**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: 3D Third person RPG game

Дипломант:                    Научен ръководител:

*Георги Демирев                              Петър Георгиев*

СОФИЯ

2020

Дата на заданието: 15.11.2019 г. Утвърждавам:..............................

Дата на предаване: 15.02.2020 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Георги Антонов Демирев 12а клас

1.Тема: 3D Third person RPG game

2.Изисквания:

2.1 История разделена на нива с различни мисии, които могат да са взаимозависими

2.2 Система за създаване на предмети

2.3 Различни видове NPC

2.4 Противници, с които да се водят битки

2.5 Сухоземно и водно придвижване на героя

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :...........................................

Ръководител:..........................................

/Петър Георгиев /

Директор:................................................

/ доц. д-р инж. Ст. Стефанова /

**Отзив от научен ръководител**

Дипломното задание е изпълнено много добре. За времето, което имаше за изпълнение, дипломантът успя да навлезе и да разбере много добре работата с „Unity 3D“. Също така успя да се запознае с множество практики за писането на качествен игрови код и организиране на игрови проект. Въпреки сблъсъка с различни проблеми, които са характерни за тази сфера, той успя да намери адекватно решение и да имплементира функционалностите и механиките които си беше заложил. Играта съдържа няколко интересни и нетривиални механики, които са имплементирани по добър начин. Освен основната бойна механика, играта съдържа интересни механики за свързани куестове, създаване на нови вещи от материали, събрани от играча и т.н. Такъв тип системи стоят в основата на множество „Action RPG“ игри и смятам, че опитът който дипломантът придоби по време на разработката ще му е от голяма полза при бъдещите му начинания. Освен добрата имплементация, той се е старал да спазва различни конвенции за писане на качествен код.

На база на използваните технологии предлагам за рецензент Владимир Владимиров.

Ръководител: ……………………...........

/ Петър Георгиев /

# Увод

Дипломната работа представлява игра, разработена на игровия двигател Unity. Жанрът на играта е RPG (ролева игра) в който играчът играе като определен персонаж и следва определена история, разделена в отделни мисии, като тези мисии могат да бъдат взаимозависими, което означава, че всяко решение на играча може да се отрази на развитието на историята. Играчът гледа героя от перспективата на трето лице и светът е 3D. Обстановката на играта наподобява средновековието. Играчът ще има възможност да контактува с множество персонажи, които имат различни функции в света на играта. Ще има възможността и да води битки с различни противници. Съществуват различни типове предмети, които ще се събират от играча, като например: брони, оръжия, различни ресурси и др. Тези предмети ще могат да се получават при успешно изпълнение на някои мисии или чрез закупуване с валутата в играта. Играчът ще има определен брой жизнени точки. По време на битка, той губи точки пропорционално на силата на противника с когото се бие, но част от щетата се поема от екипировката на героя, ако има такава. При изчерпване на жизнените точки героят умира и се отваря панел, който дава възможност на играча да се съживи на последната запазена позиция или да се върне към началното меню.

# ПЪРВА ГЛАВА.

# МЕТОДИ И ТЕХНОЛОГИИ ЗА СЪЗДАВАНЕТО НА ВИДЕО ИГРИ

## 1.1 Технологии за разработване на видео игри

### 1.1.1 Разработка на игри от нулата

Един от начините за разработка на видео игра е започване от нулата. С достатъчно опит в програмирането и свободно време, човек може да напише собствени библиотеки за графика, звук, физика, анимация и т.н. или да ползва готови, за да разработи собствена видео игра. Такъв тип работа отнема доста време, но позволява на програмиста да конфигурира всички аспекти на играта според собствените си нужди, тъй като има контрол над всички части от процеса по създаването на една игра.

### 1.1.2 Използване на игрови двигател

Друг вариант за разработка на видео игри е използването на готов игрови двигател. За голяма част от по-малките компании и самостоятелни разработчици, тази алтернатива е доста по-добра. Игровият двигател дава голям набор от готови инструменти за създаване на видео игри, като по този начин спестява време и средства на разработчиците. Чрез използването на такъв тип „абстракция“, програмиста може с лекота да създава всякакъв тип системи и визуализации за своята игра, без да се затормозява с изграждането на базовите компоненти.

### 1.1.3 Създаване на собствен игрови двигател

Този вариант се използва от по-големите компании, които могат да си позволят времето и средствата за създаването на собствен игрови двигател. Когато една компания притежава собствен игрови двигател, това и позволява гъвкавост по отношение на разработваните игри, тъй като може да преправя и добавя функционалности по всяко време, според нуждите си. Голяма група от хора разработват игрови двигатели, като в повечето случаи големите компании имат цели отдели, които се занимават само с това.

### 1.1.4 Компоненти на игровите двигатели

Всички компоненти на игровите двигатели са описани в [1].

#### 1.1.4.1 Двигател за рендиране

Едно от най-важните неща за една игра е графиката. Игровите двигатели дават възможност за създаване на най-различни графични обекти, които подобряват визията на играта. Това обикновено става, чрез използването на активи, които се създават от външни програми за 3D моделиране. Тези активи съдържат модели и текстури, които се вмъкват в игровия двигател. Там те могат да бъдат местени, завъртани и оразмерявани, за да се създаде оформлението на нивото. Двигателят за рендиране генерира анимирани 3D изображения. Вместо да бъдат програмирани за директно изпълнение върху процесора или графичния процесор на компютъра, модерните двигатели за рендиране се изграждат върху програмни интерфейси като: Direct3D, OpenGL, и Vulkan, които предоставят софтуерна абстракция на графичния процесор.

#### 1.1.4.2 Физичен двигател

Друг важен компонент на игровия двигател е така наречения физичен двигател. Той отговаря за точното и консистентно прилагане на законите на физиката в света на играта. Двигателя е отговорен за засичането на колизии, симулирането на гравитация, прилагането на сила в отделни части на графичните обекти и много други физични явления. Физичните двигатели се разделят на две големи групи. Едната група са двигателите с висока точност, които калкулират физичните явления с голяма прецизност и отнемат огромно количество хардуерна мощност. Другият тип са физичните двигатели в реално време, които нямат толкова голяма точност, но за сметка на това успяват да калкулират физиката в реално време. Игровите двигатели използват втория тип, тъй като не могат да си позволят забавяне в изчисленията.

#### 1.1.4.3 Звуков двигател

Звуковият двигател е компонент, изграден от алгоритми, свързани с зареждането, модифицирането и произвеждането на звук от клиентската звукова система. Минималните изисквания към стандартен звуков двигател са да може да зарежда, декомпресира и пуска звукови файлове. По-напредналите звукови двигатели са способни да калкулират и произвеждат неща като ехо, височинна, амплитудна модулация, Доплеров ефект и др. Изчисленията могат да се правят директно върху процесора или чрез някои от съществуващите програмни интерфейси („OpenAL“, „SDL audio“, „XAudio 2“, „Web Audio“ и т.н.).

#### 1.1.4.4 Изкуствен интелект

Модулът за изкуствен интелект обикновено е отделен от основната логиката на играта, за да бъде проектиран и написан от софтуерни инженери със специални познания в областта. Повечето игри имплементират много различни системи за изкуствен интелект и затова тази част от игрите се счита за специфична спрямо дадения проект, за който е създадена. Много от модерните игрови двигатели съдържат различни алгоритми за намиране на пътища като А-звезда и подпрограми, които подпомагат по-бързото скриптиране на държанието на изкуствения интелект в играта.

