Министерство образования Республики Беларусь

Учреждение образования

«БелорусскиЙ государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Основы программирования информационных систем»

|  |
| --- |
| *К защите допустить*: |
| Руководитель курсовой работы  канд. техн. наук, доцент |
| \_\_\_\_\_\_\_\_\_\_\_\_А.В.Михалькевич  \_\_\_.\_\_\_. 20\_\_\_ |

**Пояснительная записка**

к курсовой работе

**РАЗРАБОТКА 2D ИГРЫ**

БГУИР КР 1-40 05 01-10 011 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент | (подпись студента) | В. В. Харко |
|  |  | Курсовая работа представлена на проверку \_\_\_.\_\_\_. 20\_\_\_ |
|  |  | (подпись студента) |

Минск 2020

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc532638732)

[1. Описание проекта 5](#_Toc532638733)

[2. Обоснование выбора технологий 10](#_Toc532638740)

[3. Инструментарий 17](#_Toc532638750)

[4. Архитектурный шаблон проектирования MVC 22](#_Toc532638771)

[5. Шаблон проектирования практических задач 37](#_Toc532638793)

[Заключение 46](#_Toc532638841)

[Список литературных источников 47](#_Toc532638841)

[Приложения 48](#_Toc532638841)

**ВВЕДЕНИЕ**

Ни для кого не секрет, что видео игры прочно заняли свою позицию в современной индустрии развлечений. Существуют попытки выделить компьютерные игры как отдельную область искусства, наряду с театром, кино и т.п. Разработка игр может оказаться не только увлекательным, но и прибыльным делом, примеров этому предостаточно в истории. Первые примитивные компьютерные и видео игры были разработаны в 1950-х и 1960-х годах. Они работали на таких платформах, как осциллографы, университетские мейнфреймы и компьютеры EDSAC. Самой первой компьютерной игрой стал симулятор ракеты, созданный в 1942 году Томасом Голдсмитом Младшим (англ. Thomas T. Goldsmith Jr.) и Истл Рей Менном (англ. Estle Ray Mann). Позже, в 1952 году, появилась программа "OXO", имитирующая игру "крестики-нолики", созданная А.С. Дугласом как часть его докторской диссертации в Кембриджском Университете. Игра работала на большом университетском компьютере, известном как EDSAC (Electronic Delay Storage Automatic Calculator). В настоящее время, разработка игры - это многомиллионный процесс, в котором задействована целая команда разработчиков, сложные современные технологии и даже маркетинговые ходы.

Так как при написании курсовой работы в начале 2 курса мне пришлось познакомиться с таким языком программирования как Python и несколькими библиотеками, я решил для себя, что хочу попробовать себя в написании игры на Python (заранее понимал, что это достаточно сложная задача, т.к. до этого Python я не изучал).

Цель курсовой работы состояла в создании 2D игры на Python при помощи библиотеки pygame. Суть игры состояла в том, что Волк должен бежать, прыгая через препятствия, попутно собирая жизнь и энергию (энергия дает возможность Волку выстрелить). Необходимо было реализовать красивый и простой интерфейс меню для пользователя, где будет кнопка, по нажатии которая запускается сама игра, и кнопка выхода из игры. Также требовалось создать красивый, удобный и интересный интерфейс главного окна самой игры.

Согласно задачам, которые я себе поставил - необходимо написать игру «Ну Погоди» на языке Python. Игра должна быть разработана как приложение ОС Windows. Предполагаемый размер поля 800х600 пикселей. Необходимо создания двух окон (меню и главное окно игры), реализовать кнопки перехода с меню в основное окно игры и запуск игры соответственно. Стояла задача прорисовки анимации бега самого Волка, также добавление на экран фона, препятствий и доп. ресурсов. На экране должен был отображаться счет (количество пройденных препятствий), количество жизней и энергии. В ходе выполнения работы я решил добавить Зайца, который будет вылетать из верха экрана и атаковать Волка (на данный момент времени реализация убийства Зайца реализована, а отнимание жизни у Волка – нет). Также должно быть предусмотрено звуковое сопровождение в меню, игре, при ударе об препятствие, а также при получении жизни или энергии. Соответственно должна быть реализована пауза для остановки игры.

В дальнейшем планируется улучшать игру допустим выбором персонажа, уровня, оружия и т.д. А также добавить таблицу рекордов, которые будут храниться в файле.

1. **Описание проекта**

Перед тем, как начать составлять описание проекта мной был более подробно изучен язык Python, а именно библиотека pygame.

Что такое Pygame?

Pygame — набор [модулей](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (библиотек) языка программирования [Python](https://ru.wikipedia.org/wiki/Python), предназначенный для написания [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0) и [мультимедиа](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)-приложений. Pygame базируется на мультимедийной библиотеке [SDL](https://ru.wikipedia.org/wiki/Simple_DirectMedia_Layer).

Изначально Pygame был написан [Питом Шиннерсом](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%B8%D1%82%D0%BE%D0%BC_%D0%A8%D0%B8%D0%BD%D0%BD%D0%B5%D1%80%D1%81%D0%BE%D0%BC&action=edit&redlink=1) (Pete Shinners). Начиная примерно с [2004](https://ru.wikipedia.org/wiki/2004)/[2005 года](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4) поддерживается и развивается сообществом свободного программного обеспечения.

Pygame – это библиотека модулей для языка Python, созданная для разработки 2D игр. Также Pygame могут называть фреймворком. В программировании понятия "библиотека" и "фреймворк" несколько разные. Но когда дело касается классификации конкретного инструмента, не все так однозначно.

В любом случае, фреймворк является более мощным по-сравнению с библиотекой, он накладывает свою специфику на особенности программирования и сферу использования продукта. С точки зрения специфики Pygame – это фреймворк. Однако его сложно назвать "мощным инструментом". По своему объему и функционалу это скорее библиотека.

Также существует понятие "игрового движка" как программной среды для разработки игр. По своему назначению Pygame можно считать игровым движком. В то же время, с точки зрения классификации программного обеспечения, Pygame является API для Питона к API библиотеки SDL.

API – это интерфейс (в основном набор функций и классов) для прикладного (часто более высокоуровневого) программирования, который предоставляет, например, та или иная библиотека. SDL – это библиотека, которая работает с мультимедийными устройствами компьютера.

В этом смысле Pygame можно сравнить с Tkinter, который через свои функции и классы предоставляет Питону доступ к графической библиотеке Tk.

Pygame — это «игровая библиотека», набор инструментов, помогающих программистам создавать игры. К ним относятся:

Графика и анимация;

Звук (включая музыку);

Управление (мышь, клавиатура, геймпад и так далее);

Игровой цикл.

В сердце каждой игры лежит цикл, который принято называть «игровым циклом». Он запускается снова и снова, делая все, чтобы работала игра. Каждый цикл в игре называется кадром.

В каждом кадре происходит масса вещей, но их можно разбить на три категории:

Обработка ввода (события).

Речь идет обо всем, что происходит вне игры — тех событиях, на которые она должна реагировать. Это могут быть нажатия клавиш на клавиатуре, клики мышью и так далее.

Обновление игры.

Изменение всего, что должно измениться в течение одного кадра. Если персонаж в воздухе, гравитация должна потянуть его вниз. Если два объекта встречаются на большой скорости, они должны взорваться.

Рендеринг (прорисовка).

В этом шаге все выводится на экран: фоны, персонажи, меню. Все, что игрок должен видеть, появляется на экране в нужном месте.

Время.

Еще один важный аспект игрового цикла — скорость его работы. Многие наверняка знакомы с термином FPS, который расшифровывается как Frames Per Second (или кадры в секунду). Он указывает на то, сколько раз цикл должен повториться за одну секунду. Это важно, чтобы игра не была слишком медленной или быстрой. Важно и то, чтобы игра не работала с разной скоростью на разных ПК. Если персонажу необходимо 10 секунд на то, чтобы пересечь экран, эти 10 секунд должны быть неизменными для всех компьютеров.

Pygame-приложения могут работать под [Android](https://ru.wikipedia.org/wiki/Android) на телефонах и планшетах с использованием подмножества Pygame для Android (pgs4a). На этой платформе поддерживаются звук, вибрация, клавиатура, акселерометр. Это довольно важный аспект, т.к. это может быть одна из будущих целей моей игры (размещение ее на Google Play).

После изучения библиотеки pygame, а также просмотра обучающих видеороликов я перешел к составлению описания проекта.

Проект заключается в следующем – первое, что видит пользователь – это меню игры, в котором после нажатия клавиши «Start game» запускается игра, а по нажатии клавиши «Quit» выход соответственно. Для более красивого отображения интерфейса реализована анимация бега Волка, а также изменения цвета кнопки при наведении мышкой. Звуковое сопровождение также присутствует.

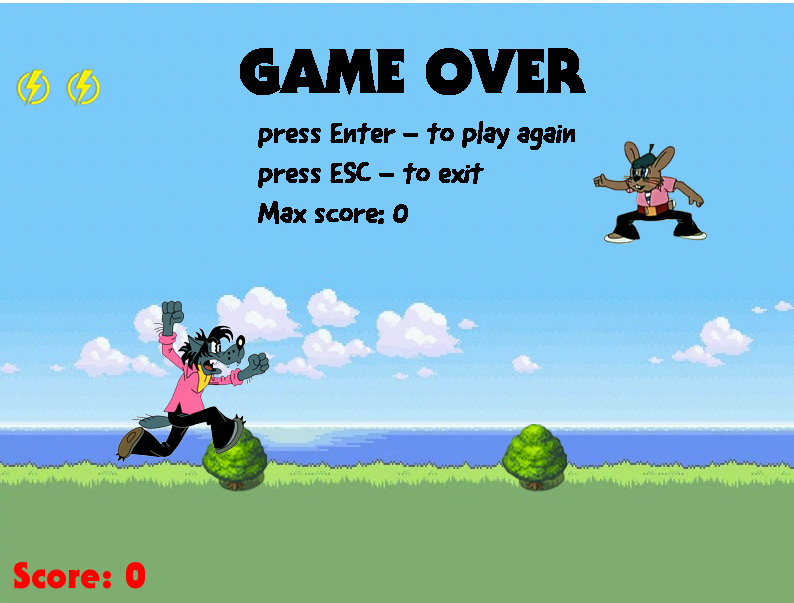


После нажатия кнопки открывается окно игры. В левом верхнем углу отображается количество жизней и энергии. Снизу отображается Ваш текущий счет. Реализован прыжок по нажатии клавиши «Пробел» и стрельба Волка по нажатии клавиши «X» (энергия и является счетчиком возможных выстрелов). Также реализовано оповещение текущего кулдауна (время, через которое Волк снова может выстрелить). Создана случайная генерация 3 типов блоков (препятствий). Реализовано столкновение Волка с препятствием и отнимание жизни при их столкновении. Также присутствует случайный вылет Зайца и его стрельба по бегущему Волку (реализация отнимания жизни у Волка на данный момент не реализована), однако убийство Зайца при попадании выстрела Волка в него реализовано (однако Заяц вернётся, чтобы помешать Волку через определенный промежуток времени).



