МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах
(название института полностью)

Кафедра/департамент «Информационные системы»
(наименование кафедры/департамента полностью)
09.03.02 Информационные системы и технологии
(код и наименование направления подготовки/специальности)
Информационные системы и технологии
(наименование профиля/специальности)

КУРСОВАЯ РАБОТА

по дисциплине
Объектно-ориентирование программирование
(наименование дисциплины)
на тему **Создание компьютерной игры противоборство-shooter**

Выполнили: обучающиеся
группы: ИС/б-20-2-о

А.А. Воронухин
А.Н. Филозоп
(инициалы, фамилия)

« ___» _____ 20 22 г.

Научный руководитель:

_____
(инициалы, фамилия)
« _____» _____ 20 22 г.

Оценка _____

« ___» _____ 20 22 г.

Севастополь
20 22

АННОТАЦИЯ

Назначением текущего документа является описание процесса проектирования и реализации программного продукта - игровая симуляция типа противоборство-shooter. В документе описывается процесс абстрагирования, выделения классов, их проектирование; выделения состояний объектов, потоков данных между объектами; определения жизненного цикла как всей программы, так и её части — отдельно взятого объекта. После проектирования описывается процесс создания приложения на одном из языков программирования и тестирования готового продукта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках настоящей курсовой работы ведется разработка программы на основании технического задания, выданного организацией Севастопольский государственный университет. Дата выдачи задания: 17.02.2022.

Целями курсовой работы являются изучение современного подхода к программированию на основе объектно-ориентированной технологии, приобретение навыков написания программ на языке с поддержкой ООП (в частности C++) на примере написания программы согласно варианту задания.

Для достижения каждой из поставленных целей были поставлены следующие задачи:

- выбор варианта задания и языка реализации, детализация поставки задачи;
- абстрагирование, разработка классов и их иерархии;
- написание текста (кодирование) разработанных классов на выбранном языке;
- разработка тестовых примеров;
- тестирование и отладка программы;
- разработка программных документов в соответствии с действующими стандартами.

По окончанию выполнения курсовой работы должны быть получены навыки решения задач объектно-ориентированным подходом.

СПИСОК ИСПОЛНИТЕЛЕЙ

Текущая курсовая работа была исполнена студентами:

– Воронухин Александр Александрович, группа ИС/б-20-2-о. Вклад: разработка идеи, геймдизайн, пользовательский интерфейс, написание кода.

– Филозоп Алексей Николаевич, группа ИС/б-20-2-о. Вклад: документирование, геймдизайн, написание кода.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Цель разработки

Целью разработки является создание стратегической игры типа «Противоборство. Shooter».

1.2 Описательная постановка задачи

Требуется создать игру типа «Противоборство. Shooter». Цель игры заключается в следующем: главному герою (далее — ГГ) требуется набрать наибольшее количество очков за отведённое время (60 секунд) и получить наибольшее значение точности.

Помимо ГГ на уровне также должны находиться интерактивные объекты: поднимаемые объекты и движущиеся объекты. Очки за поднимаемые объекты будут выдаваться или отниматься только в случае поднятия (физического контакта) объекта главным героем. Очки за движущиеся объекты будут выдаваться или отниматься только в случае ранения или убийства объекта главным героем с помощью пуль.

Значение точности будет высчитываться следующим образом (1.1):

$$Точность = \frac{КоличествоПопаданий}{КоличествоВыстрелов} \quad (1.1)$$

Попаданием будет считаться выстрел, ранящий или убивший движущийся объект.

1.3 Условия выполнения и ограничения

Программа должна быть реализована на языке, полностью поддерживающем объектно-ориентированный подход. Также игра должна будет поддерживать ввод с клавиатуры и мыши.

2 ПРОЕКТНОЕ РЕШЕНИЕ

2.1 Абстрагирование и выделение объектов

При разработке требований были выделены классы, описанные ниже.

Для ГГ будет использован класс FPC. В соответствии с главной целью игры класс будет иметь следующие свойства:

- количество поднятых объектов (KillsHittableItems);
- количество попаданий по движущимся объектам (TotalHits);
- количество выстрелов (TotalShots);
- счёт (Score);
- время игры (PlayTime);
- времени раунда прошло (PassedTime).

Каждый ГГ имеет возможность стрелять, поэтому для класса будет определён метод Shoot.

Каждый объект, с которым может взаимодействовать ГГ, является экземпляром класса BaseThing. Для него определены методы: получить урон (TakeDamage), умереть (EffectOnDeath), — а также свойства: очки здоровья (Health), очки за убийство объекта (OnDeathPoints).

Так как каждый интерактивный объект принадлежит к одному из двух типов — подбираемый и движущийся —, были выделены классы HittableThing и ShootableThing соответственно. Класс HittableThing имеет свой метод TakeDamage. Класс ShootableThing имеет свой метод TakeDamage, а также добавляет свои свойства:

- очки за ранение объекта (OnHitPoints);
- скорость движения (Speed).

Для каждого объекта, с которым может взаимодействовать ГГ определены следующие классы:

- поднимаемый объект «Монетка» (Coin);
- поднимаемый объект «Ёж» (Hedgehog);
- движущийся объект «Друг» (Friend);
- движущийся объект «Враг» (Emeny).

За взаимодействие с «Ежами» и «Друзьями» очки ГГ уменьшаются, за взаимодействие с «Монетками» и «Врагами» очки увеличиваются.

