VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HERNÍ BOJOVÝ SYSTÉM ZALOŽENÝ NA SLEDOVÁNÍ POHYBU OČÍ

EYE TRACKING BASED GAMING COMBAT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JÁN KUČERA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ CHLUBNA**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kučera Ján**
Program: Informační technologie
Název: **Herní bojový systém založený na sledování pohybu očí**
Eye Tracking Based Gaming Combat System
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s vývojovým prostředím herních enginů (Unity, Unreal Engine...).
2. Vyberte vhodné řešení pro sledování pohybu očí hráče a nastudujte s tím spojenou problematiku.
3. Navrhněte aplikaci využívající detekci pohybu očí v rámci herního soubojového systému.
4. Demo aplikaci implementujte.
5. Proveďte měření, případně uživatelskou studii hodnotící implementované výsledky.
6. Vytvořte video reprezentující výsledky vaší práce.

Literatura:

- Gregory, Jason. *Game engine architecture*. crc Press, 2018. ISBN 1351974289, 9781351974288
- Bishop, Lars, et al. "Designing a PC game engine." IEEE Computer Graphics and Applications 18.1 (1998): 46-53.
- Adams, Ernest, and Joris Dormans. Game mechanics: advanced game design. New Riders, 2012. ISBN 0321820274, 9780321820273

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, experimenty vedoucí k bodu 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cieľom tejto práce je vytvorenie 3D bojovej hry v ktorej bude užívateľ môcť mieriť pomocou pohľadu jeho očí. Oči sú sledované špeciálnymi okuliarmi Pupil Core, ktoré sledujú pohľad hráča za pomoci infračervených kamier. Samotná hra je implementovaná v hernom engine Unity a komunikácia medzi okuliarmi a hrou je realizovaná pomocou lokálneho serveru. Výsledkom je 3D hra bojového žánru s mechanikou mierenia pomocou pohľadu.

Abstract

Aim of this thesis is to create a 3D fighting game in which, a player is able to aim via their eye gaze. Eyes are tracked using special glasses Pupil Core, that track players eye gaze using infrared camera. The game itself is implemented in a game engine Unity and communication between game and glasses is implemented via local server. The result is a 3D fighting game with an aiming mechanic using eye gaze.

Klíčová slova

Unity, Blender, 3D, C#, Pupil Labs, Pupil Core, sledovanie pohybu očí, bojový systém, mierenie očami, hra ovládana pohľadom, video hra, bojová hra

Keywords

Unity, Blender, 3D, C#, Pupil Labs, Pupil Core, eye tracking, combat system, eye based aiming, game controlled via eye gaze, video game, fighting game

Citace

KUČERA, Ján. *Herní bojový systém založený na sledování pohybu očí.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Chlubna

Herní bojový systém založený na sledování pohybu očí

Prohlášení

Prehlasujem že som túto prácu vypracoval samostatne pod vedením pána inžiniera Tomáša Chlubnu. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Ján Kučera
8. května 2022

Poděkování

Ďakujem pánovi Chlubnovi za ochotu a cenné rady pri vedení tejto práce.

Obsah

Kapitola 1

Úvod

Počítačové hry sa v priebehu posledných desaťročí stali neoddeliteľnou súčasťou zábavného priemyslu. Tento priemysel sa aktívne snaží udržovať krok s trendami ako sú žánre, mechaniky alebo grafický vzhľad. Preto hry, ktoré boli vydávané koncom minulého storočia alebo aj pred desiatimi rokmi sú často považované za zastaralé. Rovnako sa menia a posúvajú technológie a nástroje využívane na ich tvorbu. O niektorých z nich sa píše v kapitole 2, kde sa popisujú nástroje použité v tejto práci.

Nie je ale nutné príjsť na trh vždy s niečím úplne novým, väčšinou stačí obohatiť už existujúcu formulu o niečo nové a moderné. Môže to napríklad byť spojenie existujúcich žánrov do nového žánru, ako v prípade žánru Battle Royale, alebo môže ísť o pridanie nového typu vstupu pre ovládanie hry. A práve k niečomu takému pred pár rokmi došlo.

Šlo o periférie, ktoré umožňovali využiť oči hráča, konkrétne sledovanie pohľadu hráča a kam sa hráč pozerá, ako vstup pre ovládanie hry. Daná mechanika rozšírila možnosti rôznych hier a tým spestrila herný trh, a mnohé známe tituly od veľkých spoločností ako napríklad Ubisoft alebo Square Enix, tento typ vstupu a s ním spojené mechaniky aktívne implementujú do svojich hier. Tento vstup sa ale väčšinou implementuje do hier, v ktorých figuruje pohľad z prvej osoby, hlavne strielačiek alebo do hier kde slúži iba na interakciu s UI hry. A to sa pokúša zmeniť práve výsledný produkt tejto práce.

Táto práca má za cieľ vytvoriť hru, ktorá spája klasický žáner bojových hier ako je napríklad Street Fighter alebo Tekken spolu s mechanikou mierenia na základe vstupu z periférie pre sledovanie pohľadu hráča. Nakoľko daný žáner nemá na aktuálnom trhu zástupcu, ktorý by sa o niečo také pokúšal, návrh a implementácia tejto hry bude viac závisieť na aktívnom testovaní než na výskume už existujúcich riešení. O návrhu tejto práce sa píše v kapitole 3. Popisujú sa tu mechaniky, ktoré je nutné implementovať v rámci žánru a to ako sú tieto mechaniky ovplyvnené mechanikou mierenia. V kapitole 4 sa popisuje ako boli realizované viaceré časti práce a aké problémy v niektorých z nich nastali. V kapitole 5 sa vysvetlí metóda testovania vyhodnotí sa použiteľnosť danej mechaniky mierenia a či hra ako celok pozitívne rozšírila bojový žáner hier.

Kapitola 2

Teória

Táto kapitola obsahuje úvod k bojovým hrám a opis nástrojov a technológií použitých v tejto práci. Na začiatku v sekcii 2.1 je opísaný bojový žáner videohier a jeho mechaniky. V sekcii 2.2 sú popísané všeobecné vlastnosti herných enginov a enginu Unity ktorý je v tejto práci použitý. Obsahuje definície pojmov používaných v Unity a opis užívateľského rozhrania (ďalej UI). Sekcia 2.3 popisuje tvorbu modelov. Opisuje program Blender, jeho UI pre tvorbu 3D modelov a animácií. Posledná sekcia 2.4 sa venuje základom sledovania očí a ich pohybu. Následne obsahuje popis hardwaru použitého v tejto práci a to okuliare Pupil Core, od firmy Pupil Labs.

2.1 Bojové hry

V týchto hrách bojuje hráč proti oponentovi, ktorý je ovládaný počítačom alebo iným hráčom. Hráči vidia seba aj svojho oponenta z bočného pohľadu, nezávisle na tom či je hra 3D alebo 2D. Úlohou hráčov je poraziť súpera, znížením jeho životov na nulu, za pomoci rôznych kombinácií úderov a kopov. Jednotlivé údery sú rozdelené podľa množstvo zranenia ktoré spôsobujú na ľahký, stredný alebo ťažký úder, kde ľahký úder ma najnižšie zranenie a najvyššiu rýchlosť. Každý úder ma pevne daný smer a cieľ zásahu. Príkladom takýchto hier je Street Fighter alebo Mortal Kombat. Na obrázku 2.1 sa nachádza ukážka týchto dvoch hier. Táto práca rozširuje tento bojový systém o možnosť mierenia jednotlivých úderov pomocou pohľadu hráča, čím každý úder získa možnosť zasiahnuť oponenta na rôznych miestach, čo má za následok závislosť zranenia aj na mieste zásahu, nie len zdroja úderu [3].

Konkrétne mechaniky tohto žánru obsahujú rôzne obmedzenia alebo požiadavky na ovládanie a vstup. U pohybu sa očakáva pohyb vľavo a vpravo, skok, čupnutie a tzv. dash, ktorý umožňuje krátky rýchly pohyb vpred alebo vzad. Bojový sýstém okrem delenia útokov podľa sposobeného zranenia ešte musí podporovať mechaniku reťazenia úderov, ktorá po stlačení konkrétnej sekvencie tlačidiel, umožní vykonať takzvané kombo. Údery v kombe umožňujú sposobiť väčšie poškodenie a majú špecifickú choreografiu ktorú nie je možné dosiahnuť klasickými údermi a kopmi. Okrem toho sú dostupné špeciálne pohyby, u ktorých ide o stlačenie špecifickej kombinácie tlačidiel (môže ísť aj o sekvenciu tlačidiel ako u komb), ktoré umožnia vykonať útok spôsobujúci vysoké poškodenie. Niekedy je vykonanie tohto typu útoku obmedzené nejakými požiadavkami, napríklad množstvom dostupnej energie, ktorá sa minie pri použití špeciálneho útoku, nezávisle na tom či bol útok zablokovaný oponentom. Súčasťou boja je aj blokovanie úderov, ktoré neguje poškodenie väčšiny útokov.

(a) Ukážka z hry Street Fighter.[1]

(b) Ukážka z hry Mortal Kombat.[2]

Obrázek 2.1

Niektoré útoky dokážu spôsobiť zranenie aj napriek tomu že oponent blokuje úder. Blokovanie je zároven závislé na tom či postava stojí alebo je v drepe, v stoji blokuje len údery ktoré nesmerujú na nohy, v drepe blokuje údery ktoré nesmerujú na trup a hlavu (napríklad pri skoku alebo údere do hlavy ak útočník stojí). Na obrázku 2.2 sa nachádza ukážka z hry Street Fighter V, kde je vidiet realizáciu mechaniky špeciálneho útoku a blokovania.

2.2 Herný engine

Herný engine je balík funkcií a metód ktoré sú určené na uľahčenie tvorby videohier pre vývojárov. Ponúkajú základné prvky ktoré sú v hrách opakovane vyžadované, ako napr. simulácia fyziky, spracovanie kolízií, manažment zvukov, prehrávanie animácii, tvorba UI, rendering, sieťová komunikácia a ďalšie. Herný engine môže byť vo forme frameworku alebo samostatného programu ktorý umožňuje vývoj hry cez vstavaný editor. Editor zabezpečuje aj kompiláciu zdrojových kódov [3].

[1] Prevzaté z https://us.streetfighter.com/media/
[2] Prevzaté z https://mortalkombat.fandom.com/wiki/Scorpion/Gallery

Obrázek 2.2: Ukážka zo hry Street Fighter V, kde postava Ryu (vpravo) blokuje špeciálny úder postavy Bison (vľavo).[3]

2.2.1 Unity

Unity[4] je herný engine vyvíjaný firmou Unity Technologies, ktorý slúži na tvorbu 2D a 3D hier pre väčšinu, dnes používaných platforiem pre hranie hier. Medzi tieto platformy patria mobilné zariadenia, počítače, herné konzoly, virtuálna realita a webové prehliadače. Pre tvorbu skriptov sa využíva jazyk C# [6]. Unity ponúka niekoľko verzií svojho enginu, bezplatné, Personal a Student, a platené, Plus, Pro a Enterprise. Platené verzie ponúkaju dodatočné servisy a prioritnú technickú podporu. Okrem tvorby hier sa Unity tiež používa pre tvorbu 3D animácii v oblastiach filmov, architektúry a inžinierstva. V nasledujúcich sekciách sú detailnejšie popísané objekty, komponenty a skripty s ktorými sa pracuje pri vývoji hier.

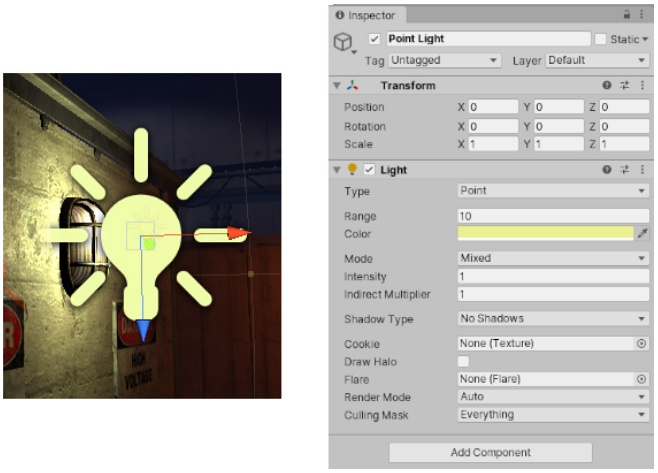
2.2.2 GameObject

Herný objekt je hlavný stavebný blok s ktorým vývojár pracuje. Každý objekt v hernom svete je herný objekt. Môže ísť o 3D model, 2D sprite, osvetlenie, kameru, elementy UI alebo Empty GameObject (ďalej prázdny objekt), ktorý obsahuje len skripty, prípadne iné komponenty, pre riadenie ostatných objektov, napr. aktualizovanie UI, prepnutie kamery alebo aktualizovanie údajov o objektov (životy nepriateľov, herný čas, počet nepriateľov, atď.). Každý herný objekt môže mať k sebe pripojené ďalšie herné objekty. To ovplyvňuje delegácia správ pri kolíziách, pozíciu potomkov a vyhľadávanie komponentov objektu v skriptoch.

[3] https://us.streetfighter.com/
[4] https://unity.com/

2.2.3 Component

Komponenty sú prvky ktoré umožňuju pridávať alebo upravovať chovanie herných objektov. Každý objekt má implicitne pripojený komponent *Transform*, ktorý nejde odstrániť. Určuje jeho pozíciu, rotáciu a mierku v hernom svete. Komponenty často definujú o aký herný objekt ide. Na obrázku 2.3 je objekt s komponentom osvetlenia, čiže herný objekt osvetlenia. Objekty môžu obsahovať neobmedzený počet komponentov. Niektoré komponenty vyžadujú iné komponenty aby vedeli fungovať správne alebo úplne, napr.: komponent *Rigidbody* zodpovedný za fyziku a detekciu kolízií vyžaduje komponent *Collider*, ktorý definujú tvar objektu v rámci detekcie kolízií (nereprezentuje ich pomocou meshu ktorý hráč vidí).



Obrázek 2.3: Herný objekt s komponentom osvetlenia.[5]

2.2.4 Skripty

V Unity sa skripty kategorizujú ako komponenty, ktoré sa priraďujú k jednotlivým herným objektom [6]. Skript môže byť priradený ku viacerým objektom, pričom každý objekt ma vlastnú inštanciu daného skriptu. Každý vytvorený skript obsahuje predom vygenerovanú triedu pomenovanú podľa názvu súboru, ktorá dedí z triedy *MonoBehaviour*. Táto trieda ponúka základne metódy, pomocou ktorých sa upravuje chovanie herných objektov. Jednotlivé skripty je možné referencovať medzi ďalšimi skriptami v rámci jedného objektu alebo aj medzi rôznymi objektami. V prípade že viac objektov používa rovnakú referenciu skriptu, každý pracuje s rovnakou inštanciou daného skriptu. Každý skript obsahuje jednu triedu ktorá má rovnaký názov ako daný skript. Preto v tejto práci budú slová skript a trieda zameniteľné.

2.2.5 Užívateľské rozhranie Unity editora

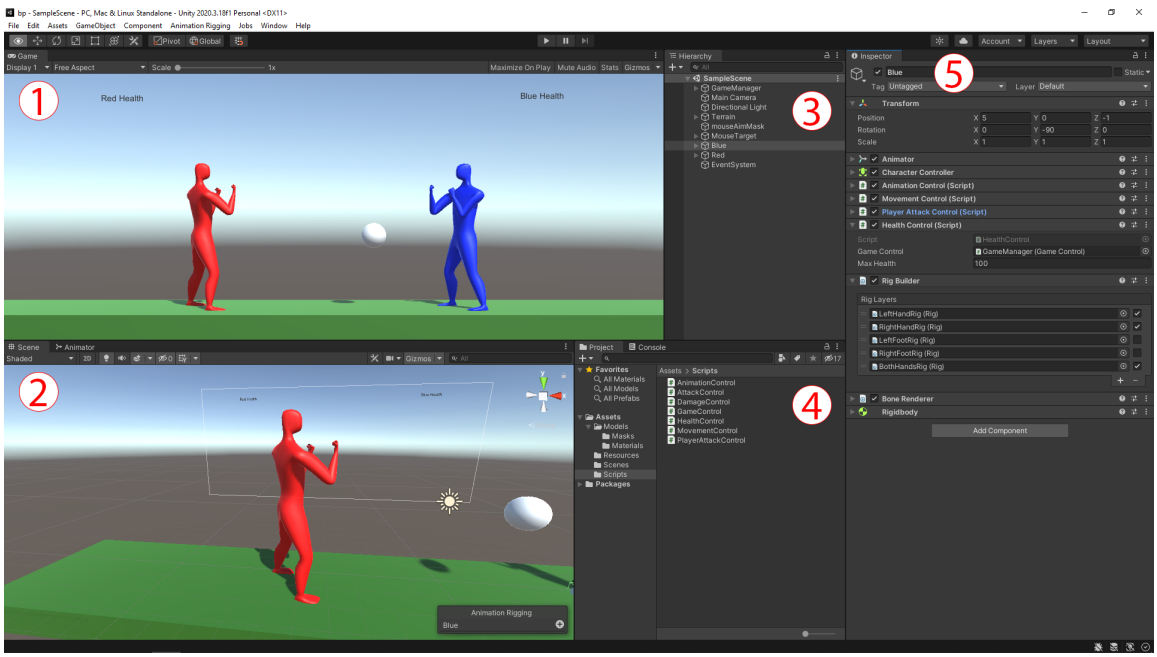
Editor, nachádzajúci sa na obrázku 2.4, je rozdelený na niekoľko častí, ktoré sú detailne popísané nižšie:

- **Game** - herné okno zobrazujúce aktuálny priebeh hry pri jej zapnutí. V rámci editora je možné hru spustiť, pozastaviť a vypnúť. Slúži to na testovanie aktuálneho stavu

[5] Prevzaté z https://docs.unity3d.com/Manual/

hry. Zmeny komponentov je možné vykonávať aj za behu hry ale po jej vypnutí sa zmeny stratia a je vyžadované ich vykonať keď je hra vypnutá aby ich editor uložil.

- **Scene** - okno scény, do ktorého sa umiestňujú herné objekty. Obsahuje aj vizuálne informácie ktoré sa nezobrazia v Game okne alebo finálnom produkte, ako napr.: collidery, osy objektov, ohraničenia UI elementov, atď.
- **Hierarchy** - obsahuje hierarchiu objektov v okne *Scene*. Umiestnenie objektov ako potomka iného objektu zjednoduší prístup ku komponentom rodiča zo skriptov potomkov. Zároveň sa delegujú detekcie kolízie potomkov na rodiča.
- **Project** - okno projektu zobrazuje hierarchiu externých súborov a zložiek. Ide o zdrojové kódy, modely, obrázky, atď. Odtialto sa súbory môžu pridávať do scény ako herné objekty alebo k objektom ako komponenty.
- **Inspector** - inšpektor umožňuje upravovať, prídávať a mazať komponenty herných objektov. Zároveň umožňuje meniť vrstvu, do ktorej objekt patrí a na základe toho s ktorými objektami má Unity detekovať kolízie, a tagy ktoré slúžia na označenie objektov čím sa zjednodušuje ich hladanie pomocou skriptov.

