Министерство науки и высшего образования РФ

Рыбинский государственный авиационный технический университет

имени П.А. Соловьева

КАФЕДРА ВЫЧИСЛИТЕЛЬЫХ СИСТЕМ

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»

на тему «Игра «Тетрис»

Выполнил Гусенков Д. В.

(фамилия, имя, отчество)

студент гр. ЗИС-23

Преподаватель Пруктишина В.А.

(фамилия, имя, отчество)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

г. Рыбинск

2025 г.

**Содержание**

[Введение 3](#_Toc197006994)

[1 Основная часть 4](#_Toc197006995)

[1.1 Анализ задания 4](#_Toc197006996)

[1.2 Механизм игры: 4](#_Toc197006997)

[1.3 Алгоритм и условия игры: 4](#_Toc197006998)

[1.4 Инструменты выполнения. 5](#_Toc197006999)

[1.5 Библиотека «Pygame». 5](#_Toc197007000)

[1.6 Среда разработки PyCharm. 6](#_Toc197007001)

[1.7 Операционная система Windows. 7](#_Toc197007002)

[1.8 Технические требования. 7](#_Toc197007003)

[2. Диаграмма классов 8](#_Toc197007004)

[3 Алгоритмы 10](#_Toc197007005)

[4 Список функций. 12](#_Toc197007006)

[5 Тестирование нагрузочное, тестирование на безопасность. 14](#_Toc197007007)

[6 Инструкция пользователя. 16](#_Toc197007008)

[Заключение 17](#_Toc197007009)

[Список использованных источников и литературы. 18](#_Toc197007010)

[Приложение А (Листинг кода) 19](#_Toc197007011)

# ****Введение****

В данной курсовой работе представлена информация о реализации классической игры «Тетрис», выполненной на языке Python с использованием библиотеки Pygame.

Механика игры заключается в управлении падающими с верхней части экрана блоками, состоящими из четырёх квадратов: их нужно складывать в один непрерывный ряд без пробелов. Когда ряд полностью заполняется, он исчезает, освобождая место для новых блоков. Игрок зарабатывает очки за каждый исчезнувший ряд. Игра продолжается до тех пор, пока блоки не заполнят весь экран.

Данная игра была разработана для обеспечения простоты и лёгкости в управлении, а также для обучения принципам программирования с использованием библиотеки Pygame. В документации рассматриваются основные компоненты игры, её архитектура, алгоритмы работы и тестирование.

# ****1 Основная часть****

## ****Анализ задания****

Задание заключалось в разработке игры «Тетрис» с использованием принципов объектно-ориентированного программирования (ООП). Игра должна быть реализована таким образом, чтобы обеспечивать простоту в добавлении новых функциональных возможностей и изменении существующих, а также демонстрировать возможности ООП, такие как инкапсуляция, наследование и полиморфизм.

В рамках задания необходимо было решить несколько ключевых задач:

## 1.2 Механизм игры:

Когда по горизонтали заполняется строка из 10 клеток, она исчезает. Очки начисляются за каждую исчезнувшую строку. Скорость падения каждой последующей фигурки нарастает

Игра заканчивается, когда новая фигурка уже не может поместиться в параметры поля, и тогда подсчитываются итоговые набранные очки за игру.

## 1.3 Алгоритм и условия игры:

Сверху в «стакан» шириной 10 и высотой 20 клеток падают разноцветные фигуры. Фигуры составлены из четырёх квадратных клеток таким образом, что каждый квадрат имеет с каким-нибудь другим квадратом общую сторону.

Условия игры:

Во время падения игрок может двигать фигуры по горизонтали (в пределах стакана) стрелками и , поворачивать их на 90° в разные стороны стрелкой , и, прицелившись, уронить их на дно стакана.

Фигурка летит, пока не наткнётся на другую фигурку либо на дно стакана. 1

Если при этом заполнился горизонтальный ряд из 10 клеток, он пропадает и всё, что выше него, опускается на одну клетку. 1

В специальном поле игрок видит фигурку, которая будет следовать после текущей — эта подсказка позволяет планировать свои действия. 1

Темп игры постепенно увеличивается. 1

Игра заканчивается, когда новая фигурка не может поместиться в стакан.

