ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ДИГИТАЛНИ НАУКИ

„СОФТУНИ СВЕТЛИНА“

# Дипломна работа

ЗА ПРИДОБИВАНЕ НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ

НА ТЕМА

**СЪЗДАВАНЕ НА ВИДЕОИГРА С „UNITY“**

ИЗГОТВИЛ: **КРИСТИЯН ДИМИТРОВ**

НАУЧЕН РЪКОВОДИТЕЛ: **ЕКАТЕРИНА МИЦЕВА**

Дата: **03.05.2023 г.**

ГР. СОФИЯ

# Съдържание

[Дипломна работа 1](#_Toc134696941)

[Съдържание 2](#_Toc134696942)

[Увод 3](#_Toc134696943)

[Проблем 3](#_Toc134696944)

[Цели на дипломния проект 4](#_Toc134696945)

[Глава 1. Проучване (смени заглавието) 5](#_Toc134696946)

[Глава 2. Проектиране и имплементация 10](#_Toc134696947)

[Играч 10](#_Toc134696948)

[Движение 10](#_Toc134696949)

[Живот 11](#_Toc134696950)

[Стрелба 12](#_Toc134696951)

[Врагове 17](#_Toc134696952)

[Живот на враговете 17](#_Toc134696953)

[Flying Enemy 18](#_Toc134696954)

[Turret Enemy 18](#_Toc134696955)

[Spawner Enemy 18](#_Toc134696956)

[Генерация на рундове / Магазин 19](#_Toc134696957)

[Интерфейс 20](#_Toc134696958)

[Версии 21](#_Toc134696959)

[Глава 3. Ръководство за потребителя 22](#_Toc134696960)

[Започване на играта 22](#_Toc134696961)

[Контроли 22](#_Toc134696962)

[Глава 4. Бъдещи планове 23](#_Toc134696963)

[Информационни източници 24](#_Toc134696964)

[Рецензия на дипломен проект 25](#_Toc134696965)

# Увод

Настоящият проект има за цел да създаде видео игра, която смесва игрови механики от различни жанрове заедно с една главна уникална идея. Фокусът е поставен върху това всяко преиграване да е уникално спрямо всяко минало с помощта на контролирана случайност и предпоставката за по – бързо преиграване. Вместо краен резултат, една игра може да е безкрайна, позволявайки на играча избор, а може и да е с цел най – високи точки до даден момент, което поставя пред играча сложни избори.

Името на проекта е „CyberGun“, което директно сочи към два основни елемента – стилистиката, направена за да наподобява модернистичния и уникален стил „Cyberpunk“, и главната уловка на играта – оръжието.

Действието се развива от първо лице, като камерата никога не напуска перспективата на играча, което създава познато усещане за повечето играчи, върху което може да се добавя.

Оригинално за тази игра е начина, по който функционира стрелбата. В повечето игри играчът е прикован към това да използва предварително създадени оръжия, докато тук се отваря възможност с развитието на действието да се получават постоянни уникални комбинации.

В малко по – дългосрочен план, играта се развива в рундове, който започват постепенно да стават по –трудни, докато играча не загуби. Тогава той трябва да започне от начало, с цел да надмине сам себе си.

## Проблем

Проблемът е да се създаде уникално и развлекателно преживяване под формата на видео игра. Като продукт главната му публика се намира във фенове на жанровете, които се намират в основата на играта: стрелбата от първо лице и “Roguelike” модела. Те споделят популярност през последните няколко години, което прави този проект тенденциозен, но с помощта на уникалната централна механика – създаването на уникално оръжие.

Друг допълнителен елемент с този жанр игри е тяхната популярност онлайн. Много от тези игри получават своята популярност със стриймъри, който показват играта на много от света (как променяме играта за да имаме предвид това ще бъде споменато по - късно)

## Цели на дипломния проект

Главните цели за тази разработка са накрая да има функционално завършен проект (в игровата индустрия това обикновено се нарича proof of concept). Това се изразява в следните елементи:

* Работещо движение на играча, което включва:
  + Стандартно ходене и скок.
  + Dash (моментно забързване в посоката на движение, позволяващо избягване на врагове в дадени ситуации).
  + Ground Slam (Забързано забиване в земята, което отново намира своята потребност когато играча трябва да се движи по – бързо).
* Развита система за стреляне:
  + Стрелба с два различни вида изстрел.
  + Работеща система за сменяне на части от оръжието.
  + Прилагането на промените от частите.
* Различни врагове:
  + Летящ враг - следва играча и го наранява при контакт.
  + Spawner - статичен враг, който създава снаряди, които изчезват след като играча бъде ударен.
  + Turret – друг статичен враг, който постепенно се зарежда докато играча е около него. Когато напълно се зареди наранява играча и почва да се зарежда отново.
* Система за процедурно генериране на рундове
* Магазин, който позволява на играча да сменя части от оръжието си

# Глава 1. Проучване (смени заглавието)

Проучването за този проект започна с два въпроса: „Каква ще е играта?“ и „Какъв размер трябва да е този проект?“.

