

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой \_\_ПОВТ и АС

\_\_\_\_\_\_\_\_\_\_\_\_ Долгов В.В.

подпись ФИО

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту по дисциплине Объектно-ориентированное программирование

на тему Объектно-ориентированная реализация игры «Snake»

Автор проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Андросов И.А.

подпись Ф.И.О

Направление/специальность, профиль/специализация:

09.03.04 Программная инженерия

коднаправления наименование профиля (специализации)

Обозначение курсового проекта (работы) КР.420000.000 Группа ВПР31

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Габрельян Б. В. подпись (должность, ФИО)

Проект (работа) защищен (а)

дата оценка подпись

г. Ростов-на-Дону

2022 г.



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой ПОВТ и АС

\_\_\_\_\_\_\_\_\_\_\_\_ Долгов В.В.

(подпись) (ФИО.)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.

**ЗАДАНИЕ**

к курсовой работе по дисциплине Объектно-ориентированное программирование

Студент Андросов И.А. КР.420000.000 Группа ВПР31

Тема Объектно-ориентированная реализация игры «Snake» выв

Срок представления проекта (работы) к защите « » 2022г.

Исходные данные для курсового проекта (работы)

Задание на выполнение курсовой работы

*Бейдер Ден, Чистый Python. Тонкости программирования для профи.. : Пер. с англ. - СПб. : ООО* *"Диалектика", 2020. - 240 с. : ил. - Парал. тит. англ. ISBN 978-5-907203-14-3*

*Лутц Марк., Язык программирования Python. ISBN 978-5-5043123-1-8*

*Core, 8-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2018 — 1328 с. : ил. — Парал. тит. англ.*

*ISBN 978-5-6040723-1-8*

Содержание пояснительной записки

ВВЕДЕНИЕ:

В разделе «Введение» рассматривается актуальность компьютерных игр жанра «Аркада» и приводится его описание.

Наименование и содержание разделов:

1. В разделе «Теоретический раздел» описываются: определение компьютерной игры, виды компьютерных игр, принципы ООП, а также производится постановка задачи.
2. В разделе «Программное конструирование» дается обоснование выбранного языка программирования, используемая библиотека, а также приводится описание основных классов и их функций.
3. В разделе «Конструирование игры» рассматривается графическая составляющая игры, то есть спрайты фона, игрока, врагов и жетона починки.
4. В разделе «Тестирование» представлена проверка игры на ее корректное функционирование во время игрового процесса.

ЗАКЛЮЧЕНИЕ:

В курсовой работе, в рамках объектно-ориентированной парадигмы программирования были реализованы все поставленные задачи, разработана рабочая игра.

Перечень графического материала:

1. Рисунок 2.1. UML диаграмма классов
2. Рисунок 3.1. Змейка
3. Рисунок 3.2. Яблоко
4. Рисунок 3.3. Фон
5. Рисунок 4.1. Начальный экран
6. Рисунок 4.2. Появление сцены и змеи
7. Рисунок 4.3. Увеличение змеи
8. Рисунок 4.4. Поражение игрока
9. Рисунок 4.5. Поражение игрока

Руководитель проекта (работы) Габрельян Б.В.

подпись, дата ФИО

Задание принял(а) к исполнению Андросов И.А.

подпись, дата ФИО

Содержание

[Содержание 4](#_Toc122869774)

[Введение 6](#_Toc122869775)

[1 Теоретический раздел 7](#_Toc122869776)

[1.1 Понятие компьютерной игры 7](#_Toc122869777)

[1.2 Понятие игры жанра «Аркада» 7](#_Toc122869778)

[1.3 Файтинг 7](#_Toc122869779)

[1.4 Платформеры 8](#_Toc122869780)

[1.5 Скроллеры 8](#_Toc122869781)

[1.6 Виртуальный тир 8](#_Toc122869782)

[1.7 Объектно-ориентированное программирование и его принципы 9](#_Toc122869783)

[1.8 Библиотека для создания игр PyGame 9](#_Toc122869784)

[1.9 Постановка задачи 10](#_Toc122869785)

[1.10 Выводы по главе 10](#_Toc122869786)

[2 Программное конструирование 11](#_Toc122869787)

[2.1 Выбор средств разработки 11](#_Toc122869788)

[2.1.1 Выбор языка программирования 11](#_Toc122869789)

[2.2.2 Выбор дополнительных инструментов 12](#_Toc122869790)

[2.3 Проектирование функциональных модулей программы 12](#_Toc122869791)

[2.4 UML-диаграммы классов 14](#_Toc122869792)

[2.5 Выводы по разделу 15](#_Toc122869793)

[3 Конструирование игры 16](#_Toc122869794)

[3.1 Графическое конструирование 16](#_Toc122869795)

[3.2 Конструирование локаций 17](#_Toc122869796)

[3.3 Выводы по главе 17](#_Toc122869797)

[4 Тестирование модулей программного средства 18](#_Toc122869798)

[4.1 Пошаговое тестирование 18](#_Toc122869799)

[4.2 Итоги тестирования 21](#_Toc122869800)

[Заключение 22](#_Toc122869801)

[Перечень использованных информационных ресурсов 23](#_Toc122869802)

[Приложение А Листинг программы 24](#_Toc122869803)

Введение

История игры «Змейка» началась за несколько лет до появления первых мобильных телефонов. В 1977 году компания Gremlin Industries выпустила [игровой автомат Hustle](http://www.arcade-history.com/?n=hustle&page=detail&id=1156), рассчитанный на одного или двух игроков, в которой нужно было управлять «змейками», направляя их на бессистемно появляющиеся цели. Для победы нужно было заполучить больше очков, чем у оппонента, преграждая по ходу игры ему путь к новым целям (в случае многопользовательской игры), или просто побить установленный на игровом автомате рекорд. В 1984 году Gremlin Industries была вынуждена закрыться, но игра Hustle начала набирать обороты: сначала появился порт для компьютеров TRS-80, затем для Commodore PET и Apple II.[6]

Оригинальная «Змейка» (Snake) от Nokia появилась в 1997 году благодаря стараниями разработчика Танели Орманто. В том же году компания выпустила первый телефон с этой игрой — Nokia 6110. Уже тогда игра была многопользовательской: телефоны общались через ИК-порты, ведь ни Bluetooth, ни тем более Wi-Fi в телефонах в то время не было. Сама змейка состояла из чёрных квадратов и могла двигаться в четырёх направлениях. Игровая зона, по которой передвигалось пресмыкающееся, была ограничена размерами экрана телефона: при ударе головы змейки о край телефона игра завершалась. «Змейка» приобрела невероятную популярность, сравнимую разве что с популярностью современных хитов «Angry Birds» и «Cut the Rope». В данной работе я разработаю свою собственную игру.[6]

# 1 Теоретический раздел

В данном разделе рассматриваются основные понятия жанра, основные понятия ООП, средства разработки компьютерных игр, определяется цель работы и задачи.