## ****Инструменты выполнения.****

Для разработки игры "Тетрис" использовались различные инструменты и технологии, которые обеспечили удобство разработки, а также позволили эффективно реализовать все необходимые функции. Ниже приведены основные инструменты, использованные для выполнения задания.

Язык программирования – Python был выбран в качестве языка программирования для разработки игры по следующим причинам:

* простота синтаксиса;
* широкая поддержка библиотек;
* простота отладки и тестирования.

## 1.5 Библиотека «Pygame».

Pygame — это популярная библиотека для разработки 2D-игр на Python, предоставляющая удобные средства для работы с графикой, аудио и событийной системой. Основные особенности и функции Pygame, использованные в проекте:

* Отображение графики: Pygame предоставляет функции для рисования объектов на экране. В нашем случае использовалась функция pygame.draw.rect(), которая позволила отобразить змейку и еду в виде прямоугольников;
* Обработка событий: для обработки ввода с клавиатуры и других взаимодействий пользователя используется система событий Pygame. С помощью pygame.event.get() программа получает информацию о нажатиях клавиш, что позволяет управлять направлением движения змейки;
* Частота обновления экрана: Pygame предоставляет механизм для управления частотой обновления экрана, что важно для поддержания плавности игры. В данном проекте это реализовано с помощью метода pygame.time.Clock() для фиксации FPS (10 кадров в секунду);
* Работа с текстом: Для отображения информации, такой как длина змейки, использована функция pygame.font.SysFont(), которая позволяет выводить текст на экране с заданным шрифтом и размером.

## 1.6 Среда разработки PyCharm.

Для написания и отладки кода была использована среда разработки PyCharm. Это IDE предназначена для Python-разработки, она имеет следующие преимущества:

* Интеллектуальная поддержка кода: PyCharm предлагает автодополнение кода, подсветку синтаксиса, рефакторинг и другие функции, которые ускоряют процесс написания и корректировки кода.
* Отладчик: PyCharm предоставляет удобный отладчик, который позволяет отслеживать выполнение программы, анализировать переменные и следить за логикой работы программы на разных этапах.
* Интеграция с виртуальными окружениями: PyCharm поддерживает работу с виртуальными окружениями Python, что позволяет изолировать зависимости проекта и избежать конфликтов между библиотеками.

## 1.7 Операционная система Windows.

Игра была разработана и протестирована на операционной системе Windows. Операционная система играет важную роль в обеспечении стабильности работы Pygame, так как для корректной работы графики и обработки событий важно учитывать особенности работы с окнами и графикой на различных платформах.

## 1.8 Технические требования.

Python версии 3.x, для использования возможностей языка и совместимости с библиотеками.

Pygame версии 2.x, для работы с возможностями библиотеки, поддержкой графики и событий.

## ****Диаграмма классов****

Для структурирования и визуализации объектов и их взаимодействий в игре была использована объектно-ориентированная модель, которая включала несколько ключевых классов, каждый из которых выполняет свою роль в процессе игры. Важно отметить, что каждый класс инкапсулирует определённую функциональность, что облегчает расширение и модификацию игры.

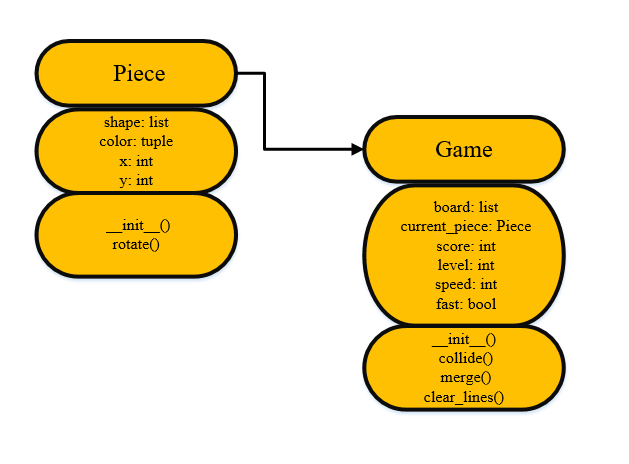


Рисунок 1 - Диаграмма классов для игры "Тетрис"

Piece: Класс, представляющий текущий блок Тетриса. Хранит информацию о форме блока, его цвете и текущем положении.

