

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
по курсу «Программирование»
на тему «Разработка программы сложной структуры методом
нисходящего программирования. Игра “Змейка”»

18.05.2026
оф.лично
Г. Генералова А.А.

Выполнил студенты группы 25BVB3:

Никитин К. А.

Принял:

к.т.н., доцент Генералова А.А.

Пенза 2026

Оглавление

Введение.....	5
Постановка задачи.....	7
Выбор решения.....	9
Описание разработки программы.....	12
Отладка и тестирование.....	14
Описание программы.....	15
Руководство пользователя.....	24
Заключение	30
Список используемых источников.....	32
Приложение А. Листинг программы.....	33
Приложение Б. Вычислительный модуль на языке Ассемблер	40
Приложение В. Результаты тестирования программного комплекса	42

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

 М.А. Митрохин

« 10 » февраля 2026 г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Кикитичу Кириллу Александровичу Группа 25 Б/В/З 3

Тема проекта: **Разработка программы сложной структуры методом нисходящего программирования.**

Исходные данные (технические требования) на проектирование

Игра "Змейка"

Компьютер генерирует препятствия и провизию для змейки. Пользователь управляет траекторией движения змейки. В процессе поедания провизии змейка увеличивается. Змейка не должна замкнуться на себя. В файл записывается и сохраняется таблица рекордов.

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания " 19 " сентября 2026 г.

Дата защиты проекта " 28 " мая 2026 г.

Руководитель И. Печеряева Н.Ф.

Задание получил " 19 " сентября 2026 г.

Студент Иванов Н.А.

Введение

В современном мире информационные технологии и программирование играют ключевую роль не только в профессиональной сфере, но и в образовании, разработке интерактивных систем и формировании алгоритмического мышления. Компьютерные программы позволяют моделировать сложные логические процессы, автоматизировать принятие решений и обеспечивать эффективное динамическое взаимодействие между пользователем и вычислительной машиной. Игровые приложения, даже с интуитивно понятным набором правил, служат отличным полигоном для отработки практических навыков проектирования архитектуры программного обеспечения (ПО), управления состояниями объектов в реальном времени и оптимизации системных ресурсов ЭВМ.

Проектирование интерактивных приложений и классических игровых механик исторически стало фундаментом для изучения структур данных, алгоритмов ветвления, работы с оперативной памятью и обработки внешних прерываний. Несмотря на активное развитие комплексных трехмерных игровых движков, самостоятельное программирование логики визуализации и двухмерных интерфейсов на базе специализированных библиотек остаётся важнейшим этапом в подготовке специалистов. Оно требует глубокого понимания потактового исполнения кода, обработки потоков ввода-вывода (клавиатурных событий, таймеров) и обеспечения кроссплатформенной совместимости программных продуктов. Подобные проекты позволяют разработчику детально освоить чистую алгоритмику и принципы функционирования графических подсистем без использования избыточных абстракций.

В настоящее время разработка мультимодульных интерактивных систем используется как эффективный инструмент для освоения принципов объектно-ориентированного программирования, системного администрирования операционных сред и низкоуровневой интеграции. Игры с динамической механикой, такие как «Змейка», позволяют на практике

реализовать алгоритмы случайной генерации объектов в замкнутом пространстве, контроль коллизий, а также межязыковое взаимодействие компонентов, написанных на языках высокого и низкого уровней абстракции. Использование механизмов прямого обращения к памяти для оптимизации критических вычислений формирует у будущего инженера фундаментальные навыки отладки, системного тестирования и проектирования расширяемых решений, устойчивых к аппаратным ограничениям.

Данная курсовая работа посвящена программной реализации игровой программы «Змейка» на высокоуровневом языке Python [1] с использованием графической библиотеки Pygame [2] и внедрением низкоуровневого вычислительного модуля на языке Ассемблер x86-64 [3,4]. Программа включает в себя интерактивный полупрозрачный графический интерфейс главного меню на базе библиотеки pygame-menu [5], систему динамического циклического рендеринга игрового поля, а также ассемблерную вставку, разработанную для расчета параметров скорости игрового процесса на аппаратном уровне. Разработка данного проекта позволяет закрепить практические навыки работы с объектно-ориентированными структурами, системными вызовами Win32 API, механизмами управления памятью и обхода защитных политик операционной системы (DEP), связывая воедино теоретические основы архитектуры вычислительных систем и современные методы прикладной разработки ПО.