Появление паузы происходит по нажатии клавиши «Esc», выход из нее по нажатии клавиши «Space» возврат к игре.





При проигрыше, т.е. потери всех жизней пользователю предлагается продолжить игру по нажатию клавиши «Space» и выход в меню по нажатию клавиши «Esc».

1. **Обоснование выбора технологий**

Так как основной задачей курсовой было написание проекта на объектно-ориентированном языке, то необходимо обосновать что такое ООП и особенности Python, как объектно-ориентированного языка.

Объектно-ориентированное программирование (ООП) — это парадигма программирования, где различные компоненты компьютерной программы моделируются на основе реальных объектов. Объект — это что-либо, у чего есть какие-либо характеристики и то, что может выполнить какую-либо функцию.

Стоит обратить внимание на то, что объектно-ориентированное программирование — не зависящая от языка программирования концепция. Это общая концепция программирования и большинство современных языков, такие как [Java](https://python-scripts.com/java-vs-python-simple-code-examples), C#, C++ и Python поддерживают объектно-ориентированное программирование.

Объектно-ориентированный язык работает по следующим принципам:

* Все данные представляются объектами;
* Программа является набором взаимодействующих объектов, посылающих друг другу сообщения;
* Каждый объект имеет собственную часть памяти и может иметь в составе другие объекты;
* Каждый объект имеет тип;
* Объекты одного типа могут принимать одни и те же сообщения (и выполнять одни и те же действия).

Идеи/принципы объектно-ориентированного программирования:

1. Наследование. Возможность выделять общие свойства и методы классов в один класс верхнего уровня (родительский). Классы, имеющие общего родителя, различаются между собой за счет включения в них различных дополнительных свойств и методов.
2. Инкапсуляция. Свойства и методы класса делятся на доступные из вне (опубликованные) и недоступные (защищенные). Защищенные атрибуты нельзя изменить, находясь вне класса. Опубликованные же атрибуты также называют интерфейсом объекта, т. к. с их помощью с объектом можно взаимодействовать. По идеи, инкапсуляция призвана обеспечить надежность программы, т.к. изменить существенные для существования объекта атрибуты становится невозможно.
3. Полиморфизм. Полиморфизм подразумевает замещение атрибутов, описанных ранее в других классах: имя атрибута остается прежним, а реализация уже другой. Полиморфизм позволяет специализировать (адаптировать) классы, оставляя при этом единый интерфейс взаимодействия.

Преимущества ООП.

В связи со своими особенностями объектно-ориентированное программирование имеет ряд преимуществ перед структурным (и др.) программированием. Выделим некоторые из них:

1. Использование одного и того же программного кода с разными данными. Классы позволяют создавать множество объектов, каждый из которых имеет собственные значения атрибутов. Нет потребности вводить множество переменных, т.к. объекты получают в свое распоряжение индивидуальные так называемые пространства имен. Пространство имен конкретного объекта формируется на основе класса, от которого он был создан, а также от всех родительских классов данного класса. Объект можно представить как некую упаковку данных.
2. Наследование и полиморфизм позволяют не писать новый код, а настраивать уже существующий, за счет добавления и переопределения атрибутов. Это ведет к сокращению объема исходного кода.

Особенность ООП.

ООП позволяет сократить время на написание исходного кода, однако ООП всегда предполагает большую роль предварительного анализа предметной области, предварительного проектирования. От правильности решений на этом предварительном этапе зависит куда больше, чем от непосредственного написания исходного кода.

Особенности ООП в Python.

По сравнению с другими распространенными языками программирования у Python можно выделить следующие особенности, связанные с объектно-ориентированным программированием:

1. Любое данное (значение) - это объект. Число, строка, список, массив и др. - все является объектом. Бываю объекты встроенных классов (как те, что перечисленные в предыдущем предложении), а бывают объекты пользовательских классов (тех, что создает программист). Для единого механизма взаимодействия предусмотрены методы перегрузки операторов.
2. Класс - это тоже объект с собственным пространством имен. Это нигде не было указано в данном цикле уроков. Однако это так. Поэтому правильнее было употреблять вместо слова «объект», слово «экземпляр». И говорить «экземпляр объекта», подразумевая под этим созданный на основе класса именно объект, и «экземпляр класса», имея ввиду сам класс как объект.
3. Инкапсуляции в Python не уделяется особого внимания. В других языках программирования обычно нельзя получить напрямую доступ к свойству, описанному в классе. Для его изменения может быть предусмотрен специальный метод. В Python же это легко сделать, просто обратившись к свойству класса из вне. Несмотря на это в Python все-таки предусмотрены специальные способы ограничения доступа к переменным в классе.

Обзор терминологии ООП.

* Класс - определенный пользователем прототип для объекта, который определяет набор атрибутов, которые характеризуют любой объект класса. Атрибутами являются члены данных (переменные класса и переменные экземпляра) и методы, доступ к которым осуществляется через точечную запись.
* Переменная класса - переменная, которая используется всеми экземплярами класса. Переменные класса определены внутри класса, но вне любого из методов класса. Переменные класса используются не так часто, как переменные экземпляра.
* Член данных - переменная класса или переменная экземпляра, которая содержит данные, связанные с классом и его объектами.
* Перегрузка функций - назначение более чем одного поведения определенной функции. Выполняемая операция варьируется в зависимости от типов объектов или аргументов.
* Переменная экземпляра - переменная, которая определена внутри метода и принадлежит только текущему экземпляру класса.
* Наследование - передача характеристик класса другим классам, которые являются его производными.
* Экземпляр - индивидуальный объект определенного класса. Например, объект obj, принадлежащий классу Circle, является экземпляром класса Circle.
* Instantiation - создание экземпляра класса.
* Метод - особый вид функции, который определен в определении класса.
* Объект - уникальный экземпляр структуры данных, который определяется его классом. Объект включает в себя как члены данных (переменные класса и переменные экземпляра), так и методы.
* Перегрузка оператора - назначение более чем одной функции определенному оператору.

Преимущества и недостатки ООП Python.

Рассмотрим несколько основных преимуществ объектно-ориентированного программирования:

Объектно-ориентированное программирование подразумевает повторное использование. Компьютерная программа, написанная в форме объектов и классов может быть использована снова в других проектах без повторения кода;

Использование модулярного подхода в объектно-ориентированном программировании позволяет получить читаемый и гибкий код;

В объектно-ориентированном программировании каждый [класс](https://python-scripts.com/data-class) имеет определенную задачу. Если ошибка возникнет в одной части кода, вы можете исправить ее локально, без необходимости вмешиваться в другие части кода;

Инкапсуляция данных (которую мы рассмотрим дальше в статье) вносит дополнительный уровень безопасности в разрабатываемую программу с использованием объектно-ориентированного подхода;

Хотя объектно-ориентированное программирование обладает рядом преимуществ, оно также содержит определенные недостатки, некоторые из них находятся в списке ниже:

Для создания объектов необходимо иметь подробное представление о разрабатываемом программном обеспечении;

Не каждый аспект программного обеспечения является лучшим решением для реализации в качестве объекта. Для новичков может быть тяжело прочертить линию в золотой середине;

С тем, как вы вносите все новые и новые классы в код, размер и сложность программы растет в геометрической прогрессии;

В следующем разделе мы рассмотрим ряд самых важных концепций объектно-ориентированного программирования.

Как и следует из названия, объектно-ориентированное программирование — это речь об объектах. Однако, перед тем как создать объект, нам нужно определить его класс.

Класс.

Класс в объектно-ориентированном программировании выступает в роли чертежа для объекта. Класс можно рассматривать как карту дома. Вы можете понять, как выглядит дом, просто взглянув на его карту.

Сам по себе класс не представляет ничего. К примеру, нельзя сказать, что карта является домом, она только объясняет, как настоящий дом должен выглядеть.

Отношение между классом и объектом можно представить более наглядно, взглянув на отношение между машиной и Audi. Да, Audi – это машина. Однако, нет такой вещи, как просто машина. Машина — это абстрактная концепция, которую также реализуют в Toyota, Honda, Ferrari, и других компаниях.

Ключевое слово class используется для [создания класса](https://python-scripts.com/python-class) в Python. Название класса следует за ключом class, за которым следует двоеточие. Тело класса начинается с новой строки, с отступом на одну вкладку влево.

Объекты.

Раннее мы поняли, что класс предоставляет чертеж объекта. Однако, чтобы на самом деле использовать объекты и методы класса, вам нужно создать объект из этого класса. Существует несколько методов и атрибутов класса, которые можно использовать вне объекта, мы рассмотрим их в следующем разделе.

Сейчас просто запомните, что по умолчанию, нам нужно создать объект класса перед тем, как мы сможем начать использовать его методы и атрибуты.

Объект также называется экземпляром. Тем не менее, процесс создания объекта класса называется инициализация. В Python, чтобы создать объект класса, нам просто нужно вписать название класса, с последующими открывающимися и закрывающимися скобками.

Теперь разберемся с библиотекой Pygame.

Pygame – это библиотека модулей для языка Python, созданная для разработки 2D игр. Также Pygame могут называть фреймворком. В программировании понятия "библиотека" и "фреймворк" несколько разные. Но когда дело касается классификации конкретного инструмента, не все так однозначно.

В любом случае, фреймворк является более мощным по-сравнению с библиотекой, он накладывает свою специфику на особенности программирования и сферу использования продукта. С точки зрения специфики Pygame – это фреймворк. Однако его сложно назвать "мощным инструментом". По своему объему и функционалу это скорее библиотека.

Также существует понятие "игрового движка" как программной среды для разработки игр. По своему назначению Pygame можно считать игровым движком. В то же время, с точки зрения классификации программного обеспечения, Pygame является API для Питона к API библиотеки SDL.

API – это интерфейс (в основном набор функций и классов) для прикладного (часто более высокоуровневого) программирования, который предоставляет, например, та или иная библиотека. SDL – это библиотека, которая работает с мультимедийными устройствами компьютера.