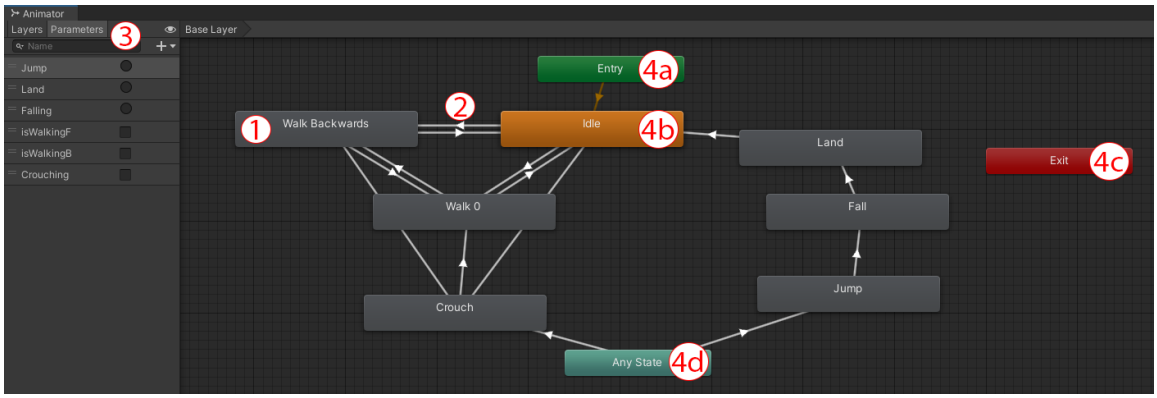


Obrázek 2.4: UI Unity editora. 1. Herné okno zobrazujúce hru ako bude vyzerať pri jej zapnutí 2. Okno scény do ktorého sa umiestňujú herné objekty. 3. Hierarchia objektov v scéne. 4. Obsah externých súborov a zložiek. 5. Inšpektor pre pridávanie, mazanie a úpravu komponentov jednotlivých herných objektov.

2.2.6 Ovládanie animácií

Pohyb modelov sa zabezpečuje animáciami. Unity poskytuje nástroje pre ich tvorbu, ale v tejto práci boli vytvorené v programe Blender, ktorý je popísaný v ďalšej sekcii 2.3.

V Unity pre ovládanie prehrávania animácií sa používa komponent *Animator*. Ide o komponent v ktorom sa vyberie tzv. *Animator Controller* (ďalej AC), ktorý spravuje animácie a ich prechody v rámci objektu. V AC sa vytvorí konečný automat (ďalej KA) v ktorom sú jednotlivé klipy animácií reprezentované stavmi medzi ktorými Unity prechádza na základe zmien parametrov, ktore sú priradené k jednotlivým stavom. Aby AC vedel manipulovať s kostrou modelu pomocou animácií, potrebuje mať štruktúru kostí vstupného súboru namapovanú na vlastnú štruktúru kostí, ktorá je reprezentovaná tzv. *Avatarom*. Avatar vie automaticky mapovať humanoidné kostry, ale v prípade iných kostier umožňuje manuálne mapovanie. Na obrázku 2.5 je ukážka konečního automatu v AC so stavmi použitými v tejto práci. Nižšie sú podrobne vysvetlené jednotlivé súčasti tohto UI:



Obrázek 2.5: UI Animatora. 1. Stav reprezentujúci klip animácie. 2. Prechod medzi jednotlivými klipmi. 3. Parametre podľa ktorých sa prechody rozhodujú. 4. Špeciálne stavy, ktoré sú unikátne: a - začiatočný stav automatu, b - stav do ktorého automat vždy prejde zo stavu Entry, c - konečný stav automatu, d - stav, ktorý symbolizuje prechod z ktoréhokoľvek stavu.

1. **State** - reprezentuje klip animácie. Stav prehrá svoju animáciu ak je aktívny. Je môžné nastaviť aby ju prehrával opakovane dokým neprestane byť aktívny.
2. **Transition** - prechod medzi dvoma stavmi, ktorý je podmienený hodnote aspoň jedného parametru. Smer šípky informuje o smere prechodu.
3. **Parameters** - premenné ktoré jednotlivé prechody kontrolujú. Majú rozne typy: Int, Float, Bool a Trigger. Trigger je špeciálny typ parametru podobný Bool, ale narozdiel od neho po nadobudnutí hodnoty True sa hned zmení na False. Používa sa napríklad pri animácii skoku, kedy sa po skoku animácia prepne automaticky do ďalšieho stavu.
4. **Special states** - ide o špeciálne stavy ktoré Unity vytvorí sama, nedajú sa odstrániť a sú unikátne. Sú farebne rozlíšené od bežných (šedých) stavov.
   (a) **Entry** - stav ktorý rozhoduje v ktorom stave KA začína. Je označený zelenou farbou. V prípade tohto projektu začína vždy stavom *Idle*.
   (b) **Default** - stav do ktoré sa automat snaží presunúť zo stavu Entry. Je označený oranžovou farbou.
   (c) **Exit** - stav ktorý zabezpečuje vstup späť do stavu Entry. V prípade že sa používa viacero KA, stavom Exit je možné sa vrátiť do predošlého automatu. V tom projekte nebol tento stav využitý. Je označený červenou farbou.

(d) **Any** - stav ktorým je možné zabezpečiť prechod do niektorého konkrétneho stavu kedykoľvek. Je možné vytvoriť prechody zo všetkých stavov do konkrétneho stavu, ale to môže zhoršiť prehľadnosť automatu. Je označený tyrkysovou farbou.

2.3 3D modelovanie

3D modelovanie je proces vytvárania 3D reprezentácie objektov. Môže ale nemusí ísť o reálne objekty. Povrchová reprezentácia je definovaná polygónmi ktoré sú definované vrcholmi a hranami. Tieto modely sa v dnešnej dobe využívajú vo videohrách, špeciálnych efektoch vo filmoch, architektúre, inžinierstve alebo marketingu. Pre tvorbu týchto modelov sa využívajú rôzne programy, napr. AutoCAD, Blender, 3DS MAX. V tomto projekte bol použitý program Blender [1].

2.3.1 Blender

Blender[6] je bezplatný open-source program určený pre tvorbu 3D modelov, rendering, tvorbu animácií, rigging modelov a editovanie videí. Je vyvíjaný neziskovou organizáciou Blender Foundation, ale k vývoju programu a dokumentácie prispieva aj komunita. Jediný platený servis sa volá Blender Cloud, ktorý umožňuje prístup k návodom a otvoreným projektom, ako napríklad filmy, ktorých autormi sú zamestnanci Blender Foundation.

2.3.2 Užívateľské rozhranie

UI sa delí na podobné časti ako u Unity, a to oblasť kde sa pracuje s modelom, hierarchiu aktívnych objektov a vlastnosti objektov alebo efektov v scéne. Na obrázku 2.6 sa nachádza okno *Modelling* v ktorom sa vytvárajú modely. Toto rozhranie je detailne popísané nižšie:

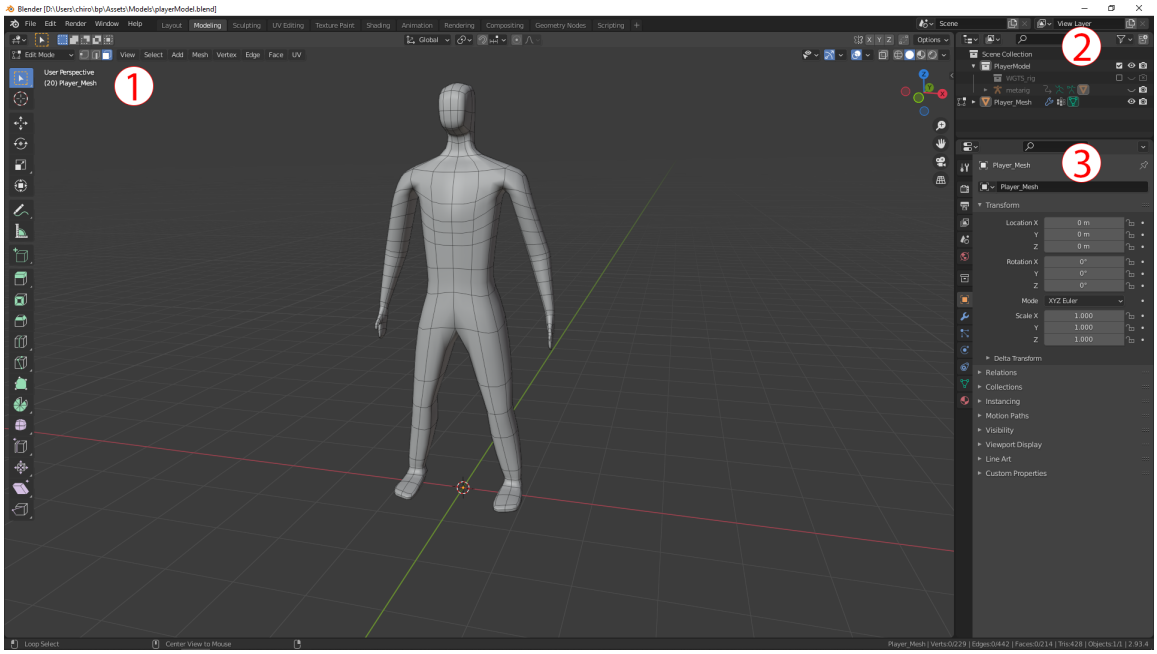
1. **3D Viewport** - hlavná oblasť vktorej sa pracuje s modelom. V tomto prípade okno Modelling obsahuje nástroje pre tvorbu modelov zo základných tvarov ako sú kocky, sféry, kapsula, atď.

2. **Outliner** - zobrazuje hierarchiu dát ako budu uložené v blend súbore. Ide o súbor zodpovedný za ukladanie dát 3D modelov, ktorý využíva program Blender. Na poradí niektorých objektov môže záležať pre korektné delegovanie pohybu kostí pri animáciách. Je možné jednotlivé dáta schovať pre lepšiu prehľadnosť (napr.: viac modelov v tesnej blízkosti).

3. **Properties** - časť poskytujúca úpravu vlastností hlavnej oblasti a modelov v nej. Okrem základných úprav ako zmena velkosti a mierky je možné aj pridať inverznú kinematiku ku kostre objektu, upravovať prepojenie kostí modelu, atď.

Časť editora zodpovedná za animovanie je popísaná nižšie. Na obrázku 2.7 je zobrazené toto okno *Animation*. Je vidieť že väčšina častí UI je zachovaná z predošlej časti a jednotlivé ponuky daných nástrojov sa nachádzajú na ľavej strane UI. Popis UI pre tvorbu animácii sa nachádza nižšie:

[6] https://www.blender.org/



Obrázek 2.6: UI pre tvorbu 3D modelov v Blenderi. 1. 3D Viewport zobrazujúci model na ktorom sa pracuje 2. Outliner zobrazuje hierarchiu objektov 3. Properties dovoľuje modifikovať a pridávať vlastnosti do sceny a jej objektov.

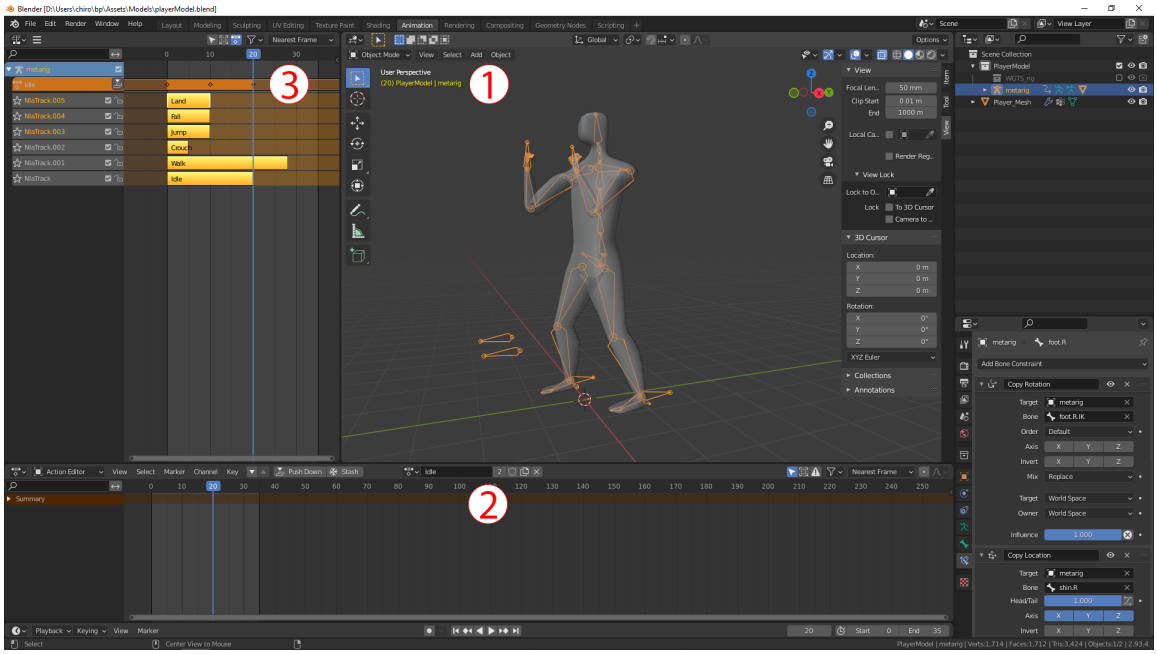
1. **3D Viewport (Animation)** - oblasť v ktorej sa manipuluje s modelom pomocou kostry. Polohy sa ukladajú do časovej osy. Zmeny môžu byť uložené len pre kosti ktoré vykonávajú pohyb v animácií, namiesto ukladania nového stavu pre celú kostru.

2. **Timeline** - časová os ktorá zobrazuje aktuálnu animáciu, tzv. (*Action*). Akcia v Blenderi obsahuje uložené pozície modelu. Jednotlivé polohy je možné aktualizovať, mazať alebo presúvať na časovej osy.

3. **NLA editor** - editor animácií pomocou ktorého sa anímácie upravujú bez manipulácie s konkrétnymi uloženými polohami. Využíva sa prevažne na zmeny ktoré sú jednoduchšie a nevyžadujú úplne zmeny polôh kostí.

2.4 Sledovanie pohybu očí

Technológia sledovania pohybu očí umožňuje sledovať kam sa užívateľ pozerá v reálnom čase na základe pohybu a polohy zreničiek. Oči sú monitorované pomocou webkamier, špeciálnych kamier s infračerveným svetlom alebo okuliarov ktoré disponujú infračervenými kamerami na ráme. Získané dáta po spracovaní sa používajú pre rôzne účely ako marketing, testovanie UI alebo vedecký výzkum [4]. Sledovanie pohybu očí sa využíva aj vo videohrách ako náhrada kurzoru myši, sústredenie kamery na objekt ktorý užívateľ sleduje alebo mierenie so zbraňou [7].



Obrázek 2.7: Editor pre tvorbu animácií v programe Blender. 1. Animation - oblasť v ktorej sa vytvárajú animácie 2. Timeline, časová os aktuálnej animácie 3. NLA editor - editor animáci pre menej komplexné zmeny animácií.

2.4.1 Blízké infračervené žiarenie

Pre sledovanie pohybu očí sa používa blízke infračervené svetlo namierené do jedného alebo oboch očí [2]. Toto svetlo sa odráža od rohovky a zreničky a je následne detekované infračervenou kamerou. Medzi týmito dvoma bodmi sa vypočíta ich vektor, vďaka čomu sa vie kam sa očí pozerajú. Tieto body sú viditeľné na obrázku 2.8. Využitie infračerveného svetla je preferované pretože viditeľné svetlo nedostatočne zvýrazňuje rozdieli medzi rohovkou a zreničkou, kvôli čomu dochádza ku veľkému poklesu presnosti. Zároveň infračervené svetlo nie je pre ľudské oko viditeľné, takže nesposobí rozptýlenie v čase snímania. Periférie používané pre tento typ sledovania pohybu očí sa delia na stacionárne a mobilné.

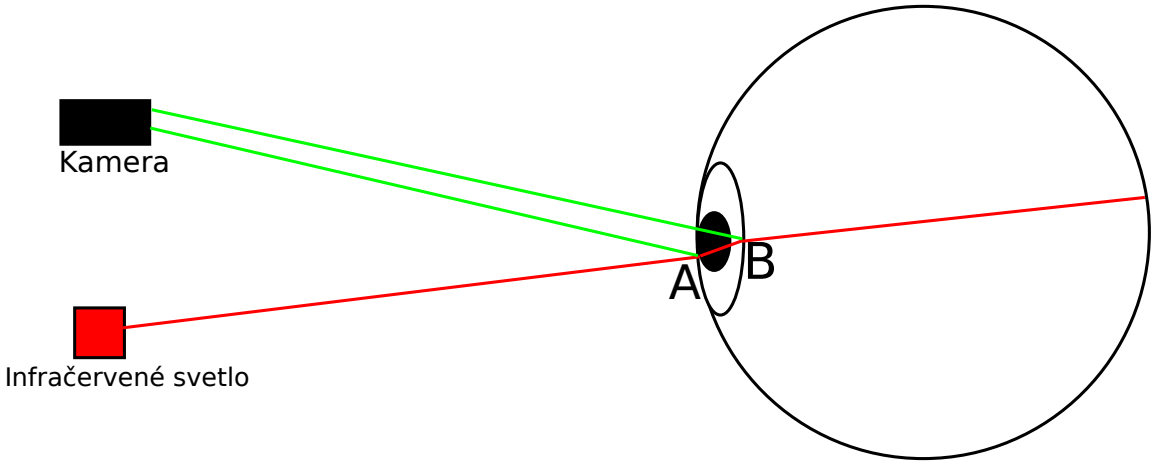
Stacionárne periférie sa nachádzajú v blízkosti monitora a počas snímania sa ich poloha nemení. Oči sú snímané iba v rámci určitej hranice, tzv. *headbox*. Zvyčajne sa používajú ak sa pohľad užívateľa sústreduje na monitor alebo nejakú obrazovku. Očakáva sa že užívateľ sa nepohne od danej obrazovky na príliš veľkú vzdialenosť.

Mobilné periférie sú vyhotovené v tvare okuliarov, čo umožňuje užívateľovi sa pohybovať. Zároveň sa umožňuje sledovať pohyb oči bez obmedzenia na displej, čiže je možné ho sledovať v rámci reálneho sveta. Používajú sa na sledovanie pohľadu objektov a pri úkonoch v reálnom prostredí (real-life) alebo virtuálnom, napríklad pri testovaní produktov. Na obrázku 2.9 sa nachádzajú oba typy periférií od firmy Tobii.

2.4.2 Pupil Core

Tento produkt[7] sa skladá z okuliarov pre snímanie údajov o pohybu očí a trojice programov ktoré tieto údaje spracovávajú (Pupil Capture, Pupil Service, Pupil Player). Okuliare dis-

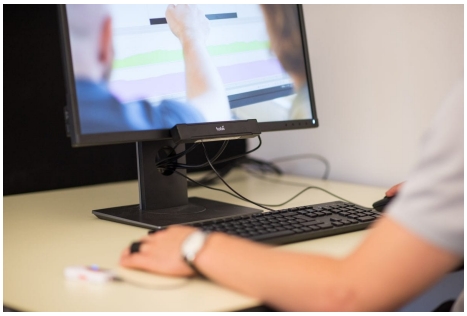
[7] https://pupil-labs.com/products/core/



Obrázek 2.8: Ukážka bodov na oku, od ktorých sa odráža infračervené svetlo a odrazené svetlo je detekované kamerou.

ponujú niekoľkými infračervenými kamerami a klasickou kamerou ktoré spolu snímaju očí užívateľa a to kam sa pozera a tieto dáta preberá program bežiaci na počítači ku ktorému sú okuliare pripojené. Program je open-source a umožňuje pridávanie funkcionalít vo forme pluginov alebo priamej úpravy zdrojového kódu. Pluginy sa píšu v jazyku Python a samotný program je napísaný Pythone a C++.

(a) Okuliare ktoré pri sledovaní pohybu očí umožňujú užívateľovi sa pohybovať.