Постановка задачи

Необходимо разработать интерактивное игровое приложение «Змейка», функционирующее в режиме реального времени и обладающее развитым графическим интерфейсом пользователя. Программа должна иметь интуитивно понятное визуальное оформление, обеспечивающее комфортное взаимодействие человека с вычислительной системой. В рамках проектирования необходимо определить названия и функционал элементов управления, построить эффективные алгоритмы обработки координатных данных для управления динамическими объектами на игровом поле, исключить возникновение ошибочных ситуаций при обработке коллизий (столкновений объектов), а также реализовать низкоуровневый расчет параметров игрового процесса для обеспечения межязыковой интеграции компонентов.

Многомодульность программы. Программа должна быть разделена на логические и функциональные модули. Использование модульной структуры позволяет изолировать высокоуровневую логику визуализации интерфейса от алгоритмов обработки игровых прерываний и низкоуровневых вычислений. Это существенно упрощает поиск и локализацию ошибок при комплексном тестировании приложения, повышает читаемость исходного кода и обеспечивает возможность беспрепятственного расширения функционала программы в будущем.

Использование сложных типов данных – структур, списков, файлов. Применение структурированных типов данных необходимо для естественного и интуитивного представления игровых объектов в коде программы. Сегменты игровых объектов должны описываться специальными структурами (классами) для хранения координатных параметров, а само динамическое тело объекта должно эффективно обрабатываться с помощью последовательных списков. Использование внешних файлов требуется для хранения элементов графического оформления, что позволяет отделить ресурсы приложения от исполняемого кода.

Режим работы видеосистемы – текстовый/графический. Программа должна функционировать в графическом режиме работы видеосистемы. Для реализации проекта необходимо детально изучить способы инициализации графического окна фиксированного разрешения, механизмы циклического рендеринга растровых изображений, координатную отрисовку геометрических примитивов и вывод текстовых шрифтов на экран.

Устройство ввода информации – клавиатура, мышь. В разрабатываемом приложении необходимо обеспечить полноценную поддержку устройств ввода. Программа должна непрерывно осуществлять опрос системных очередей событий, уметь четко идентифицировать, различать и обрабатывать сигналы, поступающие от клавиатуры (для управления пространственным перемещением в игровом пространстве) и от манипулятора мышь (для навигации в элементах управления интерфейса), что значительно облегчит эксплуатацию программного продукта.

Пользовательский интерфейс на основе меню. Графический интерфейс должен быть построен на основе интерактивного меню. Это необходимо для создания структурированного, эстетичного и интуитивно понятного пространства управления, где пользователь может вводить текстовые параметры (например, имя игрока), запускать игровой цикл или корректно завершать работу с приложением без принудительного аварийного закрытия окна.

Наличие заставки. Заставка необходима для того, чтобы пользователь, запустивший программу, смог мгновенно идентифицировать приложение и получить стартовую визуальную информацию о нем. Заставка должна органично сочетаться с темой игрового процесса, служить фоновым пространством для главного меню и подготавливать графическую подсистему к переходу в основной игровой режим.

Выбор решения

При запуске разработанного программного обеспечения инициализируется графическая подсистема и на экране открывается стартовое окно приложения. Процесс работы программы разделен на три основных этапа: отображение заставки с интерактивным меню, выполнение основного игрового цикла и обработка событий завершения игры.

1. Отображение заставки и функционирование главного меню.

Сразу после инициализации окна на экране рендерится полноэкранное фоновое изображение-заставка (logo.png). Поверх заставки накладывается интерактивный полупрозрачный интерфейс управления, построенный на базе специализированного модуля `pygame-menu`. Меню включает в себя три основных компонента взаимодействия:

- **Поле текстового ввода «Имя:».** Позволяет пользователю с помощью клавиатуры ввести алфавитно-цифровой идентификатор игрока. По умолчанию в поле установлено значение «Игрок 1». Данные сохраняются в строковой переменной проекта.
- **Кнопка «Играть».** При клике левой кнопкой мыши по данному пункту меню временно скрывается, и программа передает управление функции `start_the_game()`, которая инициализирует переменные новой игровой сессии.
- **Кнопка «Выйти».** Вызывает стандартное системное событие `pygame_menu.events.EXIT`, корректно освобождает занятые графические ресурсы и закрывает окно программы.

