o model, ktorý je podporovaný SuperEnginom. Model je možné vytvoriť z grafického 3DS modelu nasledovným spôsobom:

```
ENGINE::LoadModel("./models/newDroneModels/bestDrone.3ds")
```

Následne je tento model prichádzajúci ako vstupný parameter v konštruktore vyklonovaný nasledovným spôsobom:

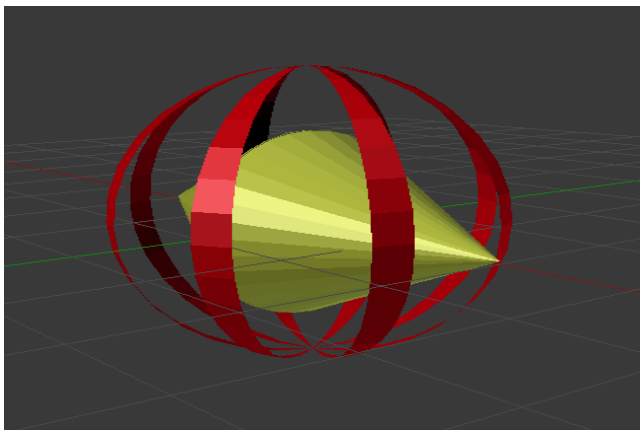
```
ENGINE::CloneModel(baseDroneModel)
```

Na obrázku 14 je zobrazený príklad základného modelu pre drona.



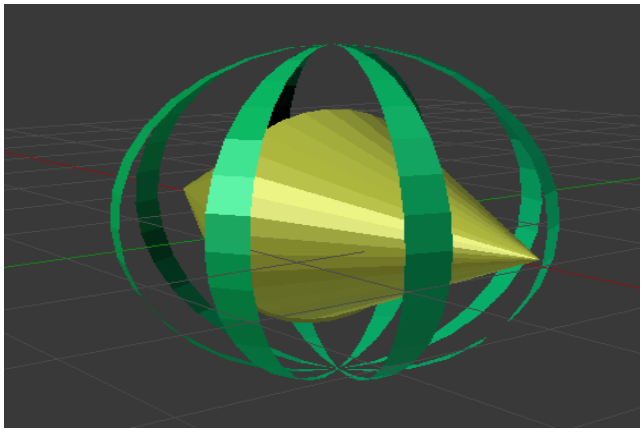
Obr. 14 Príklad základného modelu pre drona

- **`targetIndicator`** – ide o grafický model pre označenie, ktoré sa nad dronom objaví na začiatku hry v prípade, že je typu cieľ. V prípade, že dron je typu distraktor, tento grafický model nebude použitý, avšak kvôli znovupoužiteľnosti dronov a kvôli tomu, aby daný dron mohol byť použitý aj ako cieľ a inokedy aj ak distraktor sa každej grafickej reprezentácii priradzuje cieľ aj distraktor indikátor. Na obrázku 15 je zobrazený základný model drona spolu s inicializačným indikátorom pre objekt typu cieľ.



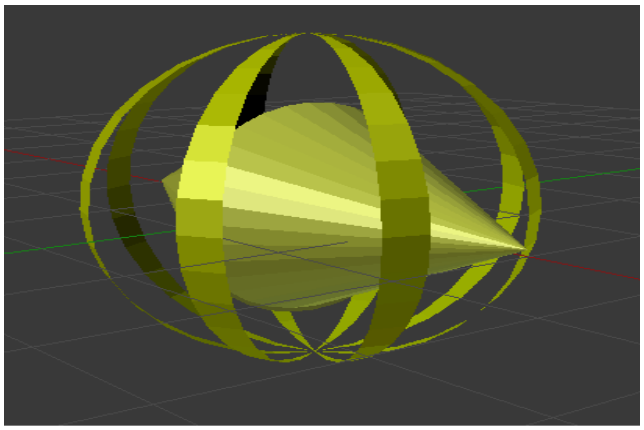
Obr. 15 Základný model drona spolu s indikátorom objektu typu cieľ

- **`distractorIndicator`** – ide o grafický model pre označenie, ktoré sa nad dronom objaví na začiatku hry v prípade, že je typu distraktor. Rovnako ako pri základnom modeli drona, tak aj pri cieľ a distraktor indikátoroch je potrebné vytvoriť SuperEngine model z 3ds modelu iba raz, následne sa už do konštruktora iba posiela referencia a model sa automaticky vyklonuje pomocou metódy `CloneModel` ako bolo ukázané vyššie. Na obrázku 16 je zobrazený základný model drona spolu s inicializačným indikátorom pre objekt typu distraktor.



Obr. 16 Základný model drona spolu s indikátorom objektu typu distraktor

- **droneMarker** – ide o grafický model pre označenie, ktoré sa nad dronom objaví v prípade, že je zameraný. Slúži ako indikátor pre správne rozpoznanie pre koncového hráča, aby vedel, že na daného drona má namierené. Na obrázku 17 je zobrazený základný model drona spolu s indikátorom, ktorý označuje, že na daného drona je aktuálne zamierené.



Obr. 17 Základný model drona spolu s indikátorom zamierenia

4.1.2.2. Špecifikácia vlastností drona

Táto časť sa zaoberá vlastnosťami samotného drona, ako sú rýchlosť a iné. Trieda `DroneSpecification`, ktorá to definuje je implementovaná v súbore `droneSpecification.rb`. Ide o dátovú triedu, ktorá obsahuje iba hodnoty samotných vlastností, neobsahuje teda žiadnu logiku. Slúži iba ako kontajner pre uloženie vlastností daného drona. Táto trieda bola navrhnutá generický a na princípe znovupoužiteľnosti, čize obsahuje veľa premenných definujúcich vlastnosti drona, z ktorých nie všetky sa pri momentálnom scenári hry používajú. Všetky definovateľné a použiteľné vlastnosti sú tieto:

- **timeOfFirstOccurence** – čas, v ktorom sa má daný dron zobraziť od jeho spustenia.
- **timeOfLastOccurence** - čas, v ktorom sa má daný dron skryť od jeho spustenia. Spolu s predchádzajúcim atribútom `timeOfFirstOccurence` takto vieme definovať aktívnu dobu drona.
- **speedInit** – definuje začiatočnú rýchlosť, akou sa bude daný dron pohybovať po svojej trajektórii. Táto rýchlosť môže byť neskôr menená definovaním atribútu `speedChange`.
- **speedMin** – minimálna rýchlosť, ktorou sa dron môže pohybovať. Tento atribút má zmysel pri zadaní zápornej hodnoty pre atribút `speedChange`. V tom prípade bude

rýchlosť drona postupne klesať s pribúdajúcimi behmi hry, avšak maximálne klesne na hodnotu definovanú práve týmto atribútom.

- **speedMax** – maximálna rýchlosť, ktorou sa dron môže pohybovať. Význam tohoto atribútu je podobný ako pri `speedMin` atribúte. Avšak tento sa aplikuje pri použití kladnej hodnoty ako `speedChange`.
- **speedChange** – definuje zmenu rýchlosti po jednotlivých behoch hry. Každým behom sa aktuálna rýchlosť drona, na začiatku `speedInit`, zráta s touto hodnotou a definuje tak novú aktuálnu rýchlosť drona. V prípade, že je táto hodnota záporná, dron bude s pribudajúcimi behmi spomaľovať, v prípade kladnej hodnoty bude zrýchľovať. Pri zadaní hodnoty 0 svoju rýchlosť nebude počas jednotlivých behov meniť.
- **towerDecreasePowerInit** – definuje počiatočný vplyv drona na vežu, z ktorej strieľa daný hráč, teda, koľko energie daný dron pridá resp. uberie, v prípade kladnej hodnoty, veži, z ktorej hráč strieľa na drony, počas streľby. Táto hodnota, rovnako ako tomu bolo aj pri rýchlosti, sa môže s pribúdajúcimi behmi meniť definovaním atribútu `towerDecreasePowerChange`.
- **towerDecreasePowerMin** – minimálna hodnota energie, ktorou môže daný dron ovplyvniť vežu pri streľbe. Platí podobný princíp ako pri analogických atribútoch pre rýchlosť.
- **towerDecreasePowerMax** – maximálna hodnota energie, ktorou môže daný dron ovplyvniť vežu pri streľbe.
- **towerDecreasePowerChange** – definuje zmenu vplyvu na energiu po jednotlivých behoch hry. Keďže sa jedná prioritne o úbytok energie, podľa toho je aj daný atribút pomenovaný, tak v prípade, že je táto hodnota kladná, dron bude s pribudajúcimi behmi uberať viac a viac energie, v prípade zápornej hodnoty bude naopak pridávať veži čoraz viac energie. Pri zadaní hodnoty 0 sa jeho vplyv nebude počas jednotlivých behov meniť a bude sa uplatňovať hodnota definovaná v `towerDecreasePowerInit`.
- **towerDecreaseProbabilityInit** – definuje počiatočnú pravdepodobnosť zásahu veže dronom, ktorý naňho strieľa. Zadáva sa hodnota v percentách od 0 po 100. Čím vyššia pravdepdobnosť, tým je väčšia šanca, že daný dron pri streľbe pridá resp. uberie veži energiu.
- **towerDecreaseProbabilityMin** – minimálna hodnota pravdepodobnosti zásahu veže pre daného drona.