(b) Kamera pre sledovanie pohľadu očí. Je nutné aby užívateľ bol v zornom poli kamery, takže pohyb je obmedzený.

Obrázek 2.9: Periférie od firmy Tobii, v stacionárnom (vľavo) aj mobilnom (vpravo) prevedení.[8]

2.4.3 Pupil Core software

Pupil Capture je hlavný program s ktorým okuliare pracujú, poskytujúci kalibráciu okuliarov a nahrávania snímaných údajov z okuliarov [5]. Zároveň poskytuje možnosť testovať presnosť pohľadu. Pri 2D mapovaní pohľadu by mala byť presnosť dostatočne podobná voči reálnemu pohľadu, zatiaľ čo u 3D mapovania môže dôjsť k nepresnostiam medzi 1,5 až 2,5 stupňa. **Pupil Service** je alternatíva k Pupil Capture určená pre virtuálnu realitu. Má

[8] Prevzaté z https://imotions.com/blog/eye-tracking-work/

zjednodušené UI a neobsahuje výstup z kamery sveta. Na obrázku 2.10 je zobrazené UI Pupil Capture, zobrazujúce čo vidí kamera sveta, ktorá sníma na čo sa užívateľ pozerá a pomocou dvoch sekundárných kamier každé oko v samostatnom okne. Naľavo sa nachádza panel s hlavnými funkciami kalibrácie (**C**), testovania presnosti predikcie pohľadu (**T**) a nahrávania pohľadu (**R**). **Pupil Player** umožňuje prehrávať nahrávky vytvorené v Pupil Capture a zvýraznovať rôzne informácie spojené s priebehom sledovania pohybu očí, ako trajektóriu pohľadu, teplotnú mapu alebo body fixácie.

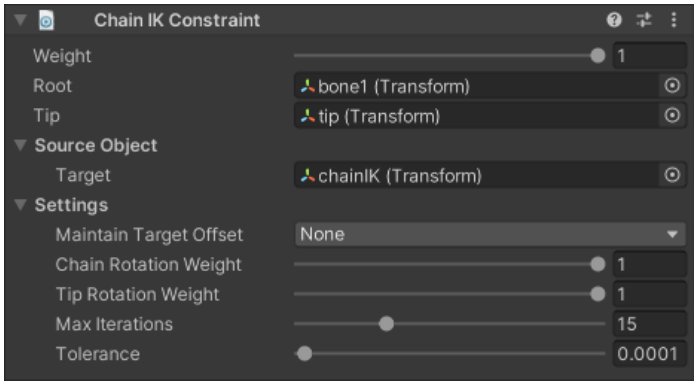
Obrázek 2.10: UI Pupil Capture s hlavnou kamerou sveta a sekundárnymi kamerami pre každé oko.

2.4.4 Animovanie mierených útokov

Mierenie daných úderov vyžaduje animovanie pohybu modelu na základe bodu kam sa užívateľ pozerá. Lenže samotné animácie nie sú samy o sebe upraviteľné a je nutné využiť iné spôsobi animovania za týmto účelom. Unity ponúka určité nástroje práve pre tento účel. K týmto nástrojom patria *Blend Trees*, ktorý zjednocuje viaceré klipy animácií, a *Animation Rigging*, ktorý manipuluje s kostrou modelu na základe iného objektu [6].

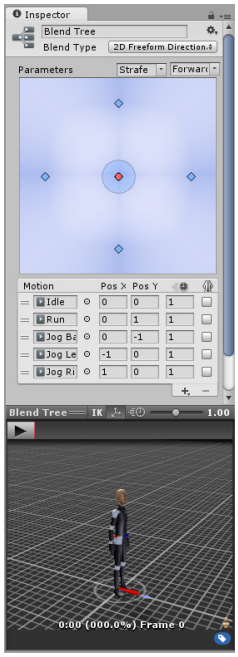
Animation Rigging je balíček od Unity ktorý je možné nainštalovať pomocou správcu balíčkov *Package Manager*. Tento balíček pridáva komponenty, ktoré umožňujú upravovať rotáciu a pozíciu kostí objektu na základe referenčnéhp objektu, vďaka čomu je možné manipulovať s kostrou modelu počas akejkoľvek prehrávanej animácie. Výhoda tohto nástroja je že odpadá nutnosť vytvárať dodatočné animácie pre jednotlivé útoky, nakoľko všetko je možné riešiť cez daný komponent. Zároveň je zabezpečená vysoká presnosť pohybov pretože cieľom pohybu je konkrétny objekt v hernej scéne, ktorého umiestnenie sa nastavuje priamo v kóde. Taktiež odpadá závislosť výsledného pohybu na existujúcich animáciach. Nevýhodou je nutnosť nadmernej práce s komponentom v kóde, hlavne ak ide o komplexné a dynamické pohyby. Ďalšou výhodou je že dané komponenty realizujú inverznú kinematiku, ktorá spravuje pohyb rodičovských kostí na základe pohybu kostí, ktoré sú potomkami v hierarchii kostri (napr. chodidlo alebo ruka). Na obrázku 2.11 je ukážka ako vyzerá komponent Chain IK, ktorý umožňuje delegovať pohyb medzi kosťami označenými *Root* a *Tip* smerom k referenčnému objektu *Source*.

Blend Trees je súčasť Unity, konkrétne súčasťou nástroja **Animator Controller**, ktorý umožňuje vytvárač prechody medzi animáciami. Slúži pre plynulé spojenie viacerých



Obrázek 2.11: Komponent Chain IK, ktorý deleguje pohyb kostí medzi dvoma kosťami pričom kosť Tip sa hýbe smerom k objektu Source a kosť Root nasleduje Tip.

animácií do jednej animácie. Výhodou je že tento prístup nevyžaduje manipuláciu objektov v hernom svete, iba zvolenie vhodných parametrov (pomeru váh jednotlivých vstupných klipov animácií). Nevýhodou tohto prístupu je nutnosť pre každy typ úderu, komba a špeciálneho úderu vytvoriť väčší počet animácií pre rôzne uhly začiatočného a cieľového bodu. Zároveň presnosť a plynulosť výsledných animácií je priamo závislá na množstve týchto animácií. Na obrázku 2.12 je ukážka ako vyzerá konfigurácia Blend Tree a jednotlivých animácií v ňom.



Obrázek 2.12: Blend Tree s piatimi animáciami, kde sa pomocou nastavenia bodov jednotlivých animácií dosahuje rôznej miery ich zmiešania. Dole sa prehráva ako animácia vyzera po ich zmixovaní. [7]

[7] Prevzaté z https://docs.unity3d.com/

Kapitola 3

Návrh

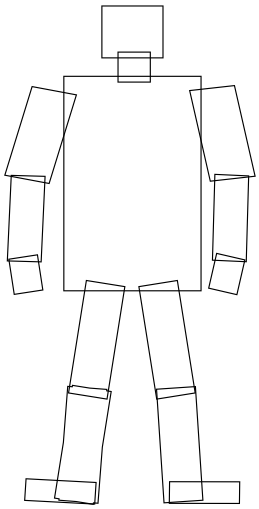
Táto kapitola popisuje návrh hry v Unity a jej prepojenia s okuliarmi Pupil Core. Prvá sekcia 3.1 sa zaoberá návrhom 3D modelu postavy, sekcia 3.2 požadovanými mechanikami bojového žánru a aké mechaniky musí obsahovať výsledná aplikácia. Tiež sa tu spomenie tréningový mód, v ktorom bude AI vypnuté. Sekcia 3.3 popisuje ako bude mechanika mierenia integrovaná do bojového žánru tejto hry. Popíše sa tu čo je nutné realizovať pre fungovanie tejto mechaniky a taktiež ako táto mechanika ovplyvní dizajn umelej inteligencie. V ďalšej sekcii 3.4 sa popisuje aký nástroj bol zvolený pre animovanie útokov. Tiež sa popíšu na aké veci sa treba sústrediť pri implementácii. V poslednej sekcii 3.5 sa popíšu požiadavky na umelú inteligenciu (ďalej AI) postavy riadenej počítačom, spôsob rozhodovania a typy reakcií na ofenzívu, prípadne defenzívu hráča.

3.1 3D model postavy

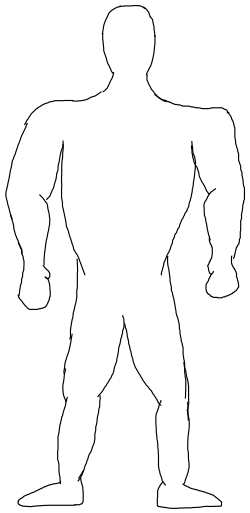
Je možné nájsť a použiť už hotové modely ktoré sú dostupné cez Unity v ich *Asset Store*, kde je možné získať rôzne hotové animácie, modely a skripty, či už zdarma alebo za určitý poplatok. V tomto prípade by ale bol problém nájsť modely s konkrétnymi animáciami, ktoré sú vyžadované v rámci mechaník popísaných v nasledujúcej sekcii 2.1. Preto sa samotný model vytvorý celý, v programe Blender a rovnako animácie budú vytvorené od základu, taktiež v programe Blender.

Pri návrhu modelu je nutné zohľadniť že ide o bojový žáner, preto je vhodné zvoliť humanoidné modely postáv, viz. 2.2. Kvôli mechanike mierenia musí kostra umožňovať vizuálne realistický pohyb s jednotlivými končatinami aby bolo možné realizovať útoky, ktoré nemaju statický cieľový bod. Z praktického hľadiska stačí vytvoriť 3D model skladajúcu sa z jednoduchých 3D objektov, konkrétne kvádrov. Tieto základne objekty sú preferované pre tvary ich tzv. *colliderov*, presnejšie *Box Colliders*, pre menej náročnú kalkuláciu kolizií zo strany Unity. Tieto kolízie sa využívajú pri detekcií úderov a následnom výpočte zranenia. Nakoľko ale hry v tomto žánry majú postavy ktoré vyzerajú ako ľudia, ako vidieť na obrázku nižšie 2.2, a nie len zhluky kvádrov, je vhodné upraviť výsledný model. To by ale spôsobilo zkomplikovanie jeho colliderov, nakoľko by už nešlo o jednoduché tvary a Unity by muselo využívať tzv. *Mesh Colliders*, ktorých detekcia kolízií je náročná na výpočet. Preto sa využijú obe formy výsledného modelu, pričom jednoduchý model, skladajúci sa z neupravených základných objektov, sa využije len na vygenerovanie *Box Colliderov*, ale bez meshu ktorý hráč vidí, a bude zodpovedný za detekciu kolízií. Upravený model si ponechá mesh ale nebude mať vygenerované *Mesh Collidery*. Týmto sa zabezpečia obe požiadavky,

štandardný vizuál hier v tomto žánry a jednoduchší výpočet kolízií v Unity. Na obrázku 3.1 sa nachádza náčrt oboch typov model.



(a) Náčrt modelu pre generovanie colliderov. Vidieť že sa skladá z jednoduchých objektov.



(b) Náčrt modelu s meshom. Model je detailnejší a nie je „hranatý“ ako model vľavo.

Obrázek 3.1: Náčrty oboch modelov. Vľavo model pre generovanie colliderov. Vpravo model s meshom.

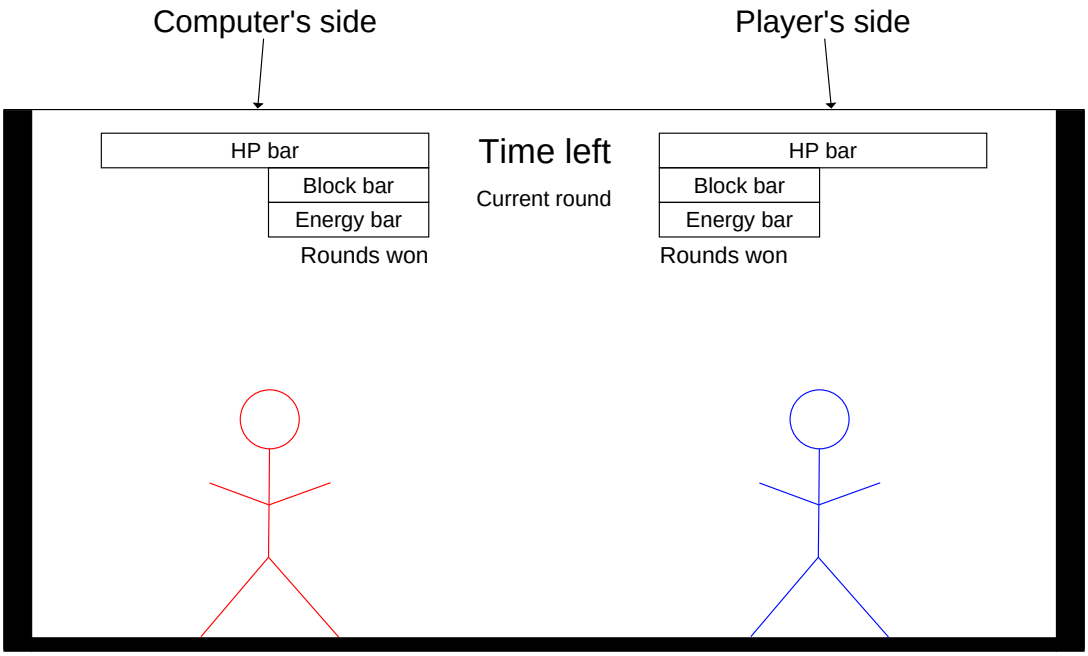
3.2 Demo aplikácia

Funkcionalita mierenia je testovateľná len v rámci aplikácie. Nakoľko táto mechanika je hlavným bodom tejto práce, je vhodné najprv implementovať len nutné mechaniky a aby bolo možné začať testovať presnosť a odozvu mierenia (komunikácia Unity s Pupil Core). Základne mechaniky ktoré sú vyžadované od herného dema sú pohyb, základné údery a blokovanie, bez komb a špeciálnych úderov. Ďalšie mechaniky mimo ovládania sú zobrazenie a sledovanie životov oboch postáv a ukončenie hry v prípade že životy jednej z postáv sa dostanú na nulu.

Vo výslednej aplikácii je vyžadované doplniť v rámci súbojov odozvu na zásah a to veľmi krátkou dobou (zlomok sekundy) kedy je hráč zasiahnutý a nemôže reagovať. Týmto sa zabezpečí nutnosť pri útoku sa snažiť zreťaziť čo najviac úderov pre maximálny efekt a zároveň možnosť okamžitej odvety ak útočník nechá oponentovi čas na zotavenie zo zásahu. Mimo súboja sú vyžadované aj ďalšie mechaniky ktoré nie sú priamo spojené s bojovým systémom, konkrétne časovač ktorý sa začne každé kolo, po ktorého uplnutí sa vyhlási víťaz kola s väčším počtom životov. Toto implikuje rozdelenie zápasov na kolá. Hráč musí zvíťaziť vo dvoch kolách aby vyhral zápas.

Vyššie spomenutá mechanika rozbitia obrany a špeciálne údery budú využívať zdroje, ktoré sa budú získavať aktívnym poškodzovaním. Vizuálne pôjde o indikátory postupu (ďalej progress bary), rovnako ako u bodov života, ktoré budú ubúdať so zranením. Progress bary pre rozbitie obrany (ďalej block bar) a špeciálne údery (ďalej energy bar) sa narozdiel

od progress baru životov (ďalej HP bar) budú dopĺňať. Energy bar sa bude napĺňať pri úspešnom zranení oponenta alebo úspešnom zasiahnutí oponentom. Toto doplnenie aj v prípade straty životov zabezpečí možnosť pre hráča ktorému sa nedarí pokúsiť sa o tzv. „comeback“. Samozrejme špeciálne údery sú blokovateľné, takže stále je comeback závislý na schopnostiach hráča. Rozbitie obrany zase môže nastať pri naplnení block baru, ktorý sa napĺňa len pri úspešnom zablokovaní útoku. Na obrázku 3.2 je plánovaný dizajn UI so všetkými vyššie spomenutými prvkami.



Obrázek 3.2: Návrh herného UI.

3.2.1 Tréningový mód

Okrem klasického módu je vhodné vytvoriť testovací mód ako je napríklad *Training Mode* zo série Street Fighter.[1] V tomto móde bude AI deaktivované, životy oboch postáv budú nekonečné a časovač bude pozastavený. Tento mód primárne slúži na zoznámenie sa s rôznymi typmi úderov, kombo úderov, špeciálnych úderov, ako fungujú rôzne mechaniky a celkové osvojenie si hry.

3.3 Mechanika mierenia

Bojové mechaniky spomenuté v predošlej sekcii sú v tejto práci obohatené o mierenie pohľadom. Mierenie sa bude realizovať pomocou okuliarov Pupil Core, ktoré bude mať hráč na sebe počas hrania hry. Údaje o bode pohľadu sa budú zasielať Unity a Unity ich bude periodicky aktualizovať. Následný vstup od hráča pre vykonanie úderu alebo blokovania

[1] https://streetfighter.fandom.com/wiki/Training_Mode

bude ovplyvnený koordinátmi tohto bodu. To znamená že útoky budú zasahovať oponenta tam kam sa hráč díva. Napríklad pri pohľade na rameno sa vykoná úder, ktorého cieľ je pozícia ramena. Ak sa po údere oponent rozhodne čupnúť a hráč sa stále díva na rameno a zaútočí, trajektória pohybu sa zmení nakoľko sa zmenil bod pohľadu aj keď pôjde o rovnaký typ úderu. V prípade obrany sa bude hráč brániť úderom z tej strany na ktorú sa díva.

Samotný bod pohľadu hráča by mal byť zvýraznený, nie len z dôvodu vývoja a testovania funkčnosti danej mechaniky ale aj z dôvodu že pri snímaní pohľadu očí môže dôjsť k problémom pri kalibrácii, ktoré môžu viesť k nepresnosti mierenia, ktoré nejde ovplyvnyť úplne. Takže je nutné informovať hráča zobrazením bodu pohľadu, ktorý detekovali okuliare aby sa vedel prispôsobiť prípadnej nepresnosti mierenia.

V prípade úderov sa očakáva možnosť zasiahnuť s každým typom úderu alebo kopu kamkoľvek v rámci jeho fyzického dosahu. Blokovanie s mierením umožňuje v drepe sa dostať do bodu kedy je hráč schopný blokovať úplne všetky údery z každej strany. To je problém, ktorý je riešený pridaním komb a úderov, ktoré sú neblokovateľné alebo mechaniky „rozbitia obrany“, kedy sa sleduje určitá hodnota, ktorá sa znižuje s každým zablokovaným úderom, a keď dosiahne hodnoty 0, hráč, ktorý blokoval je na moment bezbranný. Pre kombá a špeciálne útoky je potrebné aby sa choreografia pohybov dokázala zreťaziť počas meniaceho sa cieľu zásahu. Ale pre zachovanie plynulosti pohybu sa vyžaduje vytvoriť obmedzenia kedy sa cieľ pohybu už nezmení pre práve vykonávaný útok (prípadne sekvenciu komba alebo špeciálneho útoku).

AI bude realizovať útoky podobným spôsobom, kedy jednotlivé údery bude nutné namieriť na nejakú časť tela. Rovnako pri obrane bude musieť namieriť na určitú časť tela hráča ak sa bude chcieť brániť úderom z toho smeru. Tieto zmeny v mierení sa budú realivať okamžite, čiže nepôjde o plynulé zmeny, ktoré by simulovali postupné prechádzanie pohľadom po postave hráča. To samozrejme zabraňuje hráčovi predvídať kam AI zaútočí, preto sa pre vyrovnanie obtiažnosti nebude predvídanie implementovať ani u AI.