2. Инициализация и логика игрового процесса. После нажатия кнопки «Играть» программа переходит в активный графический игровой режим реального времени. На данном этапе логика программы реализуется следующим образом:

- **Инициализация объектов.** Создается начальный связанный список `snake_blocks`, состоящий из трех сегментов (экземпляров класса `SnakeBlock`) с фиксированными стартовыми

координатами. Вызывается внутренняя функция `get_random_empty_block()`, которая с помощью генератора случайных чисел `random.randint` определяет координаты объекта «Яблоко» (`apple`) на игровом поле. При этом циклически проверяется условие коллизии: новые координаты еды не должны совпадать ни с одним из сегментов змейки.

- **Внедрение низкоуровневых вычислений.**

Непосредственно перед стартом игрового цикла через модуль `ctypes` инициализируется ассемблерная вставка. Выделяется защищенный буфер в оперативной памяти ЭВМ, системным вызовом Win32 API `VirtualProtect` ему присваиваются права на исполнение (`PAGE_EXECUTE_READWRITE`), туда записывается шестнадцатеричный машинный код x86-64, и регистрируется вызываемая C-совместимая функция расчета параметров скорости.

- **Обработка ввода информации.** Внутри игрового цикла программа непрерывно опрашивает очередь событий клавиатуры. Нажатия клавиш со стрелками (`K_UP`, `K_DOWN`, `K_LEFT`, `K_RIGHT`) изменяют переменные вектора направления движения (`buf_row`, `buf_col`). Программа блокирует возможность совершения некорректного разворота змейки «в саму себя» на 180 градусов (например, движение влево невозможно, если текущее направление — вправо).

- **Игровой цикл и рендеринг.** С частотой, задаваемой таймером, экран очищается, закрашивается фоном с чередующейся шахматной текстурой блоков (`GREEN` и `LIME`), и поверх отрисовываются: верхняя панель инструментов (`HEADER_MARGIN`) со шрифтовым выводом текущего счета (`Total`) и скорости (`Speed`), яблоко (красный квадрат) и сегменты змейки (синие квадраты).

3. Механика взаимодействия объектов и условия завершения. На каждом такте таймера выполняется проверка пространственных условий:

- **Поглощение еды.** Если координаты головы змейки (`head`) совпадают с координатами яблока (`apple`), переменная набранных очков `total` инкрементируется. Новое значение передается в низкоуровневую ассемблерную функцию, которая выполняет целочисленное деление на уровне регистров процессора и возвращает обновленное значение скорости `speed`. К хвосту змейки добавляется новый блок, а яблоко регенерируется в случайном свободном месте поля.
- **Движение.** К координатам головы прибавляются смещения, формируя объект `new_head`, который добавляется в конец списка, а первый элемент списка (хвост) удаляется методом `.pop(0)`.
- **Столкновение со стеной.** Метод `.is_inside()` проверяет, вышли ли координаты головы за пределы диапазона от 0 до `COUNT_BLOCKS - 1`. Если условие нарушено, в консоль выводится сообщение `crash`, игровой цикл прерывается оператором `break` и управление возвращается обратно в главное меню.
- **Столкновение с собой.** Оператор `if new_head in snake_blocks` проверяет наличие новой координаты головы в списке существующих сегментов тела. В случае пересечения фиксируется коллизия `crash yourself`, и программа аналогично возвращает пользователя на экран заставки меню с сохранением возможности начать новую игру.

Навигация в интерфейсе приложения носит гибридный характер: управление элементами главного меню осуществляется с помощью манипулятора мышь, а управление динамическим поведением главного игрового объекта — с помощью клавиатуры.

Описание разработки программы

Разработанная игровая программа имеет модульную структуру. Проектирование и программная реализация компонентов распределялись по логическим модулям и выполнялись в следующем порядке:

1. **Модуль инициализации графического контекста (pygame)** — базовый модуль видеосистемы. Отвечает за создание игрового окна фиксированного размера, задание заголовка приложения и запуск системного таймера контроля частоты кадров (pygame.time.Clock).

2. **Структурный модуль элементов (class SnakeBlock)** — внутренний модуль описания базовой единицы игрового пространства. Хранит координаты x и y, содержит перегруженный метод сравнения `__eq__` для обнаружения коллизий и логический метод `is_inside()` для контроля выхода за границы поля.