- **towerDecreaseProbabilityMax** – maximálna hodnota pravdepodobnosti zásahu veže pre daného drona.
- **towerDecreaseProbabilityChange** – definuje zmenu pravdepodobnosti po jednotlivých behoch.
- **towerDecreaseFrequencyInit** – definuje počiatočnú frekvenciu streľby. Koľko krát daný dron vystrelí za minútu.
- **towerDecreaseFrequencyMin** – minimálna hodnota frekvencie streľby.
- **towerDecreaseFrequencyMax** – maximálna hodnota frekvencie streľby.
- **towerDecreaseFrequencyChange** – definuje zmenu frekvencie streľby po jednotlivých behoch.
- **cityOneLifeChangeInit** – definuje počiatočný vplyv drona na dané mesto, ktoré sa nachádza za nami, teda, koľko energie daný dron pridá resp. uberie, v prípade kladnej hodnoty, mestu, ktoré sa nachádza za nami, pri prekročení pomyselnej hranice. Narozdiel od streľby, táto hodnota vplýva na mesto maximálne raz za daný beh pre daného drona, práve po prekročení pomyselnej hranice mesta.
- **cityOneLifeChangeMin** – minimálna hodnota ovplyvnenej energie mesta.
- **cityOneLifeChangeMax** – maximálna hodnota ovplyvnenej energie mesta.
- **cityOneLifeChangeChange** – definuje zmeny hodnoty energie, ktorou dron ovplyvní mesto pri prekročení pomyselnej hranice.
- **towerOneLifeChangeInit** – definuje počiatočný vplyv drona na hráčovu vežu, teda, koľko energie daný dron pridá resp. uberie, v prípade kladnej hodnoty, veži, ktoré sa nachádza za nami, pri prekročení pomyselnej hranice. Narozdiel od streľby, táto hodnota vplýva na mesto maximálne raz za daný beh pre daného drona, práve po prekročení pomyselnej hranice veže.
- **towerOneLifeChangeMin** – minimálna hodnota energie, ktorou môže daný dron ovplyvniť vežu pri prekročení pomyselnej hranice.
- **towerOneLifeChangeMax** – maximálna hodnota energie, ktorou môže daný dron ovplyvniť vežu pri prekročení pomyselnej hranice.
- **towerOneLifeChangeChange** – definuje zmeny hodnoty energie, ktorou ovplyvní vežu pri prekročení pomyselnej hranice.
- **oneTimeEventProbabilityInit** – definuje počiatočnú pravdepodobnosť nastatia istého náhodného javu počas behu. V prípade scenáru našej hry ide o zatmenie, resp. výpadok prúdu.

- **oneTimeEventProbabilityMin** – minimálna hodnota pravdepodobnosti nastatia náhodnej udalosti.
- **oneTimeEventProbabilityMax** – maximálna hodnota pravdepodobnosti nastatia náhodnej udalosti.
- **oneTimeEventProbabilityChange** – definuje zmenu pravdepodobnosti nastatia náhodnej udalosti po jednotlivých behoch.
- **respawnTime** – čas trvania jedného behu pre daného drona v sekundách. Po uplynutí tohoto času sa dron vráti na svoju počiatočnú polohu na trajektórii a pohybuje sa od začiatku. Práve pri znovuvytvorení drona dochádza k prepočitávaniu všetkých vyššie definovaných atribútov, ktoré sa zmenia o ich hodnotu zmeny.

4.1.2.3. Trajektórie

Implementácia trajektórií sa nachádza v triede `Trajectories` v súbore `trajectories.rb`. Trajektórie sú definované ako polia, ktoré sú vo formáte podporovanom lineárnym interpolátorom v SuperEngine. Ide o dvojrozmerné pole, kde prvý rozmer určuje obtiažnosť, a druhý rozmer sú samotné trajektórie. Je možné to demonštrovať na nasledujúcom príklade.

```
@trajectories[0][4] = [0, 95.0, 1037.0, 40.0, 10, 864.25, 100.0, 10.0]
```

Na príklade vyššie je definovaná trajektória. Prvý rozmer v poli, číslo 0, ktoré je vyznačené hrubším písmom, definuje obtiažnosť, pre ktorú je trajektória definovaná, v našom prípade ide o obtiažnosť 0 – ľahká obtiažnosť. Ďalší rozmer v poli, číslo 4, taktiež zvýraznené hrubším písmom, definuje poradie trajektórie, v našom prípade teda definujeme 5. trajektóriu (indexovanie od 0) pre ľahkú obtiažnosť.

Táto trieda sa taktiež stará o priraďovanie trajektórií dronom. Dron jednoducho zavolá metódu `getFreeTrajectory(difficulty)`, a tá mu vráti voľnú trajektóriu, ktorú môže použiť. V triede je teda zabezpečený mechanizmus, ktorý si pamätá, ktoré trajektórie sú už použité a v prípade, že dron požiada o trajektóriu, vráti mu to iba voľnú trajektóriu. Tým je zabezpečené to, že v rovnakom čase nebudú drony zdieľať rovnakú trajektóriu, čo by mohlo spôsobovať kolízie. Vstupným parametrom do tejto metódy je difficulty, čiže obtiažnosť, pre ktorú danú trajektóriu chceme. Tento parameter nieje povinný a v prípade jeho absencie pri volaní sa použije prednastavená hodnota 0 – ľahká obtiažnosť.

V tejto triede je takisto implementovaná metóda `resetTrajectories`, ktorá sa volá na konci epizódy, aby sa všetky trajektórie znovu sprístupnili. V tejto metóde sa iba jednoducho označí každá trajektória ako nepoužitá.

4.1.2.4. Logika drona

Táto časť je z pohľadu drona najdôležitejšia a taktiež najkomplexnejšia, keďže spája grafický dizajn drona a špecifikáciu vlastností drona. Takisto pridáva logiku pre daného drona, čo zahŕňa jeho pohyb, streľbu, znovuvytvorenie, zmiznutie pri zásahu, náhodnú udalosť, ktorá má nastať a podobne. Celá logika je implementovaná v triede `Drone`, ktorá sa nachádza v súbore `drone.rb`. Parametre v konštruktore sú nasledovné:

- **`type`** – definuje typ drona – povolené hodnoty sú *target* alebo *distractor*, podľa toho, o aký typ drona ide.
- **`droneDesign`** – inštancia triedy `DroneDesign` definujúca grafickú reprezentáciu drona.
- **`droneSpecification`** – inštancia triedy `DroneSpecification` definujúca vlastnosti drona.
- **`trajectories`** – inštancia triedy `Trajectories`.

Základné metódy implementované v tejto triede sú:

- **`respawn`**
- **`recalculateSpecifications`**
- **`move`**
- **`shootOnTower`**
- **`destroy`**
- **`update`**

4.1.3. Utility

V tomto module sú implementované triedy rozličných utilít a pomocných metód potrebných pre hru. Konkrétne sú pre túto hru implementované 4 typy utilít, a to:

- Všeobecné pomocné metódy pre hru
- Utilita pre riešenie priesečníkov
- Utilita pre riešenie zvukov v hre
- Utilita pre parsovanie štatistík

V nasledujúcich kapitolách budú bližšie popísané prvé 3 utility zo zoznamu. Poslednej, ktorá slúži na parsovanie herných štatistík sa venuje Bc. Dominik Trojčák vo svojej diplomovej práci.

4.1.3.1. Všeobecné pomocné metódy pre hru

Všeobecné pomocné metódy pre hru sú implementované v triede `GameUtils`, ktorá sa nachádza v súbore `gameUtils.rb`. Ide o nasledujúcu sadu metód:

- **`checkEndOfGame`** – metóda vráti true v prípade, že hra už skončila, inak vráti false.
- **`elapsedTime`** – metóda vráti uplynutý čas od začiatku hry v sekundách.
- **`elapsedTimeMili`** – metóda vráti uplynutý čas od začiatku hry v milisekundách.
- **`timeFormatting(time)`** – metóda naformátuje čas na formát čitateľný pre človeka.
- **`selectRandomFromArray(inputArray)`** – metóda vyselektuje a vráti náhodný prvok vybraný zo vstupného poľa.

4.1.3.2. Utilita pre riešenie priesečníkov

Táto utilita je zo všetkých 3 najdôležitejšia, pretože rieši samotné priesečníky v hre. Na základe logiky v nej implementovanej sa rozhoduje, či hráč má alebo nemá správne namierené na drona, či ho teda pri streľbe zostrelí alebo nie. Je implementovaná v triede `IntersectUtils` v súbore `intersectUtils.rb`. Obsahuje nasledujúce 2 metódy a to:

- **`intersect`**
- **`intersect2D`**

Metóda `intersect` rieši priesečník priamo v 3D na základe 2 kolmých priemetov do 2D. Metóda `intersect2D` je teda pomocná metóda, ktorá rieši priesečník v 2D rovine.

4.1.3.3. Utilita pre riešenie zvukov v hre

Táto utilita sa stará o zvukové efekty použité v hre. Je implementovaná v triede `SoundUtils` v súbore `soundUtils.rb` a jej úlohou je načítať zvuky používané v hre. To sa deje na základe volania metód SuperEngine. Pre načítanie zvuku boli použité 2 metódy zo SuperEngine, a to:

- **`ENGINE::BckgSoundLoad`** – slúži pre načítanie zvuku do pozadia
- **`ENGINE::LoadSound`** – slúži pre načítanie ľubovoľného zvuku

Samotné volanie metódy vyzerá nasledovne:

```
$bckgSound = ENGINE::BckgSoundLoad("./soundsOgg/background_1.ogg")
```

Zvukové súbory sú vo formáte OGG, ktorý je podporovaný SuperEnginom.

Okrem samotného načítavania zvukov sa táto utilita stará aj o správne nastavenie hlasitostí pre zvuky. Hlasitosť je možné nastaviť nasledovným spôsobom:

```
$bckgSound.SetVolume(0.01)
```

Hodnota hlasitosti uvádzaná ako parameter metódy `SetVolume` môže byť z intervalu o 0 po 1, kde 0 je minimálna a 1 maximálna hlasitosť.

V hre sú celkovo použité 3 rozličné zvuky:

- Zvuk nachádzajúci sa v pozadí, ktorý sa prehráva v slučke – prehrávaný súbor je `background_1.ogg`
- Zvuk lasera pri výstrele – súbor `laser2.ogg`
- Zvuk výbuchu pri zasiahnutí drona – súbor `explosion.ogg`

4.1.4. Hráč

Modul, ktorý sa stará o hernú logiku hráča obsahuje iba jednu implementovaná triedu, a to `Player`, ktorá sa nachádza v súbore `player.rb`. Samotná entita hráča zahŕňa dve časti. Prvou je otáčavá podložka, platforma, na ktorej sa hráč nachádza. Svojim spôsobom je tento objekt naviazaný na hráča, keďže sa podľa jeho pohybov otáča doľava resp. doprava. Druhou časťou je zbraň, ktorú hráč vlastní a ktorú môže ovládať a používať za účelom zostreľovania dronov. Na pohyby zbrane je naviazaný ukazovateľ mierenia, ktorý kopíruje pohyby hráča smerom hore a dole a pohybuje sa po scéne. Jednotlivé časti hráča sú detailnejšie vysvetlené a popísané v nasledujúcich podkapitolách.