## 1.2 Съществуващи програмни продукти

### 1.2.1 Unity 3D

„Unity 3D“ е считан за един от най-добрите игрови двигатели, защото предлага широк набор от инструменти, които могат да бъдат използвани дори и от хора без опит в програмирането. Една от най-важните функционалности, които „Unity“ предлага е неговата интеграция с множество платформи, която позволява на разработчиците да избират между голям брой налични платформи. Примери за такива платформи са „Android“, „Linux“, „macOS“, „iOS“ и „Windows“. „Unity“ поддържа двадесет и пет платформи, за разлика от конкурентни продукти като „Unreal Engine“ и „CryEngine“, които поддържат респективно десет и пет. Друго огромно предимство на „Unity“ е голямата общност, която помага при всякакъв тип затруднения. Тази общност е от особена голяма важност за начинаещите програмисти, които тепърва навлизат в разработката на игри и имат нужда от помощ за някои от базовите неща в двигателя. Един от аспектите на „Unity“, който му дава преднина пред конкуренцията е способността му да поддържа голям брой файлови формати, използвани във водещите софтуерни продукти за 3D моделиране като: 3D „Max“, „Blender“, „CINEMA“, „Maya“, „Softimage“ и много други. С „Unity“ разработчиците имат достъп до над 15,000 безплатни активи, като: 3D модели, аудио, анимации, скриптове и много други. „Unity“ използва „C#“ или „JavaScript“, които са доста по-предпочитани от програмистите, за разлика от „C++“, защото е много по-лесна смяната от Java към „C#“, отколкото към „C++“. Двигателят притежава изчистен и опростен интерфейс и е достатъчно олекотен, за да работи дори и на доста стари хардуерни конфигурации. [2]

### 1.2.2 Unreal Engine

„Unreal Engine“ е най-новият двигател, разработен от една от най-големите компании за разработка на видео игри и софтуер „Epic Games“. „Unreal Engine 4“ е наследен от „Unreal Development Kit“, по-познат като „UDK“ в гейминг средите. „Unreal Engine 4“ притежава уникални графики и функционалности като динамична светлина, които дават възможност на разработчиците да създават реалистична обстановка в своите игри. Една от най-новите функционалности на двигателя е неговата нова система за създаване на частици, която може да поддържа до един милион частици едновременно.

„Unreal Engine“ е изцяло безплатен за всеки който иска да го ползва, но компанията получава пет процента от всяка игра разработена с него. Компанията взима част от печалбата от всяка форма на приход на една игра, като например: вградени реклами, вградени покупки и приходите от самата продажба. Създателите на „Unreal Engine 4“ позволяват на разработчиците да използват пълната версия на двигателя безплатно, в случай, че печалбата която печелят от една игра е до три хиляди долара.

„Unreal“ използва така наречената „Blueprint“ система за визуално програмиране, която позволява разработването на игри да става дори и от хора без никакви знания в програмирането. Тези игри, естествено, ще имат своите ограничения. Един от минусите на двигателя е че не позволява разработката на игри за по-стари модели конзоли. [3]

### 1.2.3 CryEngine

Първоначално, „CryEngine“ е представен от голямата софтуерна корпорация „Crytek“, в първата „Far Cry“ игра. „CryEngine“ е един от най-мощните игрови двигатели в днешно време. Това което му помага да се изкачи в листата на най-добрите софтуерни решения за този сектор, са неговите графични възможности, които надхвърлят способностите на „Unity“ и са рамо до рамо с тези на „Unreal Engine 4“. Въпреки че е мощен инструмент за разработка на видео игри, „CryEngine“ отнема малко повече време на човек, за да се научи да използва платформата ефективно и е малко по-труден за научаване от хората, които нямат предишен опит с други игрови двигатели. „CryEngine“ поддържа разработване на игри за виртуална реалност и има невероятни визуални ефекти, като например: обемна мъгла и система за рендиране на облаци, които дават на облаците пълно пространствено 3D рендиране и реалистична визуализация на мъгла и други времеви ефекти. Един от плюсовете на двигателя е, че няма такса за употреба, която да се извлича от приходите по всяка игра, но има фиксирана такса от десет долара на месец за получаването на достъп до самия продукт. Двигателят притежава и огромен форум, където разработчиците могат да намерят отговор на много от въпросите, които възникват по време на разработка. [5]

### 1.2.4 GameMaker Studio

„GameMaker Studio“ е един от по-старите игрови двигатели, но с новото пускане на пазара на новия „GameMaker Studio 2“, двигателят стана доста по-мощен. Със своята бързина, приятен интерфейс и поддръжка на множество платформи, игровият двигател прави възможна бързата реализация на игра и нейното пускане на множество платформи. Двигателят е доста лесен за научаване и дава възможност на разработчиците за по-добър и точен контрол над логиката на играта със своя език „GameMaker Language“. Най-големият проблем на „GameMaker“ е цената му. Със своят ограничен скриптов език, зле разработен редактор за нива и трудна интеграция, цената на двигателя е прекалено голяма. [3]

### 1.2.5 Godot

„Godot“ първоначално е пуснат на пазара през 2014 година от Juan Linietsky, Ariel Manzur, което го прави прекалено нов двигател за вършене на работа. Двигателят е мултиплатформен, основно за разработка на игри за телефони и компютри и се стреми да стане един от най-добрите. Двигателят няма графичния капацитет на останалите игрови двигатели. Може да се използва като добра входна точка за учещите се, които с времето да започнат да правят по-сложни и трудни проекти. „Godot“ притежава интегриран редактор на анимации, доста е лек и е с отворен код, но е доста зле документиран и терминологията която използва е доста объркана. [4]

# ВТОРА ГЛАВА.

# ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА ВИДЕО ИГРАТА

## 2.1 Функционални изисквания към видео играта

От играта се изисква да бъде имплементирана история, разделена в мисии, които могат да бъдат взаимозависими. Това означава, че всяко решение което се вземе от играча ще има тежест над развитието на играта и историята. В играта трябва да има система за създаване на предмети. Освен самия играч, в света на играта трябва да има и други персонажи, наречени „NPC“ („Non-playable characters“) и играчът да има възможност да контактува с тях. Трябва да се направят противници, с които играчът да води битки. Героят трябва да може да се придвижва по земя и във вода.

## 2.2 Избор на програмни средства и развойната среда

### 2.2.1 Unity

„Unity“ има изключително голяма общност от разработчици и всеки елементарен проблем или въпрос, които може да възникне по време на разработката на играта, бива решен изключително бързо. Друго преимущество на този игрови двигател е огромният му набор от активи. Например, чрез закупуването на готови 3D модели от други потребители на двигателя, разработчика може да подобри външния вид на своята игра, без да губи време и усилия в направата на собствени модели. Една от важните характеристики на двигателя е неговата бързина и лекота на действие. За много кратко време може да се разработи доста голям проект.

### 2.2.2 Microsoft Visual Studio

Що се отнася до текстови редактори, има много разнообразни възможности.Преимуществото на „Visual Studio“ пред останалите е добрата му интеграция с игровия двигател. Още от самата инсталация на версията на игровия двигател, се дава възможност на потребителя да изтегли „Visual Studio“. Единствения недостатък на този текстов редактор е че е доста тежък спрямо компютърни ресурси. Ако компютърната конфигурация не е достатъчно мощна, това може адски много да забави зареждането на проекта и промяната на кода. Затова в такива случаи може да се ползват и други алтернативи като: „Sublime Text 3“, „Visual Studio Code“, „Atom“ и т.н.

## 2.3 Структура на видео играта

Цялата структура на играта е базирана на основния модел на игровия двигател „Unity“, базиран на разделянето на компоненти. Всеки скрипт сам по себе си е компонент, който е закачен за обект от играта. По подразбиране модела за програмиране на игри в „Unity“ е базиран на ООП, тъй като всеки скрипт в играта е клас, който съдържа обекти от играта, информация както и функции, които определят държанието на обектите и тяхната информация. Всичко това се събира в един файл, който се превръща в компонент на даден обект от играта и наследява главния клас „MonoBehaviour“. В случай, че скриптът няма нужда да манипулира каквито и да било обекти от самата игра, той може да бъде просто обикновен клас.

### 2.3.1 Проектиране на главния герой

Движението на главния герой се поема от скриптът „PlayerMovement“. Скриптът завърта плавно героя към посоката в която гледа камерата. Самото движение на героя става чрез движението на анимациите, а те се контролират чрез Animation Controller компонент, който е закачен за модела на главния герой. Скриптът „InputController“ свърза натискането на бутони от играчът със движението на героя. Скриптовете „DealDamage“ и „ReceiveHit“ са двата универсални скрипта, които карат един обект съответно, да може да напада и да бъде нападан. Те се използват както при главния герой, така и при противниците. За да се опишат всичките параметри на главния герой, както и на останалите персонажи се използва скриптът „CharacterProps“. В него се пази оставащият живот на героя, неговото име, количеството защита и атака, които притежава, нивото, колко пари има и много други параметри. Количеството защита и атака се изчислява на база на екипировката на героя, а при противниците тя се задава предварително. Всеки елемент на Екипировката притежава стойност на атака и защита, които се използват при изчисленията.

