Управление образованием города Алматы

Международный колледж бизнеса и коммуникаций

КУРСОВОЙ ПРОЕКТ

Игра “Змейка” на языке программирования Python

06130100 - Программное обеспечение (по видам)
4S06130105 – Техник информационных систем

Выполнил(а):
Б.Курбанов

____ _____
2023года

Руководитель: А. Серикова

____ _____
2023года

Защита курсового проекта

___ _____
2023 года

Полученная оценка

_____

Алматы 2023

Содержание

Аннотация

Курбанов Б.Р.«Игра змейка на языке программирования Python».Курсовой проект. МКБиК 2023. Курсовой проект состоит из введения и трех глав, заключения, списка используемой литературы и рисунков; содержит 10 рисунков. Объем курсового проекта 27 страниц машинописного текста. Список использованной литературы включает 14 наименований. Ключевые слова: Синтаксис, программа, команда.

В данном курсовом проекте исследуется популярная игра «Змейка», реализованная на языке программирования Python. Курсовая начинается с введения в игру и ее правил, после чего следует обзор языка программирования Python и его графических и возможности программирования игр. Также обозревается полное описание проекта.

Курсовой проект также включает раздел по исправления ошибок игры, включая использование различных инструментов и методов отладки Python. Также были добавлены дополнительные графические элементы: Счет, надпись об окончании игры и задний фон.

Annotation

Kurbanov B.R. "The snake game in the Python programming language".Course project. ICBiC 2023. The course project consists of an introduction and three chapters, a conclusion, a list of used literature and drawings; contains 10 drawings. The volume of the course project is 25 pages of typewritten text. The list of references includes 6 titles. Keywords: Syntax, program, command.

This course project explores the popular game "Snake", implemented in the Python programming language. The course begins with an introduction to the game and its rules, followed by an overview of the Python programming language and its graphical and game programming capabilities. A full description of the project is also reviewed.

The course project also includes a section on fixing game errors, including the use of various Python debugging tools and methods. Additional graphic elements were also added: the score, the inscription about the end of the game and the background.

Аннотация

Құрбанов б. р. "Python бағдарламалау тіліндегі жылан ойыны".Курстық жоба. Icdc 2023. Курстық жоба кіріспеден және үш тараудан, қорытындыдан, пайдаланылған әдебиеттер мен суреттердің тізімінен тұрады; 10 суреттен тұрады. Курстық жобаның көлемі-25 бет терілген мәтін. Пайдаланылған әдебиеттер тізіміне 6 атау кіреді. Кілт сөздер: Синтаксис, бағдарлама, команда.

Бұл курстық жоба Python бағдарламалау тілінде жүзеге асырылатын танымал "жылан" ойынын зерттейді. Курстық жұмыс ойынға және оның ережелеріне кіріспеден басталады, содан кейін Python бағдарламалау тілі мен оның графикалық және ойын бағдарламалау мүмкіндіктеріне шолу жасалады. Жобаның толық сипаттамасы да көрсетілген.

Курстық жоба сонымен қатар ойын қателерін түзету бөлімін, соның ішінде Python-различных жөндеудің әртүрлі құралдары мен әдістерін қолдануды қамтиды. Қосымша графикалық элементтер де қосылды: ұпай, ойынның аяқталуы туралы жазба және фон.

Введение

Змейка - игра для телефонов с которого знаком и играл практически каждый человек на земле. В игре участвует один игрок, который контролирует движение змейки по горизонтали и вертикали, а точнее по математической координатной прямой по осям абцисс и ординат. Победителем будет считаться игрок который дошёл до максимального размера змейки (заполнил весь экран), победить в этой игре представляется достаточно сложно, для победы нужна хорошая реакция, внимательность и ловкость игрока, а также опыт игры.

Змейка является отличным источником опыта для начинающих программистов, код строится не так сложно и позволяет ознакомиться со всем синтаксисом Python и билиотеки Pygame.

В курсовом проекте я расскажу как сделать игру “Змейка” на Python. Что для этого потребуется, какие методы и этапы разработки будут использованы а также сложности которые могут встретиться на пути. Также добавим приятный графический интерфейс с возможностью следить за своим прогрессом и проигрышем.

В конце проекта мы сможем сделать выводы и чему я смог научиться по ходу изучения языка и библиотек для игр Python и Pygame. С этим опытом я вполне могу продолжить практиковаться в написании игр и выйти на более высокий уровень в программировании на Python.

Основная часть

1.1 Синтаксис языка

В этом проекте будут использованы: Функции, условные операторы, переменные, циклы итерирования, словари и списки, библиотеки связанные с созданием движка игр, вызывания случайных, для настройки управлении пользователем программы.

Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции **def**.

Определим простейшую функцию:

```
def add(x, y):

    return x + y
```

Инструкция **return** говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму x и y.