2.2 Построение иерархии классов

На основании предыдущего раздела построено графическое отображение иерархии классов, представленное на рисунке А.1.

2.3 Построение информационной модели

Построена диаграмма классов, представленная на рисунке А.2.

2.4 Описание жизненного цикла программы

Жизненный цикл движущегося объекта представлен на рисунке А.3.

2.5 Описание жизненного цикла одного из объектов

Жизненный цикл движущегося объекта представлен на рисунке А.4.

2.6 Диаграмма переходов состояний

Диаграмма переходов состояний представлена на рисунке А.5.

2.7 Диаграмма потоков данных и действий

Диаграмма потоков данных и действий представлена на рисунке А.6.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Обоснование выбора системы программирования

Было принято решение реализовать проект в игровом движке Unreal Engine 4 (далее — UE4). Причины выбора приведены ниже:

- предоставление готовой системы работы с трёхмерными моделями;
- автоматическая оптимизация программы для потребителя;
- возможность создания кроссплатформенного проекта;
- язык программирования C++ как основной язык разработки программы.

Для редактирования и компилирования кода была выбрана интегрированная система разработки Microsoft Visual Studio 22 Community Edition. Причины выбора следующие:

- интегрирование с UE4;
- предоставление различных средств для создания качественного кода, в т.ч. профилирование, автодополнение, отладка.

Операционная система Windows является платформой, для которой будет вестись разработка.

3.2 Описание реализации основных классов и их методов

В ходе разработки были созданы классы, аналогичные описанным в разделе 2.1: ABaseThing, AHittableThing. AShootableThing, ACoin, AHedgehog, AFriend, AEmeny, AFPC. Приставка «A» обозначает то, что класс наследуется от предоставляемого игровым движком класса Actor. Далее будет описано о применении основных принципов ООП.

Наследование было применено следующим образом. Для каждого объекта базовым является количество очков, начисляемых ГГ при его уничтожении. Для обстреливаемых объектов являются обязательными свойства скорость, здоровье, количество очков, начисляемых ГГ при попадании. Таким образом, каждому классу, наследуемому от ABaseThing, будет присуще свойство OnDeathPoints, а каждому наследуемому от AShootableThing классу — свойства Health,

OnHitPoints и Speed.

Принцип инкапсуляции был реализован следующим образом. Каждый из экземпляров класса AShootableThing имеет здоровье Heath. Так как свойство объявлено в разделе protected, никто, кроме самого класса и его наследников, читать и изменять значение поля не может. Но чтобы объект мог получить урон, существует метод класса TakeDamage, одним из параметров которого является величина урона. К классу AFPC также применён метод инкапсуляции: закрытое свойство Score_ можно изменить только через открытую функцию ChangePoints. Таким образом, чтобы изменить значение закрытого свойства, необходимо воспользоваться открытым методом.

Принцип полиморфизма использован следующим образом. В базовом классе объявлен открытый чисто виртуальный метод TakeDamage. В наследнике AHittableThing этот метод переопределён и смысл его только лишь в том, чтобы вызвать метод родительского класса EffectOnDeath. В наследнике AShootableThing метод переопределён и имеет уже другой смысл: при получении урона отнимать здоровье у объекта и начислять OnHitPoints ГГ, а в случае смертельного удара — вызвать метод базового класса EffectOnDeath. Таким образом, независимо от того, по кому было совершено попадание, при вызове BaseThing::TakeDamage будет гарантированно вызван метод, принадлежащий поражённому объекту.

3.3 Общее описание взаимодействия программных продуктов

Программа состоит из восьми модулей. Каждому модулю соответствует созданный класс. Только между модулями с классами AFPC и ABaseThing есть взаимодействие: AFPC вызывает метод TakeDamage у ABaseThing при попадании, а AbaseThing – метод changePoints класса AFPC для изменения очков.

3.4 Интерфейс пользователя

На рисунках 3.1 — 3.5 представлен игровой процесс.

На рисунке 3.1 показано главное меню — экран, показывающийся игроку

при заходе в игру или после завершения раунда. В главном меню игрок может выбрать любой из трёх представленных режимов: «Полигон», игровой режим 1 «Только враги», игровой режим 2 «Враги и друзья» —, или выйти из игры.

Рисунок 3.1 — Главное меню

Рисунок 3.2 демонстрирует пример обзора игрока на уровне в режиме «Полигон». Пользовательский интерфейс состоит из: оставшееся время раунда вверху посередине (в данном режиме время не ограничено); счёт игрока и точность в левой верхней части экрана. На уровне можно встретить все типы объектов, а также важную игровую информацию.

Рисунок 3.3 демонстрирует экран пользователя в режиме «Только враги». Интерфейс пользователя в этом режиме аналогичен представленному в «Полигоне», но вместо «**infinite**» отображения реальное оставшееся время раунда. Движущимися объектами является только экземпляры класса Emeny.

Рисунок 3.4 демонстрирует экран игрока в режиме «Враги и друзья». Режим аналогичен «Только враги». Отличием является то, что движущимися объектами являются экземпляры классов Emeny и Friend.