В этом смысле Pygame можно сравнить с Tkinter, который через свои функции и классы предоставляет Питону доступ к графической библиотеке Tk.

Особенности разработки компьютерных игр.

Игры событийно-ориентированны, также как любое приложение с графическим интерфейсом пользователя. Поэтому какие-никакие, но игры можно было бы писать с помощью Tkinter, в частности на его экземплярах холста. Но поскольку основное назначение библиотеки графического пользовательского интерфейса совсем другое, то пришлось бы изобретать велосипеды. В то время как библиотека, специально предназначенная для написания игр, уже содержит необходимые объекты, что упрощает разработку.

Например, чтобы определить, столкнулись ли два объекта, надо написать код, проверяющий совпадение координат. Это может быть непростой задачей, так как надо учесть области перекрытия, форму объектов и др. В то же время игровой движок может включать готовую функцию проверки коллизии (столкновения) с необходимыми опциями настройки.

При всем этом Pygame достаточно низкоуровневый игровой движок, если его можно так называть. Это значит, что многое в нем не остается за кадром, а дается программисту на доработку, вынуждает его понимать, как работают "шестеренки". Так в Pygame отсутствует эмуляция физических явлений. Если вам надо смоделировать движение с ускорением или по дуге, программируйте это сами, предварительно взяв из курса физики соответствующую формулу.

Игры относятся к мультимедийным приложениям. Однако, в отличие от других приложений этой группы, для них характерна сложная программная логика и нередко много математики, хотя достаточно простой, плюс эмуляция физических явлений. В играх программируется подобие искусственного интеллекта. В многопользовательской игре, хотя пользователи играют друг с другом, а не с ИИ, создаются виртуальные миры, существующие по законам, заложенными разработчиками.

В программном коде игры выделяют три основных логических блока:

Отслеживание событий, производимых пользователем и не только им.

Изменение состояний объектов, согласно произошедшим событиям.

Отображение объектов на экране, согласно их текущим состояниям.

Эти три этапа повторяются в цикле бесчисленное количество раз, пока игра запущена.

Место Pygame среди инструментов разработки игр.

Популярна ли библиотека pygame, пишут ли на ней игры состоявшиеся программисты? Хотя на Pygame есть востребованные игры, в подавляющем случае – нет. Для программирования под андроид и десктоп существуют более функциональные игровые движки.

Для создания двумерных браузерных игр инди-разработчики (от слова independent – независимый, здесь понимается как "одиночка", "не работающий в команде или на фирму") часто используют JavaScript и его игровые библиотеки, так как JS родной для веба язык.

В чем тогда преимущество Pygame? Оно в легком вхождении в отрасль и прототипировании. Pygame – небольшая библиотека. Сам Python позволяет писать короткий и ясный код. Так что это хорошее начало, чтобы познакомиться с особенностями разработки игр. Более опытными программистами Pygame может использоваться для быстрого создания прототипа игры, чтобы посмотреть, как все будет работать. После этого программа переписывается на другом языке. Другими словами, преимущество Pygame в легком обучении и быстрой разработке.

1. **Инструментарий**

Обоснование используемых инструментов.

[PyCharm](https://python-scripts.com/pycharm-download) предоставляет умную **проверку кода**, быстрое выявление ошибок и оперативное исправление, вкупе с автоматическим [рефакторингом кода](https://python-scripts.com/code-analysis), и богатыми возможностями в навигации. Кроме этого, **скачать PyCharm** не составит никакого труда.

# Умный редактор кода.

Редактор кода **PyCharm** предоставляет первоклассную поддержку для Python, JavaScript, CoffeeScript, TypeScript, CSS, популярных языковых и многого другого. Оцените преимущества возможности поддержки нескольких языков, **обнаружения ошибок** и их исправления!

### Умная навигация.

Воспользуйтесь умным поиском для перехода к любому [классу](https://python-scripts.com/python-class), [файлу](https://python-scripts.com/work-with-files-python), символу, или даже событию IDE и окну инструментов. Всего один клик, и вы переходите к супер-методу, тестированию, использованию, реализации, декларации и это как минимум!

### Быстрый и безопасный рефакторинг.

С **PyCharm** вам открывается разумное **обновление кода** с безопасным удалением и переименованием, методом извлечения, вводной переменной, встроенной переменной или методом, и другими видами рефакторинга. Ориентированные на фреймворк и язык программирования рефакторинги помогут выполнить любое изменение в рамках целого проекта.

# Встроенные Инструменты Разработки.

Большая коллекция инструментов PyCharm включает в себя интегрированный отладчик и запуск тестирования

* профайлер Python;
* встроенный терминал;
* интеграцию с большими VCS и встроенными инструментами [баз данных](https://python-scripts.com/database);
* возможность удаленной разработки с удаленными интерпретаторами;
* интегрированный **терминал ssh**;
* интеграция с **Docker** и **Vagrant**.

### Отладка, Тестирование и Профилирование

Используйте мощный отладчик с **графическим интерфейсом** под Python и JavaScript. Создавайте и проводите тестирования с поддержкой кода и драйвером текстов на основе графического пользовательского интерфейса. Получите полный контроль над своим кодом благодаря интеграции **Python Profiler**.

### VCS, Развертывание и Удаленная Разработка

Сохраните свое время благодаря унифицированному пользовательскому интерфейсу для работы с Git, SVN, Mercurial и другими системами контроля версий. Запускайте и проводите **отладку своего приложения** в удаленном режиме. Вам доступна простая конфигурация автоматического развертывания для удаленного хоста или VM и управление вашей инфраструктурой при помощи Vagrant и Docker.

### Инструменты Базы Данных

* Access Oracle
* SQL Server
* PostgreSQL
* MySQL

И другие [базы данных](https://python-scripts.com/database) прямо из IDE. Вы можете рассчитывать на помощь **PyCharm** при редактировании кода SQL, выполнении запросов, просмотре данных и внесении изменений в схемы.

# Веб разработка.

В дополнении к Python, **PyCharm** предоставляет первоклассную поддержку различных фреймворков **веб разработки** от Python, отдельных языков, JavaScript, CoffeeScript, TypeScript, HTML/CSS, AngularJS, Node.js и других.

### Веб фреймворки Python.

**PyCharm** предоставляет отличную поддержку отдельных фреймворков для современных фрейморков веб разработки, таких как [Django](https://python-scripts.com/django-obzor), [Flask](https://python-scripts.com/flask-vs-django), Google App Engine, Pyramid, и web2py, включая отладчик **шаблонов Django**, инструменты manage.py и appcfg.py, автозаполнение и навигацию.

### JavaScript и HTML.

**PyCharm** предоставляет первоклассную поддержку для JavaScript, CoffeeScript, TypeScript, HTML и CSS, а также их современных преемников. Отладчик JavaScript также включен в PyCharm и является интегрированным с конфигурацией запуска сервера Django.

### Live Edit.

Live Editing Preview позволяет вам открывать страницу в редакторе и браузере для поиска и мгновенного нахождения недавно внесенных изменений в браузере. **PyCharm** сохраняет ваши изменения в автоматическом режиме, в то время как браузер сам обновляет страницу, показывая результаты изменений.

# Научные инструменты.

**PyCharm** интегрируется с IPython Notebook, имеет интерактивную консоль Python и поддерживает как Anaconda, так и множество других пакетов, включая **Matplotlib** и [NumPy](https://python-scripts.com/numpy).

### Интеграция IPython Notebook.

PyCharm интегрируется с IPython Notebook и обеспечивает решение, которое комбинирует возможности IPython Notebook с дополнительными преимуществами, которые может предложить большая часть интеллектуальной среды разработки Python, включая автоматическое завершение, навигацию, проверку ошибок, и так далее.

### Интерактивная консоль Python.

Еще одна причина [скачать PyCharm](https://python-scripts.com/pycharm-download) заключается в том, что вы можете запустить консоль Python REPL в **PyCharm**, что дает массу преимуществ: мгновенная проверка синтаксиса с дополнительными проверками, сопоставления скобок и кавычек, и, конечно, завершением кода.

### Поддержка научного стека.

**PyCharm** предоставляет специальную поддержку научных библиотек. Он поддерживает Anaconda, Numpy, Matplotlib и другие научные библиотеки, предоставляя пользователю глубокое понимание кода, **интерактивные графики**, просмотр массивов и многое другое.

# Настраиваемая и кроссплатформенная IDE.

Вы можете [скачать PyCharm](https://python-scripts.com/pycharm-download) и установить на Windows, Mac OS и Linux используя один лицензионный ключ. Наслаждайтесь отлично настроенным рабочим пространством с настраиваемыми цветовыми схемами и горячими клавишами, с доступной эмуляцией VIM.

### Настраиваемый пользовательский интерфейс.

Существует ли хотя бы один разработчик, который не любит настраивать свои инструменты? Мы с таким еще не сталкивались, так что к вашим услугам – простая настройка UI от **PyCharm**. Наслаждайтесь удобным рабочим пространством с настраиваемыми цветовыми схемами и горячими клавишами.

### Кроссплатформенная среда разработки.

Вы можете [скачать PyCharm](https://python-scripts.com/pycharm-download) и установить его на Windows, Mac OS или Linux. Кроме этого, вы можете установить его на такое количество компьютеров, которое сами пожелаете, пользуясь той же средой и функционалом на любом компьютере.

Использование системы контроля версий GIT.

**Git** — распределённая [система управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8)(СУВ). Среди проектов, использующих Git — [ядро Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), [Swift](https://ru.wikipedia.org/wiki/Swift_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Android](https://ru.wikipedia.org/wiki/Android), [Drupal](https://ru.wikipedia.org/wiki/Drupal), [Cairo](https://ru.wikipedia.org/wiki/Cairo), [GNU Core Utilities](https://ru.wikipedia.org/wiki/GNU_Coreutils), [Mesa](https://ru.wikipedia.org/wiki/Mesa_3D), [Wine](https://ru.wikipedia.org/wiki/Wine), [Chromium](https://ru.wikipedia.org/wiki/Chromium), [Compiz Fusion](https://ru.wikipedia.org/wiki/Compiz_Fusion), [FlightGear](https://ru.wikipedia.org/wiki/FlightGear), [jQuery](https://ru.wikipedia.org/wiki/JQuery), [PHP](https://ru.wikipedia.org/wiki/PHP), [NASM](https://ru.wikipedia.org/wiki/NASM), [MediaWiki](https://ru.wikipedia.org/wiki/MediaWiki), [DokuWiki](https://ru.wikipedia.org/wiki/DokuWiki), [Qt](https://ru.wikipedia.org/wiki/Qt). Основное отличие Git От любой другой системы контроля версий это подход к работе с данными. Большинство СУВ хранят информацию в виде списка изменений в файле. Эти системы представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени. Git же в свою очередь хранит информацию в виде ссылок на файлы предыдущих версий, с возможностью доступа к ним. Также к преимуществам Git можно отнести то, что для работы большинства операций ему достаточно локальных файлов и ресурсов.