Теперь мы ее можем вызвать:

```
>>>
>>> add(1, 10)
11
>>> add('abc', 'def')

'abcdf'e
```

Функция может быть любой сложности и возвращать любые объекты (списки, кортежи, и даже функции!):

```
>>>
>>> def newfunc(n):
...     def myfunc(x):
...         return x + n
...     return myfunc
...
>>> new = newfunc(100)  # new - это функция
>>> new(200)
```

300

Функция может и не заканчиваться инструкцией return, при этом функция вернет значение None:

```
>>>
>>> def func():
...     pass
...
>>> print(func())

None
```

что такое условные операторы?

Они отвечают за изменение поведения программы в зависимости от входных параметров, определенных в проверке. Проще говоря: если будет число 1, то программа запустит скрипт one, а если число 2 – скрипт two. Внутри условных операторов могут быть другие такие же условия для уточнения полученных данных. В рамках одного оператора можно сразу проверить пару условий. Для того, чтобы проверить несколько условий нужно их разделить элементом and (логическое и).
Пример создания условия:

```
a = 2
if a != 0 and a != 1:
        print ("Проверка сработала")
```

На экране будет показана запись лишь в том случае, когда переменная «a» не будет равна значению 0 и значению 1. То есть обе проверки в операторе должны выдать результат – true.
Есть возможность произвести проверку с помощью or - логическое или. При использовании данного оператора достаточным поводом для запуска сообщения «Заработало» станет соответствие хотя бы одного из условий.
Пример:

```
a = 1.1
if a != 1.1 or a > 0:
        print ("Проверка сработала")
```

Исходный код
Условный оператор «if-else»

```
user_data = int(input("Введите число: "))
```

```
isHappy = True

if isHappy or user_data == 6:
    print("User is happy")
elif user_data == 5:
    print("Number is 5")
elif user_data == 7:
    print("Number is 7")
else:
    print("User is unhappy")
```

Оператор else срабатывает когда не один из предыдущих аргументов не срабатывает.

Переменные

Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python.

Например, создадим переменную:

```
name = "Case"
```

Здесь определена переменная name, которая хранит строку "Case".

В пайтоне применяется два типа наименования переменных: camel case и underscore notation.

Camel case подразумевает, что каждое новое подслово в наименовании переменной начинается с большой буквы. Например:

```
userName = "Case"
```

Underscore notation подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания. Например:

user_name = "Case"
И также надо учитывать регистрозависимость, поэтому переменные name и Name будут представлять разные объекты.

```
# две разные переменные
name = "Case"
Name = "Case"
```

Определив переменную, мы можем использовать в программе. Например, попытаться вывести ее содержимое на консоль с помощью встроенной функции print:

```
name = "Case"  # определение переменной name
print(name)   # вывод значения переменной name на консоль.
```

Отличительной особенностью переменной является то, что мы можем менять ее значение в течение работы программы:

```
name = "Tom"  # переменной name равна "Tom"
print(name)   # выводит: Tom
name = "Bob"  # меняем значение на "Bob"
print(name)   # выводит: Bob
```

Циклы позволяют выполнять некоторое действие в зависимости от соблюдения некоторого условия. В языке Python есть следующие типы циклов:

- **while**
- **for**

Цикл while
Цикл while проверяет истинность некоторого условия, и если условие истинно, то выполняются инструкции цикла. Он имеет следующее формальное определение:

```
while условное выражение:
```

инструкции

После ключевого слова while указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала ключевого слова while.

```
number = 1

while number < 5:
    print(f"number = {number}")
    number += 1
print("Работа программы завершена")
```

В данном случае цикл while будет выполняться, пока переменная number меньше 5.

Сам блок цикла состоит из двух инструкций:

```
print(f"number = {number}")
number += 1
```

Обратите внимание, что они имеют отступы от начала оператора while - в данном случае от начала строки. Благодаря этому Python может определить, что они принадлежат циклу. В самом цикле сначала выводится значение переменной number, а потом ей присваивается новое значение. .

Также обратите внимание, что последняя инструкция print("Работа программы завершена") не имеет отступов от начала строки, поэтому она не входит в цикл while.

Весь процесс цикла можно представить следующим образом:

Сначала проверяется значение переменной number - меньше ли оно 5. И поскольку вначале переменная равна 1, то это условие возвращает True, и поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 1. И далее значение переменной number увеличивается на единицу - теперь она равна 2. Однократное

выполнение блока инструкций цикла называется итерацией. То есть таким образом, в цикле выполняется первая итерация.

Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 2, поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 2. И далее значение переменной number опять увеличивается на единицу - теперь она равна 3. Таким образом, выполняется вторая итерация.