## **1.1 Понятие компьютерной игры**

Компью́терная игра́  — компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра.[1]

## 

## 1.2 Понятие игры жанра «Аркада»

Само слово arcade переводится как приключенческая игра. В наше время в России это термин, который обозначает, в первую очередь, компьютерные игры с простым процессом. Данный вид нельзя рассматривать как отдельный жанр, скорее это игровое направление. Весьма часто аркадам противопоставляют симуляторы.[8]

## **1.3** Файтинг

В драках два персонажа дерутся на арене, применяя различные удары, броски и комбинации. Характеризуется большим количеством персонажей (бойцов) и ударов (иногда больше ста для каждого персонажа). Жанр малопопулярен на PC из-за ориентации на совместную игру, а на клавиатуре довольно проблематично одновременно играть вдвоем. Однако хорошо развит на игровых приставках. По некоторым играм этого жанра даже проводятся мировые чемпионаты. Зачастую в некоторых играх на арене могут сойтись и четыре противника одновременно, например в Guilty Gear Isuka. Для управления в таких играх рекомендуется геймпад**.**[8]

## **1.4 Платформеры**

Понятие платформеров пришло с игровых приставок (консолей). Именно там этот жанр наиболее популярен. Основной задачей игрока является преодоление препятствий (ям, шипов, врагов и т. д.) с помощью прыжков. Зачастую приходится прыгать по абстрактно расставленным в воздухе «палочкам» (т. н. платформам), отсюда и пошло название жанра. **.**[8]

## 1.5 **Скроллеры**

В скроллерах экран непрерывно движется в одну из сторон, а игроку предлагается уничтожать появляющихся врагов и собирать появляющиеся бонусы. По направлению движения различают вертикальные (снизу вверх) и горизонтальные (слева направо) скроллеры. Жанр был очень популярен в середине 90-х годов, сейчас скроллеры практически не выпускаются. **.**[8]

## 1.6 Виртуальный тир

Впервые зародился на игровых автоматах, впоследствии перешел на многие игровые платформы, включая PC. Игровой процесс представляет собой отстрел неожиданно появляющихся врагов, но в отличие от экшенов мы не можем управлять движением игрока или камерой, всю игру мы как едем по «рельсам». В связи с этим иногда делают видеотиры, то есть всю игру снимают на видеокамеру, в определенных местах подставляя разные варианты видео отрывков. **.**[8]

## 1.7 Объектно-ориентированное программирование и его принципы

Объектно-ориентированное программирование (ООП) — это парадигма программирования, в которой концепциями являются понятия объектов и классов.   
Четыре основных принципа объектно-ориентированного программирования:  
 • Абстракция. Моделирование требуемых атрибутов и взаимодействий сущностей в виде классов для определения абстрактного представления системы.  
 • Инкапсуляция. Скрытие внутреннего состояния и функций объекта и предоставление доступа только через открытый набор функций.  
 • Наследование. Возможность создания новых абстракций на основе существующих.  
 • Полиморфизм. Возможность реализации наследуемых свойств или методов отличающимися способами в рамках множества абстракций.[3]

## 1.8 Библиотека для создания игр PyGame

Pygame — набор модулей языка программирования Python, предназначенный для написания компьютерных игр и мультимедиа-приложений. Pygame базируется на мультимедийной библиотеке SDL. Изначально Pygame был написан Питом Шиннерсом. **.**[5]

## **1.9 Постановка задачи**

Для создания игры типа «Змейка» для персональных компьютеров необходимо решить следующие задачи:

* Реализовать движение змейки
* Реализовать поедание яблока
* Реализовать поражение после попадание в стену
* Реализовать поражение после попадание в себя
* Реализовать генерацию разных уровней

## **1.10 Выводы по главе**

В данной главе были рассмотрены основные понятия ООП и виды компьютерных игр жанра «Аркада». Также была рассмотрена библиотека pygame.

# 2 Программное конструирование

В данном разделе проводится обоснование выбора средств разработки программного средства, описание структуры программного средства и основных функций, которые реализуют необходимый функционал.

## 2.1 Выбор средств разработки

Среда разработки была выбрана PyCharm из-за удобного интерфейса. PyCharm — интегрированная среда разработки для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django. PyCharm разработана компанией JetBrains на основе IntelliJ IDEA. **.**[7]

### 2.1.1 Выбор языка программирования

Требования по выбору языка программирования для решения поставленной задачи:

* высокая производительность;
* поддержка объектно-ориентированного программирования;
* наличие стандартных и сторонних подключаемых библиотек;
* контроль типов и обработка исключений;
* распространенность;
* доступность справочной информации и информационных ресурсов.

На основе этих требований был выбран язык программирования Python. Это современный объектно-ориентированный и типобезопасный язык программирования. Python позволяет разработчикам создавать разные типы безопасных и надежных приложений.[2]

Преимущества языка Python:

* понятный синтаксис;
* низкий порог вхождения;
* наличие стандартной библиотеки с широкими возможностями;
* возможность подключения сторонних библиотек;
* возможность выбора средств разработки;
* абстракция данных;

### 2.2.2 Выбор дополнительных инструментов

В качестве дополнительных инструментов были выбраны Queue для реализации очереди, а также collections. Модуль collections - предоставляет специализированные типы данных, на основе словарей, кортежей, множеств, списков. **.**[4]

## 2.3 Проектирование функциональных модулей программы

В процессе создания игры были реализованы следующие классы:

Для реализации поведения яблока на сцене бы создан класс Apple. Класс Apple описан в таблице 2.1

Таблица 2.1 – Класс Apple

|  |  |
| --- | --- |
| Класс | Apple |
| Описание класса | Представляет из себя общее поведение яблока |
| Методы класса | Описание метода |
| Collide\_with | Попадание змеи в яблоко |
| \_\_init\_\_ | Рисовка яблока |
| respawn | Появление яблока на карте |

Для создания змеи был создан класс Snake. Класс Snake описан в таблице 2.2.