### 2.3.3 Проектиране на системата за мисии

Системата за мисии в играта е съставена от няколко основни скрипта. Скриптът който съдържа информацията за една мисия се нарича „Quest“ и наследява класа „ScriptableObject“, което го превръща в актив, който не може да бъде закачен за обект от играта. „Quest“ съдържа в себе си информация за описанието, целта и наградата за една мисия. Отделните мисии се управляват от главният клас „QuestController“, който извършва всички операции свързани със мисиите, като: започване, отказване и приключване на дадена мисия. Скриптът „QuestUI“ отговаря за визуализацията на панела с мисии в играта. Този панел дава възможност на играча да прегледа всички свой активни и приключени мисии, както и детайлите по тях. Тъй като играчът трябва да може да прави избори, докато преминава през различните мисии, скриптът „Quest“ съдържа в себе си класа „Goal“, който от своя страна описва началната цел на играча. При приключването на дадената цел, пред играча се поставя прозорец за избор. След като играчът направи своя избор, играта зарежда следващата цел. При достигана на финалната цел, мисията е изпълнена и играчът получава съответната награда. След като мисията е изпълнена, скриптът „QuestController“ изчислява спрямо изборите на играча коя е следващата мисия. По този начин изборите на играча се отразяват на историята на играта и тя се променя спрямо тях.

### 2.3.4 Проектиране на система за създаване на предмети

Предметите, както и мисиите, имат един основен базов клас „Items“, който наследява класа „ScriptableObject“ и съдържа в себе си базовата информация за един предмет, като: име, цена, изображение и т.н. Този клас може да бъде наследяван от други класове, които да създават специфични типове предмети. Предметите, които притежава играчът се съдържат в инвентар, които е описан от класа „Inventory“. Те могат да бъдат купувани от персонажи в играта, които играят роля на продавачи. Предмети се получават и при успешното изпълняване на мисии. Самата функция за създаване на предмети се поема от персонаж в играта, наречен „Blacksmith“, който получава от играча два предмета и рецепта за това какъв предмет може да получи от смесването на тези два предмета. При създаването на новия предмет играчът заплаща съответната цена описана в рецептата.

### 2.3.5 Проектиране на останалите персонажи в играта и интеракциите с тях

Всеки предмет или персонаж в играта с който може да взаимодейства играчът, притежава скрипта „Interactable“, или някой от скриптове, които го наследяват. Главният герой притежава скрипта „Interact“, който засича наличието на обекти със възможност за интеракция срещу самия герой. Различните персонажи имат различни функции и затова използват наследници на „Interactable“, които определят държанието на персонажа при интеракция. Съществува скрипта „InteractionUI“, който се грижи за зареждането на потребителския интерфейс по време на интеракция. Тези скриптове са структурирани според концепцията на „Unity“ за компоненти и затова се ползват както за персонажите в играта, така и за предметите и мисиите.

### 2.3.6 Проектиране на противниците

Всички противници в играта притежават скрипта „EnemyBehaviour“, който определя базовите механики на всички противници, като: засичане на герой, движение и атака. Тъй като постоянно трябва да се създават нови противници в съответните им зони, се използва скрипта „EnemySpawner“, който се грижи за първоначалната инициализация на определен брой противници в зоната и след като някой от тях умре, се създава нов, така че да не се изчерпва бройката. Движението на противниците се контролира от вградения в „Unity“, „NavMesh“. Чрез скрипт се поставя дестинация на навигацията и чрез алгоритъм за намиране на път, противника се придвижва през картата до зададената дестинация.

### 2.3.7 Проектиране на вода и водно придвижване

Графиката на водата е взета от стандартния пакет с активи на „Unity“. Засичането на вода и движението на главния герой във вода са реализирани в скрипта „Swimming“. При навлизането на главният герой на определена дълбочина, той сменя изцяло своята анимация на движение. Действието на гравитацията е спряно, за да не потъва играча и той може свободно да се придвижва из водната повърхност. При натискането на бутона „Running“, героят започва да плува по-бързо.

### 2.3.8 Проектиране на система за запазване на прогреса

Прогресът на играча се запазва в бинарен файл. Този файл се формира от данните на скрипта „PlayerData“. „PlayerData“ скриптът съдържа информацията от „CharacterProps“, последно запазена позиция на героя, както и лист с предметите в инвентара на героя и лист за неговата екипировка. След като тази информация бива превърната в бинарен файл, след повторно пускане на играта, предметите се зареждат на съответното място и играчът бива зареден с всичките му параметри на съответната позиция.

# ТРЕТА ГЛАВА.

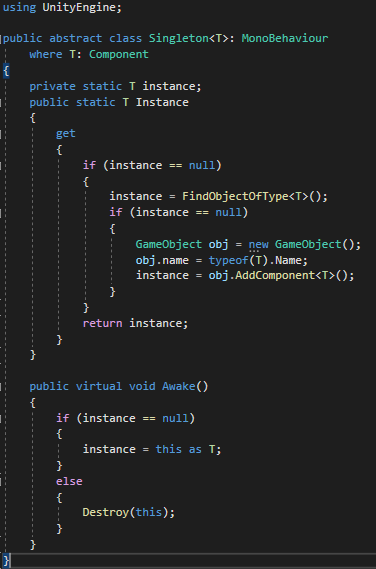
# РЕАЛИЗАЦИЯ НА ВИДЕО ИГРАТА

## 3.1 Базови скриптове

Голяма част от системите в играта разчитат на базови класове и интерфейси, като по този начин, чрез прилагане на някои ООП принципи се намалява големината на кода и се подобрява структурата на целия проект.

### 3.1.1 Имплементация на Сингълтън модела

Тъй като голяма част от класовете в играта имат само един обект, използването на Сингълтън модела подпомага връзката между отделните компоненти и обработката на информация. На фиг. 3.1 е показана имплементацията на Сингълтън модела. Целта е класът да държи в себе си инстанция на самия него, която да е статична и да може да се достъпва от външни класове. При поискване на тази инстанция се проверява дали тя е инициализирана и в случай, че не е се прави нов игрови обект и на него се присвоява даденият скрипт. При първоначалното инициализиране на този скрипт, инстанцията става самият скрипт, който се инициализира. Самият клас Singleton е генерализиран, за да може да се използва във всякакъв тип други класове, достатъчно е само те да го наследяват.



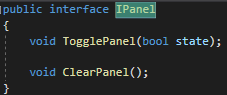
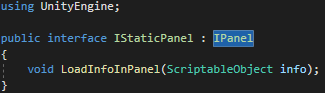
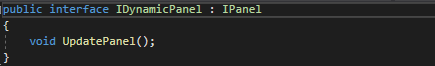
**Фиг. 3.1. Имплементация на Сингълтън модела**

### 3.1.2 Компоненти за управление на потребителския интерфейс

Визуализирането на информацията в играта се контролира от различни скриптове. Тези скриптове се грижат за зареждането на информация в панелите. Базовите операции като зареждане на текст в поле, изключване/включване на даден и т.н. са изнесени в скрипта „UIController“. Той предоставя на всички останали скриптове за визуализация необходимите методи, като по този начин намалява значително обема и сложността на кода. Всички скриптове за контрол на потребителския интерфейс са наследници на класа „Singleton“, тъй като в цялата сцена имат само по една инстанция и по този начин лесно се осъществява връзката с тях.

### 3.1.2.1 Структура на панелите за информация

Панелите, които играят роля на контейнери за информация се разделят на два основни типа: статични и динамични. За съответния тип панели има различен интерфейс, който определя задължителните методи за контрол на съдържанието. (фиг. 3.2) Всеки такъв скрипт имплементира съответните методи в зависимост от своето предназначение.

