**ТЕХНОЛОГИЧНО УЧИЛИЩЕ „ЕЛЕКТРОННИ СИСТЕМИ"**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: 3D шутър игра от трето лице в обстановката на Дивия Запад

Дипломант: Научен ръководител:

*Виктор Георгиев Михаил Пейков*

СОФИЯ

2019

**TЕХНОЛОГИЧНО УЧИЛИЩЕ**



**ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 06.11.2018 г. Утвърждавам:

Дата на предаване: 06.02.2019 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Виктор Георгиев Георгиев 12 А клас

1.Тема: 3D шутър игра от трето лице в атмосферата на Дивия Запад

2.Изисквания:

2.1. Играта да се разработи, използвайки Unreal Engine 4

2.2. Играта да е от трето лице с камера зад гърба на героя

2.3. Обстановката да наподобява на Дивия Запад от холивудските филми

2.4. Играчът да има възможност за минимум два различни типа оръжие

2.5. Играчът да се изправи срещу минимум два различни типа противници

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант:

Ръководител:

/ Михаил Пейков /

Директор:

/ доц. д-р инж. Ст. Стефанова /

Дипломното задание е изпълнено на високо ниво, с много старание.Резултатът надминава първоначалните ми очаквания като качество. Атмосферата на Дивия Запад е предадена ефектно, без излишни усложнения, но с внимание към детайлите на правилните места.

Показани са умения както по програмиране така и за работа с Unreal Engine. Проекта е добре структуриран, кода е четлив, спазва униформен стил, без излишни усложнения.

София, Ръководител:

27.02.2019 / М. Пейков /

# **УВОД**

Дипломната работа представлява игра, разработена на игровия двигател Unreal Engine 4. Играта е 3D шутър от трето лице в атмосферата на Дивия Запад. Обстановката е базирана и повлияна от тези, често показвани в холивудските филми. Играчът ще има възможността да обикаля свободно в едно малко градче, изпълнено с опасностите на Дивия Запад. Целта му е да оцелее колкото може по-дълго на това опасно място. Камерата от трето лице ще му позволява лесно и плавно да се оглежда около героя, когото управлява. Още в началото на играещия ще бъде предоставен револвер. Ще може да го вземе като се приближи достатъчно близо до него. Малко по-късно ще се появи възможност играчът да вземе и другото налично оръжие в играта – пушката. Тези две оръжия ще са му от огромна подкрепа в изправянето му срещу противниците. Револверът ще се отличи с голямата си скорост на стреляне, но ще бъде много по-неточен от пушката, която обаче ще стреля сравнително по-бавно. Враговете ще са няколко видове, всеки предоставящ различни опасности. Първият вид, срещнат в играта, ще преследва главния герой докато го стреля с револвер. Следващият ще се показва мигновено от прозорците на някои от сградите в града, стреляйки с пушка. Някои противници ще се показват и на терасите, а други ще се крият, изчаквайки появата на играча в кръчмата. Той ще може да влезе в нея през вратите й, където да се скрие за няколко секунди, докато не влязат враговете. Героят, управляван от него, ще има уникалната способност да забавя времето, което ще му даде преднина срещу многото съперници. Оръжията ще бъдат с изчерпаеми патрони, затова играещият ще трябва да обикаля и да се снабдява, преди да е останал беззащитен. За повишаване на визията на играта, героите ще извършват множество движения посредством действията си, които, благодарение на анимации и визуални ефекти, ще са видими. Потребителският интефейс ще посрещне играчът с главно меню след стартирането на играта, откъдето ще има възможността да избере нивото на трудност, пред което да се изправи. Той също така и ще насърчава играча, като в долния ляв ъгъл на екрана ще има две ленти, показващи нивата на здравето и способността на героя. В долния десен ъгъл ще се изписват оставащите амуниции за съответното оръжие. По време на игра, тя ще може да бъде паузирана, отваряйки се друго меню, даващо възможност за настойване на силата на звука и чувствителността на мишката, както и връщане към главното меню. Ще бъде наличен и звук, обхващащ всякакви звукови ефекти, породени от събитията, и фонова музика, вдъхващи допълнителен живот в играта.

# **ПЪРВА ГЛАВА.**

# **МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА ВИДЕО ИГРИ**

## **Oсновни принципи, технологии и развойни среди за реализиране на видео игри**

### **Игрови двигатели**

Игровият двигател поставя софтуерната структура за изграждане и създаване на видео игри. Той предоставя всякакви функционалности от анимации до изкуствен интелект. Игровите двигатели са отговорни за визуализирането на графики, откриването на сблъсъци, управлението на паметта и много други опции.

Игровите енджини предоставят на разработчиците инструменти за създаване на многобройни игрални приложения. Дизайнерите често използват повторно тези двигатели, за да създават други игри, което го прави ценна инвестиция.

Те съдържат пет основни компонента:

* основната игрална програма, която съдържа логиката на играта;
* двигател за визуализация, който може да се използва за генериране на 3D анимирани графики;
* аудио двигател, който се състои от алгоритми, свързани със звуци;
* физически двигател за прилагане на „физически“ закони в системата; и
* изкуствен интелект - модул, предназначен да бъде използван от софтуерните инженери със специално предназначение.

### **Компонентите на съвременните игрови двигатели**

#### **1.1.2.1. Контроли**

Един от най-важните аспекти на една игра са обширните начини, по които може да се играе. Затова енджините обикновено поддържат набор от типове за контрол: клавиатура, мишка, геймпад и допир са основните като има и много други по-малко известни. Много са различните начини за управление на входа, но най-използваният са събитията. Входните събития работят като компютърът слуша за някаква форма на въвеждане (натиснат е бутон на мишката, клавишът на клавиатурата е освободен, осите на джойстика са променени, сензорът за допир е натиснат) и е активиран нашият персонализиран код. Това може да се комбинира с "картографираща таблица", която свързва клавишите на клавиатурата / контролера / мишката към дадени действия, наречени например "скачане" или "стреляне", така че да можем да изградим логиката на играта ни, без да се притесняваме от желанието на потребителя да играе, използвайки различно оформление от това, с което е изградена играта. Проучването обикновено се прави, когато става въпрос за позиционни стойности, като например координатите на мишката или размера на аналоговия стик на геймпада. Игровият механизъм осигурява средства за извличане на тези стойности всеки път, когато разработчикът иска, и зависи от разработчика да реагира на промените в тези стойности, независимо дали става въпрос за преместване на символ или промяна на позицията на потребителския курсор на мишката.

#### **1.1.2.2. Графика**

Също така, каква би била играта, без качествена графика? 3D игрите са изградени върху 3D активи, които обикновено се създават във външна програма за 3D рендериране, като например Maya или Blender, и се внасят в енджина. Игровите двигатели поддържат много от тези формати, за да могат да се внесат и директно да се използват.Те също така разполагат с множество технологии за осветление и ефекти, които дават живот на добавените от нас активи, както и управлението на техните анимации, включително по-сложни неща като смесване на анимации за преминаване между бягане / скокове / стрелба по правдоподобен начин.

За пълното и подробно описание на всички графични функции, които осигуряват игровите двигатели, ще е необходимо да бъдат изписани доста страници и би отнело значително време, но по същество всички двигатели на играта са там, за да направят задачата ни възможно най-проста и ясна. Разработчиците не желаят да се отделят време и усилия за превръщането на внимателно изработените 3D модели в загадъчни формати или ръчно да създават мета-данни, за да ги покажат правилно. Двигателите премахват всички тези досадни задачи. Това, в съчетание с ефектите след обработка, изграждането на терена и партикъл ефекти, означава, че можем да създадем цял игрален свят в игровия енджин.

#### **1.1.2.3. Звук**

Звукът също е неразделна част от игрите, въпреки че през повечето време се пренебрегва. Добавянето на звукови ефекти към игрите не е толкова просто, колкото бихте си помислили, особено с появата на 3D игрите.

Звуковите ефекти обикновено не излизат само от високоговорителите, тъй като са били записани, но повечето двигатели на игри имат средствата да стартират звуци в 3D света, които ще си променят силата в зависимост от товакъде се намира нашият герой в пространството. Също така има много начини да се подобри реалистичността на звука, като се добави модулация на звука и реверберация, за да изглежда, че звукът отскача от стените на заобикалящата го среда.

Музиката и интерфейсните звуците работят по различен начин, тъй като те се добавят без оглед на 3D позиция. Очевидно е, че двигателят трябва да осигури средствата за регулиране на музиката, за да отговаря на настроението на играта, а не просто да я изрече с пълен обем.

#### **1.1.2.4. Физики**

Физиките в игрите включват въвеждането на законите на физиката, с цел да се направят ефектите по-реалистични за наблюдателя. Обикновено симулирането на физиката е бледа имитация на действителната физика в реалния живот и изчисленията се извършват с помощта на дискретни стойности. За да бъдат симулирани физики към куб, например, и да реагира на гравитацията или да бъде изтласкана / изстреляна от потребителя, трябва да му е дадена физическа форма, която може да не е същата като визуалната форма, както и маса, триене, подскачане и други свойства, за да се създаде обект, който може да взаимодейства със света около него. Има няколко елемента, които формират компонентите на физиката на симулацията, включително физическия двигател, програмния код, който се използва за симулиране на Нютоновата физика в околната среда, и откриване на сблъсъци, използвани за решаване на проблема с определянето кога два или повече физически обекта в околната среда се засичат.

#### **1.1.2.5. Скриптиране**

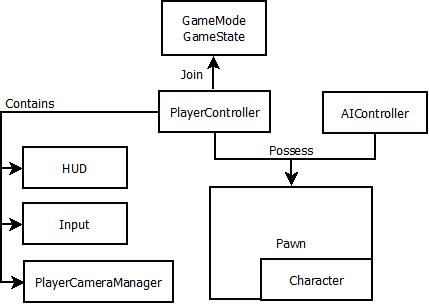
Друга огромна част от игровите двигатели са предварително изградени скриптове, които могат да бъдат прикрепени към обекти в света на играта. Терминологията се различава от един двигател до друг двигател, като някои използват термина "поведение", докато други говорят за "обекти на играта", които се различават от "визуални обекти". Повечето игрови двигатели идват със записки за иницииране на игра с играч в определена позиция, добавяне и преместване на камери, стартиране / спиране на генератори на частици, манипулиране на светлини, задействане на събития, когато играч се премести в област и цял куп други функции. В някои случаи може да е толкова просто, колкото да се добави 3D модел в света на играта и да се определи като играч от трето лице, а двигателят на играта ще има предварително изградени скриптове за преместването и анимирането на героя, завъртане на камерата и светът реагира на позицията на играча.

## **Съществуващи решения и реализации**

### **Unreal Engine**

#### **Контролите и игровата структура**

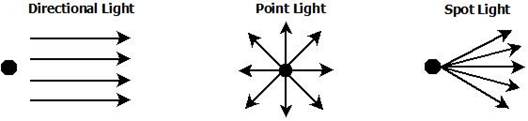
Unreal Engine се състои от входна система, която преобразува натискане на бутони и клавиши от играча в събития, извършени от управлявания герой по време на игра. Тази входна система може да бъде конфигурирана чрез рамката на играта. Тя съдържа функционалността за проследяване на напредъка и контрола на правилата на играта. Heads-up дисплеите (HUDs) и потребителските интерфейси (UIs) са част от рамката на играта, за да осигурят обратна връзка към играча по време на игра. Игровите класове като GameMode, GameState и PlayerState определят правилата и контролират състоянието на играта. Героите в играта се контролират от играещия (използвайки класа PlayerController) или от компютърния AI (използвайки класа AIController). Независимо дали са контролирани от играча или от компютъра, те са част от базовия клас Pawn. Класът Character е подмножество на класа Pawn, който е специално създаден за човекоподобни герои с вертикална ориентация.



