Задача: Провести объектно-ориентированный анализ задачи и определить объекты, которые потребуется использовать (подсказка: необходимо инкапсулировать изменяемые аспекты поведения объектов):

• создается компьютерная игра, поддерживающая одновременное управление несколькими персонажами;

• каждый персонаж умеет бегать, стрелять и собирать предметы;

• некоторые персонажи умеют летать;

• каждый тип действий сопровождается определенной анимацией;

• собираемые предметы могут менять характеристики действий и соответствующую анимацию (например, вдвое ускорять или, наоборот, замедлять бег персонажа).

Решение:

Алгоритм игры примерно следующий: несколько разных по внешнему виду объектов одновременно движутся с некоторой анимацией: бегают, стреляют, собирают предметы. Некоторые объекты летают. При движении объекты собирают предметы, которые меняют характеристики движения и соответствующую анимацию. Объекты по совершаемым действиям можно разделить на два типа: первый - которые бегают, летают, собирают предметы; и второй тип - которые делают вышеуказанные три действия и дополнительно могут летать. Внешний вид объектов определяется переменной типа объекта. Движение и анимация объектов определяются функциями бега, стрельбы, сбора предметов, полета. Характеристики движения объектов зависят от переменной, которую возвращает функция сбора предметов.

Таким образом ООП модель управления персонажами компьютерной игры включает в себя:

1. родительский класс персонажей hero(hero\_type), где переменной hero\_type определяется тип (внешний вид) персонажа. В этом классе создадим методы:

hero\_run(action\_parameter) управляет бегом и анимацией персонажа, скоростью бега, в зависимости от аргумента action\_parameter;

hero\_shoot(action\_parameter) управляет стрельбой и анимацией персонажа, скоростью стрельбы, в зависимости от аргумента action\_parameter;

hero\_collects\_items() управляет сбором предметов и анимацией персонажа, возвращает некоторое значение в зависимости от собранного предмета, которое в качестве аргумента может передаваться в функции управления бегом, стрельбой и полетом.

2. Наследуем от родительского класса hero дочерний класс hero\_fligter(hero\_type), где переменной hero\_type определяется тип (внешний вид) персонажа. В дочернем классе создадим метод hero\_flight(action\_parameter), который управляет полетом и анимацией персонажа, в зависимости от аргумента action\_parameter.