# TURINYS

[ĮVADAS 2](#_Toc515399507)

[1. INFORMACINIŲ ŠALTINIŲ APŽVALGA 3](#_Toc515399508)

[2. UŽDUOTIES FORMULAVIMAS 4](#_Toc515399509)

[3. UŽDUOTIES ANALIZĖ 5](#_Toc515399510)

[3.1. Žaidimų analizė 5](#_Toc515399511)

[3.1.1 One Finger Death Punch 5](#_Toc515399512)

[3.1.2 Ultimate Mortal Kombat 3 9](#_Toc515399513)

[3.2. Naudojamos programinės įrangos analizė 12](#_Toc515399514)

[4. PROJEKTINĖS DALIES APRAŠYMAS 13](#_Toc515399515)

[4.1. Žaidimo scenos 13](#_Toc515399516)

[4.2. Žaidimo valdymas ir mechanikos 15](#_Toc515399517)

[4.3. Žaidimo veikėjų spraitai ir animacijos. 19](#_Toc515399518)

[4.4. Žaidimo garsai ir muzika. 21](#_Toc515399519)

[4.5. Žaidimo programinė dalis. 22](#_Toc515399520)

[5. NAUDOTOJO INSTRUKCIJA 28](#_Toc515399521)

[5.1. Žaidimo diegimas. 28](#_Toc515399522)

[5.2. Žaidimo išdiegimas. 29](#_Toc515399523)

[5.3. Žaidimo apribojimai. 29](#_Toc515399524)

[6. REZULTATAI 30](#_Toc515399525)

[7. IŠVADOS 31](#_Toc515399526)

[INFORMACIJOS ŠALTINIŲ SĄRAŠAS 34](#_Toc515399527)

[PRIEDAI 35](#_Toc515399528)

ĮVADAS

**Darbo aktualumas.** Kompiuteriniai žaidimai tapo neatsiejama šiuolaikinės kultūros dalimi. Šiandien į kompiuterinius žaidimus žvelgiama kaip į populiarią laisvalaikio leidimo priemonę, kuri turi įvairios naudos ir paskirčių. Kadangi kompiuterinių žaidimų galima sutikti beveik visur, tiek namuose, tiek darbe ar mokymo įstaigose, žaidimų kūrėjai ieško sričių, kur kompiuteriniai žaidimai galėtu padėti mūsų kasdieniame gyvenime. Viena iš tų sričių yra žmogaus reakcija. Kompiuteriniai žaidimai tampa viena populiariausių priemonių ląvinti žmogaus reakciją. Būtent muštynių žaidimų gebėjimas įtraukti vartotoją į žaidimo pasaulį ir tuo pačiu ląvinti jo greitajį mąstymą yra patvirtintas, kaip vienas iš efektyviausių būdų tobulinti žmogaus reakciją.

**Darbo tikslas** – Sukurti 2D muštynių žaidimą.

**Darbo uždavinai:**

1. Atlikti 2D mušynių žaidimų analizę.
2. Remiantis atlikta analizę, pasirinkti įrangą žaidimo kūrimui.
3. Žaidimo dizainas.
4. 2D modelių ir jų animacijų kūrimas.
5. Žaidimo mechanikų programavimas.
6. Muzikinio takelio kūrimas.
7. Žaidimo testavimas.

**Darbe vartojamos sąvokos:**

1. **2D** – Kompiuterinė grafika, kurioje visi elementai yra dviejų dimencijų aplinkoje.
2. **Unity –** Programinė įranga skirta kompiuterinių žaidimų kūrimui.
3. **Animacija** – Besikeičiančių kadrų procesas, sukuriantis tam tikro veiksmo iliuziją.
4. **Spraitas –** Objektas nupieštas 2D aplinkai.
5. **Skriptas –** Programinio kodo dalis skirta tam tikrai žaidimo mechanikai.
6. INFORMACINIŲ ŠALTINIŲ APŽVALGA
7. https://unity3d.com/

Oficiali „Unity“ programinės įrangos produkto svetainė. Buvo naudojamasi svetainės forumuose pateikta informacija įvairiais žaidimo kūrimo klausimais, bei programavimo kodų pavyzdžiais.

1. <https://docs.unity3d.com/>

Oficiali „Unity“ žaidimų kūrimo variklio dokumentacijos svetainė. Šioje svetainėje galima rasti visą informaciją susijusią su „Unity“ kodavimo teorine ir praktine dalimis.

1. <https://www.youtube.com/>

Svetainė, kurioje galima rasti įvairios vaizdinės medžiagos apie žaidimų kūrimą, „Unity“ žadimų variklį, garso takelių kūrimą, spraitų animaciją.

1. [www.mortalkombatwarehouse.com/](http://www.mortalkombatwarehouse.com/)

Svetainė, kurioje yra sutelkta visa informacija apie vieno iš analizuoto 2D muštynių žaidimo animacijas, garso takelius, efektus ir pan.

1. <https://stackoverflow.com/>

Svetainė, kurioje galima rasti įvairios informacijos mechanikų programavimo klausimais.

1. UŽDUOTIES FORMULAVIMAS

Šio projekto tikslas yra sukurti 2D muštynių kompiuterinį žaidimą, kuris reikalatu iš žaidėjo greitos reakcijos. Žaidėjas valdydamas pagrindinį veikėja su kairiu ir dešiniu pelės mygtukais kovoja su priešais besiklausant adrenaliną keliančios muzikos.

Norint įgyvendinti šį projekta reikia atlikti šiuos uždavinius:

Analitinė dalis:

* Kruopščiai išanalizuoti du ar daugiau žaidimų, iš kurių bus imtasi įkvėpimo šiam projektui.
* Sugalvoti ir aprašyti žaidimo dizainą, grafinį stilių, veikėjus, aplinką
* Sugalvoti žaidimo mechanikas

Projektinė dalis:

* Pagrindinio veikėjo spraitų piešimas ir animacija.
* Priešų spraitų piešimas ir animacija.
* Žaidimo aplinkos spraitų piešimas ir animacija.
* Kitų spraitų piešimas
* Muzikinio takelio kūrimas.
* Žaidimo mechanikų programavimas
* Vartotojo sąsajos kūrimas.
* Žaidimo testavimas.
* Parengtą projekta paruošti naudojimui pasirinktai platformai.

Šiem darbam atlikti bus naudojama ši programinė įranga:

* Adobe PhotoShop – spraitų piešimas ir animacija.
* Unity variklis – žaidimo kūrimas.
* MonoDevelop – programinio kodo rašymas.
* FruityLoops Studio 12 – muzikinio takelio kūrimas.

1. UŽDUOTIES ANALIZĖ
   1. Žaidimų analizė
2. One Finger Death Punch



**1 pav**. „One Finger Death Punch“ plakatas



**2 pav**. Viena iš „One Finger Death Punch“ veiksmo scenų.

**Žanras**: 2D veiksmo, Beat‘em up žaidimas.

**Kūrėjas**: Silver Dollar Games.

**Leidėjas**: Silver Dollar Games, Mobirix Co., Ltd

**Išleidimo data**: 2013 m. birželio 23 d.

**Platformos**: Android, iOS, Microsoft Windows, Xbox 360

Didžiulės sekmės susilaukęs ir pasižymintis itin paprasta žaidimo idėja „One Finger Death Punch“ dar iki šių dienų išlieka vienu geriausių „Beat‘em up“ stiliaus žaidimų. Šis

žaidimas, sukurtas keturių žmonių komandos, žaidėją labai greitai įtraukia į veiksmą deka paprastų, tačiau efektyvių vaizdinių efektų, garso takelio ir nostalgiją sukeliančiais „Stickman“ veikėjais, kurie atmenami iš ankstesniųjų interneto dienų, besikaunančiais įvairiais tolimųjų rytų kovų menų stiliais.

**Žaidimo mechanikos**.

Pagrindinė žaidimo mechanika, nors paprasta, išnaudojama vos ne tobulai – žaidėją puola armija priešų iš abiejų ekrano pusių, kurią žaidėjas turi nugalėti naudojantis kairiuoju ir dešiniuoju pelės mygtukais, kai priešai priena pakankamai arti. Žaidimo veiksmas vyksta greitai ir ,priklausant nuo žaidėjo gabumų, vis greitėja. Taip vartotojas yra įtraukiamas į žaidimą ir nenorima atsitraukti, nebaigus lygio. Kad žaidimas nebūtu labai pasikartojantis, žaidėjas susiduria su 18 skirtingo tipo priešais:

* Pilkas priešas turintis 1 gyvybę,
* Žalias priešas turintis 2 gyvybes,
* Raudonas priešas turintis 3 gyvybes,
* Mėlynas priešas turintis 2 gyvybes, iš kurių praradęs vieną peršoka žaidėją ir puola iš kitos pusės,
* Geltonas priešas turintis 3 gyvybes, iš kurių praradęs dvi peršoka žaidėją ir puola iš kitos pusės,
* Oranžinis priešas turintis 3 gyvybes, iš kurių praradęs primąją peršoka žaidėją ir puola iš kitos pusės,
* Violetinis priešas turintis 3 gyvybes, iš kurių praradęs kiekvieną peršoka žaidėją ir puola iš kitos pusės,
* Šviesiai žalias priešas turintis 4 gyvybes,
* Rožinis priešas turintis 4 gyvybes, iš kurių praradęs dvi peršoka žaidėją ir puola iš kitos pusės,
* Tamsiai žydras priešas, turintis 4 gyvybes, iš kurių praradęs tris peršoka žaidėją ir puola iš kitos pusės,
* Tamsiai rožinis priešas, turintis 4 gyvybes, iš kurių praradęs pirmąją ir antrąją gyvybes peršoka žaidėją ir puola iš kitos pusės,
* Šviesiai žalias priešas, turintis 4 gyvybes, iš kurių praradęs pirmąją peršoka žaidėją ir puola iš kitos pusės,
* Šviesiai rudas priešas, turintis 4 gyvybes, kuris praradęs trečiąją gyvybę peršoka žaidėją ir puola iš kitos pusės,
* Tamsiai žalias priešas, turintis 4 gyvybes, kuris kiekvieną kartą praradęs gyvybę peršoka žaidėją ir puola iš kitos pusės,
* Tamsiai raudonas priešas, turintis 4 gyvybes, kuris praradęs pirmąją ir trečiąją gyvybes peršoka žaidėją ir puola iš kitos pusės,
* Juodasis priešas, kuris gali tureti bet kurias kito priešo savybes, tačiau žaidėjas to nežino,
* Kovotojas priešas, kurį galima atpažinti iš jo karūnos virš jo galvos. Šis priešas gali turėti 4 arba 25 spaudimų kovos sceną, kurioje žaidėjas privalo tam tikra tvarka suspaudinėti pelės mygtukus.
* Bosas, kuris kiekviena kartą keičia spalvą atitinkančia kito priešo, kai praita spalva buvo nugalėta. Šis procesas kartojasi 5 arba 10 kartu priklausant nuo žaidimo sunkumo lygio.

6 skirtingo tipo ginklais, iš kurių kiekvienas turi savo ypatumų:

* Lazda – padidina žaidėjo atakos atstumą, turi savas animacijas.
* Kardas – padidina žaidėjo atakos atsumą, tačiau ne tiek, kiek lazda, bet turi daugiau atakų iki susidevėjimo, turi savas animacijas.
* Kūjis – Nedaug padidina žaidėjo atakos atsumą, tačiau susidėvi lečiausiai.
* Lankas – Žaidėjas iššuna strėlę, kuri užmušą pirmąjį palietusį priešą.
* Mėtomas durklas – Žaidėjas meta durkla, kuris užmuša pirmą palietusį priešą. Durklas juda greičiau nei strelė.
* Bomba – Žaidėjas meta bomba, kuri užmuša pirmą palietusį priešą.