**Фиг 1.1. Блок-схема, изобразяваща връзките между контролерите, геоите, интерфейсите и режима на игра**

#### **Осветлението**

Светлината е мощен инструмент, използван за създаването на играта. Може да се употребява по много начини, като например да се създаде настроение на дадена сцена или да се съсредоточи вниманието на играча върху дадени обекти в играта. Unreal Engine 4 осигурява набор от основни светлини, които могат лесно да бъдат поставени на нивото. Те са насочена светлина (Directional Light), точкова светлина (Point Light), прожекторна светлина (Spot Light) и небесна светлина (Sky Light). Насочената светлина излъчва лъчи от паралелни светлини, точковата светлина излъчва светлина като крушка (от една точка, радиално навън във всички посоки), прожекторната светлина излъчва светлина в конусообразна форма и небесното осветление имитира светлината от небето, насочена върху сцената:

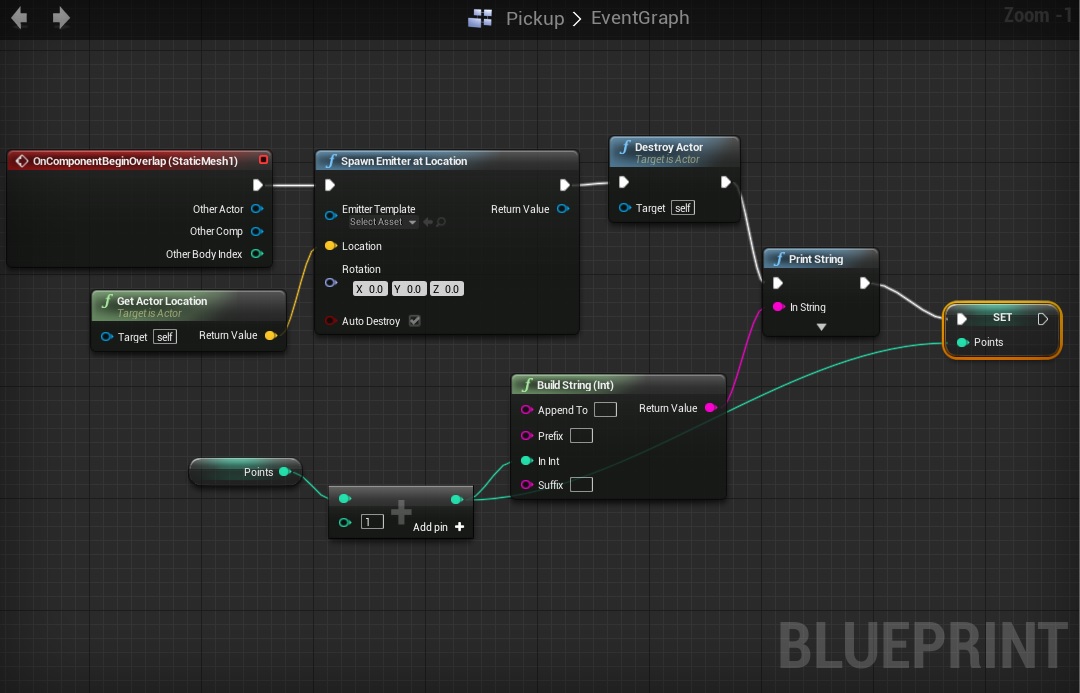


**Фиг. 1.2. Визуализация на светлините**

Ефективният дизайн на светлината създава и реалистични сенки. Избирайки правилните типове светлина в нивото, можете да повлияете както на настроението, така и времето, необходимо за визуализиране на сцената, което на свой ред влияе на кадрите в секунда на играта. В играта можете да имате два вида сенки: статични и динамични. Статичните сенки могат да бъдат предварително изчислени в сцената и затова бързо се рендират. Динамичните сенки се променят по време на изпълнение и са по-изискващи за изобразяване.

#### **Blueprint визуалният скриптов език**

Blueprint системата е нова функционалност в Unreal Engine. Unreal Engine 4 е първият двигател, който използва тази революционна система. За тези, които са запознати с Unreal Engine 3, може да се мисли за подобрена и подобрена комбинирана версия на Unreal scripting system, Kismet и Prefab функционалностите. Системата за визуално скриптиране Blueprint ни дава възможност за разширяване функционалността на кода, използвайки визуален скриптов език (диаграми на потоци, подобни на кутии, свързани с линии). Тази възможност означава, че не е необходимо да пишете или компилирате код, за да създавате, подреждате и персонализирате поведението / взаимодействието на обекти по време на игра. Това също така дава възможност на не-програмисти (художници / дизайнери) да създават прототип или да създават бързо ниво и да манипулират геймплея, без да се налага да се справят с предизвикателствата на програмирането на логиката на играта. Blueprint функцията работи подобно на обикновените езици за програмиране. Например: можем да създадем променлива, като кликнем върху обекта и изберем Create Variable. Не се изиксва и един написан ред от код. Разработчиците могат да го направят, без да отделят време за сложно кодиране.



**Фиг. 1.3. Примерен вид на логика, изградена на Blueprint**

### **Unity**

Игровият двигател Unity предлага широк набор от функцоналности и е сравнително лесен за използване, поради удобния му интерфейс. Разработените на него игрит могат бързо и лесно да бъдат пренесени на различни мобилни операционни системи като Android, iOS, Windows Phone 8 и BlackBerry, което го прави чудесен енджин за развитието на мобилните игри. Той също така има възможности за разработка на конзоли, за което се изисква да достъп до конзолни SDK, които се недостъпни за навлизащите разработчици. Двигателят поддържа активи от големите 3D софтуерни приложения като 3ds Max, Maya, Softimage, CINEMA 4D, Blender и други, което означава, че няма реални ограничения за типа файлови формати, които поддържа. Той има безброй 2D възможности, функционалности и 2D физика, което го прави чудесен игрови двигател, който да се използва за разработването на 2D игри. Той също така е и много удобен за направата и на 3D игри, поради леснотата на използване и широкоразпространеното общество.

Въпреки че двигателят поддържа интеграция на почти всяко 3D приложение, той обаче страда от възможностите за редактиране в редактора на двигателя. Unity няма реални моделиращи или конструктивни характеристики извън няколко примитивни форми, така че всичко ще трябва да бъде създадено в 3D приложението от трета страна. Той обаче се гордее с голяма библиотека с активи, в която може да се изтегли или закупи голямо разнообразие от активи.

# **ВТОРА ГЛАВА.**

# **ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА ВИДЕО ИГРАТА**

## **2.1. Функционални изисквания към видео играта**

За създаването на видео игра е необходим игрови двигател, предоставящ всички основни компоненти, нужни за придаването на играта завършен вид. Това са двигатели за рендъринг, физики, скриптиране, звук, анимации и др. Другата съставна част е софтуерно приложение за интегрирана среда за разработка, предоставящо компилатор, редактор на код, инструменти за автоматизиране построяването на изходното приложение и др. На него се написва логиката, изграждаща механиките на играта. За този проект използвам Unreal Engine 4, който работи единствено с Visual Studio.

## **2.2. Съображения за избор на програмни средства и развойната среда**

Избрах Unreal Engine 4 поради неговата лекота на използване и удобен потребителски интефейс. Визуалният скриптов език Blueprints, вграден в енджина, затвърди избора ми, заради бързината му и лесния начин за създаване на прототипи. Visual Studio е единствената развойна среда, съвместима с двигателя. Програмният език за разработка е C++, на който е изграден и самият енджин. Другият избор, който можех да направя, беше игровия двигател Unity, но го пренебрегнах поради по-малкия ми опит с него.

## **2.3. Проектиране на героите**

За улесняването на имплементацията на всички видове герои е използван абстрактеният клас CharacterBase, изпълняващ ролята на база. Той наследява Character класа, специално създаден за човекоподобни герои. Класът-родител му предоставя някои компоненти, улесняващи неговата реализация като капсула, скелетово тяло и компонент за движение. Също така дефинира конструктор, методите BeginPlay, който се изпълнява при пускане на играта, и Tick, който се извиква всеки кадър. CharacterBase класът дефинира променливи и методи, реализиращи функционалности за движението, стрелбата и елиминацията на героите.

## **2.4. Главният герой и неговото управление**

Създаден е класът MyCharacter, наследяващ базовия CharacterBase, за главния герой, когото играчът управлява. Към него са добавени компонентите камера и пружинен лост, както и променливи и методи обхващащи прицелването с оръжие, движението на камерата, капацитета на амунициите на оръжията, смяната на оръжие и забавянето на времето. Камерата е закачена за лоста, за да стои винаги на еднакво разстояние от героя и да гледа към него. Играчът има възможността да обикаля с героя навсякъде, където му е позволено. Това става благодарение на клавишите W, A ,S ,D и мишката, с която управлява движението на камерата. Натискането на W премества героя напред, A - наляво, D - надясно и S - назад. В настройките на проекта се задават тези клавиши и стойност, която да върнат при тяхното натискане. Посоката на движението на героя е относителна към посоката на камерата.



**Фиг. 2.1. Визуализация на изпълнението на движението**

За движението напред се използва методът MoveForward, който се извиква при натискане на W и S, а за настрани - MoveRight, който се извиква при натискане на A и D. За преместването на камерата хоризонтално (по Yaw оста) се използва методът LookSide, който се извиква при преместването на мишката хоризонтално. За преместването на камерата вертикално (по Pitch оста) се използва методът LookUp, който се извиква при преместването на мишката вертикално. Тези методи завъртат пружинения лост, а не камерата, която е с константна ротация.

## **2.5. Проектиране на оръжията**

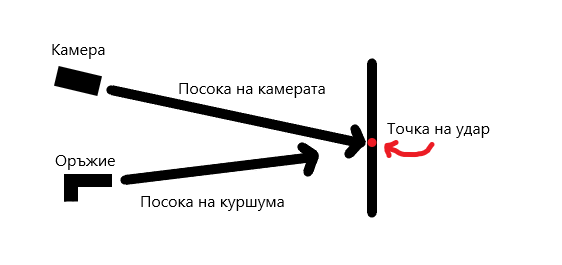
За улесняването на имплементацията на двата вида оръжие е използван абстрактеният клас GunBase, изпълняващ ролята на база. Той наследява Actor класа, предоставящ многобройни методи, без които нямаше да е възможна неговата реализацията. Едни от тях са вече споменатите BeginPlay и Tick. GunBase съдържа член-променливи и методи, използвани от всички наследяващи класове. Използвани са компонентите сфера и статично тяло, както и променливи и методи, реализиращи функционалности за вземането на оръжието, изчисляването на точността и инстанцирането на куршума.

## **2.6. Видовете оръжия**

Първото оръжие, което играчът взима в играта е револверът. Реализиран е чрез създаването на класа Revolver, с GunBase като клас-родител. Приближавайки се достатъчно близо до него, героят ще го вземе, ако все още не се е сдобил с такова. Ако пък вече има, той ще си запълни капацитета от амуниции за даденото оръжие. Револверът ще се закрепи отдясно на ханша му. Това се извършва в OnEnterSphere метода. Другото оръжие в играта е пушката. Тя е имплементирана в класа Rifle като отново за родител е използван класът GunBase. Ще се закрепи на гърба на героя, след като бъде взета. Играчът трябва да каже на героя кое оръжие да извади чрез натискане на бутоните 1 и 2. Само тогава ще може да ги използва. Клавишът 1 е за револвера, а 2 - за пушката. Той ще може да разменя оръжията по всяко време, освен ако не се е прицелил. Тази логика се извършва в ChangeToPistol и ChangeToRifle методите в MyCharacter класа. Те имат различна скорост на стреляне и щета. Скоростта се управлява от таймери отново в класа на главния герой.

## **2.7. Куршумите**

Създаден е класът Projectile, за да се реализират куршумите, изстрелвани от оръжията. Отново е наследен класът Actor. Употребени са компонентите сфера, статично тяло и специализираният компонент ProjectileMovement, даващ движение и симулиращ физики. Методът OnHit реализира логиката при уцелването на даден обект. Засичането на уцелването функционира само когато са симулирани физики към куршума, за което се погрижва ProjectileMovement компонентът . OnHit се извиква, когато сферата засече друг обект и ProjectileMovement се заема с преместването на куршума в пространството. При изстрелване, куршумът ще се появи на края на дулото на оръжието и бързо ще се придвижи в посока централната точка на екрана, оставяки зад себе си диря. Също така на края на дулото ще се стартира партикъл ефект, наподобяваща експлозия, в класа на оръжието чрез SpawnEmitter метода. Докато се движи куршума в пространството, ще се стартира и партикъл ефект, наподобяващ диря. Дирята особено помага на играча да разбере откъде го стрелят, ако враговете не са видими на камерата. За изстрелването на куршума в посока на мерника е написан алгоритъм в GetHitRot метода в GunBase класа. Той преценява с каква ротация да се инстанцира куршумът в зависимост от мястото, където ще се спре.



**Фиг. 2.2. Визуализация на логиката на алгоритъма в GetHitRot метода**

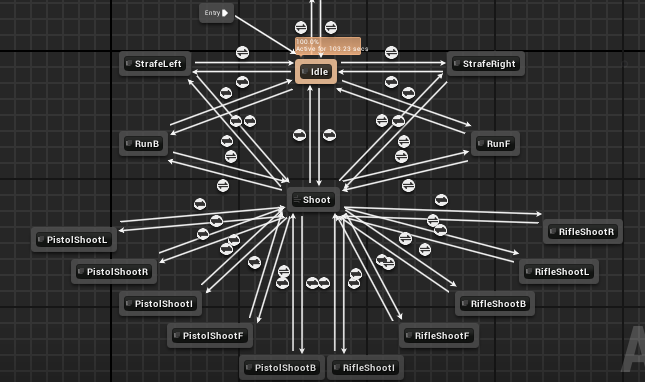
С уцелването на някакъв обект се установява неговият тип. Уцелен ли е герой, ще се намали неговата кръв в зависимост от това дали изстрелът е попаднал в тялото или главата му. Също така ще се пусне анимация, показваща реакцията му от изстрела. Тази анимация ще се стартира най-малко половин секунда след края на миналата поради високата скорост на стреляне на оръжията. Това е реализирано с помощта на таймери. Ако след уцелване му бъде свалена цялата кръв, ще се стартира анимация, показваща неговата елиминация и ще хвърли оръжието си. Тогава ще се стартира партикъл ефектът, наподобяващ плискаща кръв. В противен случай инстанцираният куршум ще се унищожи и стартира партикъл ефектът, наподобяващ експлозия.