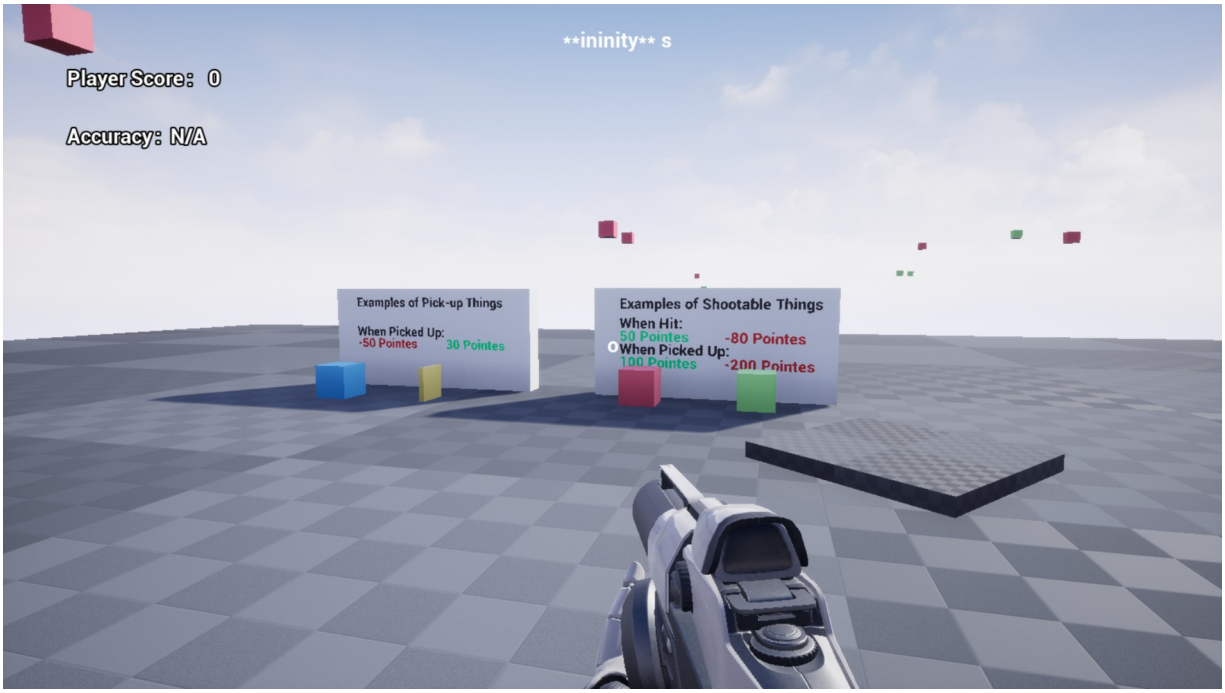

Рисунок 3.2 — Режим «Полигон»

Рисунок 3.3 — Режим «Только враги»

Рисунок 3.4 — Режим «Враги и друзья»

В режимах «Только враги» и «Враги и друзья» по окончанию времени игра будет приостановлена. Рисунок 3.5 демонстрирует экран пользователя. На экране пользователь видит посередине результаты своей игры: итоговый счёт, количество попаданий и точность. По нажатию на кнопку «Main Menu» игрока вернёт в главное меню.

Рисунок 3.5 — Экран подведения итогов раунда

3.5 Критерии качества программной системы

К основным критериям качества программной системы были отнесены:

- функциональность;
- надёжность;
- эффективность;
- эргономичность;
- модифицируемость;
- мобильность.

ВЫВОДЫ

При выполнении курсовой работы были выполнены следующие задачи. Было проведено абстрагирование, разработаны классы и их иерархии. Также был написан код программы на языке программирования C++, результатом чего является понимание особенностей реализации объектно-ориентированного подхода в рамках данного языка программирования. Были проведены тестирование и отладка программы, что способствовало изучению инструментов тестирования и отладки, предоставляемых системами Unreal Engine 4 и Visual Studio 22. Заключающим этапом стала разработка данного документа.

Итогом завершения каждой из поставленных задач является достижение цели — изучение современного подхода к программированию на основе объектно-ориентированной технологии. Результатом достижения цели являются текущий документ и программа. В программе реализованы все основные принципы ООП (инкапсуляция, наследование, полиморфизм), о чём написано в разделе 3.2.

Полученные навыки закрепили знания, полученные в ходе изучения курса ООП, что позволит применять их при решении любых задач.

ПЕРЕЧЕНЬ ССЫЛОК

1. А. Куксон Разработка игр на Unreal Engine 4 / А. Куксон, Р. Даулингсока, К. Крамплер; пер. с англ. М.А. Райтмана — М.: Эксмо, 2019. — 528 с.

2. Барков, И.А. Объектно-ориентированное программирование [Электронный ресурс] : учебник / И.А. Барков. — Электрон. дан. — Санкт-Петербург : Лань, 2019. — 700 с. — Режим доступа: https://e.lanbook.com/book/119661.

3. Г. Шилдт C++ для начинающих. Серия «Шаг за шагом» / Г. Шилдт; пер. с англ. — М.: ЭКОМ Паблишерз, 2013. — 643 с.

4. Официальная документация к Unreal Engine 4.27 [Электронный ресурс] — Режим доступа: https://bit.ly/3NrRZ3x.

5. Р. Вайнер. C++ изнутри / Р.Вайнер. Л.Пинсон. — Киев: НПИФ "Диа-Софт", 1993. — 304 с.

6. R. Grimm C++20 / R. Grimm — Режим доступа: https://leanpub.com/c20.

7. С. Сантелло Разработка RPG в Unreal Engine / С. Сантелло, А. Стэнгер; пер. с англ. — М.: XBeria, 2017. — 358 с.

8. У. Шериф Изучаем C++ создавая игры в UE4 / У. Шериф; пер. с англ. — М.: Packt, 2016. — 301 с.