Първият въпрос определи това кои други игри взимам за основа жанрово: в този случай смесицата от стрелба от първо лице и Roguelike жанра.



Игрите тип стрелба от първо лице са популярни от дълго време. Още от времето на Doom (снимка 1) са един от най – популярните стилове игри, и с право. Перспективата е натурална за нас като хора, защото е тази в която живеем животите си. От 90-те насам жанра е еволюирал по много начини: добавянето на различни типове оръжия, игра между много играчи и по – сложно движение заедно постепенно водят до формулирането на модерната идея за този жанр.

Снимка

Един специфичен поджанр, който пасна на останалите идеи е Arena Shooter. Базиран на стилистиката на Quake (снимка 2) и Unreal Tournament, този жанр е базиран на смесица от висока скорост и намален реализъм. От там можем да вземем забързаното движение и логиката зад оръжията. Важен избор във всички тези игри е как работят оръжията. Има два основни вида: HitScan и Projectile.

Снимка

HitScan е избора за повечето модерни игри, като CS: GO, Call of Duty и тн. Работи на основа създаването на лъчи, които симулират движението на куршум, без да се появява реално такъв обект. Голям авантаж на този стил е скоростта, тъй като не се изисква създаването на истински обекти или на каквато и да форма физика. Но тук и е голям минус, тъй като се губи реалния ефект на движещи се куршуми през бойното поле и се губи взаимодействието с физиката, което някои по – реалистични игри предпочитат.

Projectile е избора на по – малко игри, въпреки че е доказано да може да работи добре. Често срещани примери за такива игри са Battlefield и Arma, който прекрасно демонстрират как този тип стрелба може да се използва за добавен реализъм, въпреки по – голямото натоварване над машините на потребителите си.

Знаейки това, идва въпросът за това какво да избера за този проект. В този случай реших да се обърна към по – старите игри от типа Arena Shooter за отговор. В тях има много примери за оръжия и от двата типа, съответно стигаме до една от първите механики в нашия проект – главното оръжие, което се използва, може да се сменя между двата вида, спрямо това кое играча предпочита в дадена ситуация.

Тук е редно да спомена една от главните игри, която беше определяща за някои идеи в този проект: Ultrakill (снимка 3). Играта е разработена от малък екип и се базира на много от основните идеи на Arena Shooter поджанра - бързо движение, ретро стилистика и характерно нелепи оръжия с уникални едно от друго функции. 

Снимка

От там за този проект взех някои части от движението: главно Dash и Ground Smash. За точните им имплементации в този проект ще спомена по – късно, но важно за сега е че изпълняват помагаща функция в движението.

Следващата ключова част е Roguelike елемента.

Roguelike взима името си от играта Rogue (снимка 4) – игра от 1980, чието значение в този случай идва от една главна част: процедурната генерация на нива. Постепенно тази идея се пренася с годините в други игри, като през 2011 намира дом с The Binding of Isaac – игра създадена с от Едмунд Макмилън – американски създател на игри. Тази игра показва отново как постоянната генерация на нови нива и врагове може да доведе до игра, която безкрайно да се преиграва, без да губи вниманието на играчите си. Тя го постига през смесица от случайно създавани нива, отделни предмети, които заедно могат да водят до уникални комбинации и тн.



Снимка

От тук се роди идеята за това играта да е безкрайна чрез генерирането на случайни предмети, които могат да работят заедно, както и идеята за това колкото се може повече от играта да се създава случайно, за да се създаде чувство за безкрайна повторимост.

Друго място, което може да се посочи като основа за тази игра, са аркадните игри. За много от тях структурата се върти около следния цикъл – играча плаща за един живот, играчът стига колкото може по – далеч с този живот, като по пътя събира точки, накрая играча се състезава с други играчи за най – висок резултат.

Ако се обосновем на идеята играча при смърт да се връща в началото директно можем да видим как се случва връзката между тези две идеи – аркадните игри от рано имат идеята за един живот, на който виси цялата игра, докато Roguelike жанра показва как можем между тези животи да създаваме по – дълбока динамика чрез случайна генерация.