Kiekvieną raundą žaidėjui atsitiktinai išrenkamas vienas iš 5 skirtingų Kung-fu stilių, nuo kurio priklausys kokios animacijos bus naudojamos. Žaidėjas taip pat gali atrakinti specialius judesius ar patobulinimus istorijos režime. Šie specialūs judesiai skiriasi nuo pasigydymo iki atakos atstumo pailginimo ar net ekrane matomų priešų išnaikinimo, tačiau žaidėjas neturi pasirinkimo kada naudoti šiuos judesius, jie panaudojami automatiškai nugalėjus tam tikrą skaičių priešų, pavyzdžiui, kad pasigydyti, žaidėjas turi nugalėti 99 priešus.

Žaidimas taip pat turi specialius atvėjus, kurie veikia atsitiktinai, užmušus priešą (žr. 3 pav.). Tokių įvykių žaidime yra pakankamai, kad jaustusi orginalūs, ir per daug nepasikartojantys.



**3 pav**. Laiko sulėtinto spyrio su X-Ray effektu specialus įvykis.

Žaidimas turi 13 skirtingų žaidimo režimų, iš kurių kiekvienas savaip išbando žaidėjo gabumus. Režimai skiriasi nuo pagrindinio „Minios raundo“, kuriame žaidėjas privalo nugalėti visus priešus, iki, gerbejų mėgstamiausio, „Šviesos kardo raundo“, kuriame žaidėjui duodamas šviesos kardas, ir jis privalo nugalėti labai greit bepuolančius priešus (žr. 4 pav.).



**4 pav**. Scena iš „Šviesos kardo“ raundo

**Žaidimo aplinka.**

One Finger Death Punch veiksmas vyksta įvairiose vietovėse – nuo paprasto kiniško kaimelio iki sprogstančio ugnikalnio. Kiekviena iš žaidimo teritorijų turi savus sunaikinamus objektus, kurie, užmušus netoliese prieša, sužaidžia tam tikras animacijas, pavyzdžiui, priešas pramušamas pro namo sieną.

**Žaidimo muzika.**

One Finger Death Punch neturi didelio muzikos garsų takelio sarašo, tačiau naudojama muzika labai dera ir pritampa prie ekrane matomo veiksmo. Žaidimas turi 9 skirtingus muzikos takelius. Naudojama muzika yra roko žanro sujungta su tolymųjų rytų kultūriniais instrumentais, kas sudaro įspūdį, kad žaidėjas iš tiesų yra rytų kovų menų meistras.

1. Ultimate Mortal Kombat 3



**5 pav**. Ultimate Mortal Kombat 3 plakatas.



**6 pav**. Kova tarp dviejų ninzių veikėjų.

**Žanras**: Muštynių žaidimas.

**Kūrėjas**: Midway

**Leidėjas**: Midway

**Išleidimo data**: 1995 m. lapkričio 6 d.

**Platformos**: Arcade, Sega Saturn, Sega Genesis, Super NES, Game Boy Advance, Nintendo DS, iOS, Playstation 2, Windows.

Ultimate Mortal Kombat 3 yra muštynių žaidimas iš Mortal Kombat serijos, sukurtas ir išleistas studijos Midway 1995 m. Tai yra ketvirta instaliacija Mortal Kombat serijoje, kuri atnaujina ir patobulina ankstesnįjį Mortal Kombat 3. Savo laiku, Mortal Kombat buvo vienas geriausiai grafiškai atrodančių žaidimų ir dauguma jį laiko turinti nesanstantį stilių, tačiau labiausiai Mortal Kombat žaidimai atmenami del jo smurtinių veiksmų ir Ultimate Mortal Kombat 3 iš tiesų žaidėjams tai ir pristato – 22 skirtingi veikėjai, iš kurių kiekvienas turi 3 savo žymes paliekančius mirtinus judėsius, kurie padaros pergales tik dar saldesnes ir žaidėjus priverčia norėti žaisti daugiau ir daugiau.

**Žaidimo mechanikos.**

Ultimate Mortal Kombat 3 mechanikos šiais laikais turbūt niekeno nenustebintu. Žaidimo idėja yra nugalėti priešininką naudojant įvairias atakas, jų kombinacijas ir specialias atakas. Žaidėjui suteikiamas paprastas judėjimo valdymas, auškto smūgio ataka, žemo smūgio ataka, aukšto spyrio (žr. 6, 8 pav.) ir žemo spyrio atakos, bei šių atakų variacijos priklausant ar veikėjas pašokęs į orą ar pritūpęs. Kiekvienas iš 22 veikėjų turi skirtingas animacijas kiekvienai iš savo atakų ir jų kombinacijoms, bei specialiom atakom. Žaidėjas nugalėjęs savo priešą turi kelių sekundžių laiko tarpą įvesti specialiai judesių kombinacijai, kad žaidėjo veikėjas įvykdytu mirtiną ataką priešui (žr. 7 pav.), kas yra pagrindinė šios frančizės žymė.



**7 pav.** Vienas iš veikėjo Sheeva mirtinų atakų

**Žaidimo aplinka.**

Ultimate Mortal Kombat 3 pasižymi kruopščiai nuopieštomis ir animuotomis arenomis. Žaidimo veiksmas vyksta keliose vietose, kaip, pavyzdžiui, metro stotyje, arba pragaro dimencijoje (žr. 6 pav.).

**Žaidimo muzika.**

Muzika šiame žaidime dėmesio nepatraukia, tačiau tikrai suteikia kiekvienai arenai savitą jausmą, ir leidžia žaidėjui pasinerti į veiksmą. Žaidime yra 23 skirtingi muzikos takeliai, kurie neturi tam tikro žanro, tačiau įkvėpimo traukia iš 90 – ųjų garsų sintezatorių sujungta su rytų kovų menų instrumentais.

**Žaidimo animacija.**

Mortal Kombat serija visada išsiskyrė iš kitų muštynių žaidimų savo savitomis animacijomis ir jų kūrimo būdu. Spraitų animacijos ir kūrimo procesas vyko taip, kad buvo pasamdyti tikri aktoriai, kurie vykdytu skirtingas pozas priešais žaliąjį ekraną. Šitoks metodas leido, tais laikais labai ribotom kompiuterinėm mašinoms, sukurti itin tikroviškas ir patikimas žaidėjui animacijas (žr. 8 pav.).



**8 pav.** Veikėjo Jax aukšto spyrio animacijos kadrai.

* 1. Naudojamos programinės įrangos analizė

**Žaidimo kūrimo platforma.**

Šiam projektui įgyvendinti buvo pasirinkta „Unity“ žaidimų kūrimo variklis. Kadangi studentas turi daugiausia patirties su šia platforma, kitos netiko. „Unity“ savo vartotojams siūlo lengva prieigą pradedantiesiems kurti žaidimus. Vartotojams galima rinktis programuoti viena iš „Unity“ palaikomų programavimo kalbų: C# ir JavaScript, iš kurių buvo naudojama C# programavimo kalba.

**Spraitų piešimo programa.**

Nors šiomis dienomis yra daugelis programų, skirtų būtent 2D kompiuterinių žaidimų spraitų piešimui ir animacijai, buvo pasirinkta Adobe PhotoShop programa, dėl studento turimos patirties. Adobe PhotoShop savo vartotojams leidžia greit ir patogiai kurti 2D spraitus ir dėl Unity galimybės naudoti Adobe PhotoShop .psd formato failus iš kart, šis pasirinkimas buvo labai priimtinas.

**Programavimo aplinka.**

Šiam projektui atlikti buvo pasirinkta kartu su „Unity“ žaidimų kūrimo platforma ateinanti „MonoDevelop“ programavimo aplinka. Nors Microsoft „Visual Studio“ turi labiau išvystytą vartotojo sąsają ir patogesnį valdymą, kaip, pavyzdžiui, greitųjų komandų iškvietimas, dėl sunaudojamų kompiuterio resursų ir ilgo krovimo „Visual Studio“ nebuvo tinkamas pasirinkimas.

**Muzikos kūrimo aplinka.**

Projekto muzikai kurti buvo pasirinkta „FruityLoops Studio 12“ programa. Ši programa yra viena populiariausių ir labiausiai išvystytų, norint kurti modernią muziką. Programa palaiko įvairius priedus, kurie leidžia vartotojams naudotis įvairiomis garsų kūrimo priemonėmis. Studentas taip pat turi patirties su šia programa.

1. PROJEKTINĖS DALIES APRAŠYMAS
   1. Žaidimo scenos

„BoneBreaker” turi dvi scenas: pagrindinis meniu (žr. 9 pav.) ir pirmasis žaidimo lygis.



**9 pav.** Žaidimo pagrindinis meniu.

**Pagrindinis meniu.**

Tai pirmoji scena, kurią pamato žaidėjas įjungęs žaidimą. Scenoje bandoma atvaizduoti ateivišką „Sci-fi” pasaulį – neoninėmis šviesomis apšviesti keisti pastatai, ir dešinėje ekrano pusėje matomas pagrindinis veikėjas, kiborgas, savo neutralios būsenos animacijoje.

Žaidėjui duodami trys pasirinkimo mygtukai:

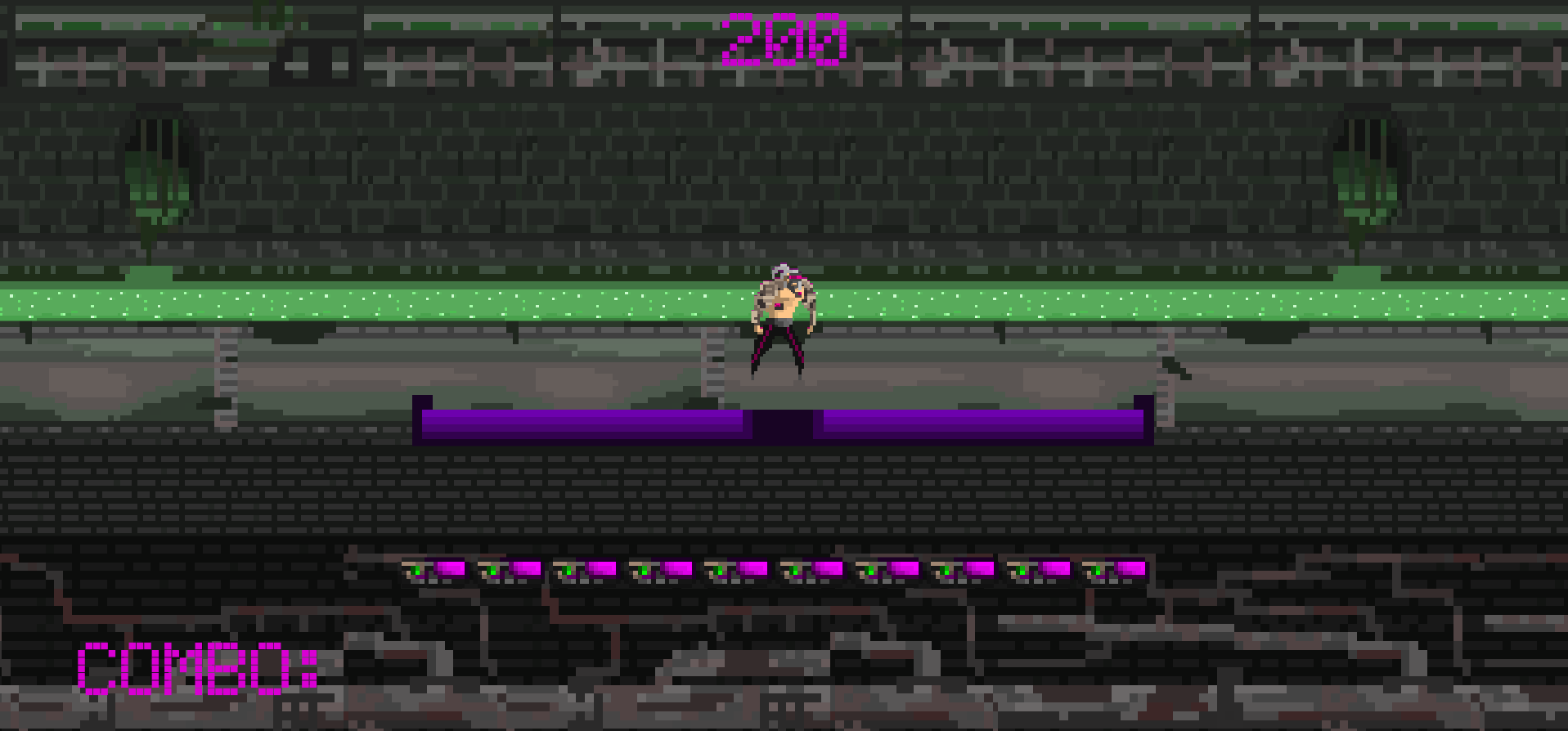
* “Play” - Pradedamas žaidimas. Scena perkeliama į pirmąjį žaidimo lygį.
* “Options” – Žaidimo nustatymai. Žaidėjui leidžiama nustatyti žaidimo ir muzikos garsą. (žr. 10 pav.)
* “Quit” – Baigiamas žaidimas ir uždaroma programa.