В системах контроля версий существует ветвление. Оно позволяет отделяться от основной линии разработки и при этом продолжать работу. В большинстве СУВ эта функция реализована но при этом это достаточно ресурсозатратный процесс поскольку приходится создавать новую копию директории что может занимать значительное время при работе с большими проектами. В Git же этот процесс выполняется гораздо быстрее из–за того что файлы хранятся локально.

Для работы с Git необходимо знать ряд базовых команд. В эти команды входит: git add, git commit, gid status, git rm, git mv,. Функция git add добавляет содержимое рабочей директории в индекс для последующего коммита. Git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок. Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс, какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов. Команда git rm это команда, обратная команде git add. То есть она удаляет файлы из текущей директории или индекса. Git mv или же git move это простое перемещение файлов между директориями.

Существует отдельный блок команд для создания и перемещения между ветками. Для создания ветки используется команда git branch. Команда git checkout создает ветку и переходит на неё. Если ветка с заданным именем уже существует, эта команды позволят перейти на неё.

Ветка в Git — это простой перемещаемый указатель на один из ваших коммитов. По умолчанию, имя основной ветки в Git — master. Как только вы начнёте создавать коммиты, ветка master будет всегда указывать на последний коммит в основной ветке. Каждый раз при создании коммита указатель ветки master будет передвигаться на следующий коммит автоматически. Так же существует ещё один указатель – HEAD. Это указатель на текущее местоположение. При создании новой ветки указатель master останется на основной ветке, а HEAD перейдет на новую.

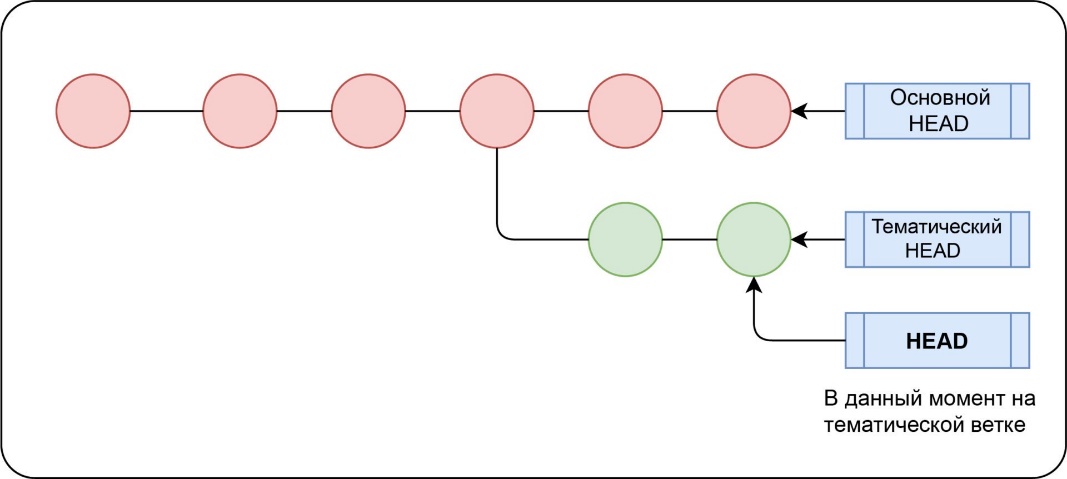


Рисунок 3.1 – Ветвление

При работе над проектом более одного человека необходимо взаимодействовать и обмениваться кодом, в этом может помочь GitHub.

[GitHub](https://github.com/) — сервис онлайн–хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше. Обычно он используется вместе с Git и даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах. Он имеет полную совместимость с Git, что позволяет загружать обновление на сайт из интерфейса командной строки Git.

Ссылка на Git репозиторий с моим проектом:

https://github.com/Valery228/game

**4. Архитектурный шаблон проектирования MVC**

MVC — это фундаментальный паттерн, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам.

Впервые паттерн MVC появился в языке SmallTalk. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес логики, а бизнес логику от данных. Таким образом, в классическом варианте, MVC состоит из трех частей, которые и дали ему название. Рассмотрим их:

Модель.

Под Моделью, обычно понимается часть содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель.

Модель обладает следующими признаками:

* Модель — это бизнес-логика приложения;
* Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
* Для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);
* Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения;

Представление (View).

В обязанности Представления входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Представление обладает следующими признаками:

* В представлении реализуется отображение данных, которые получаются от модели любым способом;
* В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.  
  Примеры представления: HTML-страница, WPF форма, Windows Form.  
  Различия MVC & MVVM & MVP  
  Наиболее распространенные виды MVC-паттерна, это:
* Model-View-Controller
* Model-View-Presenter
* Model-View-View Model  
  Рассмотрим и сравним каждый из них.

###### **Model-View-Presenter** https://habrastorage.org/getpro/habr/post_images/f1b/cbf/d44/f1bcbfd44fc367c0e07e00d957ac188a.png

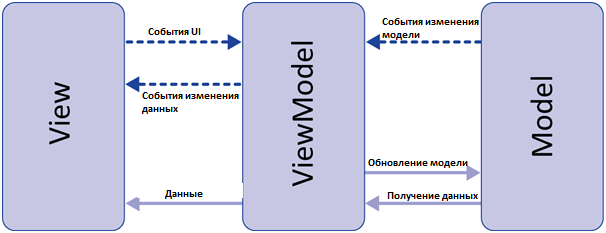
###### Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель. Признаки презентера:

* Двухсторонняя коммуникация с представлением;
* Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
* Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
* Один экземпляр презентера связан с одним отображением.

Реализация:

Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, **IView**.ShowErrorMessage(string msg)). Презентер должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе.  
Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и практически никогда не обрабатываются логикой представления (в т.ч. создания других представлений).  
Пример использования:**Windows Forms.**

###### **Model-View-View Model**



Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.  
Признаки View-модели:

* Двухсторонняя коммуникация с представлением;
* View-модель — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели
* View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)
* Один экземпляр View-модели связан с одним отображением.

Реализация:

При использовании этого паттерна, представление не реализует соответствующий интерфейс (IView).

Представление должно иметь ссылку на источник данных (DataContex), которым в данном случае является View-модель. Элементы представления связаны (Bind) с соответствующими свойствами и событиями View-модели.  
В свою очередь, View-модель реализует специальный интерфейс, который используется для автоматического обновления элементов представления. Примером такого интерфейса в WPF может быть INotifyPropertyChanged.

Пример использования: **WPF**

###### **Model-View-Controller** https://habrastorage.org/getpro/habr/post_images/01c/c4f/3f2/01cc4f3f2646eaea356bc50dccce40d6.png

###### Основная идея этого паттерна в том, что и контроллер и представление зависят от модели, но модель никак не зависит от этих двух компонент. **Признаки контроллера:**

* Контроллер определяет, какие представление должно быть отображено в данный момент;
* События представления могут повлиять только на контроллер.контроллер может повлиять на модель и определить другое представление.
* Возможно несколько представлений только для одного контроллера;

**Реализация:**

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Mодель, посредством вызова соответствующего метода. После изменения Модель использует событие о том что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

Пример использования: **MVC ASP.NET**

**Резюме**

Реализация MVVM и MVP-паттернов, на первый взгляд, выглядит достаточно простой схожей. Однако, для MVVM связывание представления с View-моделью осуществляется автоматически, а для MVP — необходимо программировать  
MVC, по-видимому, имеет больше возможностей по управлению представлением.

**Общие правила выбора паттерна**

**MVVM**

* Используется в ситуации, когда возможно связывание данных без необходимости ввода специальных интерфейсов представления (т.е. отсутствует необходимость реализовывать IView);
* Частым примером является технология WPF.

###### **MVP**

* Используется в ситуации, когда невозможно связывание данных (нельзя использовать Binding);
* Частым примером может быть использование Windows Forms.

###### **MVC**

* Используется в ситуации, когда связь между представление и другими частями приложения невозможна (и Вы не можете использовать MVVM или MVP);
* Частым примером использования может служить ASP.NET MVC.

Cтрого придерживаться только одному паттерну — не всегда лучший выбор.

[MVC](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) – паттерн проектирования позволяющий эффективно разделить модель данных, представление и обработку действий. В состав MVC входят три паттерна проектирования: [наблюдатель](http://en.wikipedia.org/wiki/Observer_pattern), [стратегия](http://en.wikipedia.org/wiki/Strategy_pattern) и [компоновщик](http://en.wikipedia.org/wiki/Composite_pattern). В статье рассматривается классическая схема MVC с активной моделью, описанная в книге Эрика Фримена «[Паттерны проектирования](http://www.books.ru/books/patterny-proektirovaniya-850800/?show=1)».

Для дальнейшей работы потребуется:

* Python 3.8
* PyQt 4.9.1
* IDE (я использовал PyCharm Community 2020.1)

Коротко о паттерне MVC

Как следует из названия, паттерн MVC включает в себя 3 компонента: Модель, Представление и Контроллер. Каждый из компонентов выполняет свою роль и является взаимозаменяемым. Это значит, что компоненты связаны друг с другом лишь некими четкими интерфейсами, за которыми может лежать любая реализация. Такой подход позволяет подменять и комбинировать различные компоненты, обеспечивая необходимую логику работы или внешний вид приложения. Разберемся с теми функциями, которые выполняет каждый компонент.

Модель.

Отвечает за внутреннюю логику работы программы. Здесь мы можем скрыть способы хранения данных, а также правила и алгоритмы обработки информации.

Например, для одного приложения мы можем создать несколько моделей. Одна будет отладочной, а другая рабочей. Первая может хранить свои данные в памяти или в файле, а вторая уже задействует базу данных. По сути это просто паттерн Стратегия.

Представление.

Отвечает за отображение данных Модели. На этом уровне мы лишь предоставляем интерфейс для взаимодействия пользователя с Моделью. Смысл введения этого компонента тот же, что и в случае с предоставлением различных способов хранения данных на основе нескольких Моделей.

Например, на ранних этапах разработки мы можем создать простое консольное представление для нашего приложения, а уже потом добавить красиво оформленный GUI. Причем, остается возможность сохранить оба типа интерфейсов.