Сега, имайки основните идеи за проекта, можем да се фокусираме на въпрос две: „Какъв размер трябва да е този проект?“

Първоначално този въпрос изглежда странен и заради това е важно да разберем защо го задаваме. Ключова част от разработката на всяка програма е изборът на инструменти, и това е особено важно за игрите. Така нареченият Game Engine (който от тук на сетне ще наричам игрови двигател) е основата на всички механики в една игра на базово ниво: физика, осветление, функционалности и тн.

За този проект проучването ме доведе до изборът на Unity като игрови двигател. Няколко ключови елемента за този избор бяха следните:

* Достатъчно качествено 3D за този проект
* Поддръжката на физика
* Голямото количество вече съществуващи инструкции за това как се използват всички функционалности
* Наличността на качествени ефекти, които да добавят към вида на играта, без да изискват много работа
* Предлага добър баланс между качество на графиките и тежест върху системата
* Миналият ми личен с програмата за минали проекти
* Поддръжката на C# , програмен език, на който лично съм свикнал
* Не на последно място по значение – безплатният достъп за създаването на такива проекти

Поддръжката на C# тук е важната, тъй като заедно с нея ще се изградят някои от по – важните елементи в структурата на кода по – нататък, като много от тези функционалности иначе биха били по – сложни или по – трудно разбираеми. С помощта на принципите на обектно ориентираното програмиране много от работата по – нататък става по проста, особено когато се добави това че всички предмети в Unity съществуват и като лесно достъпни елементи в кода, които можем да манипулираме като типични обекти за C#.

В проучването си намерих и други сходни програми, и хипотетично всяка една от тях би могла да свърши работата, но всяка една отпадна поради една или друга причина:

* Unreal Engine – Популярният избор за много от най – големите компании в момента, главно поради умението си да работи с всички популярни платформи. За този проект не ми се видя като подходящ избор поради смесица от излишната тежест и това че не съм запозната с езика на двигателя (C++).
* CRYENGINE – Друг пример за програма, използвана от големи компании. Много от същите забележки към Unreal се прилагат и тук, особено когато се добави липсата на някои функционалности, които не могат да се добавят безплатно.
* GameMaker Studio – В другия край на създаването на игри, GMS е много популярна програма за хора които те първа започват или желаят да работят с по – проста система. Главният проблем тук беше твърде зле представящото се 3D, особено когато се включи с това, че изисква такса за да се извади проект за нормална платформа.
* Source – Двигател създаден от Valve преди доста години, този двигател е още използван в различни игри от някои студия (скорошен пример е Apex Legends, която работи с модификация на двигателя). Тук се показва и главния проблем – смесицата от възраст с нуждата за модификации чисто логически водят до това този труд да е излишен, спрямо това което е нужно да се постигне.
* Godot – Единият от малкото двигатели с отворен лиценз, Godot бавно се развива през годините, от версия на версия. Въпреки че те първа набира популярност, смесицата от уникален език и някои нови идеи го правят интересен за нови потребители. За този проект само че отново липсата на качествено 3D (засега) е достатъчно сериозен проблем за да не желая да го използвам в този случай.

С Unity като финален избор, вече можем да разглеждаме какво друго може да е нужно.

Най – важното нещо право след това е програма за създаване на 3D модели. В този случай използвах Blender за създаването на прости модели, които имат за цел да покажат функционалностите на играта. Тук няма да навлизам в излишен детайл, но отново смесицата от ниска цена и достъп до много учебни материали го направи лесен избор.

С това, проучването пое завои към по – конкретни примери, с които да мога постепенно да се ориентирам към началото на проекта (най – важните от тях ще са поставени в информационни източници)

В този момент това е края на проучването, и от тук може да започне разработката на проекта.

# Глава 2. Проектиране и имплементация

Проектирането на този проект беше направено с цел колкото се може повече компоненти да са модулирани и да позволяват за по – нататъшно развитие с колкото се може по – малко нужда за промяна на вече създаван код.

Основните секции се делят спрямо най – важните си елементи:

* Играч, където са описани уменията на играча, както и функционалността на срелбата в тази игра
* Врагове, където са описани всички отделни компоненти, които може да има един враг, и това как всеки един вече съществуващ враг ги използва.
* Генерация на рундове / Магазин, където са описани как се събират всички тези системи и това как се създават новите рундове, както и магазина, от който играча може да подобрява арсенала си.
* Внедряването, в което е описано как се изпълнява проекта не една машина.