Таблица 2.2 – Класс Snake

|  |  |
| --- | --- |
| Класс | Snake |
| Описание класса | Класс, описывающий змею |
| Методы класса | Описание метода |
| \_\_init\_\_ | Инициализация хитбокса и параметров движения змеи |
| update | Процесс добавления блока в туловищу |
| draw | Рисовка туловища и головы после поглощения яблока |
| change\_direction | Изменение направления движения |

Для реализации сцен был создан класс \_Scene. \_Scene описан в таблице 2.3

Таблица 2.3 – Класс \_Scene

|  |  |
| --- | --- |
| Класс | \_Scene |
| Описание класса | Класс, описывающий статичные сцены |
| Методы класса | Описание метода |
| startup | Рисовка начального экрана |
| reset | Логика кнопки рестарта игры |
| update | Сцена запуска игры |

Сцена поражения реализована классом AnyKey. Класс AnyKey описан в таблице 2.4

Таблица 2.4 – Класс AnyKey

|  |  |
| --- | --- |
| Класс | AnyKey |
| Описание класса | Класс, описывающий логику сцена поражения |
| Методы класса | Описание метода |
| make\_text | Создание текста на сцене |
| draw | Реализация мигающего текста |
| get\_event | Реализация запуска после нажатия |

Для реализации генерации игры был реализован класс Game. Класс Game описан в таблице 2.5.

Таблица 2.5 – Класс Game

|  |  |
| --- | --- |
| Класс | Game |
| Описание класса | Класс реализующий генерацию игры |
| Методы класса | Описание метода |
| reset | Подготовка к следующей попытке |
| make\_walls | Создание границ сцены и загрузка случайного уровня |
| draw | Прорисовка уровней и генерация сцены |

Для реализации механик игры был реализован класс Control. Класс Control описан в таблице 2.6

Таблица 2.6 – Класс Control

|  |  |
| --- | --- |
| Класс | Control |
| Описание класса | Класс реализующий cоздание игры |
| Методы класса | Описание метода |
| \_\_init\_\_ | Настройка и создание начальной сцены |
| event\_loop | Обработка выхода |
| update | Обновить текущую сцену и переключиться при необходимости |

Для реализации уровней игры был реализован класс Makedraw. Класс Maledraw описан в таблице 2.7

Таблица 2.7 – Класс Makedraw

|  |  |
| --- | --- |
| Класс | Makedraw |
| Описание класса | Класс, который хранит созданные уровни |

Для создания переходов был создан класс Start. Класс Start описан в таблице 2.8

Таблица 2.8 – Класс Start

|  |  |
| --- | --- |
| Класс | MainMenu |
| Описание класса | Класс, реализующий переходы между сценами проекта и выход из игры при нажатии кнопок |
| Методы класса | Описание метода |
| main | Переход на сцену с игровым процеcсом |

## 

## 2.4 UML-диаграммы классов

Для более подробного рассмотрения на рисунке 2.1 приведена UML диаграмма основных классов.

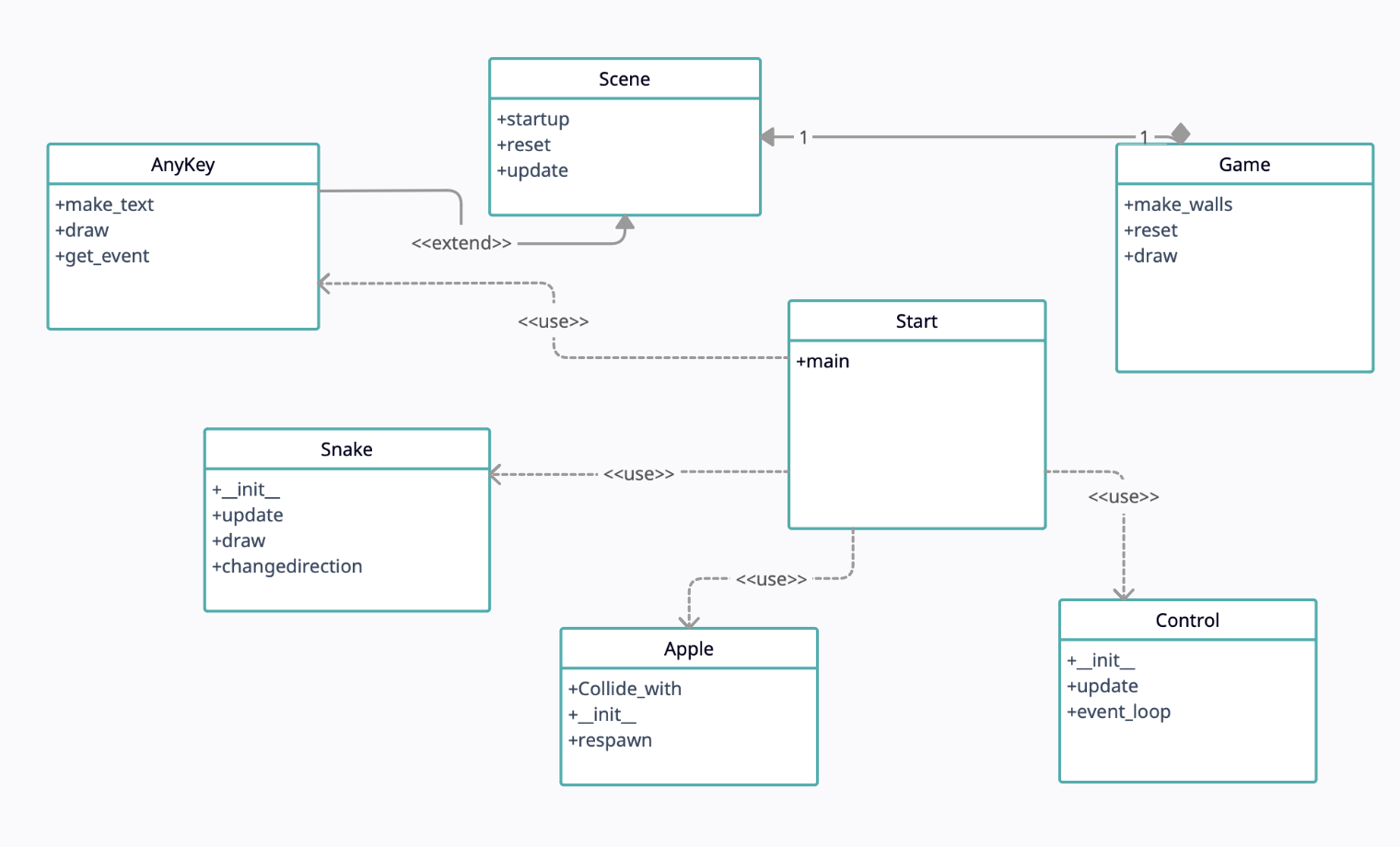


Рисунок 2.1 – UML диаграмма классов

## 2.5 Выводы по разделу

В данной главе был обоснован выбор языка программирования, а также среда программирования для создаваемого программного средства. Кроме того, были описаны основные классы программного средства.

# 3 Конструирование игры

На данном этапе выполняется разработка графических компонентов игры, а также разработка самих локаций в игре.