4.1.4.1. Pohyb hráča

Keďže hráč sa môže vo virtuálnom prostredí pohybovať iba do strán, sú pre jeho pohyb implementované nasledujúce dve metódy, a to `rotateLeft`, pre otáčanie hráča smerom doľava a `rotateRight` pre otáčanie doprava. Otáčanie je hráčovi vo virtuálnej scéne umožňené pomocou otáčavej platformy, na ktorej sa nachádza. Tieto metódy vyzerajú veľmi podobne, na ukážku je nižšie zobrazená implementácia metódy `rotateRight`:

```
# Rotate tower to the right
def rotateRight(power = 1)
     if @actualHorizontalAngle > ($startHorizontalAngle -
$rangeOfRotation)
          rotateHorizontal(true, power)
          setLaserVerticalRotation
     end
end
```

Vstupným parametrom do metódy je `power` – táto hodnota určuje rýchlosť otáčania na základe intenzity vychýlenia páky do strán pri použití pákového ovládača. Čím vačšia je táto hodnota, tým rýchlejšie sa bude platforma otáčať. Pri ostatných typoch ovládačov je táto hodnota implicitne nastavená na predvolenú hodnotu 1. V implementácii vidíme taktiež vstupnú podmienku pre to, aby vôbec ku otočeniu došlo. Táto podmienka zabezpečuje, aby sa hráč resp. platforma vedela otočiť do strany iba do istého ihlu, kde `$rangeOfRotation` predstavuje maximálny uhol otočenia do strany v stupňoch a premenná `@actualHorizontalAngle` predstavuje aktuálnu hodnotu natočenia do strany v stupňoch. Premenná `$startHorizontalAngle` označuje začiatočnú hodnotu otočenia, čím je zabezpečené to, že nie vždy musí byť počiatočná hodnota otočenia 0 stupňov, ale dá sa konfigurovať. To je využiteľné napríklad v prípadoch, keď nám počiatočné otočenie nevyhovuje, napríklad kvôli nevhodnej scéne pred nami. Vtedy môžeme buď prispôsobovať scénu, pokiaľ nebude vyhovujúca, alebo jednoducho otočiť hráča o istý stupeň.

4.1.4.2. Zameriavanie na dronov

O zameriavanie dronov sa stará metóda `focusingOnDrones`, ktorej vnútro je implementované nasledovne:

```
# Reset focused drone index
@focusedDroneIdx = -1

# Calculating if player focused some drone
# using IntersectUtils class
$drones.each_with_index {|drone, droneIdx|

      # Possible to focus only on drone which is active
      if drone.active
            if IntersectUtils.intersect($camera, @marker,
            drone.baseDroneModel)
                  @focusedDroneIdx = droneIdx
                  break
            end
      end
}
```

Algoritmus v tejto metóde iteruje všetkými dronmi, odfiltruje iba drony, ktoré sú aktuálne aktívne, to znamená, že sú viditeľné pre hráča, a následne pomocou metódy `intersect` z triedy `IntersectUtils` skontroluje, či má hráč namierené na príslušného drona, ak nie, pokračuje ďalej v iterovaní, pokiaľ nepreiteruje všetkými dronmi. Cyklus sa môže ukončiť 2 spôsobmi:

1. Nenájde žiaden dron, ktorý je zároveň aktívny a na ktorého má hráč zároveň namierené
2. Nájde sa dron, ktorý spĺňa tieto podmienky. V takomto prípade sa ukončí iterovanie pomocou konštrukcie *break* a zameraný dron sa označí žltým indikátorom.

4.1.4.3. Streľba na dronov

Streľba na dronov je umožnená hráčovi pomocou namapovaného tlačidla na ovládači. Po stlačení tohoto tlačidla sa okamžite vykoná metóda shoot, ktorá je implementovaná v triede hráča. Táto metóda sa riadi nasledujúcim algoritmom:

1. Najprv sa skontroluje, či je zbraň nabitá, či má dostatočnú energiu na výstrel. Táto funkcionalita je implementovaná z jednoduchého dôvodu, a to aby hráč nemohol jednoduchým spôsobom podvádzať a to tak, že by stále držal tlačidlo streľby stlačené. O splnenie tejto funkcionality sa stará nasledujúca časť metódy:

   ```
   if @actualTowerShootPower >= $requiredTowerShootPower
   ```

   Ak je táto podmienka splnená, metóda pokračuje v svojom vykonávaní, ak nie, ukončí sa. Premenná `@actualTowerShootPower` predstavuje aktuálnu hodnotu energie, kdežto premenná `$requiredTowerShootPower` predstavuje minimálnu hodnotu energie potrebnú k výstrelu.

2. Ak bola predošlá podmienka splnená, dôjde k výstrelu, zobrazí sa laser a vynuluje sa energia zbrane. Takisto sa prehrá zvuk výstrelu. Následne sa skontroluje, či hráč mal alebo nemal úspešne zamierené na nejakého drona. O túto funkcionalitu sa stará nasledovný kus kódu:

   ```
   if @focusedDroneIdx > -1
         # Mark drone as non active
         $drones[@focusedDroneIdx].setActive(false)
   end
   ```

   V prípade, že hráč mal úspešne zamerané na nejakého drona, to je vtedy, ak hodnota `@focusedDroneIdx`, ktorá označuje index drona z poľa dronov, na ktorého je aktuálne zamerané, je väčšia ako -1, dôjde k deaktivovaniu príslušného drona pomocou akcie `setActive` s argumentom false.

4.1.5. Hlavný modul

Hlavný modul je modul, ktorý všetky ostatné moduly spája dohromady a vytvára tak ucelenú hru. Je to modul, ktorý obsahuje hernú slučku. Tá je už popísaná v časti tejto práce, ktorá sa zaoberá návrhom hry, konkrétne v kapitole 3.1., ktorá sa zaoberá návrhom scenáru hry. Celý hlavný modul je implementovaný v súbore `main.rb`. Hernú slučku podporuje priamo SuperEngine, a to implementovaním metódy `ENGINE::Actions`. Tá sa následne vykonáva v slučke donekočna resp. do explicitného zastavenia programu. V tejto hernej slučke je vyriešený scenár hry na základe aktuálneho stavu hry, ktorý môže nadobudnúť jednu z nasledujúcich hodnôt, ktoré sú vytvorené ako enumeračné typy v triede `GameState`:

```
class GameState
      GameStarted = 'GameStarted'
      EpisodeFirstLoop = 'EpisodeFirstLoop'
      EpisodeEnded = 'EpisodeEnded'
      WaitingForNextEpisode = 'WaitingForEpisode'
      EpisodeRunning = 'EpisodeRunning'
      LevelEnded = 'LevelEnded'
      GameEnded = 'GameEnded'
      GameStopped = 'GameStopped'
End
```

Herná slučka s použitím týchto stavov hry potom vyzerá nasledovne:

```
# Main game loop
def ENGINE::Actions
      # Handling data coming from optitrack
      # Applied only if used environment is CAVE
      if $CAVE == 1
            ENGINE::UpdateCaveUserFromOptitrack()
      end

      actualGameState = updateGameState

      case (actualGameState)
            when (GameState::GameStopped)

            when (GameState::GameStarted)
                  $startTimeOfGame = Time.now.to_i

            when (GameState::EpisodeFirstLoop)
                  episodeFirsLoopAction

            when (GameState::EpisodeRunning)
                  episodeRunningAction

            when (GameState::EpisodeEnded)
                  episodeEndAction

            when (GameState::WaitingForNextEpisode)
                  episodeWaitingAction

            when (GameState::LevelEnded)
                  levelEndAction

            when (GameState::GameEnded)

            else
                  $log.Error('Invalid game state')
            end
end
```

Na začiatku hernej slučky dôjde vždy k aktualizovaniu stavu hry pomocou metódy `updateGameState`. Výsledok tejto metódy je aktuálny herný stav, ktorý je vyrátaný na základe aktuálnych hodnôt premenných v programe. Následne sa na základe stavu hry vždy vykonáva príslušná namapovaná akcia. Napríklad, ak sa hra nachádza v stave `EpisodeRunning`, vykonáva sa akcia `episodeRunningAction`, pre stav `LevelEnded` je to `levelEndAction`. Podobne je tomu pre ďalšie stavy hry.

Záver

Úvod tejto práce bol venovaný stručnému opisu práce, čo bolo hlavnou motiváciou pre vytvorenie daného systému, čo je cieľom experimentu, ktorý je popísaný v tejto práci a ako má výsledok práce dopomôcť pri jeho samotnej realizácii. Na začiatku sa v práci opisuje analýza požiadaviek, ktoré boli zadané zadávateľmi, ktorými boli ľudia, odborníci v oblasti kognitívnej vedy, zo Slovenskej akadémie vied a Univerzity Komenského v Bratislave. Analýza sa zaoberá najmä možnosťami realizovateľnosti daných požiadaviek v CAVE prostredí a možnosťami SuperEngine, ktoré boli zvolené ako prostriedok pre realizáciu tejto hry. Na základe podrobnej analýzy sa podarilo navrhnúť, implementovať a prispôsobiť hru v prostredí virtuálno-realitnej jaskyne v laboratóriu LIRKIS na Technickej univerzite v Košiciach.

Počas testovania implementovanej hry bola hra vhodne vyladená tak, aby zodpovedala požiadavkam zadávateľov a bola vhodne pripravená na použitie ako jeden z neodmysliteľných prostriedkov pre experiment. Bolo ukázané, že boli splnené všetky požiadavky na hru, ktoré boli zadané zadávateľom. Samotný zadávatelia si hru vo virtuálno-realitnom prostredí aj sami vyskúšali a zhodnotili, že spĺňa ich požiadavky a očakávania.

Výsledok tejto práce, samotná hra, spolu s jej druhou časťou, ktorú implementoval kolega Bc. Dominik Trojčák a ktorá sa zaoberá konfiguráciou samotnej hry, zberom štatistických údajov z nej a ich následným vyhodnocovaním, je tak hra plne funkčná a pripravená na reálne použitie v experimente, ktorý je aktuálne, v čase písania tejto práce, vo fáze prípravy a prvotného testovania. Hlavným prínosom práce je teda podpora experimentu, ktorý má za úlohu posúdiť zmenu kognitívnych funkcií človeka vplyvom dlhodobješieho hrania špeciálne prispôsobenej hry vo virtuálno-realitnom prostredí, konkrétne v prostredí virtuálno-realitnej jaskyne. Vedľajším prínosom práce je čiastočné zmapovanie istých častí vizualizačného nástroja SuperEngine a jeho rozhrania pre programovanie aplikácií, kde sú konkrétne popísané niektoré z jeho možností a dostupných funkcií.