**10 pav.** Pagrindinio menių nustatymų langas.

Šios scenos metu grojama kelių žemų natų melodija. Mygtukai taip pat turi savo garsus, kai būna paspaudžiami ar virš jų užvedus pelę. Tai ledžia išbandyti žaidėjui kokio garso nustatymų jis norėtu.

**Pirmasis žaidimo lygis.**



**11 pav.** Pirmasis žaidimo lygis.

Pirmojo lygio veiksmas vyksta kanalizacijos sistemoje. Aplink matomos plaukiančios ir žibančios toksinės atliekos ir chemikalai, purvas ir surūdije vamzdžiai. Scenos aplinka sudaryta iš 5 pagrindinių 2D modelių (žr. 1 lentelę.), kurie yra kopijuojami į kairę ir dešinę ekrano puses, priklausant nuo veikėjo judėjimo krypties.

**1 lentelė.** Pirmos scenos aplinkos 2D modeliai.

|  |  |
| --- | --- |
| Viršutinės kanalizacijos grotos ir vamzdžiai. |  |
| Kanalizacijos siena ir iš jos tekančios atliekos ir chemikalai. Šis objektas yra animuotas – iš matomo įtekėjimo dešinėje sienos pusėje varva chemikalai. |  |
| Scenos toksinės atliekos ir chemikalai. Objektas yra animuotas – atliekos žiba ir banguoja. |  |
| Kanalizacijos platforma, ant kurios vyksta pagrindinis žadimo veiksmas. |  |
| Apatiniai kanalizacijos vamzdžiai po žeme. |  |

Ši aplinka pasirinkta dėl įkvėpimo iš vieno „Ultimate Mortal Kombat 3” lygių. Buvo bandoma sukurti nemalonią tamsią aplinką, kurioje matytusi rožinės spalvos neoniniai žaidimo efektai.

* 1. Žaidimo valdymas ir mechanikos

Visas žaidimas valdomas tik kairiu ir dešiniu pelės mygtukais. Paspaudus vieną iš mygtukų, žaidėjo valdomas veikėjas atlieka tam tikrą veiksmą ir kairę ar dešinę pusę, priklausant nuo paspausto mygtuko.

Pagrindinė žaidimo mechanika yra priešų naikinimas, jiem priėjus pakankamai arti prie žaidėjo, kuriam paspaudus atitinkantį pelės mygtuką bus puolamas arčiausias iš mygtuko pusės esantis priešas.

**Pagrindinio veikėjo valdymo mechanikos.**

Paspaudus vieną iš pelės mygtuku veikėjas atlieka viena iš dviejų veiskmų: puola priešą arba, per anksti paspaudus mygtuka ar paspaudus ne tą mygtuką, nepataiko. Žaidėjui leidžiama pulti prieša, kai jis priartėja pakankamai arti veikėjo ir raudona spalva užsidega indikatoriai, kurie parodo veikėjo atakų atstumą. Užsidegus indikatoriui ir žaidėjui paspaudus atitinkantį pelės mygtuką, veikėjas greitai priartėja prie priešo, panaudojama viena iš veikėjo puolimo animacijų, ir priešui sudaroma žala (žr. 12 pav.).



**12 pav.** Žaidėjas puola prieša jam priėjus pakankamai arti.

Žaidėjui per anksti paspaudus pelės mygtuką ar paspaudus ne tą mygtuką, veikėjas yra apsvaiginamas tam tikram laikui, kurio metu žaidėjas negali valdyti veikėjo ir jis yra pažeidžiamas priešų atakom (žr. 13 pav.).



**13 pav.** Veikėjas yra apsvaigęs, nes puolė į kairę pusę.

**Pagrindinio veikėjo gyvybių ir kombinacių atakų mechanikos.**

Žaidimas pradedamas turint 10 gyvybių(žr. 14 pav.), kurias žaidėjas praranda priešam sudarus žalos. Netekus visų gyvybių žaidimas yra sustabdomas ir žaidėjas sutinkamas su žaidimo baigimo meniu (žr. 15 pav.).

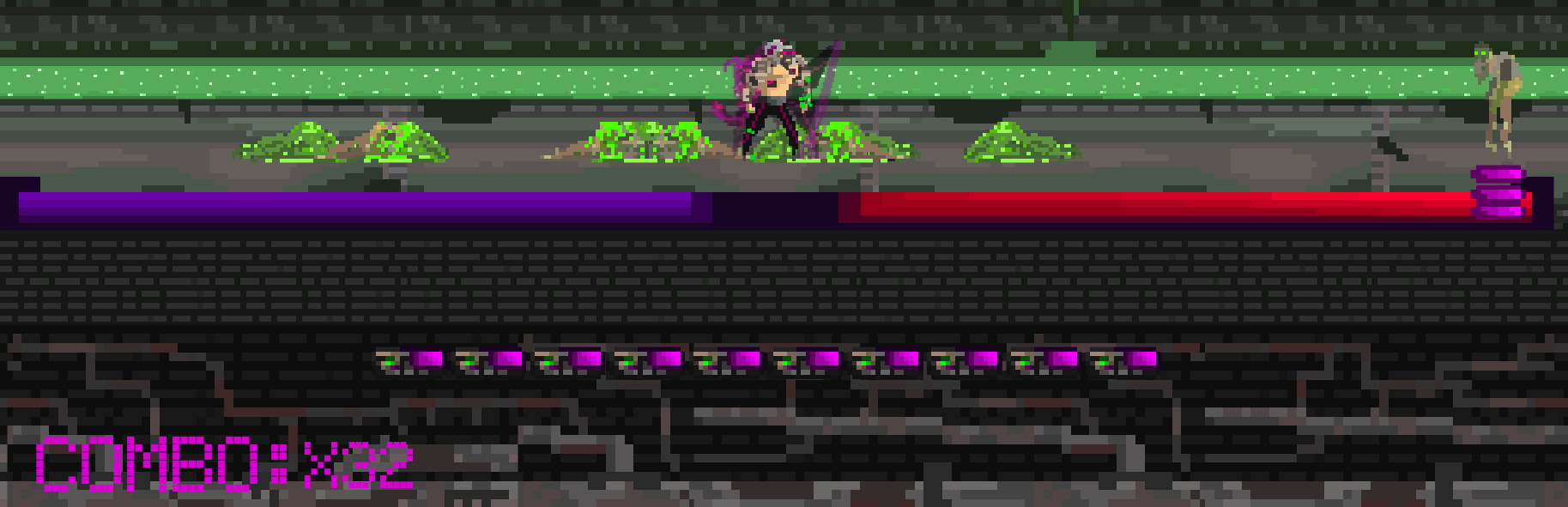


**14 pav.** Pagrindinio veikėjo gyvybės.



**15 pav.** Žaidėjas neteko dešimties gyvybių ir žaidimas sustabdomas.

Žaidėjui sėkmingai iš eilės puolus priešus, pildosi kombinacinių atakų skaičius, kuriom pasiekus tam tikrą kiekį atliekamą tam tikrą funkcija padedanti žaidėjui. Žaidėjui atlikus 30 kombinacinių atakų, veikėjui suteikiamos ilgesnis atakų atstumas, kol nesunaikinti 5 priešai (žr. 16 pav.). Atlikus 50 kombinacinių atakų, veikėjas yra pagydomas vienu gyvybės tašku. Jeigu veikėjas turėjo visas 10 gyvybių, niekas neatsitinka.



**16 pav.** Žaidėjas atliko 30 kombinacinių atakų ir yra suteikiamas ilgesnis atakų atstumas.

**Priešų tipai ir jų mechanikos.**

Žaidime yra 2 pagrindinio tipo priešai: zombiai ir gleivės, kurie dar išskirstomi į kelius tipus, kurie nustatomi pagal priešų gyvybes. Priešui mirus, jo vietoje sukuriama “lelė”, kuri atlieka mirties animacija ir lieka gulėti iki žaidimo pabaigos.

* **Zombis.**



**17 pav.** Priešas zombis.

Zombiai turi 2 arba 3 gyvybes, kurias žaidėjui puolant zombį, yra sunaikinamos. Netekus visų gyvybių zombis miršta ir parodoma viena iš dviejų zombio mirties animacijų. Zombiui priartėjus prie pat veikėjo pradedamas skaičiuoti laikas iki zombie galimos atakos. Praėjus tam laikui zombis atlieka atakos animaciją, kurios gale žaidėjui sudaroma žala.

* **Gleivė.**



**18 pav.** Priešas gleivė.

Gleivės turi ypatumą peršokti žaidėją ir pulti iš kitos puses, netekus tam tikra kiekį savo gyvybių. Šios gleivės yra 4 tipų, kurie atpažinami pagal jų gyvybių indikatorių spalvą (žr. 18 pav.). Tos spalvos parodo kada ir iš kurios pusės gleivė puls žaidėją. Rožinė spalva reiškia, kad gleivė puls iš tos pusės, kurios ateina, o žydra spalva reiškia, kad, turint likus tos spalvos gyvybei, peršoks žaidėją (žr. 19 pav.). Šia mechanika žaidėjas priverčiamas nuolat laikyti dėmesį ir nepulti gleivių aklai. Gleivių atakos veikia taip pat kaip zombių. 

**19 pav.** Žaidėjui puolus gleivė ateinančia ir dešinės pusės, ji peršoka į kitą pusę.

**Žaidimo lygio mechanikos.**

Žaidimo lygis trunka kol nesunaikintas tam tikras priešų skaičius, kuris nurodomas ekrano viršuje (žr. 20 pav.). Kiekviena kart sunaikinus priešas, žaidimas yra pagreitinamas: priešai greičiau juda, greičiau atsiranda, ir trumpiau laukia kol pradės pulti, priartėjus prie žaidėjo.



**20 pav.** Ekrano viršuje nurodomas skaičius, kiek liko sunaikinti priešų.

* 1. Žaidimo veikėjų spraitai ir animacijos.

**Pagrindinis veikėjas.**

Pagrindinis veikėjas turi 3 tipų spraitų animacijas: neutrali būsena, puolimas ir apsvaigimas.

Neutrali animacija yra sudaryta iš 8 kadrų(žr. 21 pav.).



**21 pav.** Neutralios būsenos spraitai.

Veikėjas turi 5 skirtingas puolimo animacijas (žr. 22 pav.)

**22 pav.** Pagrindinio veikėjo puolimo spraitai.

Viena apsvaigimo animacija (žr. 23 pav.).



**23 pav.** Apsvaigimo spraitai.

**Priešų spraitai ir animacijos.**

Priešas zombis turi 8 kadrų judėjimo animaciją, vieną puolimo animaciją ir dvi mirties animacijas.