Вътре в тези редове код също съществуват и обясненията за това кое, къде и как се изобразява, освен главното меню, което е достатъчно стандартно, че не изисква излишно внимание.

## Играч

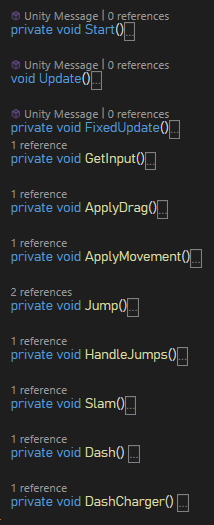
Играчът е най – ключовата част в почти всички игри, от най – простите до най – сложните проекти, и съответно не малко от сложните компоненти са там.

За яснота тази секция се дели на няколко компонента – Движение, Живот, Стрелба и Интерфейс.

### Движение

Движението на играча се съдържа изцяло в един script – PlayerMovement.cs

В началото се инициализират много променливи, но много от тях стават важни само на няколко места, тъй че ще бъдат назовани когато стават важни. Повечето код е структуриран в методи, които се викат когато е нужно в void Start, void Update, void FixedUpdate, които са функции, които автоматично се изпълняват от двигателя. Надолу са изредени, заедно с това какво изпълняват:



* GetInput() - задава посоката ни на движение, като взима посоките, в които е натиснал играчът.
* ApplyDrag() – има за цел просто да заложи триенето на rigidbody-то, закачено за играчът.
* ApplyMovement() – прилага сила върху rigidbody-то на играчът, като променя това колко сила се прилага спрямо това дали играчът е във въздуха.
* Jump() – просто прилагане на сила, което симулира скок на герой, използва се както и за скокът на играч първия път, така и за двойния скок от въздуха.
* HandleJumps() – проверява дали играчът иска да скочи, както и дали може да скочи и първия, и втория път.
* Slam() – Важна функция – едновременно ни позволява да имаме забиването на играча в земята, когато използва съответното умение, но останалото време прилага нормална гравитация на играча. Не може да се използва вградената гравитация на Unity, тъй като тогава е не може да се промени движението на играча по този начин
* Dash() – След проверка за това дали играчът може и жела да използва Dash, играчът получава моментно голямо ускорение в посоката, в която вече се движи. Може да се използва заедно със скок за да се правят гигантски скоци или просто за по – лесно навигация около по – трудни врагове.
* DashCharger() – проста проверка която презарежда това колко Dash-а може да направи играчът

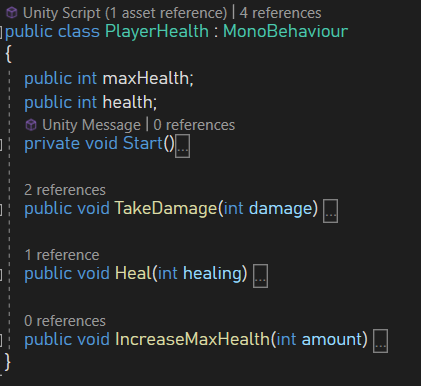
Dash е един от ясно видимите елемнти от играчът: малката чертичка под мерникът на играча показва до колко е зареден следващият ни скок, като когато се напълни спира.

### Живот

Най – простия компонент от играчът, живота на играча се държи в PlayerHealth.

Кодът е сравнително прост, като следи колко живот максимално и в момента има играчът.

Функциите вътре са следните:

* TakeDamage() – функцията, която се достъпва от враговете, за да може да нарани играчът. Вътре и спираме играта, когато живота на играча падне на нула.
* Heal() – позволява да се върне живот на играча, без да надминава рамките на максималния живот.
* IncreaseMaxHealth() – Позволява да се увеличи максималния живот на играчът, съществува за употреба по – нататък.