Опять проверяется условие number < 5. Оно по прежнему равно True, так как number = 3, поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 3. И далее значение переменной number опять увеличивается на единицу - теперь она равна 4. То есть выполняется третья итерация.

Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 4, поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 4. И далее значение переменной number опять увеличивается на единицу - теперь она равна 5. То есть выполняется четвертая итерация.

И вновь проверяется условие number < 5. Но теперь оно равно False, так как number = 5, поэтому выполняются выход из цикла. Все цикл - завершился. Дальше уже выполняются действия, которые определены после цикла. Таким образом, данный цикл произведет четыре прохода или четыре итерации.

Словари

В языке программирования Python словари (тип dict) представляют собой еще одну разновидность структур данных наряду со списками и кортежами. Словарь - это изменяемый (как список) неупорядоченный (в отличие от строк, списков и кортежей) набор элементов "ключ:значение".

"Неупорядоченный" – значит, что последовательность расположения пар не важна, вследствие чего обращение к элементам по индексам невозможно.

В других языках структуры, схожие со словарями, называются по-другому. Например, в Java подобный тип данных называется отображением.

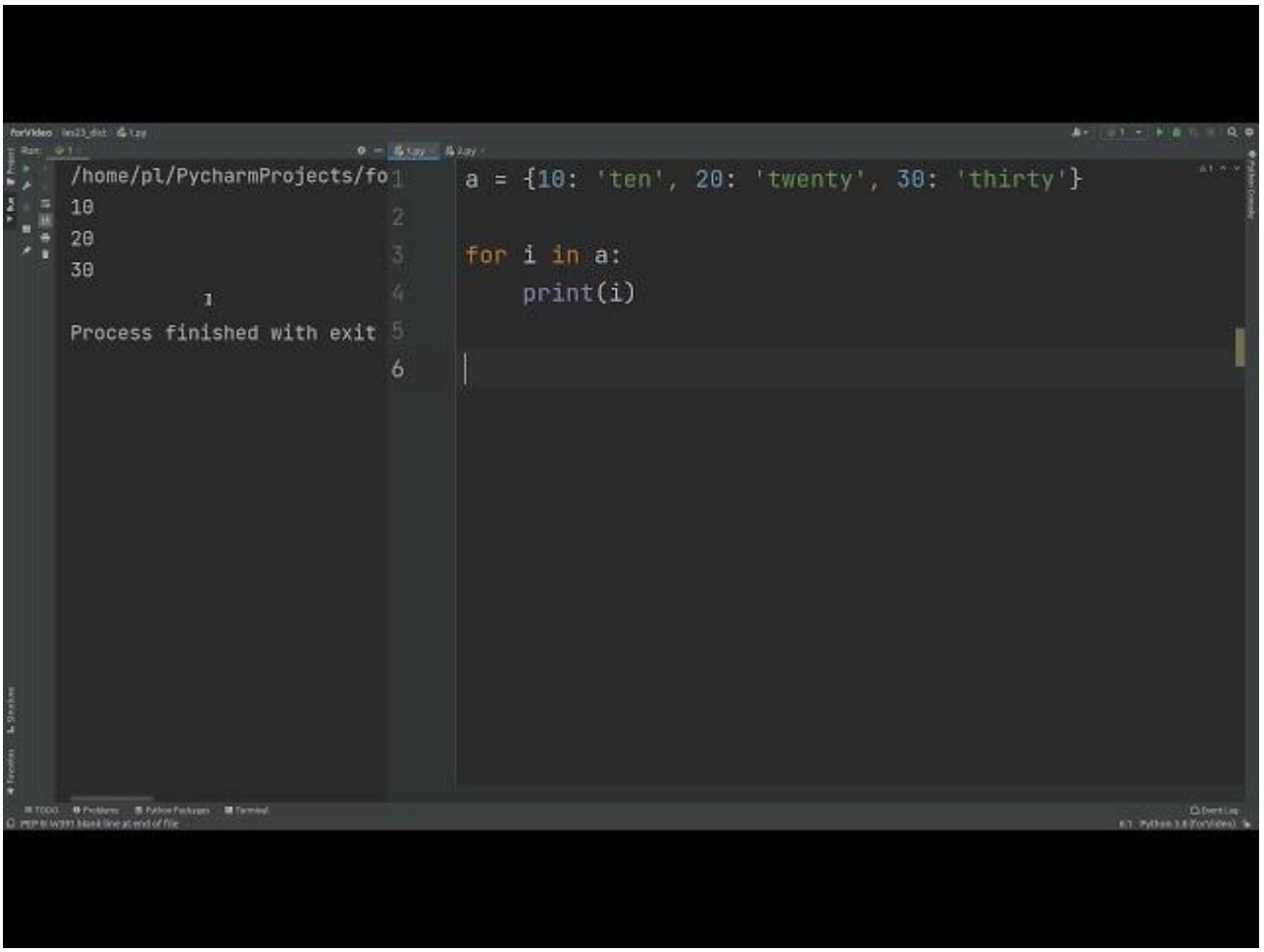

Рисунок №1 - пример словаря

Чтобы представление о словаре стало более понятным, проведем аналогию с обычным словарем, например, англо-русским. На каждое английское слово в таком словаре есть русское слово-перевод: cat – кошка, dog – собака, table – стол и т. д. Если англо-русский словарь описать с помощью Python, то английские слова можно сделать ключами, а русские – их значениями:
', 'bird': 'птица'} {'cat': 'кошка', 'dog': 'собака
Обратите внимание, что для определения словаря используются фигурные скобки. Синтаксис словаря на Питоне описывается такой схемой:

В словаре доступ к значениям осуществляется не по индексам, а по ключам, которые заключаются в квадратные скобки (по аналогии с индексами списков):
>>> a['cat']
'кошка'
>>> a['bird']

'птица'
Словари, как и списки, являются изменяемым типом данных: позволительно изменять, добавлять и удалять элементы (пары "ключ:значение"). Изначально словарь можно создать пустым (например, d = {}) и потом заполнить его элементами. Добавление и изменение имеет одинаковый синтаксис: словарь[ключ] = значение. Ключ может быть как уже существующим (тогда происходит изменение значения), так и новым (происходит добавление элемента словаря). Удаление элемента осуществляется с помощью встроенной оператора del языка Python.

```
>>> a = {}
>>> a[1] = 2.34
>>> a[2] = 4.88
>>> a
{1: 2.34, 2: 4.88}
>>> a[1] = 3.01
>>> a
{1: 3.01, 2: 4.88}
>>> del a[2]
>>> a
{1: 3.01}
```

В словаре не может быть двух элементов с одинаковыми ключами. Однако могут быть одинаковые значения у разных ключей.