**24 pav.** Zombio judėjimo spraitai.

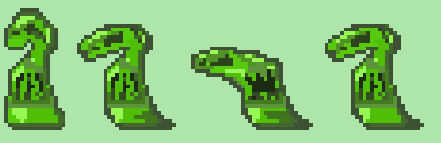


**25 pav.** Zombio atakos spraitai.

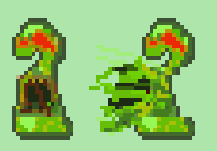


**26 pav.** Zombio mirčių spraitai.

Priešas gleivė turi 4 kadrų judėjimo animaciją, viena atakos animaciją ir vieną mirties animaciją.



**27 pav.** Gleivės judėjimo spraitai.



**28 pav.** Gleivės atakos spraitai.



**29 pav.** Gleivės mirties spraitai.

* 1. Žaidimo garsai ir muzika.

Žaidimo garsui buvo panaudotas „Ultimate Mortal Kombat 3” garsų paketas. Naudojami mygtukų paspaudimo, pelės užvedimo, atakų ir mirčių garsai.

„BoneBreaker” turi 2 muzikinius takelius sukurtus naudojant Fruity Loops Studio 12 programą. Pirmasis takelis yra girdimas pagrindiniame meniu. Tai yra kelių žemų natų melodija. Antrasis takelis girdimas pirmame žaidimo lygyje. Takelio paskirtis yra sukelti žaidėjui adrenalino jausmą ir suteikti žaidimui gyvybingumo.

* 1. Žaidimo programinė dalis.

Žaidimo programavimui buvo pasitelkta C# kalba naudojant „Unity” integruota programavimo aplinka „MonoDevelop”. Žaidime naudojama 14 skirtingų programinių skriptų žaidimo funkcijom atlikti. Visas naudojamas programinis kodas pateikiamas 1 priede.

Skriptai:

* AudioScript.cs. Skriptas saugo ir groja atitinkamus žaidimo garsus ir muziką.
* BackgroundTiling.cs. Skriptas kuria naujus aplinkos objektus priklausant nuo žaidimo kameros pozicijos.
* EnemyController.cs. Skriptas, kuris valdo priešus, nustato jų gyvybes, valdo jų atakas, mirtis ir kitus nustatymus.
* EnemyZoneTrigger.cs. Skriptas, kuris žiuri priešo nuo žaidėjo atstumo indikatorių.
* GameSpeedScript.cs. Skriptas, kuris keičia žaidimo greitį.
* MainMenu.cs. Skriptas, kuris nustato pagrindinio menių mygtukų funkcijas.
* PauseMenu.cs. Skriptas, kuriame nurodomos pauzės meniu funkcijos.
* PlayerAttackZoneTrigger.cs. Skriptas kuriame saugomi žaidėjo puolimo indikatoriaus funkcijos.
* PlayerController.cs. Skriptas, kuriame nustatomas žaidėjo valdymas, animacijos, atakos ir kombinacinių atakų funkcija.
* PlayerStats.cs. Skriptas, kuris tikrina žaidėjo gyvybes ir tikrina ar žaidėjas mirė ar ne.
* SpawnEnemy.cs. Skriptas, kuriame pasitelkiamas „Object Pooling” metodas priešų kūrimui ir jų aktyvavimui.
* WaveCounter.cs. Skriptas, kuris rūpinasi lygio eiga ir vaizduojama vartotojo sąsaja.
* ZombieDeathAnimationPicker.cs. Skriptas, kuris suteikia zombie priešo lėlėi viena iš dviejų mirties animacijų.

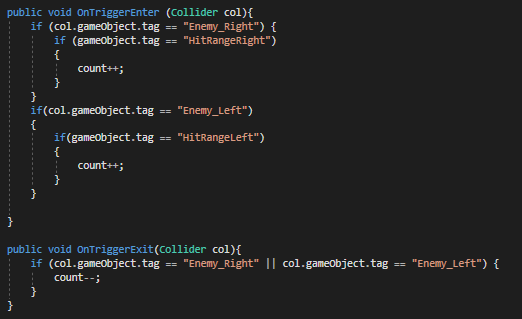
Svarbiausios žaidimo funkcijos:

* DamagePlayer() ir HealPlayer() fukcijos, randamos PlayerStats.cs skripte (žr. 30 pav.).



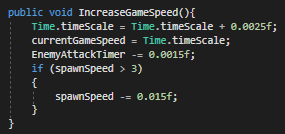
**30 pav.** DamagePlayer() ir HealPlayer() funkcijos.

* PlayerAttackZoneTrigger.cs indikatorių jungiklio funkcijos. Priešui užėjus virš atakos indikatoriaus nustatomas galimų pulti priešų skaičius tame indikatoriuje. (žr. 31 pav.)



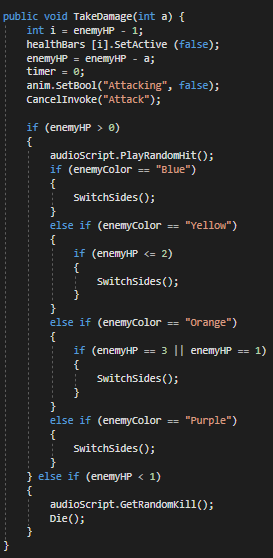
**31 pav.** PlayerAttackZoneTrigger.cs indikatorių funkcijos.

* Žaidimo greitinimo funkcija, randama GameSpeedScript.cs skripte. Kiekvieną kartą mirus priešui yra iškviečiama ši funkcija, kuri pagreitina žaidimo greitį. (žr. 32 pav.)



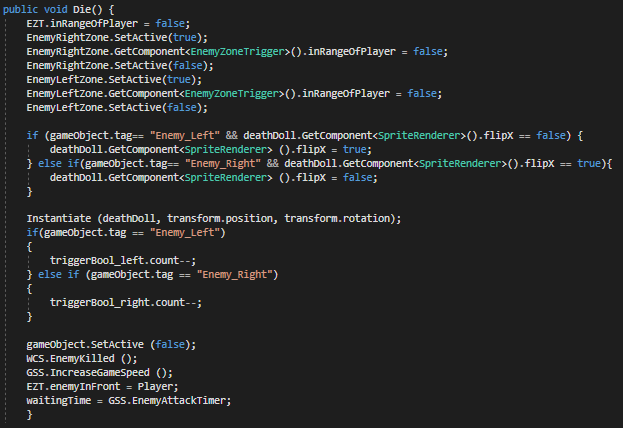
**32 pav.** GameSpeedScript.cs žaidimo greitinimo funkcija.

* TakeDamage() funkcija, kuri skaičiuoja priešo žalą ir, atitinkant jo tipą, atlieka papildomus veiksmus, bei iškviečiama priešo mirties funkcija. Randama EnemyController.cs skripte. (žr. 33 pav.)



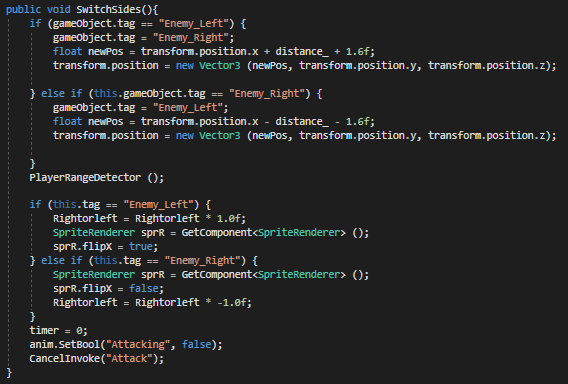
**33 pav.** Priešo TakeDamage() funkcija.

* Die() funkcija, kuri atlieka reikiamus veiksmus įvykus priešo mirčiai. Iškvietus šia funkcija gražinami priešo pradiniai nustatymai, sukuriama priešo mirties “lelė”, pasakoma pagrindinio veikėjo atakos indikatoriui, kad priešas mire ir atnaujinama reikiama žaidimo eigos informacija. (žr. 34 pav.)



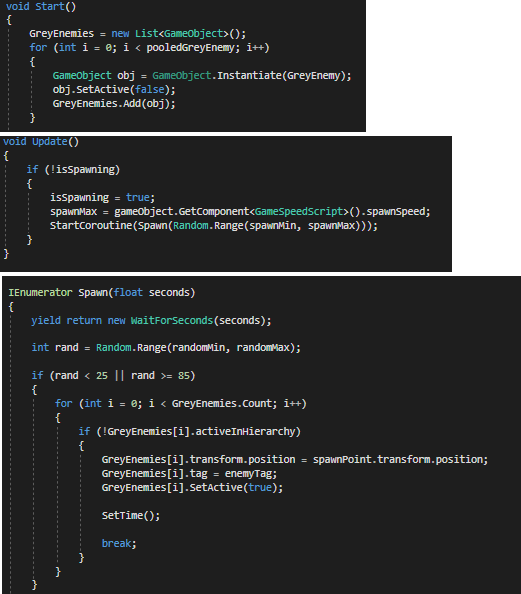
**34 pav.** Priešo mirties Die() funkcija.

* SwitchSides() funkcija, kurioje aprašoma gleivės priešo peršokimo funkcija. (žr. 35 pav.)



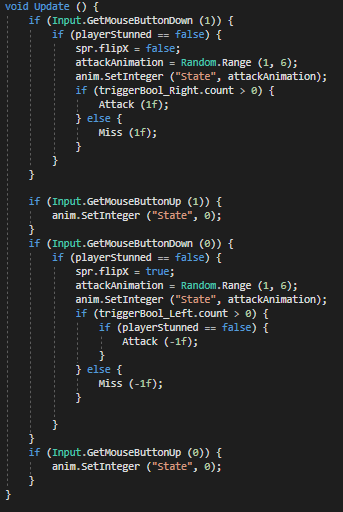
**35 pav.** SwitchSides() funkcija.

* Priešų kūrimo scenoje kodas, pasitelkiant Object Pool metodą. Randamas SpawnEnemy.cs skripte(žr 36 pav.)



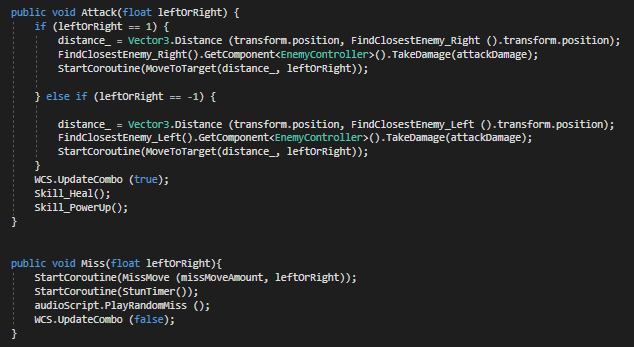
**36 pav.** Vieno iš priešo tipo kūrimas scenoje naudojant Object Pool metodą.

* Pagrindinio veikėjo valdymas. Iškviečiamos reikiamos funckijos paspaudus viena iš pelės mygtukų. Randamas PlayerController.cs skripte. (žr. 37 pav.)



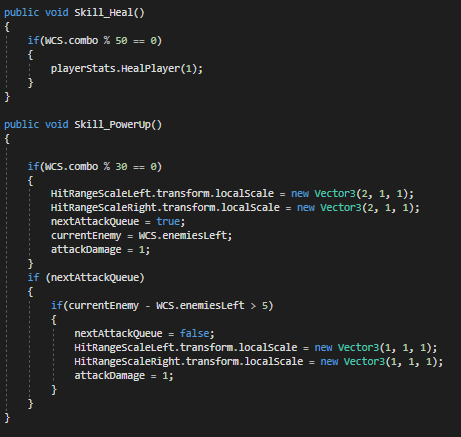
**37 pav.** Pagrindinio veikėjo valdymas.