Zoznam použitej literatúry

[1]. KOREČKO, Štefan; HUDÁK, Marián; SOBOTA, Branislav; MARKO, Martin; CIMROVÁ, Barbora; FARKAŠ, Igor; ROSIPAL, Roman. Assessment and training of visuospatial cognitive functions in virtual reality: proposal and perspective. 9th IEEE International Conference on Cognitive Infocommunications CogInfoCom 2018: PROCEEDINGS: 22. – 24.8.2018: Budapest P. 39 – 43 Danvers: IEEE, 2018.

[2]. BURDEA, Grigore C.; COIFFET, Philippe. Virtual reality technology. John Wiley & Sons, 2003.

[3]. SCHULTHEIS, Maria T.; RIZZO, Albert A. The application of virtual reality technology in rehabilitation. Rehabilitation psychology, 2001, 46.3: 296.

[4]. YOUNGBLUT, Christine. Educational Uses of Virtual Reality Technology. INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA, 1998.

[5]. GAO, Zaifeng, et al. Contralateral delay activity tracks object identity information in visual short term memory. Brain research, 2011, 1406: 30-42.

[6]. BOWMAN, Doug A.; MCMAHAN, Ryan P. Virtual reality: how much immersion is enough?. Computer, 2007, 40.7.

[7]. SOBOTA, Branislav; HROZEK František. Systémy virtuálnej reality. 2015.

[8]. CRUZ-NEIRA, Carolina, et al. Scientists in wonderland: A report on visualization applications in the CAVE virtual reality environment. In: Virtual Reality, 1993. Proceedings., IEEE 1993 Symposium on Research Frontiers in. IEEE, 1993. p. 59-66.

[9]. BROOKS, Frederick P. What's real about virtual reality?. IEEE Computer graphics and applications, 1999, 19.6: 16-27.

[10]. D. Kravitz, K. Saleem, C. Baker, and M. Mishkin, “A new neural framework for visuospatial processing,” Nature Reviews Neuroscience, vol. 12, no. 4, pp. 217–230, 2011.

[11]. N. de Bruin, D. Bryant, J. MacLean, and C. Gonzalez, “Assessing visuospatial abilities in healthy aging: A novel visuomotor task,” Frontiers in Aging Neuroscience, vol. 8, pp. 1–9, 2016.

[12]. Baddeley, “Working memory: Theories, models, and controversies,” Annual Review of Psychology, vol. 63, no. 1, pp. 1–29, 2012.

[13]. R. Shepard and J. Metzler, “Mental rotation of three-dimensional objects,” Science, vol. 171, no. 3972, pp. 701–703, 1971.

[14]. R. Polana, M. Nitsche, C. Korman, G. Batsikadze, and W. Paulus, “The importance of timing in segregated theta phase-coupling for cognitiveperformance,” Current Biology, vol. 22, no. 14, pp. 1314–1318, 2012.

[15]. P. Toril, J. Reales, J. Mayas, and S. Ballesteros, “Video game training enhances visuospatial working memory and episodic memory in older adults,” Frontiers in Human Neuroscience, vol. 10, pp. 1–14, 2016.

[16]. Barman, A. Chatterjee, and R. Bhide, “Cognitive impairment and rehabilitation strategies after traumatic brain injury,” Indian Journal of Psychological Medicine, vol. 38, no. 3, pp. 172–181, 2016.

[17]. N. Dijkstra, P. Zeidman, S. Ondobaka, M. V. Gerven, and K. Friston, “Distinct top-down and bottom-up brain connectivity during visual perception and imagery,” Scientific Reports, vol. 7, no. 1, pp. 1–9, 2017.

[18]. Z. Shipstead, T. Harrison, and R. Engle, “Working memory capacity and visual attention: Top-down and bottom-up guidance,” Quarterly Journal of Experimental Psychology, vol. 65, no. 3, pp. 401–407, 2012.

[19]. W. Paulus, “Transcranial electrical stimulation (tes - tdcs; trns, tacs) methods,” Neuropsychological Rehabilitation, vol. 21, no. 5, pp. 602–617, 2011.

[20]. M. I. Posner, M. K. Rothbart, and Y. Y. Tang, “Enhancing attention through training,” Current Opinion in Behavioral Sciences, vol. 4, pp.1–5, 2015.

[21]. C. Meneghetti, E. Borella, and F. Pazzaglia, “Mental rotation training: transfer and maintenance effects on spatial abilities,” Psychological Research, vol. 80, no. 1, pp. 113–127, 2016.

[22]. C. H. Li, X. He, Y. J. Wang, Z. Hu, and C. Y. Guo, “Visual working memory capacity can be increased by training on distractor filtering efficiency,” Frontiers in Psychology, vol. 8, no. FEB, pp. 1–12, 2017.

[23]. Neubauer, S. Bergner, and M. Schatz, “Two- vs. three-dimensional presentation of mental rotation tasks: Sex differences and effects of training on performance and brain activation,” Intelligence, vol. 38, no. 5, pp. 529–539, 2010.

[24]. M. T. Schultheis, J. Himelstein, and A. A. Rizzo, “Virtual reality and neuropsychology: upgrading the current tools,” The Journal of head trauma rehabilitation, vol. 17, no. 5, pp. 378–394, 2002.

[25]. R. J. Matheis, M. T. Schultheis, L. A. Tiersky, J. DeLuca, S. R. Millis, and A. Rizzo, “Is learning and memory different in a virtual environment?” The Clinical Neuropsychologist, vol. 21, no. 1, pp. 146–161, 2007.

[26]. T. D. Parsons, A. R. Carlew, J. Magtoto, and K. Stonecipher, “The potential of function-led virtual environments for ecologically valid measures of executive function in experimental and clinical neuropsychology,” Neuropsychological rehabilitation, vol. 27, no. 5, pp. 777–807, 2017.

[27]. M. Hudak, v. Korečko, and S. Branislav, “On architecture and performance of lirkis cave system,” in 8th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), 2017, pp. 295–300. ThalmicLabs. (2018) Myo armband homepage, https://www.myo.com. [Online]. Available: https://www.myo.com NaturalPoint. (2018) Optitrack homepage, https://optitrack.com. [Online]. Available: https://optitrack.com

[28]. M. Hudak, Š. Korečko, and B. Sobota, “Special input devices integration to lirkis cave,” Open Computer Science, vol. 8, no. 1, pp. 1–9, 2018.

[29]. E. Vogel and M. Machizawa, “Neural activity predicts individual differences in visual working memory capacity,” Nature, vol. 428, no. 6984, pp. 748–751, 2004.

[30]. S. Luck, An introduction to the Event-related Potential Technique. London: The MIT Press, 2014.

[31]. R. Luria, H. Balaban, E. Awh, and E. Vogel, “The contralateral delay activity as a neural measure of visual working memory,” Neuroscience and Biobehavioral Reviews, vol. 62, pp. 100–108, 2016.

[32]. P. Baranyi and A. Csapo, “Definition and synergies of cognitive infocommunications,” Acta Polytechnica Hungarica, vol. 9, no. 1, pp. 67–83, 2012.

[33]. Horvath and A. Sudar, “Factors contributing to the enhanced performance of the maxwhere 3d vr platform in the distribution of digital information,” Acta Polytechnica Hungarica, vol. 15, no. 3, pp. 149–173, 2018.

[34]. V. Kovecses-Gosi, “Cooperative learning in vr environment,” ¨ Acta Polytechnica Hungarica, vol. 15, no. 3, pp. 205–224, 2018

Prílohy

Príloha A CD médium – záverečná práca v elektronickej podobe, prílohy v elektronickej podobe, vizualizačný nástroj spolu so spustiteľným programom a zdrojové kódy aplikácie.

Príloha B Používateľská príručka

Príloha C Systémová príručka

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Používateľská príručka
Príloha B k diplomovej práci

2019

Bc. Peter Vasiľ

Obsah

Zoznam obrázkov

Zoznam tabuliek

1. Program a jeho funkcia

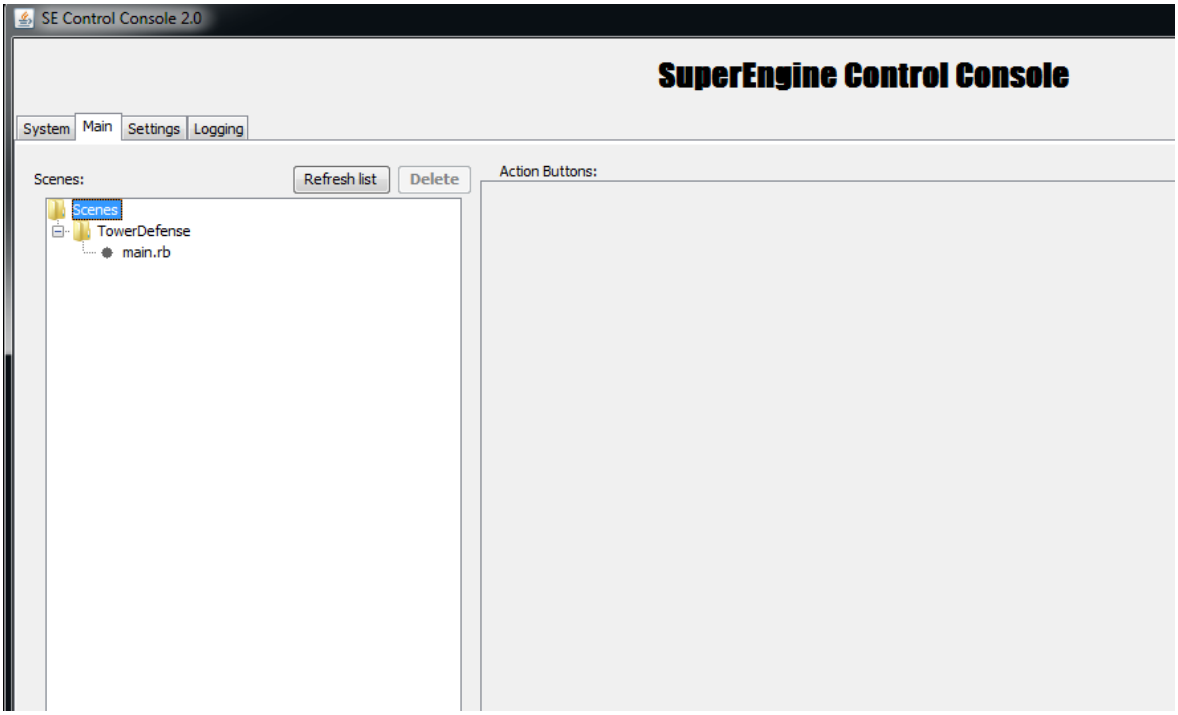
Hlavnou úlohou systému je simulácia hry. Táto hra sa odohráva vo virtuálnom priestore reprezentovanom vesmírom, kde hráč sa nachádza na otočnej plošine (veži) a pomocou zbraní sa snaží zničiť nepriateľské objekty (ciele). Používateľ je v tomto prípade človek, ktorý sa nachádza v prostredí virtuálno-realitnej jaskyne a zabezpečuje chod hry a jej nastavenie. V tejto príručke je teda definovaný postup a spôsob, akým používateľ zabezpečí, aby hra bola spustená korektne a ako výstup produkovala požadované dáta. Taktiež je v tejto príručke popísaný návod k samotnej hre.