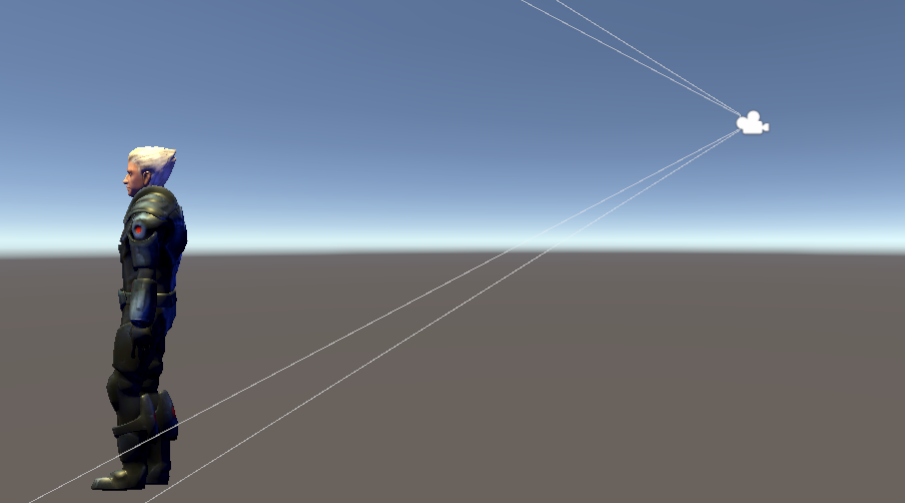
  

**Фиг. 3.2. Интерфейси за панелите с информация**

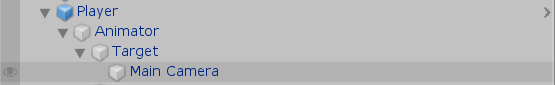
## 3.2 Реализация на камерата от трето лице

### 3.2.1 Разположение на камерата спрямо героя

Тъй като камерата трябва да се движи заедно с героя, тя е закачена като дете на обекта на героя. По този начин всяко едно движение или завъртане на героя се предава на камерата. Камерата е дете на обекта „Target”, който служи за насочване на самата камера към една определена точка. (фиг. 3.4)



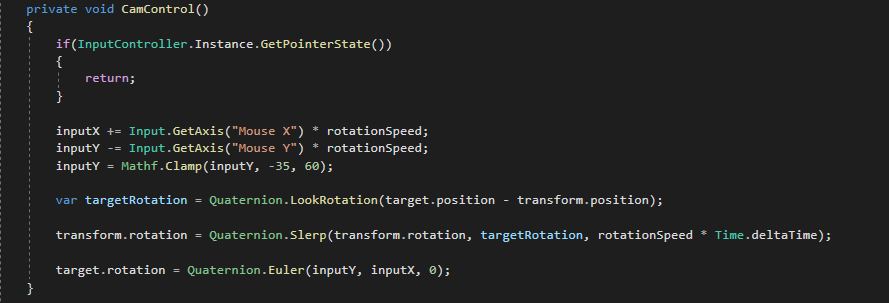
**Фиг 3.3. Позиция на камерата спрямо героя**



**Фиг 3.4. Разположение на камерата в йерархията на сцената**

### 3.2.2 Движение на камерата около героя

Движението на камерата е реализирано, чрез скрипта „ThirdPersonCamera“. В него е дефиниран метода „CamControl“, който завърта камерата около героя, спрямо движението на мишката. (фиг. 3.5) В началото на метода се проверява дали мишката е пусната, тоест дали в момента играчът взаимодейства с графичните панели в играта или управлява самия герой. В случай че не управлява героя, функцията се прекратява, за да не се получава въртене докато играчът не управлява героя. След това се изчислява посоката в която трябва да е завъртяна камерата на база на позицията на героя и камерата. Камерата се завърта в тази посока със предварително зададена скорост и накрая се прилага завъртането което се получава от координатите на мишката.

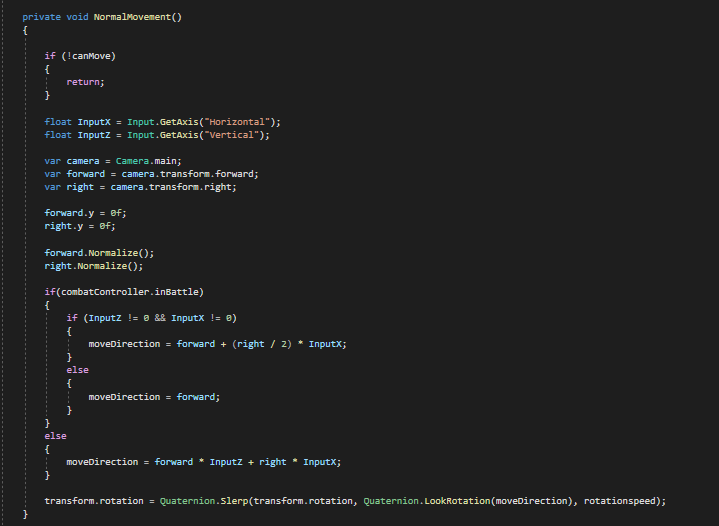


**Фиг 3.5. Функция за завъртане на камерата около героя**

## 3.3 Реализация на управлението на главния герой

### 3.3.1 Завъртане на героят в посоката на камерата

Посоката на движение се изчислява в зависимост от това дали героят се намира в режим на битка, защото тогава движението се променя. Когато героят не се намира в режим на битка, завъртането се изчислява на база посоката за напред на камерата умножена по стойността на клавишите за движение напред и назад и посока настрани, умножена по стойността на клавишите за наляво и надясно. (фиг. 3.6) Когато героят е в режим на битка, стойностите се умножават само по стойността на клавишите за странично движение, тъй като героят не трябва да може да се завърта назад, а само наполовина в страни. След като се изчисли посоката на въртене, героят се завърта със предварително зададена скорост.



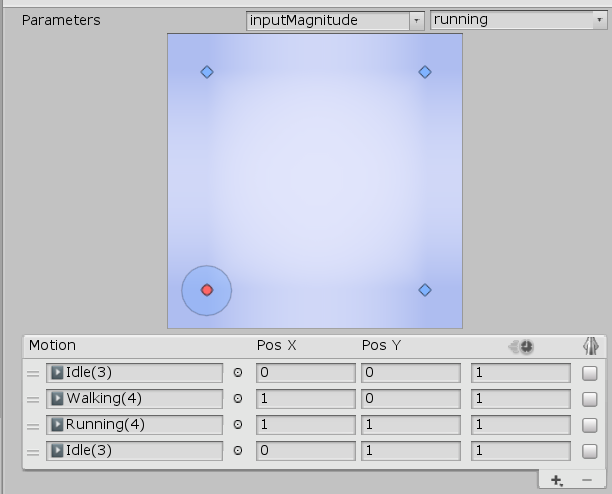
**Фиг 3.6. Функция за движение на героя**

### 3.3.2 Движение на героя

Самото движение в пространството на героя се извършва от анимацията, която се използва. Функцията „InputMagnitude“ в скрипта „PlayerMovement“ се грижи за контрола над анимацията, така че героят да се придвижва спрямо натиснатите клавиши. Анимациите се контролират от два параметъра: „inputMagnitude“ и „running“. Двата параметъра се изчисляват на база на натиснатите клавиши, като „inputMagnitude“ е резултата от изчислената дължина на двуизмерен вектор, който се състой от стойностите на клавишите за движение, повдигнати на втора степен. (фиг. 3.7) След като стойностите им се приложат в контролера за анимации, той изчислява спрямо стойностите, коя е анимацията която трябва да се зареди. (фиг. 3.8)



**Фиг 3.7. Функция за изчисляването на движението спрямо натиснатите клавиши**



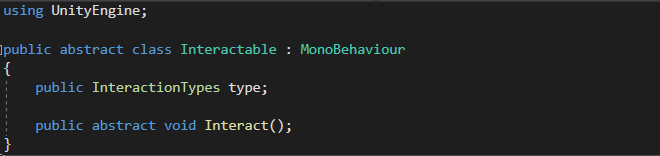
**Фиг 3.8. Стойностите за преход между различните анимации в контролера за анимации на играча при движение извън битка**