## **2.8. Стрелбата и способността за забавяне на времето**

При наличието на оръжие играчът ще може да стреля в движение или в стационарно оложение чрез натискане на левия бутон на мишката. Тогава се извиква методът Fire от MyCharacter. Играещият също така решава дали да изстрелва патроните директно или от целева позиция чрез задържане на десния бутон на мишката. Тогава камерата се приближава към героя и застава от дясната му страна, чрез метода CameraZoom. При отпускане на бутона се извиква CameraOutZoom с цел връщане на камерата в началното й положение. Това дава по-добра видимост към противника и по-голяма точност на оръжието, но героят ще се движи сравнително по-бавно от обикновеното. С всеки един убит противник се увеличава нивото на способността му за забавяне на времето, който се пуска с натискането на клавиша E, докато не стигне максималния си капацитет. С клавиша E се извиква EnterSlowMo методът. Колкото по-високо е нивото, толкова по-дълго ще може да използва способността си. Това дава огромно предимство на играча срещу противниците, тъй като се забавя всичко освен движението на героя. Това засяга и скоростта на стреляне и скоростта на регенериране на кръвта. При убиване на протвиник също така има 25% шанс, ако той е на определено разстояние от главния герой, да се активира ефект, приближаващ камерата към съперника и забавящ времето. Този ефект е изцяло козметичен. Приближаването на камерата е реализирано чрез промяната на FOV на камерата, което реално я мащабира. При изстрелването на куршум се извиква SpawnProjectile методът от класа на даденото оръжие. Той ще инстанцира куршума на края на дулото и ще прецени дали ще отиде в целта или ще се добави леко отлконение. Тази преценка се оставя на случайността.

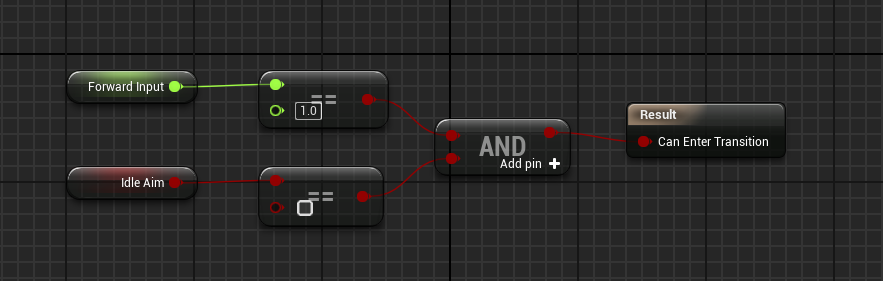
## **2.9. Проектиране на анимациите на главния герой**

Създаден е класът AnimationHandler, наследяващ AnimInstance класа, специализиран за управление на анимации. В него имаме член-променливи, съдържащи информация за движението, оръжието в дадения момент, момент на стрелба и др. на главния герой. Единственият метод е UpdateAnimationProperties. Неговата цел е да достъпва управлявания от играча герой и да актуализира дефинираните променливи. С тях ще се извършват транзициите между различните анимации. За самия контрол над анимациите на героя е създаден Animation Blueprint, който използва вече създадния AnimationHandler за клас-родител. В графата му за скриптиране е предварително дефиниран BlueprintUpdateAnimation евентът, който се извиква постоянно. Към него свързваме метода UpdateAnimationProperties. Така ще актуализираме стойностите постоянно. В графата за анимации е създадена машина за състояния, изкарваща като изход една анимация. В нея имаме зададена начална анимация, свързана към Entry. Това е анимацията, работеща когато се създаде героя и не се натискат никакви клавиши, които го управляват.



**Фиг. 2.3. Реализация на движението на главния герой в графата**

Чрез променливи нагласяваме преходите между отделните анимации. Тези променливи са наследени от класа-роител. За улеснение на подреждането и преходите между анимациите е използван и т. нар. Conduit. Той се използва за представянето на едно действие по няколко различни начина. На примера са използвани различни анимации в зависимост от посоката на движение на героя. Използван е и Conduit за прицелването, тъй като оръжията се хващат по различни начини. Във всяко едно различно блокче е зададена анимацията според даденото действие.

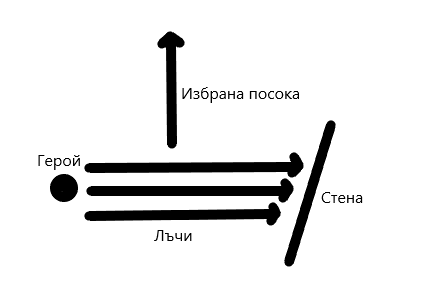


**Фиг. 2.4. Транзицията от стационарно състояние в тичане напред**

Транзицията в обратна посока се постига по същия начин.

## **2.10. Противникът, преследващ главния герой**

Първият вид противник е този, тичащ към нашия герой, докато го стреля с револвер. Създаден е класът GroundEnemy, наследяващ CharacterBase. Към него са добавени член-променливите за дистанцията между него и главния герой, това дали се завърта на едно място и класът на револвера. Методите са за намирането на дистанцията, промяна на посоката и намирането на път. Той ни преследва винаги, докато се поддържа определено разстояние между него и героя. Ако се намали това разстояние, той ще застане на едно място, докато то отново не се увеличи. Тук се използва булевата променлива за завъртане на едно място. Чрез нея ще се стартира анимацията за завъртане. Предназначено е да се контролира движението му само направо в MoveForward метода, докато маневрите му се управляват чрез завъртането на героя по Yaw оста в Rotate метода. За да ходи след главния герой, противникът трябва да гледа към него. Това е постигнато чрез използването на метода RotateToCharacter. В града има сгради и постройки, в които той ще може да се блъсне и съответно да забие на едно място. Затова постоянно се извиква LineTrace методът, който намира по-подходящата посока, която да поеме. Веднага след като заобиколи дадения обект, ще се върне към преследването на играча. За написването на скрипта са използвани лъчи. Той също така ще внимава да не се блъсне и в други врагове от неговия тип.



**Фиг. 2.5. Визуализация на логиката на в LineTrace метода**

Героят стреля постоянно като се извиква Fire методът. В него се извиква SpawnProjectile методът в класа на револера. Неговият е с по-ниска скорост от този на героя, управляван от играча. Шансът куршумът да уцели желаната цел отново е случайно избран. Стрелбата му е изчислена с помощта на готовия статичен метод FindLookAtRotation. При загубване на здравето си, ще се стартира една от две възможни анимации на случаен принцип, показвайки успешно елиминиране на героя.

## **2.11. Противникът, изритващ вратите на кръчмата**

Това е леко изменен вариант на обикновения преследващ противник. Той излиза от кръчмата, минавайки през вратите чрез ритник. Той наследява GroundEnemy класа, даващ му достъп до всички променливи и методи. Към него единствено са добавени булева променлива за индикиране края на ритника и таймер. След приключването на анимацията, той ще започне да се държи като неговия родител без никакви допълнителни промени.

## **2.12. Противникът, показващ се от прозорците**

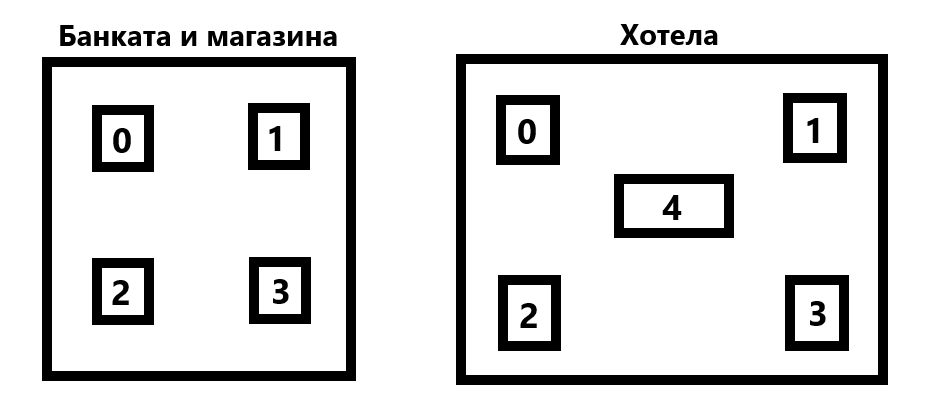
Другият главен съперник, срещу когото ще се изправи играчът, се показва от сградите, отваряйки прозорците за видимост или излизайки на терасата. Има възможност да се появи и зад бара в кръчмата, изчаквайки появата на играча. Той използва пушката като свое оръжие. Имплементиран е в WindowEnemy класа, наследяващ CharacerBase класа. Добавени са променливи за съдържането на класа на пушката и съответния прозорец, на който се показва героят. Методът LimitRotation ограничава завъртането му до някаква степен. Появява се точно зад отворения прозорец, винаги гледайки към главния герой. Това отново е постигнато чрез RotateToCharacter, използван и при другите противници. Той може да се покаже както на първия етаж на сградите, така и на втория. Появата му на горния етаж ще измени леко завъртането му по Pitch оста, правейки го леко наведен с цел подобряване на ефекта. Също като другите съперници, той стреля при всяка възможност като всеки един герой от този тип, който се покаже на нивото, ще има различна скорост на стрелба. Тя се избира на случаен принцип. При загубване на здравето си, ще се стартира една от две възможни анимации на случаен принцип, показвайки успешно елиминиране на героя.

## **2.13. Проектиране на анимациите на противниковите герои**

Реализирани са отново чрез използването на Animation Blueprint за всеки герой като за разлика от главния герой, не е създаден отделен клас. Стойностите на променливите се достъпват директно от графата за сриптиране в Blueprint-а. Превключването между анимациите е постигнато по същия начин, както при главния герой.

## **2.14. Проектиране на сградите**

За реализацията на всички видове сгради е използван абстрактният клас BuildingBase, наследявайки Actor класа. Той съдържа променливи за самите модели на сградите и компоненти за четирите прозорци, реализирани в отделения клас Windows. За улесняване на контрола над появата на противниците и отварянето и затварянето на прозорците в правилния момент е създадена структурата EnemyHandler. Тя съдържа прозореца и героя, който съответно ще се появи от него, както и местоположенията за противници, показващи се на терасата и в кръчмата. В класа също така имаме масив, запазващ всички дефинирани променливи на структурата. При започване на играта, в BeginPlay метода се дефинират пет променливи на структурата. Всяка една пази информация за всеки отделен прозорец от сградите и съответния противник. След това се добавят в масива. Единственият ръчно създаден метод е SpawnEnemy. Подава се към него аргумент на коя позиция да инстанцира героят. Видовете сгради в града с опасност от поява на враговете са банката, хотелът, магазинът и кръчмата. За всяка една е създаден отделен клас, наследяващ BuildingBase. Банката и магазинът се управляват по един и същ начин поради еднаквия брой на възможни съперници. Хотелът има и тераса, водейки до леко изменяне на SpawnEnemy в Hotel класа. Кръчмата единствено не използва BuildingBase като клас-родител, поради факта че няма прозорци. Героите могат да се появят от три места. Първото е зад бара, второто на терасата и третото зад вратите. Третото място е запазено едниствено за преследващия герой, изритващ вратите. Отново имаме масив, който ще съдържа този път три елемента на структурата.



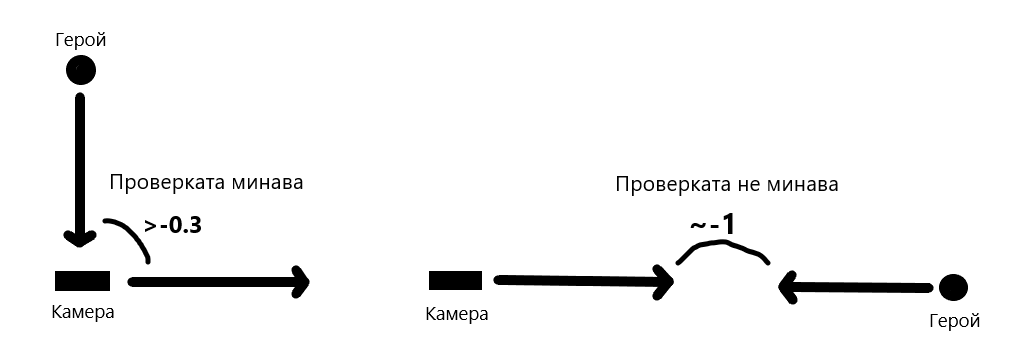
**Фиг. 2.6. Визуализация на подредбата на прозорците и терасата на BuildingBase сградите в масива**

### **2.15. Проектиране на прозорците**

Гореспоменатият клас Windows е създаден за контролиране на отварянето и затварянето на прозорците. В него имаме член-променливи за самите тела и методите Open и Close, обхващащи движението им . Това става чрез завъртане на двата модела по Yaw оста, докато не достигнат крайната стойност на завъртане. Имплементиран е и помощният метод GetIsClosed, който ще укаже дали са затворени прозорците. Той се извиква от сградите при инстанциране на противници.

## **2.16. Управление на нивото**

Управлението на нивото е реализирано в абстрактния LevelHandler. Той съдържа няколко компонента с форма на кутия, пойнтъри към главния герой и сградите. Също така са предефинирани позициите, на които могат да се появят преследващите противници. Методите обхващат инстанцирането на всеки един вид протвиник. Специализираните методи WEnemyHandler и GEnemyHandler преценяват на коя позиция да се покаже съответният герой, като голяма част се пада на случаен принцип. Поставена е невидима кутия малко по-напред на пътя в града. Когато главният герой я настъпи, ще се извика методът OnEnterBox, който ще инстанцира протвиника, излизащ от кръчмата с ритник. Две секунди след това явление, постигнато с таймери, ще се появи и съперник на втория етаж на банката, една секунда след което ще се покаже и протвиник някъде от хотела. Така се задава началото на играта. Преследващите герои могат да се появят до началното място, където играта започва, в обора и до водната кула. Всички врагове, които не се показват от прозорците, ще се появяват само когато ирачът не гледа към мястото, откъдето изскачат. Целта е да не се появяват в очите на играещия. За тази цел е създаден IsEnemyInSight методът, който изчислява дали в този момент противникът ще се появи в обсега на камерата на играча. Тази проверка ще се прави за преследващите съперници единствено ако главният герой се намира в прецизно поставени невидими кутии. Общо две са на брой. Изключение правят героите, показващи се в началото, и тези на терасите, поради факта че могат да бъдат забелязани почти отвсякъде.