3.4 Animovanie mierených úderov

Ako bolo spomenuté v sekcii 2.1, animácie úderov majú pevne daný smer a cieľový bod. Tieto animácie sú realizované v Blenderi. Na to aby bolo možné mieriť jednotlivé údery je potrebné dynamicky meniť tieto animácie, konkrétne polohy a rotácie kostí v kostre modelu. Samotné animácie takéto zmeny neumožňujú, nakoľko každá animácia obsahuje len tie pohyby s ktorými bola vytvorená. Z toho dôvodu je treba zaistiť spôsob ako upravovať alebo vytvárať pohyb kostri nezávisle na aktuálne bežiacej animácii, pričom cieľ pohybu bude reprezentovaný prázdnym objektom, ktorého súradnice budu zodovedať súradniciam bodu pohľadu užívateľa.

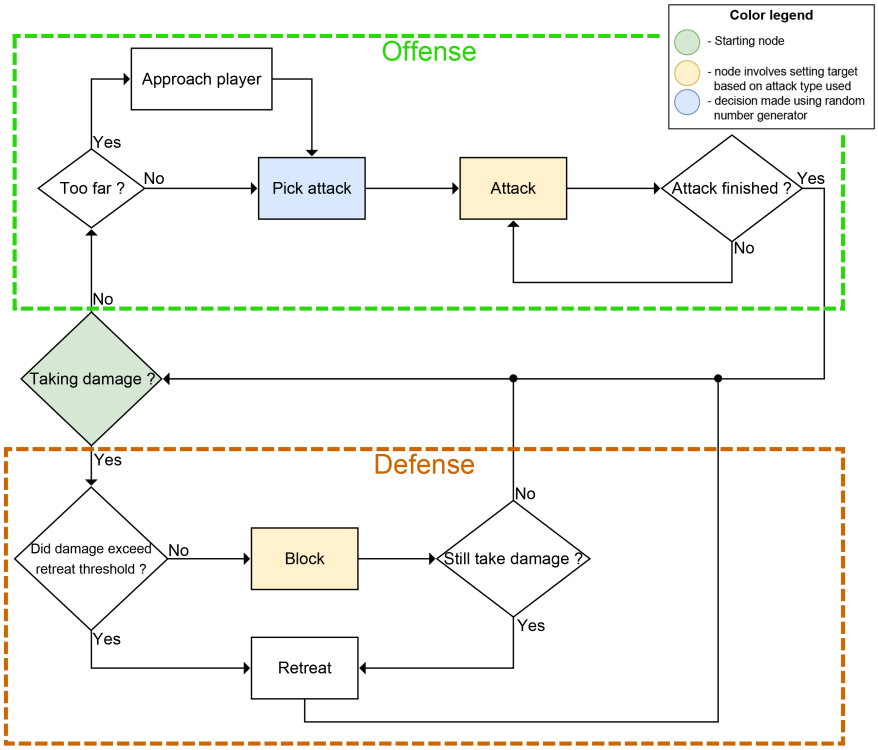
Na základe výhod a nevýhod spomenutých v kapitole teórie 2.4.4, bol zvolený balíček **Animation Rigging**, pretože s ním bude najjednoduchšie integrovať získavanie koordinátov pohľadu hráča s cieľovým bodom úderu, pomocou vyššie spomenutého Source objektu. Tento balíček bude vyžadovať vytvorenie skriptov pre priamu manipuláciu kostí modelu. Taktiež bude nutné zabezpečiť plynulosť daných pohybov ale zachovať určitú rýchlosť pohybov, nakoľko ide o bojový žáner a súboje v nich sú rýchle a dynamické. Tieto pohyby budú súčasťou kombo systému, takže je nutné sprístupniť jednotlivé metódy animovanie aj iným skriptov aby sa dal daný kombo systém centralizovať v rámci jednotlivých postáv.

Priebeh pohybov bude nutné monitorovať a ukončovať na základe nejakých kritérii, ako napríklad trvanie pohybu alebo vzdialenosť pohybu od nejakého bodu. Tieto kontroly sú nutné pretože nepôjde o klasické animácie, ktoré si tieto dáta uchovávajú v sebe. Taktiež

bude treba zaobstarať spätnú väzbu na zásahy a správne reagovanie na vstup hráča pri kombe (vstup počas animácie útoku by nemal mať vplyv na jej priebeh).

3.5 Návrh umelej inteligencie

Nakoľko sa v hre bude bojovať proti počítačom kontrolovanému nepriateľovi, je potrebná implementácia AI, ktoré bude ovládať postavu počítača na základe toho čo sa deje v hernom svete. AI sa dá rozdeliť do defenzívnej a ofenzívnej časti. Chovanie defenzívnej časti bude závislé na počte stratených životov, mieste zásahu a vzdialenosti od hráča. Podľa toho ako veľa životov bolo stratených od poslednej kontroly životov, AI sa môže rozhodnúť prejst do stavu blokovania úderov, stavu vzdalovania sa od hráča alebo využitia mechaniky dashu spomenutej v sekcii 2.1. U ofenzívy je základ kontrola vzdialenosti od hráča, aby AI vedelo kedy môže útočiť. Samotné rozhodovanie o použití úderu je vhodné realizovať do určitej miery náhodne, no implementovať nejaké očakávateľné obmedzenia, ktoré môžu uľahčiť hráčovi jeho rozhodovanie pri boji s AI. Napríklad AI by sa malo snažiť reagovať na to či hráč je v drepe alebo nie, prípadne či posledný zásah zo strany AI bol úspešný, ak nie, malo by zmeniť cieľ úderu. Ak sa netrafí, malo by skontrolovať vzdialenosť od hráča. Rôzne stavy AI by nemali byť vždy ovplyvnitelné inými, napríklad, ak je AI v stave vzdalovania sa, nemalo by hned prejsť do ofenzívy, kým nemá istotu že je v bezpečí (buď je určitú vzdialenosť od hráča alebo trvanie vzdalovania sa dosiahlo stanovenej hodnoty), naopak, ak sa nachádza u steny, už nemá kam ustupovať, takže by sa malo rozhodovanie obmedziť na ofenzívu a blokovanie. Pri rozbitej obrane by sa malo sústrediť primárne na ofenzívu. Na obrázku 3.3 sa nachádza návrh logiky AI vo forme diagramu.



Obrázek 3.3: Diagram návrhu logiky AI.

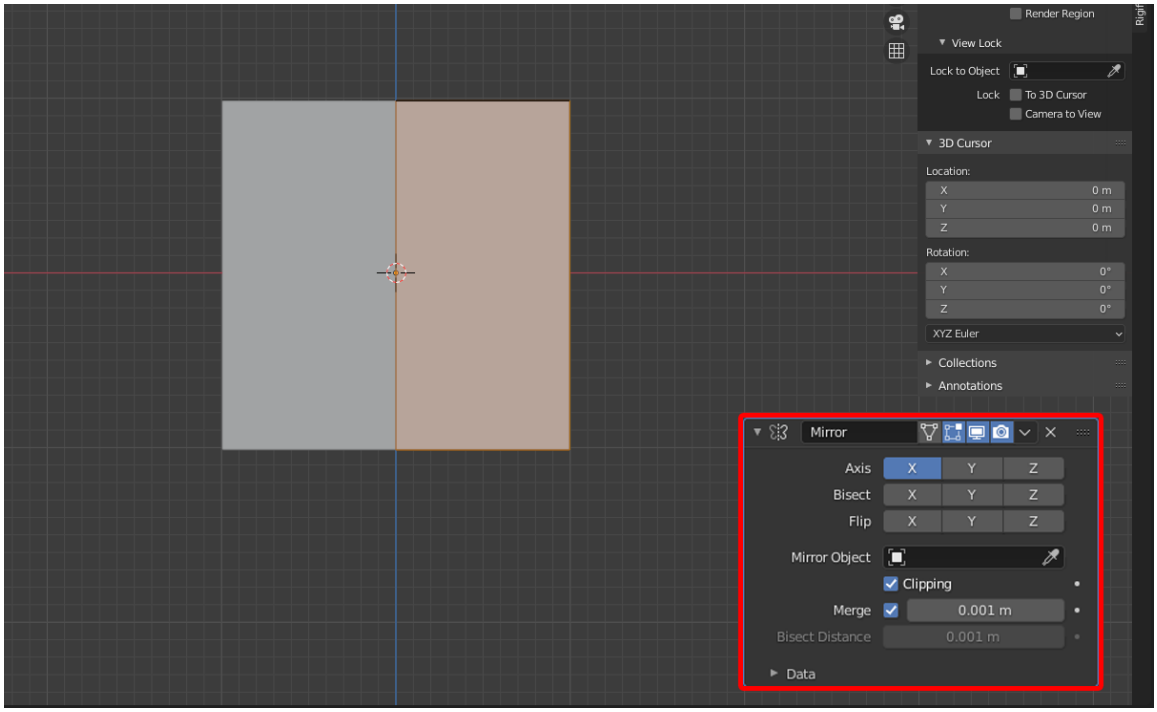
Kapitola 4

Implementácia

Táto kapitola sa zaoberá detailným popisom implementácie hlavných aspektov tejto práce, popisom konkrétnych technológií použitých v tejto práci a zároveň popisuje rozhodnutia, ktoré viedli k určitým spôsobom implementácie. V sekcii 4.1 sa vysvetlí proces tvorby 3D modelu a jeho animácií v Blenderi. Ďalej popisuje tvorbu colliderov a prácu s *Animatorom* v Unity. Ďalej v sekcii 4.2 sa vysvetlí tvorba základného pohybu a jeho prepojenia s animáciami modelu. Sekcia 4.3 obsahuje popis jednotlivých prvkov UI, ako je UI prepojené s jednotlivými údajmi hračov (životy, rozbitie obrany, atď), ako sleduje stav zápasu (časovač kola, počet víťazných kôl) a ako tieto údaje aktualizuje. Sekcia 4.4 opisuje bojový systém, ako sa spracováva vstup od hráča a počítača, ako sa realizuje strata životov a ostatných údajov postáv, ako sa detekujú kolízie v rámci súboja, a ako sa animujú jednotlivé údery. Mechanika mierenia sa popisuje v kapitole 4.5. Opisuje sa v nej riešenie mierenia v rámci Unity a ako sú potrebné dáta o pohľade získavané z *Pupil Core* a aktualizované v Unity. Taktiež sa spomenie problémy spojené s presnosťou mierenia. Posledná sekcia 4.6 vysvetluje implementáciu bojového AI. Popisuje sa tu rozhodnutia o tom kedy má prejsť do defenzívy alebo ofenzívy. Tiež sa vysvetlí ako funguje náhodnosť rozhodovania v prípade ofenzívy.

4.1 Tvorba 3D modelu

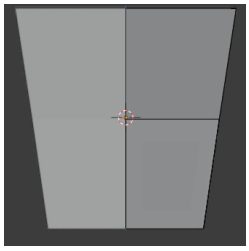
Ako bolo spomenuté v sekcii 3.1, výsledný model sa bude skladať z dvoch verzií. Naďalej budú tieto dve verzie označované ako **mesh model** v prípade verzie zodpovednej za mesh reprezentáciu a **collider model** v prípade modelu určeného pre generovanie colliderov. V prípade mesh modelu sa vygeneruje aj kostra modelu, pretože animácie budú vytvárané pre tento model. Collidery, ktoré následne v Unity budú vytvorené, budú priamo napojené na kostru mesh modelu, takže collidery sa budú hýbať spolu s jednotlivými kosťami ktorých pohyb je zabezpečený animáciami. U oboch verzií je vhodné zaistiť symetriu modelu. Preto sa využil modifikátor *Mirror* dostupný v Blenderi, v ktorom je možné si zvoliť na ktorej osy (je možné nastaviť viac ako jednu) má Blender produkovať symetrickú kópiu objektu a modifikácií, ktoré sú na modely vykonané. Na obrázku 4.1 je panel nastavení daného modifikátoru a kocka na ktorej sú farebne oddelené obe strany na ose X. Rozdelenie objektu v rámci osy sa začína na nulovom koordináte danej osy.



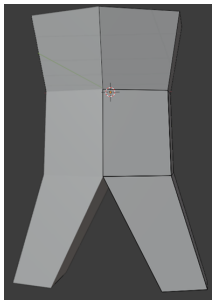
Obrázek 4.1: Objekt kocky z ktorého je vytvorený výsledný model. Ľavá a pravá strana na ose X sú farebne rozlíšení, pretože je zapnutý modifikátor Mirror. Vpravo dole v červenom ráme sa nachádza konfigurácia tohto modifikátoru.

4.1.1 Mesh model a Collider model

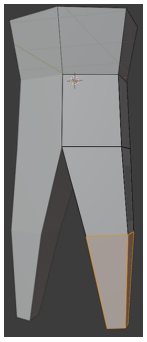
Mesh model bol vytvorený pomocou úpravy *extrude* (ďalej extrúzia), ktorá umožňuje rozšíriť body, hrany alebo steny 3D objektu. Extrúzia týmto spôsobom umožňuje vytvoriť komplikované modely za pomoci jedného základného tvaru, ktorý sa zvolí na začiatku, u tohto modelu šlo konkrétne o kocku. U všetkých novovytvorených stien, hrán a bodov je možné presúvať, rotovať alebo meniť ich mierku, nezávisle od zvyšku 3D objektu. Zároveň je možné pridávať hrany samostatne, čím sa vytvárajú nové oblasti, ktoré je ďalej možné modifikovať extrúziou. Na obrázku 4.2 je príklad toho ako sa dá použiť extrúzia na tvorbu končatín humanoidného modelu.



(a) Zúženie kocky v strede a následna extrúzia do úrovne pásu.



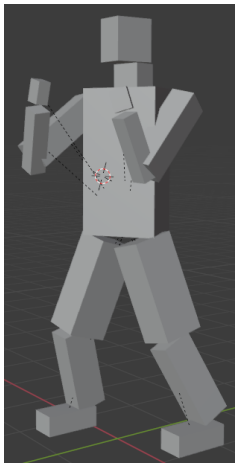
(b) Vytvorenie stehien extrúziou z pásu.



(c) Následná extrúzia spodnej časti nôh, so zúžením u začiatku chodidiel.

Obrázek 4.2: Tvorba trupu a spodnej časti tela pomocou extrúzie (začiatok zľava). Je zároveň vidieť že hrany sú zvýraznené len na ľavej strane modelov, nakoľko sa pracuje s modifikátorom *Mirror* a všetky úpravy na jednej strane sa prenesú symetricky na druhú.

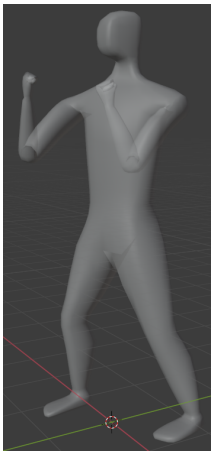
Collider model, použitý pre vygenerovanie colliderov bude, narozdiel od mesh modelu, vytvorený pomocou viacerých objektov. Vďaka čomu každá časť tela (trup, predlaktie, stehno, atď) bude mať vlastný objekt. Tieto objekty budú umiestňované v rámci mesh modelu, aby sa správne zachoval pomer velkostí jednotlivých častí tela. Pretože ale pôjde o základne objekty, nebudu úplne presné, to ale u tzv. *hitboxov* (collidery zodpovedné za detekciu zásahov v súboji) nie je problém. Na obrázku 4.3 sa nachádza ako tento model vyzerá. Každá končatina sa skladá z viacerých objektov, čím sa zabezpečí že pri detekcií úderov, bude možné nastaviť rôzne hodnoty zranenia pre rôzne časti tela.



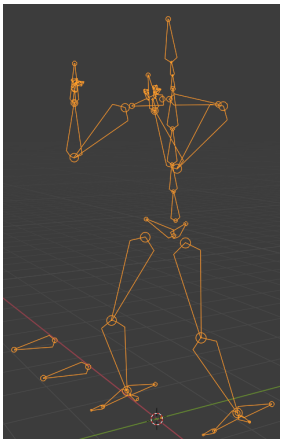
Obrázek 4.3: Finálna podoba modelu na základe ktorého sa vygenerujú collidery pre modely postáv.

4.1.2 Kostra u mesh modelu

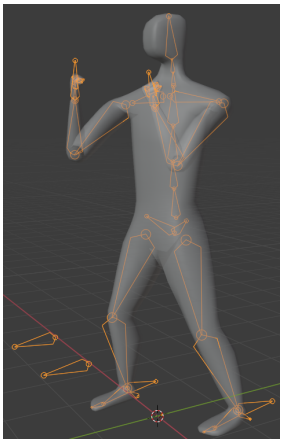
Tvorba kostry je možná manuálnym umiestňovaním jednotlivých kostí do modelu. Blender ale disponuje možnosťou vygenerovania kostry pre humanoidné modely, ktorá vytvorí kostru so správne usporiadanými kosťami. Nakoľko vytvorený mesh model nemá žiadne špecifické tvary, je vhodné tento nástroj použiť. Lenže táto kostra sa nevygeneruje presne pre model ktorý bol vytvorený, čiže je nutné upraviť mierku nielen celej kostry ale aj jednotlivých kostí. Následne je potreba správne umiestniť celú kostru v rámci celého mesh modelu a jednotlivé kosti v rámci individuálnych častí tela, aby niektoré kosti nezasahovali do inej časti tela, za ktorú má byť zodpovedná iná kosť, napr: kosť v krku by nemala zasahovať do hlavy, čím by vo výslednej animácii mohlo dôjsť k deformácii modelu pri pohyboch s krkom alebo hlavou. Nie všetky kosti vygenerovanej kostry sú nevyhnutné pre správne fungovanie animácií, konkrétne v tomto prípade, kosti v ruke reprezentujúce prostredník a prsteník boli odstránené, nakoľko mesh model nemá prsty. Následne je možné dokončiť tento proces napojením kostry na mesh model, označením kostry ako rodiča modelu alebo vygenerovaním tzv. *Advanced Rig* (ďalej pokročilá kostra), ktorý uľahčuje tvorbu animácií. U tohto modelu ale animácie neboli natoľko komplexné aby bolo treba vytvárať pokročilú kostru. Po prepojení kostry s modelom je možné začať vytvárať animácie modelu v Blenderi. Aj keď pokročilá kostra nebola využitá, je možné si uľahčiť tvorbu animácií pomocou tzv. **inverznej kinematiky** (ďalej IK), ktorá je popísaná v kapitole teórie 2.4.4. IK sa využilo pri animovaní nôh, nakoľko sú aktívne vo všetkých animáciach. Pre jednoduchšiu manipuláciu s IK, sa vytvorili pomocné kosti pred kolenami, ktoré delegujú pohyb k stehenným kostiam z kostí nižšie v hierarchii. U IK na rukách nebolo potrebné vytvoriť pomocné kosti, pretože ich pozície sa v animáciách vytvorených v Blenderi upravujú minimálne. Na obrázku 4.4 sa nachádzajú mesh model, vygenerovaná kostra modelu a následne ako model vyzerá s vygenerovanou kostrou.



(a) Hotový mesh modelu vytvorený v programe Blender.



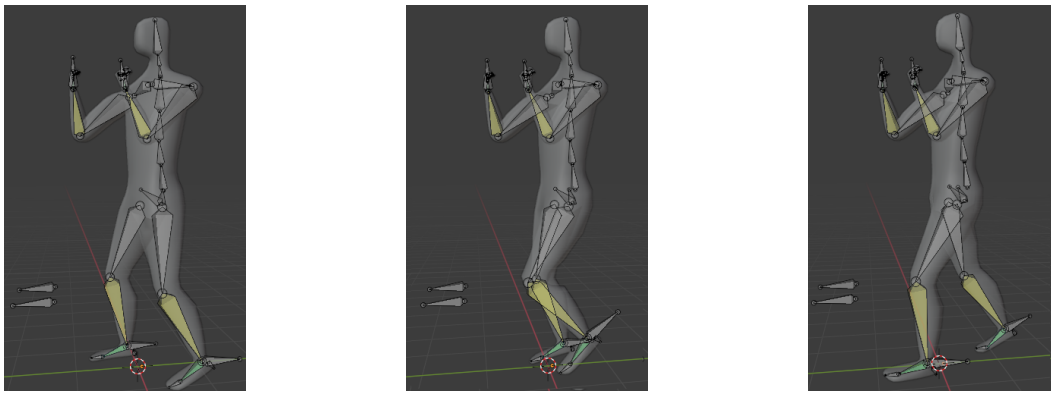
(b) Blenderom vygenerovaná kostra umiestnená v mesh modeli. Pred kolenami a na pätách a dlaniach vidieť pomocné kosti pre zjednodušenie animovania.