Ключом может быть любой неизменяемый тип данных. Значением – любой тип данных. Значения словарей вполне могут быть структурами, например, другими словарями или списками.

```
>>> nums = {'one': (1, 'I'), 'two': (2, 'II')}
>>> person1 = {'name': 'Tom', 'data': [4, 2.5]}
>>> person1['data'][0]
```

Библиотеки Python — это файлы с шаблонами кода. Их создали для того, чтобы люди не набирали каждый раз заново один и тот же код: достаточно открыть файл, вставить свои данные и получить результат. Рассказываем, какие библиотеки часто используют разработчики на Python

в моём проекте будут использоваться следующие библиотеки:

Pygame (рус. Пайгейм) — набор модулей (библиотек) языка программирования Python, предназначенный для написания компьютерных игр и мультимедиа-приложений. Pygame базируется на мультимедийной библиотеке SDL. Изначально Pygame был написан Питон Шинером (Pete Shinners). Начиная примерно с 2004/2005 года поддерживается и развивается сообществом свободного программного обеспечения.
Одна из библиотек предоставляющих доступ к API SDL (существуют и другие). В то же время дает возможность написания более высокоуровневого кода.
Pygame-приложения могут работать под Android на телефонах и планшетах с использованием подмножества Pygame для Android (pgs4a). На этой платформе поддерживаются звук, вибрация, клавиатура, акселерометр.
Модули Pygame:
pygame.event.get - получает события из очереди

pygame.event.poll - получить одно событие из очереди

pygame.event.wait - ждёт одиночного события из очереди

pygame.event.post - поместить новое событие в очередь

pygame.event.Event - создать новый объект события

pygame.image - Загружает и сохраняет изображение

pygame.event - Управление внешними событиями

pygame.cdrom - Доступ к CD-приводам и управление ими

pygame.cursors - Загружает изображения курсора

pygame.display - Доступ к дисплею

Модуль - в языке Python представляет отдельный файл с кодом, который можно повторно использовать в других программах.

Для создания модуля необходимо создать собственно файл с расширением *.py, который будет представлять модуль. Название файла будет представлять название модуля. Затем в этом файле надо определить одну или несколько функций.

Допустим, основной файл программы называется main.py. И мы хотим подключить к нему внешние модули.

Для этого сначала определим новый модуль: создадим в той же папке, где находится main.py, новый файл, который назовем message.py. Если используется PyCharm или другая IDE, то оба файла просто помещаются в один проект.

Соответственно модуль будет называться message. Определим в нем следующий код:

```
1
2
3
4
5
hello = "Hello all"


def print_message(text):
    print(f"Message: {text}")
```

Здесь определена переменная hello и функция print_message, которая в качестве параметра получает некоторый текст и выводит его на консоль.