2. Inštalácia

Hra je spustiteľná pomocou SuperEnginu, ktorý musí byť nainštalovaný podľa príručky k samotnému nástroju. Na spustenie celého systému je potrebné mať na lokálnom disku umiestnenú celú zložku Slovakia Supercomputers a nainštalované potrebné softvérové vybavenie (Java Runtime Enviroment 8, 3DxSoftware_v3-8-1, ImageMagick-6.3.7-8-Q8-windows-dll, LAVFilters-0.55.3, vcredist_x86).

2.1. Inštalácia a spustenie hry

Inštalácia samotnej hry pozostáva zo skopírovania adresára „TowerDefense“ do adresára SuperEnginu („Slovakia Supercomputers/SuperEngine/Packages/“). Po prekopírovaní a spustení systému sa adresár zobrazí priamo v konzole, ako je možné vidieť na obrázku 1 nižšie. Hra je dostupná v dvoch verziách, ktoré sú rozdelené samostatne. Jedna verzia je určená pre CAVE a druhá pre prezentačné účely. Verzia 2 je prispôsobená pre bežný počítač. V oboch prípadoch adresár obsahuje spustiteľnú verziu v podobe skriptu „main.rb“, ako je znázornené na obrázku č.1.



Obr. 18 Zobrazenie zložky hry v konzole

3. Konfigurácia levelov

Vytvorenie a konfigurácia levelov je nevyhnutná pre obe verzie. Štandardne má hra preddefinovaných 30 levelov, pričom je možné ich pridávať, odoberať a modifikovať v stanovenom formáte. Levely je možné konfigurovať v zložke hry („/levels/levels.csv“). Po zmene levelov a opätovnom načítaní scény sa zmena v používateľskom prostredí prejaví tak, že používateľ bude môcť vyberať z aktuálne nakonfigurovaných levelov.

Základná konfigurácia levelov:

Level	Episode time	Swarm angle	# Targets	# Distractors
1	Long	Easy	2	2
2	Long	Easy	2	3
3	Long	Easy	3	3
4	Long	Medium	3	3
5	Long	Medium	3	4
6	Long	Medium	4	3
7	Long	Medium	4	4
8	Long	Hard	4	4
9	Long	Hard	4	5
10	Long	Hard	5	4
11	Middle	Easy	3	3
12	Middle	Easy	3	4
13	Middle	Easy	4	4
14	Middle	Medium	4	5
15	Middle	Medium	5	4
16	Middle	Medium	5	5
17	Middle	Hard	4	5
18	Middle	Hard	5	5
19	Middle	Hard	5	6
20	Middle	Hard	5	7
21	Short	Easy	3	4
22	Short	Easy	4	4
23	Short	Medium	4	5
24	Short	Medium	5	4
25	Short	Medium	5	5
26	Short	Hard	4	5
27	Short	Hard	5	5
28	Short	Hard	5	6
29	Short	Hard	5	7
30	Short	Hard	6	7

Tab. 10 Základná konfigurácia levelov

4. Konfigurácia hráčov

Pri plnej verzii hry, ktorá sa spúšta v laboratóriu LIRKIS, je potrebné nadefinovať hráčov, ktorí sa zúčastňujú experimentu. Podobne ako pri leveloch je potrebné týchto hráčov vytvoriť v csv súbore, ktorý sa nachádza v adresári hry („/players/players.csv“). Používateľ môže hráčov pridávať, odoberať a editovať. Neodporúča sa zmena ID hráčov počas experimentu, pretože na základe ID sa následne každému hráčovi prideľuje skóre. Po zmene hráčov sa táto zmena v používateľskom prostredí prejaví tak, že bude možné daných hráčov vybrať zo zoznamu, ktorý sa zobrazuje pomocou číselníka.

Príklad vytvorených hráčov v csv súbore players.csv:

Meno	**Priezvisko**	**Id**
Dominik	Trojcak	1
Peter	Vasil	2

Tab. 11 Základná konfigurácia hráčov

5. Spustenie hry

5.1. Postup spustenia hry

5.1.1. Výber hráča

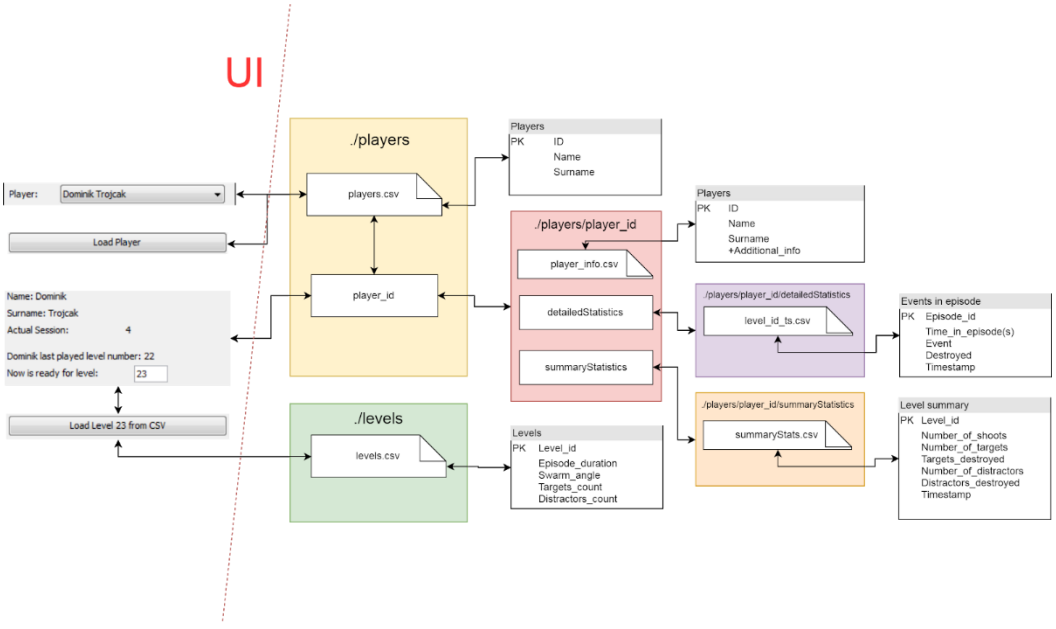
- Výber hráča pomocou číselníka v sekcii “Player info”.
- Po zvolení hráča je potrebné stlačiť tlačidlo “Load Player”, čím sa inicializuje hráčov profil. Zobrazia sa informácie o aktuálnom sedení a o poslednom absolvovanom leveli. V prípade, že daný hráč hrá hru prvýkrát, na pozadí systém dotvorí všetky potrebné súbory.

5.1.2. Načítanie levelu

- Používateľské rozhranie umožnuje spustiť ľubovoľný level v rozsahu 1 až *n*, kde *n* je celkový počet levelov definovaný v csv súbore levels.csv.
- Po zvolení čísla levelu sa level inicializuje stlačním tlačidla “Load level x from CSV”
- Systém načíta dáta z csv a preklopí ich do časti “Level setup”, pričom dáta z csv nie je možné ďalej upravovať. Čas levelu je stanovený na 8 minút. Podľa tohto času systém vyráta počet epizód na základe dĺžky epizódy a dĺžky prestávky medzi epizódami.

5.1.3. Spustenie levelu

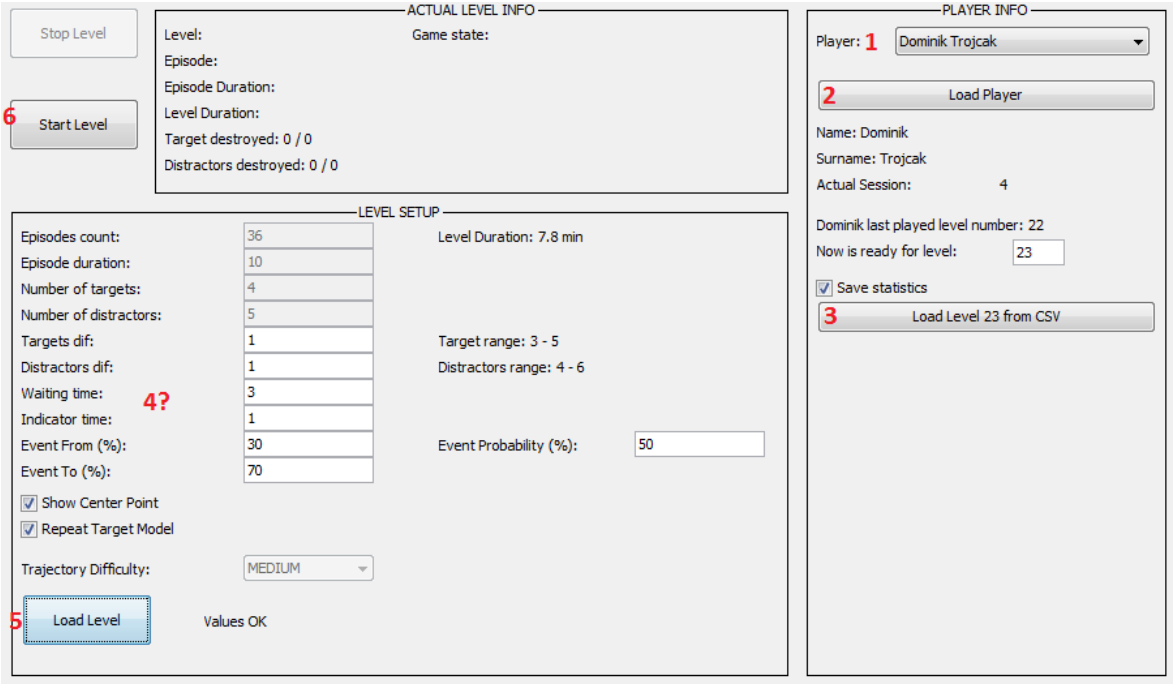
- Pomocou tlačidla “Load level” v časti “Level setup” sa nastaví celková konfigurácia v hre.
- Následne sa sprístupní tlačidlo “Start level”, ktorým sa spustí daný level.