## 3.4 Реализация на системата за взаимодействие

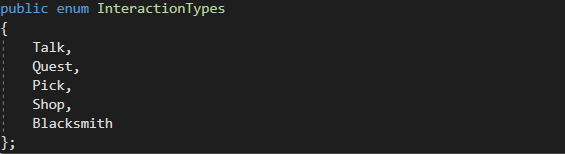
Всички скриптове, които служат за взаимодействие между героя и различни обекти наследяват абстрактния клас „Interactable“. (фиг. 3.9) Той съдържа типът на взаимодействието и абстрактния метод „Interact“. Има няколко типа взаимодействие за различните видове взаимодействие в играта като например: събиране на предмети, разговаряне с неигровите персонажи, стартиране на мисия и създаване на предмети. (фиг. 3.10)

### 3.4.1 Засичане на възможност за интеракция

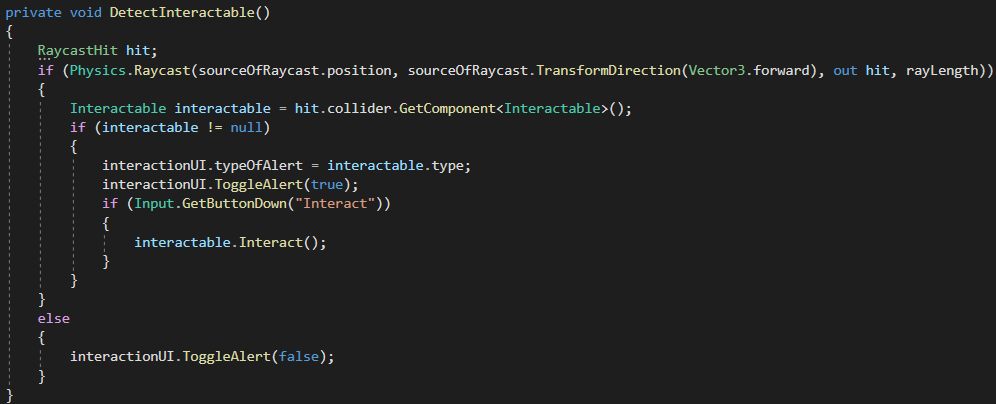
За да може героят да взаимодейства с останалите предмети и персонажи в играта е необходимо да се засече всеки потенциален обект, който притежава възможност за интеракция. Тази задача се изпълнява от скрипта „Interact“, който е компонент, закачен за обекта на героя в сцената. (фиг. 3.11) При своята инициализация, скриптът „Interact“ получава координати. Тези координати се използват за да се създаде лъч.Дължината на лъча се определя от променливата константа „rayLength“. В случай, че лъчът засече обект пред себе си, който съдържа компонента „Interactable“, се задейства надпис на екрана, който показва типа на интеракцията и в случай, че играчът натисне бутона за интеракция, се извиква функцията „Interact“ на класа „Interacable“.



**Фиг 3.9. Дефиниция на класа „Interactable“**



Фиг 3.10. Дефиниция на типовете интеракции



Фиг 3.11. Метод за засичане на интеракция

### 3.3.2 Потребителски интерфейс на интеракциите

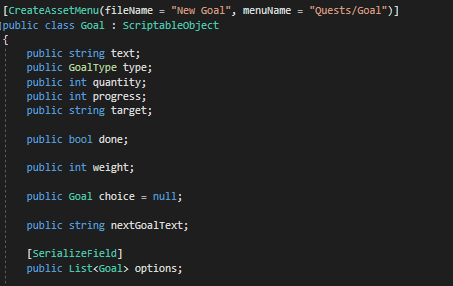
Интерфейсът на интеракциите е реализиран в скрипта „InteractionUI“, който наследява абстрактния генерализиран клас „Singleton“, тъй като може да съществува само една инстанция от този клас. Тъй като панела за интеракции е статичен, класът „InteractionUI“ имплементира интерфейса „IStaticPanel“. След извикването на функцията „Interact“ на класа „Interacable“, тази функция прави панела за интеракции видим и зарежда необходимата информация за играча, чрез използването на функциите „TogglePanel“ и „LoadInfoInPanel“, които „InteractionUI“ имплементира от интерфейса „IStaticPanel“. (фиг. 3.12)



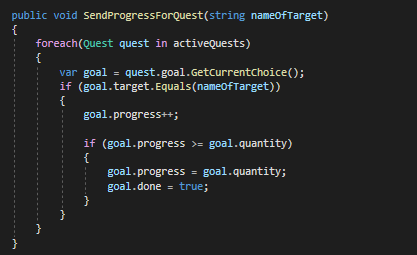
**Фиг. 3.12. Имплементация на функциите от интерфейса „IStaticPanel“ в класа „InteractionUI“**

## 3.4 Имплементация на взаимозависими мисии

Тъй като класът „Quest“ е наследник на „Scriptableobject“, това позволява много лесно да се създават активи от тип „Quest“, в които да се описват различните полета на една мисия. След като се създаде една мисия, тя се присвоява към съответното „NPC“, което трябва да я направи достъпна за интеракция, посредством компонента „QuestGiver“. При интеракция от страна на играча с персонаж, който притежава мисия се зарежда информация за мисията в панела за интеракции. В случай, че играчът приеме мисията се зарежда първоначалната цел, която се съдържа в обекта Goal, в класа „Quest“. (фиг. 3.13) За да се следи прогреса на играча за дадена мисия, при интеракция се извиква метода „SendProgressForQuest“ на скрипта „QuestController“ със съответното име на обекта или персонажа и този метод променя прогреса на играча за мисията. (фиг. 3.14) След като прогресът на първата цел достигне 100%, тя се счита за изпълнена. Това се означава, чрез булевата променлива „done“ на класа „Goal“, която се променя от метода „SendProgressForQuest“. Класът „Quest“ също притежава такава променлива, но тя обозначава дали цялата мисия е завършена, а не само части от нея.

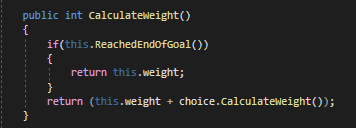


**Фиг. 3.13. Структура на класа „Goal“**

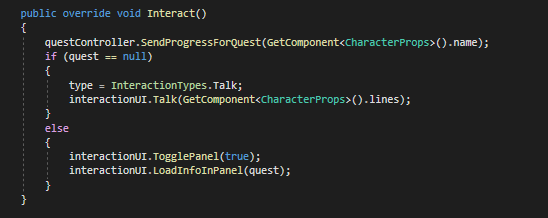


Фиг. 3.14. Имплементация на метод за следене на прогреса по мисиите на героя

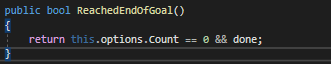
След като играчът изпълни първата цел, в случай, че има още неизпълнени цели, пред него се зарежда менюто за избор, което се управлява от скрипта „ChoiceUI“. След като играчът направи избор за това как да продължи мисията, този избор се запазва и се зарежда избраната цел. При изчерпване на целите, персонажа, който притежава мисията, извиква вградения метод на класа „Quest“, „GetNextQuest“, която извиква метода „CalculateWeight“, описан на фиг. 3.16 Методът изчислява резултата от сбора на „weight“ на всички избрани цели от героя и в резултат, „GetNextQuest“, връща една от две възможни следващи мисии, в зависимост от това дали числото е положително или отрицателно. В случай, че не са зададени референции към следващи мисии, персонажа с мисиите променя своя тип на интеракция на персонаж за говорене и спира да дава мисии, докато не се зареди нова. (фиг. 3.17) Когато всички цели бъдат изпълнени от играча, той трябва да завърши мисията при съответния персонаж, от когото я е получил. Когато играчът взаимодейства с този персонаж, скриптът „InteractionUI“ проверява при зареждането на информация в метода „LoadInfoInPanel“, дали всички цели са изпълнени, чрез метода „ReachedEndOfGoal“, описан на фиг. 3.18, който се извиква на последната цел. В случай, че резултатът е положителен, стойността на променливата „done“ става „true“ и се зарежда бутонът „Complete”, към който се добавя функцията на QuestController, „CompleteQuest“. След като играчът натисне бутона, метода се извиква и съответната награда, чиято структура е описана на фиг. 3.20, се прехвърля към героя под формата на пари, точки за вдигане на ниво и предмети. (фиг. 3.19)