9. Х. М. Дейтел, П. Дж. Дейтел. Как программировать на C++ (полное издание) / Х. М. Дейтел, П. Дж. Дейтел.; под ред. В.В.Тимофеева. — М.: Бином, 2008. — 1454 с.

10. Цикл статей по Unreal Engine 4 на HABR [Электронный ресурс] — Режим доступа: https://bit.ly/3xmLKbo.

ПРИЛОЖЕНИЕ А

UML-диаграммы

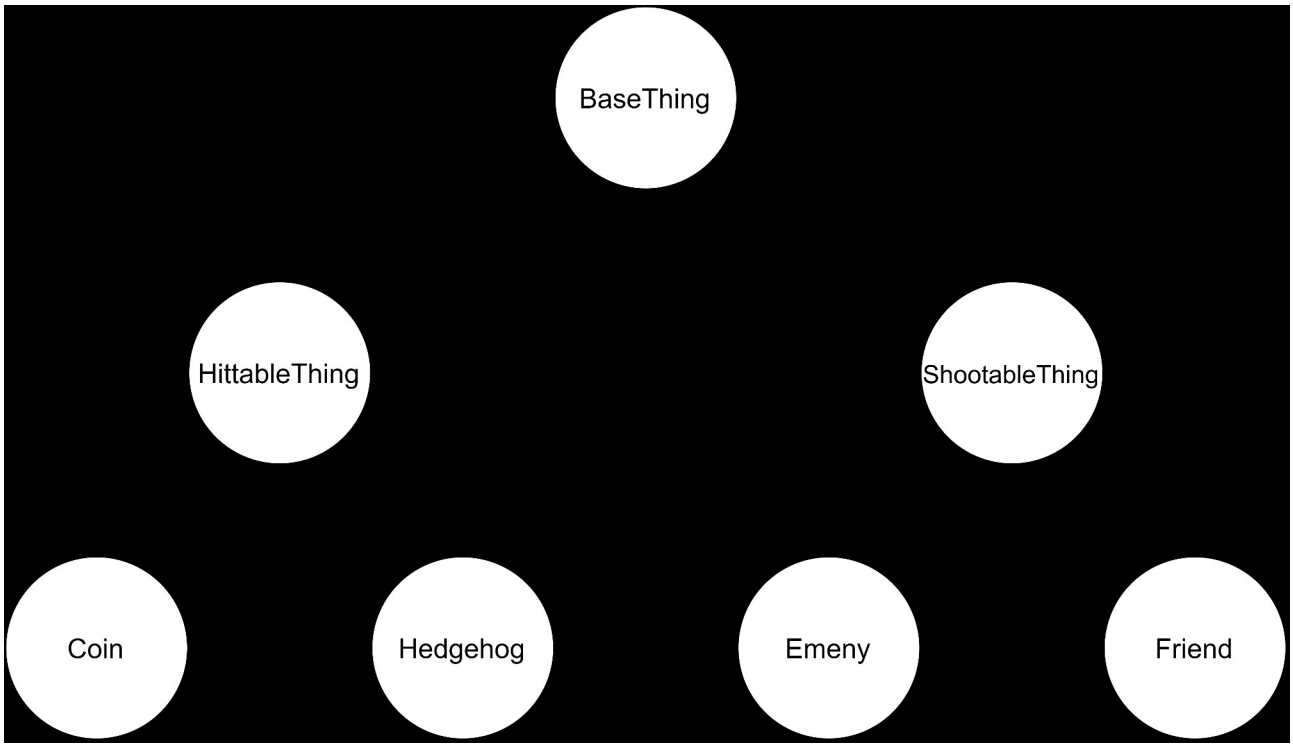

Рисунок А.1 — Иерархия классов

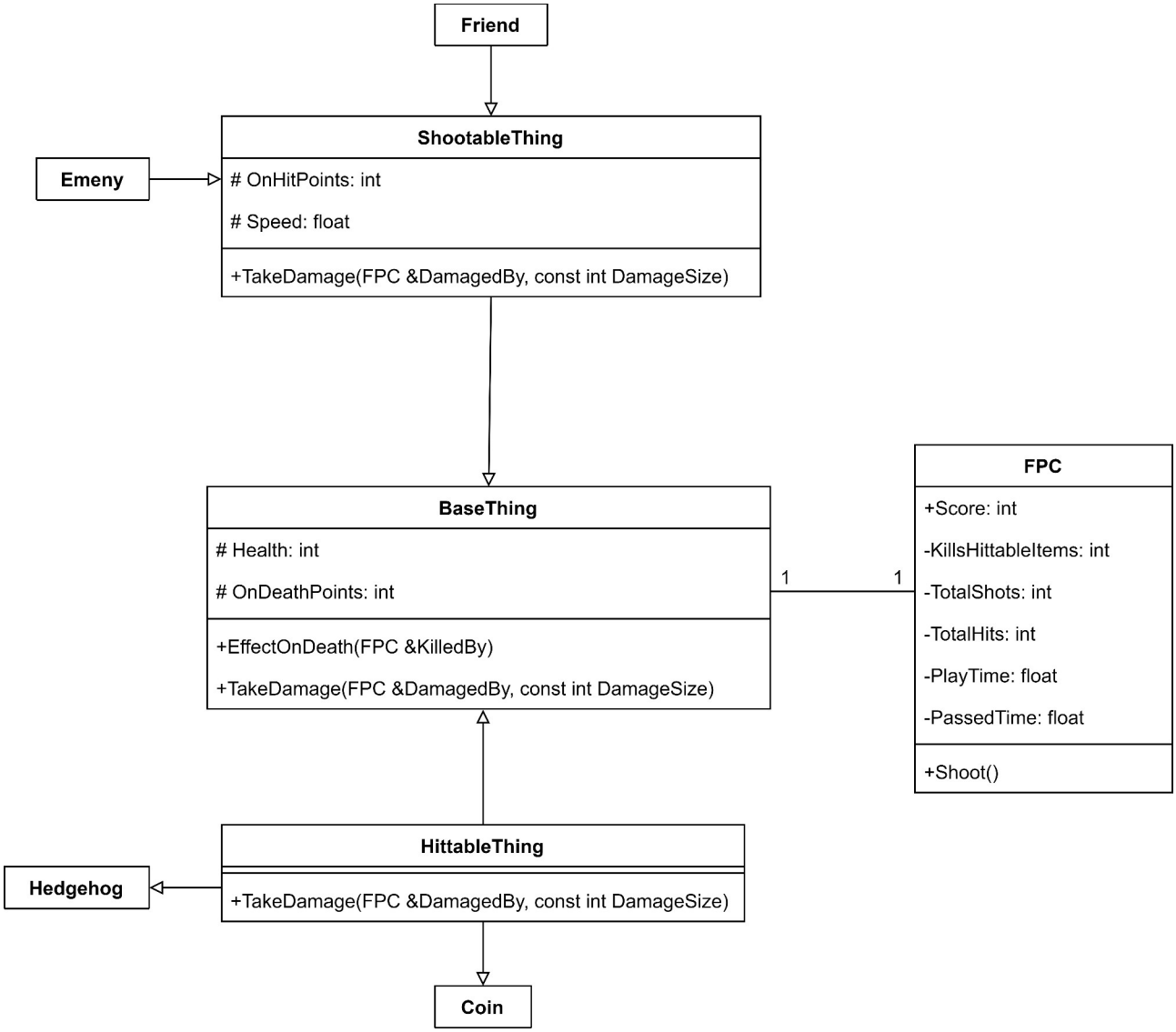

Рисунок А.2 — Диаграмма классов

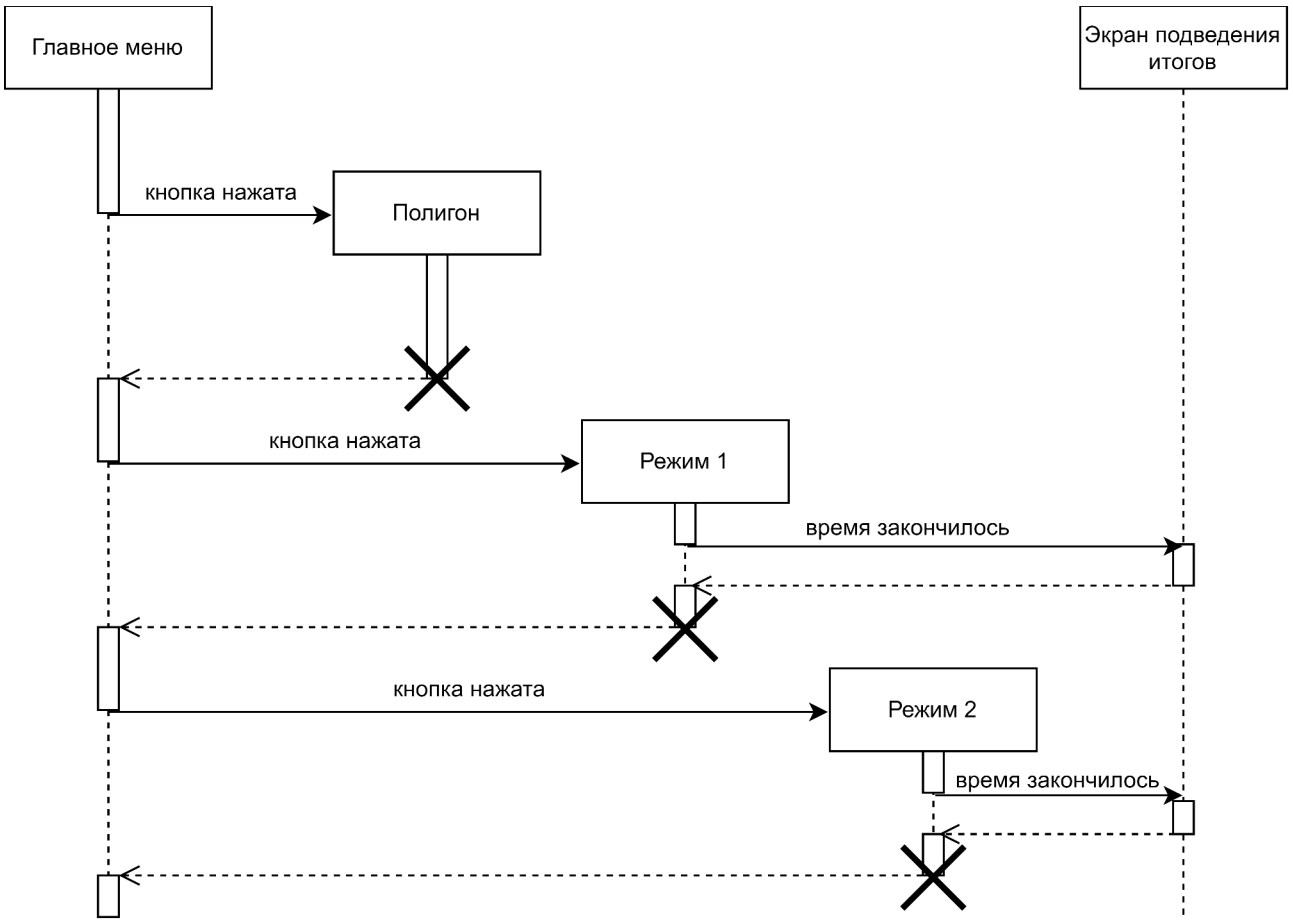

Рисунок А.3 — Жизненный цикл программы

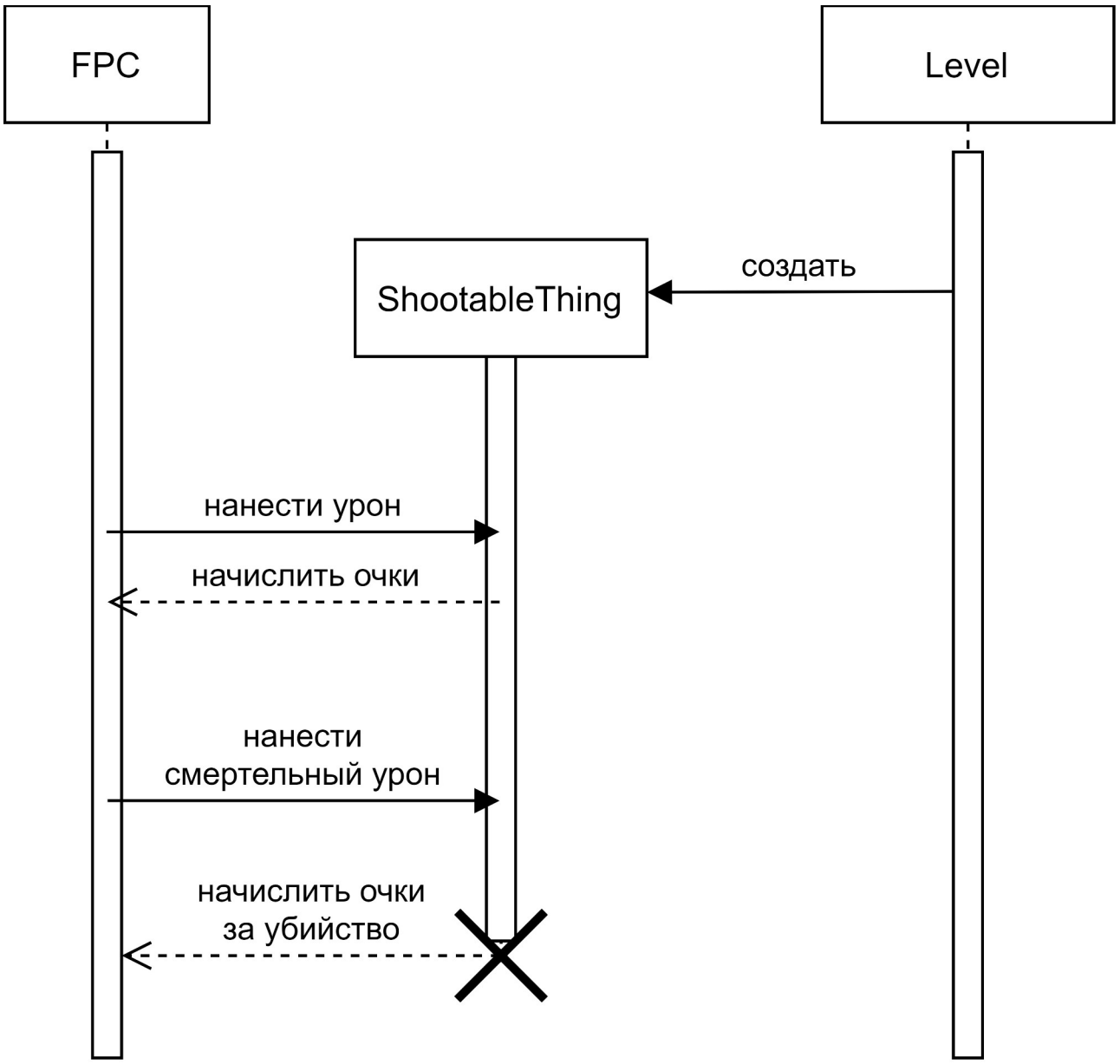

Рисунок А.4 — Жизненный цикл объектов класса ShootableThing

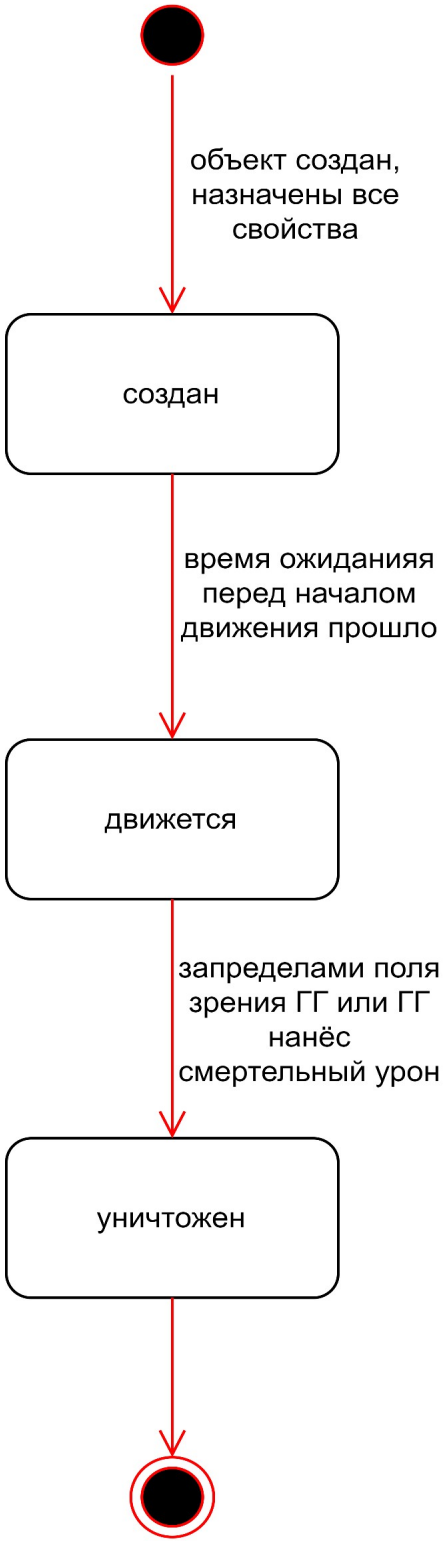

Рисунок А.5 — Диаграмма переходов состояний объектов класса ShootableThing

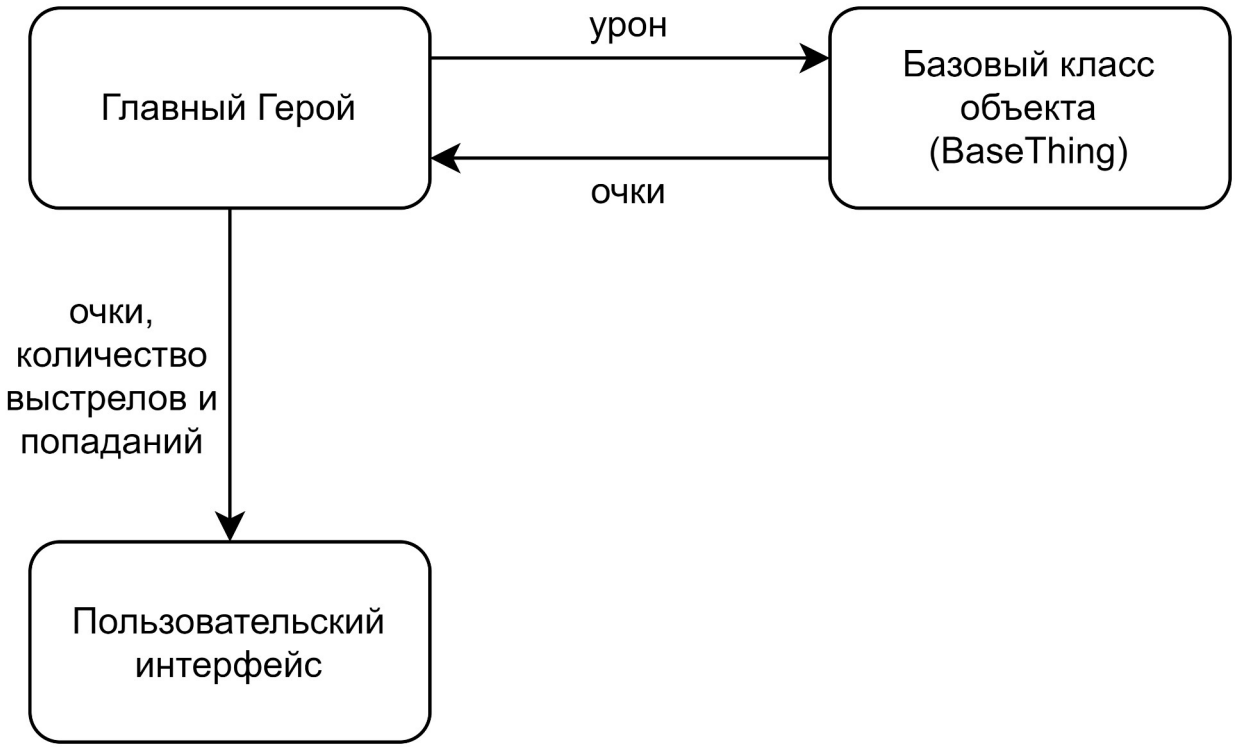

Рисунок А.6 — Диаграмма потоков данных

ПРИЛОЖЕНИЕ Б

Код

```
class ABaseThing : public AActor
{
    GENERATED_BODY()
    public:
        ABaseThing (int Health, const int OnDeathPoints);
        void EffectOnDeath (AFPC &KilledBy);
        virtual void TakeDamage (AFPC &DamagedBy, const int DamageSize) = 0;
    protected:
        int Health_;
        const int OnDeathPoints_;
};