3. **Модуль рендеринга статических объектов (draw_block)** — функция графического отображения элементов. Рассчитывает точные пиксельные координаты блоков с учетом заданных отступов (MARGIN) и размеров (SIZE_BLOCK) для отрисовки шахматного поля, яблока и сегментов змейки.

4. **Модуль процедурной генерации (get_random_empty_block)** — алгоритмический блок распределения ресурсов. Использует генератор псевдослучайных чисел для размещения яблока и содержит циклический фильтр коллизий со структурами тела змейки.

5. **Низкоуровневый модуль вычислений (asm_bytes / ctypes)** — ассемблерная вставка, интегрированная в оперативную память. Отвечает за потактовое деление и инкремент регистров на уровне процессора x86-64 для вычисления задержки игрового цикла [6].

6. **Системный интерфейсный модуль операционной системы (ctypes.windll.kernel32)** — подмодуль конфигурации ядра ОС Windows. Используется для изменения прав дескриптора выделенной памяти (VirtualProtect) с целью обхода защитных аппаратных ограничений выполнения данных (DEP) [7].

7. **Основной управляющий модуль игры (start_the_game)** — инкапсулирует в себе переменные состояния сессии, игровой цикл обработки клавиатурных прерываний, логику перемещения головы змейки и её хвоста, а также условия фиксации проигрыша.

8. **Модуль пользовательского интерфейса (pygame_menu)** — внешняя библиотека управления визуальными компонентами. Отвечает за создание полупрозрачной темы меню (THEME_DARK), обработку мыши и конфигурацию интерактивных виджетов ввода имени и кнопок управления.

9. **Модуль загрузки внешних ресурсов (pygame.image.load)** — обеспечивает файловый ввод, считывая внешнее растровое изображение logo.png и преобразуя его в графическую подложку (заставку) для экрана меню.

10. **Основной диспетчерский цикл программы (main)** — координирует переключение контекстов между заставкой-меню и игровым окном, непрерывно обновляя экран через pygame.display.update().

По порядку реализации компонентов можно судить о том, что при разработке программного комплекса применялся **восходящий метод реализации программы**. Данный подход основан на первоначальном проектировании и отладке базовых, низкоуровневых элементов (таких как класс отдельного блока, функции отрисовки и генерации объектов), с последующим их объединением в крупные функциональные узлы (игровой цикл) и финишным переходом к глобальной управляющей надстройке — интерактивному графическому меню-заставке. Применение восходящего

метода позволило эффективно протестировать математическую корректность обработки координат и коллизий до момента создания интерфейса пользователя, что гарантирует высокую стабильность работы итогового приложения.

Отладка и тестирование

В качестве интегрированной среды разработки (IDE) программного комплекса была выбрана программа **JetBrains PyCharm**. Данная среда обладает всем спектром инструментов, необходимых для написания, рефакторинга и отладки приложений на языке Python. Для поиска скрытых дефектов и локализации ошибок использовались такие возможности встроенного отладчика, как точки останова, пошаговая трассировка алгоритмов и анализ текущего содержимого переменных в оперативной памяти.

Тестирование разработанного программного обеспечения проводилось как в процессе написания отдельных модулей (в рамках восходящего тестирования), так и после завершения интеграции всех компонентов в единую систему. В ходе отладки были выявлены и успешно устранены проблемы, связанные с согласованием вызовов между языками высокого и низкого уровней (особенности соглашения о вызовах x64 calling conventions при работе с системным буфером оперативной памяти), корректной инициализацией графических ресурсов интерфейса меню, а также обработкой граничных состояний игрового поля (коллизии с телом змейки, выход за пределы координатной сетки и динамическое изменение задержки системного таймера).

Итоговое комплексное тестирование подтвердило стабильность работы программы, корректность межязыкового интерфейса и полное соответствие разработанного приложения всем исходным техническим требованиям.

Описание программы

Основным модулем разработанного программного комплекса является файл `main.py`. При запуске приложения инициализируются графические подсистемы и подгружается внешний файл заставки `logo.png`, на фоне которого разворачивается полупрозрачное интерактивное меню пользователя. Пользователю предоставляется возможность ввести идентификационное имя игрока для локальной системы учета, запустить сессию или завершить работу.