Obr. 19 Schéma výberu hráča a levelu

5.1.4. Postup v bodoch (obr. č. 3) - CAVE

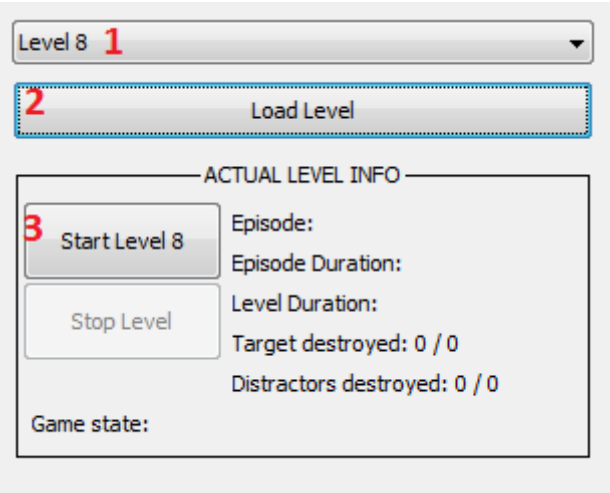
1. Výber hráča z číselníka
2. Načítanie hráča – “Load Player”
3. Načítanie levelu z CSV – “Load Level *x* from CSV”
4. Zmena konfigurácie – nepovinná
5. Načítanie levelu z konfigurácie do hry – “Load Level”
6. Spustenie levelu – “Start Level”



Obr. 20 Postup spustenia levelu - CAVE

5.1.5. Popstup v bodoch (obr. č. 4) - Desktop

1. Výber levelu z číselníka
2. Načítanie zvoleného levelu – “Load Level”
3. Spustenie levelu – “Start Level”

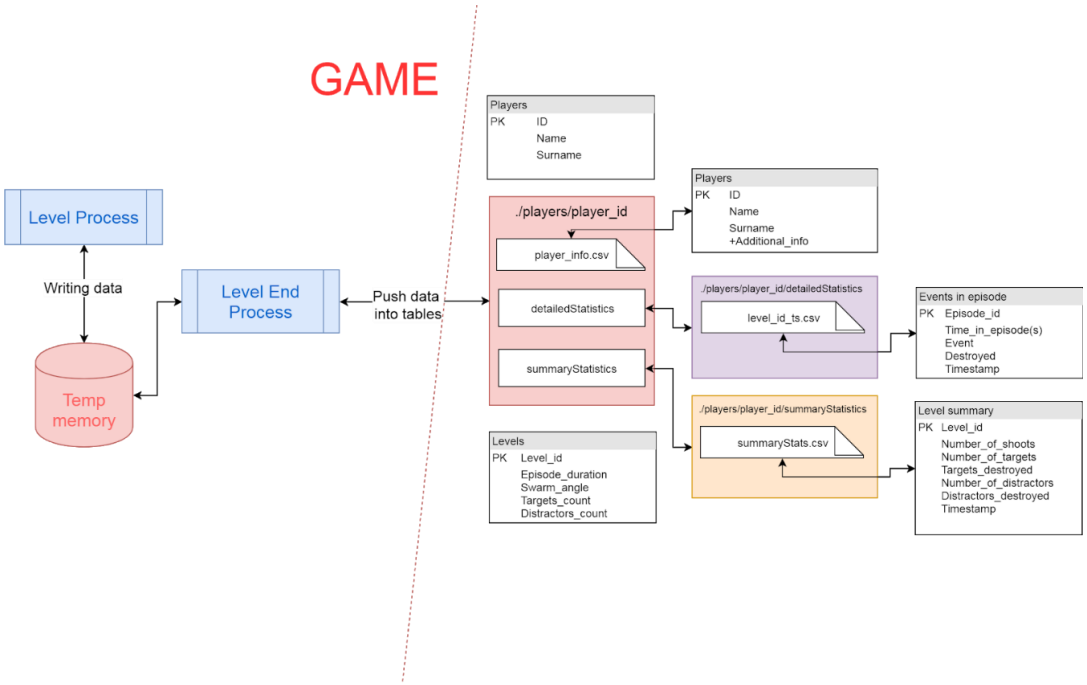


Obr. 21 Postup spustenia levelu - Desktop

6. Zaznamenávanie a ukladanie štatistík

Každému hráčovi sa po každom leveli vygenerujú príslušné štatistiky, pričom platia nasledujúce pravidlá:

- Zaznamenávanie hráčových štatistík prebieha automaticky na pozadí hry.
- Hráčovi sa štatistiky zapíšu do jeho zložky po korektnom ukončení levelu.
- Štatistiky sú rozdelené na sumárne a detailné**.**
- Sumárne štatistiky uchovávajú informácie pre celkové skóre levelu. Jeden level = jeden riadok v tabuľke. Záznam obsahuje informácie, koľkokrát hráč vystrelil, koľko bolo v leveli cieľov, koľko bolo v leveli distraktorov, koľko cieľov zničil, koľko distraktorov zničil a zároveň časovú známku pre každý z týchto údajov.
- Detailné štatistiky predstavujú konkrétne udalosti v danom leveli. Jeden level = jeden csv súbor.
- Csv súbor má v názve číslo levelu a časovú známku.
- Záznam obsahuje číslo epizódy, čas od začiatku epizódy v sekundách, udalosť ktorá nastala, info o zostrelení/nezostrelení objektov a časovú známku.
- Udalosti sú následovné: výstrel, zatmenie, začiatok epizódy.



Obr. 22 Schéma zaznamenávania štatistík

7. Hra

7.1. Ovládanie

Hru je možné ovládať pomocou klávesnice alebo herným ovládačom v závislosti od toho, či je hra spustená v desktopovej verzií alebo vo verzií pre virtuálnu jaskyňu.

7.1.1. Ovládanie – desktopová verzia

Ovládanie dekstopovej verzie je možné pomocou klávesnice:

- Otočenie vpravo – šípka vpravo
- Otočenie vľavo – šípka vľavo
- Otočenie hore – šípka hore
- Otočenie dole – šípka dole
- Streľba – medzerník

7.1.2. Ovládanie – verzia pre virtuálnu realitu

Vo virtuálnej jaskyni je ovládanie možné využitím herného ovládača. Po zapojení ovládača do niektorých z USB portov riadiaceho počítača je porebné spustiť príslušný program slúžiaci na prepojenie ovládača a SuperEnginu.

Ovládanie (obrázok č. 6):

- Otočenie vpravo – 2
- Otočenie vľavo – 1
- Otočenie hore – 4
- Otočenie dole – 3
- Streľba – 5 – tlačidlo umiestnené na opačnej strane ovládača

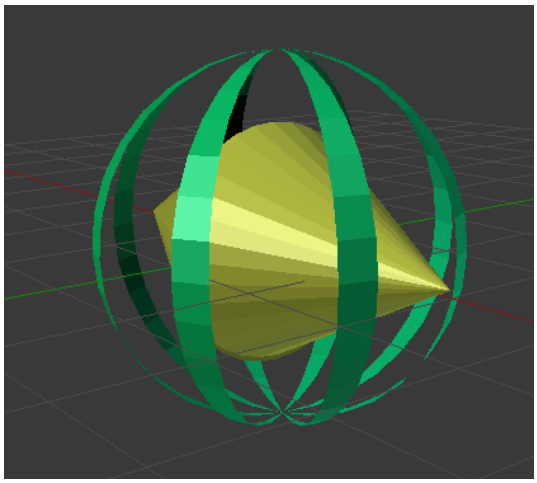
Obr. 6 Ovládanie herným ovládačom (joystick)

7.2. Postup hry

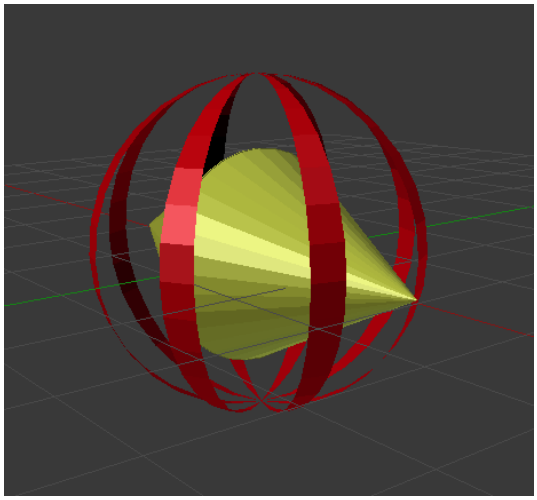
Postup hry je popísaný v nasledujúcich bodoch:

1. **Začiatok náletu**

   V tejto fáze sú hráčovi zobrazené identifikátory, ktoré označujú, či sa jedná o cieľ (červený – obrázok č. 8) alebo distraktor (zelený – obrázok č. 7).