Game: Класс, управляющий логикой игры, включая игровое поле, текущую фигуру, подсчет очков и уровней.

**Piece.**

Piece: представляющий текущий блок Тетриса.

shape: список, определяющий форму фигуры.

color: кортеж с цветом фигуры.

x: текущая позиция по оси X.

y: текущая позиция по оси Y.

init(): инициализация новой фигуры с случайной формой и цветом.

rotate(): поворот фигуры на 90 градусов по часовой стрелке.

**Game.**

Game: управляющий логикой игры, включая игровое поле, текущую фигуру, подсчет очков и уровней.

board: двумерный список, представляющий игровую доску.

current\_piece: текущая фигура (Piece).

score: текущий счет.

level: текущий уровень игры.

speed: время в миллисекундах между падениями фигур.

fast: булевый флаг (не используется в текущей версии, может быть для будущих улучшений).

init(): инициализация игрового состояния, создание пустой доски и текущей фигуры.

collide(): проверяет коллизию текущей фигуры с границами доски или другими фигурками.

merge(): сливает текущую фигуру с игровой доской, добавляя цветовые значения в матрицу.

clear\_lines(): очищает заполненные линии, обновляет очки и уровень.

## ****3 Алгоритмы****

Инициализация игры:

* запускается библиотека Pygame.
* определяются размеры окна, блоки и цвета фигур.
* определяются фигуры Тетриса.

Создание классов:

* piece: хранит информацию о текущей фигуре и ее состоянии.
* game: отвечает за игровую логику, состояние игры, подсчет очков и уровня.

Основной игровой цикл:

* обновление состояния окна и обработка событий (нажатия клавиш).
* передвижение и вращение фигуры.

Проверка коллизий:

* если фигура достигла нижней границы или соприкасается с другими фигурами, фиксируем ее на поле.
* обновление поля: очищаем заполненные линии.
* отрисовка текущего состояния игры: фигуры и игровое поле, актуальные значения очков и уровня.
* для управления змейкой используется клавиатура. Игрок может изменять направление движения и поворота помощью «Клавиш стрелок».

Доработка:

уровень увеличивается при достижении определенного количества очков, а скорость увеличивается, чтобы усложнить игру.