* Pagrindinio veikėjo Attack() ir Miss() atakos ir nepataikymo fukcijos. (žr. 38 pav.) Atakos metu surandamas atstumas, kurį turės nukeliauti žaidėjas iki priešo, tada jam pirma sudaroma žala, poto tik pradeda judėti pagrindinis veikėjas. Dėl žaidimo sklandumo teko butent taip rikiuoti funkcijas.



**38 pav.** Pagrindinio veikėjo atakos ir nepataikymo funkcijos.

* Žaidėjo pagydimas ir atakos atstumo pailginimo funkcijos, kai žaidėjas atlieka tam tikrą skaičių kombinacinių atakų. (žr 39 pav.)

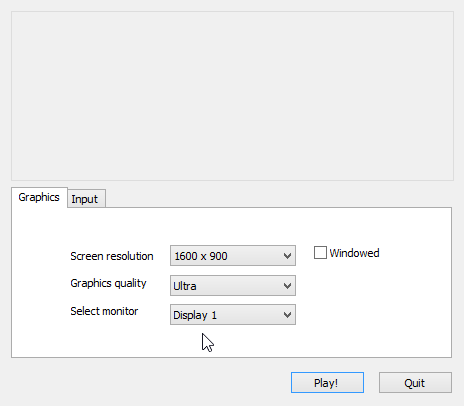


**39 pav.** Žaidėjo pagydimo ir atakos atstumo pailginimo funkcijos.

1. NAUDOTOJO INSTRUKCIJA
   1. Žaidimo diegimas.

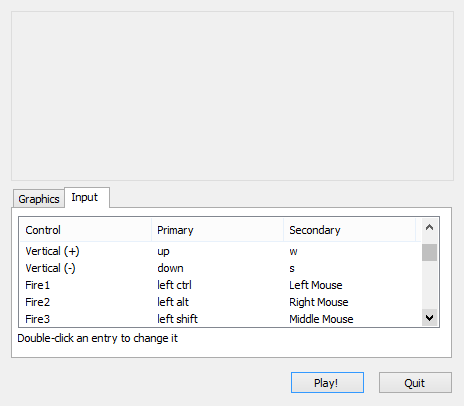
Atsisiuntus žaidimo **BoneBreaker.rar** failą, failo duomenys išimami ir patalpinami į norimą kompiuterio vietą. Žaidimas įjungiamas paspaudus **BoneBreaker.exe** failą. Paleidus žaidimo vykdomajį failą, naudotojas sutinkamas su Unity konfiguracijos langu, kuriame naudotojas gali pasirinkti ar keisti tam tikrus nustatymus (žr 40 pav.), kaip:

* Žaidimo ekrano rezoliucija
* Žaidimo ekrano režimas
* Žaidimo grafikos kokybė
* Monitoriaus pasirinkimas



**40 pav.** BoneBreaker.exe konfiguracijos langas.

„Input” skyriuje naudotojas gali keisti „Unity” variklio suprantamą valdymą. Šiuo atvėju reikėtu keisti Fire1 ir Fire2 nustatymus, norint pakeisti “BoneBreaker” valdymą (žr. 41 pav.)



**41 pav.** Žaidimo valdymo konfiguracinis langas.

Žaidimas pradedamas paspaudus „Play!” mygtuką. Pasirinkus „Quit“ mygtuka, uždaromas konfiguracijos langas ir žaidimas nepradedamas.

* 1. Žaidimo išdiegimas.

Norint išdiegti žaidimą, naudotojas turi pasirinkti iš **BoneBreaker.rar** failo ištrauktus failus ir juos sunaikinti paspaudus klaviatūros *Shift + Del* klavišus.

* 1. Žaidimo apribojimai.
* Žaidimas skirtas asmeninio kompiuterio platformoms, turinčioms 16:9 ekrano rezoliucijos mastelį.
* Žaidimas turi vieną lygį.
* Žaidime nėra taškų sistemos

1. REZULTATAI

Atlikus darbą gauti šie rezultatai:

1. Atlikta 2D muštynių žaidimų apžvalga ir analizė
2. Apžvelgta ir išsirinkta projektui naudota programinė įranga.
3. Pasitelkus turimas ir naujai įgytas teorines ir praktines žaidimų kūrimo žinias, buvo sukurtas 2D muštynių žaidimas reikalaujantis greitos reakcijos, kuriam buvo:
   * Nupiešti ir animuoti 2D spraitai.
   * Sukurti 2 garso takeliai.
   * Suprogramuotos žaidimo mechanikos.
4. IŠVADOS
5. Žaidimų rinkoje būtent tokio tipo 2D muštynių žaidimų yra tik vienetai, dėl to gauti viešos informacijos apie tų žaidimų mechanikas buvo neįmanoma. Tai reiškias, kad pagrindinės žaidimo mechanikos ir jų idėjos yra visiškai orginalios.
6. Projekto idėja skamba paprastai, tačiau igyvendinti tai buvo daug sunkiau. Susidurta su įvairiais žaidimo sklandumo ir optimizavimo klausimais, kurių atvėjais buvo atlikti kompromisai.
7. Projektas galimas tobulinti ateičiai naujomis idėjomis, bei gali būti pritaikytas papildomom platformom, kaip „Android“ ar „iOS“.
8. Žaidimas lengvai prieinamas naujam vartotojui.

**VILNIAUS VERSLO KOLEGIJA**

**INFORMACINIŲ TECHNOLOGIJŲ KATEDRA**

**ŽAIDIMŲ KŪRIMO STUDIJŲ PROGRAMA**

**2D MUŠTYNIŲ ŽAIDIMAS „BoneBreaker“**

*BAIGIAMOJO DARBO ANOTACIJA*

Galaktion Davyd Ušakov

Kalba: Lietuvių

Šio baigiamojo darbo tikslas buvo sukurti 2D muštynių kompiuterinį žaidimą. Kadangi vienas iš dažniausių muštynių žaidimų reikalavimų yra greita reakcija, šio projekto dėmesio centras būtent tai ir yra – greita reakcija, žaidėjo gebėjimas nepanikuoti ir ramiai priimti optimalius sprendimus. Žaidimo konceptas yra, kad žaidėja puola priešai, iš kairės ir dešinės ekrano pusių, kuriuos žaidėjas, naudodamas kairį ir dešinį pelės mygtukus, turi sunaikinti. Tačiau kad žaidimas iš tiesų būtu įdomus ir linksmas žaisti, būtina sukurti sklandžia žaidimo eigą, sąžiningas žaidimo mechanikas, tinkamus garsus ir muziką, bei patrauklią grafiką. Galutiniame projekto rezultate būtent tai ir buvo atlikta: naudojant „Unity“ žaidimų variklį sukurtas žaidimas, turintis sąžiningas ir suprantamas žaidimo mechanikas, yra lengvai prieinamas naujam vartotojui, panaudoti seno tipo 16 bitų muštynių garsai, naudojant „Fruity Loops Studio 12“ sukurtas adrenaliną keliantis muzikinis takelis, kuris dera prie žaidimo aplinkos ir, „Photoshop“ programoje, nupiešti „Pixel“ grafikos stiliaus 2D objektai. Šiuos dalykus sujungus gaunamas įdomus ir linksmas, greito veiksmo 2D muštynių žaidimas.

**VILNIAUS BUSINESS COLLEGE**

**DEPARTMENT OF INFORMATION TECHNOLOGIES**

**GAME DESIGN STUDY PROGRAM**

**2D FIGHTING GAME „BoneBreaker“**

*FINAL WORK ANNOTATION*

Galaktion Davyd Ušakov

Language: English

The objective of this project was to create a 2D fighting video game. Because one of the most common demand of fighting games is a fast reaction, the focus point of this project is exactly that – fast reaction, the player‘s ability to not panic and calmly make optimal decisions. The concept of this video game is, that the player is being attacked by enemies, from the left and right sides of the screen, whom the player, using the left and right mouse buttons, has to destroy. However, in order for the game to be actually interesting and fun to play, it has to have smooth gameplay, fair game mechanics, appropriate sounds and music and also graphics. What has been done in the final result of this project is exactly that: using the „Unity“ game engine a game was created, which has fair and understandable game mechanics, is easily accessible to a new player, is using old-style 16bit fighting sounds, using the „Fruity Loops Studio 12“ a soundtrack capable of boosting one‘s adrenaline was created, which fits nicely into the overall game feel and, using „Photoshop“, „Pixel“ style 2D objects had been drawn. Having combined all these things the end result is an interesting and fun, fast paced 2D fighting game.

INFORMACIJOS ŠALTINIŲ SĄRAŠAS

1. Unity variklio objektinio programavimo dokumentacija. Prieiga per internetą: <https://docs.unity3d.com/>
2. Mortal Kombat Warehouse. Prieiga per internetą: <http://www.mortalkombatwarehouse.com/> [žiūrėta 2018-05-29]
3. Stackoverflow. Prieiga per internetą: <https://stackoverflow.com/>
4. Muzikos žanro kūrimo pamoka. Prieiga per internetą: <https://www.youtube.com/watch?v=-r1Ou4btT6s> [žiūrėta 2018-05-29]
5. Informacija apie „One Finger Death Punch”. Prieiga per internetą: <https://en.wikipedia.org/wiki/One_Finger_Death_Punch> [žiūrėta 2018-05-29]
6. Brackeys YouTube kanalas. Prieiga per internetą: <https://www.youtube.com/user/Brackeys> [žiūrėta 2018-05-29]

PRIEDAI

***1 priedas. Žaidimo programiniai kodai***

**AudioScript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class AudioScript : MonoBehaviour {

public AudioClip[] hitClips;

public AudioClip[] killClips;

public AudioClip[] missClips;

public AudioClip[] uiButtonHoverClips;

public AudioClip[] uiButtonClickClips;

public AudioSource musicSource;

public AudioSource missSoundSource;

public AudioSource hitSoundSource;

public AudioSource killSoundSource;

public AudioSource UISource;

public Slider soundSlider;

public Slider musicSlider;

public static AudioScript instance;

void Awake() {

if (instance == null) {

instance = this;

soundSlider = GameObject.FindGameObjectWithTag ("SoundSlider").GetComponent<Slider> ();

musicSlider = GameObject.FindGameObjectWithTag ("MusicSlider").GetComponent<Slider> ();

} else {

Destroy (gameObject);

return;

}

DontDestroyOnLoad (gameObject);

soundSlider = GameObject.FindGameObjectWithTag ("SoundSlider").GetComponent<Slider> ();

musicSlider = GameObject.FindGameObjectWithTag ("MusicSlider").GetComponent<Slider> ();

}

// Use this for initialization

void Start () {

hitSoundSource.loop = false;

killSoundSource.loop = false;

missSoundSource.loop = false;

soundSlider = GameObject.FindGameObjectWithTag ("SoundSlider").GetComponent<Slider> ();

musicSlider = GameObject.FindGameObjectWithTag ("MusicSlider").GetComponent<Slider> ();

}

private AudioClip GetRandomHitClip(){

return hitClips [Random.Range (0, hitClips.Length)];

}

private AudioClip GetRandomKillClip(){

return killClips [Random.Range (0, killClips.Length)];

}

private AudioClip GetRandomMissClip(){

return missClips [Random.Range (0, missClips.Length)];

}

private AudioClip GetRandomHoverClip(){

return uiButtonHoverClips [Random.Range (0, uiButtonHoverClips.Length)];

}

private AudioClip GetRandomClickClip(){

return uiButtonClickClips[Random.Range(0, uiButtonClickClips.Length)];

}

public void PlayRandomHit(){

hitSoundSource.clip = GetRandomHitClip ();

hitSoundSource.Play ();

}

public void GetRandomKill(){

killSoundSource.clip = GetRandomKillClip ();

killSoundSource.Play ();;

}

public void PlayRandomMiss(){

missSoundSource.clip = GetRandomMissClip ();

missSoundSource.Play ();