## 3.1 Графическое конструирование

В нашей компьютерной игре должны присутствовать следующие графические элементы: змея, фон, яблоко. Все эти элементы можно нарисовать самостоятельно, используя библиотеку PyGame.

Все, их примеры можно увидеть на рисунках 3.1 – 3.3.



Рисунок 3.1 – Змейка



Рисунок 3.2 – Яблоко

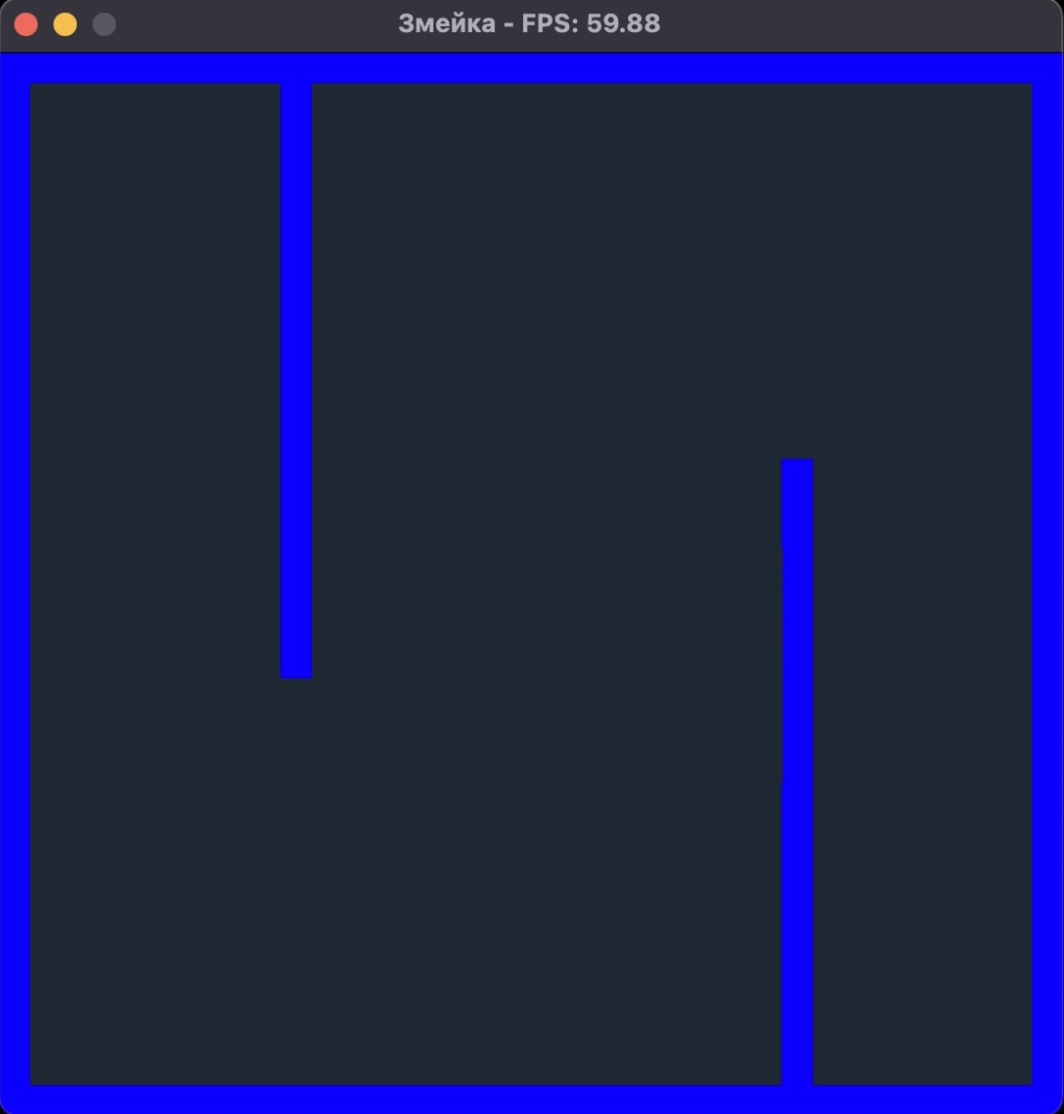


Рисунок 3.3 – Фон

## 3.2 Конструирование локаций

Игра будет состоять из одной локации – первого уровня. Уровень будет представлять собой полет в космосе с генерирующимися противниками, иллюзию движения создаст сдвигающийся сверху вниз, циклически повторяющийся фон.

Главной целью игрока будет продержаться как можно дольше, попутно набирая счет за победу над врагами.

## 3.3 Выводы по главе

Были созданы и описаны подходящие графические элементы игры и была разработана идея и логика локации.

# 4 Тестирование модулей программного средства

## 4.1 Пошаговое тестирование

При запуске игры пользователь попадает в игровое меню, где необходимо нажать любую кнопку для начала игры (рисунок 4.1).