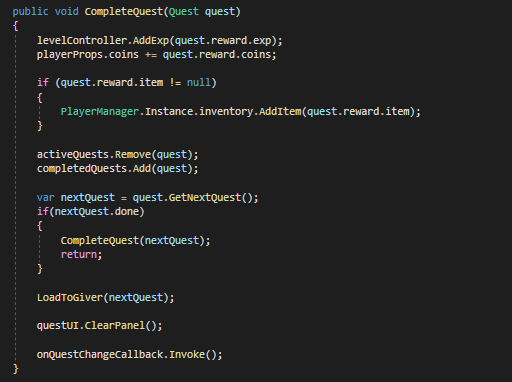
Фиг. 3.16. Имплементация на метода „CalculateWeight“ в скрипта „Goal“



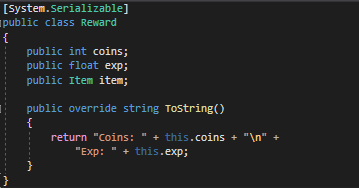
Фиг. 3.17. Имплементация на метода „Interact“ в скрипта „QuestGiver“



**Фиг. 3.18. Имплементация на метода „ReachedEndOfGoal“ в скрипта „Goal“**



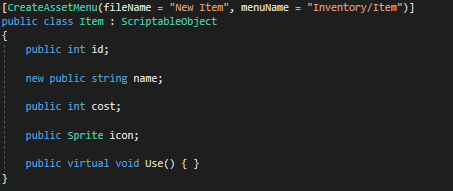
Фиг. 3.19 Метод на скрипта „QuestController“, който се извиква при натискане на бутона за завършване на мисия: „Complete“



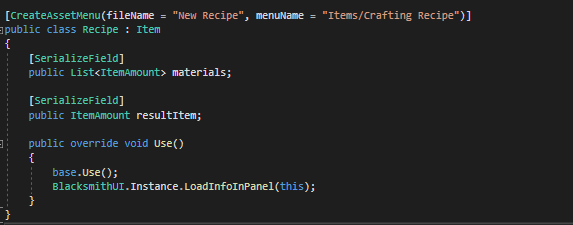
Фиг. 3.20. Клас контейнер „Reward“, който описва структурата на наградата, която се получава при успешно завършване на мисия

## 3.5 Имплементация на система за създаване на предмети

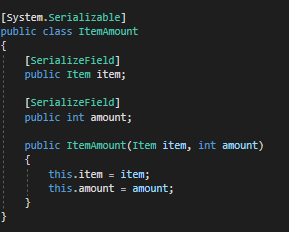
За интеракцията с персонажите, които дават възможност на играча да създава предмети се грижи скрипта „Blacksmith“, който е наследник на класа Interactable. Той съдържа два метода, предефинирания „Interact“, който извиква методите за зареждане на екрана за създаване на предмети в скрипта „BlacksmithUI“ и метода „Craft“, който извършва операцията по създаването на самия предмет. Метода „Craft“ проверява дали играчът има достатъчно пари и необходимите премети, за да създаде описания в рецептата предмет. Рецептата, описана на фиг. 3.22, представлява клас, който наследява класа „Item“ и съдържа списък от предмети за материали, както и предметът, който ще се създаде. В случай, че някъде трябва да има повече от един предмет, се ползва структурата „ItemAmount“, която съдържа предмет и съответна бройка, тази структура се ползва в системите за инвентар, магазина и създаване на предмети. (фиг. 3.23)



**Фиг. 3.21. Имплементация на класа „Item“**



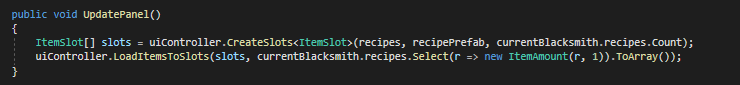
Фиг. 3.22. Имплементация на класа „Recipe“



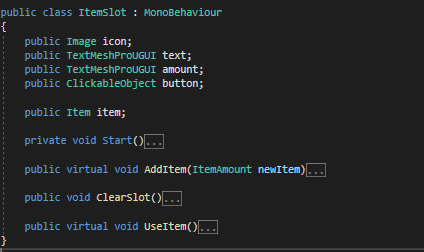
Фиг. 3.23. Структура за следене на количеството предмети в дадена система

### 3.5.1 Екран за създаване на предмети

Екранът за създаване на предмети се дели на две части, едната е листа с рецептите, който е динамичен, а другата е екранът, който показва информация за заредената рецепта и е статичен. Затова скриптът за контролиране на панела за създаване на предмети имплементира и двата интерфейса за панели и съответно предефинира методите им. Частта с рецептите, описана на фиг. 3.24, използва методите на класът „UIController“ за създаване и зареждане на слотове в панелите. Тези слотове представляват предварително създадени игрови обекти, които имат различна форма според предназначението им. В случая на системата за предмети, тези слотове притежават икона на предмета, име на предмета и количество. Информацията в тези слотове се поддържа от скрипт, наследник на класа „ItemSlot“. (фиг. 3.25) Статичната част от екрана се изгражда чрез панели, но в случая те нямат нужда да бъдат многократно създавани и затова са предварително инициализирани в сцената на играта.



Фиг. 3.24. Функция за опресняване на съдържанието на панела в системата за създаване на предмети

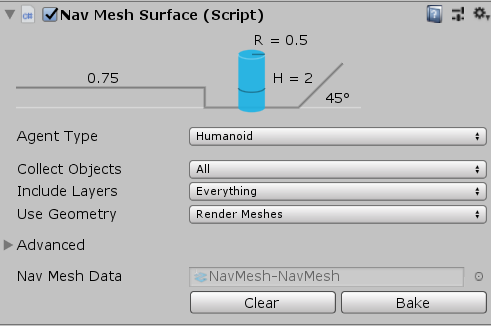


Фиг. 3.25. Дефиниция на класа „ItemSlot“

## 3.6 Имплементация на противници

### 3.6.1 Движение на противниците

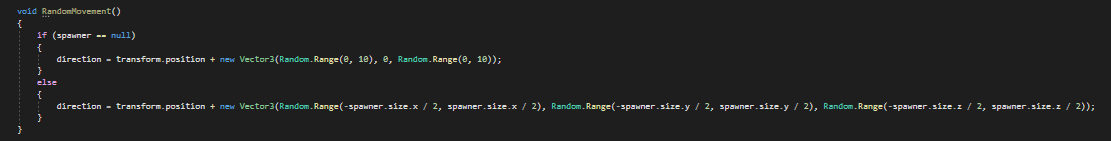
Тъй като противниците в играта не се контролират от играча и трябва да се движат самостоятелно, тяхното движение е имплементирано, чрез вградената „NavMesh“ система на игровия двигател. Тя осигурява готова функционалност за придвижване на предмети през терена, като се взима в предвид най-късото разстояние до дадена точка и определени обекти, маркирани като препятствия. За по-улеснена имплементация на връзки между различни терени за навигация и динамично зареждане по време на игра се използват допълнителни компоненти на „NavMesh“ системата, които не са вградени в самия двигател. (фиг. 3.26) Всеки противник в играта притежава скрипт „EnemyController“. Той използва метода „DetectPlayer“, който се грижи за засичането на играча и генерирането на произволна посока на движение за противниците. (фиг. 2.27) Засичането на играча става, чрез вградената функция „Vector3.Distance“ на игровия двигател, която изчислява разстоянието между две точки в пространството. След като се изчисли тази дистанция, тя се сравнява с предварително зададено число за големината на областта в която противниците засичат героя. В случай, че героят се намира в областта на противник, този противник започва да го преследва. Когато стигне достатъчно близо до героя, противника започва да го атакува, докато не напусне обсега му. В случай, че героят се намира извън този обсег, противникът започва произволно движение, генерирано от функцията „RandomMovement“, описана на фиг. 2.28.