Кроме того, следует учитывать, что в обязанности Представления входит лишь своевременное отображение состояния Модели. За обработку действий пользователя отвечает Контроллер, о которым мы сейчас и поговорим.

Контроллер.

Обеспечивает связь между Моделью и действиями пользователя, полученными в результате взаимодействия с Представлением. Координирует моменты обновления состояний Модели и Представления. Принимает большинство решений о переходах приложения из одного состояния в другое.

Фактически на каждое действие, которое может сделать пользователь в Представлении, должен быть определен обработчик в Контроллере. Этот обработчик выполнит соответствующие манипуляции над моделью и в случае необходимости сообщит Представлению о наличии изменений.

Моя проект состоит из 10 файлов:

game.py (самый важный файл, здесь прописана вся логика игры)

import parameters as p  
from button import \*  
from enemy import \*  
from object import \*  
from effects import \*  
from images import \*  
  
  
class Game:  
 def \_\_init\_\_(self):  
 pygame.display.set\_caption('Ny pogodi')  
 pygame.display.set\_icon(icon)  
  
 pygame.mixer\_music.load('music/muz.mp3')  
 pygame.mixer\_music.set\_volume(0.2)  
  
 self.block\_options = [60, 430, 54, 400, 60, 421]  
 self.img\_counter = 0  
 self.health = 2  
 self.energy = 2  
 self.jump\_counter = 30  
 self.scores = 0  
 self.max\_scores = 0  
 self.max\_above = 0  
 self.cooldown = 0  
 self.make\_jump = False  
  
 def show\_menu(self):  
 pygame.mixer\_music.pause()  
 pygame.mixer.Sound.play(menu\_sound)  
 pygame.mixer.Sound.play(menu\_sound, -1)  
 menu\_sound.set\_volume(0.2)  
  
 start\_btn = Button(200, 50)  
 quit\_btn = Button(120, 50)  
  
 show = True  
  
 while show:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 display.blit(fon, (0, 0))  
 display.blit(menu\_bckgr, (0, 0))  
 display.blit(ny, (195, 50))  
  
 if self.img\_counter == 36:  
 self.img\_counter = 0  
 display.blit(volk\_img[self.img\_counter // 6], (280, 130))  
 self.img\_counter += 1  
 start\_btn.draw(300, 350, ' Start game', self.start\_game)  
 quit\_btn.draw(340, 410, ' Quit', quit)  
 pygame.display.update()  
 clock.tick(60)  
  
 def start\_game(self):  
 self.health = 2  
 self.energy = 2  
  
 while self.game\_cycle():  
 self.scores = 0  
 self.health = 2  
 self.make\_jump = False  
 self.jump\_counter = 30  
 p.usr\_y = display\_height - usr\_height - 110  
 self.cooldown = 0  
  
 def game\_cycle(self):  
  
 pygame.mixer\_music.play(-1)  
  
 game = True  
 block\_arr = []  
 self.create\_block\_arr(block\_arr)  
  
 heart = Object(display\_width, 220, 30, health\_img, 4)  
 energ = Object(display\_width, 220, 30, energy\_img, 4)  
  
 all\_btn\_bullets = []  
 all\_ms\_bullets = []  
  
 zaych1 = Enemies(-150)  
  
 all\_enemies = [zaych1]  
  
 while game:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 keys = pygame.key.get\_pressed()  
 mouse = pygame.mouse.get\_pos()  
 click = pygame.mouse.get\_pressed()  
  
 if keys[pygame.K\_SPACE]:  
 self.make\_jump = True  
 if keys[pygame.K\_ESCAPE]:  
 menu\_sound.set\_volume(0.2)  
 self.pause()  
  
 if self.make\_jump:  
 self.jump()  
  
 self.count\_scores(block\_arr)  
  
 display.blit(land, (0, 0))  
 print\_text1('Score: ' + str(self.scores), 15, 550)  
  
 self.draw\_array(block\_arr)  
 self.draw\_volk()  
  
 if not self.cooldown:  
 if self.energy >= 1:  
 if keys[pygame.K\_x]:  
 self.energy -= 1  
 pygame.mixer.Sound.play(bullet\_sound)  
 all\_btn\_bullets.append(Bullet(p.usr\_x + p.usr\_width, p.usr\_y + 28))  
 self.cooldown = 60  
 elif click[0]:  
 pygame.mixer.Sound.play(bullet\_sound)  
 add\_bullet = Bullet(p.usr\_x + p.usr\_width, p.usr\_y + 28)  
 add\_bullet.find\_path(mouse[0], mouse[1])  
 all\_btn\_bullets.append(add\_bullet)  
 self.cooldown = 60  
 else:  
 print\_text('Time: ' + str(self.cooldown // 10), 20, 150)  
 self.cooldown -= 1  
  
 for bullet in all\_btn\_bullets:  
 if not bullet.move():  
 all\_btn\_bullets.remove(bullet)  
  
 for bullet in all\_ms\_bullets:  
 if not bullet.move\_to():  
 all\_ms\_bullets.remove(bullet)  
  
 heart.move()  
 self.heart\_plus(heart)  
  
 energ.move()  
 self.energ\_plus(energ)  
  
 if self.check\_collision(block\_arr):  
 game = False  
 self.show\_health()  
 self.show\_energy()  
  
 self.draw\_enemy(all\_enemies)  
 self.check\_enemies\_dmg(all\_btn\_bullets, all\_enemies)  
  
 pygame.display.update()  
 clock.tick(65)  
  
 return self.game\_over()  
  
 def jump(self):  
 if self.jump\_counter >= -30:  
 p.usr\_y -= self.jump\_counter / 2  
 self.jump\_counter -= 1  
 else:  
 self.jump\_counter = 30  
 self.make\_jump = False  
  
 def create\_block\_arr(self, array):  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
 array.append(Object(display\_width + 10, height, width, img, 4))  
  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
 array.append(Object(display\_width + 300, height, width, img, 4))  
  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
 array.append(Object(display\_width + 600, height, width, img, 4))  
  
 def object\_return(self, objects, obj):  
 radius = self.find\_radius(objects)  
  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
  
 obj.return\_self(radius, height, width, img)  
  
 @staticmethod  
 def find\_radius(array):  
 maximum = max(array[0].x, array[1].x, array[2].x)  
  
 if maximum < display\_width:  
 radius = display\_width  
 if radius - maximum < 50:  
 radius += 150  
 else:  
 radius = maximum  
 choice = random.randrange(0, 5)  
 if choice == 0:  
 radius += random.randrange(10, 15)  
 else:  
 radius += random.randrange(200, 350)  
  
 return radius  
  
 def draw\_array(self, array):  
 for block in array:  
 check = block.move()  
 if not check:  
 self.object\_return(array, block)  
  
 def draw\_volk(self):  
 if self.img\_counter == 36:  
 self.img\_counter = 0  
 display.blit(volk\_img[self.img\_counter // 6], (p.usr\_x, p.usr\_y))  
 self.img\_counter += 1  
  
 @staticmethod  
 def pause():  
 paused = True  
  
 pygame.mixer\_music.pause()  
 pygame.mixer.Sound.play(pause\_sound)  
 pygame.mixer.Sound.play(pause\_sound, -1)  
 pause\_sound.set\_volume(0.2)  
  
 while paused:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 print\_text0('PAUSE', 320, 30)  
 print\_text('press ENTER, to continue', 258, 115)  
  
 keys = pygame.key.get\_pressed()  
 if keys[pygame.K\_RETURN]:  
 pygame.mixer\_music.unpause()  
 pause\_sound.set\_volume(0)  
 paused = False  
  
 pygame.display.update()  
 clock.tick(15)  
  
 def check\_collision(self, barriers):  
 for barrier in barriers:  
 if barrier.y >= 0:  
 if not self.make\_jump:  
 if barrier.x <= p.usr\_x + p.usr\_width - 80 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 elif self.jump\_counter == 10:  
 if p.usr\_y + p.usr\_height - 5 >= barrier.y:  
 if barrier.x <= p.usr\_x + p.usr\_width - 5 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 elif self.jump\_counter >= -1:  
 if p.usr\_y + p.usr\_height - 5 >= barrier.y:  
 if barrier.x <= p.usr\_x + p.usr\_width - 50 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 else:  
 if p.usr\_y + p.usr\_height - 5 >= barrier.y:  
 if barrier.x <= p.usr\_x + 5 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 return False  
  
 def count\_scores(self, barriers):  
 above\_block = 0  
 if -20 <= self.jump\_counter < 25:  
 for barrier in barriers:  
 if p.usr\_y + p.usr\_height - 5 <= barrier.y:  
 if barrier.x <= p.usr\_x <= barrier.x + barrier.width:  
 above\_block += 1  
 elif barrier.x <= p.usr\_x + p.usr\_width <= barrier.x + barrier.width:  
 above\_block += 1  
 self.max\_above = max(self.max\_above, above\_block)  
 else:  
 if self.jump\_counter == -30:  
 self.scores += self.max\_above  
 self.max\_above = 0  
  
 def game\_over(self):  
 pygame.mixer\_music.pause()  
 pygame.mixer.Sound.play(quit\_sound)  
 pygame.mixer.Sound.play(quit\_sound, -1)  
 quit\_sound.set\_volume(0.2)  
  
 if self.scores > self.max\_scores:  
 self.max\_scores = self.scores  
  
 stopped = True  
 while stopped:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 print\_text0('GAME OVER', 240, 30)  
 print\_text('press Enter - to play again', 260, 110)  
 print\_text('press ESC - to exit', 260, 150)  
 print\_text('Max score: ' + str(self.max\_scores), 260, 190)  
  
 keys = pygame.key.get\_pressed()  
 if keys[pygame.K\_RETURN]:  
 quit\_sound.set\_volume(0)  
 return True  
 if keys[pygame.K\_ESCAPE]:  
 quit\_sound.set\_volume(0)  
 menu\_sound.set\_volume(0.2)  
 return False  
  
 pygame.display.update()  
 clock.tick(15)  
  
 def show\_health(self):  
 show = 0  
 x = 10  
 while show != self.health:  
 display.blit(health\_img, (x, 10))  
 x += 50  
 show += 1  
  
 def check\_health(self):  
 self.health -= 1  
 if self.health == 0:  
 return False  
 else:  
 pygame.mixer.Sound.play(loss\_sound)  
 loss\_sound.set\_volume(0.2)  
 return True  
  
 def heart\_plus(self, heart):  
 if p.usr\_x <= heart.x <= p.usr\_x + p.usr\_width:  
 if p.usr\_y <= heart.y <= p.usr\_y + p.usr\_height:  
 pygame.mixer.Sound.play(heart\_sound)  
 if self.health <= 2:  
 self.health += 1  
  
 radius = display\_width + random.randrange(500, 700)  
 heart.return\_self(radius, heart.y, heart.width, heart.image)  
  
 def show\_energy(self):  
 show = 0  
 x = 10  
 while show != self.energy:  
 display.blit(energy\_img, (x, 60))  
 x += 50  
 show += 1  
  
 def energ\_plus(self, energ):  
 if p.usr\_x <= energ.x <= p.usr\_x + p.usr\_width:  
 if p.usr\_y <= energ.y <= p.usr\_y + p.usr\_height:  
 pygame.mixer.Sound.play(heart\_sound)  
 if self.energy <= 2:  
 self.energy += 1  
  
 radius = display\_width + random.randrange(100, 800)  
 energ.return\_self(radius, energ.y, energ.width, energ.image)  
  