**Фиг. 2.7. Визуализация на изчисленията, които се извършват в алгоритъма в IfEnemyInSight метода**

Ако се провали опит да се извика преследващ герой, методът SpawnGroundEnemy ще се извика рекурсивно, докато не се инстанцира успешно. Този метод ще избере на случаен принцип на което място да се покаже героят. Елиминирането на противник, показващ се от сградите, ще извика метода, управляващ този тип врагове. Тук също се изчаква в зависимост от настоящото здраве на главния герой.

## **2.17. Проектиране на потребителския интерфейс**

Менютата и HUD-а са реализирани чрез употребата на Widget Blueprint.

Той предоставя канвас, на който ще могат да се подредят и пресонализират различни компоненти, точно както разработчикът би искал да изглеждат на екрана. Използвани са бутони за избирането на различните опции, в които е вмъкнат текст. Логиката при натискането на бутоните с мишката е изградена в съседната графа. Лика на главното меню е създаден на отделно ниво. Изкарването на Widget-ите на екрана се извършва с извкиването на CreateWidget и AddToViewport в Level Blueprint, където се написва логиката специализирана единствено за даденото ниво. Всяко ниво си има отделен Level Blueprint. Натискането на Start Game ще отвори главното ниво, с помощта на функцията OpenLevel, откъдето започва играта. За превключването между различните нива на трудност също е използван бутон.

Натискането на Quit Game бутона извиква QuitGame функцията, спираща играта. Менюто за пауза е реализирано по същия начин както и главното меню, като то едниствено се изкарва по време на игра, чрез натискането на Escape клавиша. Тогава играта ще се замрази с помощта на задаването на Set Game Paused като ‚вярно‘. Натиснат ли е отново Escape, менюто ще се затвори. Continue Game бутона извършва същата функционалност. Натискането на Options ще покаже два плъзгача, чрез които ще може да се нагласят нивото на чувствителност на мишката и звука. Нивата им няма да се рестартират, докато играта не бъде затворена, независимо колко пъти се започва наново. Това е постигнато чрез създаването на класа MyProjectGameInstance, наследяващ GameInstance. В него са декларирани две променливи. Едната за нивото на звук и другата за мишката. В този клас стойностите на променливите ще се запазят, независимо от отвореното ниво. При пускането на играта могат да се забележат в долния ляв ъгъл две ленти. Голямата показва нивото на здравето на героя, а по-малката - нивото на специалната му способност. Когато вземе оръжие в ръката си, в противоположния ъгъл ще се покаже картинка на оръжието и в нея стойност, показваща оставащите патрони в пълнителя. В центъра на екрана седи една бяла точка, която изобразява мерника на героя. По този начин изглежда HUD-ът. За начертаването на бялата точка е използван специализиран метод, който се извиква постоянно в ръчно създадения GameHUD Blueprint. Той също така гарантира, че ще се изрисува в центъра на екрана, независимо от резолюцията на монитора на играещия. Към лентите са свързани променливите в класа на героя, държащи нужната информация. Достъпването им е имплементирано през графата му, като постоянно се правят проверки кое оръжие държи героят в дадения момент.

## **2.18. Проектиране на денонощния цикъл**

Осветяването на нивото се извършва от насочената светлина, добавена в нивото. Тя е подадена като ориентир в обекта на небесната сфера за позицинирането на слънцето и небесните цветове. За да може да се симулират залези и изгреви, осветлението, получено от насочения светлинен източник трябва да е динамично. Единствено то позволява светлинния източник да променя позицията си по време на изпълнение. Самото преместване се реализира чрез завъртането на обекта на насочената светлина по Pitch оста. Със снижаването на слънцето, цвета на небето ще става все по-тъмен и звездите все по-ярки. По принцип нивото винаги ще е осветено, независимо дали има поставени светлинни източници в нивото. В този случай, изискването е през нощта да е почти изцяло тъмно, освен ако не се поставят светлини. Този проблем се оправя чрез модификацията на EnvironmentIntensity и EnvironmentColor променливи в World Settings. В случая EnvironmentIntensity е променено на 0.25 от 1.0 и на EnvironmentColor е сложен тъмно-сив цвят. Независимо дали слънцето се е позиционирало под сцената, за да се премахне свтлината от насочения източник, трябва да се модифицира Intensity променливата. Последната стъпка, която трябва да се направи, с цел да се стъмни сцената е и промяната на стойността на Intensity променливата в SkyLight обекта. Това ще направи сенките изцяло тъмни.

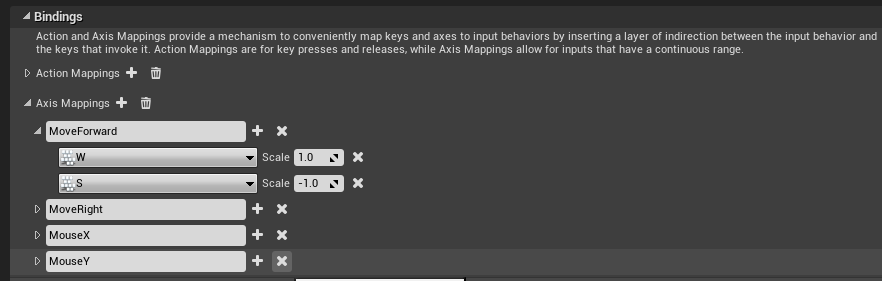
# **ТРЕТА ГЛАВА.**

# **ПРОГРАМНА РЕАЛИЗАЦИЯ НА ВИДЕО ИГРАТА**

## **3.1. Реализация на управлението на главния герой**

### **3.1.1. Настройване на клавишите за контрол**

В настройките на самия проект се настройват клавишите, с които ще се упарвлява главния герой, чрез създаването на Action и Axis mapping-и, които ще следят за натискането на избраните клавиши. Axis mapping-ите се използват за непрестанни действия, когато е нужно да знаем състоянието на клавиша във всеки един момент, като движението на героя, камерата и др. Те се проверяват постоянно и връщат стойност според това дали клавишът е натиснат или не. Един от тях е MoveForward, следящ за движението на героя напред и назад. Затова в него са избрани клавишите W и S. При натискане на W се връща числото с плаваща запетая 1.0,, S връща -1.0, и ако нито един от двата клавиша не е натиснат в дадения момент, се връща 0.0.



**Фиг. 3.1. Вид на настройките на клавишите**

Action mapping-ите се използват, когато не е нужно да знаем състоянието на клавиша във всеки един момент. Такива примери са стрелянето, смяната на оръжие и др. В MyCharacter класа е автоматично имплементиран SetupPlayerInput методът, специализиран за обвързването на mapping-ите с ръчно създадени методи, който да управлява техните функционалности чрез Input компонент, следящ техните въвеждания. Това обвързване става чрез извикването на BindAxis и BindAction функциите на дадения компонент. BindAxis изисква в параметрите си името, зададено от нас на самия mapping пойнтър към обекта, който ще го използва, и ръчно създаден метод, който ще съдържа неговата функционалност. Този метод трябва да има един единствен параметър, приемащ число с плаваща запетая. Стойността, която върне mapping-ът, ще му бъде подадена. BindAction има допълнителен параметър, изискващ състоянието на клавиша, при което ще се извика методът. Използвал съм IE\_Pressed и IE\_Released, които са съответно за натискане и отпускане на клавиша.



**Фиг. 3.2. Обвързване на клавишите в SetupPlayerInput метода**

### **3.1.2. Реализация на движението на героя**

MoveForward и MoveRight методите управляват преместването на обекта на MyCharacter. MoveForward управлява движението напред и назад, а MoveRight - наляво и надясно.



**Фиг. 3.3. Декларация на методите за движение**

Когато се извика MoveForward и Input параметърът му е различен от 0.0, т.е един от двата клавиша е натиснат, се декларира и дефинира векторът SpringArmForward. Той намира посоката, в която гледа пружиненият лост, чрез извикването на GetForwardVector функцията.



**Фиг. 3.4. Намиране на посоката, в която гледа камерата**

За самото преместване на героя е използвана AddMovementInput функцията. Като нейни параметри се подават SpringArmForward, обозначаващ посоката, в която да се движи и Input, индикиращ дали ще е в обратната посока, водейки до придвижване назад. AddMovementInput работи единствено с компонент за движение, който в този случай е CharacterMovement. Той симулира отделно физиките на героя, даващ възможност не само за ходене и тичане, но и за плуване и летене. Скоростта, с която ще се придвижва героят се определя от MaxWalkSpeed променливата в CharacterMovement компонента. Също така са имплементирани гравитация и независимост от кадрите в секунда. Необходима е, за да се извършва преместването на еднакво разстояние, независимо от броя на кадрите, с които се играе всяка секунда.



**Фиг. 3.5. Имплементация на MoveForward метода**

MoveRight методът е аналогичен на MoveForward, като единствената разлика е че вместо SpringArmForward имаме SpringArmRight, който намира посоката вдясно от посоката, в която гледа пружинения лост. Бъде ли Input отрицателно, придвижването ще се извърши наляво.

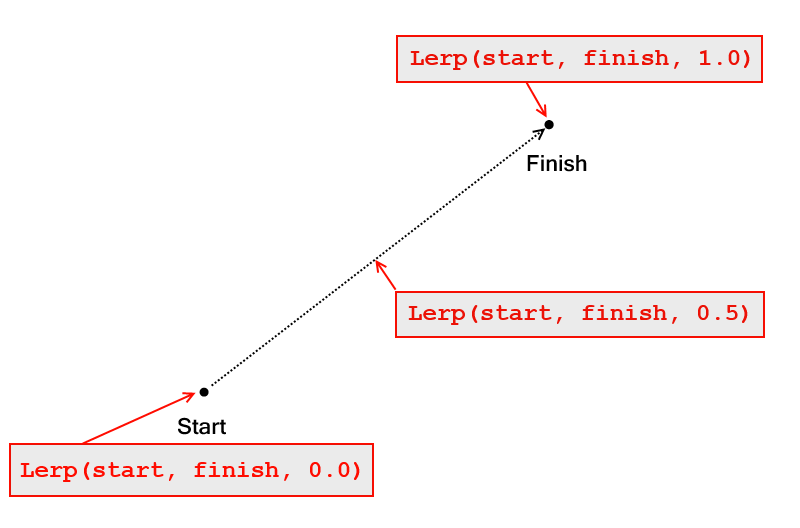
### ***3.1.2.1. Реализация на завъртането на героя към камерата***

За тази цел е имплементиран LerpPlayerToCamera, който променя ротацията на тялото на героя, така че да е горе-долу равна с тази на пружинения лост.

LerpPlayerToCamera приема един параметър Speed от тип число с плаваща запетая, определящ скоростта, с която ще се завърта единствено тялото на обекта. В тялото на метода се декларират две променливи от тип число с плаваща запетая. Първата е CurrYawRot, намираща градуса на завъртане на тялото на героя по Yaw оста. Втората е NewYawRot, която намира градуса на завъртане на пружинения лост по Yaw оста. Тези стойности важат за ротациите им в локалното 3D пространство на класа, а не на света.

След това се декларират и дефинират още две променливи от тип ротатор. Първата е CurrRot, на която като по Yaw оста се подава CurrYawRot, а другите оси остават 0.0. Другата е NewRot, на която като по Yaw оста се подава NewYawRot, а другите оси също остават непроменени със стойност 0.0.

Завъртането на тялото на героя става чрез извикването на SetRelativeRotation метода, който променя ротацията му в локалното пространство. Като параметър му се подава нова ротация, изчислена от Lerp метода. Той приема начална, крайна ротация и число с плаваща запетая, индикиращо каква част от началото до края да бъде измината. Минималната стойност е 0.0 и максималната е 1.0. Ако му бъде подадена стойност 0.5, ще върне ротация точно по средата между началната и крайната.



**Фиг. 3.6. Визуализация на логиката в Lerp метода**

В този случай към Lerp са подадени CurrRot за начална ротация, NewLoc за крайна и Speed, умножено по изминатото време между всеки кадър, взето чрез GetDeltaSeconds функцията. GetDeltaSeconds се достъпва от World обекта, който се достъпва от GetWorld метода. Отново е желателна функционалност, независима от кадрите в секунда.



**Фиг. 3.7. Имплементация на LerpPlayerToCamera метода**

LerpPlayerToCamera се извиква в MoveForward и MoveRight с цел да се изпълнява, само когато героят се движи.

### **3.1.3. Реализация на движението на камерата около героя**

Въртенето на камерата се управлява в LookSide и LookUp методите. Преместването на мишката по абсцисата вдясно ще върне 1.0 и вляво -1.0, а преместването по ордината нагоре -1.0 и надолу 1.0. В LookSide, когато Input не е 0.0, към текущата ротация по Yaw оста на пружинения лост се добавя число с плаваща запетая, получено от умножението на Input и LookSpeed, управляващо скоростта. AddRelativeRotation е използван за тази цел.



**Фиг. 3.8. Имплементация на LookSide метода**

LookUp метода е изграден по идентичен начин, с разликата че се добавя число към ротацията на лоста по Pitch оста, за да се завърта над героя. Това действие е ограничено чрез проверка преди самото извикване на AddRelativeRotation.



**Фиг. 3.9. Имплементация на LookUp метода**

### **3.1.4. Реализация на регенерацията на кръв**

Регенерацията на кръвта на героя се извършва в Tick метода като към текущата му кръв се прибавя умножението между числото с плаваща запетая HealthRegenSpeed, управляващо скоростта на регенерация, и времето, изминато между всеки кадър DeltaTime, докато не дотигне максималната си кръв.



