

Глава 0

Увод

Глава 1

Разработка на игри с Unreal Engine 4

Глава 2

Подбор на технологии за разработка на игри с Unreal Engine 4

2.1 Функционални изисквания към разработката

Разработката на играта има за цел да постави основите на качествено разработен проект, който ще позволява лесно надграждане върху него.

2.1.1 Изисквания към оръжията

- Параметри на оръжията, които позволяват висока разновидност.
- Функция за пълнител и презареждане на пълнителя.
- Функция за пиближаване (zoom).
- Различни режими на стрелба.
 - Полуавтоматичен (Semi-automatic).
 - Залпов (Burst).
 - Автоматичен (Automatic).
- Стрелба и откриване на удари чрез лъчево проследяване.

2.1.2 Изисквания към системата за кръв

- Щит (Shield), които се възстановява за определено време, след поемане на удар.
- Живот (Кръв, Health), който не се възстановява от самосебе си.
- Предмет, който може да бъде използван от играча, ако животът му е под максималния.

2.1.3 Изисквания към слотовете за оръжия

- Възможност за пазене на 2 оръжия и смяна между двете по време на игра.

2.1.4 Изисквания към изкуствения интелект

- TO DO

2.2 Избор на Unreal Engine 4

2.3 Избор на програмен език

За разработка на игри в Unreal Engine 4 се използва C++. При стандартна работа със C++ в Unreal Engine се използват макрота, които позволяват по-лесна работа с класовете и дефиниране на типовете нужни за работа с обектите в Unreal Engine.

2.4 Избор на среда за разработка

2.4.1 Игрови двигател (Game Engine)

За игрови двигател е, естествено, избран Unreal Engine, като по-точно версия 4.27. Разликата между версия 4.27 и 4.26 е най-вече във визуалните подобрения, но най-важното за тази версия, отнасящо се до

разработката на проекта е новите алгоритми за компресиране на файловете нужни на Unreal Engine и файловете използвани от разработчиците на проекта. Това прави крайния продукт по-малък от страна на размер, скорост на зареждане и предаване на мрежови пакети.

Unreal Engine е избран вместо други игрови двигатели поради следните причини:

- Използва се C++ за разработка на функционалностите.
- Кодът му е свободно достъпен за всеки разработчик.
- Разработен е с 3D игри в предвид и позволява лесна работа в 3D пространството.
- Има всичко нужно за разработването на игри и в повечето случаи не се нуждае от външни добавки за да може да върши добра работа.

Разликата между Unreal Engine и други игрови двигатели, Unity например, е че Unreal Engine е направен за 3D игри. Unity е фокусиран върху 2D и мобилни игри, а Unreal Engine - към 3D игри за настолни компютри и конзоли. Разликата между Unreal Engine и други по-известни и големи игрови двигатели, е това че кодът на Unreal Engine е напълно отворен за четене от всеки разработчик на продукти с Unreal Engine. Това позволява по-лесно откриване на проблеми в кода, по-добро разбиране базовите класове, върху които работят всички други части на Unreal Engine.

2.4.2 IDE (Integrated Development Environment)

За среда на разработка на проекта е избран Visual Studio 2019 Community Edition. Той се интегрира лесно с Unreal Engine, дебъгването с него става лесно и компилирането на проекта става сравнително бързо. IntelliSense, който е част от Visual Studio помага на по-бърза разработка, завършването на кода работи добре и се интегрира добре с начина на писане на код за Unreal Engine.

2.4.3 Среда за контрол на версиите

За контрол на версиите е избран github, както и интеграцията му с git LFS (Large File Storage). Git LFS позволява качването на големи файлове, най-често бинарни, които се използват често при работата с Unreal Engine. Github позволява лесен начин за интегриране на нови свойства и промени на проекта, както и проверка на минали версии на проекта.