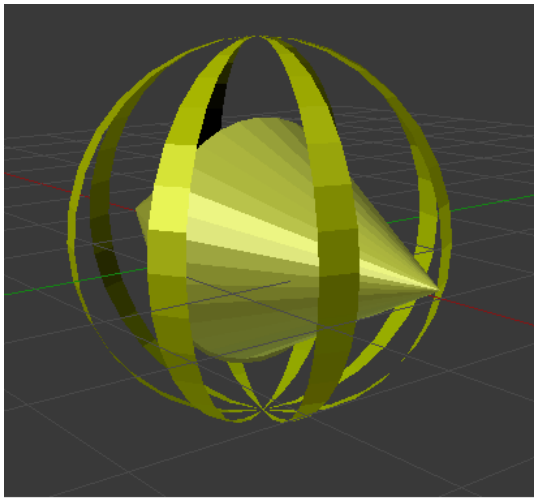
Obr. 7 Identifikátor distraktora



Obr. 8 Identifikátor targetu

2. **Priebeh epizódy**

   Počas epizódy sa hráč snaží zostreliť nepriateľské objekty (ciele) a nezosreliť priateľské (distraktory). Pri zameraní jednej alebo druhej skupiny sa v okolí objektu zobrazí žltý identifikátor (obrázok č. 9). Zameraný objekt je možné zostreliť stlačením príslušného tlačidla na príslušnom ovládači.



Obr. 8 Identifikátor zamerania objektu

3. **Záver epizódy**

   Epizóda končí v okamihu, kedy uplynie jej čas. Čas epizódy je pevný a od neho sa odvíja jednotlivých rýchlosť dronov.

4. **Čakanie na ďalšiu epizódu**

   Po skončení epizódy hráč čaká na ďalšiu, až kým neskončí daná úroveň (level). V prípade, že nasleduje ďalšia úroveň pokračuje sa bodom č.1.

5. **Ukončenie levelu**

   Po uplynutí poslednej epizódy úroveň (level) končí. V prípade desktopovej verzie sa hráčovi zobrazí skóre a sumárna štatistika priamo na obrazovke. V prípade verzie pre virtuálnu jaskyňu sa skóre zapíše do príslušných csv súborov so štatistikami.

8. Webová aplikácia

Webová aplikácia slúži pre publikáciu výsledkov hráčov. Z používateľského hľadiska je potrebné pre zobrazenie aktuálnych výsedkov nahrať zložku „/players“, ktorá je súčasťou balíka hry, do pripraveného repozitára servra. Odtiaľ si už aplikácia automaticky berie dáta a nie je potrebný žiaden ďalší manuálny zásah.

8.1. Inštalácia

Pre spustenie webovej aplikácie je potrebné mať k dispozícií server, na ktorom bude možné spustiť react.js aplikáciu a node.js server.

Požiadavky na server:

- nainštalovaný webServer – Apache, Nginx - https://www.nginx.com/
- nainštalovaný node - https://nodejs.org/en/
- nainštalované npm - https://www.npmjs.com/

8.2. Spustenie serverovej aplikácie – back end

1. Otvoriť terminál v zložke projektu - Zdrojové kódy\Webová aplikácia\Server
2. Inštalácia balíčkov – npm install
3. Build projektu – npm build
4. Presun node_modules do buildu - mv node_modules ./dist/
5. Synchronizácia adresára rsync -avzc --no-times --no-perms --no-owner --no-group --delete ./dist/ <server>:<doména> /api/api/
6. Reštart servera - sudo systemctl restart – priamo na serveri
7. Test - = <doména API>/v1/docs – zobrazí dokumentáciu

8.3. Spustenie klientskej aplikácie – front end

1. Otvoriť termínal v zložke projektu - Zdrojové kódy\Webová aplikácia\Klient
2. Nastavenie .env premenných

   - v projekte je potrebné nastaviť globálne premenné na správne hodnoty nasledovne:

   - REACT_APP_DOMAIN= <doména API>/v1, lokálne = localhost:9999/v1
3. Inštalácia balíčkov – npm install
4. Build projektu (POZOR – API musí bežať) – npm build
5. Synchronizácia adresára rsync -avzc --no-times --no-perms --no-owner --no-group --delete ./build/ <server>:<doména> /toverDeffense /public/

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Systémová príručka
Príloha C k diplomovej práci

2019

Bc. Peter Vasiľ

Obsah

1. Program a jeho funkcia

Hlavnou úlohou systému je simulácia hry. Táto hra sa odohráva vo virtuálnom priestore reprezentovanom vesmírom, kde hráč sa nachádza na otočnej plošine (veži) a pomocou zbraní sa snaži zničiť nepriateľské objekty (ciele). Používateľ je v tomto prípade človek, ktorý sa nachádza v prostredí virtuálno-realitnej jaskyne a zabezpečuje chod hry a jej nastavenie. V tejto príručke sú opisané jednotlivé moduly, ich funkcie a premenné potrebné pre požadovaný chod systému.

2. Program a jeho riešenie

2.1. Zoznam modulov

- **Hlavný modul**
  - main.rb
- **Controllers - ovládače**
  - gamepadController.rb
  - joystickController.rb
  - keyboardController.rb
- **Drones - drony**
  - drone.rb
  - droneDesign.rb
  - droneSpecification.rb
  - trajectories.rb
  - droneModelPool.rb
- **Utils – utility**
  - gameUtils.rb
  - intersectUtils.rb
  - soundUtils.rb
  - statisticsParserUtils.rb
- **Player – hráč**
  - player.rb
- **Ostatné**
  - gameStates.rb
  - levelConfig.rb
  - paricipant.rb

2.2. Popis metód v hlavnom module main.rb

Hlavný skript programu, v ktorom prebieha celý životný cyklus programu.

- **guiInit()**

  - inicializovanie prvkov grafického rozhrania

- **refreshGUI(gameState)**

  - zmena používateľského rozhrania na základe stavu hry „gameState“

- **hasTempCorrectValue**

  - kontrola vstupných hodnôt. Návratové hodnoty false/true.

- **updateLevel()**

  - aktualizuje konfiguráciu úrovne vytvorením novej inštancie objektu LevelConfig

- **loadLevel()**

  - načíta úroveň z csv súboru využitím globálnej premennej $levelToLoad

- **loadPlayer()**

  - načíta hráča z csv súboru využitím globálnej premennej $actualParticipant

- **getParticipants()**

  - načíta zoznam hráčov z players.csv. Návratová hodnota je zoznam objektov „Participants“

- **getParticipantLabels(participants)**

  - vytvorí list hráčov (z participants), ktorý je použiteľný pre číselník. Návratová hodnota je zoznam reťazcov

- **handleInputChange(event)**

  - odchytáva jednotlivé udalosti elementov používateľského rozhrania

- **initLevel()**

  - inicializácia úrovne. Zavolá sa pri stlačení tlačidla „Start Level“

- **updateGameState()**

  - aktualizuje stav hry na základe stanovených podmienok

- **logStateChange(actualState, previousState)**

  - zaznamenáva zmenu stavu previousState na actualState a v prípade zmeny zapíše do konzoly SE

- **ENGINE::OnSceneStart()**

  - herná slučka

- **ENGINE::OnSceneStart()**

  - začiatok scény

- **ENGINE::OnExit()**

  - ukončenie scény

- **episodeRunningAction()**

  - akcia vykonávaná v prípade stavu hry „EpisodeRunning“

- **episodeEndAction()**

  - akcia vykonávaná v prípade stavu hry „EpisodeEnded“

- **episodeWaitingAction()**

  - akcia vykonávaná v prípade stavu hry „WaitingForNextEpisode“

- **episodeFirsLoopAction()**

  - akcia vykonávaná v prípade stavu hry „EpisodeFirstLoop“

- **levelEndAction()**

  - akcia vykonávaná v prípade stavu hry „EpisodeFirstLoop“

- **generateEpisodeData()**

  - vygeneruje dáta pre aktuálnu epizódu

- **cleanGameField()**

  - na konci epizódy vyčistí hracie pole a uvoľní premenné

2.2.1. Opis premenných v hlavnom module main.rb

Opis premenných, ktoré nie sú štandardnou súčasťou skriptu SuperEnginu. Nie su zahrnuté pomocné premenné. Konštanty majú pridelenú aj hodnotu.

- **$gm** – GUI manažér
- **$osd** – debug foreground
- **$camera** – kamera
- **$cameraStartX** = 1000 – počiatočná poloha kamery X
- **$cameraStartY** = 100 – počiatočná poloha kamery Y
- **$cameraStartZ** = 10 – počiatočná poloha kamery Z
- **$LEVEL_DURATION** = 480 – dĺžka jednej úrovne (levelu)
- **$numberOfDestroyedTargets** – počítadlo zostrelených cieľov
- **$numberOfDestroyedDistractors** – počítadlo zostrelených distraktorov
- **$detailedStatisticsArray** – pole na uloženie štatistík
- **$numberOfEpisodesTemp** – počet epizód
- **$episodeDurationTemp** – trvanie epizódy
- **$numberOfTargetsTemp** – počet cieľov
- **$numberOfTargetsDifferenceTemp** – variabilná zložka počtu cieľov
- **$numberOfDistractorsTemp** – počet distraktorov
- **$numberOfDistractorsDifferenceTemp** – variabilná zložka počtu distraktorov
- **$waitingTimeBetweenEpisodesTemp** – čas medzi epizódami
- **$droneIndicatorDurationTemp** – čas zobrazenia indikátora
- **$eventTimeFromTemp** – začiatok možnosti nastatia udalosti v %

- **$eventTimeToTemp** – koniec nastatia udalosti v %
- **$eventProbabilityTemp** – pravdepodobnosť nastatia udalosti
- **$actualLevel** – aktuálna úroveň
- **$levelToLoad** – úroveň, ktorá sa má načítať
- **$showCenterPointValue** – zobrazený stredový bod = false/true
- **$sameTargetModelIsPossible** – opakovanie sa rovnakého modelu cieľu = false/true
- **$lastTargetModel** – posledný model cieľového objektu
- **$saveStatistics** – indikátor uloženia štatistík = false/true
- **$numberOfTargetsInLevel** – počet cieľov
- **$numberOfDistractorsInLevel** – počet distraktorov
- **$shootCount** – počet výstrelov
- **$actualTrajectoryDifficulty** – obtiažnosť rozptylu trajektórií
- **$playersDirectory** = './players/' – adresár hráčov
- **$levelsDirectory** = './levels/' – adresár úrovní
- **$actualParticipant** – aktuálny hráč
- **$isLevelLoadedFromCSV** – indikátor, či bol hráč načítaný z csv súboru = false/true
- **$participants** – zoznam hráčov

Hlavný skript main.rb obsahuje aj ďalšie premenné týkajúce sa používateľského rozhrania.