**Блок-схема алгоритмов**

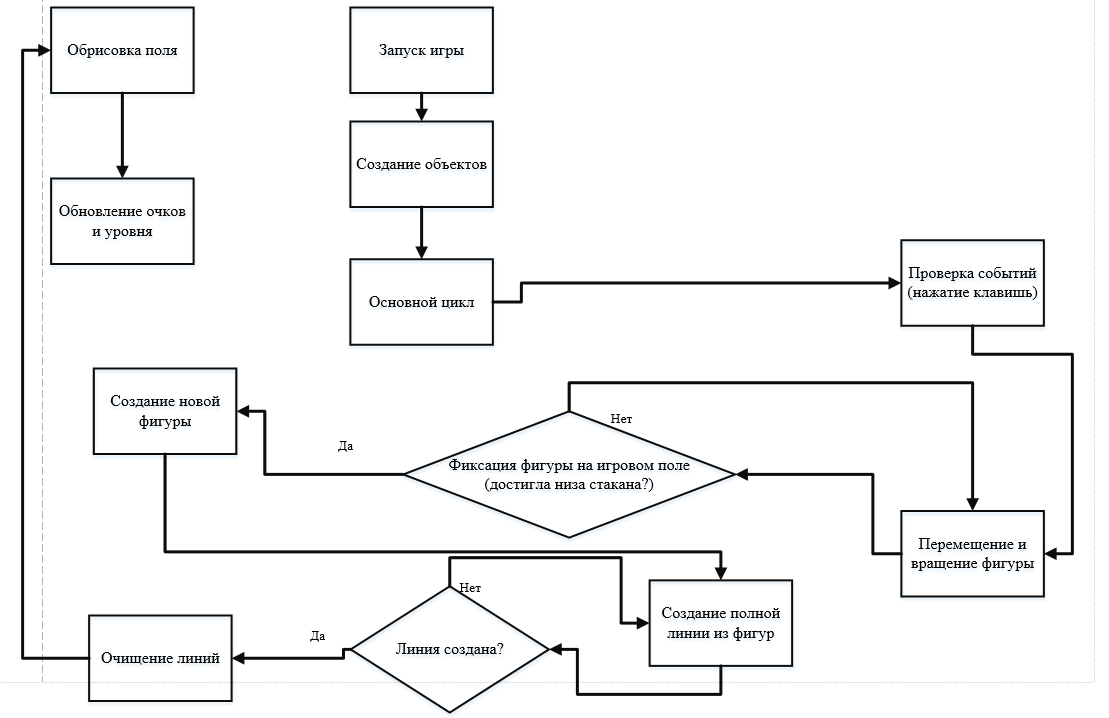


Рисунок 2 - Блок-схема алгоритмов игры "Тетрис"

## ****Список функций.****

**init()** (в классе Piece):

Описание: Инициализирует новый объект фигуры, выбирая случайное направление и цвет, а также устанавливает начальные координаты (x=3, y=0).

Возвращает: N/A

**rotate()** (в классе Piece):

Описание: Поворачивает фигуру на 90 градусов по часовой стрелке. Применяется операция зипа для смены местоположения элементов в матрице.

Возвращает: N/A

**init()** (в классе Game):

Описание: Инициализирует новое игровое состояние, создавая пустую доску (матрицу) и устанавливая начальные параметры игры (текущая фигура, счет, уровень и скорость).

Возвращает: N/A

**collide()** (в классе Game):

Описание: Проверяет, есть ли коллизия между текущей фигурой и границами доски, или если фигура касается другой фигуры на доске.

Возвращает: True или False

**merge()** (в классе Game):

Описание: Объединяет (сливает) текущую фигуру с игровой доской, заполняя соответствующие ячейки цветом.

Возвращает: N/A

**clear\_lines()** (в классе Game):

Описание: Проверяет заполненные линии на доске. Если линия заполнена, она очищается (удаляется), и новые пустые строки добавляются сверху. Также обновляются счет и уровень.

Возвращает: N/A

**main():**

Описание: Главная функция, управляющая игровым циклом, обрабатывающая ввод от пользователей и обновляющая состояние игры по мере ее выполнения. Также отвечает за отрисовку фигуры и игрового поля.

Возвращает: N/A

## ****Тестирование нагрузочное, тестирование на безопасность.****

В процессе разработки игры «Тетрис» было проведено базовое (функциональное) тестирование, направленное на проверку корректности работы при стандартных и граничных условиях. Нагрузочное тестирование проводилось с целью определить стабильность поведения игры при увеличении длины змейки, а также при максимальной частоте кадров.

Во время игрового процесса принцип создания фигур и выбора цвета происходил в штатном режиме. При этом игра сохраняет стабильную частоту обновления кадров и не демонстрирует признаков деградации производительности.

Тестирование на безопасность проводилось с акцентом на устойчивость программы к возможным некорректным действиям пользователя. Были проверены сценарии, при которых игрок быстро и хаотично меняет направления движения фигуры, программа отработала без сбоев и ошибок, завершение игры происходило корректно с освобождением всех ресурсов.

Также была проверена защита от выхода за границы экрана. Функциональное тестирование подтвердило устойчивость и надёжность работы приложения в рамках поставленной задачи.