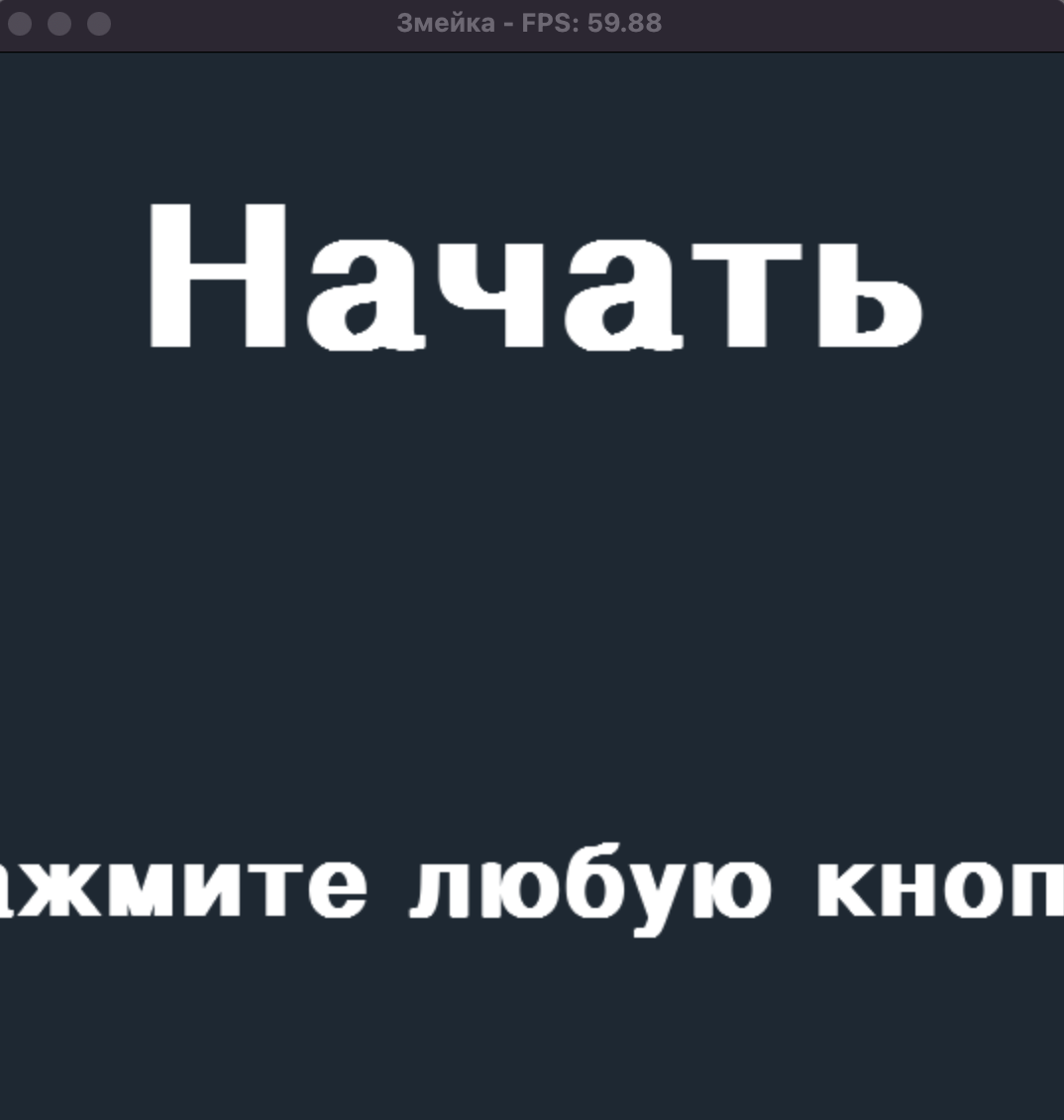


Рисунок 4.1 – Начальный экран

После небольшого ожидания должны появляться сцена для игры, преграды, а также сама змейка и яблоко (рисунок 4.2).

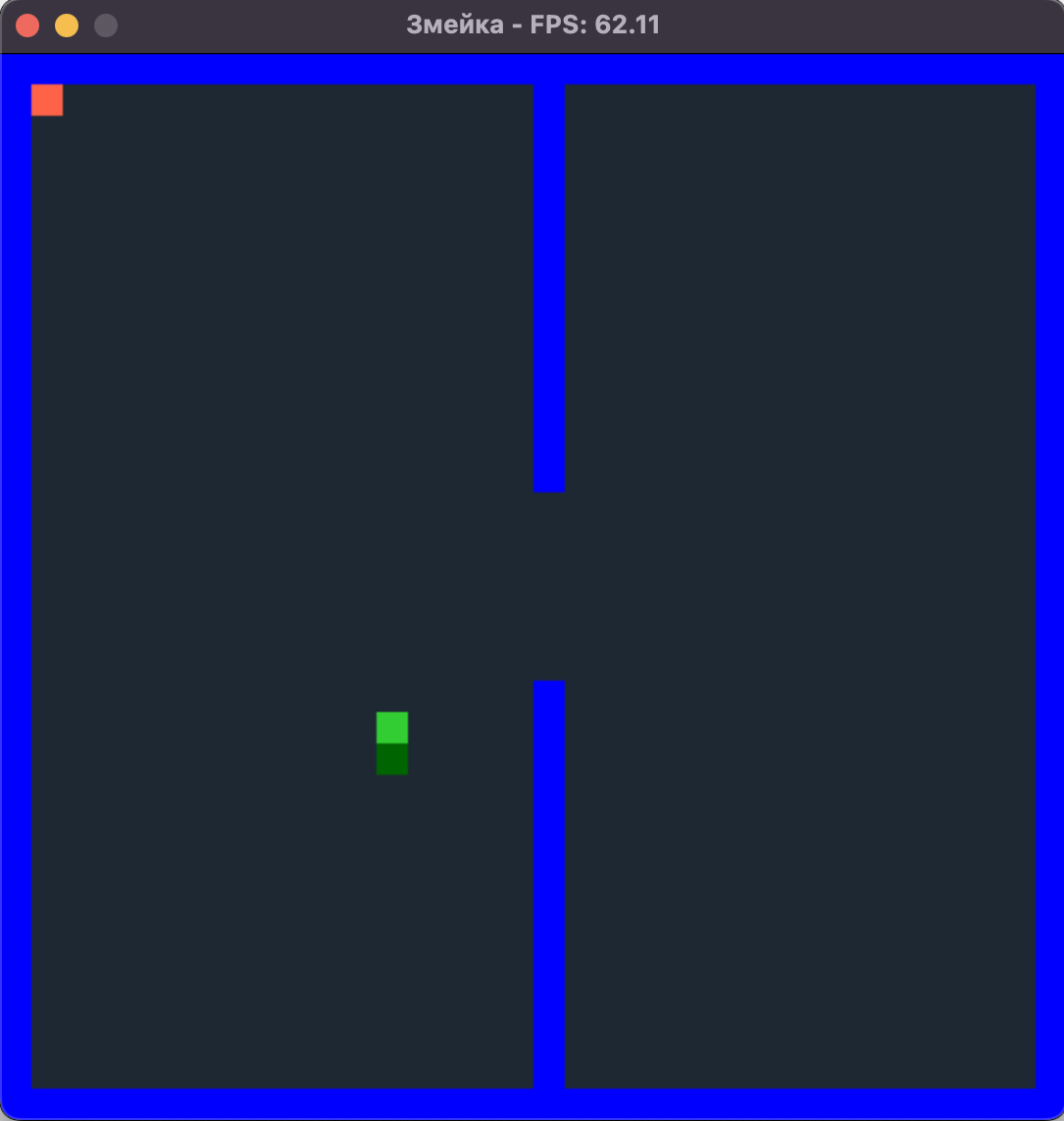


Рисунок 4.2 – Появление сцены и змеи

При поглощении яблока змея увеличивается на 1 блок и продолжает движение в этом же направлении (рисунок 4.3).