После инициализации игрового процесса управление передается выделенному циклу, который функционирует в графическом режиме и осуществляет непрерывный опрос системных прерываний ввода (клавиатуры), контроль пространственных коллизий, а также обращение к изолированной ассемблерной функции для расчета параметров задержки таймера. Полное описание используемых в программе констант, глобальных переменных, классов и внутренних функций представлено в таблицах 1, 2 и 3.

Таблица 1 — Глобальные константы и параметры видеосистемы

Идентификатор	Тип данных	Назначение / Описание компонента
<code>SIZE_BLOCK</code>	Целое число (<code>int</code>)	Размер одного логического игрового блока (ячейки) в пикселях (значение: 20).
<code>COUNT_BLOCKS</code>	Целое число (<code>int</code>)	Количество ячеек по вертикали и горизонтали игрового поля (размерность сетки: 20x20).
<code>HEADER_MARGIN</code>	Целое число (<code>int</code>)	Высота верхней информационной панели инструментов (счет, скорость) в пикселях.
<code>MARGIN</code>	Целое число (<code>int</code>)	Ширина разделительной рамки между игровыми ячейками в пикселях.
<code>size</code>	Список (<code>list</code>)	Двухэлементный массив, определяющий итоговое пиксельное разрешение окна приложения.

Продолжение таблицы 1

FRAME_COLOR	Кортеж (tuple)	Цвет внешней ограничительной рамки игрового поля в формате RGB.
GREEN / LIME	Кортеж (tuple)	Цвета для генерации шахматной текстуры подложки игрового поля.
RED	Кортеж (tuple)	Цвет графического отображения статического объекта «Яблоко».
SNAKE_COLOR	Кортеж (tuple)	Цвет графического рендеринга динамических сегментов тела змейки.
WHITE	Кортеж (tuple)	Цвет шрифтового вывода текстовой информации на панели инструментов.

Таблица 2 — Структурные компоненты и объекты классов

Идентификатор	Тип данных / Базовый класс	Назначение / Описание компонента
SnakeBlock	Объектный класс (class)	Структурный тип данных, описывающий единичный блок на координатной сетке.
SnakeBlock.x	Целое число (int)	Координата текущего блока по горизонтальной оси игровой сетки.
SnakeBlock.y	Целое число (int)	Координата текущего блока по вертикальной оси игровой сетки.
screen	Модульный объект (Surface)	Главный графический контекст (окно отображения), созданный через <code>pygame.display.set_mode</code> .
timer	Системный объект (Clock)	Экземпляр системного таймера <code>pygame.time.Clock</code> для контроля частоты кадров.

Продолжение таблицы 2

<code>courier</code>	Шрифтовой объект (Font)	Системный шрифт типа Courier, инициализированный для вывода текстовых метрик.
<code>bg_image</code>	Графический объект (Surface)	Растровое изображение заставки, загруженное из внешнего файла <code>logo.png</code> .
<code>main_theme</code>	Объект темы (Theme)	Структура конфигурации визуального стиля главного меню (THEME_DARK).
<code>menu</code>	Объект интерфейса (Menu)	Экземпляр интерактивного графического меню пользователя.

Таблица 3 — Локальные переменные и функции игрового процесса `start_the_game`

Идентификатор	Тип данных / Сигнатура	Назначение / Описание компонента
<code>snake_blocks</code>	Список объектов (list)	Динамический массив (связанный список), хранящий сегменты тела змейки.
<code>apple</code>	Объект (SnakeBlock)	Экземпляр класса, содержащий текущие случайные координаты еды на поле.
<code>d_row/d_col</code>	Целое число (int)	Текущие активные смещения (вектор движения) змейки по осям координат.
<code>buf_row/ buf_col</code>	Целое число (int)	Буферные переменные для безопасной фиксации направления до начала нового такта.
<code>total</code>	Целое число (int)	Счетчик игрового прогресса (количество поглощенных яблок).

Продолжение таблицы 3

<code>speed</code>	Целое число (int)	Текущий уровень скорости игрового процесса (задает частоту тактов таймера).
<code>asm_bytes</code>	Строка байт (bytes)	Машинный код ассемблерной функции (инструкции x86-64 процессора).
<code>buf_addr</code>	Целое число (int)	Физический 64-битный адрес выделенного буфера памяти под байт-код.
<code>asm_calculate_speed</code>	Вызываемый интерфейс (ctype)	Низкоуровневый межязыковой указатель для прямого обращения к ассемблерному модулю.
<code>get_random_empty_block()</code>	Локальная функция	Процедура случайной генерации яблока с циклическим контролем коллизий.
<code>draw_block()</code>	Глобальная функция	Процедура отрисовки цветных геометрических примитивов ячеек на экране.