(c) Mesh modelu s kostrou umiestnenou v modeli. Takýto model už je možné animovať.

Obrázek 4.4: Mesh model, kostra daného modelu a mesh model s umiestnenou kostrou.

Animácie boli vytvorené pre státie, chôdzu, drep, skok, padanie a dopad. Anímácie úderov budú popísané v sekcii 4.5. Animácie sa skladajú z rôznych polôh modelu, ktoré sú uložené na časovej osy. Animácie vyžadujú rôzne množstvá pozícií aby boli dostatočne plynulé. Na obrázku 4.5 sa nachádza hotová animácia chôdze, ktorá je najdlhšia (29 snímok) a skladá sa zo siedmych pozícií. Prechod medzi jednotlivými pozíciami vykonáva Blender a čím bližšie k sebe jednotlivé pozície na osy sú, tým rýchlejší je daný prechod.



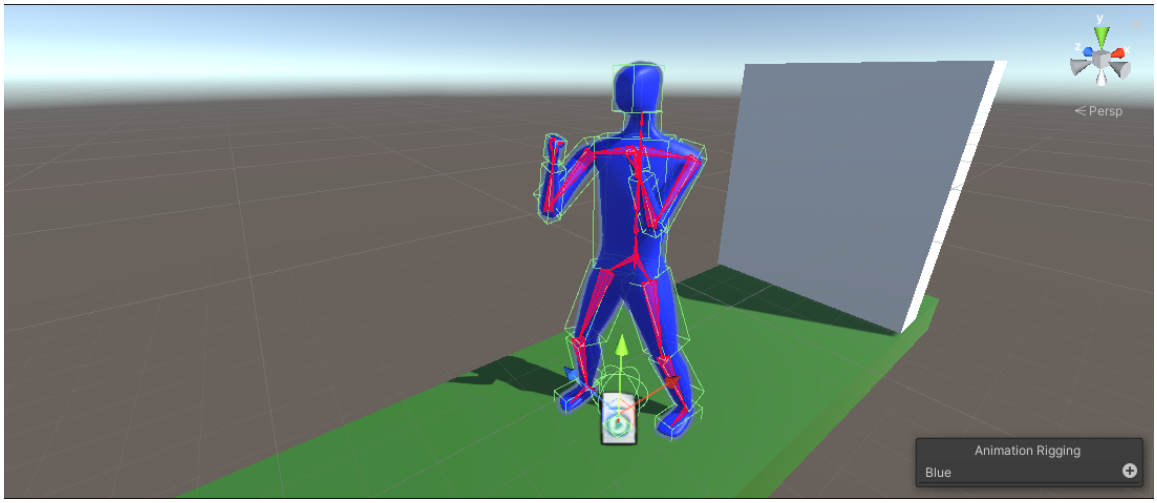
(a) Snímok č.1. (b) Snímok č.2. (c) Snímok č.3.

Obrázek 4.5: Snímky z animácie chôdze pre ktorej dostatočnú plynulosť sa vytvorilo sedem rôznych pozícií modelu.

4.1.3 Výsledný model v Unity

Oba modely sú následne importované a uložené do hierarchie ako potomok prázdneho objektu pre lepšiu prehľadnosť v rámci projektu v Unity. Potom sa umiestnia jednotlivé objekty collider modelu na individuálne kosti mesh modelu. Unity automaticky generuje mesh komponent pre jednotlivé objekty, čiže bolo nutné ich odstrániť pre každý objekt v rámci collider modelu. Potom sa pridal komponent *Box Collider* pre všetky tieto objekty. Na obrázku 4.6 je vidieť ako model vyzerá v Unity po vygenerovaní colliderov.

Do rodičovského objektu ktorý obsahuje modely je pridaný komponent *Animator* pre prácu s animáciami. Pomocou *Avataru* sa namapovala kostra modelu a vytvoril sa AC pre tvorbu KA. Oba boli následne pridané do Animator komponente. V AC sa ďalej pridali jednotlivé animácie, prechody medzi nimi a premenné, ktoré tieto prechody využívajú. V prípade animácie chôdze sa vytvorila kópia s negatívnou rýchlosťou prehrávania (hodnota -1), čím sa animácia prehráva pozadu. Tým sa zabezpečila chôdza vpred aj vzad. V AC jednotlivé stavy reprezentujú animácie, preto sa pri nasledujúcom popise budu tieto dve pomenovania využívať s rovnakým významom. KA začína v stave *Idle*. Stav *Idle* nevyžaduje žiadny vstup od hráča, pretože ide o predvolený stav, čiže je to animácia, ktorá je prehrávaná neustále, kým sa nedetekuje vstup od hráča alebo počítača. Z tohto stavu je možné prejsť do stavov chôdze: *WalkBackward* a *WalkForward*. Stav *Crouch* je dostupný z akéhokoľvek iného stavu, ale nie je možné sa pohybovať ani skákať kým je postava v tomto stave. Pri stave *Jump* sa postupne prejdú stavy *Fall* a *Land*. Tieto animácie nie sú priamo závislé vstupom zo strany hráča ani počítača. Po skoku (ktorý je závisly na vstupe) je ich prehratie podmienené parametrami, ktoré sú upravované pomocou skriptu, ktorý kontroluje status modelu (padanie, dopadnutie) a tým mení dané parametre. KA sa nakoniec dostane znova do stavu *Idle*. Na obrázku 4.7 sa nachádza KA, ktorý je využívaný postavami hráča a



Obrázek 4.6: Model hráča ako vyzerá v hernej scéne v Unity s hlavnými časťami modelu farebne rozlíšenými. Mesh model má modrú farbu, collidery sú zelené a kostra je červená.

počítača, pričom je na ňom vidieť prechod zo stavu skoku do stavu padania a podním je realizácia tejto animácie v hernej scéne.

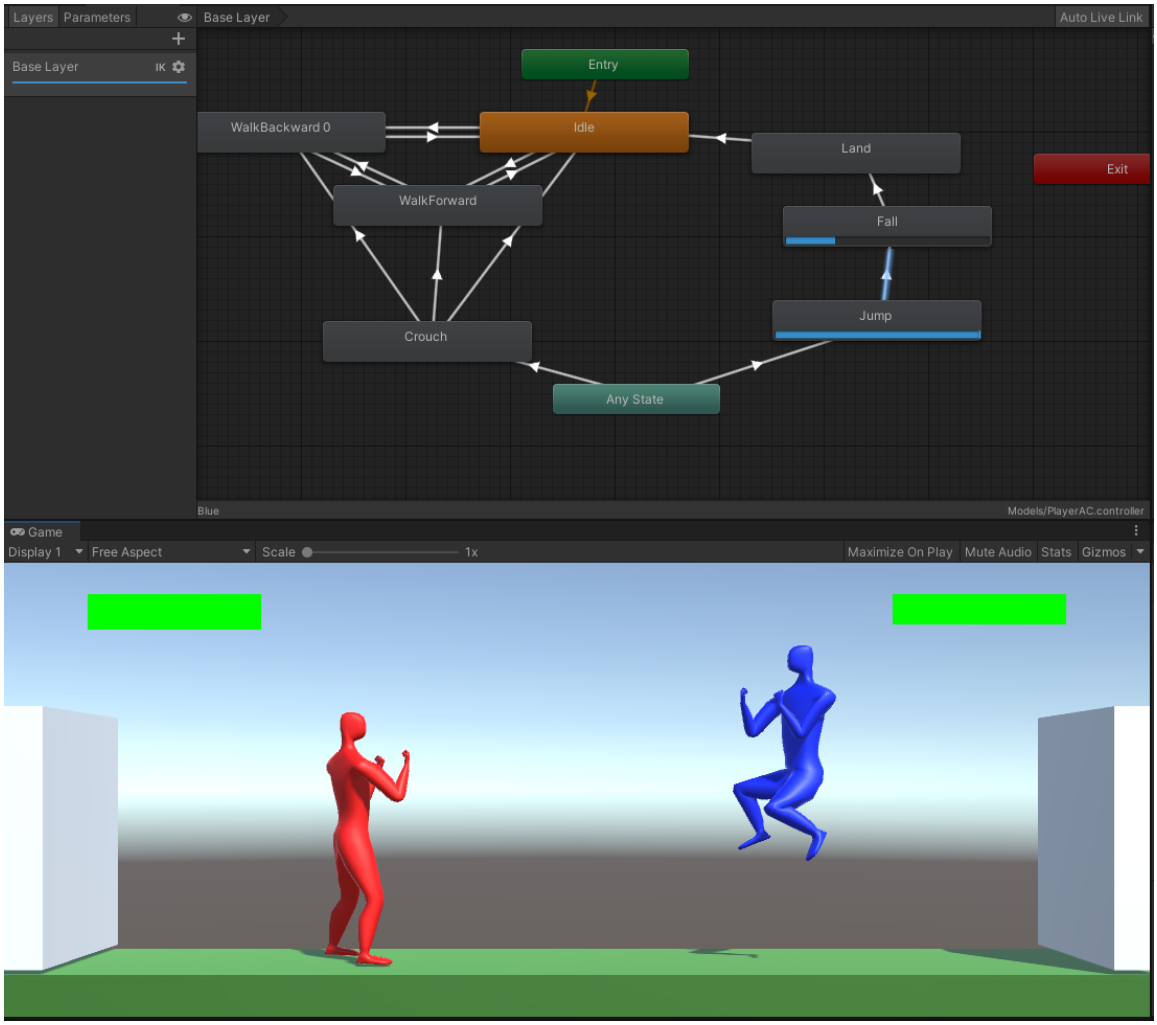
4.2 Základný pohyb

Hlavný skript pre pohyb je `MovementControl`, ktorý obsahuje abstraktné metódy (metódy, ktoré podobne ako u rozhraní nemajú telo a musia ho implementovať potomkovia triedy) pre základné úkony ako chôdza, skok, drep a dash a metódu `Move`, ktorá realizuje všetky dané úkony a kontroly pre ich správne vykonanie. Abstraktné metódy sa implementujú v skriptoch `MovementInput` a `MovementHandler`, ktoré z tohto skriptu dedia. V prípade hráča sa v nich kontroluje vstup a na jeho základe vykonávajú jednotlivé pohyby. V prípade postavy počítača sa v týchto metódach priamo vykonávajú dané pohyby bez kontroly. Dané metódy sú ale volané zo strany AI na základe rozhodovania v priebehu súboja (priblíženie a vzdialenie sa od hráča).

Metóda `Dash` nevykonáva samotný dash ale slúži ako verejná metóda pre kontrolu vstupu a následnom volaní metódy, ktorá dash vykoná. Implementácia tejto metódy v hráčskom skripte obsahuje kontrolu dvoch stlačení rovnakých tlačidiel (rovnaký smer) pre chôdu za sebou v danom časovom intervale. Pri stlačení sekvencie dvoch rôznych tlačidiel sa nič nestane. Takže dash je možné u hráča vykonať len pri tzv. „double tape“ (rýchle stlačenie toho istého tlačidla dvakrát za sebou) tlačidla pre chôdzu smerom vpravo alebo vľavo. V prípade počítača sa rovnako ako pri predošlích metódach, priamo volá metóda pre vykonanie daného pohybu bez kontroly.

Okrem abstraktných metód je v tejto hlavnej triede implementovaná metóda pre aplikovanie gravitácie, ktorá je nastavená na hodnotu -20. Táto hodnota je vyšše dvojnásobok než reálna hodnota gravitácie, a to z dovôdu príliš pomalého padania počas súboja.

Pre pohyb (chôdza, skok) sa využil komponent `Character Controller` (ďalej CC) namiesto `Rigidbody`, pretože v tejto hre sa nepožaduje akcelerácia na začiatku pohybu alebo simulácia fyziky objektu pri kolízii. S týmto komponentom sa pracuje priamo v skriptoch, konkrétne v metóde `Update`, ktorá je volaná každy snímok pre každý skript, ktorý ju obsahuje. Tento komponent vyžaduje nastavenie hodnôt pohybu pomocou premennej typu



Obrázek 4.7: Dole je ukážka skoku s modelom hráča, a v hornej časti je vidieť AC v ktorom sa dokončilo prehrávanie animácie skoku, a následný prechod do stavu padania a prehrávania tejto animácie.

`Vector3`, ktorý reprezentuje 3D vektory používane pre prácu s polohami a pohybmi objektov. Daný vektor je nutné vynásobiť výstupom metódy `deltaTime`, ktorá je prístupná z triedy Time [6]. Táto metóda vracia čas medzi aktuálnym a posledným snímkom. Túto hodnotu je nutné použiť z toho dôvodu, pretože volanie metódy `Update` zo všetkých inštancií skriptov, ktoré sú aktívne za behu hry, sa nevykonávajú vždy v rovnakom poradí a zároveň, každý snímok nemusí trvať rovnako dlho. Preto vynásobenie vektora touto hodnotou zabezpečuje, že jednotlivé zmeny v pohybe budú plynulé a nezávislé na počte snímkov za sekundu. Následne sa volá metóda `Move` objektu CC (nemýliť si s metódou Move v skripte MovementControl), ktorá následný pohyb aplikuje na herný objekt. Pri detekcii stlačenia tlačidla skoku sa zvýši vertikálna rýchlosť CC. Pri stlačení tlačidla drepu sa naopak aktívne nuluje horizontálna rýchlosť CC a nepovoluje sa skok, rovnako ako sa nepovoluje čupnutie kým postava skáče alebo padá. Preto sa drep kontroluje v rámci pohybu aj keď tento úkon postavou nepohybuje, no aj tak priamo ovplyvňuje jej pohyblivosť.

Mechanika dashu sa vykonáva v metóde `ExecuteDash`. Táto metóda využíva triedu `Coroutine` (ďalej korutina) [6]. Ide o špeciálny typ triedy v Unity, ktorý sa používa pre

vykonávanie metód v priebehu viacerých snímkov. Toto zabezpečuje plynulejšie vykonanie určitých akcií v rámci dlhšieho časového úseku. Taktiež sa týmto spôsobom dokáže simulovať paralelné spracovanie určitých úkonov, aj napriek tomu že dané korutiny sa vykonávajú sekvenčne. Na to aby sa metódy mohli vykonávať počas viacerých snímkov musia vracať typ `IEnumerator`[1]. Ide o typ ktorý umožňuje vracať hodnoty za pomoci kľúčového slova `yield`. Vďaka tomu si Unity vie zaznačiť kde v danej metóde sa skončilo vyhodnocovanie a pri ďalšom snímku pokračovať od daného bodu. Korutiny sa spúštajú metódou `StartCoroutine`, v ktorej sa jej aj odovzdajú parametre danej korutiny, ak ich vyžaduje. V tomto prípade daná korutina pravidelne volá metódu Move zodpovedajúceho CC počas doby dashu. Technicky teda ide len o klasický pohyb. Nakoľko sa ale využije niekoľkonásobne vyššia rýchlosť pohybu a neaktivuje sa animácia chôdze, výsledný pohyb je prudší, čiže vizuálne ide o dash.

Pre schopnosť detekovania kolízií v rámci pohybu sa využíva `Capsule Collider`, ktorého veľkosť a umiestnenie je možné upraviť. Nie je možné zmeniť typ collidera, nakoľko je súčasťou Character Controlleru a ten neumožňuje zmeniť jeho typ. Je ale možné zmeniť na akej „vrstve“ sa budú dané kolízie detekovať. Vďaka tomu CC vie detekovať kolízie len v rámci dotyku s terénom alebo s capsule colliderom oponenta, ale pritom ignoruje kolízie z collider modelu, ktorý sa využíva pre detekciu kolízií v rámci boja. Toto je problém z toho dôvodu že zásah do rôznych častí tela spôsobuje rôzne hodnoty zranenia ale collider CC detekuje kolízie len v rámci jedného veľkého collideru. Preto sa CC nepoužíva pre detekciu kolízii v rámci súboja.

4.3 Užívateľské rozhranie

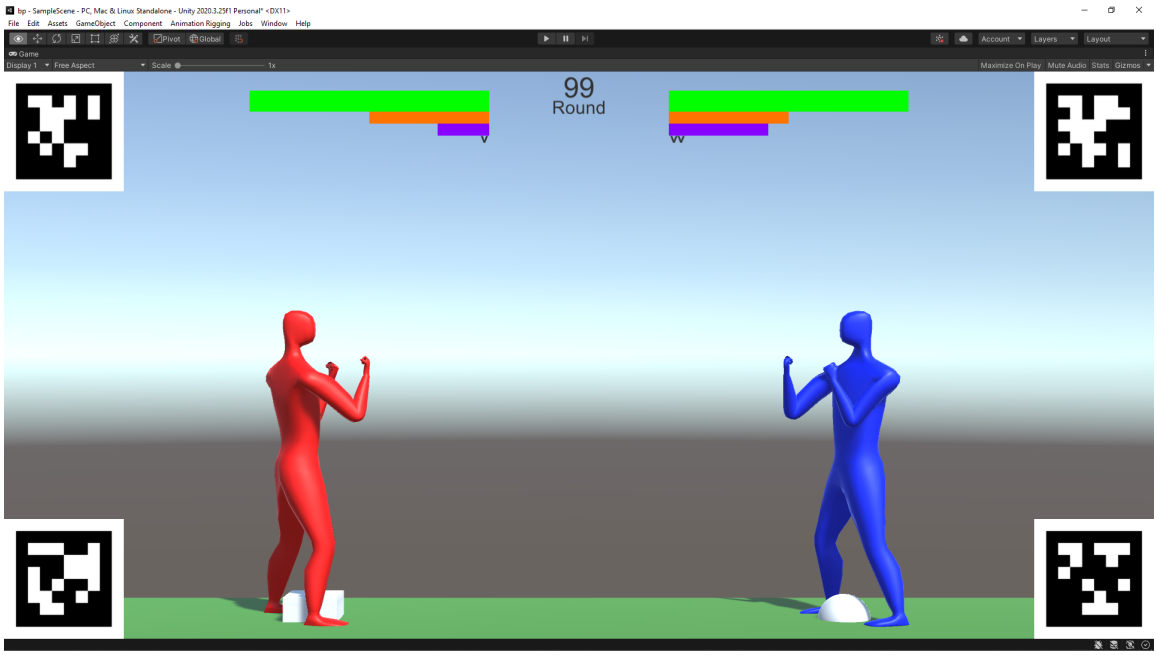
Pri tvorbe UI sa najprv vytvoril herný objekt `Canvas`. Tento objekt slúži ako wrapper, do ktorého sa umiestnili všetky UI prvky. Zároveň, Canvas zjednodušuje polohovanie týchto prvkov, nakoľko každý UI prvok je možné zakotviť do bodu na Canvase. Vďaka tomu sa jednotlivé prvky UI prispôsobujú veľkosti okna, napr. prvky vo vrchnej polovici obrazovky, ktoré vidieť vo finálnej podobe UI na obrázku 4.8, sú potomkami prázdneho objektu, ktorý je zakotvený v hornej časti obrazovky. Tým sa zabezpečilo že dané prvky sa budú vždy nachádzať nad postavami, kde budú najviac viditeľné.