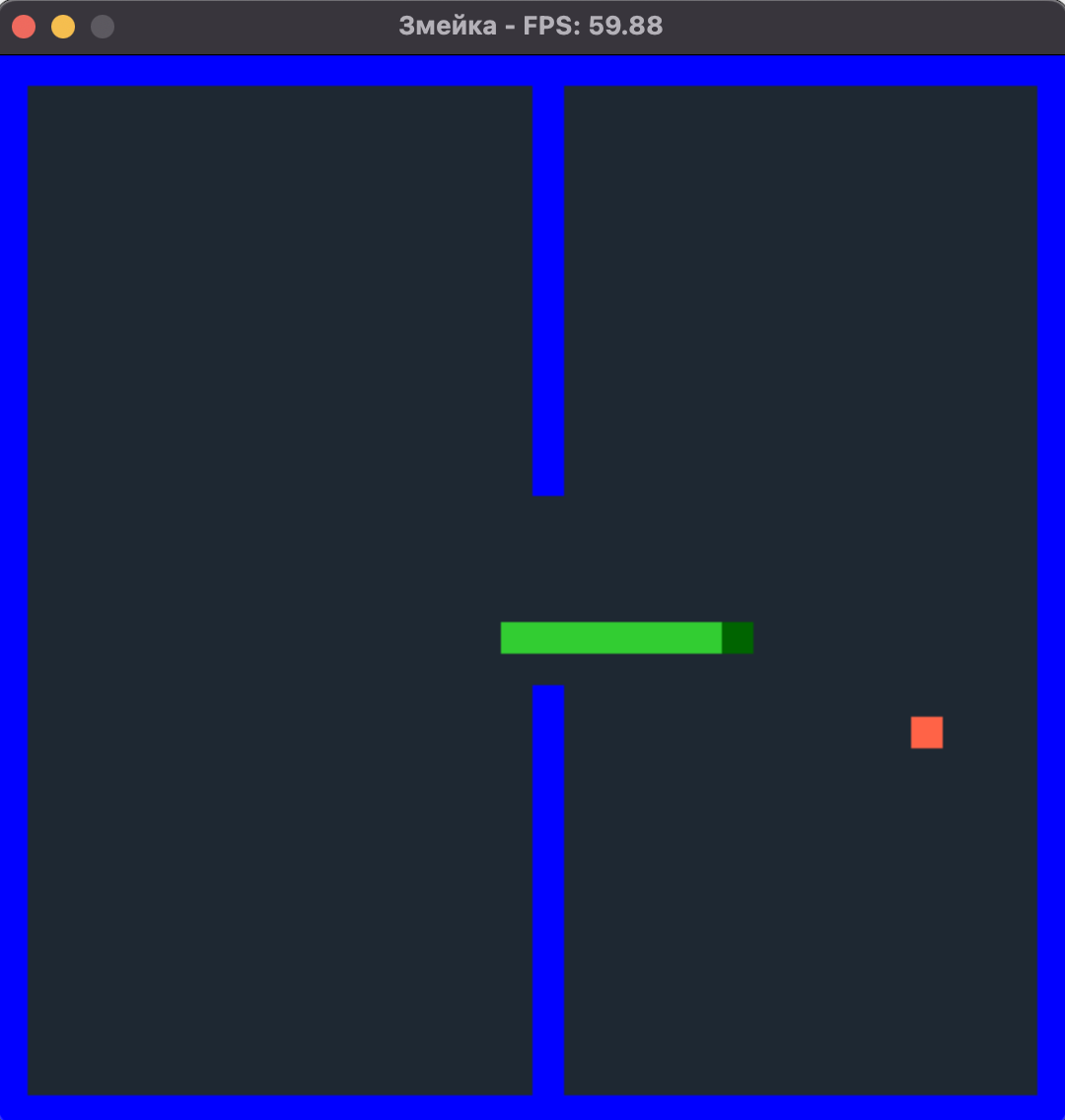


Рисунок 4.3 – Увеличение змеи

При попадании змеи в саму себя игрок проигрывает и может начать игру заново нажатием любой кнопки (рисунок 4.4).



Рисунок 4.4 – Поражение игрока

При попадании змеи в преграды или в стену игрок также проигрывает и может начать игру заново (рисунок 4.5).