**Фиг. 3.10. Реализацията в Tick метода**

### **3.1.5. Реализация на мащабирането на камерата**

Приближаването и отдалечаването на камерата е реализирано в CameraZoom и CameraOutZoom функциите. При задържане на десния бутон на мишката ще се извика CameraZoom, а и при отпускане ще се извика другият метод. В него ще се провери дали героят е с оръжие в ръка. Ако не е, няма да се извърши преместването. То става чрез първо скъсяването на самия пружинен лост. За тази цел е използван FInterpTo, изискващ начална, крайна стойности от тип число с плаваща запетая и скорост на интерполация. Модифицирана е TargetArmLength променливата на лоста. FInterpTo забавя движението при достигане на крайната цел. Освен това е модифицирана височината на мястото, където е закачена камерата и позицията на камерата. За тази цел е използван VInterpTo, който е съвместим с векторни стойности. Тези методи сами се справят с независимостта от кадрите в секунда. CameraOutZoom връща началните им стойности. Тези методи се извикват в Tick за плавната интерполация.



**Фиг. 3.11. Имплементация на CameraZoom метода**

### **3.1.6. Реализация на стрелбата и смяната на оръжие**

Стрелбата с оръжията е реализирано в Fire метода. Той се извиква при натискане на левия бутон на мишката. В тялото му се прави проверка с кое оъжие героят борави в дадения момент. Извиква се SpawnProjectile методът от класа на оръжието. Скоростта на стреляне се управлява с помощта на таймер. Той се задава чрез извикването на SetTimer от мениджъра на таймери. В параметрите му се подават Handle, отговарящ за таймера, пойнтър към обекта, който го използва, метод и число с плаваща запетая, задаващо след колко секунди ще се извика съответният метод. В този случай методът се казва ResetPistolFire и ще промени булевата променлива bCanFirePistol, спираща героя от изстрелване на куршум, докато не изтече таймерът. Има възможност и да се повтаря таймерът, което не е нужно в този случай.



**Фиг. 3.12. Имплементация на Fire метода**

Смяната на оръжие се извършва в ChangeToPistol и ChangeToRifle методите. При наличието на едно от тези оръжия и когато не се е прицелил играчът, обектът на оръжието ще се закачи за гнездо, създадено в скелета на тялото на героя чрез AttachToComponent метода. Този метод изисква в параметрите си пойнтър към компонента на самото тяло, правило при закачане и името на гнездото. Смененото оръжие ще се закачи за началното гнездо.



**Фиг. 3.13. Имплементация на ChangeToPistol метода**

### **3.1.7. Реализация на забавянето на времето**

Забавянето на времето е реализирано в EnterSlowMo метода. Използвана е специализираната функция SetGlobalTimeDilation, контролираща времеразделянето. Към него се подава число с плаваща запетая, като нормалната скорост по подразбиране е 1.0. За да се забави времето, трябва да се намали неговата стойност. В този случай е подадено 0.4, забавяйки времето 2.5 пъти. Забавянето обхваща всичко, свързано с време, като DeltaTime и таймерите. В тялото на функцията се извиква SetGlobalTimeDilation и всички времезасягащи променливи в MyCharacter се настройват, така че да не се засегне движението и стрелбата на главния герой. Ако вече е активирана способността, се извършва обратното.



**Фиг. 3.14. Реализацията на активирането на SlowMo ефекта**

### **3.1.8. Реализация на приближаването на камерата към свален противник**

В MyCharacter класа е дефиниран метода ZoomedKills, който има един параметър от тип число с плаваща запетая. Той задава дистанцията, на която се намира убития съперник от главния герой. Времето се забавя по същия начин, както и в EnterSlowMo метода, като този път забавянето е по-мащабно. За да не се хващат в рамките на камерата главния герой и оръжията, те биват скрити. За целта се използва SetActorHiddenInGame функцията, предоставена ни от Actor класа. FOV променливата на камерата може да бъде настроена с минимална стойност от 5 и максимална от 170. Колкото по-малка е стойността, толкова повече се приближава камерата. За генерирането на FOV, съвместимо с дистанцията до героя, е използвана формулата:



**Фиг. 3.15. Реализацията на генерирането на ново FOV**

Докато е активиран ефектът, игачът няма да може да управлява героя, с цел да не размести камерата. DisableInput функцията е използвана като към нея е подаден контролера, управляващ героя, използвайки GetPlayerController функцията. Промяната на FOV се извършва чрез извикването на SetFieldOfView метода от класа на камерата. Използван е таймер за настройването на времетраенето на ефекта. След като свърши таймера, се извиква ResetZoomedKills метода, който ще нулира всички промени, направени в ZoomedKills.

## **3.2. Реализация на куршумите**

Декларираният компонент ProjectileMovement се захваща за самото изстрелване на куршума в 3D пространството. Също така симулира и физики към тялото му, без които колизиите на обекта няма да бъдат засечени. Нагласяват се InitialSpeed, MaxSpeed и Gravity променливите на компонента, за скоростта му на придвижване и дали да е наличен ефектът на гравитацията.

В този случай наличието на гравитация е ненужно. Имплементиран е методът OnHit, в който се извършва логиката при уцелване на герой. В BeginPlay е зададено да се извика OnHit, когато някакъв обект прекрачи сферата, заобикаляща тялото на куршума.



**Фиг. 3.15. Дефиниране на събитието, извикващо OnHit метода**

В параметрите на функцията ще бъдат запаметени данни от сорта на прекрачилия обект и компонент, точка на прекрачване, уцелена кост и др. В тялото й се прави проверка, ако главния герой е засякъл куршума, изстрелян от него. В такъв случай той ще се унищожи. Също така се проверява дали противников герой е уцелил свой съюзник. Тогава отново ще се унищожи. При уцелване на скелетово тяло, уточнено чрез използването на Hit структурата от параметрите, ще се провери дали е уцелен черепът, за да се удвои нанесената щета. Ще се извика и SpawnEmitterAtLocation, за да се пусне партикъл ефектът, наподобяващ плискаща кръв. След това, ако е свалена цялата кръв на героя, той ще се унищожи и ще се стартира анимация, показваща падане на героя на земята. Ако е елиминиран противников герой, физиките на оръжията ще се симулират, за да паднат на земята. Това става чрез извикването на SetSimulatePhysics на тялото на оръжието.



**Фиг. 3.16. Част от имплементацията на уцелване на герой в OnHit метода**

## **3.3. Реализация на оръжията**

### **3.3.1. Реализация на вземането на оръжие**

В GunBase класа е деклариран OnEnterSphere методът, в който ще се извърши главната функционалност. В BeginPlay е зададено да се извика OnEnterSphere, когато някакъв обект прекрачи сферата, заобикаляща телата на оръжията. В параметрите на функцията ще бъдат запаметени данни от сорта на прекрачилия обект и компонент, точка на прекрачване и др.



**Фиг. 3.17. Дефиниране на събитието, извикващо OnEnterSphere метода**

Когато главният герой влезе в сферата, ще се викне OnEnterSphere, в който се прави проверка дали героят вече не се е сдобил с такова оръжие. Ако все още не е, обектът на оръжието ще се закачи за предназначеното гнездо на тялото му с помощта на AttachToComponent. Ако вече има такова оръжие, ще се направи проверка дали е запълнен капацитетът от амуниции. Ако не, ще вземе оръжието и ще допълни дадения капацитет.



**Фиг. 3.18. Част от имплементацията на OnEnterSphere метода**

### **3.3.2. Реализация на изстрелването на куршум**

В SpawnProjectile метода се извършва инстанцирането на куршумите чрез използването на SpawnActor функцията. В тялото се декларират векторите SpawnLocation и SpawnRotation. На SpawnLocation отначало е зададена позицията на ръчно създадено гнездо на тялото на оръжието, която се взима с помощта на GetSocketLocation функцията. Това гнездо е на върха на дулото. Тя се взима чрез GetSocketLocation. Тя приема името на гнездото, дадено от потребителя. SpawnRotation ще бъде зададена впоследствие. След това се проверява от какъв тип герой се прави опит за инстанция на куршум. Ако главният герой го изстрелва, се генерира цялото число ChanceToHit на случаен принцип между 1 и 100 с помощта на RandRange функцията. След това се проверява дали героят се е прицелил като се извиква GetZooming от MyCharacter. В този случай се проверява дали генерираното число е по-голямо или равно от 30, правейки шанса да се даде ротация, така че да е в целта, 30%. Тази ротация се изчислява в GetHitRot метода, имплементиран в GunBase. Параметрите й изискват местоположението на гнездото и пойнтър към обекта на главния герой. В тялото се декларират структурите HitResult и CollisionParams. Също така се декларира и дефинира вектора EndLoc, който се получава по следния алгоритъм:



**Фиг. 3.19. Изчисляване на крайната позиция EndLoc**

След това се изстрелва лъч, с помощта на LineTraceSingleByChannel функцията. Към параметрите й се подават HitResult, който ще запази информация за уцеления от лъча обект, начална позиция, която е местоположението на камерата, EndLoc, каналът на колизиите и CollisionParams. Каналът на колизиите се избира между Camera и Visibility. В този случай е Camera, защото ще спре лъча на първата уцелена повърхност. Ако бъде уцелен обект до крайната точка на лъча, функцията ще върне ротация, изчислена чрез FindLookAtRotation метода. Той има два параметъра. Единият е локацията на гнездото, а другият е позицията на пресичане на лъча, която се взима от HitResult. Ще се получи ротация, гледаща към точката на уцелване.



**Фиг. 3.20. Имплементация на GetHitRot метода**

Тази ротация се подава на SpawnRotation в GetHitRot. Ако ChanceToHit се окаже по-голямо от 30, към Pitch и Yaw осите на SpawnRotation ще бъде прибавено случайно избрано число с плаваща запетая, реализирайки отклонение. Ако играчът не се е прицелил, шансът е 15% и отклонението ще бъде по-голямо, имплементирано по същия начин. При изстрелването на куршум от противник, SpawnRotation ще бъде изчислена по следния начин:



**Фиг. 3.21. Намиране на ротацията на куршума, ако бъде изстрелян от противников герой**

Отклонението отново е имплементирано по същия начин. SpawnActor приема пойнтър към класа на куршума, SpawnLocation, SpawnRotation и структура, определяща правилата при инстанциране. Използвани са таймери за забавянето на партикъл ефектите, когато играчът не се е прицелил. Логиката е реализирана изцяло за визуални цели. Ефектите се стартират в SpawnEmitter функцията, използвайки SpawnEmitterAtLocation. На същото място се стартират и звукови ефекти. Употребява се PlaySound2D, когато главният герой изстрелва куршумите и PlaySoundAtLocation, когато противниците стрелят. PlaySoundAtLocation стартира звуковия ефект на определно място в 3D пространството.



**Фиг. 3.22. Имплементация на пускането на звукови ефекти в SpawnEmmiter метода**



**Фиг. 3.23. Част от имплементацията на инстанцирането на куршум, извършено от главния герой.**

## **3.4. Реализация на противниците**

### **3.4.1. Реализация на протвиника, преследващ главния герой**

#### **3.4.1.1. Реализация на движението на героя**

В MoveForward е реализирано неговото придвижване в нивото. Ще се движи напред, докато не достигне определено пространство между него и главния герой. GetDistanceToMain се заема с тази задача. В тялото й се декларира и дефинира DistanceVector векторът с изваждането на позицията на този герой и на главния герой. Взима се дължината му чрез извикването на Size метода на вектора и тази стойност се връща под тип число с плаваща запетая.



**Фиг. 3.24. Имплементация на GetDistanceToMain метода**

Самото придвижване напред е извършено с помощта на AddActorWorldOffset функцията. Тя добавя вектор към текущата позиция на героя в глобалното пространство. Векторът, който ще се подаде на AddActorWorldOffset се изчислява по следния начин:



Чрез извикването на MaxWalkSpeed от CharacterMovement се взима скоростта на движение. За да променя посоката си на движение, се изиква Rotate методът, към който се подава число с плаваща запетая, определящо посоката. При 1.0 завива наляво, при -1.0 - надясно и при 0.0 - продължава напред. Ако е подадено 0.0, завъртането се изпълнява с извикването на RotateToCharacter. В нейното тяло се подава ротация към SetActorRotation. Тя се изчислява с RInterpConstTo. Към нея се подава текущата ротация на обекта и крайната. Текущата е взета чрез GetActorRotation, а крайната - от LookAtChar функцията. LookAtChar връща нова ротация, получена отново чрез използването на FindLookAtRotation. Това е ротацията, с която обектът на противниковия герой ще гледа към главния герой.



**Фиг. 3.25. Имплементация на LookAtChar метода**

Ако е подадено 1.0 или -1.0, завъртането се реализира с помощта на AddActorWorldRotation. Тази функция добавя число към текущата ротация. В този случай това число се добавя към Yaw оста. bIsRotating булевата променлива проверява кога героят се завърта на едно място. Тя се използва за управлението на анимацята за стационарно завъртане.



**Фиг. 3.26. Имплементация на Rotate метода**

В MoveForward към Rotate е подаден LineTrace методът. Той определя завъртането на обекта, когато трябва да се заобиколи някакъв обект на пътя. В тялото й се декларират 3-те структири OutHitFront, OutHitFrontL, OutHitFrontR. По-късно ще им бъдат зададени стойности. След това се създават начална и крайна позиция на три лъча, правейки общо 6 вектора. Първият лъч започва от местоположението на обекта на героя и завършва с 250 пункта по-напред. Вторият е със същата дължина като стартовата позиция е 50 пункта вляво. Последният лъч е с 50 пункта вдясно. Изстрелването на лъчите се извършва с помощта на LineTraceSingleByChannel. Подават се горесъздадените структури и вектори. Проверява се дали някой от тях е пресечен. Ако е се изчислява кой от двата странични вектора е с по-голяма дължина, ще върне 1.0, когато левият е по-дълъг, -1.0 - ако десният е по-дълъг и 0.0 - ако не е пресечен нито един от лъчите.