 @staticmethod  
 def draw\_enemy(enemies):  
 for enemy in enemies:  
 action = enemy.draw()  
 if action == 1:  
 enemy.show()  
 elif action == 2:  
 enemy.hide()  
 else:  
 enemy.shoot()  
  
 @staticmethod  
 def check\_enemies\_dmg(bullets, enemies):  
 for enemy in enemies:  
 for bullet in bullets:  
 enemy.check\_dmg(bullet)

Также есть файлы effects.py, enemy.py, button.py, sounds.py, images.py, bullet.py, object.py, parameters.py, main.py, который в совю очередь отвечают:

effects.py – за всю текстовую информацию, которая присутствует в игре (фунции вывода текста на экран).

sounds.py – за всё звуковое сопровождение в игре (функции загрузки музыки).

images.py – за все изображения и картинки, использующиеся в проекте (функции загрузки изображения).

bullet.py – за все пули, которыми стреляет Волк и Заяц (class Bullet).

object.py – за все объекты, типа припятсвтвий. Используемых в проекте (class Object).

enemy.py – файл отвечающий за прорисовку и работу Зайца, как объекта (class Enemies).

parameters.py – файл, хранящий настройки проекта (настройки дисплея).

main.py – файл отвечающий за запуск программы.

**5. Шаблон проектирования практических задач**

[Python](https://proglib.io/p/python-from-newbie-to-professional/) – это мощный динамический язык высокого уровня. Он обладает простым синтаксисом, отличной модульной структурой и огромным количеством удобных функций.

Подходит ли Python для паттернов?

## Но поскольку Python настолько мощный и гибкий, разработчикам необходимы некоторые правила (или шаблоны) для программирования.

## Любой язык программирования подходит для паттернов, в том числе и Python.

## Шаблоны программирования тесно связаны с контекстом их использования. Синтаксис и особенности конкретного языка задают собственные правила для разработчиков. Эти факторы в разных языках могут различаться и обычно не имеют конкретной цели – они, по большей части, технические. С другой стороны, те ограничения, которые связаны с использованием шаблонов, целенаправлены. Они рассказывают, как нужно делать и как не нужно.

## Философия Python базируется на хорошо продуманных лучших практиках программирования. Многие шаблоны проектирования уже встроены в язык. Разработчики используют их, даже не задумываясь. Ряд популярных паттернов очень легко реализовать благодаря динамической природе языка. А некоторые не используются в Python, так как в них нет необходимости.

## Например, смысл шаблона Фабрика – скрывать логику создания новых объектов. Но в Python это не нужно, так как этот процесс динамичен по своей сути. Конечно, Фабрику можно реализовать, если есть желание. Иногда это действительно полезно, но такие случаи – больше исключение, нежели правило.

## Композиция vs. Наследование.

## Композиция в Python элегантна и естественна, это принцип для языка очень близок.

Вместо подобного фрагмента:

class User(DbObject):

pass

Разработчик может написать что-то вроде этого:

class User:

\_persist\_methods = ['get', 'save', 'delete']

def \_\_init\_\_(self, persister):

self.\_persister = persister

def \_\_getattr\_\_(self, attribute):

if attribute in self.\_persist\_methods:

return getattr(self.\_persister, attribute)

Преимущества второго варианта очевидны. Экземпляр persister вводится прямо во время выполнения программы! Таким образом, сегодня это может быть реляционная база данных, а завтра что-то другое. Важно лишь, чтобы сохранялся необходимый интерфейс.

Итератор.

Итераторы встроены в Python. Это одна из самых мощных возможностей языка. Во многом именно итераторы делают его таким удобным. Чтобы разобраться в паттерне Итератор, следует просто изучить механизм работы итераторов и генераторов языка.

Цепочка обязанностей.

Этот шаблон предлагает удобный способ обработать запрос с использованием нескольких различных методов. Каждый из них может обращаться к определенной части запроса.

Как известно, одним из лучших принципов хорошего кода является принцип единой ответственности. Каждая часть кода должен делать одно, и только одно.

Как раз этим и занимается Цепочка обязанностей.

Например, если необходимо отфильтровать некоторый контент, можно создать различные фильтры. Каждый из них реализует один точный и четко определенный тип фильтрации, например, нецензурную лексику или рекламу.

class ContentFilter(object):

def \_\_init\_\_(self, filters=None):

self.\_filters = list()

if filters is not None:

self.\_filters += filters

def filter(self, content):

for filter in self.\_filters:

content = filter(content)

return content

filter = ContentFilter([

offensive\_filter,

ads\_filter,

porno\_video\_filter])

filtered\_content = filter.filter(content)

### Команда

Шаблоны программирования не придумывают, их обнаруживают. Они существуют, программист просто должен найти их и использовать.

Иногда требуется разделить во времени подготовку операции и ее совершение. Все подготовительные шаги объединяются в одной Команде. Это позволяет добавлять дополнительные функциональные возможности. Так можно реализовать отмену совершенного действия или его повтор.

Простой и часто используемый пример на языке Python:

class RenameFileCommand(object):

def \_\_init\_\_(self, from\_name, to\_name):

self.\_from = from\_name

self.\_to = to\_name

def execute(self):

os.rename(self.\_from, self.\_to)

def undo(self):

os.rename(self.\_to, self.\_from)

class History(object):

def \_\_init\_\_(self):

self.\_commands = list()

def execute(self, command):

self.\_commands.append(command)

command.execute()

def undo(self):

self.\_commands.pop().undo()

history = History()

history.execute(RenameFileCommand('docs/cv.doc', 'docs/cv-en.doc'))

history.execute(RenameFileCommand('docs/cv1.doc', 'docs/cv-bg.doc'))

history.undo()

history.undo()

## Порождающие Паттерны.

Python очень гибок, когда дело касается создания объектов. Поэтому порождающие шаблоны в нем обычно не используются. Можно сказать, что Фабрика уже встроена в язык.

Решения этой группы позволяют скрывать логику создания объектов. Таким образом, можно получить экземпляр класса, не используя оператор new. Но в Python и так нет этого оператора!

Тем не менее, порождающие шаблоны можно реализовать средствами языка.

### Одиночка

Шаблон Одиночка используется, если нужны гарантии, что существует единственный экземпляр данного класса. Во время выполнения программы не должны появляться другие. На самом деле, в Python проще намеренно создать один экземпляр, а затем использовать его.

Python позволяет вносить изменения в процесс создания экземпляра класса. Для этого существует метод \_\_new\_\_. Им и нужно воспользоваться для реализации паттерна Одиночка.

class Logger(object):

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if not hasattr(cls, '\_logger'):

cls.\_logger = super(Logger, cls

).\_\_new\_\_(cls, \*args, \*\*kwargs)

return cls.\_logger

Для решения той же задачи в Python есть ряд альтернатив:

* использование модуля;
* создание одного экземпляра на верхнем уровне приложения, например, в файле конфигурации;
* передача экземпляра каждому объекту, которому он нужен.

Последнее решение – это инъекция зависимости. Ее тоже можно отнести к шаблонам проектирования.

## Внедрение зависимости

Этот механизм можно отнести к группе порождающих. Он определяет, где и когда создается экземпляр класса.

Смысл шаблона в том, что зависимость производится в одном месте, а потребляется в другом. Код пользователя просто получает готовый внешний объект и использует его.

Если вам хочется пить, не нужно идти к холодильнику и брать из него напиток самостоятельно. Обозначьте свою потребность. Скажите родителям, что хотите выпить что-нибудь с обедом. Это неплохой пример внедрения зависимостей.

Python предлагает удобные способы реализации шаблона. Задумайтесь о том, как это выглядело бы на Java или C#. Простота и красота Python становится еще очевиднее.

class Command:

def \_\_init\_\_(self, authenticate=None, authorize=None):

self.authenticate = authenticate or self.\_not\_authenticated

self.authorize = authorize or self.\_not\_autorized

def execute(self, user, action):

self.authenticate(user)

self.authorize(user, action)

return action()

if in\_sudo\_mode:

command = Command(always\_authenticated, always\_authorized)

else:

command = Command(config.authenticate, config.authorize)

command.execute(current\_user, delete\_user\_action)

В Command вводятся методы аутентификации пользователя. При этом классу не нужно беспокоиться о том, как они реализованы. Он может просто выполнять необходимые действия. Таким образом, выполнение программы не зависит от конкретных механизмов.

Это был пример введения зависимостей через конструктор. То же самое можно делать непосредственно свойства объекта.

command = Command()

if in\_sudo\_mode:

command.authenticate = always\_authenticated

command.authorize = always\_authorized

else:

command.authenticate = config.authenticate

command.authorize = config.authorize

command.execute(current\_user, delete\_user\_action)

Узнать больше о мощном механизме инъекции зависимостей можно [здесь](http://stackoverflow.com/questions/2461702/why-is-ioc-di-not-common-in-python) и [здесь](https://ru.wikipedia.org/wiki/%D0%92%D0%BD%D0%B5%D0%B4%D1%80%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8).

Использование этого шаблона раскрывает большие возможности для [модульного тестирования](https://www.toptal.com/python/an-introduction-to-mocking-in-python). Оно позволяет менять данные прямо на лету. Многое сразу становится проще, не так ли?