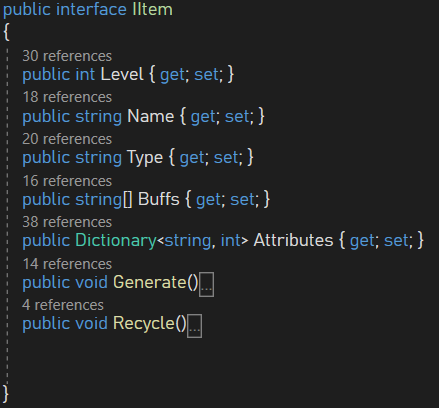
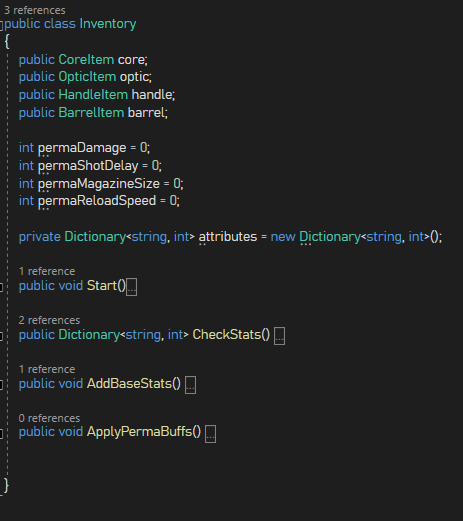
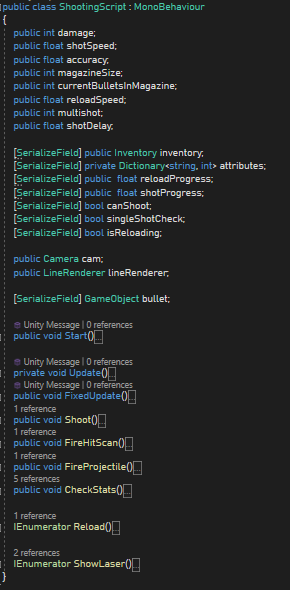
От тук на сетне е въпрос на това къде извън този код ще се използват тези функции за да може да се постигнат каквито ефекти са нужни.

Също така е важно да се отбележи че за сегашната версия на играта не се използва връщането на живот, но то въпреки това е там в случай че решим да го използваме в по – късна версия.

### Стрелба

За разлика от миналото, това е най – сложната механика от играчът, защото точно тук може да се разгледа цялата специална логика за оръжието.

То следва следния ред:

1. Имаме интерфейс Item, който позволява на всички части от оръжието да следват една и съща основна логика. Този интерфейс също се използва от магазина за да могат да се създават всички на куп, без да е нужно това да се притесняваме за всяко едно по отделно. Вътре има следните полета и функции:
   1. Level – int – определя нивото на даденото парче от оръжието. По – нататък то се използва за да може да се определи колко силно да бъде.
   2. Name - string – името на частта, тази която се написва заедно с останалите неща.
   3. Type - string – това поле за сега се използва само от core и handle, като им позволява да следят специални свойства: типа стреляне в handle – Auto и Single, което променя това дали оръжието стреля автоматично между изстрели или дали играчът трябва да натиска за всеки един, а в core определя това дали оръжието ще работи на основа HitScan или Projectile.
   4. Buffs – string[] – Масив от стрингове, който след това се използват за да може да се създадат атрибутите на оръжието
   5. Attributes – Dictionary<string, int> – Тук се определя какво добавя една част към оръжието. String-ът показва коя стойност ще става по – силна, докато int – с колко.
   6. Generate() – Този метод създава свойствата на една част, като пълни Attributes спрямо нивото. Където е нужно също показва какъв тип ще е частта.
   7. Recycle() – неизползван метод, създаден с идеята по – нататък да се използва за това играча да може да си връща част от цената на част.
2. Този интерфейс се наследява от 4 части – CoreItem, BarrelItem, OpticItem и HandleItem. Всеки един от тях използва интерфейса по един и същ начин, като разликите между тях се намират в Buffs и Type, което съответно води и до леки изменения в имплементацията на Generate().
3. Всички тези предмети съществуват заедно в Inventory. Този клас има за цел да събере всичко и от там на сетне да извади нужните атрибути от тях. Всичките заедно се събират в attributes в метода CheckStats(), като преди събирането на всички тях всеки се добавят основни стойности от AddBaseStats(), без които може да има проблеми. Също така използваме вградената Start() функция за създаването на частите вътре и се извиква CheckStats().Тук има друга още неизползвана идея под формата на ApplyPermaBuffs(), като идеята тук е в по – нататъшното развитие на проекта да има начин играчът да получава дълготрайни бонуси, които да не зависят от това какво използва.
4. От тук влизаме в ShootingScript. Тук всичките атрибути заемат своите стойности като променливи и започват да се използват за да стреля играчът. Ето методите и корутините:
   1. Start() – Вграден метод на Unity, извиква се автоматично със създаването на класът. В този случай се използва за да се инициализира инвентара и да се проверят финално всички атрибути чрез CheckStats(). Заедно с това се задават стойностите на някои основни параметри.
   2. Update() – Вграден метод на Unity, който се извиква постоянно. Използваме го за да проверяваме какво иска играчът да прави, като са тук всички проверки с някаква форма вход.
   3. FixedUpdate() – Друг вграден метод на Unity, който се извиква постоянно на регулярни интервали. Тук го използваме за да променяме променливи, които трябва да се изменят с времето (главно проверките за време между изстрели и прогреса на презареждане)
   4. Shoot() – Тук започва процеса на изстрел. Когато играча натисне един от бутоните на мишката FixedUpdate() извиква Shoot(). Първата проверка е за това дали играча може да стреля, спрямо брой патрони, дали презарежда и дали е позволено спрямо Single типа Handle. След това идва проверка за това какъв тип изстрел правим – HitScan или Projectile, и спрямо това викаме съответните методи. В случай че оръжието ни е Single тук също така спираме играча от това да стреля докато не пусне бутона.
   5. FireHitScan() – Тази функция се извиква когато играчът трябва да стреля с тип „HitScan“. За целта правим Raycast, което представлява изстрелването на лъч от играча към посоката на камерата. Ако удари нещо проверява дали е враг чрез tag системата и ако е достъпва класът EnemyHealth, като намаля живота спрямо силата на оръжието.
   6. FireProjectile() – Тази функция се извиква когато играчът трябва да стреля с тип “Projectile”. За целта прави нова инстанция на обекта Bullet, след което се добавя сила в посоката на камерата. Самият куршум има собствен код, който проверява типа на целта и ако удари враг съответно намаля от живота му.
   7. CheckStats() – Подобно на метода в Inventory, този метод има за цел да вземе атрибутите и да ги разпредели по променливи, които можем да използваме:
      1. damage – определя силата на ударите
      2. shotSpeed – определя скоростта на движение на изстрел, в случай че използваме Projectile тип оръжие
      3. accuracy – определя точността на оръжието. За целта използва Random.value, което връща стойност между 0 и 1, намаля го с – 0.5, което прави обхвата от -0.5 до 0.5 и го умножава по точността. Идеята е че когато добавя точността към стойностите тя всъщност е 1 / точност, което води до това по – голямата стойност на точност да намалява девиацията на изстрелите.
      4. magazineSize – определя размера на пълнителя на оръжието, като когато свърши не позволява на играча да стреля докато не се презареди. За да помогне има и променливата currentBulletsInMagazine, в която следи колко патрона има още играча преди да трябва да зареди.
      5. reloadSpeed – определя скоростта на презареждане.
      6. multishot – определя колко куршума изстрелва играча. Multishot = 1 води до това играчът да изстрелва по един изстрел, и за всяко едно след това изстрелва по още един (multishot = 5 -> изстрелване на 5 изстрела). В случай че играчът използва тип „HitScan“ играта просто добавя към силата на изстрелите.
      7. shotDelay – определя времето между изстрели. Колкото по – висока е тази стойност, толкова по – бързо се изпълнява.
   8. IEnumerator Reload() – Тази корутина извършва презареждането. Когато се викне при натискане на бутон “R” и се отчете от Update() метода води до това да започне процеса. За целта спираме играча от това да стреля и караме играта да изчака време, спрямо времето което се задава от reloadSpeed. Когато мине се напълват патроните и позволяваме на играчът отново да стреля.
   9. IEnumerator ShowLaser() – Тази корутина се използва от FireHitScan() за да покаже визуално изстрела. За целта използва LineRenderer, чиито позиции задава спрямо позициите на играча и или далечна точка в посоката, в която стреля играчът, или точката, в която е ударила Raycast от метода FireHitScan(). След къс момент LineRenderer се изключва за да се създаде илюзията за изстрелването на лазер.

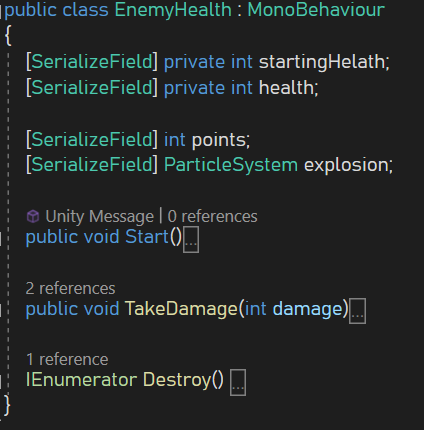
Всички тези стъпки заедно водят до пълното усещане за стрелба на играчът, като много от това как се прилагат стойностите вътрешно е под въпрос постоянно по време на разработката с цел постепенно стойностите да се докарат до момент, в който играта е балансирана т.е. играчът получава преживяването, което ние искаме, а не много по – трудно или много по – лесно.