Jednotlivé UI prvky sú herné objekty, ktoré sú priamo vkladané do hernej scény. Prvok časovača je textové pole, ktorého obsah sa aktualizuje každú sekundu. V prípade progress barov, ide o objekt typu `Slider`, ktorého výplň je objekt obrázku konštantnej farby. Obrázok ma svoje rohy zakotvené na rohy daného slideru, a slider následne upravuje šírku obrázku na základe hodnoty, ktorú prijíma zo skriptu `GameControl`. Počítadlo víťazných kôl sa skladá z textového poľa do ktorého sa postupne pridávajú písmená „V“, vždy keď dotýčná postava zvíťazí.

Samotná komunikácia UI so zvyškom objektov sa zabezpečuje v hernom objekte `GameManager`, v ktorom sa nachádza samotný `Canvas`. Tento objekt obsahuje skript `GameControl`, ktorého metódy sú určené na aktualizovanie UI prvkov, ku ktorým sa pristupuje pomocou referencií. Tieto metódy sú volané v prípade zmeny nejakej hodnoty, na ktorej je závislý nejaký prvok UI, napr. HP bar hráča je závislý na premennej s jeho počtom životov v skripte `CharacterStatus` a v ňom je cez referenciu na `GameControl` volaná metóda pre aktualizovanie HP baru s novou hodnotou vždy keď sa táto hodnota zmení pri zranení vyvolaného zo skriptu `HitDetection`.

[1] https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerator?view=net-6.0

Okrem UI elementov sa v Canvase nachádzajú tzv. **AprilTags**[2]. Ide o špeciálne značky, ktorými sa v Pupil Core označujú povrchy [5]. Tieto značky sú vždy umiestnené v rohoch obrazovky pomocou zakotvenia a sú reprezentované pomocou objektov typu `Raw Image`, ktorý je určený pre textúry alebo pozadia. Tieto značky sa nemenia počas behu hry a zostavajú celý čas zobrazené.



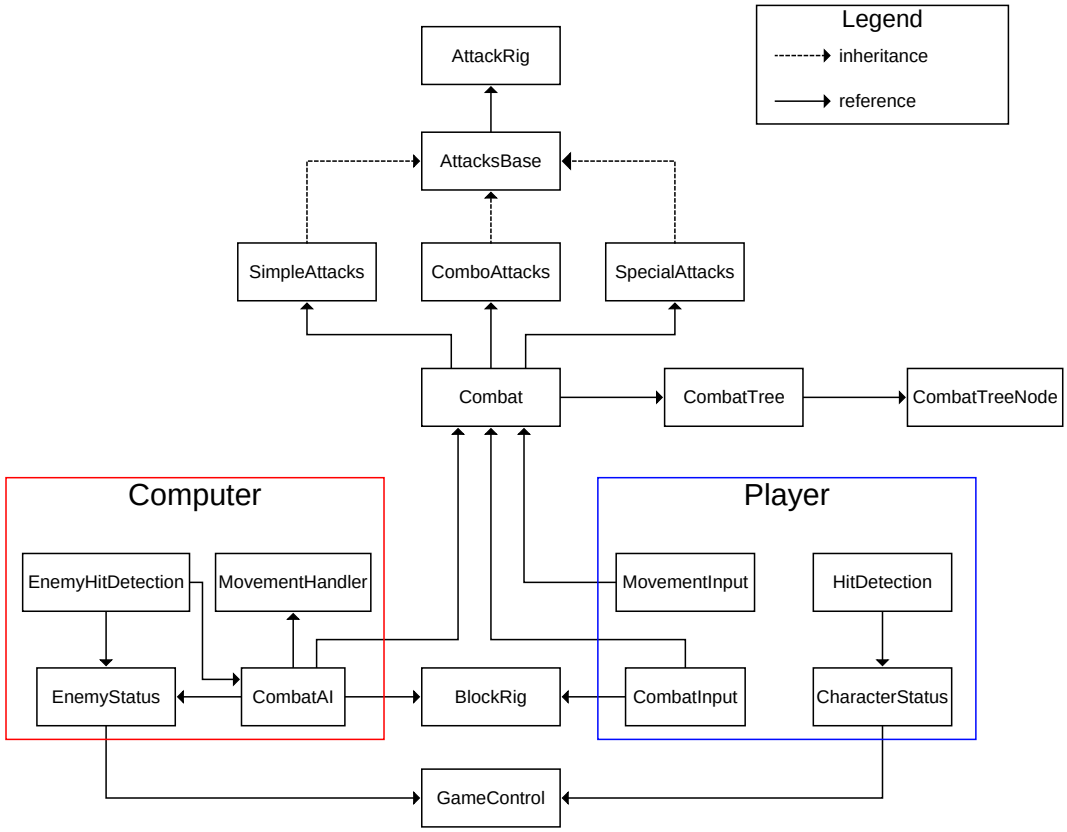
Obrázek 4.8: Výsledná podoba UI. Je vidieť že sa podarilo dosiahnúť plánovaného dizajnu 3.2. V rohoch sa nachádzajú April Tags, ktoré sú vyžadované okuliarmi Pupil Core pre korektné vyhodnocovanie pozície pohľadu hráča na obrazovke.

4.4 Bojový systém

Bojový systém sa skladá z dvoch hlavných častí. Prvá vykonáva detekciu zásahov a rieši výpočet zranenia medzi hráčmi a následne oznamuje túto skutočnosť GameManageru, ktorý aktualizuje UI. Druhá časť je zodpovedná za detekciu konkrétnych sekvencií vstupov a následnej realizácie základnych úderov a kopov, kombo útokov a špeciálnych útokov podľa aktuálnej sekvencie vstupov. Súčasťou tejto časti je aj animovanie úderov, pomocou metód, ktoré manipulujú s kostrou (ďalej rig) modelu. Čiže dané animovanie je dynamické, z dôvodov vysvetlených v kapitole návrhu 3.4. Diagram na obrázku 4.9 obsahuje vizualizáciu prepojenia jednotlivých skriptov v rámci tohto systému. Tento systém začína u vstupu hráča a počítaču. V prípade počítaču nie je nutné kontrolovať kedy sa pokúša o vykonanie nejakého úderu narozdiel od hráča, pretože podobne ako v prípade pohybu 4.2, počítač volá dané funkcie na základe rozhodovania AI v skripte `CombatAI`. V prípade hráča sa pomocou skriptu `CombatInput` pravidelne kontroluje vstup pre tlačidlá štyroch základných úderov a blokovania. Pri mierení pomocou myši sú dostupné len ľahký úder, ťažký úder a blokovanie. Po skontrolovaní vstupu hráča (prípadne po rozhodnutí AI zaútočiť), sa zavolá skript `Combat`, ktorý obsahuje referencie na skripty, ktoré sú zodpovedné za animovanie úderov.

[2] https://april.eecs.umich.edu/software/apriltag.html

Okrem nich obsahuje dátovú štruktúru stromu typu `CombatTree`, ktorý sa používa na vyhodnocovanie toho akú animáciu úderu treba prehrať na základe aktuálnej pozícii v strome. Pri zásahu, skripty `HitDetection` a `EnemyHitDetection`, ktoré sú pripojené ku všetkým colliderom na postavách hráča a počítača, volajú metódy zo skriptu `CharacterStatus`, respektíve `EnemyStatus`, ktoré aktualizujú hodnoty životov a bloku, zavolajú metódy skriptu `GameControl` a ten aktualizuje UI na základe týchto nových hodnôt. Prípadne ukončí kolo ak došlo k porazeniu jednej z postáv (počet životov je na nule, prípadne nižší než druhej postavy pri skončení časovača). Ak ide o druhé víťazné kolo pre niektorú z postav, ukončí sa celý zápas.



Obrázek 4.9: Diagram bojového systému, zobrazujúci ako jednotlivé skripty sú medzi sebou referencované.

4.4.1 Spracovanie sekvencií vstupov

Narozdiel od základných úderov, pri kombo útokoch a špeciálnych útokoch je nutné byť schopný detekovať správnu sekvenciu vstupov, na základe ktorých sa vykonávajú konkrétne útoky. To sa realizuje v triede `CombatTree`. Ide o strom, ktorého uzly reprezentujú jednotlivé typy úderov a útokov. Pre konzistentnosť systému, daná trieda obsahuje aj základné údery a sú reprezentované uzlami ktoré sú potomkami koreňového uzlu (ďalej koreň). V prípade koreňa a uzlov pre pohyb vľavo a vpravo nejde o uzly reprezentujúce útoky, čiže dané uzly

neobsahujú žiadne spustitelné metódy. Dané pohyby sú ale v hre klasifikované ako pohyby vpred a vzad, čiže vstup s nimi je závislí na pozícii postavy.

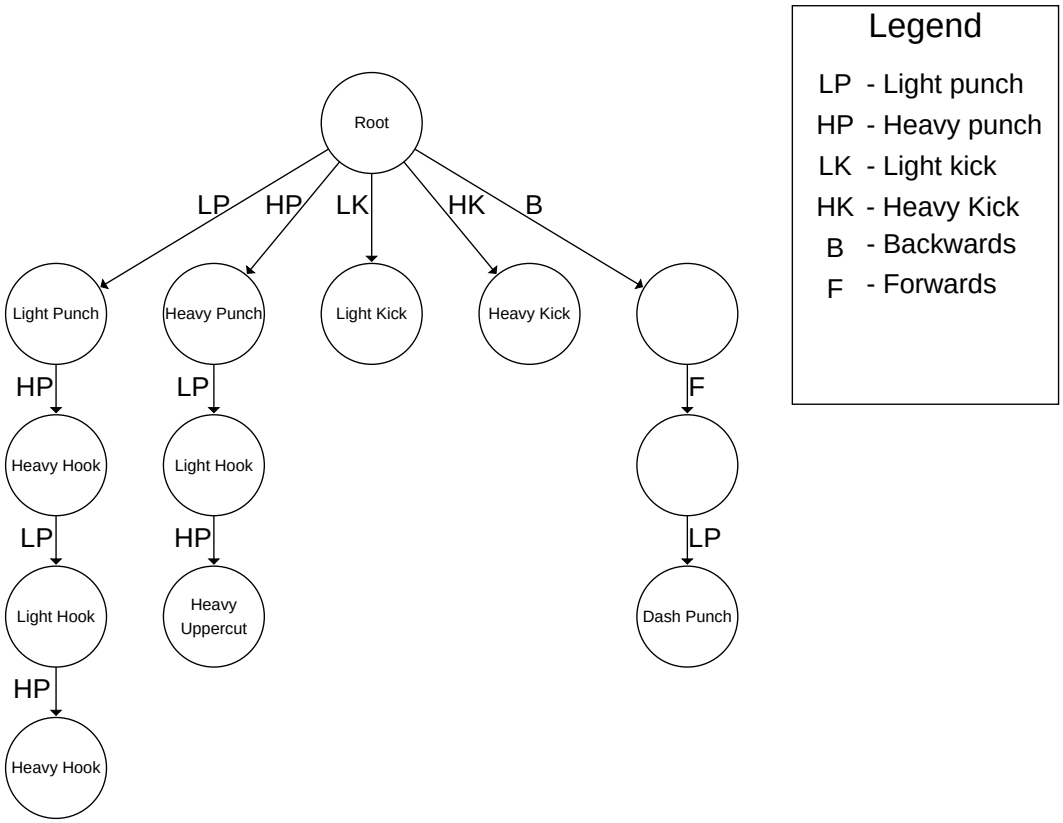
Jednotlivé uzly sú inštanciami triedy `CombatTreeNode`, ktorá obsahuje meno uzlu a metódu, ktorá sa má volať v prípade že sa strom dostal do daného uzlu, čo znamená že bola zadaná správna sekvencia. Metódy sú reprezentované typom `Action`, ktorý umožňuje ukladať referenciu na inú metódu, ktorá nevracia žiadnu hodnotu. V tejto premennej sa ukladajú verejné metódy skriptov, ktoré obsahujú jednotlivé typy úderov a útokov. Uzly taktiež obsahujú dátovú štruktúru `dictionary` v ktorej kľúč je typ úderu a hodnota je ďalší uzol (potomok). Takže dané uzly sa vyhľadávajú na základe práve zadaného vstupu, v prípade že sa daný uzol nenájde, prejde sa do koreňového uzlu a strom vykoná hľadanie znova, ak sa daný uzol nenájde ani v koreni, daný vstup sa ignoruje. Strom kontroluje či je postava na zemi, inak neprehľadáva všetky uzly ale vždy iba koreň, čím sa zabezpečuje vykonávanie iba základných úderov vo vzduchu. Zároveň strom kontroluje čas od vykonania posledného úderu, čo slúži pre vytvorenie časového okna v ktorom musia byť dané vstupy vykonané aby sa brali ako vstup pre kombo alebo špeciálny úder. Aktuálna vstupná sekvencia je reprezentovaná ako typ `Queue`, čiže rada, z ktorej sa postupne vyberajú jednotlivé vstupy a vyhodnocujú. Jednotlivé typy sú definované v pomocnom skripte `CombatUtilities` formou celočíselných konštánt. Inštancia stromu sa nachádza v skripte `Combat`, ktorý ho vytvorí a naplní všetkými typmi úderov. Daný skript je referencovaný vo vstupných skriptoch hráča a počítača, a využívajú ho pre odovzdávanie vstupných sekvencií pre vyhodnotenie stromom. Na obrázku 4.10 sa nachádza diagram daného stromu.

4.4.2 Detekcia zásahov a spracovanie zranenia

V Unity je nutné pre detekciu dotyku dvoch objektov využiť okrem komponentu collider aj komponent Rigidbody. Tento komponent je v Unity využívaný pre prácu s fyzikou a kolíziami. Lenže fyzika je v tomto žánry nežiadúca, preto je možné nastaviť daný komponent ako tzv. „trigger“. To spôsobí že Unity nebude pre daný objekt daného komponentu Rigidbody simulovať fyziku. Ale má aj za následok zmenu v spôsobe detekcie kolízií objektov. Nakoľko sa nesimuluje ich náraz a posôbenie ich vzájomných síl, Unity iba detekuje kedy sa jeden z triggerov dostane do collideru, či sa v triggery nacháza aspoň z časti iný collider (dochádza ku clippingu) a či trigger opustil hranice collideru. Z toho dôvodu sa tomuto typu interakcie v Unity nehovorí kolízia, ale pre konzistentnosť tejto sekcie a vyhnutiu sa zbytočnému komplikovaniu pojmov, sa dotyk triggerov s collidermi bude v tejto práci označovať ako kolízia.

Tieto kolízie sa spracovávajú v skripte `HitDetection`, respektíve `EnemyHitDetection` u postavy počítača. Tieto skripty obsahujú funkciu `OnTriggerEnter`, ktorá je volaná v danom skripte pri prvotnej kolízie triggeru s colliderom. Nakoľko sa útočiaci objekt nezastaví z dôvodu chýbajúcej simulácie fyziky, pri zásahu sa u útočníka stav útočenia nastaví na hodnotu false a tým sa ukončí animácia úderu, čiže to vizuálne pôsobí ako keby došlo k nárazu ako pri kolízii bez triggerov.

V prípade kolízie sa kontroluje či zasiahnutá postava bola v stave blokovania a aká časť tela bola zasiahnutá. V prípade stavu blokovania sa následne skontroluje či daná postava je v pozícii drepu. Ak áno, kontroluje sa či bola zasiahnutá do ruky alebo nohy, ak nie, tak sa kontroluje či bola zasiahnutá do ruky. V oboch prípadoch ak bol zásah vykonaný do daných častí tela je volaná metóda pre zranenie bloku danej postavy, nakoľko blokovanie sa aplikuje len pre ruky v stoji, respektívne ruky a nohy v drepe. Daná metóda sa nachádza v skripte `CharacterStatus`, prípadne `EnemyStatus` u postavy počítača. Ak postava neblokuje, je



Obrázek 4.10: Diagram stromu popisujúci vstupy pre útoky. Označenia veďla čiary označujú aky vstup je nutný pre prechod do daného uzlu. Obsah uzlu označuje aký typ útoku sa vykoná po prechode do neho.

volaná metóda pre zranenie životov. V prípade počítaču, dané skripty taktiež aktualizujú informácie o tom či úder zo strany počítača zasiahol hráča alebo či bol zablokovaný a u zranenia sa ukladá posledná zasiahnutá časť tela postavy počítača. V prípade zablokovania aj zasiahnutia oponenta, si obe postavy generujú energiu. Množstvo vygenerovanej energie aj zranenia obrany a životov je závislé na použitom type útoku a zasiahnutej časti tela, kde útok ma predom stanovenú hodnotu zranenia a časť tela má predom definovanú hodnotu, ktorou sa hodnota útoku násobí. Na tabuľke 4.1 sa nachádzajú hodnoty zranení jednotlivých typov úderov a hodnoty častí teľa, ktoré tieto zranenia násobia.

Skript pre detekciu kolízií sa nachádza na každej časti collider modelu, nakoľko u každej kolízie je nutné poznať meno daného objektu na ktorom sa collider nachádza. Rigidbody komponent je len na hlavnom objekte postavy, pretože collidery sa automaticky prepájajú s Rigidbody komponentom, ktorý sa nachádza v akomkoľvek rodičovi daného objektu alebo potomkovi jeho rodiča. Skript pre správu statusu postavy sa nachádza iba na rodičovskom objekte a referenciu na neho obsahujú všetky objekty collider modelu, rovnako ako pri predošlom skripte.

Typy útokov a hodnoty zranenia	
Typ úderu	Zranenie
DashPunch	40
HeavyHook	25
LightHook	20
HeavyUppercut	20
LowerUppercut	10
HeavyKick	15
LightKick	10
HeavyPunch	10
LightPunch	5

(a) Tabuľka hodnôt zranenia jednotlivých útokov.

Časti tela a hodnoty modifikátorov	
Časť tela	Modifikátor
Head	1.5
Neck	1.4
Torso	1.2
RUpperarm	1
LUpperarm	1
RForearm	1.1
LForearm	1.1
RHand	1
LHand	1
RThigh	1
LThigh	1
RShin	1.1
LShin	1.1
RFoot	1
LFoot	1

(b) Tabuľka modifikátorov jednotlivých častí tela.

Tabulka 4.1: Tabuľky hodnôt zranenia pre jednotlivé časti teľa a hodnoty zranenia jednotlivých typov útokov.

4.4.3 Animovanie úderov a kopov

Ako bolo spomenuté v druhej kapitole 3.4, pre realizáciu animácií v súboji bol zvolený balíček **Animation Rigging**. Pre prácu s týmto balíčkom je nutné pridať určité objekty a komponenty k objektom postáv hráča a počítača. Následne sa s týmto balíčkom pracuje v skriptoch, ktoré boli spomenuté v predošlej sekcii.

Pre realizáciu animovania súboja sa ako prvé vytvorili jednotlivé rigy pre všetky končatiny na rodičovskom objekte danej postavy. Toto automatický vytvorí komponent *Rig Builder* na danom objekte. V tomto komponente sa dajú nastaviť, ktoré rigy majú byť aktívne a s akou váhou majú byť ich zmeny aplikované na kosti zvolené v komponentoch daného rigu. Každý rig je zodpovedný za prepisovanie len tých kostí, ktoré sú v ňom zvolené. Jednotlivé váhy je možné upravovať aj na daných rigoch a aj jednotlivých komponentoch. Každý rig môže obsahovať viac než jeden komponent, čím sa môže docieliť komplexnejší pohyb. Konkrétne v tejto práci sa využili komponenty *Chain IK* na ľavej ruke, *Two Bone IK* na pravej ruke a oboch nohách a *Multi-Aim Contraint* u oboch rúk. U rúk bolo nutné použiť Multi-Aim Constraint pre korektné natočenie zápestia. Všetky komponenty v tomto nástroji vyžadujú Source objekt, ktorý využívajú ako referenčný a snažia sa k nemu priblížiť polohou alebo napodobniť jeho rotáciu (balíček ponúka aj iné komponenty ale tie neboli využité, takže sa v tejto práci popíšu len tieto funkcionality). Ako Source objekt sa pre každú postavu vytvoril prázdny objekt. Tento objekt má u hráča potomka objekt jednoduchej sféry a slúži ako vizuálna pomôcka pri hraní aj vývoji. U každého komponentu sa taktiež zvolili kosti, ktoré bude daný komponent animovať.