В основном файле программы - main.py используем данный модуль:

```
1
2
3
4
5
6
import message      # подключаем модуль message
```

```
# выводим значение переменной hello
print(message.hello)        # Hello all
# обращаемся к функии print_message
message.print_message("Hello work")  # Message: Hello work
```

Для использования модуля его надо импортировать с помощью оператора import, после которого указывается имя модуля: import message.

Чтобы обращаться к функциональности модуля, нам нужно получить его пространство имен. По умолчанию оно будет совпадать с именем модуля, то есть в нашем случае также будет называться message.

Получив пространство имен модуля, мы сможем обратиться к его функциям по схеме

пространство_имен.функция
Например, обращение к функции print_message() из модуля message:

```
1
message.print_message("Hello work")
```

И после этого мы можем запустить главный скрипт main.py, и он задействует модуль message.py. В частности, консольный вывод будет следующим:

```
Hello all
Message: Hello work
```

Библиотека **random** управляет генерацией случайных чисел. Его основные функции:

random(): генерирует случайное число от 0.0 до 1.0

randint(): возвращает случайное число из определенного диапазона

randrange(): возвращает случайное число из определенного набора чисел

shuffle(): перемешивает список

choice(): возвращает случайный элемент списка
random.seed([X], version=2) - инициализация генератора случайных чисел. Если X не указан, используется системное время.

random.getstate() - внутреннее состояние генератора.

random.setstate(state) - восстанавливает внутреннее состояние генератора. Параметр state должен быть получен функцией getstate().

random.getrandbits(N) - возвращает N случайных бит.

random.randrange(start, stop, step) - возвращает случайно выбранное число из последовательности.

random.randint(A, B) - случайное целое число N, $A \leq N \leq B$.

random.choice(sequence) - случайный элемент непустой последовательности.

random.shuffle(sequence, [rand]) - перемешивает последовательность (изменяется сама последовательность). Поэтому функция не работает для неизменяемых объектов.

random.sample(population, k) - список длиной k из последовательности population.

random.random() - случайное число от 0 до 1.

random.uniform(A, B) - случайное число с плавающей точкой, $A \leq N \leq B$ (или $B \leq N \leq A$).

random.triangular(low, high, mode) - случайное число с плавающей точкой, $low \leq N \leq high$. Mode - распределение.

random.betavariate(alpha, beta) - бета-распределение. alpha>0, beta>0. Возвращает от 0 до 1.

Функция random() возвращает случайное число с плавающей точкой в промежутке от 0.0 до 1.0. Если же нам необходимо число из большего диапазона, скажем от 0 до 100, то мы можем соответственно умножить результат функции random на 100.

Библиотека keyboard - Модуль клавиатуры представляет собой легкую и простую библиотеку, используемую для моделирования нажатий клавиш и простой автоматизации в Python.

В этом модуле есть много функций, которые можно использовать для имитации действий клавиатуры.

- keyboard.write(message, [delay])- пишет сообщение с задержкой или без нее.
- keyboard.wait(key) - блокирует программу до тех пор, пока не будет нажата клавиша. Ключ передается в виде строки ("пробел", "esc" и т.д.)
- keyboard.press(key)- нажимает клавишу и удерживается до вызова функции release(key)
- keyboard.release(key)- выпускает ключ.
- keyboard.send(key)- нажимает и отпускает клавишу.
- keyboard.add_hotkey(hotkey, function)- создает hotkey, которая при нажатии выполняет function.
- keyboard.record(key)- записывает активность клавиатуры до нажатия key.
- keyboard.play(recorded_events, [speed_factor]) - воспроизводит события, записанные with keyboard.record(key) функция, с дополнительным speed_factor.

Разработка игры

Подготовка

Для разработки нам понадобиться установленный на ПК язык программирование Python версии 3.11.3 а также интерпретатор или IDE, я буду использовать IDE. IDE - интегрированная среда разработки.
Для разработки будет использоваться PyCharm.

PyCharm — это кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains на основе IntelliJ IDEA на языках Java и Python. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик. Пользователи могут сами писать свои плагины, тем самым расширять возможности PyCharm. Некоторые плагины из других JetBrains IDE могут работать с PyCharm. Существует более тысячи плагинов, совместимых с PyCharm.

Создаём новый проект в PyCharm, IDE автоматически создает виртуальное окружение, это очень важно при разработке чтобы другие библиотеки не конфликтовали. В виртуальное окружение входит терминал, вывод и среда написания самого кода, также основные функции Python.
Далее открываем терминал и устанавливаем необходимые библиотеки для разработки а именно - PyGame, Keyboard, random.