Блок-схема основного алгоритма управления приложением изображена на рисунке 1. В начале программы происходит вызов системных методов инициализации графической подсистемы `pygame.init()`. Далее выполняется загрузка внешнего графического файла заставки `logo.png` и конфигурация полупрозрачной темы интерактивного меню `main_theme`. Программа создает объект `menu` и добавляет в него виджеты ввода имени пользователя и кнопок навигации.

Затем программа входит в бесконечный цикл обработки контекстов. На каждой итерации на экране отрисовывается растровая подложка-заставка. Программа опрашивает системную очередь событий ОС Windows. Если зафиксировано событие закрытия графического окна (`pygame.QUIT`), управление передается процедуре `exit()`, которая освобождает дескрипторы и завершает работу приложения. В противном случае, если меню активно, вызываются методы `menu.draw(screen)` и `menu.update(events)`, которые переводят программу в режим ожидания действий пользователя. При клике на кнопку «Выйти» приложение закрывается, а при нажатии на кнопку «Играть» происходит вызов предопределенной функции игрового процесса `start_the_game()`. После завершения игровой сессии (вследствие проигрыша) управление бесконечным циклом снова возвращается к отрисовке заставки и главного меню.



Рисунок 1 – Основной алгоритм

Блок-схема алгоритма выполнения игровой сессии приведена на рисунке 2 и 3. Внутри функции `start_the_game` на подготовительном этапе инициализируется ассемблерная вставка. Машинный код x86-64 записывается в выделенный буфер памяти, которому с помощью вызова функции `VirtualProtect` присваиваются права на исполнение, и регистрируется указатель `asm_calculate_speed`. Далее формируется начальный связанный список `snake_blocks` из трех сегментов, устанавливается стартовый вектор движения, обнуляется счетчик очков `total` и генерируется случайная стартовая позиция яблока `apple` через вызов локальной функции `get_random_empty_block()`.

После этого программа входит в бесконечный цикл игрового процесса. Сначала опрашивается очередь событий ввода. Если нажаты стрелки клавиатуры, буферные переменные направления изменяются, за исключением попыток совершить разворот на 180 градусов. Далее экран очищается и закрашивается фоновой шахматной текстурой блоков, рендерится верхняя панель со значениями счета и скорости, а также выводятся яблоко и все сегменты тела змейки.

Затем проверяется условие выхода головы змейки за границы координатной сетки $0 \leq x, y < 20$. Если условие нарушено, цикл прерывается оператором `break` и функция завершается. Далее проверяется коллизия: если координаты головы змейки совпали с координатами яблока, переменная `total` увеличивается на единицу, вызывается низкоуровневая ассемблерная функция для вычисления новой скорости `speed`, яблоко регенерируется, а его старая позиция добавляется в хвост змейки. В противном случае змейка совершает стандартный шаг: к голове прибавляется текущий вектор смещения, новая координата заносится в конец списка, а первый элемент из начала списка удаляется. В конце итерации проверяется коллизия головы с собственным телом. Если факт пересечения подтвержден, игровой цикл прерывается, и программа осуществляет выход из функции, возвращая пользователя к главному меню.