Таблица 1 - Результаты тестирования игры «Змейка»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Описание** | **Ожидаемая реакция** | **Фактическая реакция** | **Вывод** |
| **1** | Игрок меняет положение фигуры в самом низу в последний момент | Игра должна оставаться плавной, без снижения FPS или ошибок | Игра продолжает работать стабильно, без зависаний или тормозов | Нагрузочное тестирование пройдено |
| **2** | Игрок сам ускоряет падение фигуры в низ. | Игра должна оставаться плавной, без снижения FPS или ошибок | Игра продолжает работать стабильно, без зависаний или тормозов | Устойчивость к пользовательским действиям подтверждена |
| **3** | Фигуры не выходят за границы игрового поля | Игра продолжает работу. | Игра просто не пускает фигуру за пределы экрана. | Проверка на выход за границы пройдена |
| **4** | Фигурки складываются, но не образуют полную лини. | Игра не заканчивается | Фигурки продолжают создаваться и падать. | Проверка на пересечение фигур пройдена |
| **5** | Полная линия и фигур создана. | Игра не заканчивается | Полная линия пропадает, счетчик очков увеличивается, фигуры продолжают создаваться. | Функциональность подтверждена |
| **6** | Игровое пространство полностью заполнено | Игра заканчивается | Фигуры перестают создаваться, потому что пространство полностью заполнено. | Функциональность подтверждена |
| **7** | Попытка запустить игру повторно при уже работающем экземпляре (если реализовано) | Игра не должна запускать второй экземпляр или должна корректно разделять процессы | Запуск второго экземпляра возможен, но работает корректно и независимо | Поведение допустимо, сбои отсутствуют |
| **8** | Проверка на утечку памяти при длительной игре (10+ минут) | Игра не должна потреблять чрезмерно оперативную память или замедляться | Потребление памяти остаётся стабильным, утечек не обнаружено | Тест пройден, производительность стабильна |

## ****Инструкция пользователя.****

Начать игру на самом лёгком уровне. По ходу игры сложность уровней будет увеличиваться.

Передвигать фигурки тетрамино вправо и влево при помощи кнопок или клавиш со стрелками. Нужно попробовать передвинуть тетрамино так, чтобы они сложились в ряд.

Использовать разные фигурки тетрамино. В традиционной игре есть семь типов тетрамино: I-образные, O-образные, L-образные, J-образные, S-образные, Z-образные и T-образные.

Вращать тетрамино так, чтобы фигура заполнила нужный пробел. Для вращения использовать стрелке «вверх», для перемещения стрелки «Право» или «Лево». Заполнять все ряды, чтобы завершить уровень и получить очки. Чем больше рядов заполнено одновременно, тем больше очков получено.

# ****Заключение****

В рамках данной курсовой работы была разработана игра «Тетрис» с использованием языка программирования Python и библиотеки Pygame. Проект был реализован с соблюдением принципов объектно-ориентированного программирования, что позволило структурировать код, повысить его читаемость и упростить сопровождение.

В процессе работы были реализованы базовые алгоритмы движения, столкновения, управления и перезапуска, проведено тестирование стабильности и нагрузки, а также подготовлена пользовательская инструкция. Все эти шаги обеспечили корректную и устойчивую работу игры в различных условиях.

# Список использованных источников и литературы.

1. Статья «тетрис на Python», информационный ресурс

https://proglib.io/p/pishem-tetris-na-python-s-pomoshchyu-biblioteki-pygame-2022-05-30

1. Статья «Как создать простую игру «Тетрис» на Python и Pygame»,

информационный ресурс https://thepythoncode.com/code/create-a-tetris-game-with-pygame-in-python

# Приложение А (Листинг кода)

import pygame

import random

# Инициализация Pygame

pygame.font.init()

# Размеры игрового окна

WIDTH, HEIGHT = 300, 600

BLOCK\_SIZE = 30

# Определения цветов

COLORS = [

(0, 0, 0), (255, 0, 0), (0, 255, 0), (0, 0, 255),

(255, 255, 0), (255, 165, 0), (75, 0, 130), (238, 130, 238)

]