Pre jednoduchší prístup k jednotlivým rigom, bol vytvorený skript `AttackRig`, ktorý obsahuje referencie na jednotlivé rigy na postave, ku ktorej je daný skript pripojený. Tento

skript vo funckii `Start()` vyhľada všetky rigy zodpovedné za údery a inicializuje ich váhy na nulu. K jednotlivých rigom sa následne pristupuje cez verejné metódy.

Použité komponenty

Komponent **Two Bone IK** realizuje inverznú kinematiku (popísané v sekcii 2.4.4) v rámci dvoch rodičovských kostí voči tretej kosti (list), ktorá sleduje referenčný objekt. Týmto komponentom sú realizované útoky: Light punch (ľahký úder s pravou rukou), Light kick (ľahký kop s pravou nohou) a Heavy kick (ťažký kop s ľavou nohou). Ako list sa v prípade ruky použila kosť dlane a v prípade nôh kosť chodidla. U ruky sa ako rodičovské kosti použili kosť predlaktia a ramenná kosť, čiže výsledná animácia úderu končí pri ramene. V prípade nôh sa zvolili holená a stehenná kosť, takže animácia končí pri pánve u oboch typov kopu. Je patrné že kostra modelu nie je úplne realistická, keďže chýbajú určité kosti (u ruky chýba lakťová kosť a u nohy ihlica). To z dôvodu že pre animovanie modelu stačí jedna kosť pre danú časť tela či končatiny, takže použitý model nemusel zodpovedať realite.

Komponent **Chain IK** sa taktiež využíva pre inverznú kinematiku, ale narozdiel od Two Bone IK, nemá pevne daný počet kostí, pre ktoré sa má daná kinematika vykonávať. Tento komponent sa použil pre útok Heavy Punch (ťažký úder s ľavou rukou). Dôvod použitia tohto kompotentu bol z dôvodu zvýšenia dosahu daného úderu a aby aj vizuálne šlo rozoznať že ide o ťažký úder. Ako vrchol IK sa zvolila kosť dlane a ako konečná chrbtová kosť. Vďaka tomu sa vo finálnom pohybe zapojí aj rameno a časť horného chrbta.

Všetky skripty zodpovedné za animovanie úderov, dedia zo skriptu `AttacksBase`, ktorý obsahuje referenciu na skript obsahujúci rigy postavy a skript pre sledovanie stavu postavy. Taktiež všetky typy úderov sú animované za použitia triedy Coroutine, ktorá bola spomenutá v sekcii 4.2. Algoritmus 1 znázorňuje animovanie základných úderov za pomoci plynulej zmeny váh rigu.

Algoritmus 1 Pseudokód vykonávajúci korutinu pre plynulé zvýšenie a následné zníženie váhy rigu.

```
attackingState ← true
weight ← 0
while weight < 1 and attackingState is true do
    yield return null
    weight ← weight + (transitionSpeedTime.deltaTime)
end while
while weight > 0 and attackingState is true do
    yield return null
    weight ← weight − (transitionSpeedTime.deltaTime)
end while
attackingState ← false
```

Animovanie v rámci skriptov

Animovanie základných úderov a kopov sa vykonáva v skripte `SimpleAttacks`. Hlavná metóda pre animovanie základných úderov sa volá `SimpleAttack`. Parametre tejto metódy udávaju s ktorým rigom sa bude pracovať a akou rýchlosťou sa má daný pohyb vykonať. Animácia sa vykonáva postupnou zmenou váhy zvoleného rigu v cykle `while`, čím sa vrchol rigu (kosť dlane) periodicky posúva smerom k referenčnému objektu. Následne sa prejde do

druhého cyklu, ktorý túto váhu postupne znižuje. Hodnota zmeny sa ešte násobí hodnotou `Time.deltaTime`, rovnako ako pri implementácii pohybu popísaného v sekcii 4.2, aby sa zaistila konzistencia pohybu medzi snímkami. Oba cykly sa vykonávajú dokým nedosiahnu hraničnej hodnoty váhy rigu (minimum 0, maximum 1) a kým je postava v stave útočenia. Tento stav sa zmeni na nepravdu, keď postava zasiahne oponenta, čím sa automaticky preskočia oba cykly a hodnota váhy sa nastaví na nulu. Táto prudká zmena v prípade zásahu je nutná pre plynujelší priebeh súboja. Pri korutine je možné vracať rôzne typy hodnôt určené pre použitie s kľúčovým slovom yield. Samotná funkcia vracia iba null, nakoľko tu nie je nutné čakať na nejakú udalosť. Samotná metóda pre zmenu váhy rigu nie je verejná a skript obsahuje verejné metódy pre volanie tejto korutiny so správnym rigom a rýchlosťou prechodu.

Kombo údery sa nachádzajú v skripte `ComboAttacks`, kde hlavná metóda sa volá `Hook`. Metóda prijíma tri parametre, pričom prvé dva sú typu `Bool` a tretí je rovnako ako u jednoduchého úderu typu Rig ako u základných úderov. Prvý parameter `isUppercut` udáva aký typ pohybu sa ma animovať, horizontálny alebo vertikálny. Druhý parameter `isLightPunch` udáva aký komponent sa má vyhľadať v danom rigu (Chain IK alebo Two Bone IK). Narozdiel od základných pohybov už nestačilo len postupne upravovať hodnotu váhy, pretože dané komponenty sa snažia priblížiť referenčnému objektu čo najkratšou cestou, ale v tomto prípade sa má animovať tzv. hák a spodný hák. Takže je nutné animovať pohyb, pri ktorom kosť dlane nasleduje kruhovú trajektóriu. Toho sa dosiahlo nastavením váhy na maximum a následného plynulého pohybu samotného referenčného objektu. Tento pohyb trvá maximálne pol sekundy alebo dokým nedôjde k zásahu. Následne sa váha plynule zníži na nulu. V prípade zásahu sa nuluje okamžite. Aby sa vyhlo geometrickým výpočtom, použila sa metóda `RotateAround`. Táto metóda sa volá na referenčný objekt a prijimá ako parametre objekt (ďalej stredový objekt) okolo ktorého sa má rotácia vykonať, na akej osy sa má rotácia vykonávať a rýchlosť rotácie v uhloch za sekundu. Keďže táto metóda vykoná iba jednu iteráciu danej rotácie za jeden snímok, daná rýchlosť sa násobí hodnotou `Time.deltaTime`. Ďalši problém pri tomto type úderu bol že táto metóda vykonáva iba dvojdimenzionálnu rotáciu okolo stredového objektu. Aby sa zabezpečil pohyb na všetkých troch osách, využilo sa toho že daná funkcia berie do úvahy lokálnu rotáciu stredového objektu. Tento objekt sa vždy vytvorí pred začiatkom cyklu, v ktorom sa vykonáva rotácia. Jeho poloha sa nastaví na stred medzi pozíciou kosti dlane (začiatočná pozícia) a pozíciou, na ktorú hráč alebo počítač mieri (konečná pozícia). Následne sa nastaví jeho rotácia pomocou funkcie `LookRotation` tak, aby bol natočený smerom na cieľovú pozíciu. Metóda RotateAround teda pracuje s naklonenou osou (konrétna os je zvolená na základe parametru isUppercut) stredového objektu a výsledný pohyb je vďaka tomu realizovaný na všetkých osách. Algoritmus 2 znázorňuje pseudokód implementácie tohto pohybu.

Animovanie špeciálnych úderov sa vykonáva v skripte `SpecialAttacks`. Tento skript pracuje aj so skriptom `MovementControl`, na koľko špeciálny úder „Dash Punch“, využíva mechaniku dashu, ktorá je vysvetlená v sekcii o základnom pohybe 4.2. Pri tomto type úderu sa nastaví váha rigu pravej ruky na maximum a následne sa spustí korutina dashu so zvýšenou rýchlosťou smerom k oponentovi, čo ma za výsledok úder s veľmi rýchlym posunom smerom k počítaču. Korutina tohto úderu následne čaká kým sa nedokončí dash a následne sa zníži, prípadne vynuluje váha rigu. Tento útok vyžaduje aby daná postava mala naplnený energy bar aspoň na hodnotu 60 (maximum je 100). Táto hodnota sa odpočíta z energy baru postavy, ktorá útok použila.

V prípade blokovania sa referencia o rigu ukladá v skripte `BlockRig`. Rig animuje obe ruky, pričom pozícia referenčného objektu sa upravuje pomocou metódy `Target2BlockPosi-`

Algoritmus 2 Pseudokód realizujúci rotáciu referenčného objektu `aimTarget` okolo stredového objektu `centerObject`.

$centerObject \leftarrow$ new $GameObject()$
$centerObject.position \leftarrow (endPosition - startPosition)/2$
$centerObject.rotation \leftarrow LookRotation(directionTowardsEndPosition)$
$attackingState \leftarrow$ **true**
while $currentDuration < totalDuration$ **and** $attackingState$ is **true** **do**
    yield return null
    $aimTarget.RotateAround(centerObject.position, centerObjectAxis, rotationSpeed \times Time.deltaTime)$;
end while
$attackingState \leftarrow$ **false**
$Destroy(centerObject)$

`tion`, ktorá volá pomocnú metódu `MoveInFrontOf` s parametrom vzdialenosti nastaveným na hodnotu 1. Tým sa dosiahne konštantná vzdialenosť rúk od postáv hráča alebo počítača vždy keď blokujú údery. Začatie a ukončenie blokovania sa vykonáva metódami `StartBlocking` a `StopBlocking`. Dané metódy nastavujú váhu rigu na nulu, respektíve na hodnotu 1 a status blokovania na hodnotu true, respektíve false.

4.5 Mierenie úderov

Pupil Core spracováva dáta z okuliarov programom **Pupil Capture**. Pupil Capture je ale externý program, takže je potrebná implementácia ich komunikácie, kde Pupil Capture zasiela dáta o pohľade a Unity ich číta a aktualizuje ich pre bojový systém. Toto sa dá vyriešiť pomenovanou pipeline alebo serverom s API. Z hľadiska odozvy by bolo vhodné zvoliť pipeline, lenže nakoľko Pupil Capture a hra pobežia na rovnakom stroji, výsledná komunikácia bude iba lokálna, takže odozva bude zanedbateľná. Zároveň, Pupil Core ponúka API z ktorého je už možné prijímať potrebné dáta, čiže nie je nutné implementovať celé API ale len komunikáciu s ním zo strany hernéj aplikácie.

Toto API využíva pre komunikáciu knižnicu **ZeroMQ**[3], ktorá umožňuje zasielať správy medzi aplikáciami v rámci siete bez využitia middlewaru. Ponúka rôzne vzory pre komunikáciu medzi zariadeniami. Konkrétne Pupil Capture využíva vzor *Publisher-Subscriber* (Vydavateľ-Odoberateľ), kde Pupil Capture je v roli vydavateľa a periodicky zasiela správy s dátami o pohľade hráča. Herná aplikácia bude vystupovať v roli odoberateľa a periodicky dané správy prijímať a čítať. Pretože sa táto práca píše v jazyku C#, využije sa knižnica *NetMQ*[4], ktorá je portom knižnice ZeroMQ do jazyka C#.

Okrem tejto knižnice sa ešte zo strany Pupil Capture využíva dátový formát **MessagePack**[5], ktorý je alternatívou formátu *JSON*. Hlavnou výhodou tohto formátu je malá veľkosť oproti JSONu, vysoká rýchlosť a použiteľnosť vo veľkej škále programovacích jazykov [5]. Rovnako ako pri predošlej knižnici bol použitý serializér tohto formátu pre jazyk C#.[6]

[3] https://zeromq.org/
[4] https://github.com/zeromq/netmq
[5] https://msgpack.org/
[6] https://github.com/neuecc/MessagePack-CSharp

4.5.1 Komunikácia a spracovanie dát

Pre získavanie údajov o pohľade z okuliarov Pupil Core sa využíva skript `AimDataReceiver`. Pomocou knižnice `NetMQ` sa realizuje komunikácia s okuliarmi a pomocou knižnice `MessagePack` sa deserializujú dáta z formátu msgpack. Samotná komunikácia prebieha v dvoch fázach. Najprv sa zašle požiadavka na port 50020 (predvolený port okuliarov) pre získanie portu pre komunikáciu typu subscriber-publisher, následne sa s pomocou novehé portu vytvorí nové spojenie (prvé spojenie je ukončené po získaní nového portu). Komunikácia prebieha na localhoste. Obe fázy komunikácie bežia na samostatnom vlákne, nakoľko daná komunikácia využíva nekonečný cyklus, a ak by bežala na hlavnom vlákne, došlo by k blokovaniu celého behu Unity. Komunikáciu je možné vypnúť a opätovne zapnúť stlačením klávesy čísla 1 na numpade. Dáta získavané z okuliarov sú normalizované do rozsahu <0,1>, takže sa dané koordináty ešte násobia veľkosťou okna hry. Tieto dáta sú následne dostupne cez verejnú funkciu, ktorá je volaná zo vstupového skriptu hráča každý snímok.

Presnosť pohľadu ale nie je perfektná. U Pupil Core dochádza k periodickému „traseniu sa“ bodu na ktorý sa hráč pozera, pretože sa dáta pravidelne aktualizujú. To z dôvodu že aj keď sa hráč díva na ten istý bod, výsledne koordináty sú mierne odlišné od predošlých a tým vzniká spomínaný efekt trasenia sa sledovaného bodu. Kvôli tomu nie je možné dosiahnuť úplnu presnosť. Toto sa z časti vyriešilo zpriemerovaním prichádzajúcich koordinátov v rámci posuvného okna. Algoritmus 3 popisuje fungovanie tohto posuvného okna. Pre dostatočnú presnosť sa zvolila veľkosť okna 50. V prípade rýchlej zmeny bodu pohľadu sa z daného okna odstráni prvá polovica hodnôt, pre rýchlejšiu reakciu mierenia bez veľkej straty presnosti. Toto ale nevyriešilo problém presnosti úplne a stále sú dané koordináty závislé na kvalite kalibrácie v Pupil Capture. Nakoľko daný program nie vždy vie dokončiť kalibráciu, je nutné je opakovať niekedy viac krát pre dosiahnutie požadovanej (v rámci možností) presnosti.

S vyšše spomenutým problém presnosti vznikol aj ďalší problém, a to že cieľ pohľadu niekedy sa nachádzal nie na časti tela nepriateľa i keď sa na neho hráč díval. Z toho dôvodu bolo nutné obmedziť zmeny pozície bodu pohľadu len na osy x a y. Čiže daný bod je na ose z v konštantnej pozícii v strede hernej oblasti.

Algoritmus 3 Pseudokód posuvného okna použitého pre spresnenie mierenia hráča.

```
window ← newList()
minDistance ← 0.05
maxCount ← 50
while communicationAlive is true do
    window.Add(newCoordinate)
    if window.Count > maxCount then
        window.RemoveAt(0)
    end if
    if newCoordinate.Distance(currentCoordinate) > minDistance then
        window.RemoveRange(0, window.Count/2)
    end if
    newCoordinate ← window.SumAll()/window.Count
end while
```

4.6 Implementácia umelej inteligencie

AI je implementované v skripte `CombatAI`, v ktorom sa rozhodovanie delí na dve časti. Ofenzívna časť je zodpovedná za útoky a približovanie sa k hráčovi a defenzívna časť za blokovanie a vzdalovanie sa od hráča. Rozhodnutie či sa má prejsť do ofenzívy alebo defenzívy je závislé na tom či počítač bol zranení od poslednej kontroly životov.

V prípade že áno, prechádza sa do **defenzívy** v ktorej sa kontroluje množstvo strateného zdravia. Pri menšom množstve AI prechádza do ofenzívy, v prípade prekonania prvej nastavenej hranice dôjde k blokovaniu, ak nie je v stave rozbitej obrany. Pri prekonaní tejto hranice, sa začne počítač vzdialovať chôdzou od hráča, v prípade prekonania druhej hranice sa vykoná dash smerom od hráča. Vzďalovanie sa rieši ale len v prípade že AI nie je u steny levelu. Vtedy sa rozhoduje iba o blokovaní.

V prípade že zdravie sa nezmenilo prechádza sa do **ofenzívy**. Najprv sa vždy kontroluje vzdialenosť od hráča a následne sa pokúsi počítač priblížiť, za pomoci chôdze alebo dashu, podľa toho ako veľmi daleko sa nachádza. Následne sa prejde k rozhodovaniu o type útoku, ktorý sa má použiť. V tejto časti sa využíva skript `RNGHandler`, v ktorom sa nachádza inštancia triedy `Random`[7]. Táto trieda ponúka generovanie náhodných čísel, pričom skript využíva túto triedu spolu s pomocnými premennými pre určitú kontrolu náhodnosti. Konkrétne ide o hodnoty, ktoré ovplyvňujú výsledne rozhodovanie sa medzi využitím základného úderu alebo kombo útoku. Ak sa využil základný úder, navýši sa šanca pre použitie kombo útoku. Ak dôjde k použitiu kombo útoku, resetuje sa ovplyvňujúca hodnota na nulu. Rovnaká kontrola prebieha pri rozhodnutí či sa má použiť speciálny úder, ale u tohto rozhodovania sa predtým skontroluje je dostatok energie na jeho vykonanie. Pri zvolení základného úderu, je možné výber obmedziť. Napríklad ak je hráč v drepe, AI sa môže rozhodnúť taktiež prejsť do drepu a následne zvoliť úder rukami. V prípade že sa rozhodne stáť, výber úderu je obmedzený na kopy (z dôvodu dosahu). Toto obmedzenie funguje obdobne aj u zvolení cieľa úderu, v prípade že sa použil kop, môže sa mieriť kamkoľvek, v prípade úderu rukou sa mieri na hornú polovicu teľa hráča. Skript pre riešenie náhodnosti taktiež obsahuje všetky kombá, ktoré bojový systém obsahuje a po náhodnom zvolení odovzdá sekvenciu potrebných úderov AI skriptu, ktorý ich postupne vykonáva a ukončí ich vykonávanie keď sa prejdú všetky vstupy komba. V prípade špeciálnych úderov sa pridajú všetky vstupy okamžite, pretože tento typ útokov sa skladá iba z jedného hlavného pohybu, zatial čo kombá sa skladajú z niekoľkých animácií útokov, ktoré je nutné nechať dohrať. Všetky kontroly sa vykonávajú vo funkcii Update, takže bolo nutné obmedziť ako často sa môže AI dostať do ofenzívy, nakoľko vykonávanie každy snímok by znemožnilo hráčovi efektívne reagovať na útoky AI. Rozostup medzi jednotlivými ofenzívami bol preto nastavený na 0.25 sekundy. Príloha A obsahuje diagram popisujúci finálnu verziu AI založeného na diagrame v kapitole návrhu 3.3. Diagram neobsahuje všetky aspekty rozhodovania z dôvodu zachovania prehľadnosti diagramu.

[7] https://docs.microsoft.com/en-us/dotnet/api/system.random?view=net-6.0

Kapitola 5

Testovanie

Táto kapitola sa venuje testovaniu výslednej aplikácie, lokálne aj na užívateľoch a následom vyhodnotení užívateľského testovania. Prvá sekcia 5.1 popisuje ako prebiehalo užívateľské testovanie a čo sa testovanie snažilo zistiť. Sekcia 5.2 konštatuje dosiahnuté výsledky testovania a taktiež vyvodzuje na základe dát z dotazníkov, ktoré aspekty aplikácie splnili ciele práce a ktoré sú problematické.