}

public void musicVolume(){

musicSource.volume = musicSlider.value;

}

public void soundVolume(){

killSoundSource.volume = soundSlider.value;

missSoundSource.volume = soundSlider.value;

hitSoundSource.volume = soundSlider.value;

}

public void UIHoverSound(){

UISource.clip = GetRandomHoverClip ();

UISource.Play ();

}

public void UIClickSound(){

UISource.clip = GetRandomClickClip ();

UISource.Play();

}

}

**BackgroundTiling.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BackgroundTiling : MonoBehaviour {

public int offsetX = 2;

public bool hasARightBuddy = false;

public bool hasALeftBuddy = false;

public bool reverseScale = false;

private float spriteWidth = 0f;

private Camera cam;

private Transform myTransform;

void Awake() {

cam = Camera.main;

myTransform = transform;

}

// Use this for initialization

void Start () {

SpriteRenderer sRenderer = GetComponent<SpriteRenderer> ();

spriteWidth = sRenderer.sprite.bounds.size.x - 0.02f;

}

// Update is called once per frame

void Update () {

if (hasALeftBuddy == false || hasARightBuddy == false) {

float camHorizontalExtend = cam.orthographicSize \* Screen.width / Screen.height;

float edgeVisiblePositionRight = (myTransform.position.x + spriteWidth / 2 - camHorizontalExtend);

float edgeVisiblePositionLeft = (myTransform.position.x - spriteWidth /2 + camHorizontalExtend);

if (cam.transform.position.x >= edgeVisiblePositionRight - offsetX && hasARightBuddy == false) {

MakeNewBuddy (1);

hasARightBuddy = true;

} else if (cam.transform.position.x <= edgeVisiblePositionLeft + offsetX && hasALeftBuddy == false) {

MakeNewBuddy (-1);

hasALeftBuddy = true;

}

}

}

void MakeNewBuddy(int rightOrLeft) {

Vector3 newPosition = new Vector3 (myTransform.position.x + spriteWidth \* rightOrLeft, myTransform.position.y, myTransform.position.z);

Transform newBuddy = Instantiate (myTransform, newPosition, myTransform.rotation) as Transform;

if (reverseScale == true) {

newBuddy.localScale = new Vector3 (newBuddy.localScale.x \* -1, newBuddy.localScale.y, newBuddy.localScale.z);

}

newBuddy.parent = myTransform.parent;

if (rightOrLeft > 0) {

newBuddy.GetComponent<BackgroundTiling> ().hasALeftBuddy = true;

} else {

newBuddy.GetComponent<BackgroundTiling>().hasARightBuddy = true;

}

}

}

**EnemyController.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyController : MonoBehaviour {

private GameObject self;

private GameObject Player;

public float enemyMoveSpeed;

public int health;

public int enemyHP;

public float distance\_;

public float distanceEnemyInFront;

public GameObject EnemyInFrontTarget;

public GameObject[] healthBars;

public float attackDelayTimer = 1.0f;

public GameObject zona;

public int zoneCount;

public GameObject EnemyLeftZone;

public GameObject EnemyRightZone;

private bool moveorno;

public GameObject TARGET;

public GameObject CURRENTTARGET;

float timer;

public float waitingTime;

public Transform deathRelocator;

public string enemyColor;

private float Rightorleft = 1.0f;

public GameObject deathDoll;

[Header("Can Attack Player")]

public bool canAttack;

public GameObject HitRangeRight;

public GameObject HitRangeLeft;

EnemyZoneTrigger EZT\_l;

EnemyZoneTrigger EZT\_r;

EnemyZoneTrigger EZT;

PlayerAttackZoneTrigger triggerBool\_left;

PlayerAttackZoneTrigger triggerBool\_right;

PlayerController PlayerControllerCall;

PlayerStats playerStats;

Animator anim;

WaveCounterScript WCS;

GameSpeedScript GSS;

AudioScript audioScript;

void OnDisable()

{

enemyHP = health;

for (int i = 0; i < healthBars.Length; i++) {

healthBars [i].SetActive (true);

}

}

void Awake () {

Player = GameObject.FindGameObjectWithTag ("Player");

}

// Use this for initialization

void Start () {

HitRangeRight = GameObject.FindGameObjectWithTag("HitRangeRight");

HitRangeLeft = GameObject.FindGameObjectWithTag("HitRangeLeft");

triggerBool\_right = HitRangeRight.GetComponent<PlayerAttackZoneTrigger>();

triggerBool\_left = HitRangeLeft.GetComponent<PlayerAttackZoneTrigger>();

enemyMoveSpeed = Random.Range(3, 7);

audioScript = GameObject.FindGameObjectWithTag("AudioGM").GetComponent<AudioScript>();

enemyHP = health;

GameObject GMobj = GameObject.FindGameObjectWithTag ("GameController");

WCS = GMobj.GetComponent<WaveCounterScript> ();

GSS = GMobj.GetComponent<GameSpeedScript> ();

waitingTime = GSS.EnemyAttackTimer;

PlayerControllerCall = Player.GetComponent<PlayerController> ();

playerStats = Player.GetComponent<PlayerStats> ();

anim = GetComponent<Animator> ();

PlayerRangeDetector ();

EZT.enemyInFront = Player;

if (this.tag == "Enemy\_Left") {

Rightorleft = Rightorleft \* 1.0f;

SpriteRenderer sprR = GetComponent<SpriteRenderer> ();

sprR.flipX = true;

} else if (this.tag == "Enemy\_Right") {

SpriteRenderer sprR = GetComponent<SpriteRenderer> ();

sprR.flipX = false;

Rightorleft = Rightorleft \* -1.0f;

}

}

// Update is called once per frame

void Update () {

CURRENTTARGET = EZT.enemyInFront;

distance\_ = Vector3.Distance (transform.position, Player.transform.position);

distanceEnemyInFront = Vector3.Distance (transform.position, EZT.enemyInFront.transform.position);

CheckYAxis (); // tikrina ar enemy tam paciam Y kaip ir player

PlayerRangeDetector();

MoveToPlayer();

AttackCheck ();

StunAttackCheck (distance\_);

}

public void TakeDamage(int a) {

int i = enemyHP - 1;

healthBars [i].SetActive (false);

enemyHP = enemyHP - a;

timer = 0;

anim.SetBool("Attacking", false);

CancelInvoke("Attack");

if (enemyHP > 0)

{

audioScript.PlayRandomHit();

if (enemyColor == "Blue")

{

SwitchSides();

}

else if (enemyColor == "Yellow")

{

if (enemyHP <= 2)

{

SwitchSides();

}

}

else if (enemyColor == "Orange")

{

if (enemyHP == 3 || enemyHP == 1)

{

SwitchSides();

}

}

else if (enemyColor == "Purple")

{

SwitchSides();

}

} else if (enemyHP < 1)

{

audioScript.GetRandomKill();

Die();

}

}

public void Die() {

EZT.inRangeOfPlayer = false;

EnemyRightZone.SetActive(true);

EnemyRightZone.GetComponent<EnemyZoneTrigger>().inRangeOfPlayer = false;

EnemyRightZone.SetActive(false);

EnemyLeftZone.SetActive(true);

EnemyLeftZone.GetComponent<EnemyZoneTrigger>().inRangeOfPlayer = false;

EnemyLeftZone.SetActive(false);

if (gameObject.tag== "Enemy\_Left" && deathDoll.GetComponent<SpriteRenderer>().flipX == false) {

deathDoll.GetComponent<SpriteRenderer> ().flipX = true;

} else if(gameObject.tag== "Enemy\_Right" && deathDoll.GetComponent<SpriteRenderer>().flipX == true){

deathDoll.GetComponent<SpriteRenderer> ().flipX = false;

}

Instantiate (deathDoll, transform.position, transform.rotation);

if(gameObject.tag == "Enemy\_Left")

{

triggerBool\_left.count--;

} else if (gameObject.tag == "Enemy\_Right")

{

triggerBool\_right.count--;

}

gameObject.SetActive (false);

WCS.EnemyKilled ();

GSS.IncreaseGameSpeed ();

EZT.enemyInFront = Player;

waitingTime = GSS.EnemyAttackTimer;

}

public void PlayerRangeDetector() { //RANDA PAGAL KURI PLAYER RANGE FIELDA ZIURETI PRIESA

if (this.tag == "Enemy\_Right") {

EnemyRightZone.SetActive (false);

EnemyLeftZone.SetActive (true);

EZT\_l = EnemyLeftZone.GetComponent<EnemyZoneTrigger> ();

EZT = EZT\_l;

} else if (this.tag == "Enemy\_Left") {

EnemyLeftZone.SetActive (false);

EnemyRightZone.SetActive (true);

EZT\_r = EnemyRightZone.GetComponent<EnemyZoneTrigger> ();

EZT = EZT\_r;

}

}

public void AttackCheck(){

if (EZT.inRangeOfPlayer == true) {

canAttack = true;

timer += Time.deltaTime;

if (timer > waitingTime) {

Invoke ("Attack", waitingTime);

timer = 0;

anim.SetBool ("Attacking", false);

}

}

}

public void StunAttackCheck(float dist){

if (PlayerControllerCall.playerStunned == true && dist <= 5.0f) {

if (EZT.enemyInFront.activeInHierarchy == false || EZT.enemyInFront == null || EZT.enemyInFront == Player){

canAttack = true;

StunAttack ();

Debug.Log ("Attacked a stunned player");

}

}

}

public void MoveToPlayer(){

if (distance\_ > 0.3f)

{

if (EZT.inRangeOfPlayer == false)

{

anim.SetBool("Walking", true);

transform.position = Vector3.MoveTowards(transform.position, Player.transform.position, enemyMoveSpeed \* Time.deltaTime);

}

else if (EZT.inRangeOfPlayer == true)

{

anim.SetBool("Walking", false);

}

} else

{

transform.position = Vector3.MoveTowards(transform.position, Player.transform.position, enemyMoveSpeed \* -1 \* Time.deltaTime);

}

}

public void Attack() {

if(canAttack == true) {

anim.SetBool ("Attacking", true);

}

}

public void StunAttack() {

anim.SetBool ("Attacking", true);

}

public void CheckYAxis(){

if (transform.position.y != Player.transform.position.y) {

transform.position = new Vector3 (transform.position.y, Player.transform.position.y);

}

}

public void DamagePlayer()

{

playerStats.DamagePlayer (1);

PlayerControllerCall.DamagedAnimation ();

PlayerControllerCall.playerStunned = false;

anim.SetBool("Attacking", false);

timer = 0;

}

public void AttackAnimationEnded(){

anim.SetBool ("Attacking", false);

}

public void MoveToPlayerOnStunAttack(){

transform.Translate (new Vector3 ((distance\_ - 0.5f) \* Rightorleft, 0, 0), Space.Self);

}

public void SwitchSides(){

if (gameObject.tag == "Enemy\_Left") {

gameObject.tag = "Enemy\_Right";

float newPos = transform.position.x + distance\_ + 1.6f;

transform.position = new Vector3 (newPos, transform.position.y, transform.position.z);

} else if (this.gameObject.tag == "Enemy\_Right") {

gameObject.tag = "Enemy\_Left";

float newPos = transform.position.x - distance\_ - 1.6f;

transform.position = new Vector3 (newPos, transform.position.y, transform.position.z);

}

PlayerRangeDetector ();

if (this.tag == "Enemy\_Left") {

Rightorleft = Rightorleft \* 1.0f;

SpriteRenderer sprR = GetComponent<SpriteRenderer> ();

sprR.flipX = true;

} else if (this.tag == "Enemy\_Right") {

SpriteRenderer sprR = GetComponent<SpriteRenderer> ();

sprR.flipX = false;

Rightorleft = Rightorleft \* -1.0f;

}

timer = 0;

anim.SetBool("Attacking", false);

CancelInvoke("Attack");