class AHittableThing : public ABaseThing
{
    GENERATED_BODY()
    public:
        AHittableThing(const int OnDeathPoints);
        void TakeDamage (AFPC &DamagedBy, const int DamageSize);
        virtual void OnHit (Hit HitInfo);
        virtual void OnActorBeginOverlap (AActor &OtherActor);
};
class ACoin : public AHittableThing
{
    GENERATED_BODY()
    public:
        ACoin(const int OnDeathPoints);
};
class AHedgehog : public AHittableThing
{
    GENERATED_BODY()
    public:
        AHedgehog(const int OnDeathPoints);
};
class AShootableThing : public ABaseThing
{
    GENERATED_BODY()
    public:
        AShootableThing (const int Health, const int OnHitPoints, const int
OnDeathPoints);
        virtual void TakeDamage (AFPC &DamagedBy, const int DamageSize)
override;
    protected:
        const int   OnHitPoints_;
        const float Speed_;
        const float LifeSpan_;
};
class AFriend : public AShootableThing
{
    GENERATED_BODY()
    public:
        AFriend(const int Health, const int OnHitPoints, const int
OnDeathPoints);
};
```

```
class AEmeny : public AShootableThing
{
    GENERATED_BODY()
    public:
        AEmeny(const int Health, const int OnHitPoints, const int