Кодирование

*Кодирование начинается с импорта библиотека PyGame.
“Import Pygame” и генератора случайных чисел “Import random from randrange”

*Задаем разрешение нашего окна
Двумя переменными: RES = 800 SIZE = 60

*Змейка и яблоки будут появляться случайно в любой области экрана при помощи библиотеки randrange, также для регулировки скорости змейки задаём FPS = 5, начальная длина змейки length = 1, так же змейка будет двигаться по координатам X и Y создаём список shake = [(x,y)] и dy dx нужны корректной работы движения змейки. (скриншот №1)

*Посредством библиотеки PyGame мы инициализируем его и создаём рабочее окно. Для регулирования скорости змейки будет использован метод Clock. После создаем основной цикл While.

*В начале цикла задаем цвет фона нашего окна со змейкой, далее начинаем отрисовку самой змейки, отрисовываться она будет в виде зеленых квадратов, используем списковые включения.

*Отрисовка яблока происходит аналогично змейки за исключением координат, их будем передавать в построение квадрата.

```
import pygame
from random import randrange

RES = 800
SIZE = 80

x, y = randrange(0, RES, SIZE), randrange(0, RES, SIZE)
apple = randrange(0, RES, SIZE), randrange(0, RES, SIZE)
length = 1
snake = [(x, y)]
dx, dy = 0, 0
score = 0
fps = 5

pygame.init()
sc = pygame.display.set_mode([RES, RES])
clock = pygame.time.Clock()

while True:
    sc.fill(pygame.Color('black'))
    # отрисовка
    [(pygame.draw.rect(sc, pygame.Color('green'), (i, j, SIZE - 2, SIZE - 2))) for i, j in snake]
    pygame.draw.rect(sc, pygame.Color('red'), (*apple, SIZE, SIZE))
```

Рисунок №2 - основные переменные и цикл

*далее мы добавляем проверку окна игры для корректного отображения при помощи функции pygame.flip.display () и задаём задержку FPS

*Включаем проверку событий на закрытие приложения, это нужно чтобы не было ошибок и игра корректно завершалась. За это будет отвечать цикл for event с включенной в него функцией pygame.event.get(), после мы указываем оператор if который и будет проверять закрытие игры, после совершение данного цикла игра будет закрыта без ошибок.

*Начинаем прописывать движение нашей змейки, длина движения будет равна размеру головы нашей змейки то есть 1 клетки, для совершения движение мы делаем 2 переменные с такими значениями x += dx * SIZE y +=dy * SIZE

*Продолжая программировать управления змейки мы задаем клавиши для управления змейкой при помощи оператора if она будет смещаться по координатам x и y

*Также каждый шаг змейки мы будем добавлять в список координат пишем функцию shake.append((x , y))

*на этом этапе разработки змейка будет бесконечной, для правильной отображения длины мы будем ее срезать по уровню переменной которая отвечает за её длину shake = shake[-lenght:]

```
    # Движение змейки
    x += dx * SIZE
    y += dy * SIZE
    snake.append((x, y))
    snake = snake[-length:]
    # Скушать яблочко
    if snake[-1] == apple:
        apple = randrange(0, RES, SIZE), randrange(0, RES, SIZE)
        length += 1
        fps += 0.5
        score += 1

    pygame.display.flip()
    clock.tick(fps)
# закрытие приложения

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit()
```

Рисунок №3- движение змейки

*Теперь сделаем чтобы игра определяла когда мы проиграем.
Оканчиваться игра будет в 2 случаях когда змейка врежется в край окна или врежется в саму себя.

*Так же сразу сделаем надпись “Game over” после окончания игры.
в оператор в котором определятся проигрыш игры мы добавляем Цикл While который методом render будет выводить нужный нам текст, туда же переносим проверку на закрытие приложения.

```
# game over
if x < 0 or x > RES - SIZE or y < 0 or y > RES - SIZE or len(snake) != len(set(snake)):
    while True:
        render_end = font_end.render('GAME OVER', 1, pygame.Color('orange'))
        sc.blit(render_end, (RES // 2 - 180, RES // 2.5))
        pygame.display.flip()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit()
```

Рисунок №4 - проверка на проигрыш

*Теперь нужно запретить змейки проходить саму в себя. Для решение этой проблемы создаем словарь Dirs{} в котором поместим все клавиши для управление (W,S,D,A) с пометкой true. Далее мы добавляем в параметры управления дополнительные параметры, указываю что если будет нажата клавиша вперёд нельзя будет нажать назад и так далее.

```
# Управление
key = pygame.key.get_pressed()
if key[pygame.K_w] and dirs['W']:
    dx, dy = 0, -1
    dirs = {'W': True, 'S': False, 'A': True, 'D': True, }
if key[pygame.K_s] and dirs['S']:
    dx, dy = 0, 1
    dirs = {'W': False, 'S': True, 'A': True, 'D': True, }
if key[pygame.K_a] and dirs['A']:
    dx, dy = -1, 0
    dirs = {'W': True, 'S': True, 'A': True, 'D': False, }
if key[pygame.K_d] and dirs['D']:
    dx, dy = 1, 0
    dirs = {'W': True, 'S': True, 'A': False, 'D': True, }
```

Рисунок №5 - Управление

*Теперь добавим счет игры в левом верхнем углу. Создаём переменную Score = 0, добавляем вторую переменную score +=1 в фрагменте где змейка кушает яблоко.