}

}

**EnemyZoneTrigger.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyZoneTrigger : MonoBehaviour {

public bool moveBackOrNot;

public GameObject enemyInFront = null;

public bool inRangeOfPlayer = false;

public float startTime;

// Use this for initialization

void Awake() {

moveBackOrNot = false;

inRangeOfPlayer = false;

}

public void OnTriggerEnter(Collider col){

if (col.gameObject.tag == "Enemy\_Right" || col.gameObject.tag == "Enemy\_Left") {

//moveBackOrNot = true;

enemyInFront = col.gameObject;

}

if (col.gameObject.tag == "Player") {

inRangeOfPlayer = true;

startTime = Time.time;

}

}

public void OnTriggerExit(Collider col){

if (col.gameObject.tag == "Enemy\_Right" || col.gameObject.tag == "Enemy\_Left") {

moveBackOrNot = false;

}

if (col.gameObject.tag == "Player") {

inRangeOfPlayer = false;

startTime = 0.0f;

}

}

}

**GameSpeedScript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameSpeedScript : MonoBehaviour {

public float EnemyAttackTimer = 1.0f;

public float currentGameSpeed;

public float spawnSpeed = 4.0f;

// public float enemyMoveSpeed = 5.0f;

void Awake(){

currentGameSpeed = EnemyAttackTimer;

}

public void IncreaseGameSpeed(){

Time.timeScale = Time.timeScale + 0.0025f;

currentGameSpeed = Time.timeScale;

EnemyAttackTimer -= 0.0015f;

if (spawnSpeed > 3)

{

spawnSpeed -= 0.015f;

}

}

}

**MainMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour {

public void PlayGame(){

SceneManager.LoadScene (1);

}

public void QuitGame(){

Application.Quit ();

}

}

**PauseMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using TMPro;

public class PauseMenu : MonoBehaviour {

public static bool GameIsPaused = false;

public GameObject pauseMenuUI;

public GameObject optionsMenuUI;

public GameObject gameOverUI;

public static bool GameOverBool = false;

GameSpeedScript GSS;

PlayerController PC;

void Awake(){

GameIsPaused = false;

GameOverBool = false;

}

void Start(){

GSS = GameObject.FindGameObjectWithTag ("GameController").GetComponent<GameSpeedScript>();

PC = GameObject.FindGameObjectWithTag ("Player").GetComponent<PlayerController> ();

}

public void MainMenu(){

GameOverBool = false;

Time.timeScale = 1.0f;

SceneManager.LoadScene (0);

PC.enabled = true;

}

public void QuitGame(){

Application.Quit ();

}

public void Restart(){

SceneManager.LoadScene (1);

Time.timeScale = 1.0f;

GameOverBool = false;

PC.enabled = true;

}

public void Resume(){

pauseMenuUI.SetActive (false);

optionsMenuUI.SetActive (false);

Time.timeScale = GSS.currentGameSpeed;

GameIsPaused = false;

PC.enabled = true;

}

void Update(){

if (!GameOverBool) {

if (Input.GetKeyDown (KeyCode.Escape)) {

if (GameIsPaused) {

Resume ();

} else {

Pause ();

}

}

if (Input.GetKeyDown(KeyCode.P))

{

if (GameIsPaused)

{

Resume();

} else

{

Freeze();

}

}

}

}

void Freeze()

{

Time.timeScale = 0f;

GameIsPaused = true;

PC.enabled = false;

}

void Pause(){

pauseMenuUI.SetActive (true);

optionsMenuUI.SetActive (false);

Time.timeScale = 0f;

GameIsPaused = true;

PC.enabled = false;

}

public void GameOver(){

gameOverUI.SetActive (true);

optionsMenuUI.SetActive (false);

pauseMenuUI.SetActive (false);

Time.timeScale = 0f;

GameOverBool = true;

PC.enabled = false;

gameOverUI.transform.GetChild(2).GetComponent<TextMeshProUGUI>().text = "Game Over";

}

public void LevelWon()

{

gameOverUI.SetActive(true);

optionsMenuUI.SetActive(false);

pauseMenuUI.SetActive(false);

Time.timeScale = 0f;

GameOverBool = true;

PC.enabled = false;

gameOverUI.transform.GetChild(2).GetComponent<TextMeshProUGUI>().text = "You Won";

}

}

**PlayerAttackZoneTrigger.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerAttackZoneTrigger : MonoBehaviour {

public Time triggerZoneTime;

public int count;

public GameObject[] RangeIndicators;

public void Start() {

RangeIndicators [0].SetActive (true);

RangeIndicators [1].SetActive (false);

count = 0;

}

public void Update() {

if (count >= 1) {

RangeIndicators [0].SetActive (false);

RangeIndicators [1].SetActive (true);

} else {

RangeIndicators [0].SetActive (true);

RangeIndicators [1].SetActive (false);

}

if (count < 0)

{

count = 0;

}

}

public void OnTriggerEnter (Collider col){

if (col.gameObject.tag == "Enemy\_Right") {

if (gameObject.tag == "HitRangeRight")

{

count++;

}

}

if(col.gameObject.tag == "Enemy\_Left")

{

if(gameObject.tag == "HitRangeLeft")

{

count++;

}

}

}

public void OnTriggerExit(Collider col){

if (col.gameObject.tag == "Enemy\_Right" || col.gameObject.tag == "Enemy\_Left") {

count--;

}

}

}

**PlayerController.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerController : MonoBehaviour {

//public bool EnemyInRange = false;

private Transform player;

public GameObject playerCamera;

//private float speed = 50f;

public float missMoveAmount = 5.0f;

private float distance\_;

public bool playerStunned = false;

public ParticleSystem[] PunchEffect;

public Transform[] attackeffectloc;

private int attackAnimation;

public Vector3 right = new Vector3 (1f, 0, 0);

public Vector3 left = new Vector3 (-1f, 0, 0);

private bool nextAttackQueue = false;

private int currentEnemy;

public int attackDamage = 1;

[Header("AttackZoneTriggers")]

public GameObject HitRangeRight;

public GameObject HitRangeLeft;

public GameObject HitRangeScaleLeft;

public GameObject HitRangeScaleRight;

//var relativePoint;

PlayerAttackZoneTrigger triggerBool\_Right;

PlayerAttackZoneTrigger triggerBool\_Left;

Animator anim;

SpriteRenderer spr;

CameraShake camShake;

WaveCounterScript WCS;

AudioScript audioScript;

PlayerStats playerStats;

//EnemyController enemy;

// Use this for initialization

void Start () {

audioScript = GameObject.FindGameObjectWithTag ("AudioGM").GetComponent<AudioScript> ();

//destination = transform.position;

camShake = playerCamera.GetComponent<CameraShake> ();

WCS = GameObject.FindGameObjectWithTag ("GameController").GetComponent<WaveCounterScript> ();

anim = GetComponent<Animator> ();

spr = this.GetComponent<SpriteRenderer> ();

triggerBool\_Right = HitRangeRight.GetComponent<PlayerAttackZoneTrigger> ();

triggerBool\_Left = HitRangeLeft.GetComponent<PlayerAttackZoneTrigger> ();

InvokeRepeating ("SpawnTrailPart", 0, 0.05f);

playerStats = gameObject.GetComponent<PlayerStats>();

}

// Update is called once per frame

void Update () {

if (Input.GetMouseButtonDown (1)) {

if (playerStunned == false) {

spr.flipX = false;

attackAnimation = Random.Range (1, 6);

anim.SetInteger ("State", attackAnimation);

if (triggerBool\_Right.count > 0) {

Attack (1f);

} else {

Miss (1f);

}

}

}

if (Input.GetMouseButtonUp (1)) {

anim.SetInteger ("State", 0);

}

if (Input.GetMouseButtonDown (0)) {

if (playerStunned == false) {

spr.flipX = true;

attackAnimation = Random.Range (1, 6);

anim.SetInteger ("State", attackAnimation);

if (triggerBool\_Left.count > 0) {

if (playerStunned == false) {

Attack (-1f);

}

} else {

Miss (-1f);

}

}

}

if (Input.GetMouseButtonUp (0)) {

anim.SetInteger ("State", 0);

}

}

public void Attack(float leftOrRight) {

if (leftOrRight == 1) {

distance\_ = Vector3.Distance (transform.position, FindClosestEnemy\_Right ().transform.position);

FindClosestEnemy\_Right().GetComponent<EnemyController>().TakeDamage(attackDamage);

StartCoroutine(MoveToTarget(distance\_, leftOrRight));

} else if (leftOrRight == -1) {

distance\_ = Vector3.Distance (transform.position, FindClosestEnemy\_Left ().transform.position);

FindClosestEnemy\_Left().GetComponent<EnemyController>().TakeDamage(attackDamage);

StartCoroutine(MoveToTarget(distance\_, leftOrRight));

}

WCS.UpdateCombo (true);

Skill\_Heal();

Skill\_PowerUp();

}

public void Miss(float leftOrRight){

StartCoroutine(MissMove (missMoveAmount, leftOrRight));

StartCoroutine(StunTimer());

audioScript.PlayRandomMiss ();

WCS.UpdateCombo (false);

}

public GameObject FindClosestEnemy\_Right() {

GameObject[] gos;

gos = GameObject.FindGameObjectsWithTag("Enemy\_Right");

GameObject closest = null;

float distance = Mathf.Infinity;

Vector3 position = transform.position;

foreach (GameObject go in gos)

{

Vector3 diff = go.transform.position - position;

float curDistance = diff.sqrMagnitude;

if (curDistance < distance)

{

closest = go;

distance = curDistance;

}

}

return closest;

}

public GameObject FindClosestEnemy\_Left() {

GameObject[] gos;

gos = GameObject.FindGameObjectsWithTag("Enemy\_Left");

GameObject closest = null;

float distance = Mathf.Infinity;

Vector3 position = transform.position;

foreach (GameObject go in gos)

{

Vector3 diff = go.transform.position - position;

float curDistance = diff.sqrMagnitude;

if (curDistance < distance)

{

closest = go;

distance = curDistance;

}

}

return closest;

}

//PIRMA PADAROMA ZALA POTO ZAIDEJAS JUDA

public void KillEnemy\_Right(){

var enemyC = FindClosestEnemy\_Right().GetComponent<EnemyController>();

if (FindClosestEnemy\_Right ().GetComponent<EnemyController> ().enemyHP < 1) {

FindClosestEnemy\_Right ().GetComponent<EnemyController> ().Die ();

triggerBool\_Right.count--;

} else {

}

}

public void KillEnemy\_Left() {

var enemyC = FindClosestEnemy\_Left().GetComponent<EnemyController>();

if (FindClosestEnemy\_Left ().GetComponent<EnemyController> ().enemyHP < 1) {

FindClosestEnemy\_Left ().GetComponent<EnemyController> ().Die ();

triggerBool\_Left.count--;

} else {

}

}

public IEnumerator StunTimer(){

playerStunned = true;

//Debug.Log ("Stunned");

anim.SetInteger ("State", 7);

yield return new WaitForSeconds (1.0f);

playerStunned = false;

anim.SetInteger ("State", 0);

//Debug.Log ("Stun ended");

}

public void SpawnTrailPart()

{

GameObject trailPart = new GameObject ();

SpriteRenderer trailPartRenderer = trailPart.AddComponent<SpriteRenderer> ();

trailPartRenderer.sprite = GetComponent<SpriteRenderer> ().sprite;

trailPartRenderer.sortingOrder = 7;

trailPart.transform.position = transform.position;

trailPartRenderer.color = new Color (255, 0, 216);

if (spr.flipX == true) {

trailPartRenderer.flipX = true;

} else if (spr.flipX != true) {

trailPartRenderer.flipX = false;

}

StartCoroutine (FadeTrailPart (trailPartRenderer));

Destroy (trailPart, 0.5f);

}

public void DamagedAnimation(){

GetComponent<SpriteRenderer> ().color = Color.red;