#### 

**Фиг. 3.27. Изстрелването на лъчите и определянето на тяхната дължина в LineTrace метода**

#### **3.4.1.2. Реализация на стрелбата на героя**

Стрелбата на героя е имплементирана подобно на MyCharacter. Инстанцирането на куршумите се управлява като се извиква SpawnProjectile от класа на оръжието. Скоростта на стреляне отново е реализирана чрез таймер. В BeginPlay се инстанцира револвер, използвайки SpawnActor. Така е дадено оръжието на героя. В Tick метода се извиква MoveForward и Fire, докато не му бъде свалена кръвта.



**Фиг. 3.28. Реализация на Fire метода в GroundEnemy класа**

#### **3.4.1.3. Реализация на унищожаването на героя**

Унищожаването на обекта се извършва в DestroyAfterTime и DestroyChar. Първо се извиква DestroyAfterTime в Projectile класа, използваща таймер, с който да извика DestroyChar след 3 секунди. Целта на този метод е да унищожи обекта на героя, след като е свършила анимацията, показваща неговата елиминация. За да се стартира анимацията в Projectile, също така се извиква и PlayEnemyDeathAnim, имплементирана в CharacterBase класа. В DestroyChar се унищожава самият обект чрез Destroy функцията.

### **3.4.2. Реализация на протвиника, показващ се от прозорците**

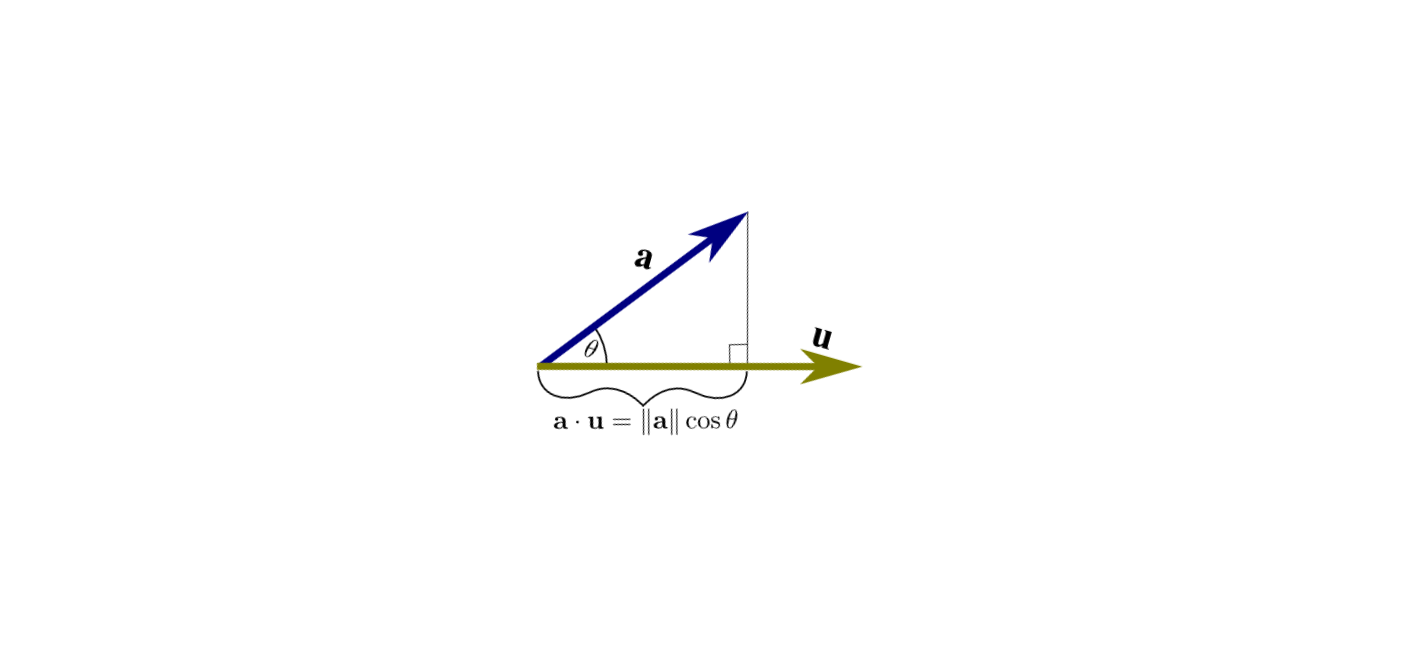
#### **3.4.2.1 Реализация на завъртането на героя**

В Tick функцията се извиква RotateToCharacter, за да се обръща към главния герой. Това завъртна е ще бъде ограничено до някаква степен, за да бъде видим през цялото време. Ограничаването е имплементирано в LimitRotation метода. В тялото й са декларирани двата вектора CharForwardVector и BuildingActorForward. CharForwardVector е дфиниран с нов вектор, получен чрез извикването на GetForwardVector, приемащ LookAtChar. GetForwardVector изчислява посоката, в която гледа един обект, базирано на неговата ротация.



**Фиг. 3.29. Дефиницията на двата вектора в LimitRotation метода**

След това се прави проверка в коя сграда е инстанциран героят. На тази сграда се взима посоката, в която гледа и BuildingActorForward я получава. Употребена е DotProduct функцията за да се изчисли косинусът на ъгъла между двата гореспоменати вектора.



**Фиг. 3.30. Визуализация на логиката в DotProduct метода**

Прецежил съм, че най-добре се получава ако полученото число, зададено на Angle променливата, е по-голямо от 0.5.



**Фиг. 3.31. Реализацията на LimitRotation метода**

Ако героят е инстанциран на прозорец, се извиква зададената стойност към WindowsPlace променливата, индикираща на кой прозорец се е появил. Ако е на един от прозорците на горните етажи, ще се промени ротацията му по Pitch оста с по-голямо отклонение. Тя се изменя с помощта на SetActorLocation функцията, като аргумент към нея е подадена новосъздадена ротация със зададено завъртане по Pitch желания наклон, под който да се надвеси героят. По Yaw е текущата му ротация и Roll остава непорменена.



**Фиг. 3.32. Имплементация на RotateToCharacter метода в WindowEnemy класа**

#### **3.4.2.2 Реализация на стрелбата на героя**

Стрелбата на героя е имплементирана сходно на другия герой. Инстанцирането на куршумите се управлява като се извиква SpawnProjectile. Скоростта на стреляне отново е реализирана чрез таймер. В BeginPlay се инстанцира револвер, използвайки SpawnActor. Така е дадено оръжието на героя. В Tick метода се извиква Fire и RotateToCharacter, докато не му бъде свалена кръвта.

## **3.5. Реализация на прозорците**

В Tick метода на Windows класа се проверяват стойностите на две булеви променливи. Едната е bIsOpening, обозначаваща дали прозорците трябва да се отворят, а другата е bIsClosing, обозначаваща дали прозорците трябва да се затворят. Ако променливите отговарят на „вярно“, съответно се извикват методите Open и Close. В Open се прави проверка дали прозорците не са се почти затворили, чрез IsNearlyEqual функцията. Към нея се подават три стойности. В този случай първата е ъгъла на завъртане по Yaw оста на единия прозорец. Втората е числото, с което се сравнява, и третата е допустимата разлика между числата. Завъртането на телата се извършва с извикването на SetRelativeRotation. Към нея с подава RInterpConstTo, преместваща плавно от текущата им ротация до целяната. Close методът е реализиран идентично на Open.



**Фиг. 3.33. Имплементацията на Open метода**

Имплементирана е функцията GetIsClosed, уточняваща дали прозорците са затворени. Отново е използван IsNearlyEqual методът.

## **3.6. Реализация на сградите**

В BuildingBase са декларирани 4 компонента, които ще бъдат назначени с класа на прозорците Windows и ще се създадат подкласови обекти. Декларирана е и структурата FEnemyHandler с променливи WindowsActor за самия обект на прозорците, EnemyActor за един от всички видове противници и векторите TerraceLoc, InLoc, DoorLoc, OutLoc. Създават се в PostInitializeComponents метода 5 стуктурни променливи на гореспоменатата структура. На първите 4 от тях се задава WindowsActor пойнтърът с обектите на прозорците. На петия се задава единствено TerraceLoc в локалното пространство на класа. SpawnEnemy методът извършва самото инстанциране на противниците. В тялото на функцията се проверява дали вече е окупирано мястото и дали прозоците са затворени. Ако мине проверката, се извиква Open на съответния прозорец и се декларират двата вектора LocOffset и RotOffset, обозначаващи отклонението от центъра на прозореца и ротацията. Инстанцирането е постигнато с помощта на SpawnActor функцията. Към параметрите й се подават класът на героя, местоположението на прозорците, събрано с LocOffset, ротацията на прозорците, събрана с RotOffset и правила при инстанциране. В този случай правилата са нагласени с AlwaysSpawn, т. е. ще се инстанцира обектът във всеки случай, дори и да няма място за него в пространството. В класа SaloonBuilding функцията SpawnEnemy е override-ната, поради факта че кръчмата няма прозорци. В тялото й се проверява на кое от трите места се прави опит за инстанция на героя.



**Фиг. 3.34. Част от имплементацията на SpawnEnemy в класа SaloonBuilding**

## **3.7. Реализация на управлението на нивото.**

Декларирани са трите компонента за засичане на колизии с форма на кутия BoxCollision, BoxCollision и BoxCollisionBarn. BoxCollision се използва за задаване началото на управлението на нивото. Когато главният герой я прекрачи, OnEnterBox ще се извика. Ще се влезе само веднъж в тялото й. Ще се появят поне четирима противникови герои в рамките на 7 секунди, в зависимот от това накъде играчът е преместил камерата. Първо се извиква SpawnEnemy от SaloonBuildingActor, който е пойнтър към близката до началото кръчма на нивото, с подаден аргумент 2. Тогава герой от преследващ тип ще излезе от кръчмата с ритник. С помощта на таймери ще се появят другите герои. Преследващите герои се показват от едно от трите места, където играчът няма да види тяхната поява. За тази цел е имплементиран методът IfEnemyInSight с един параметър - векторът Loc. В тялото се декларират двата вектора CameraForwardVector и EnemyForwardVector. За първия се извиква GetForwardVector от камерата на главния герой. За втория вектор се намира ротацията, гледаща към главния герой, от Loc местоположението. Отново е употебена FindLookAtRotation функцията. Получената ротация се подава на GetForwardVector метода, генериращ нов вектор, с изглед към главния герой. CameraForwardVector и EnemyForwardVector се подават към DotProduct функцията, връщащ косинуса на ъгъла между двата вектора. Преценил съм, че ще е най-добре ако полученото число от тип плаваща запетая е по-голямо от -0.3. В такъв случай, героят ще може да се инстанцира.



**Фиг. 3.35. Имплементация на IfEnemyInSight в LevelHandler класа**

За да може да се подаде SpawnBankEnemy и SpawnHotelEnemy с аргумент към SetTimer, са декларирани делегатните променливи BankEnemyDel, HotelEnemyDel и GEnemyDel, избирайки местоположенията на случаен принцип, чрез RandRange функцията. Съперникът от банката ще се появи след 2 секунди, този от хотела след 3 секунди и преследващият след 7.



**Фиг. 3.36. Задаване на делегатните променливи към таймерите**

Инстанцирането на героите от терасите се извършва чрез повтарящ се таймер. Декларирани са глобални делегати, които се подават към SetTimer. Таймерите ще се повтарят през 20 секунди. Четвъртият параметър на SetTimer изисква булева променлива, индикираща дали да се повтаря таймера. В този случай е зададена стойността „вярно“.



**Фиг. 3.37. Повтарящи се таймери, опитващи се да инстанцират противници на всички тераси на нивото.**

WEnemyHandler управлява инстанцирането на героите по сградите. На случаен принцип избира някоя от сградите и позицията, от която да се появи, използвайки RandRange функцията. Отново са използвани таймери, като 5 секунди по-късно ще се извика методът Spawn…Enemy, специализиран за дадения тип сграда, ако главния герой е с пoне 30% от кръвта си. В противен случай ще изчака 10 секунди. Функцията се извиква, когато противник от този тип бъде елиминиран.



**Фиг. 3.38. Имплементация на част от WEnemyHandler**

GEnemyHandler методът е реализиран подобно на WEnemyHandler, но управлява преследващите герои. Избира местоположението и извиква SpawnGroundEnemyчрез таймер. Отново се проверява нивото на кръвта на главния герой. В SpawnGroundEnemy ако избраната позиция е Start, ще се извика IfEnemyInSight, поради факта, че ще може да бъде забелязана неговата поява от всяка точка на нивото. При другите позиции влизат в употреба и споменатите в началото кутии за колизии BoxCollisionBarn и BoxCollisionEnd. Ако избраната позиция е Barn и героят не я е прекрачил, противникът ще се появи директно, без да се извиква IfEnemyInSight. Проверката за прекрачване се прави с помощта на IsOverlappingActor функцията, която се извиква от самия компонент за колизии. Ако пък главният герой я е прекрачил, ще се извика и IfEnemyInSight. При неуспешен опит да се инстанцира герой на едно от трите възможни места, SpawnGroundEnemy ще се извика рекурсивно.