*Так же сделаем надпись, выберем шрифт, размер и цвет в переменной font_score.

*Отображение секций змейки, в начале цикла нужно будет просто ввести -2 к переменной SIZE.

```
x, y = randrange(0, RES, SIZE), randrange(0, RES, SIZE)
apple = randrange(0, RES, SIZE), randrange(0, RES, SIZE)
dirs = {'W': True, 'S': True, 'A': True, 'D': True, }
length = 1
snake = [(x, y)]
dx, dy = 0, 0
score = 0
fps = 5

pygame.init()
sc = pygame.display.set_mode([RES, RES])
clock = pygame.time.Clock()
font_score = pygame.font.SysFont('Arial', 26, bold=True)
font_end = pygame.font.SysFont('Arial', 66, bold=True)

while True:
    sc.fill(pygame.Color('black'))
    # отрисовка
    [(pygame.draw.rect(sc, pygame.Color('green'), (i, j, SIZE - 2, SIZE - 2))) for i, j in snake]
```

Рисунок №6- отрисовка счёта

Рисунок №7 - графический интерфейс

*последним был добавлен фон для игры. Был открыт при помощи pygame перемещенное в папку с кодом изображение и заменен вместо заливки экрана в черный.(В рис.№8,9)

```
img = pygame.image.load('naruto.jpg').convert()

run_ctrl() #управление

while True:
    dx, dy = x_y    #передача значение управление
    sc.blit(img, (0, 0))
```