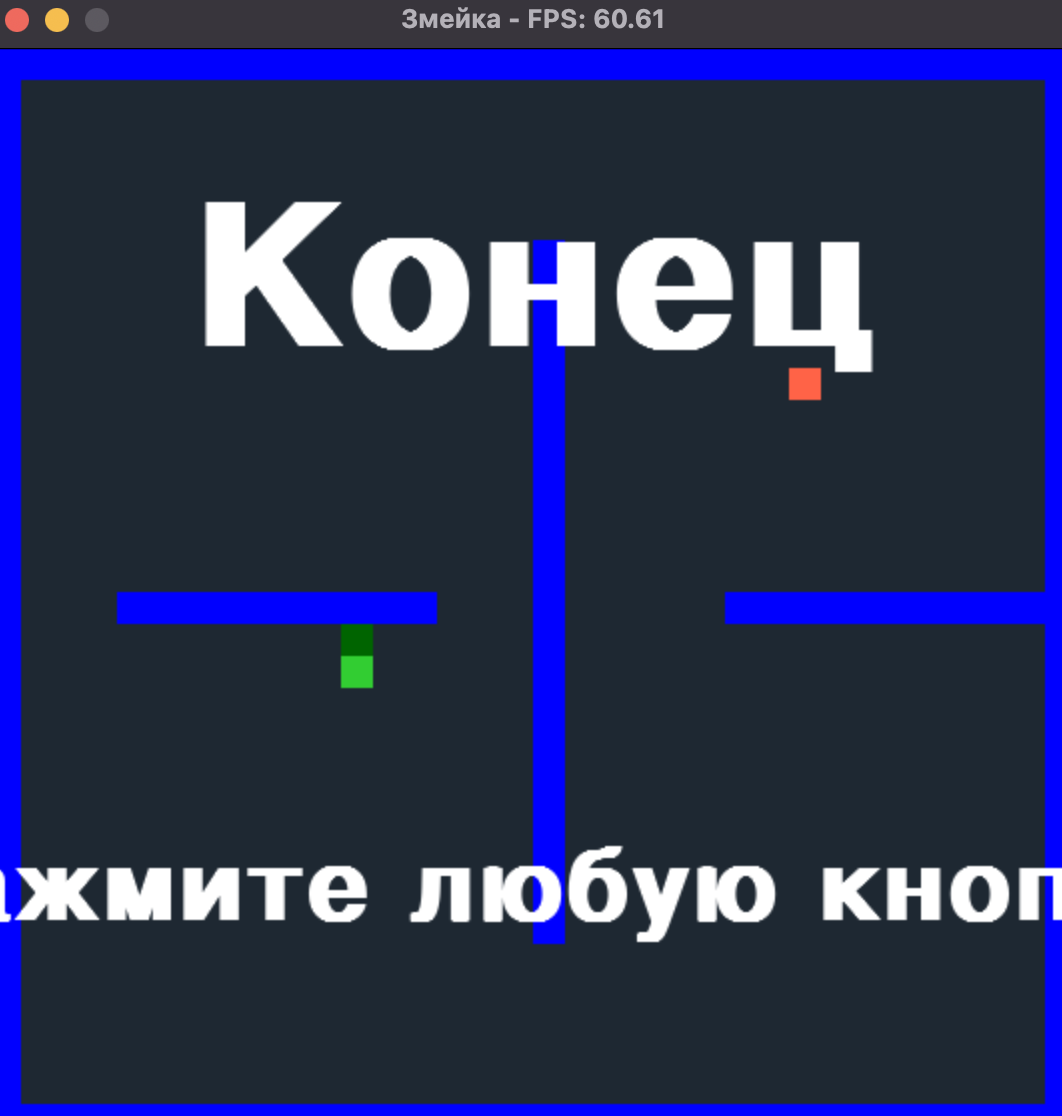


Рисунок 4.5 – Поражение игрока

## 4.2 Итоги тестирования

После успешного написания кода программы был начат процесс отладки разрабатываемого приложения. В ходе тестирования происходил запуск игры, проверялись корректные: передвижение змеи по сцене, генерация яблока на сцене, генерация преград на сцене, проверка механики изменения длины змеи, попадания в стенки и в саму себя.

В случае возникновения неполадок, ошибок и некорректной работы приложение дорабатывалось, а ошибки в коде исправлялись. В результате было получено полностью работоспособное и стабильное приложение, успешно выполняющее все возложенные на него задачи.

**Заключение**

В рамках данной работы была создана игра жанра «Аркада». В ходе разработки игры были изучены основные функции библиотеки PyGame, применены различные прицепы объектно-ориентированного программирования, получены навыки проектирования и реализации игрового процесса, разработаны основные функции для увлекательного и динамичного геймплея.

Перечень использованных информационных ресурсов

1. ru.wikipedia.org: [Сайт]. – URL: <https://ru.wikipedia.org/wiki/Компьютерная_игра> (дата обращения 10.12.2022)

2. Бейдер Ден: Чистый Python. Тонкости программирования для профи: [Книга]. (дата обращения: 10.12.2022).

3. Лутц Марк: Программирование на Python: [Книга]. (дата обращения: 10.12.2022).

4. Skillfactory [Сайт]. – URL: <https://blog.skillfactory.ru/glossary/python/> (дата обращения: 10.12.2022).

5. PythonWorld [Сайт]. – URL: https://pythonworld.ru/moduli/modul-collections.html (дата обращения: 10.12.2022).

6. ru.wikipedia.org: [Сайт]. URL: https://ru.wikipedia.org/wiki/Snake\_ (дата обращения: 02.12.2022).

7. Newart [Сайт]. – URL: <http://www.newart.ru/htm/myzavr/mz161.php> (дата обращения: 10.12.2022).

8. KvestInfo [Сайт]. – URL: https://www.kvestinfo.ru/articles/chto-takoe-arkada/ (дата обращения: 10.12.2022).

Приложение А Листинг программы

Исходный код программного средства

***Листинг А.1 – Программа «Snake» (для языка Python)***

import os

import sys

import random

import collections

import pygame as pg

try:

import Queue as queue

except ImportError:

import queue

CAPTION = "Змейка"

SCREEN\_SIZE = (544, 544)

PLAY\_RECT = pg.Rect(16, 16, 512, 512)

CELL = pg.Rect(0, 0, 16, 16)

BOARD\_SIZE = (PLAY\_RECT.w//CELL.w, PLAY\_RECT.h//CELL.h)

GROWTH\_PER\_APPLE = 3

COLORS = {"background" : (30, 40, 50), "walls" : pg.Color("blue"),

"snake" : pg.Color("limegreen"), "head" : pg.Color("darkgreen"),

"apple" : pg.Color("tomato")}

DIRECT\_DICT = {"left" : (-1, 0), "right" : ( 1, 0),

"up" : ( 0,-1), "down" : ( 0, 1)}

OPPOSITES = {"left" : "right", "right" : "left",

"up" : "down", "down" : "up"}

KEY\_MAPPING = {(pg.K\_LEFT, pg.K\_a) : "left", (pg.K\_RIGHT, pg.K\_d) : "right",

(pg.K\_UP, pg.K\_w) : "up", (pg.K\_DOWN, pg.K\_s) : "down"}

class Apple(object):

"""Яблоко длинна+1."""

def \_\_init\_\_(self, walls, snake):

self.position = self.respawn(snake.body\_set | walls)

self.walls = walls

self.color = COLORS["apple"]

def collide\_with(self, snake):

"""Респавн яблока на карте ."""

self.position = self.respawn(snake.body\_set | self.walls)

def respawn(self, obstacles):

"""Рандом спавн не в snake и не в walls."""

position = tuple(random.randrange(BOARD\_SIZE[i]) for i in (0,1))

while position in obstacles:

position = tuple(random.randrange(BOARD\_SIZE[i]) for i in (0,1))

return position

class Snake(object):

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

self.color = COLORS["snake"]

self.speed = 8 # Cells per second

self.direction = "up"

self.vector = DIRECT\_DICT[self.direction]

self.body = [(10, 25), (10, 24)]

self.body\_set = set(self.body)

self.growing = False

self.grow\_number = 0

self.timer = 0

self.dead = False

self.direction\_queue = queue.Queue(5)

def update(self, now):

"""+1 в head змеи."""

if not self.dead and now-self.timer >= 1000.0/self.speed:

self.timer = now

self.change\_direction()

next\_cell = [self.body[-1][i]+self.vector[i] for i in (0,1)]

self.body.append(tuple(next\_cell))

if not self.growing:

del self.body[0]

else:

self.grow()

self.body\_set = set(self.body)

def change\_direction(self):

"""

Проверить очередь на наличие нового направления

"""

try:

new = self.direction\_queue.get(block=False)

except queue.Empty:

new = self.direction

if new not in (self.direction, OPPOSITES[self.direction]):

self.vector = DIRECT\_DICT[new]

self.direction = new

def grow(self):

"""Увеличть grow\_number и сбросить по готовке."""

self.grow\_number += 1

if self.grow\_number == GROWTH\_PER\_APPLE:

self.grow\_number = 0

self.growing = False

def check\_collisions(self, apple, walls):

"""логика вхождения в wall,apple,в себя."""

if self.body[-1] == apple.position:

apple.collide\_with(self)

self.growing = True

elif self.body[-1] in walls:

self.dead = True

elif any(val > 1 for val in collections.Counter(self.body).values()):

self.dead = True

def get\_key\_press(self, key):

"""

Добавить направления в очередь направлений, если нажата клавиша KEY\_MAPPING.