**Фиг. 3.26. „NavMesh“ компонент за динамично изчисляване на пътя на противниците**



**Фиг. 2.27. Имплементация на метод „DetectPlayer“, който се грижи за засичането на героя**



Фиг. 2.28. Имплементация на метода „RandomMovement“, който генерира произволна посока на движение за противниците

### 3.6.2 Водене на битки

#### 3.6.2.1 От страна на противниците

Битките в играта са реализирани на база на размяната на удари между героя и противниците. За да може да се регистрира удар, се използват анимациите, като на определен фрейм от анимацията се извиква метода „EnableAttack“, който от своя страна използва скрипта „CombatController“ за да активира тригерите. Използва се вградения метод „OnTriggerEnter“ на „Unity“, който се извиква автоматично при активирането на тригера, закачен за даден персонаж. При натискане на бутона „Attack“, от страна на играчът, „EnemyBehaviour“ извиква метода „Attack“ на скрипта „DealDamage“, който от своя страна пуска анимацията за атака. Тя активира тригерите, посредством „Animation Events“ системата на „Unity“ и при засичане на противников персонаж, се открива компонент „ReceiveHit“ и се извиква метода „GetHit“, описан на фиг. 2.29, който извършва изчисленията след нанесен удар. При изчерпване на жизнените точки се извиква метода „Die“, който зарежда анимация и в зависимост от типа на персонажа, унищожава обекта или зарежда екран за умиране.

#### 3.6.2.2 От страна на играча

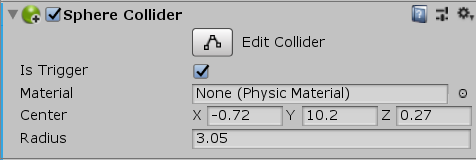
Играчът използва същите системи за водене на битки, с изключение на автоматизацията от страна на атаката. Вместо атаки през определен интервал, играчът използва скрипта „InputController“ за засичане на натискане на бутона за атака. Освен това той използва скрипта „ComboSystem“, който съдържа таймер за отброяването на изминалото време между всяко натискане. В случай, че играчът натисне последователно няколко пъти клавиша за атака в определен времеви прозорец, създаден от таймера, „ComboSystem“ продължава да изпълнява допълнителни анимации, които удължават атаката. Освен комбо атаките, играчът може да блокира удари, като това намалява отнемания живот при нанесен удар. Калкулацията на нанесената щета се изчислява, отново, в метода „GetHit“.



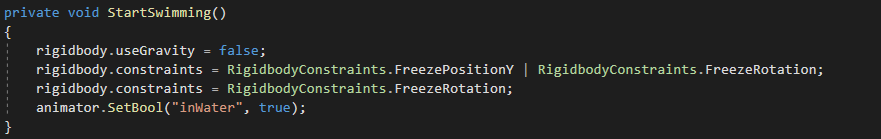
**Фиг. 2.29. Метод „GetHit“, който изчислява нанесената щета върху даден персонаж**

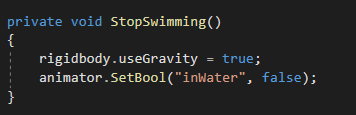
## 3.5 Имплементация на система за водно придвижване

За да може героят да се придвижва по вода е необходимо да се заредят анимации, различни от тези за сухоземно придвижване. Подмяната на анимациите се извършва при засичането на водна повърхност. Това засичане се постига, чрез вградените компоненти за колизии на „Unity“. (фиг. 2.30) Тези компоненти се намират в режим на тригери и засичат всеки обект, който премине през тях и притежава същия компонент за колизии. В случая на водата, тригер, закачен за героя засича тригера на водата. Описания в скрипта „Swimming“ метод „OnTriggerEnter“ оказва какво да се случи когато тригера на играча засече друг. В случая извиква метода „StartSwimming“ (фиг. 2.31), който спира действието на гравитацията над играча, за да не потъне във водата и задейства анимациите за придвижване във вода. Когато тригерът на играча напусне водата се извиква метода „OnTriggerExit“, който извиква метода „StopSwimming“, описан на фиг. 2.31 и той от своя страна активира гравитацията и връща анимациите за сухоземно придвижване.



**Фиг. 2.30. Компонент за засичане на колизии между обекти в режим на тригер**





Фиг. 2.31. Методи за превключване на режима на водно и сухоземно придвижване

# ЧЕТВЪРТА ГЛАВА.

# РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ

## 4.1 Изисквания към компютърната конфигурация

### 4.1.1 Операционна система

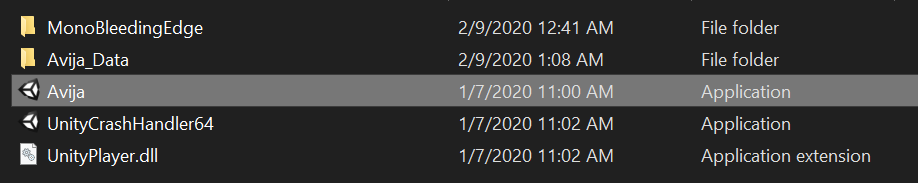
Задължително изискване за успешното инсталиране и пускане на играта е наличието на компютърна конфигурация с инсталиран „Windows 7“ или 10.

### 4.1.2 Хардуерни изисквания

Компютърната конфигурация трябва да притежава вградена в процесора или външна видео карта, съвместима с DirectX12. Рамта на компютъра трябва да е поне 1гб. Максималното пространство, което играта може да заема е 1гб. Няма специфични изисквания към процесора на машината.

### 4.1.3 Инсталация на играта

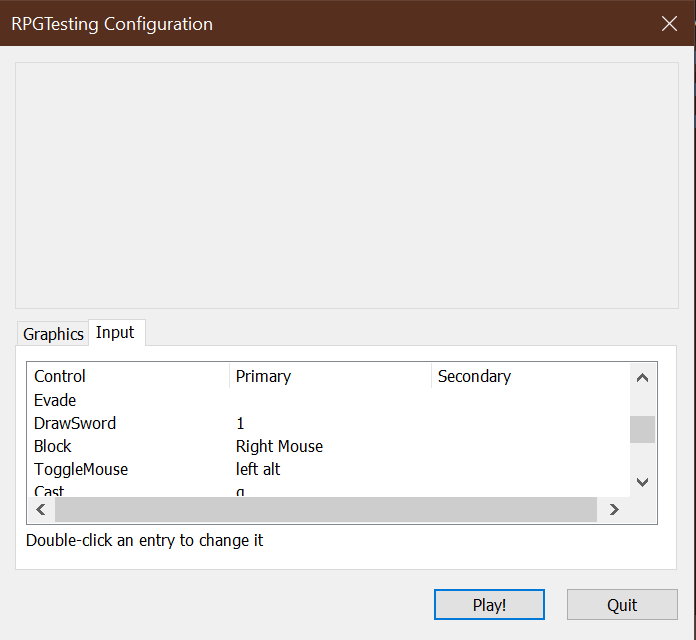
Играта не изисква предварителна инсталация. Единственото необходимо е изтеглянето на файловете от препоръчания източник. След като се изтегли, играта се стартира от изпълнимия файл „Avija.exe“, който се намира в основната папка.



Фиг. 4.1. Изпълним файл в директорията на играта

## 4.2 Настройка на клавишите за игра

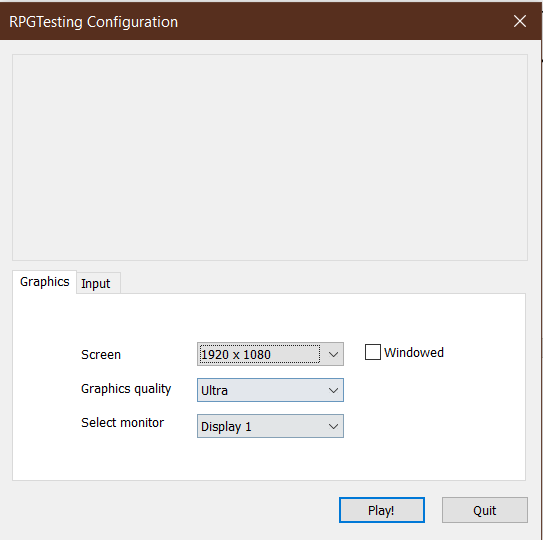
След като стартира играта, играчът има възможност за настройка на клавишите с които ще играе играта. Тази функция е вградена във всички игри, които са направени на игровия двигател „Unity“. Могат да се настроят клавишите за движение, скок, атака, скриване и показване на мишката, изстрелване на магия, блокиране на атака, избягване на атака и др.