## Врагове

Враговете в тази игра се делят на 3 типа – Flying, Turret и Spawner. Всеки един от тях има уникален смесица от уникални и общи компоненти. За по – ясно обяснение първо ще обясня общият елемент за всички тях – живота, и след това ще обясня индивидуалната имплементация на всеки един враг.

### Живот на враговете

Всички врагове споделят класът EnemyHealth. Той съдържа:



* Няколко основни променливи:
  + startingHealth
  + health
  + points
* Референция към ефект explosion
* Вградената функция Start(), която се използва за да се инициализира живота на врага.
* Функция TakeDamage(), която се използва всеки път, когато трябва да се намали живота на враг. В случай че живота на врага падне до 0 се вика корутината Destroy()
* Корутина Destroy() се използва за да унищожи врага, като преди това пуска ефект с цел да се вижда когато врагът бъде унищожен.

### Flying Enemy

Летящият враг работи въз основа проста логика – той се насочва към играча и се движи към него, като го наранява ако го докосне. За да работи използва 3 класа – EnemyHealth(обяснен по - горе), FlyingEnemy и EnemyContactDamage.

FlyingEnemy е прост клас, който просто намира играча и започва да се движи към него постоянно.

EnemyContactDamage работи въз основа OnCollisionEnter(), като вътрешно проверява две неща – дали се допира с играча, като в този случай се взима от живота на играча, и дали трябва да се унищожи след удар. Това е важно защото този клас се използва от един от другите врагове с тази промяна в логиката.

### Turret Enemy

Този враг има само два класа – Enemy Health и TurretEnemy. Работи по следния начин:

Когато играчът се доближи до него почва да се зарежда (нещо, което визуално се показва чрез нарастваща светлина). Когато се заради напълно наранява играча.

Тази логика се изпълнява в класът ZoneEnemy, като вътре едновременно повдига и намаля това колко се зарежда самият враг, но и контролираме светлината закачена за него.

### Spawner Enemy

Този враг работи, като създава снаряди, които могат да следват играча. За целта има цял отделен обект – HomingMissile. С помощта на таймер постепенно се появяват нови и нови снаряди, които изчезват след като докоснат нещо или след като бъде унищожен самият основен враг, от който се появяват.

За целта се използват класът EnemySpawner, който на редовен период създава инстанции на снаряди.

Самите снаряди използват класовете FlyingEnemy и EnemyContactDamage, които бяха обяснени в FlyingEnemy. Разликата е в това че тук се използва промяната на EnemyContactDamage, която позволява на снарядите да изчезнат след като ударят играчът.

## Генерация на рундове / Магазин

В сегашната версия на играта, всеки рунд се дели на две основни части – магазин и борба.

Цикълът между тези две части се изпълнява от LevelManager, който инстанцира враговете и използва класът ShopManager за да показва магазина. За сега, всеки рунд се създават по 1 от всеки вид враг за всеки рунд, като враговете се показват или на случайни позиции (в случая на Turret и Spawner) или

По време на магазина играчът използва точките, които е придобил по време на борбата, за да подобрява оръжието си. За целта се използва ShopManager – клас, който позволява на играча да взима части от магазина, създава части и позволява на играча да плаща за това да се показват нови части.

За целта се използват следните методи:

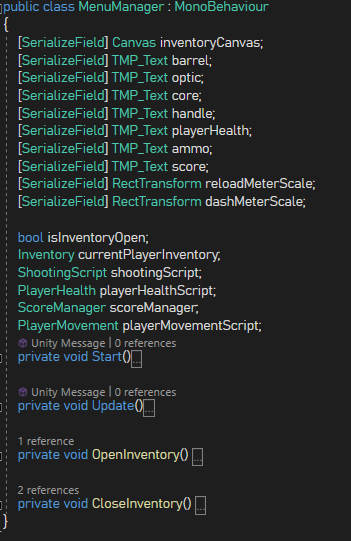
* GenerateNewShopItem() – създава нов предмет от случаен тип, както и цената му.
* TryToBuy() – проверява дали играча може да вземе даден предмет и го добавя към инвентара.
* ManageInputs() – Проверява дали играчът иска да купи даден предмет, да плати за да обнови магазина и да го затвори.
* RefreshShop() – Презарежда магазина, като го пълни с нови случайни предмети.
* DisplayShop() – Показва магазина на играча и позволява да се използва.