"""

for keys in KEY\_MAPPING:

if key in keys:

try:

self.direction\_queue.put(KEY\_MAPPING[keys], block=False)

break

except queue.Full:

pass

def draw(self, surface, offset=(0,0)):

"""Рисовка туловища, затем головы."""

for cell in self.body:

Makedraw.draw\_cell(surface, cell, self.color, offset)

Makedraw.draw\_cell(surface, self.body[-1], COLORS["head"], offset)

class \_Scene(object):

"""Сцена."""

def \_\_init\_\_(self, next\_state=None):

self.next = next\_state

self.done = False

self.start\_time = None

self.screen\_copy = None

def startup(self, now):

"""старт."""

self.start\_time = now

self.screen\_copy = pg.display.get\_surface().copy()

def reset(self):

"""Рестарт игры."""

self.done = False

self.start\_time = None

self.screen\_copy = None

def get\_event(self, event):

"""перегруз."""

pass

def update(self, now):

"""запуск."""

if not self.start\_time:

self.startup(now)

class AnyKey(\_Scene):

"""Сцены: старт,смерть."""

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

\_Scene.\_\_init\_\_(self, "GAME")

self.blink\_timer = 0.0

self.blink = False

self.make\_text(title)

self.reset()

def make\_text(self, title):

"""Текст."""

self.main = FONTS["BIG"].render(title, True, pg.Color("white"))

self.main\_rect = self.main.get\_rect(centerx=PLAY\_RECT.centerx,

centery=PLAY\_RECT.centery-150)

text = "Нажмите любую кнопку"

self.ne\_key = FONTS["SMALL"].render(text, True, pg.Color("white"))

self.ne\_key\_rect = self.ne\_key.get\_rect(centerx=PLAY\_RECT.centerx,

centery=PLAY\_RECT.centery+150)

def draw(self, surface):

"""мигание."""

surface.blit(self.screen\_copy, (0,0))

surface.blit(self.main, self.main\_rect)

if self.blink:

surface.blit(self.ne\_key, self.ne\_key\_rect)

def update(self, now):

"""обновить мигалку."""

\_Scene.update(self, now)

if now-self.blink\_timer > 1000.0/5:

self.blink = not self.blink

self.blink\_timer = now

def get\_event(self, event):

"""старт после нажатия кнопки."""

if event.type == pg.KEYDOWN:

self.done = True

class Game(\_Scene):

"""сцена игры."""

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

\_Scene.\_\_init\_\_(self, "DEAD")

self.reset()

def reset(self):

"""Подготовка к следующей попытке."""

\_Scene.reset(self)

self.snake = Snake()

self.walls = self.make\_walls()

self.apple = Apple(self.walls, self.snake)

def make\_walls(self):

"""Сделать границы и загрузить случайный уровень."""

walls = set()

for i in range(-1, BOARD\_SIZE[0]+1):

walls.add((i, -1))

walls.add((i, BOARD\_SIZE[1]))

for j in range(-1, BOARD\_SIZE[1]+1):

walls.add((-1, j))

walls.add((BOARD\_SIZE[0], j))

walls |= random.choice(LEVELS)

return walls

def get\_event(self, event):

"""Нажать любую клавишу."""

if event.type == pg.KEYDOWN:

self.snake.get\_key\_press(event.key)

def update(self, now):

"""Смерть."""

\_Scene.update(self, now)

self.snake.update(now)

self.snake.check\_collisions(self.apple, self.walls)

if self.snake.dead:

self.done = True

def draw(self, surface):

"""Прорисовка элементов."""

surface.fill(COLORS["background"])

Makedraw.draw\_cell(surface, self.apple.position,

self.apple.color, PLAY\_RECT.topleft)

for wall in self.walls:

Makedraw.draw\_cell(surface, wall, COLORS["walls"], PLAY\_RECT.topleft)

self.snake.draw(surface, offset=PLAY\_RECT.topleft)

class Control(object):

"""Содержит основной цикл, цикл событий и логику переключения сцен."""

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

"""Стандартная настройка и создание начальной сцены."""

self.screen = pg.display.get\_surface()

self.clock = pg.time.Clock()

self.fps = 60.0

self.done = False

self.state\_dict = {"START" : AnyKey("Начать"),

"GAME" : Game(),

"DEAD" : AnyKey("Конец")}

self.state = self.state\_dict["START"]

def event\_loop(self):

"""Обработка выхода."""

for event in pg.event.get():

if event.type == pg.QUIT:

self.done = True

self.state.get\_event(event)

def update(self):

"""Обновить текущую сцену и переключиться при необходимости."""

now = pg.time.get\_ticks()

self.state.update(now)

if self.state.done:

self.state.reset()

self.state = self.state\_dict[self.state.next]

def draw(self):

"""Нарисуйте текущую сцену, если она готова."""

if self.state.start\_time:

self.state.draw(self.screen)

def display\_fps(self):

"""Показать fps, для проверки."""

caption = "{} - FPS: {:.2f}".format(CAPTION, self.clock.get\_fps())

pg.display.set\_caption(caption)

def main\_loop(self):

"""Run-a-round."""

self.screen.fill(COLORS["background"])

while not self.done:

self.event\_loop()

self.update()

self.draw()

pg.display.update()

self.clock.tick(self.fps)

self.display\_fps()

class Makedraw():

def draw\_cell(surface, cell, color, offset=(0,0)):

"""surface."""

pos = [cell[i]\*CELL.size[i] for i in (0,1)]

rect = pg.Rect(pos, CELL.size)

rect.move\_ip(\*offset)

surface.fill(color, rect)

def make\_levels():

w, h = BOARD\_SIZE

r = range

levels = [

({(w//2,i) for i in r(h//2-3)}|{(w//2,i) for i in r(h//2+3,h)}),

({(w//4,i) for i in r(3\*h//5)}|{(3\*w//4,i) for i in r(2\*h//5,h)}),

({(w//2,i) for i in r(5,h-5)}|{(i,h//2) for i in r(3,w//2-3)}|

{(i+w//2+3, h//2) for i in r(3,w//2-3)})]

return levels

class Start():

def main(self):

"""

Подготовка pygame на экран.

"""

global FONTS,LEVELS

os.environ["SDL\_VIDEO\_CENTERED"] = "True"

pg.init()

pg.display.set\_caption(CAPTION)

pg.display.set\_mode(SCREEN\_SIZE)

FONTS = {"BIG" : pg.font.SysFont("helvetica", 100, True),

"SMALL" : pg.font.SysFont("helvetica", 50, True)}

LEVELS = Makedraw.make\_levels()

Control().main\_loop()

pg.quit()

sys.exit()

abc=Start()

abc.main()