StartCoroutine ("AfterHit");

}

IEnumerator FadeTrailPart(SpriteRenderer trailPartRenderer)

{

Color color = trailPartRenderer.color;

color.a -= 0.8f;

trailPartRenderer.color = color;

yield return new WaitForEndOfFrame ();

}

IEnumerator MoveToTarget(float distance, float leftright) {

float startTIme = Time.time;

while (Time.time < startTIme + 0.17f) {

if (distance > 1.5f)

{

transform.Translate(new Vector3((distance - 1.5f) \* leftright \* Time.deltaTime \* 4.5f, 0, 0), Space.Self);

}

else break;

yield return null;

}

if (leftright > 0) {

Instantiate (PunchEffect [Random.Range (0, 2)], attackeffectloc [0].position, attackeffectloc [0].rotation);

Instantiate (PunchEffect [2], attackeffectloc [0].position, attackeffectloc [0].rotation);

} else if (leftright < 0) {

Instantiate (PunchEffect[Random.Range(0,2)], attackeffectloc[1].position, attackeffectloc[1].rotation);

Instantiate (PunchEffect[2], attackeffectloc[1].position, attackeffectloc[1].rotation);

}

}

IEnumerator MissMove(float missMoveDistance, float leftright) {

float startTIme = Time.time;

while (Time.time < startTIme + 0.155f) {

transform.Translate (new Vector3 (missMoveDistance \* leftright \* Time.deltaTime \* 4.7f, 0, 0), Space.World);

yield return null;

}

}

IEnumerator AfterHit(){

yield return new WaitForSeconds (0.3f);

GetComponent<SpriteRenderer> ().color = Color.white;

}

public void Skill\_Heal()

{

if(WCS.combo % 50 == 0)

{

playerStats.HealPlayer(1);

}

}

public void Skill\_PowerUp()

{

if(WCS.combo % 30 == 0)

{

HitRangeScaleLeft.transform.localScale = new Vector3(2, 1, 1);

HitRangeScaleRight.transform.localScale = new Vector3(2, 1, 1);

nextAttackQueue = true;

currentEnemy = WCS.enemiesLeft;

attackDamage = 1;

}

if (nextAttackQueue)

{

if(currentEnemy - WCS.enemiesLeft > 5)

{

nextAttackQueue = false;

HitRangeScaleLeft.transform.localScale = new Vector3(1, 1, 1);

HitRangeScaleRight.transform.localScale = new Vector3(1, 1, 1);

attackDamage = 1;

}

}

}

}

**PlayerStats.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerStats : MonoBehaviour {

public GameObject[] hpBars;

PauseMenu PM;

public class PlayerHealth{

public const int maxHP = 10;

public int curHP = maxHP;

public Animator hpAnim;

}

public PlayerHealth playerHealth = new PlayerHealth();

public void Awake() {

}

void Start(){

PM = GameObject.FindGameObjectWithTag ("PauseMenu").GetComponent<PauseMenu> ();

}

public void DamagePlayer(int damageTaken){

playerHealth.curHP -= damageTaken;

int i = playerHealth.curHP;

playerHealth.hpAnim = hpBars [i].GetComponent<Animator> ();

playerHealth.hpAnim.SetBool ("Active", false);

if(playerHealth.curHP <= 0){

playerHealth.curHP = 0;

Debug.Log ("Player Died");

PM.GameOver ();

}

}

public void HealPlayer(int healAmount)

{

if (playerHealth.curHP < 10)

{

playerHealth.curHP += healAmount;

int i = playerHealth.curHP -1;

playerHealth.hpAnim = hpBars[i].GetComponent<Animator>();

playerHealth.hpAnim.SetBool("Active", true);

}

}

}

**SpawnEnemy.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SpawnEnemy : MonoBehaviour

{

public int randomMin = 1;

public int randomMax = 100;

public GameObject GreyEnemy;

public GameObject BlueEnemy;

public GameObject GreenEnemy;

public GameObject RedEnemy, YellowEnemy, OrangeEnemy;

private float spawnTimer;

public Transform spawnPoint;

public float spawnMin = 0.5f;

public float spawnMax = 4.0f;

bool isSpawning = false;

public string enemyTag;

public int pooledGreyEnemy = 10;

public int pooledBlueEnemy = 10;

public int pooledGreenEnemy = 10;

public int pooledRedEnemy = 10;

public int pooledYellowEnemy, pooledOrangeEnemy = 10;

List<GameObject> BlueEnemies;

List<GameObject> GreyEnemies;

List<GameObject> GreenEnemies;

List<GameObject> RedEnemies;

List<GameObject> YellowEnemies;

List<GameObject> OrangeEnemies;

List<GameObject> RandomEnemies;

// Use this for initialization

void Start()

{

GreyEnemies = new List<GameObject>();

for (int i = 0; i < pooledGreyEnemy; i++)

{

GameObject obj = GameObject.Instantiate(GreyEnemy);

obj.SetActive(false);

GreyEnemies.Add(obj);

}

BlueEnemies = new List<GameObject>();

for (int i = 0; i < pooledBlueEnemy; i++)

{

GameObject obj = GameObject.Instantiate(BlueEnemy);

obj.SetActive(false);

BlueEnemies.Add(obj);

}

GreenEnemies = new List<GameObject>();

for (int i = 0; i < pooledGreenEnemy; i++)

{

GameObject obj = GameObject.Instantiate(GreenEnemy);

obj.SetActive(false);

GreenEnemies.Add(obj);

}

RedEnemies = new List<GameObject>();

for (int i = 0; i < pooledRedEnemy; i++)

{

GameObject obj = GameObject.Instantiate(RedEnemy);

obj.SetActive(false);

RedEnemies.Add(obj);

}

YellowEnemies = new List<GameObject>();

for (int i = 0; i < pooledYellowEnemy; i++)

{

GameObject obj = GameObject.Instantiate(YellowEnemy);

obj.SetActive(false);

YellowEnemies.Add(obj);

}

OrangeEnemies = new List<GameObject>();

for (int i = 0; i < pooledOrangeEnemy; i++)

{

GameObject obj = GameObject.Instantiate(OrangeEnemy);

obj.SetActive(false);

OrangeEnemies.Add(obj);

}

}

// Update is called once per frame

void Update()

{

if (!isSpawning)

{

isSpawning = true;

spawnMax = gameObject.GetComponent<GameSpeedScript>().spawnSpeed;

StartCoroutine(Spawn(Random.Range(spawnMin, spawnMax)));

}

}

IEnumerator Spawn(float seconds)

{

yield return new WaitForSeconds(seconds);

int rand = Random.Range(randomMin, randomMax);

if (rand < 25 || rand >= 85)

{

for (int i = 0; i < GreyEnemies.Count; i++)

{

if (!GreyEnemies[i].activeInHierarchy)

{

GreyEnemies[i].transform.position = spawnPoint.transform.position;

GreyEnemies[i].tag = enemyTag;

GreyEnemies[i].SetActive(true);

SetTime();

break;

}

}

}

else if (rand >= 25 && rand < 40)

{

//Debug.Log ("Spawn blue");

for (int i = 0; i < BlueEnemies.Count; i++)

{

if (!BlueEnemies[i].activeInHierarchy)

{

BlueEnemies[i].transform.position = spawnPoint.transform.position;

BlueEnemies[i].tag = enemyTag;

if (enemyTag == "Enemy\_Right")

{

BlueEnemies[i].GetComponent<SpriteRenderer>().flipX = false;

}

else if (enemyTag == "Enemy\_Left")

{

BlueEnemies[i].GetComponent<SpriteRenderer>().flipX = true;

}

BlueEnemies[i].SetActive(true);

SetTime();

break;

}

}

}

else if (rand >= 40 && rand < 55)

{

for (int i = 0; i < GreenEnemies.Count; i++)

{

if (!GreenEnemies[i].activeInHierarchy)

{

GreenEnemies[i].transform.position = spawnPoint.transform.position;

GreenEnemies[i].tag = enemyTag;

if (enemyTag == "Enemy\_Right")

{

GreenEnemies[i].GetComponent<SpriteRenderer>().flipX = false;

}

else if (enemyTag == "Enemy\_Left")

{

GreenEnemies[i].GetComponent<SpriteRenderer>().flipX = true;

}

GreenEnemies[i].SetActive(true);

SetTime();

break;

}

}

}

else if (rand >= 55 && rand < 65)

{

for (int i = 0; i < RedEnemies.Count; i++)

{

if (!RedEnemies[i].activeInHierarchy)

{

RedEnemies[i].transform.position = spawnPoint.transform.position;

RedEnemies[i].tag = enemyTag;

if (enemyTag == "Enemy\_Right")

{

RedEnemies[i].GetComponent<SpriteRenderer>().flipX = false;

}

else if (enemyTag == "Enemy\_Left")

{

RedEnemies[i].GetComponent<SpriteRenderer>().flipX = true;

}

RedEnemies[i].SetActive(true);

SetTime();

break;

}

}

}

else if (rand >= 65 && rand < 75)

{

for (int i = 0; i < YellowEnemies.Count; i++)

{

if (!YellowEnemies[i].activeInHierarchy)

{

YellowEnemies[i].transform.position = spawnPoint.transform.position;

YellowEnemies[i].tag = enemyTag;

if (enemyTag == "Enemy\_Right")

{

YellowEnemies[i].GetComponent<SpriteRenderer>().flipX = false;

}

else if (enemyTag == "Enemy\_Left")

{

YellowEnemies[i].GetComponent<SpriteRenderer>().flipX = true;

}

YellowEnemies[i].SetActive(true);

SetTime();

break;

}

}

}

else if (rand >= 75 && rand < 85)

{

for (int i = 0; i < OrangeEnemies.Count; i++)

{

if (!OrangeEnemies[i].activeInHierarchy)

{

OrangeEnemies[i].transform.position = spawnPoint.transform.position;

OrangeEnemies[i].tag = enemyTag;

if (enemyTag == "Enemy\_Right")

{

OrangeEnemies[i].GetComponent<SpriteRenderer>().flipX = false;

}

else if (enemyTag == "Enemy\_Left")

{

OrangeEnemies[i].GetComponent<SpriteRenderer>().flipX = true;

}

OrangeEnemies[i].SetActive(true);

SetTime();

break;

}

}

}

isSpawning = false;

}

void SetTime()

{

spawnTimer = Random.Range(spawnMin, spawnMax);

}

}

**WaveCounterScript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class WaveCounterScript : MonoBehaviour {

public int enemiesLeft = 150;

public TextMeshProUGUI enemiesLeftText;

public TextMeshProUGUI ComboText;

public int combo = 0;

PauseMenu PM;

void Start () {

enemiesLeftText = GameObject.FindGameObjectWithTag("EnemiesLeftUI").GetComponent<TextMeshProUGUI> ();

ComboText = GameObject.FindGameObjectWithTag("ComboCounter").GetComponent<TextMeshProUGUI> ();

PM = GameObject.FindGameObjectWithTag ("PauseMenu").GetComponent<PauseMenu> ();

UpdateEnemies ();

UpdateCombo (false);

}

public void EnemyKilled() {

enemiesLeft -= 1;

UpdateEnemies ();

if (enemiesLeft < 0) {

PM.LevelWon ();

}

}

public void UpdateEnemies() {

enemiesLeftText.text = enemiesLeft.ToString ();

}

public void UpdateCombo(bool Hit){

if (Hit) {

combo += 1;

} else {

combo = 0;

}

if (combo > 0) {

ComboText.text = "x" + combo.ToString ();

} else {

ComboText.text = (" ");

}

}

}

**ZombieDeathAnimationPicker.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ZombieDeathAnimationPicker : MonoBehaviour {

Animator anim;

// Use this for initialization

void Start () {

anim = GetComponent<Animator> ();

anim.SetInteger("DeathClip", Random.Range(1, 3));

}

}