Рисунок №8 - установка фона

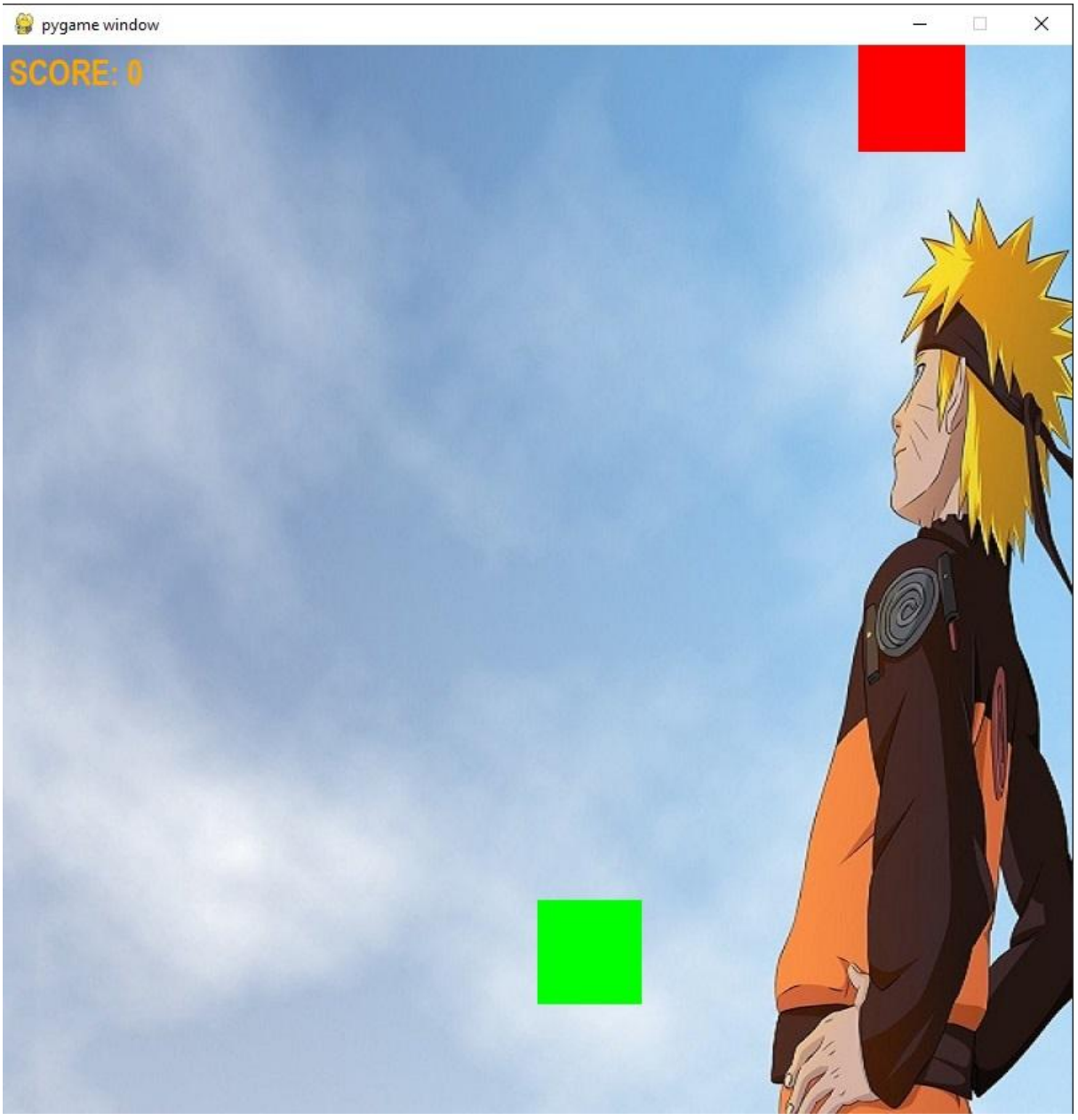

Рисунок №9 - фон игры

Исправление ошибок

После того как разработка была завершена в ходе тестирования было выявлено пара ошибок.
1) Была ошибка движка игры, из-за которой рушилась её концепция. Змейка иногда погибает не доходя до края окна. Следовательно сеанс игры преждевременно приостанавливался.
2) Также присутствовала ошибка управления вызванная задержкой выполнения команд пользователя программе. Нажатие клавиш работало через раз, это вызывает дискомфорт

Решение:
1 - проблема была решена подбором оптимального разрешения, его всегда можно поменять.

2- решением этой проблемы послужила библиотека keyboard. Баг был в том, что змейка принимала команды на поворот с задержкой, так как использовалась библиотека pygame. Исходный код остался прежним, необходимо было дописать ещё 1 функцию которая более точно контролирует управление через клавиши. И так же была добавлена дополнительная проверка в основном цикле, которая производила проверку управления перед движением объекта программы “Змейка”.

```
def run_ctrl():
    up = Control(dirs, x_y, 'W')
    down = Control(dirs, x_y, 'S')
    left = Control(dirs, x_y, 'A')
    right = Control(dirs, x_y, 'D')
    keyboard.on_press_key('W', up)
    keyboard.on_press_key('S', down)
    keyboard.on_press_key('A', left)
    keyboard.on_press_key('D', right)
```

Рисунок №10 - отладка управления

Заключение

Таким образом, в ходе курсовой работы было проведено исследование игры "Змейка" на языке программирования Python. Была рассмотрена история происхождения и развития данной игры, а также были изучены основные функции и правила игры.

Анализ различных вариантов реализации игры позволил выявить различные подходы и алгоритмы, используемые в разработке змейки на Python. Были рассмотрены основные принципы движения змейки, обработки коллизий и механизмы увеличения сложности игры.

Также в работе были проанализированы возможные стратегии и тактики игры, такие как стратегии выбора направления движения змейки, определение лучшего маршрута и избегание столкновений со стенами и самой змейкой.

Однако, разработка игры "Змейка" также включает ряд сложностей, таких как оптимизация производительности, управление состоянием игры, реализация взаимодействия с пользователем и обработка различных событий.

В итоге, курсовая работа представляет полное исследование игры "Змейка" на языке программирования Python, включая анализ основных функций и правил игры, анализ различных вариантов реализации, а также рассмотрение возможных стратегий и тактик игры. Работа также выделяет сложности, с которыми можно столкнуться при разработке данной игры, и предлагает возможные направления дальнейших исследований и улучшений в разработке змейки на Python.

Использованная литература

1. "Игры на Python с использованием Pygame" (2019) - автор Свейгад А.
2. "Python и Pygame: разработка компьютерных игр" (2017) - авторы Сэм Коллинз и Андрей Копейкин.
3. "Знакомство с Python" (2023)
4. "Python и Pygame. Создание игр" (2014) - автор Робин Аукиншоу.
5. "Python для детей. Самоучитель по программированию" (2017) - автор Джейсон Р. Бриггс.
6. "Изучаем программирование на Python" (2017) - автор Бэрри П.
7. "Программируем на Python" (2019) - автор Доусон М
8. "Учим Python, делая крутые игры" (2018) - автор Свейгард Э.
9. https://metanit.com/python/tutorial/2.2.php
10. https://pythonworld.ru/osnovy/cikly-for-i-while-operatory-break-i-continue-volshebnoe-slovo-else.html
11. https://habr.com/ru/articles/588605/
12. https://pythonworld.ru/moduli/modul-random.html
13. https://www.jetbrains.com/ru-ru/pycharm/
14. https://docs.python.org/3/