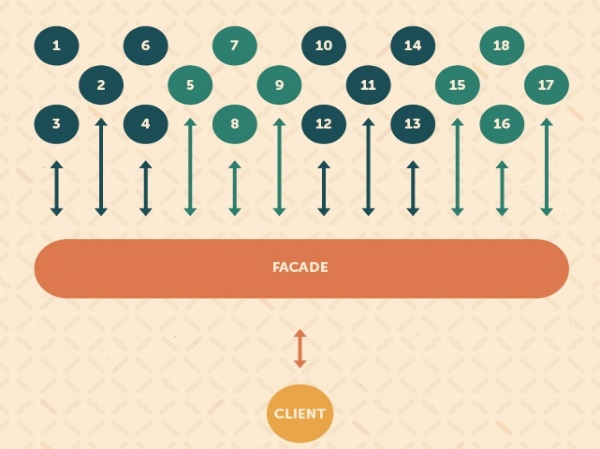
## Структурные шаблоны.

[Эти паттерны](https://proglib.io/p/structural-patterns/) занимаются объединением отдельных классов и объектов в сложные группы.

### Фасад.

Пожалуй, самый известный шаблон проектирования в Python.

Представьте, что у вас есть система со значительным количеством объектов. Каждый объект предлагает богатый набор методов API. Возможности этой системы велики, но ее интерфейс слишком сложный. Для удобства можно добавить новый объект, представляющий хорошо продуманные комбинации методов. Это и есть Фасад.



Python предлагает очень элегантную реализацию шаблона.

class Car(object):

def \_\_init\_\_(self):

self.\_tyres = [Tyre('front\_left'),

Tyre('front\_right'),

Tyre('rear\_left'),

Tyre('rear\_right'), ]

self.\_tank = Tank(70)

def tyres\_pressure(self):

return [tyre.pressure for tyre in self.\_tyres]

def fuel\_level(self):

return self.\_tank.level

Без всяких трюков и фокусов класс Car стал Фасадом.

Декоратор.

Хорошая новость! Декораторы – необычайно удобная штука, и они встроены в Python по умолчанию.

[Python – замечательный язык](https://proglib.io/p/greatest-python-one-liners/). Само его использование учит следовать лучшим практикам программирования. Их даже не обязательно осознавать, они интуитивны, словно являются второй натурой языка. Это ценят в нем и новички, и опытные разработчики.

Шаблон Декоратор позволяет расширять функциональность без использования наследования.

def execute(user, action):

self.authenticate(user)

self.authorize(user, action)

return action()

С этим примером что-то не так. Функция execute выполняет больше одной обязанности, что не соответствует принципу единой ответственности.

Было бы лучше сделать так:

def execute(action):

return action()

А любые функции авторизации и аутентификации можно реализовать в другом месте:

def execute(action, \*args, \*\*kwargs):

return action()

def autheticated\_only(method):

def decorated(\*args, \*\*kwargs):

if check\_authenticated(kwargs['user']):

return method(\*args, \*\*kwargs)

else:

raise UnauthenticatedError

return decorated

def authorized\_only(method):

def decorated(\*args, \*\*kwargs):

if check\_authorized(kwargs['user'], kwargs['action']):

return method(\*args, \*\*kwargs)

else:

raise UnauthorizeddError

return decorated

execute = authenticated\_only(execute)

execute = authorized\_only(execute)

Метод execute() теперь намного легче читать, и он выполняет только одну обязанность. Его функционал декорируется аутентификацией и авторизацией.

То же самое можно написать, используя синтаксис встроенного декоратора Python:

def autheticated\_only(method):

def decorated(\*args, \*\*kwargs):

if check\_authenticated(kwargs['user']):

return method(\*args, \*\*kwargs )

else:

raise UnauthenticatedError

return decorated

def authorized\_only(method):

def decorated(\*args, \*\*kwargs):

if check\_authorized(kwargs['user'], kwargs['action']):

return method(\*args, \*\*kwargs)

else:

raise UnauthorizedError

return decorated

@authorized\_only

@authenticated\_only

def execute(action, \*args, \*\*kwargs):

return action()

Декорировать можно не только функции, но и целые классы. Единственное требование состоит в том, что они должны быть вызываемыми. Но в Python нет проблем с этим: нужно лишь определить метод \_\_call \_\_ (self).

Еще много интересного можно найти в [модуле functools](https://docs.python.org/2/library/functools.html).

## Вывод

Использовать шаблоны проектирования в Python очень легко. На нем вообще легко программировать. Недаром главная заповедь языка – «Простое лучше, чем сложное».

Обратите внимание, ни для одного из паттернов не приведена полномасштабная реализация. Их нужно «почувствовать» и реализовать оптимальным образом. Каким именно, зависит от стиля разработчика и потребностей проекта. А Python предоставит всю необходимую мощность для создания гибкого и многоразового кода.

Однако гибкость языка дает еще больше возможностей. Она позволяет писать действительно плохой код. Не делайте этого! Следуйте принципу [DRY](https://ru.wikipedia.org/wiki/Don%E2%80%99t_repeat_yourself) и не пишите строки длиной более 80 символов. Используйте шаблоны проектирования там, где они применимы. Это один из лучших способов учиться у других и бесплатно получать опыт.

**Заключение**

Целью курсовой работы было создание 2D игры на языке программирования Python.

В ходе выполнения проекта были изучен язык Python (ранее не был я ним хорошо знаком) и работы с ним в среде разработки PyCharm Community. Для разработки данной игры требовалось решить множество задач, вследствие чего было проанализировано и изучено большое количество информации об классах, объетах, логики реализация физического мира. Также была изучена информация о работе со звуком, изображениями и текстом. Подробно был изучен вопрос дополнительных библиотек и модулей (необходимых для работы с изображениями, звуком и т.д.), в частоности самой главной библиотекой для работы - Pygame. Также было установлена модель проекта и составлено его описание. Были изучены архитектурные шаблоны проектирования, в частности MVC. Также необходимо было изучить различный шаблоны проектирования в Python и определиться с выбором.

Была установлена библиотека Pygame, позволяющая создание графического интерфейса, для моделирования 2D игры. В дальнейшем был разработан и реализован пользовательский интерфейс и функционал работы самой игры.

Убедившись в корректности работы игры после пройденных тестов, можно сказать, что результат курсовой работы «Разработка 2D игры» полностью соответствует всем исходным требованиям.

**Список литературных источников**

* 1. <https://python-scripts.com/pycharm-download>
  2. <https://habr.com/ru/post/491916/>
  3. <https://metanit.com/python/tutorial/1.3.php>
  4. <https://proglib.io/p/python-oop/>
  5. <https://python-scripts.com/object-oriented-programming-in-python>
  6. <https://gcup.ru/publ/engines/osnovy_pygame_vvedenie/2-1-0-257>
  7. <https://pythonru.com/uroki/biblioteka-pygame-chast-1-vvedenie>
  8. <https://pythonru.com/uroki>

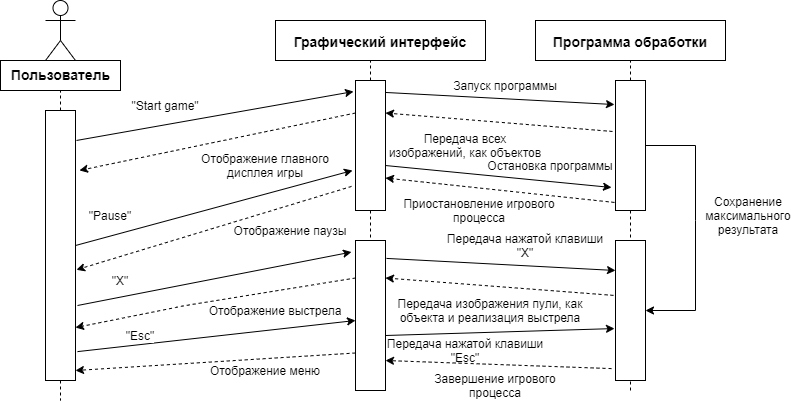
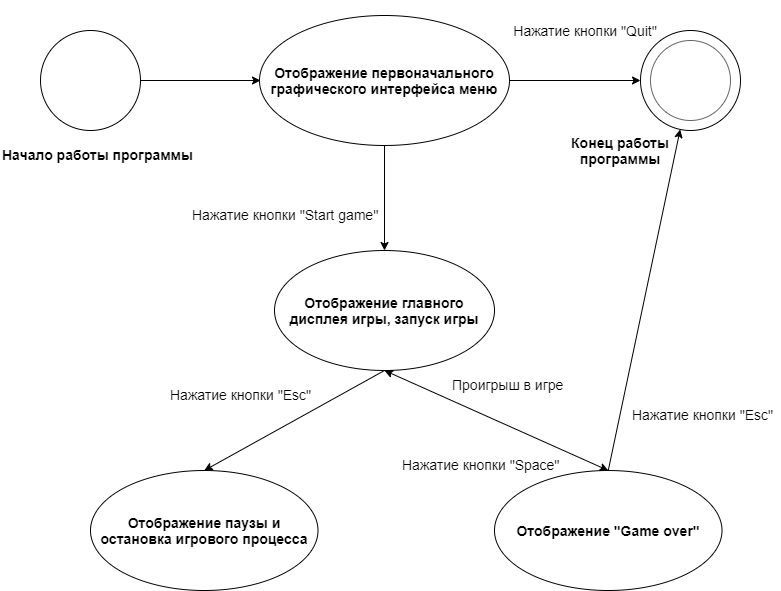
**Приложения**