OnDeathPoints);
};
class AFPC : public AFirstPersonCharacter
{
    GENERATED_BODY()
    public:
        AFPC(const float PlayTime);
        void Shoot();
        void OpenMainMenu();
        void ChangePoints(int value);
        virtual void OnTick (float DeltaSeconds) override;

    protected:
        int KillsHittableItems_;
        int TotalShoots_;
        int TotalHits_;
        float PassedTime_, PlayTime_;
        bool IsPolygon_;
        int Score_;
};
ABaseThing::ABaseThing (int Health, const int OnDeathPoints)
    : Super(), Health_(Health), OnDeathPoints_(OnDeathPoints)
{ }
void ABaseThing::EffectOnDeath (AFPC &KilledBy)
{
    KilledBy.ChangePoints (OnDeathPoints_);
    this->DestroyActor();
}
void ABaseThing::TakeDamage (AFPC &DamagedBy, const int DamageSize);
AHittableThing::AHittableThing (const int OnDeathPoints)
    : Super (1, OnDeathPoints)
{ }
void AHittableThing::TakeDamage (AFPC &DamagedBy, const int DamageSize)
{
    EffectOnDeath (DamagedBy);
}
void AHittableThing::OnHit (Hit HitInfo)
{
    auto a = Cast<AFPC> (HitInfo.Other);
    if (a != nullptr)
        TakeDamage (*a, 10000);
}
void AHittableThing::OnActorBeginOverlap (AActor &OtherActor)
