Министерство образования и науки РФ

Федеральное бюджетное образовательное учреждение

высшего профессионального образования

«Кузбасский государственный технический университет имени Т.Ф.Горбачева»

Кафедра прикладных информационных технологий

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

Пояснительная записка к курсовой работе по дисциплине Информационная безопасность для студентов специальности 230700.62Прикладная информатика в экономике

Выполнил: студент гр. ПИб-131  
А.В.Вайцеховский

Проверил: преподаватель  
А.А.Долгов

Кемерово 2015

Оглавление

[**1.** **ДИАГРАММЫ КЛАССОВ** 2](#_Toc442206153)

[**1.1.** **Диаграмма классов исходной программы** 2](#_Toc442206154)

[**1.2** **Диаграмма классов после выполнения заданий** 3](#_Toc442206155)

[**3.** **ИСПОЛЬЗОВАННЫЕ ТЕХНОЛОГИИ ООП** 5](#_Toc442206156)

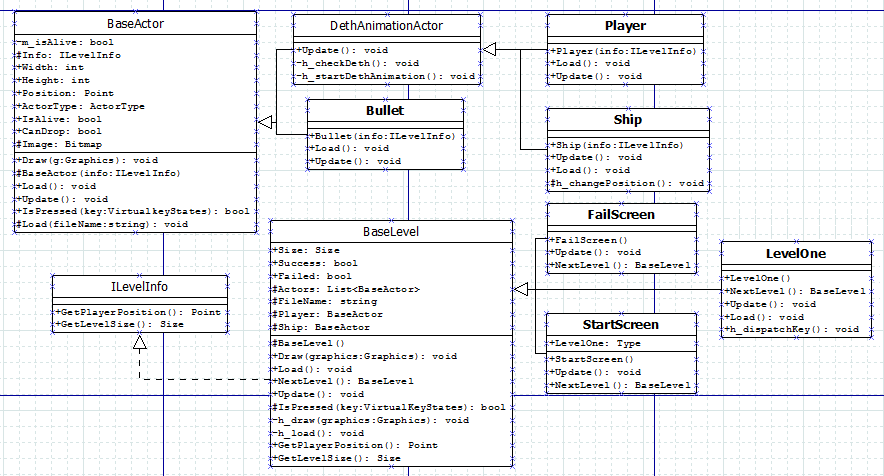
[**3.1.** **Наследование** 5](#_Toc442206157)

[**3.2.** **Инкапсуляция** 5](#_Toc442206158)

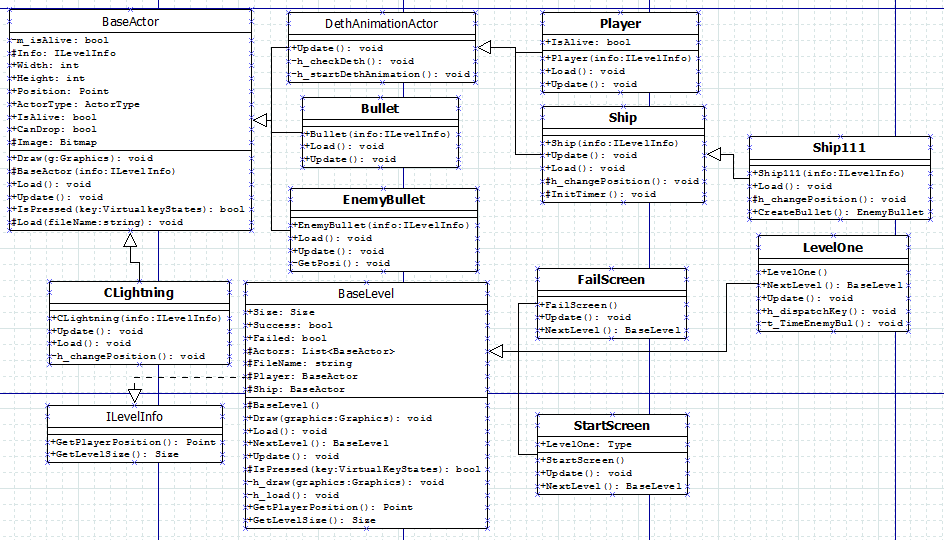
[**3.3.** **Полиморфизм** 5](#_Toc442206159)