2.3. Ovládače

Tento balíček obsahuje moduly, ktoré umožnujú ovládanie hry pomocou rôznych zariadení.

2.3.1. Modul gamepadController.rb

2.3.1.1. Popis metód

- **dataAxisX()** – načíta hodnotu X gamepadu
- **dataAxisY()** – načíta hodnotu Y gamepadu
- **dataAxisZ()** – načíta hodnotu Z gamepadu
- **dataAxisZrot()** – načíta hodnotu rotácie gamepadu
- **button1-8()** – načita stlačenie tlačidla 1-8 (každé samostatne)
- **handleControl**() – vykonáva požadovanú akciu pri zmene stavu tlačidiel

2.3.1.2. Popis premenných

- **$dataAxisX** – uchováva hodnotu X
- **$dataAxisY** – uchováva hodnotu Y
- **$dataAxisZ** – uchováva hodnotu Z

- **$dataAxisZrot** – uchováva hodnotu otočenia
- **$button1-8** – uchováva hodnotu tlačidiel 1-8 (každú samostatne)

2.3.2. Modul joystickController.rb

Identické s gamepadController.rb, s rozdielom, že sa jedná o iný typ zariadenia, pri ktorom sa zohľadnuje aj sila otáčania pomocou premenných **powerX** a **powerY**.

2.3.3. Modul keyboardController.rb

2.3.3.1. Popis metód

- **handleControl()** - vykonáva požadovanú akciu pri zmene stavu tlačidiel

2.3.3.2. Popis premenných

- **$dev_keyboard** – načítaná klávesnica

2.4. Drony

Balíček poskytuje moduly potrebné pre prácu s cieľmi a distraktormi.

2.4.1. Modul drone.rb

Predstavuje objekt drona.

2.4.1.1. Popis metód

- **initialize()** – konštruktor, pomocou ktorého sa inicializuje dron
- **reinitialize()** – reštartovanie vlastností drona
- **update()** – aktualizácia drona
- **initInterpolator(trajectory)** – inicilizácia interpolátora pohybu po stanovenej trajektórii (trajectory)
- **move()** – pohyb, zmena polohy drona na trajektórii
- **destroy()** – zničenie drona
- **revive()** – oživenie drona
- **setActive(value)** – aktivácia/deaktivácia drona podľa hodnoty (value = false/true)
- **droneShouldOccur()** – pomocná funkcia, či by sa mal dron zobraziť
- **selectTrajectory()** – náhodne vyberie dostupnú trajektóriu pre drona
- **reselectTrajectory()** – nanovo vyberie trajektóriu
- **droneIsBehindBorder()** – indikuje, či sa dron nachádza za hranicou kamery
- **showIndicator(show)** – aktivovanie/deaktivovanie indikátora (show = false/true)

2.4.1.2. Popis premenných

- **@type** – typ drona
- **@baseDroneModel** – model drona
- **@droneMarker** – ukazovateľ zameraného drona
- **@targetIndicator** – cieľový indikátor
- **@distractorIndicator** – distraktor indikátor
- **@actualSpeed** – @droneSpecification.speedInit
- **@trajectories** – trajektórie
- **@trajectory** – zvolená trajektória drona
- **@minInterpolatorKey** – minimálny interpolačný klúč
- **@maxInterpolatorKey** – maximálny interpolačný kľuč
- **@actualInterpolatorKey** – aktuálny interpolačný kľuč
- **@active** – aktívny/neaktívny
- **@lastRespawnedTime** – čas posledného respawnu
- **@lastShootOnTowerTime** – čas posledného výstrelu

2.4.2. Modul droneDesign.rb

Modul v sebe nesie modely potrebné pre drony.

2.4.2.1. Popis metód

- **initialize()** – konštruktor, pomocou ktorého sa inicializujú modely

2.4.2.2. Popis premenných

- **@targetIndicator** – model indikátora cieľa
- **@distractorIndicator** – model indikátora distraktora
- **@baseDroneModel** – model drona
- **@droneMarker** – model ukazovateľa zameraného drona

2.4.3. Modul droneModelPool.rb

Modul slúžiaci na uchovanie zoznamu dostupných dronov.

2.4.3.1. Popis metód

- **initialize()** – konštruktor, pomocou ktorého sa inicializuje zoznam
- **getFreeModel()** – vráti dostupný model
- **resetModels()** – reštartuje modely

2.4.3.2. Popis premenných

- **@modelType** – typ modelu
- **@models** – zoznam modelov

2.4.4. Modul trajectories.rb

2.4.4.1. Popis metód

- **initialize()** – konštruktor, pomocou ktorého sa incializujú trajektórie
- **getFreeTrajectory(difficulty)** – vráti pole trajektórií pre danú obtiažnosť (difficulty)
- **resetTrajectories()** – resetuje trajektórie

2.4.4.2. Popis premenných

- **@trajectories** – zoznam vytvorených trajektórií
- **@freeTrajectoriesIndexes** – pole dostupných trajektórií

2.5. Utility

Balíček utils obsahuje sadu pomocných funkcií

2.5.1. Modul gameUtils.rb

2.5.1.1. Popis metód

- **checkEndOfGame()** – kontroluje skončenie hry = false/true
- **elapsedTime()** – vráti čas od začiatku hry
- **elapsedTimeMili()** – vráti čas od začiatku hry v milisekundách
- **timeStamp()** – vráti aktuálnu časovú známku
- **timeStampMili()** – vráti aktuálnu časovú známku v milisekundách
- **selectRandomFromArray(array)** – vyberie náhodný prvok z poľa

2.5.2. Modul intersectUtils.rb

2.5.2.1. Popis metód

- **intersect(lineStartPoint,lineEndPoint, focusedObject)** – vráti informáciu, či je daný objekt (focusedObject) v prieniku s priamkou (lineStartPoint - lineEndPoint) = false/true
- **intersect2D(lx,ly,px,py,omega,factor,error)** – kontrola prieniku v 2D

2.5.3. Modul soundUtils.rb

2.5.3.1. Popis metód

- **initialize()** – incializácia zvukov v hre

2.5.3.2. Popis premenných

- **$bkg_sound_volume** – intenzita zvuku v pozadí
- **$fireSound** – zvuk výstrelu
- **$exploxionSound** – zvuk výbuchu
- **$droneFlySound** – zvuk drona

2.5.4. Modul statisticsParserUtils.rb

2.5.4.1. Popis metód

- **parseDetailedStatistics(statisticsArray, nameOfile, playerId, separator)** – zabezpečí uloženie detailných štatistík hráča (statisticsArray)
- **parseSummaryStatistics(summaryStatistics,playerId,sparator)** – zabezpečí uloženie sumárnych štatistík hráča (summaryStatistics)

2.6. Hráč

Modul predstavujúci triedu hráča. Oproti modulu „participant.rb“, táto trieda obsahuje celkovú funkcionalitu potrebnú pre hranie hry.

2.6.1. Modul player.rb

2.6.1.1. Popis metód

- **initialize()** – koštruktor, pomocou ktorého sa incializuje hráč
- **rotateHorizontal(right,power)** – horizontálna rotácia
- **rotateRight(power)** – rotácia vpravo
- **rotateLeft(power)** – rotácia vľavo
- **rotateUp (power)** – rotácia hore
- **rotateDown(power)** – rotácia dole
- **setLaserVerticalRotation()** – rotácia lasera
- **shoot()** – výstrel hráča
- **update()** – aktulizácia stavu hráča
- **focusingOnDrones()** – informácia o tom, či bol zameraný nejaký dron, false/true
- **updateLaserVisibility()** – zmena viditeľnosti lasera

2.6.1.2. Popis premenných

- **@marker** – model ukazovateľa mierenia
- **@bullet** – model náboja
- **@laser** – model lasera
- **@tower** – model veže

- **@actualTowerShootPower** – aktuálna sila lasera
- **@actualTowerLaserDurationTime** – aktuálny čas zobrazenia lasera
- **@focusedDroneIdx** – index zameraného drona
- **@actualHorizontalAngle** – aktuálny uhol otočenia

2.7. Ostatné

2.7.1. Modul gameStates.rb – enum

2.7.1.1. Hodnoty

- GameStarted = 'GameStarted'
- EpisodeFirstLoop = 'EpisodeEnded '
- EpisodeEnded = 'EpisodeEnded'
- WaitingForNextEpisode = 'WaitingForEpisode'
- EpisodeRunning = 'EpisodeRunning'
- LevelEnded = 'LevelEnded'
- GameEnded = 'GameEnded'
- GameStopped = 'GameStopped'

2.7.2. Modul levelConfig.rb

Modul uchováva informácie o konfigurácií aktuálnej úrovne(levelu).

2.7.2.1. Popis metód

- Initialize() – konštruktor, pomocou ktorého sa inicializuje konfigurácia úrovne
- hasCorrectValues() – kontrola správnosti hodnôt
- printConfig() – vypíše akutálnu konfiguráciu do kozoly SE

2.7.2.2. Popis premenných

- **@numberOfEpisodes** – počet epizód
- **@episodeDuration** – trvanie epizódy
- **@numberOfTargets** – počet cieľov
- **@numberOfDistractors** – počet distraktorov
- **@numberOfTargetsDifference** – variabilná zložka počtu cieľov
- **@numberOfDistractorsDifference** – variabilná zložka počtu distraktorov
- **@waitingTimeBetweenEpisodes** – čas medzi epizódami
- **@droneIndicatorDuration** – dĺžka trvania zobrazenia indikátoru

- **@eventTimeTo** - čas nastatia eventu do, v %
- **@eventTimeFrom** - čas nastatia eventu od, v %
- **@eventProbability** - pravdepodobnosť nastatie eventu, v 5
- **@actualTrajectoryDifficutly** – obtiažnosť trajektórií

2.7.3. Modul participant.rb

Modul reprezentujúci triedu hráča.

2.7.3.1. Popis metód

- **initialize()** – konštruktor, pomocou ktorého sa inicializuje hráč

2.7.3.2. Popis premenných

- **@name** – meno hráča
- **@surname** – priezvisko hráča
- **@id** – id hráča