{
    auto a = Cast<AFPC> (OtherActor);
    if (a != nullptr)
        TakeDamage (*a, 10000);
}
ACoin::ACoin (const int OnDeathPoints)
    : Super (OnDeathPoints)
{ }
AHedgehog::AHedgehog (const int OnDeathPoints)
    : Super (OnDeathPoints)
{ }
AShootableThing::AShootableThing (const int Health, const int OnHitPoints, const
int OnDeathPoints)
```

```
    : Super (Health, OnHitPoints, OnDeathPoints)
{ }
void AShootableThing::TakeDamage (AFPC &DamagedBy, const int DamageSize)
{
    Health_ -= DamageSize; DamagedBy.ChangePoints (OnHitPoints_);
    if (Health_ <= 0)
    { EffectOnDeath (DamagedBy); }
}
AFriend::AFriend (const int Health, const int OnHitPoints, const int
OnDeathPoints)
    : Super (Health, OnHitPoints, OnDeathPoints)
{ }
AEmeny::AEmeny (const int Health, const int OnHitPoints, const int
OnDeathPoints)
    : Super (Health, OnHitPoints, OnDeathPoints)
{ }
AFPC::AFPC (const float PlayTime)
    : Super(), PlayTime_(PlayTime), Score_(0), TotalShoots_(0), TotalHits_(0),
KillsHittableItems_(0)
{
    this->BindAxis (this, "Turn", &(Super::ControllerYawInput));
    this->BindAxis (this, "LookUp", &(Super::ControllerPitchInput));