[**4.** **НЕДОСТАТКИ ВЫПОЛНЕННОГО РЕШЕНИЯ** 7](#_Toc442206160)

1. **ДИАГРАММЫ КЛАССОВ**
   1. **Диаграмма классов исходной программы**



* 1. **Диаграмма классов после выполнения заданий**



1. **Описание ответственности добавленных классов**

**Ship111** – данный класс отвечает за второй вид кораблей противника.

**EnemyBullet** – данный класс отвечает за пули у вражеских кораблей.

**CLightning** - данный класс отвечает за молнию (является врагом для игрока и для кораблей-противников).

1. **ИСПОЛЬЗОВАННЫЕ ТЕХНОЛОГИИ ООП**
   1. **Наследование**

Это механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса или интерфейса. Потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами. Позволяет строить иерархии. Является одним из основных принципов объектно-ориентированного программирования. Ниже представлены примеры наследования в данном проекте:

* Класс DethAnimationActor – наследник класса BaseActor.
* Класс Bullet – наследник класса BaseActor.
* Класс EnemyBullet – наследник класса BaseActor.
* Класс Player – наследник класса DethAnimationActor.
* Класс Ship – наследник класса DethAnimationActor.
* Класс CLightning – наследник класса Player.
* Класс Ship111 – наследник класса Ship.
* Класс BaseLevel реализует интерфейс ILevelInfo.

На примере классов Ship и Ship1 можно рассмотреть, как наследование помогает избежать дублирования кода. Оба класса различаются лишь в загружаемом изображении (метод Load) и в передвижении (метод h\_changePosition), а константы MaxSpeed, StartFlyMs, поля m\_flying, m\_flyTimer и метод Update – не меняются. И для того, чтобы не писать те же самые методы в классе Ship1, что и в классе Ship, мы просто наследуем класс Ship1 от Ship, переопределяя при этом два метода – Load и h\_changePosition.

* 1. **Инкапсуляция**

Целью инкапсуляции является обеспечение согласованности внутреннего состояния объекта. В C# для инкапсуляции используются публичные свойства и методы объекта. Переменные, за редким исключением, не должны быть публично доступными. Ниже представлены примеры инкапсуляции в данном проекте:

Класс Ship:

* приватные константы MaxSpeed и StartFlyMs;
* защищенные поля m\_flying и m\_flyTimer;
* публичный метод Update.
  1. **Полиморфизм**

Одна из трех основных парадигм ООП. Если говорить кратко, полиморфизм — это способность объекта использовать методы производного класса, который не существует на момент создания базового. Ниже представлены примеры полиморфизма в данном проекте:

Класс BaseActor является абстрактным классом:

* Класс Bullet является подтипом класса BaseActor;
* Класс EnemyBullet является подтипом класса BaseActor.

Класс DethAnimationActor является абстрактным классом:

* Класс Player является подтипом класса DethAnimationActor;
* Класс Ship является подтипом класса DethAnimationActor;
* Класс Prepyt является подтипом класса DethAnimationActor;
* Класс Prepyt2 является подтипом класса DethAnimationActor

Это примеры *полиморфизма подтипов*.

Для реализации пуль игрока и пуль противника за основу был взят абстрактный класс BaseActor, который уже имел перечень определенных методов, которые в двух разных случаях должны были быть реализованы по разному – вражеская пуля убивает игрока, а пуля игрока – врага. Соответственно одинаковые методы у двух разных классов должны быть реализованы по-разному.

1. **НЕДОСТАТКИ ВЫПОЛНЕННОГО РЕШЕНИЯ**

Данное решение выполняет все поставленные задачи.