Фиг. 4.2. Настройка на клавишите в играта

## 4.3 Настройка на графиката на играта

Играчът има възможността и да настрой графиката на играта. Настройките включват: промяна на резолюцията на екрана, промяна на качеството на графиката в играта, избиране на екран и избор за зареждане на играта в режим на прозорец.



Фиг. 4.3. Настройка на графиката в играта

## 4.3 Навигация в играта

Играта започва с главното меню. В него потребителят има няколко възможности. Първата е да започне нова игра. Ако натисне бутона за започване на нова игра: „New Game“, той бива пренесен в началната зона, без никакви пари или предмети и е първо ниво.

В случай, че играчът има запазен файл с прогреса си, в менюто се появява допълнителен бутон „Continue“ и чрез натискането му, играчът може да продължи от последния запазен момент в играта. Чрез бутона „Exit“, играчът излиза от играта.



Фиг. 4.4. Главно меню в играта

След като играчът започне своята игра, той трябва да използва клавишите, които е задал предварително за да играе. В случай, че иска да паузира играта, играчът може да натисне „ESCAPE“ копчето на клавиатурата си, като по този начин играта ще се паузира и ще се отвори друго меню. Там играчът може да промени някои настройки, или да се върне към началното меню.



Фиг. 4.6. Снимка на менюто което се отваря при паузиране на играта

# Заключение

Настоящата дипломна работа представлява успешно разработена игра на игровия двигател „Unity“. Всички изисквания, заложени в заданието са изпълнени, като дори има допълнителни функционалности, с цел, подобрение на цялостния облик на играта. Логиката е разработена, така че да може да се модифицира с минимални усилия в бъдеще. Използвани са различни конвенции за структуриране и разработка на игри. В бъдеще могат да се добавят различни визуални ефекти, както и много други 3д модели, които да допълнят визията на играта и да създадат усещането за една реалистична околна среда. Сегашната имплементация на системите за мисии и предмети позволява лесното създаване на нови елементи и добавянето им към играта. Чрез вече съществуващата логика може да се добави ново съдържание, които да я направят по-интересна за крайния потребител. Една от възможностите за подобрение е добавянето на режим на игра в мрежа, при който различни играчи играят заедно.

# Използвана литература

1. [**https://en.wikipedia.org/wiki/Game\_engine**](https://en.wikipedia.org/wiki/Game_engine)
2. [**https://www.informit.com/articles/article.aspx?p=2031153**](https://www.informit.com/articles/article.aspx?p=2031153)
3. [**https://www.pubnub.com/blog/comparing-game-engines-unity-unreal-corona-gamemaker/**](https://www.pubnub.com/blog/comparing-game-engines-unity-unreal-corona-gamemaker/)
4. [**https://www.gamedesigning.org/engines/unity-vs-godot/**](https://www.gamedesigning.org/engines/unity-vs-godot/)
5. [**https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e**](https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e)

# Съдържание

[Увод 1](#_Toc34255580)

[ПЪРВА ГЛАВА. 2](#_Toc34255581)

[МЕТОДИ И ТЕХНОЛОГИИ ЗА СЪЗДАВАНЕТО НА ВИДЕО ИГРИ 2](#_Toc34255582)

[1.1 Технологии за разработване на видео игри 2](#_Toc34255583)

[1.1.1 Разработка на игри от нулата 2](#_Toc34255584)

[1.1.2 Използване на игрови двигател 2](#_Toc34255585)

[1.1.3 Създаване на собствен игрови двигател 3](#_Toc34255586)

[1.1.4 Компоненти на игровите двигатели 3](#_Toc34255587)

[1.1.4.1 Двигател за рендиране 3](#_Toc34255588)

[1.1.4.2 Физичен двигател 4](#_Toc34255589)

[1.1.4.3 Звуков двигател 4](#_Toc34255590)

[1.1.4.4 Изкуствен интелект 5](#_Toc34255591)

[1.2 Съществуващи програмни продукти 5](#_Toc34255592)

[1.2.1 Unity 3D 5](#_Toc34255593)

[1.2.2 Unreal Engine 6](#_Toc34255594)

[1.2.3 CryEngine 7](#_Toc34255595)

[1.2.4 GameMaker Studio 8](#_Toc34255596)

[1.2.5 Godot 8](#_Toc34255597)

[ВТОРА ГЛАВА. 9](#_Toc34255598)

[ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА ВИДЕО ИГРАТА 9](#_Toc34255599)

[2.1 Функционални изисквания към видео играта 9](#_Toc34255600)

[2.2 Избор на програмни средства и развойната среда 9](#_Toc34255601)

[2.2.1 Unity 9](#_Toc34255602)

[2.2.2 Microsoft Visual Studio 10](#_Toc34255603)

[2.3 Структура на видео играта 10](#_Toc34255604)

[2.3.1 Проектиране на главния герой 11](#_Toc34255605)

[2.3.3 Проектиране на системата за мисии 11](#_Toc34255606)

[2.3.4 Проектиране на система за създаване на предмети 12](#_Toc34255607)

[2.3.5 Проектиране на останалите персонажи в играта и интеракциите с тях 13](#_Toc34255608)

[2.3.6 Проектиране на противниците 13](#_Toc34255609)

[2.3.7 Проектиране на вода и водно придвижване 14](#_Toc34255610)

[2.3.8 Проектиране на система за запазване на прогреса 14](#_Toc34255611)

[ТРЕТА ГЛАВА. 15](#_Toc34255612)

[РЕАЛИЗАЦИЯ НА ВИДЕО ИГРАТА 15](#_Toc34255613)

[3.1 Базови скриптове 15](#_Toc34255614)

[3.1.1 Имплементация на Сингълтън модела 15](#_Toc34255615)

[3.1.2 Компоненти за управление на потребителския интерфейс 16](#_Toc34255616)

[3.1.2.1 Структура на панелите за информация 17](#_Toc34255617)

[3.2 Реализация на камерата от трето лице 17](#_Toc34255618)

[3.2.1 Разположение на камерата спрямо героя 17](#_Toc34255619)

[3.2.2 Движение на камерата около героя 18](#_Toc34255620)

[3.3 Реализация на управлението на главния герой 19](#_Toc34255621)

[3.3.1 Завъртане на героят в посоката на камерата 19](#_Toc34255622)

[3.3.2 Движение на героя 20](#_Toc34255623)

[3.4 Реализация на системата за взаимодействие 21](#_Toc34255624)

[3.4.1 Засичане на възможност за интеракция 22](#_Toc34255625)

[3.3.2 Потребителски интерфейс на интеракциите 23](#_Toc34255626)

[3.4 Имплементация на взаимозависими мисии 24](#_Toc34255627)

[3.5 Имплементация на система за създаване на предмети 29](#_Toc34255628)

[3.5.1 Екран за създаване на предмети 30](#_Toc34255629)

[3.6 Имплементация на противници 32](#_Toc34255630)

[3.6.1 Движение на противниците 32](#_Toc34255631)

[3.6.2 Водене на битки 34](#_Toc34255632)

[3.6.2.1 От страна на противниците 34](#_Toc34255633)

[3.6.2.2 От страна на играча 35](#_Toc34255634)

[3.5 Имплементация на система за водно придвижване 36](#_Toc34255635)

[ЧЕТВЪРТА ГЛАВА. 38](#_Toc34255636)

[РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ 38](#_Toc34255637)

[4.1 Изисквания към компютърната конфигурация 38](#_Toc34255638)

[4.1.1 Операционна система 38](#_Toc34255639)

[4.1.2 Хардуерни изисквания 38](#_Toc34255640)

[4.1.3 Инсталация на играта 38](#_Toc34255641)

[4.2 Настройка на клавишите за игра 39](#_Toc34255642)

[4.3 Настройка на графиката на играта 40](#_Toc34255643)

[4.3 Навигация в играта 41](#_Toc34255644)

[Заключение 44](#_Toc34255645)

[Използвана литература 45](#_Toc34255646)

[Съдържание 46](#_Toc34255647)