**Листинг кода (game.py)**

import parameters as p  
from button import \*  
from enemy import \*  
from object import \*  
from effects import \*  
from images import \*  
  
  
class Game:  
 def \_\_init\_\_(self):  
 pygame.display.set\_caption('Ny pogodi')  
 pygame.display.set\_icon(icon)  
  
 pygame.mixer\_music.load('music/muz.mp3')  
 pygame.mixer\_music.set\_volume(0.2)  
  
 self.block\_options = [60, 430, 54, 400, 60, 421]  
 self.img\_counter = 0  
 self.health = 2  
 self.energy = 2  
 self.jump\_counter = 30  
 self.scores = 0  
 self.max\_scores = 0  
 self.max\_above = 0  
 self.cooldown = 0  
 self.make\_jump = False  
  
 def show\_menu(self):  
 pygame.mixer\_music.pause()  
 pygame.mixer.Sound.play(menu\_sound)  
 pygame.mixer.Sound.play(menu\_sound, -1)  
 menu\_sound.set\_volume(0.2)  
  
 start\_btn = Button(200, 50)  
 quit\_btn = Button(120, 50)  
  
 show = True  
  
 while show:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 display.blit(fon, (0, 0))  
 display.blit(menu\_bckgr, (0, 0))  
 display.blit(ny, (195, 50))  
  
 if self.img\_counter == 36:  
 self.img\_counter = 0  
 display.blit(volk\_img[self.img\_counter // 6], (280, 130))  
 self.img\_counter += 1  
 start\_btn.draw(300, 350, ' Start game', self.start\_game)  
 quit\_btn.draw(340, 410, ' Quit', quit)  
 pygame.display.update()  
 clock.tick(60)  
  
 def start\_game(self):  
 self.health = 2  
 self.energy = 2  
  
 while self.game\_cycle():  
 self.scores = 0  
 self.health = 2  
 self.make\_jump = False  
 self.jump\_counter = 30  
 p.usr\_y = display\_height - usr\_height - 110  
 self.cooldown = 0  
  
 def game\_cycle(self):  
  
 pygame.mixer\_music.play(-1)  
  
 game = True  
 block\_arr = []  
 self.create\_block\_arr(block\_arr)  
  
 heart = Object(display\_width, 220, 30, health\_img, 4)  
 energ = Object(display\_width, 220, 30, energy\_img, 4)  
  
 all\_btn\_bullets = []  
 all\_ms\_bullets = []  
  
 zaych1 = Enemies(-150)  
  
 all\_enemies = [zaych1]  
  
 while game:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 keys = pygame.key.get\_pressed()  
 mouse = pygame.mouse.get\_pos()  
 click = pygame.mouse.get\_pressed()  
  
 if keys[pygame.K\_SPACE]:  
 self.make\_jump = True  
 if keys[pygame.K\_ESCAPE]:  
 menu\_sound.set\_volume(0.2)  
 self.pause()  
  
 if self.make\_jump:  
 self.jump()  
  
 self.count\_scores(block\_arr)  
  
 display.blit(land, (0, 0))  
 print\_text1('Score: ' + str(self.scores), 15, 550)  
  
 self.draw\_array(block\_arr)  
 self.draw\_volk()  
  
 if not self.cooldown:  
 if self.energy >= 1:  
 if keys[pygame.K\_x]:  
 self.energy -= 1  
 pygame.mixer.Sound.play(bullet\_sound)  
 all\_btn\_bullets.append(Bullet(p.usr\_x + p.usr\_width, p.usr\_y + 28))  
 self.cooldown = 60  
 elif click[0]:  
 pygame.mixer.Sound.play(bullet\_sound)  
 add\_bullet = Bullet(p.usr\_x + p.usr\_width, p.usr\_y + 28)  
 add\_bullet.find\_path(mouse[0], mouse[1])  
 all\_btn\_bullets.append(add\_bullet)  
 self.cooldown = 60  
 else:  
 print\_text('Time: ' + str(self.cooldown // 10), 20, 150)  
 self.cooldown -= 1  
  
 for bullet in all\_btn\_bullets:  
 if not bullet.move():  
 all\_btn\_bullets.remove(bullet)  
  
 for bullet in all\_ms\_bullets:  
 if not bullet.move\_to():  
 all\_ms\_bullets.remove(bullet)  
  
 heart.move()  
 self.heart\_plus(heart)  
  
 energ.move()  
 self.energ\_plus(energ)  
  
 if self.check\_collision(block\_arr):  
 game = False  
 self.show\_health()  
 self.show\_energy()  
  
 self.draw\_enemy(all\_enemies)  
 self.check\_enemies\_dmg(all\_btn\_bullets, all\_enemies)  
  
 pygame.display.update()  
 clock.tick(65)  
  
 return self.game\_over()  
  
 def jump(self):  
 if self.jump\_counter >= -30:  
 p.usr\_y -= self.jump\_counter / 2  
 self.jump\_counter -= 1  
 else:  
 self.jump\_counter = 30  
 self.make\_jump = False  
  
 def create\_block\_arr(self, array):  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
 array.append(Object(display\_width + 10, height, width, img, 4))  
  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
 array.append(Object(display\_width + 300, height, width, img, 4))  
  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
 array.append(Object(display\_width + 600, height, width, img, 4))  
  
 def object\_return(self, objects, obj):  
 radius = self.find\_radius(objects)  
  
 choice = random.randrange(0, 3)  
 img = block\_img[choice]  
 width = self.block\_options[choice \* 2]  
 height = self.block\_options[choice \* 2 + 1]  
  
 obj.return\_self(radius, height, width, img)  
  
 @staticmethod  
 def find\_radius(array):  
 maximum = max(array[0].x, array[1].x, array[2].x)  
  
 if maximum < display\_width:  
 radius = display\_width  
 if radius - maximum < 50:  
 radius += 150  
 else:  
 radius = maximum  
 choice = random.randrange(0, 5)  
 if choice == 0:  
 radius += random.randrange(10, 15)  
 else:  
 radius += random.randrange(200, 350)  
  
 return radius  
  
 def draw\_array(self, array):  
 for block in array:  
 check = block.move()  
 if not check:  
 self.object\_return(array, block)  
  
 def draw\_volk(self):  
 if self.img\_counter == 36:  
 self.img\_counter = 0  
 display.blit(volk\_img[self.img\_counter // 6], (p.usr\_x, p.usr\_y))  
 self.img\_counter += 1  
  
 @staticmethod  
 def pause():  
 paused = True  
  
 pygame.mixer\_music.pause()  
 pygame.mixer.Sound.play(pause\_sound)  
 pygame.mixer.Sound.play(pause\_sound, -1)  
 pause\_sound.set\_volume(0.2)  
  
 while paused:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 print\_text0('PAUSE', 320, 30)  
 print\_text('press ENTER, to continue', 258, 115)  
  
 keys = pygame.key.get\_pressed()  
 if keys[pygame.K\_RETURN]:  
 pygame.mixer\_music.unpause()  
 pause\_sound.set\_volume(0)  
 paused = False  
  
 pygame.display.update()  
 clock.tick(15)  
  
 def check\_collision(self, barriers):  
 for barrier in barriers:  
 if barrier.y >= 0:  
 if not self.make\_jump:  
 if barrier.x <= p.usr\_x + p.usr\_width - 80 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 elif self.jump\_counter == 10:  
 if p.usr\_y + p.usr\_height - 5 >= barrier.y:  
 if barrier.x <= p.usr\_x + p.usr\_width - 5 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 elif self.jump\_counter >= -1:  
 if p.usr\_y + p.usr\_height - 5 >= barrier.y:  
 if barrier.x <= p.usr\_x + p.usr\_width - 50 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 else:  
 if p.usr\_y + p.usr\_height - 5 >= barrier.y:  
 if barrier.x <= p.usr\_x + 5 <= barrier.x + barrier.width:  
 if self.check\_health():  
 self.object\_return(barriers, barrier)  
 return False  
 else:  
 return True  
 return False  
  
 def count\_scores(self, barriers):  
 above\_block = 0  
 if -20 <= self.jump\_counter < 25:  
 for barrier in barriers:  
 if p.usr\_y + p.usr\_height - 5 <= barrier.y:  
 if barrier.x <= p.usr\_x <= barrier.x + barrier.width:  
 above\_block += 1  
 elif barrier.x <= p.usr\_x + p.usr\_width <= barrier.x + barrier.width:  
 above\_block += 1  
 self.max\_above = max(self.max\_above, above\_block)  
 else:  
 if self.jump\_counter == -30:  
 self.scores += self.max\_above  
 self.max\_above = 0  
  
 def game\_over(self):  
 pygame.mixer\_music.pause()  
 pygame.mixer.Sound.play(quit\_sound)  
 pygame.mixer.Sound.play(quit\_sound, -1)  
 quit\_sound.set\_volume(0.2)  
  
 if self.scores > self.max\_scores:  
 self.max\_scores = self.scores  
  
 stopped = True  
 while stopped:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 print\_text0('GAME OVER', 240, 30)  
 print\_text('press Enter - to play again', 260, 110)  
 print\_text('press ESC - to exit', 260, 150)  
 print\_text('Max score: ' + str(self.max\_scores), 260, 190)  
  
 keys = pygame.key.get\_pressed()  
 if keys[pygame.K\_RETURN]:  
 quit\_sound.set\_volume(0)  
 return True  
 if keys[pygame.K\_ESCAPE]:  
 quit\_sound.set\_volume(0)  
 menu\_sound.set\_volume(0.2)  
 return False  
  
 pygame.display.update()  
 clock.tick(15)  
  
 def show\_health(self):  
 show = 0  
 x = 10  
 while show != self.health:  
 display.blit(health\_img, (x, 10))  
 x += 50  
 show += 1  
  
 def check\_health(self):  
 self.health -= 1  
 if self.health == 0:  
 return False  
 else:  
 pygame.mixer.Sound.play(loss\_sound)  
 loss\_sound.set\_volume(0.2)  
 return True  
  
 def heart\_plus(self, heart):  
 if p.usr\_x <= heart.x <= p.usr\_x + p.usr\_width:  
 if p.usr\_y <= heart.y <= p.usr\_y + p.usr\_height:  
 pygame.mixer.Sound.play(heart\_sound)  
 if self.health <= 2:  
 self.health += 1  
  
 radius = display\_width + random.randrange(500, 700)  
 heart.return\_self(radius, heart.y, heart.width, heart.image)  
  
 def show\_energy(self):  
 show = 0  
 x = 10  
 while show != self.energy:  
 display.blit(energy\_img, (x, 60))  
 x += 50  
 show += 1  
  
 def energ\_plus(self, energ):  
 if p.usr\_x <= energ.x <= p.usr\_x + p.usr\_width:  
 if p.usr\_y <= energ.y <= p.usr\_y + p.usr\_height:  
 pygame.mixer.Sound.play(heart\_sound)  
 if self.energy <= 2:  
 self.energy += 1  
  
 radius = display\_width + random.randrange(100, 800)  
 energ.return\_self(radius, energ.y, energ.width, energ.image)  
  
 @staticmethod  
 def draw\_enemy(enemies):  
 for enemy in enemies:  
 action = enemy.draw()  
 if action == 1:  
 enemy.show()  
 elif action == 2:  
 enemy.hide()  
 else:  
 enemy.shoot()  
  
 @staticmethod  
 def check\_enemies\_dmg(bullets, enemies):  
 for enemy in enemies:  
 for bullet in bullets:  
 enemy.check\_dmg(bullet)

|  |
| --- |
|  |

|  |
| --- |
|  |

********

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *БГУИР. 425723. 011* | | | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № докум | Подпись | Дата |
| Разраб. | | Харко |  |  | Диаграмма состояний Диаграмма последовательности | Литера | | | Лист | | Листов | |
| Пров. | | Михалькевич |  |  |  | у |  | | 1 | | 1 | |
|  | |  |  |  | Кафедра ПИКС, гр.814303 | | | | | | |
|  | |  |  |  |
|  | |  |  |  |