    this->BindAxis (this, "MoveForward", &(Super::MovementInput));
    this->BindAxis (this, "MoveRight", &(Super::MovementInput));
    this->BindAction (this, "Jump", PRESSED, &(Super::Jump));
    this->BindAction (this, "Jump", RELEASED, &(Super::Jump));
    this->BindKey (this, "Escape", PRESSED, &(AFPS::OpenMainMenu));
    IsPolygon_ = true;
}
void AFPC::Shoot ()
{
    auto  CameraLocation = GetComponent(NAME("FirstPersonCamera"))-
>GetWorldLocation(),
          CameraForwardVector = GetComponent(NAME("FirstPersonCamera"))-
>GetForwardVector();
    auto Result = LineTraceByChannel (CameraLocation, CameraForwardVector *
50000 + CameraLocation);
    auto TracedActor = Cast<AShootableThing> (Result.HitActor);
    if (TracedActor != nullptr)
    { TracedActor->TakeDamage (*this, 40.0); TotalHits_ += 1; }
    TotalShoots_ += 1;
}
void AFPC::OpenMainMenu()
{
    GetWorld()->OpenLevel (NAME("LevelMainMenu"));
}
void AFPC::OnTick (float DeltaSeconds)
{
    PassedTime_ += DeltaSeconds;
    if (PassedTime_ >= PlayTime_)
    {
        if (!IsPolygon_)
        {
            GetWorld()→SetGamePaused(true); OpenMainMenu();
        }
    }
}
void AFPC::ChangePoints (value points)
{
    Score_ += value;
}
```