# Фигуры Тетриса

SHAPES = [

[[1, 1, 1, 1]], # I

[[1, 1], [1, 1]], # O

[[0, 1, 0], [1, 1, 1]], # T

[[0, 1, 1], [1, 1, 0]], # S

[[1, 1, 0], [0, 1, 1]], # Z

[[1, 0, 0], [1, 1, 1]], # L

[[0, 0, 1], [1, 1, 1]] # J

] Класс блока

class Piece:

rows = 20

cols = 10

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

self.shape = random.choice(SHAPES)

self.color = random.choice(COLORS[1:])

self.x = 3

self.y = 0

def rotate(self):

self.shape = [list(row) for row in zip(\*self.shape[::-1])]

# Игровое поле

class Game:

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

self.board = [[0] \* Piece.cols for \_ in range(Piece.rows)]

self.current\_piece = Piece()

self.score = 0

self.level = 1

self.speed = 500 # Начальная скорость в миллисекундах

self.fast = True

def collide(self):

for i, row in enumerate(self.current\_piece.shape):

for j, val in enumerate(row):

if val and (self.current\_piece.y + i >= Piece.rows or

self.current\_piece.x + j < 0 or

self.current\_piece.x + j >= Piece.cols or

self.board[self.current\_piece.y + i][self.current\_piece.x + j]):

return True

return False

def merge(self):

for i, row in enumerate(self.current\_piece.shape):

for j, val in enumerate(row):

if val:

self.board[self.current\_piece.y + i][self.current\_piece.x + j] = self.current\_piece.color

def clear\_lines(self):

cleared\_lines = 0

for i in range(Piece.rows - 1, -1, -1):

if all(self.board[i]):

cleared\_lines += 1

del self.board[i]

self.board.insert(0, [0] \* Piece.cols)

self.score += cleared\_lines \* 100 # 100 очков за каждую линию

if cleared\_lines > 0:

self.level = self.score // 1000 + 1

self.speed = max(100, 500 - (self.level - 1) \* 50) # Увеличение скорости

# Основная функция

def main():

pygame.init()

win = pygame.display.set\_mode((WIDTH, HEIGHT))

clock = pygame.time.Clock()

game = Game()

running = True

fall\_time = 0

while running:

win.fill((0, 0, 0))

fall\_time += clock.get\_time()

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

game.current\_piece.x -= 1

if game.collide():

game.current\_piece.x += 1

if event.key == pygame.K\_RIGHT:

game.current\_piece.x += 1

if game.collide():

game.current\_piece.x -= 1

if event.key == pygame.K\_DOWN:

game.current\_piece.y += 1

if event.key == pygame.K\_UP:

game.current\_piece.rotate()

if game.collide():

game.current\_piece.rotate() # Возврат, если ротация не возможна

# Физика игры

if fall\_time >= game.speed:

game.current\_piece.y += 1

if game.collide():

game.current\_piece.y -= 1

game.merge()

game.clear\_lines()

game.current\_piece = Piece()

if game.collide():

print("Game Over")

running = False

fall\_time = 0

# Отрисовка

for i in range(Piece.rows):

for j in range(Piece.cols):

if game.board[i][j]:

pygame.draw.rect(win, game.board[i][j], (j \* BLOCK\_SIZE, i \* BLOCK\_SIZE, BLOCK\_SIZE, BLOCK\_SIZE))

for i, row in enumerate(game.current\_piece.shape):

for j, val in enumerate(row):

if val:

pygame.draw.rect(win, game.current\_piece.color,

((game.current\_piece.x + j) \* BLOCK\_SIZE,

(game.current\_piece.y + i) \* BLOCK\_SIZE,

BLOCK\_SIZE, BLOCK\_SIZE))

# Отображение очков и уровня

font = pygame.font.SysFont('Arial', 20)

score\_text = font.render(f'Score: {game.score}', True, (255, 255, 255))

level\_text = font.render(f'Level: {game.level}', True, (255, 255, 255))

win.blit(score\_text, (5, 5))

win.blit(level\_text, (5, 30))

pygame.display.update()

clock.tick(60)

pygame.quit()

if \_\_name\_\_ == "\_\_main\_\_":

main()