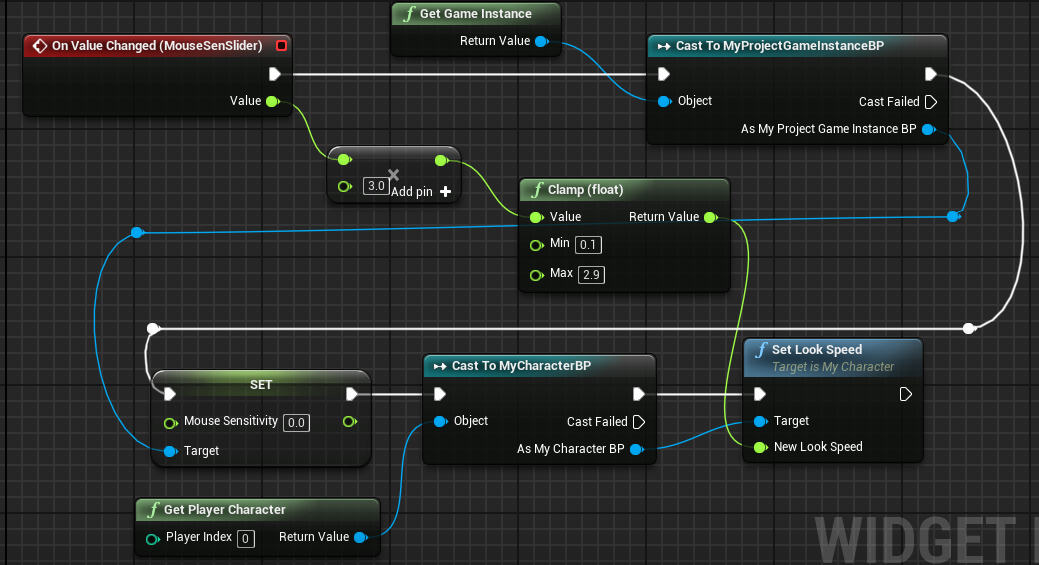


**Фиг. 3.39. Част от имплементацията на SpawnGroundEnemy**

## **3.8. Реализация на потребителския интерфейс**

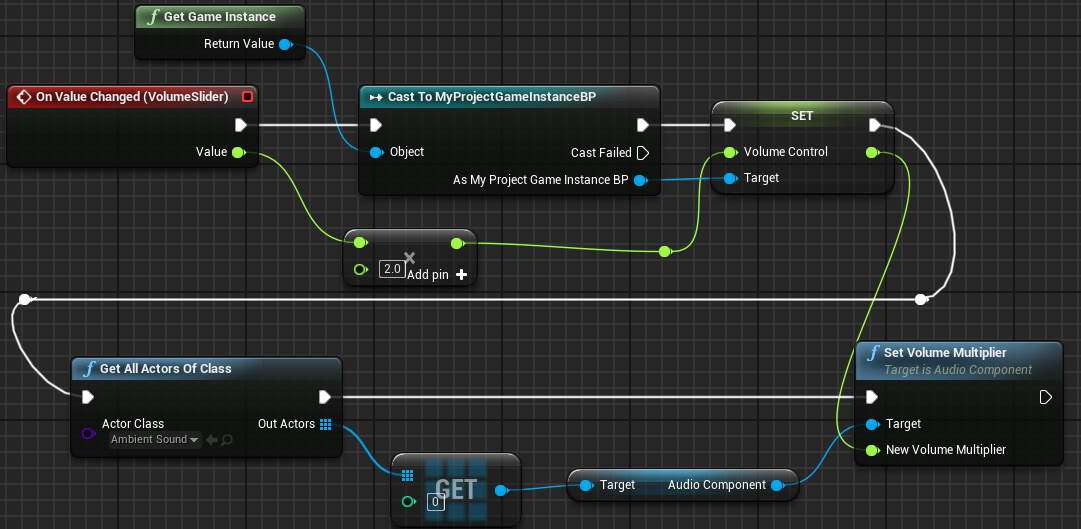
### **3.8.1. Реализация на настройването на звука и чувствителността на мишкатa**

Създаден е класът MyProjectGameInstance, наследяващ GameInstance, където са декларирани променливи, които да съхраняват стойностите, подадени от потребителя, през цялото игрово време. Това са променливите MouseSensitivity и VolumeControl. И двете са от тип число с плаваща запетая. Те ще бъдат променени, когато играчът премести плъзгачите в менюто. В графата на Widget Blueprint-а е реализирана логиката. При промяна в плъзгача за мишката, ще се извика събитието OnValueChanged, връщащо новата стойност Value, след което ще се достъпи инсъанцията на MyProjectGameInstance чрез GetGameInstance функцията. Ще се достъпи MouseSensitivty от MyProjectGameInstance, която ще бъде приравнена с Value. Така бива запаметена стойността и за в бъдеще. За да бъде променена самата чувствителност на мишката, трябва да бъде актуализирана LookSpeed в MyCharacter класа. Затова се използва GetPlayerCharacter функцията за да се достъпи героят, от който се извиква SetLookSpeed метода, на който се подава Value.



**Фиг. 3.40. Имплементацията на промяната на чувствителността на мишката**

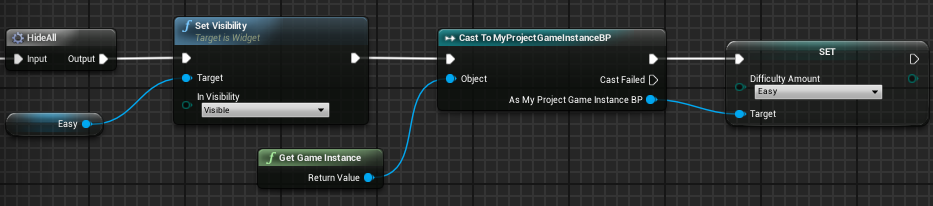
Преместването на плъзгача на звука също ще активира OnValueChanged, отново връщащо Value. По същия начин ще се модифицира VolumeControl променливата в MyProjectGameInstance. За да бъде променен и фоновият звук, който се стартира с пускането на нивото, трябва да се достъпи неговия обект, поставен в нивото. За тази цел е употребена GetAllActorsOfClass функцията. Тя приема клас, обозначаващ типа на обектите, които да търси. Ще върне масив с всички намерени обекти в него. В този случай е подаден класът AmbientSound. Тъй като е известно, че има само един обект от този тип в нивото, се достъпва нулевия елемнт на масива. За да се модифицира силата на звука му, се извиква SetVolumeMultiplier функцията от звуковия му компонент. Към нея се подава Value.



**Фиг. 3.41. Имплементацията на промяната на звука**

### **3.8.2. Реализация на настройването на трудността**

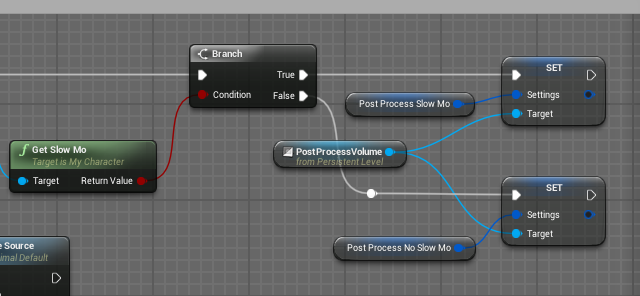
В MyProjectInstance е декларирана променливата GameDifficulty от тип енумерация. Има три стойности: Easy, Medium и Hard, които са и съответно нивата на трудност. В канваса на Widget-а на главното меню е добавен бутон с вид на стрелка. При неговото натискане се сменя текста, обозначаващ трудността, като започва от Easy. Чрез ConfigureEasy, ConfigureMedium, ConfigureHard и HideAll макросите е реализирана транзицията между трите текста. Използвана е SetVisibility функцията за тяхното скриване и показване. При показването на всеки един, се достъпва MyProjectGameInstance, отново с помощта на GetGameInstance метода. Достъпва се DifficultyAmount променливата и се приравнява със стойността, показана на менюто.



**Фиг. 3.42. Имплементацията на промяната на трудността**

### **3.8.3. Реализация на настройването на глобалния цвят на камерата**

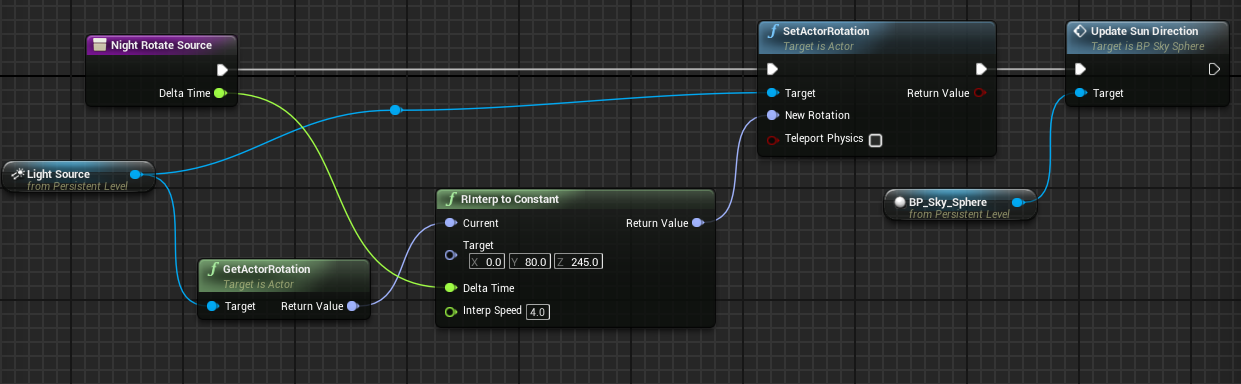
В нивото е добавен PostProcessVolume, който служи за настройването на всякакви ефекти от типа на цветово преливане, особености на рендериране и други. В този случай се използва за индикатор, че е активиран ефектът за забавяне на времето. Цветовете стават червеникави, откроявайки ги значително от предишните им стойности. За целта се модифицира контраста и гамата на глобалните цветове. Логиката се извършва в Level Blueprint, поради лесния начин, по който може да се достъпи обектът, поставен в нивото. В Tick функцията постоянно се проверява SlowMo булевата променлива, достъпена от MyCharacter класа. Бъде ли SlowMo равна на „вярно“, стойностите на PostProcessVolume ще бъдат приравнени с тези на друга променлива с име PostProcessSlowMo от този тип, на която вече ще са променени контраста и гамата. В противен случай PostProcessVolume ще бъде приравнена с променливата PostProcessNoSlowMo, която е със стартовите цветови стойности.



**Фиг. 3.43. Имплементацията на промяната на цветовите ефекти**

## **3.9. Реализация на денонощния цикъл**

В Level Blueprint-а на игралното ниво са създадени 6 функции, които постепенно да изменят ротацията и интензивността на насочената светлина и интензивността на сенките. Функциите NightRotateSource, NightSourceIntensity и NightSkyIntensity извършват залязването на слънцето и стъмването на сцената. Другите 3 функции DayRotateSource, DaySourceIntensity и DaySkyIntensity пък се грижат за изгряването на слънцето и осветяването на сцената. В NightRotateSource се интерполира ротацията на насочената светлина от нивото чрез помощта RInterpToConstant с цел движението да е плавно и константно. Генерираната ротация от RInterpToConstant се подава към SetActorRotation. След всяко преместване се извиква UpdateSunDirection от SkySphere обекта, защото местоположението на слънцето не се актуализира автоматично.



**Фиг. 3.44. Имплементацията NightRotateSource функцията**

DayRotateSource функцията е идентична като единствената разлика е крайното завъртане на светлинния източник. Останалите функциите променят интензивностите на съответните светлинни източници, подобно на гореописаните функции с разликата, че използват FInterpToConstant, получената стойност от който подават към SetIntensity.

# **ЧЕТВЪРТА ГЛАВА.**

# **РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ**

## **4.1. Изисквана компютърна конфигурация**

Потребителят трябва да разполага с компютърна система, на която да има инсталирана Windows 7 64-bit операционна система или по-нова версия. Той трябва да има налични перифериите мишка и клавиатура, без които няма да може да навигира из приложението. Препоръчително е наличието и на слушалки за най-добро изживяване. Минималните изисквания откъм хардуера са поне 500 мегабайта RAM памет и 1 гигабайт пространство на твърдия/статичния диск. Изискванията към процесора и видео картата не са специфицирани, но се предполага, че и по-старите и немощни ще се сравят добре с изкарването на голям брой кадри в секунда.

## **4.2. Инсталиране на приложението**

Единствено е необходимо игралното приложение да се изтегли от препоръчания източник. Изтеглените файлове ще заемат около 900 мегабайта от пространството на паметта. Стартирането на играта се извършва чрез кликването на изпълнимия файл MyProject в WesternShooter папката. До няколко секунди играта ще се отвори и играчът ще бъде посрещнат от главното меню.