## Интерфейс

Играта има няколко основни визуални елемента, които се показват по време на действието. Всички те се управляват от един клас, с името MenuManager. Надолу са описани елементите (наредени от горе надолу):

* Брой на точките
* Точка, използвана за прицелване
* (Обикновено скрит) – Прогрес на презареждане – черта, която се появява когато играчът презарежда, за да показва колко време остава.
* Dash Counter – Подобно на чертата за презареждане, но работи на обратно – когато е пълна, показва че играчът може да използва Dash, а не обратното
* Живот (HP) – показва количеството живот което има играчът, както и максималната възможна стойност
* Изстрели – подобно на живота, показва колко патрони има играчът в пълнителя и колко е моментният максимум

За да се поддържат тези елементи класът събира всички тях и съответно променя когато е нужно.

Повечето се извършва в Update(), като методите OpenInventory() и CloseInventory() се занимават с това да се обновява информацията на инвентара, който се показва на играча.

За всички визуални текстови елементи използва добавката TextMeshPro, която има за цел по – правилно да показва текста, независимо от резолюцията на потребителя.

## Версии

Играта в момента има версия само за Windows, която работи със стандартния за компютърните игри мишка и клавиатура.

Други версии могат да се създадат, но засега не са пробвани.

За крайния потребител процеса за пускане е лесен, тъй като изисква само да се пусне CyberGun.exe, и играта автоматично тръгва без нужда за друг софтуер.

# Глава 3. Ръководство за потребителя

## Започване на играта

За да започне играта се изисква само играчът да използва CyberGun.exe

Сегашната версия работи само на Windows и е изпробвана на Windows 10

## Контроли

Ето пълен списък с контролите:

* Движение
  + Ходене – W,A,S,D – направо, наляво, назад, надясно (стандартно за тези игри)
  + Скок - Space
  + Dash – Shift
  + Ground Slam – Left Control
* Стреляне
  + Изстрел - Mouse Button 1 (Left click)
  + Презареждане - R
* Използване на магазина
  + Купуване на предмет - 1, 2, 3, 4
  + Опресняване на магазина с нови предмети - 5
  + Затваряне на магазина - E
* Други
  + Връщане в главното меню – Escape
  + Отваряне на инвентара - Tab

# Глава 4. Бъдещи планове

Сегашната версия на играта е базова, и има много елементи, които могат да се добавят за да създадат по – пълно усещане.

* Добавяне на звук
* Добавяне на повече врагове
* Добавяне на случайни елементи като платформи за различните рундове
* Разработка на по – качествена система за създаването на нови врагове
* Различни типове игра (например игра с цел най – висок резултат до даден рунд)
* Добавяне на различни типове за OpticItem и BarrelItem
* Добавяне на предварително наредени рундове (създава опции за това играчите да играят по един и същ начин)

# 

# Информационни източници

* Unity
  + Документация: <https://docs.unity.com/>
  + Основа на движението: <https://www.youtube.com/watch?v=LqnPeqoJRFY>
  + Основа на Shader: <https://halisavakis.com/>
* Blender
  + Документация: <https://docs.blender.org/manual/en/latest/>
* Game Design (различни източници, които са част от проучването, както и официални уебсайтове на подобни игри)
  + <https://www.youtube.com/@GMTK>
  + <https://store.steampowered.com/app/1229490/ULTRAKILL/>
* Снимки от игрите са взимани от официалните им страници в <https://store.steampowered.com/>

# Рецензия на дипломен проект

|  |  |  |  |
| --- | --- | --- | --- |
| **Тема на дипломния проект** |  | | |
| **Ученик** |  | | |
| **Клас** |  | | |
| **Професия** |  | | |
| **Специалност** |  | | |
| **Ръководител- консултант** |  | | |
| **Рецензент** |  | | |
| **Критерии за допускане до защита на дипломен проект** | | **Да** | **Не** |
| Съответствие на съдържанието и точките от заданието | |  |  |
| Съответствие между тема и съдържание | |  |  |
| Спазване на препоръчителния обем на дипломния проект | |  |  |
| Спазване на изискванията за оформление на дипломния проект | |  |  |
| Готовност за защита на дипломния проект | |  |  |
| Силни страни на дипломния проект |  | | |
| Допуснати основни слабости |  | | |
| Въпроси и препоръки към дипломния проект |  | | |

**ЗАКЛЮЧЕНИЕ:**

Качествата на дипломния проект дават основание ученикът/ ученичката ................................................... да бъде допуснат/а до защита пред членовете на комисията за подготовка, провеждане и оценяване на изпит чрез защита на дипломен проект- част по теория на професията.

.........05.2023г. Рецензент:................................

град София