Глава 3

Разработка на проекта

3.1 AMultiplayerFPSFirearm

3.2 AMultiplayerFPSCharacter

3.3 UMultiplayerFPSHealthSystem

3.4 AHealthPickup

Този клас наследява AActor като дефинира предмет, който може да бъде поставен на картата и може да бъде взет от играча. Той се състои от два компонента - колизионна кутия и модел. За основен компонент е избран моделът, като кутията е прикрепна към основният компонент - моделът и позицията на кутията се смята спрямо моделът.

```
1 UCLASS()
2 class MULTIPLAYERFPS_API AHealthPickup : public AActor
3 {
4     GENERATED_BODY()
5
6 public:
7     AHealthPickup();
8
9     UPROPERTY(VisibleAnywhere, Category = "Mesh")
10    UStaticMeshComponent* HealthPickupMesh;
11
12    UPROPERTY(EditAnywhere, Category = "Mesh")
13    class UBoxComponent* BoxCollision;
14
```

```

15 protected:
16     virtual void BeginPlay() override;
17
18 public:
19     virtual void Tick(float DeltaTime) override;
20
21     UFUNCTION()
22     void OnBeginOverlap(class UPrimitiveComponent* HitComponent,
23         class AActor* OtherActor, class UPrimitiveComponent* OtherComp
24         , int32 OtherBodyIndex, bool bFromSweep, const FHitResult&
25         SweepResult);
26
27     UFUNCTION(Server, Reliable)
28     void ServerDestroyHealthPickup();
29
30     UFUNCTION(Client, Reliable)
31     void ClientDestroyHealthPickup();
32 };

```

Код 3.1: Дефиниция на класа AHealthPickup

```

1 HealthPickupMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("
    SM_HealthPickup"));
2 HealthPickupMesh->SetCollisionEnabled(ECollisionEnabled::
    QueryAndPhysics);
3
4 RootComponent = HealthPickupMesh;
5
6 BoxCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("Box
    Collision"));
7 BoxCollision->SetBoxExtent(FVector(75.0f, 60.0f, 60.0f));
8 BoxCollision->SetupAttachment(RootComponent);
9 BoxCollision->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);

```

Код 3.2: Инициализация на компонентите в AHealthPickup

В код 3.1 са дефинирани пет функции и един конструктор. В конструктора се инициализират компонентите (показано в код 3.2), BeginPlay се изпълнява след започване на tick на нивото, Tick се изпълнява на всеки кадър (frame), OnBeginOverlap се изпълнява след като кутията за колизия се пресече с друг обект за колизия, и Destroy функциите се викат съответно на сървъра и клиента.

В BeginPlay се извиква макро за извикване на AddDynamic() върху динамични делегати за многократно предаване.

Глава 4

РЪКОВОДСТВО на потребителя

4.1 Инсталиране на проекта

4.2 Работа с проекта

4.3 Стартиране на проекта

Глава 5

Заключение

Съдържание

0	Увод	1
1	Разработка на игри с Unreal Engine 4	2
2	Подбор на технологии за разработка на игри с Unreal Engine 4	3
2.1	Функционални изисквания към разработката	3
2.1.1	Изисквания към оръжията	3
2.1.2	Изисквания към системата за кръв	4
2.1.3	Изисквания към слотовете за оръжия	4
2.1.4	Изисквания към изкуствения интелект	4
2.2	Избор на Unreal Engine 4	4
2.3	Избор на програмен език	4
2.4	Избор на среда за разработка	4
2.4.1	Игрови двигател (Game Engine)	4
2.4.2	IDE (Integrated Development Environment)	5
2.4.3	Среда за контрол на версиите	6
3	Разработка на проекта	7
3.1	AMultiplayerFPSFirearm	7
3.2	AMultiplayerFPSCharacter	7
3.3	UMultiplayerFPSHealthSystem	7
3.4	AHealthPickup	7
4	Ръководство на потребителя	9
4.1	Инсталиране на проекта	9
4.2	Работа с проекта	9
4.3	Стартиране на проекта	9