5.1 Priebeh užívateľského testovania

Proces testovania nebol obmedzený z hľadiska demografie oslovených užívateľov. Rovnako skupina ľudí na ktorých sa hra testovala nemala žiadne obmedzenia na základe skúseností s video hrami. To z dôvodu že táto práca nemala vytvoriť hru so špecifickými požiadavkami, čiže bolo vhodné zaistiť testovanie aj na ľudoch, ktorý s bojovým žánrom hier alebo celkovo s hrami nemajú skúsenosť. Testovanie bolo vykonávané na ľudoch vo vysokoškolskom prostredí, konkrétne študentoch vysokých škôl.

Testovanie začínalo v tréningovom móde v režime mierenia myšou, pre naučenie sa základov hry a navyknutie si na daný typ mierenia. Dodržanie limitu v tomto móde sa striktne nekontrolovalo, nakoľko každý užívateľ mal inú rýchlosť učenia sa. Nasledovali dva zápasy v režime mierenia myšou. Následne sa obe fázy (tréningový mód a dva zápasy v normálnom móde) zopakovali s režimom mierenia pohľadom. Nasledovalo vyplnenie dotazníka, ktorý sa skladal z troch častí. Prvé dve časti sa otázkami sústreďovali na technický stav hry, pričom prvá sa pýtala na režim mierenia myšou a druhá časť na režim mierenia pohľadom. Tretia časť sa pýtala otázky ohľadom porovnania daných režimov a či respondentom prišla mechanika mierenia pohľadom ako pozitívny prídavok do bojového žánru hier.

Otázky spoločné pre oba typy režimov sa viac sústredovali na technický stav daných režimov. Odpovedať bolo mozné na škále od 1 bo 5 (nižšia hodnota bola braná ako pozitívnejšia). Otázky boli nasledovné:

- Aká bola náročnosť ovládania mierenia?
- Aká bola presnosť mierenia?
- Aká bola odozva mierenia pri zmene cieľa?

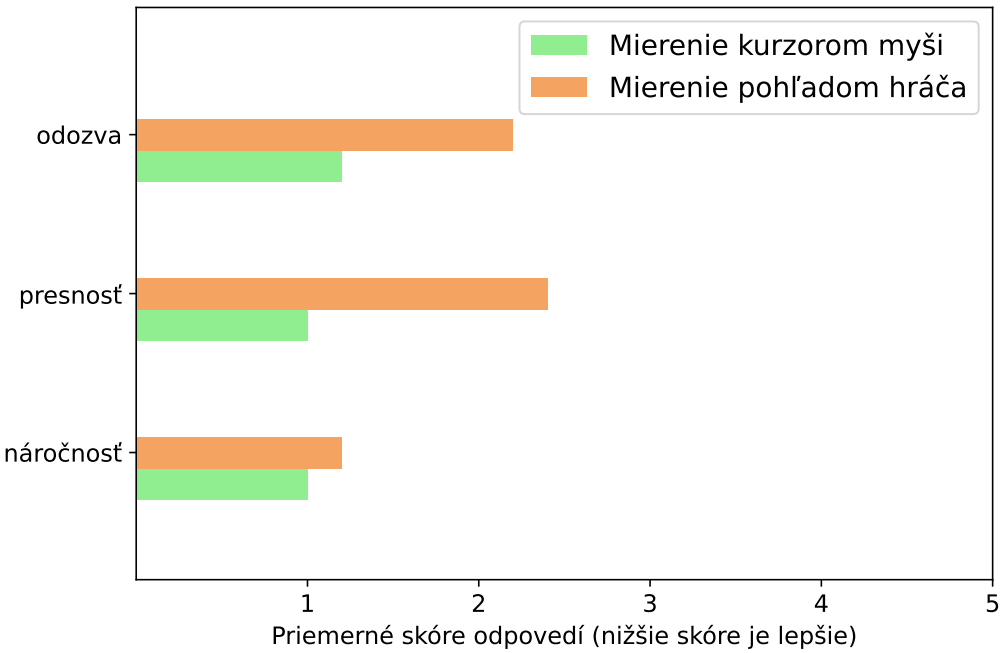
Otázky tretej časti sa pýtali užívateľa na názor na danú aplikáciu ako na herný produkt. Odpovede boli uzavreté a vždy boli na výber dve odpovede. Otázky zneli nasledovne:

- V ktorom režime ste strávili viac času učením sa ovládania a mierenia?
- Ktorý režim vám prišiel celkovo zábavnejší?
- Je podľa vás mechanika mierenia pohľadom vhodná pre žáner bojových hier?

5.2 Výsledky testovania

Testovanie bolo vykonané na 5 respondentoch. Počas hrania hry bolo umožnené respondentom komentovať ich názory na jednotlivé časti hry počas vypĺňania dotazníku. Prvotné otázky ohľadom technického stavu hry, poukázali na technické nedostatky režimu mierenia pohľadom, ako je možné vidieť na grafe 5.1. Tento výsledok sa očakával, nakoľko sa tieto problémy, opísané v kapitole implementácie 4.5.1, objavili už pri vývoji, ale aj napriek implementovaným riešeniam sa nepodarili kompletne odstrániť.

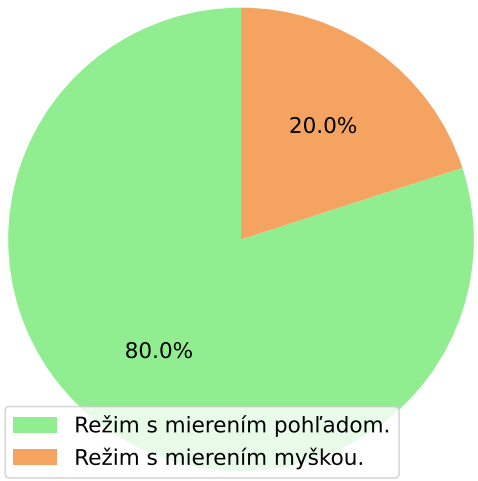
Čo sa týka náročnosti ovládania, tak je vidieť že oba režimy si sú dosť podobné. Z komentárov respondentov počas testovania, vyplynulo že skôr išlo o zvyknutie si než vyslovene učenie sa tomuto typu ovládania. Čiže z tohto hľadiska je možné konštatovať že náročnosť danej mechaniky nie je problematická.



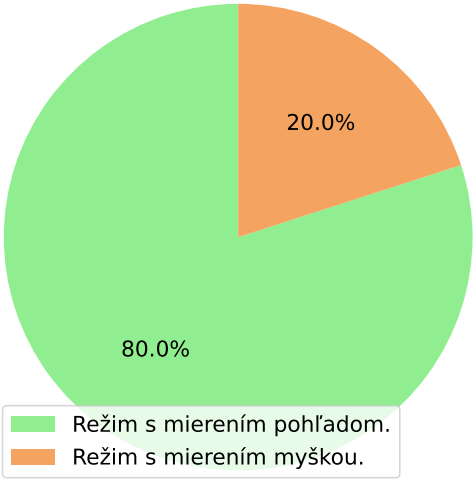
Obrázek 5.1: Graf zobrazujúci výsledky otázok, ktoré boli v rámci dotazníku spoločné pre oba režimy.

Posledná časť dotazníku riešila porovnanie oboch režimov mierenia. Ako vidieť na grafe 5.2b, režim s mierením pohľadom bol zábavnejší než režim s mierením myšou. I keď z grafu 5.2a vyplýva že mierenie pohľadom bolo náročnejšie, z predošlého grafu 5.1 je známe že rozdiel bol minimálny. Z toho sa dá vyvodiť že náročnosť nemala vplyv na „zábavnosť“

režimu. Zábavnosť je v tomto kontexte veľmi subjektívna a nie je to najlepší prediktor, ale nakoľko ide v tejto práci o počítačovú hru, ktorá implementuje novú mechaniku v danom žánry, tak zábavnosť je považovaná za validnú vlastnosť pre testovanie, keďže sa tým dá vyhodnotiť splnenie cieľa tejto práce.



(a) Graf zobrazujúci v ktorom režime hráči strávili viac času učením sa ovládania mierenia.

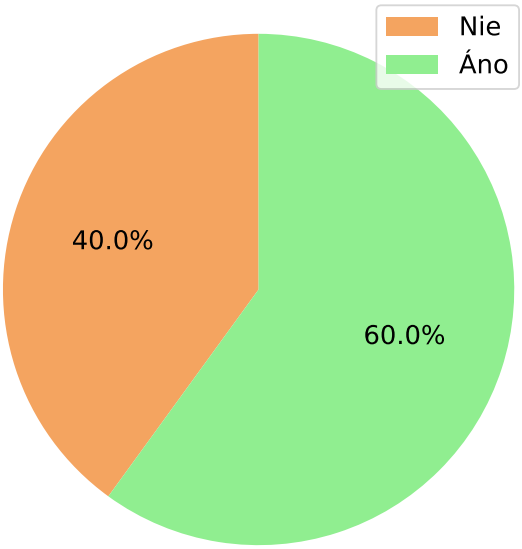


(b) Graf zobrazujúci, ktorý režim prišiel hráčom zábavnejší.

Obrázek 5.2: Grafy zobrazujúce výsledky testovania ohľadom porovnania oboch režimov mierenia.

Finálna otázka sa pýtala práve na to, či daná mechanika je vhodný prídavok pre bojový žáner hier. Z grafu 5.3 je vidieť že väčšina respondentov reagovala pozitívne, lenže rozdiel nie je natoľko veľký ako pri predošlých grafoch. Pri vyplňovaní tejto otázky padol viac krát názor že súboje v danej hre sú prirýchle, čo spôsobuje že daná mechanika nemá dostatočný efekt na priebeh hry. Tento aspekt bohužiaľ nebolo možné opraviť bez kompletného prerobenia celej hry, nakoľko dizajn hry a mechaník sa inšpiroval už existujúcimi hrami v tomto žánri. To sa ukázalo ako hlavný problém, keďže dané hry nepočítajú s mechanikou mierenia, ich implementácia súboja a mechaník spojených s ním nebola vo výsledku vhodná pre hru v ktorej táto mechanika figuruje.

Výsledky testovania poukázali na dva problémy výslednej aplikácie. Hlavný problém je dizajn mechaník súboja a priebehu súboja. Riešenie by bolo hlbšie testovanie s rôznymi zmenami parametrov, hlavne zranenia a trvania úderov, čím by bolo možné spomaliť priebeh súboja a dať väčšiu váhu mechanike mierenia. Druhotný problém bola nepresnosť mierenia pohľadom. Tuto riešenie nie je úplne očividné, nakoľko zdroj problému je hardware použitý v tejto práci a aj s dostatočnou kalibráciou, určitá nepresnosť by bola stále prítomná. Testovanie pomohlo zvýrazniť hlavné problémy aplikácie a zároveň pomohlo poukázať na časti a aspekty, ktoré fungujú správne a nie sú problematické. Tretia časť dotazníka by ale požadovala väčšiu skupinu respondentov pre presnejšie dáta a špecifickejšiu odozvu toho čo je potreba zlepšiť.



Obrázek 5.3: Graf zobrazujúci to či respondentom prišla mechanika mierenia pohľadom ako vhodná pre bojový žáner hier.

Kapitola 6

Záver

Táto práca mala za cieľ návrh a implementáciu počítačovej hry v bojovom žánri doplnenú o mechaniku mierenia pohľadom hráča. Na začiatku bolo nutné naštudovať mechaniky bojového žánru hier, ktoré musela daná hra obsahovať. Z technickej časti bolo zároveň nutné sa zoznámiť a naučiť pracovať s editormi pre tvorbu 3D modelov a hier, konkrétne Blenderu pre tvorbu modelov a herného enginu Unity pre tvorbu hier. Zároveň bolo nutné naštudovať základy sledovania pohybu a pohľadu očí. Na toto sa využili špeciálne okuliare Pupic Core, ktoré umožnili dané dáta získavať a zdielať pomocou lokálneho serveru. Potom sa vytvoril základný návrh modelu postavy, mechaník hier, komunikácie a spracovania dát z okuliarov a UI. Na základe týchto návrhov bola implementovaná bojová hra s mechanikou mierenia.

Výsledná hra umožňuje bojovať proti nepriateľovi riadenému počítačom za pomoci mechaniky mierenia buď pohľadom hráč alebo kurzorom myši. Pri mierení kurzorom myši je vstup súboja obmedzený len na dva typy úderov a blokovanie. Pri mierení pohľadom sa sprístupnia ďalšie dva typy úderov, nakoľko obe ruky môžu používať klávesnicu pre vstup. Na základe testovania sa vyhodnotilo že ovládanie oboch režimov mierenia bolo jednoduché na naučenie sa. Naopak pri režime mierenia pohľadom nie je presnosť a odozva optimálna. Tieto problémy boli spôsobené hardwarom použitým pri tejto práci a aj napriek implementovaným riešeniam boli stále prítomné. Samotná hra prišla užívateľom zábavná ale samotná mechanika mierenia nebola jednoznačne považovaná za vhodnú pre tento žáner. Na základe komentárov užívateľov pri testovaní bol hlavný problém priebeh súboja s touto mechanikou.

Pri plánovaní vhodných zmien do budúcna na základe spätnej väzby, sa došlo k záveru že by bolo nutné navrhnúť nový súbojový systém a nesnažiť sa napodobiť už existujúce hry. Vhodný dizajn súbojov by bol taký, kde údery su pomalé ale spôsobujú veľké poškodenie a miesta kam obe postavy mieria by boli viditeľné, takže plánovanie každého kroku a rýchle zmeny v rozhodnutiach by hrali väčšiu rolu. Okrem týchto zmenov v dizajne hry by sa mali pridať kombo údery a špeciálne údery, nakoľko z dôvodu animovania pomocou skriptov stihli sa pridať len tri. Taktiež by bolo možno na zváženie viac experimentovať s kombinovaním animovania pomocou klasických animácii a nástrojov pre manuálne animovanie.

Vo výsledku sa podarilo splniť všetky body zadania. Výsledná aplikácia implementovala mechaniku mierenia pohľadom v rámci bojovej hry a bola otestovaná na užívateľoch. Užívatelia dali najavo že daná hra je zábavná a mechanika mierenia im prišla ako vhodná mechanika pre bojový žáner hier a taktiež sa získali cenné poznatky na základe, ktorých je možné hru do budúcna zlepšiť a vyladiť testovaním zistené nedostatky.

Literatura

[1] Baechler, O. a Greer, X. *Blender 3D By Example: A project-based guide to learning the latest Blender 3D, EEVEE rendering engine, and Grease Pencil*. 2. vyd. Packt Publishing, 2020. ISBN 9781789617993. Dostupné z: https://books.google.cz/books?id=_4LoDwAAQBAJ.

[2] Duchowski, A. *Eye Tracking Methodology*. 2. vyd. Springer, London, 2007. ISBN 978-1-84628-609-4.

[3] Gregory, J. *Game Engine Architecture*. 3. vyd. CRC Press, 2018. ISBN 9781315267845.

[4] Horsley, M., Eliot, M., Knight, B. A. a Reilly, R. *Current Trends in Eye Tracking Research*. Illustrated. Springer Science & Business Media, 2013. ISBN 978-3-319-02868-2.

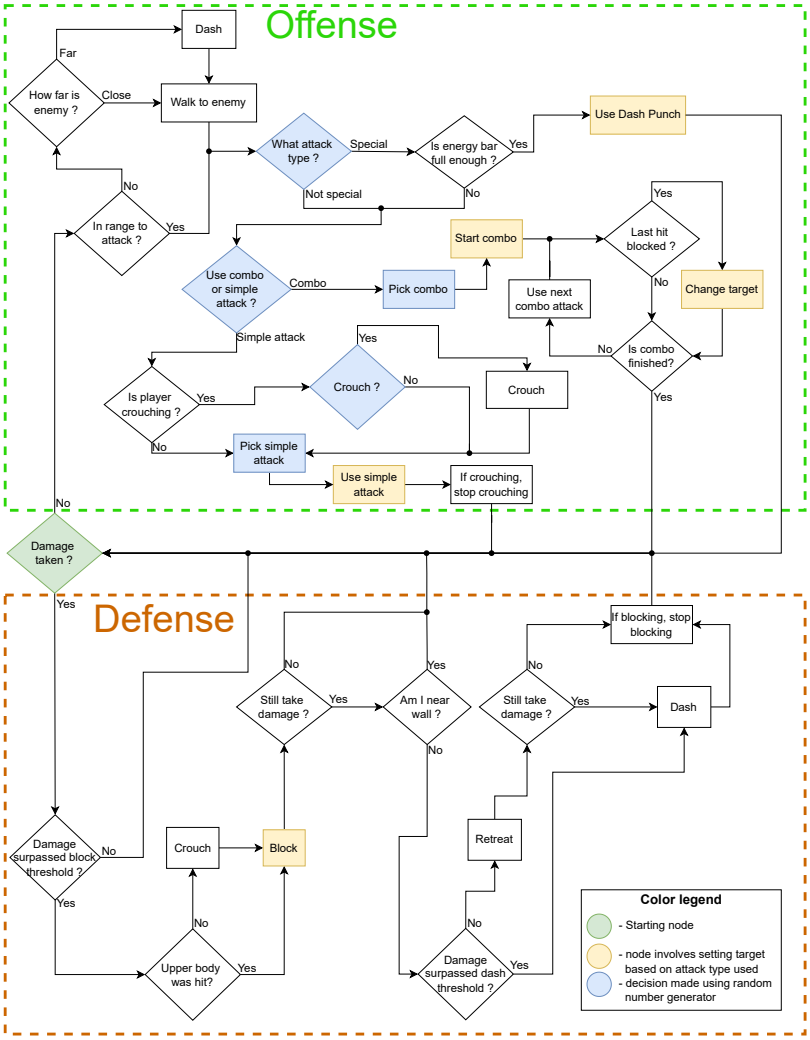
[5] Pupil Labs. *Getting Started | Pupil Labs* [online]. 2021 [cit. 2021-01-14]. Dostupné z: https://docs.pupil-labs.com/core/.

[6] Unity Technologies. *Unity - Manual: Working in Unity* [online]. 2020.3 [cit. 2021-12-09]. Dostupné z: https://docs.unity3d.com/Manual.

[7] Velloso, E. *How eye tracking gives players a new experience in video games* [online]. 2016 [cit. 2021-12-26]. Dostupné z: https://theconversation.com/how-eye-tracking-gives-players-a-new-experience-in-video-games-54595.

Příloha A

Diagram finálnej verzie AI



Obrázek A.1: Diagram popisujúci rozhodovanie AI. Určité aspekty rozhodovania neboli zahrnuté z dôvodu prehľadnosti diagramu.

Příloha B

Obsah pamäťového média

```
├──app/
│   ├──build/                  - Skompilovaná verzia hry pre jednoduché spustenie
│   ├──project/                - Unity projekt, pre zobrazenie a skompilovanie
│   │                            v Unity editore
│   └──source/                 - Zdrojové kódy hry v jazyku C#
├──license/
│   └──AprilTags/              - Licencia použitých AprilTagov
├──surface/
│   └──surface_definitions_v01 - Súbor s definíciou povrchu monitora
│                                pre okuliare Pupil Core
├──text/
│   ├──bp.pdf                  - text práce vo formáte PDF
│   └──latex.zip               - zdrojové súbory textu práce (Latex)
├──video/
│   └──clip.mp4                - Videoukážka hry v oboch režimoch mierenia
└──README.txt                  - Súbor s návodom pre spustenie hry
```