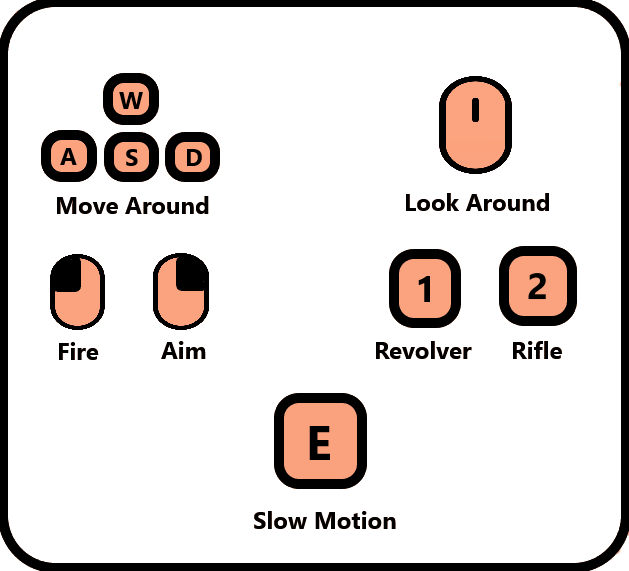
## **4.3. Навигиране в приложението**

Вляво на екрана, той ще може да избере дали да започне играта, където пише Start Game, или да излезе, където пише Quit Game. Също така ще има възможността да избере нивото на трудност. Потребителят трябва да използва мишката, за да премести курсора върху желаната от него опция и с левия бутон да я активира. Избирането на друга трудност се извършва чрез натискането на стрелката вдясно на текста.



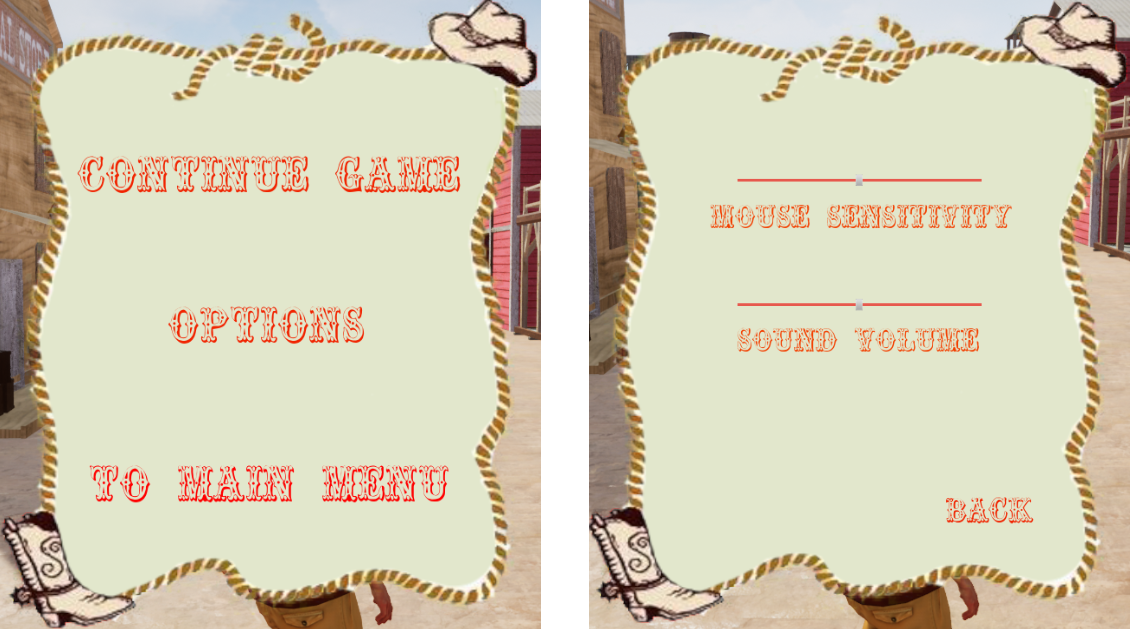
**Фиг. 4.1. Вида на главното меню**

След натискането на Start Game бутона, ще започне игралното ниво и вдясно на екрана ще бъде представено изображение, запознаващо играча с контролите. Той ще може да го затвори с натискането на F клавиша на клавиатурата.



**Фиг. 4.2. Вида на изображението, показващо контролите**

Играчът по всяко време ще може да паузира играта след като е започнала. За тази цел трябва да използва Escape клавиша. При натискане ще се появи меню, предлагащо опциите за връщане към играта, където пише Continue Game, отиване към настройките, където пише Options и насочване към главното меню, където пише To Main Menu, прекратяващо текущата играта. При натискане на Options бутона ще се появят два плъзгача – горния е за настройването на чувствителността на мишката, а долния за силата на звука. Плъзгането наляво ще намали стойностите им, а надясно – увеличи. С бутона Back потребителят ще може да се върне към предходните опции. Друг начин, по който играчът да се върне към играта е ново натискане на Escape клавиша.



**Фиг. 4.3. Вида на менюто за пауза**

**ЗАКЛЮЧЕНИЕ**

Настоящият облик на дипломната работа показва успешно изградена игра, използвайки Unreal Engine игровия двигател. Тя е достъпна за всекиго с налична компютърна система, покриваща минималните изисквания. Разработена е с целта да върви добре дори и на по-старите и немощни компютърни конфигурации. Спазени са всички заложени функционални изисквания и старателно и четливо е разработена логиката, улеснявайки направата на бъдещи модификации и добавки. Управлението е предназначено за клавиатура и мишка, които са и очевиден избор за играенето на игри с престрелки. Всяко ново пускане на играта ще предоставя различно изживяване.

Разбира се, има и голямо пространство за подобрения и добавяне на нови функционалности. Например разделението на нива е желана особеност, като на всяко следващо да се появяват нови по-трудни противници с различни оръжия.

# **ИЗПОЛЗВАНА ЛИТЕРАТУРА И РЕСУРСИ**

**Обща информация за игровите двигатели:**

* [**https://en.wikipedia.org/wiki/Game\_engine**](https://en.wikipedia.org/wiki/Game_engine)
* **https://interestingengineering.com/how-game-engines-work**
* [**https://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/**](https://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/)

**Обща информация за Unreal Engine:**

* [**https://en.wikipedia.org/wiki/Unreal\_Engine**](https://en.wikipedia.org/wiki/Unreal_Engine)
* [**https://hub.packtpub.com/overview-unreal-engine/**](https://hub.packtpub.com/overview-unreal-engine/)

**Обща информация за Unity:**

* [**https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose**](https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose)
* **https://en.wikipedia.org/wiki/Unity\_(game\_engine)**

**Програмиране на Unreal Engine:**

* [**https://docs.unrealengine.com/en-us/Programming**](https://docs.unrealengine.com/en-us/Programming)
* [**https://forums.unrealengine.com**](https://forums.unrealengine.com)
* [**https://orfeasel.com**](https://orfeasel.com)

**Използване на редактора на Unreal:**

* **https://docs.unrealengine.com/en-us/Engine/UI/LevelEditor**

**Ресурси(модели, текстури, анимации, звук и др.), взети от:**

* [**https://www.cgtrader.com**](https://www.cgtrader.com)
* [**https://free3d.com**](https://free3d.com)
* [**https://www.turbosquid.com**](https://www.turbosquid.com)
* [**https://www.mixamo.com/#/**](https://www.mixamo.com/#/)
* [**https://freesound.org**](https://freesound.org)
* **https://www.youtube.com**
* [**https://www.youtube.com/watch?v=Nt4W1B8cKy8**](https://www.youtube.com/watch?v=Nt4W1B8cKy8)
* [**https://www.youtube.com/watch?v=kLeKe2SHu78**](https://www.youtube.com/watch?v=kLeKe2SHu78)
* [**https://www.youtube.com/watch?v=kN8oRz90q38**](https://www.youtube.com/watch?v=kN8oRz90q38)

СЪДЪРЖАНИЕ

[**УВОД** 4](#_Toc8655951)

[**ПЪРВА ГЛАВА.** 6](#_Toc8655952)

[**МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА ВИДЕО ИГРИ** 6](#_Toc8655953)

[**1.1.** **Oсновни принципи, технологии и развойни среди за реализиране на видео** **игри** 6](#_Toc8655954)

[**1.1.1.** **Игрови двигатели** 6](#_Toc8655955)

[**1.1.2.** **Компонентите на съвременните игрови двигатели** 6](#_Toc8655956)

[**1.1.2.1. Контроли** 7](#_Toc8655957)

[**1.1.2.2. Графика** 7](#_Toc8655958)

[**1.1.2.3. Звук** 8](#_Toc8655959)

[**1.1.2.4. Физики** 8](#_Toc8655960)

[**1.1.2.5. Скриптиране** 9](#_Toc8655961)

[**1.2.** **Съществуващи решения и реализации** 9](#_Toc8655962)

[**1.2.1.** **Unreal Engine** 9](#_Toc8655963)

[**1.2.1.1.** **Контролите и игровата структура** 9](#_Toc8655964)

[**1.2.1.2.** **Осветлението** 10](#_Toc8655965)

[**1.2.1.3.** **Blueprint визуалният скриптов език** 11](#_Toc8655966)

[**1.2.2.** **Unity** 12](#_Toc8655967)

[**ВТОРА ГЛАВА.** 13](#_Toc8655968)

[**ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА ВИДЕО ИГРАТА** 13](#_Toc8655969)

[**2.1. Функционални изисквания към видео играта** 13](#_Toc8655970)

[**2.2. Съображения за избор на програмни средства и развойната среда** 13](#_Toc8655971)

[**2.3. Проектиране на героите** 14](#_Toc8655972)

[**2.4. Главният герой и неговото управление** 14](#_Toc8655973)

[**2.5. Проектиране на оръжията** 15](#_Toc8655974)

[**2.6. Видовете оръжия** 15](#_Toc8655975)

[**2.7. Куршумите** 16](#_Toc8655976)

[**2.8. Стрелбата и способността за забавяне на времето** 17](#_Toc8655977)

[**2.9. Проектиране на анимациите на главния герой** 18](#_Toc8655978)

[**2.10. Противникът, преследващ главния герой** 19](#_Toc8655979)

[**2.11. Противникът, изритващ вратите на кръчмата** 21](#_Toc8655980)

[**2.12. Противникът, показващ се от прозорците** 21](#_Toc8655981)

[**2.13. Проектиране на анимациите на противниковите герои** 21](#_Toc8655982)

[**2.14. Проектиране на сградите** 22](#_Toc8655983)

[**2.15. Проектиране на прозорците** 23](#_Toc8655984)

[**2.16. Управление на нивото** 23](#_Toc8655985)

[**2.17. Проектиране на потребителския интерфейс** 25](#_Toc8655986)

[**2.18. Проектиране на денонощния цикъл** 26](#_Toc8655987)

[**ТРЕТА ГЛАВА.** 27](#_Toc8655988)

[**ПРОГРАМНА РЕАЛИЗАЦИЯ НА ВИДЕО ИГРАТА** 27](#_Toc8655989)

[**3.1. Реализация на управлението на главния герой** 27](#_Toc8655990)

[**3.1.1. Настройване на клавишите за контрол** 27](#_Toc8655991)

[**3.1.2. Реализация на движението на героя** 28](#_Toc8655992)

[***3.1.2.1. Реализация на завъртането на героя към камерата*** 30](#_Toc8655993)

[**3.1.3. Реализация на движението на камерата около героя** 31](#_Toc8655994)

[**3.1.4. Реализация на регенерацията на кръв** 32](#_Toc8655995)

[**3.1.5. Реализация на мащабирането на камерата** 32](#_Toc8655996)

[**3.1.6. Реализация на стрелбата и смяната на оръжие** 33](#_Toc8655997)

[**3.1.7. Реализация на забавянето на времето** 35](#_Toc8655998)

[**3.1.8. Реализация на приближаването на камерата към свален противник** 36](#_Toc8655999)

[**3.2. Реализация на куршумите** 37](#_Toc8656000)

[**3.3. Реализация на оръжията** 38](#_Toc8656001)

[**3.3.1. Реализация на вземането на оръжие** 38](#_Toc8656002)

[**3.3.2. Реализация на изстрелването на куршум** 39](#_Toc8656003)

[**3.4. Реализация на противниците** 42](#_Toc8656004)

[**3.4.1. Реализация на протвиника, преследващ главния герой** 42](#_Toc8656005)

[**3.4.1.1. Реализация на движението на героя** 42](#_Toc8656006)

[45](#_Toc8656007)

[**3.4.1.2. Реализация на стрелбата на героя** 45](#_Toc8656008)

[**3.4.1.3. Реализация на унищожаването на героя** 45](#_Toc8656009)

[**3.4.2. Реализация на протвиника, показващ се от прозорците** 46](#_Toc8656010)

[**3.4.2.1 Реализация на завъртането на героя** 46](#_Toc8656011)

[**3.4.2.2 Реализация на стрелбата на героя** 48](#_Toc8656012)

[**3.5. Реализация на прозорците** 48](#_Toc8656013)

[**3.6. Реализация на сградите** 49](#_Toc8656014)

[**3.7. Реализация на управлението на нивото.** 50](#_Toc8656015)

[**3.8. Реализация на потребителския интерфейс** 53](#_Toc8656016)

[**3.8.1. Реализация на настройването на звука и чувствителността на мишкатa** 54](#_Toc8656017)

[**3.8.2. Реализация на настройването на трудността** 55](#_Toc8656018)

[**3.8.3. Реализация на настройването на глобалния цвят на камерата** 56](#_Toc8656019)

[**3.9. Реализация на денонощния цикъл** 57](#_Toc8656020)

[**ЧЕТВЪРТА ГЛАВА.** 58](#_Toc8656021)

[**РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ** 58](#_Toc8656022)

[**4.1. Изисквана компютърна конфигурация** 58](#_Toc8656023)

[**4.2. Инсталиране на приложението** 58](#_Toc8656024)

[**4.3. Навигиране в приложението** 58](#_Toc8656025)

[**ИЗПОЛЗВАНА ЛИТЕРАТУРА И РЕСУРСИ** 63](#_Toc8656026)