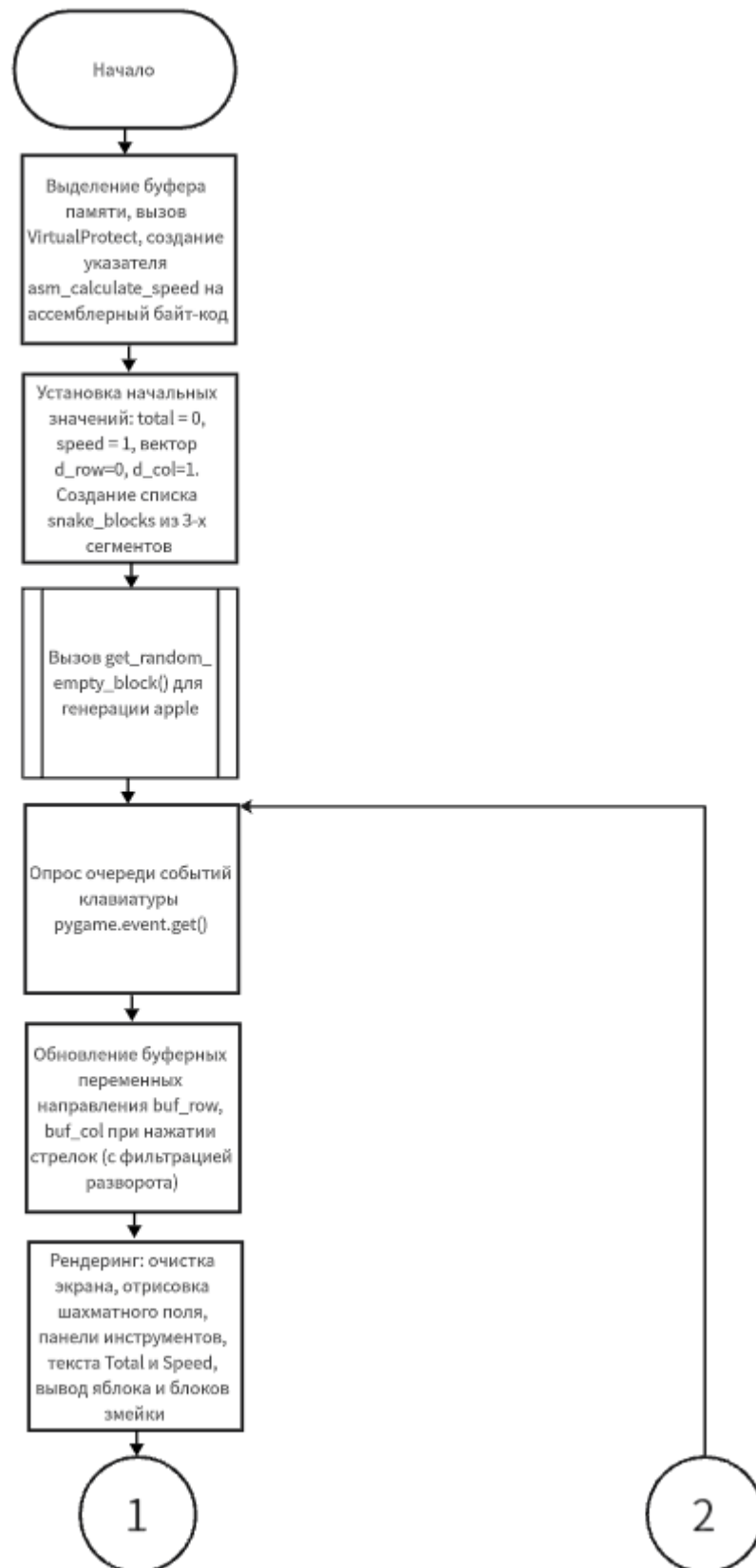


Рисунок 2 – Функция start_the_game

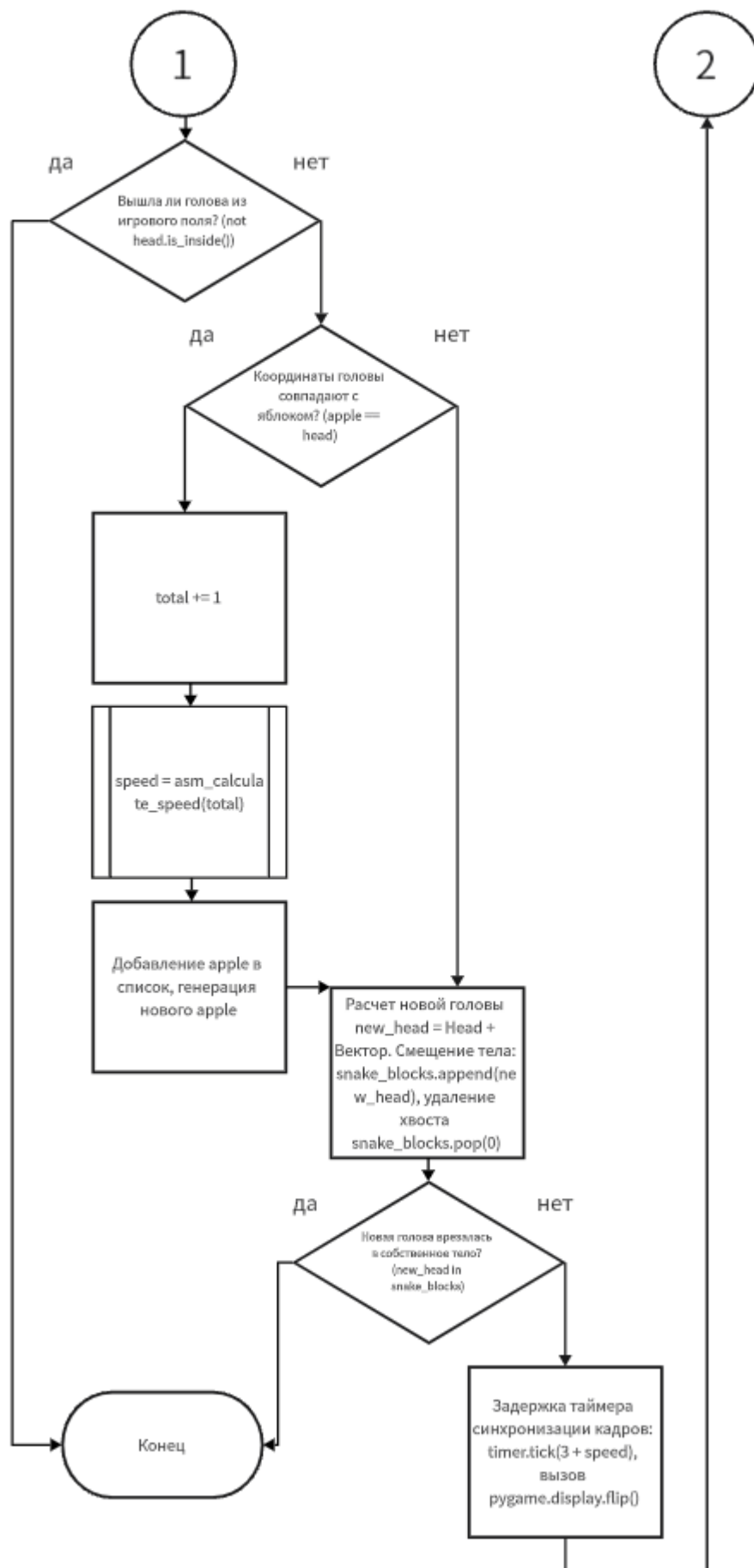


Рисунок 3 - Функция start_the_game

Таким образом в данном разделе изображены блок схемы, которые описывают алгоритм выполнения программы.

Руководство пользователя

Игровая программа «Змейка» предназначена для индивидуального использования пользователем на персональном компьютере под управлением операционной системы семейства Windows. Игра функционирует в реальном времени в двухмерном графическом режиме. Игровой процесс заключается в управлении движением виртуального объекта (змейки) с целью поглощения генерируемой в случайных местах поля еды (яблока). С каждым поглощенным яблоком длина тела змейки увеличивается, а скорость её перемещения динамически возрастает на аппаратном уровне посредством ассемблерного модуля. Ниже приведено подробное описание всех этапов взаимодействия с программой.

1. Запуск программы и главное меню

После запуска главного исполняемого файла инициализируется графическое окно приложения фиксированного размера, и перед пользователем открывается приветственный экран-заставка со встроенным полупрозрачным интерактивным меню. На данном этапе пользователь может увидеть название игры и настроить параметры текущей игровой сессии (рисунок 4).



Рисунок 4 – Главное меню

Для настройки сессии и управления элементами интерфейса используются следующие действия:

- **Ввод имени игрока:** Используя манипулятор мышь, необходимо кликнуть по текстовой области поля «Имя:», после чего с клавиатуры ввести свой алфавитно-цифровой идентификатор (например, «Игрок 1»).

- **Запуск игрового процесса:** Для перехода к активной фазе игры необходимо навести курсор мыши на интерактивную кнопку «Играть» и совершить клик левой кнопкой мыши.
- **Закрытие приложения:** Для выхода из программы до начала игровой сессии необходимо нажать кнопку «Выйти» в интерфейсе меню или воспользоваться стандартным системным крестиком закрытия окна в правом верхнем углу.

2. Процесс игры и элементы управления

После нажатия кнопки «Играть» графический контекст переключается на основное игровое поле, представляющее собой замкнутое пространство с шахматной текстурой (рисунок 5). Змейка начинает автоматическое прямолинейное движение со стартовой скоростью.

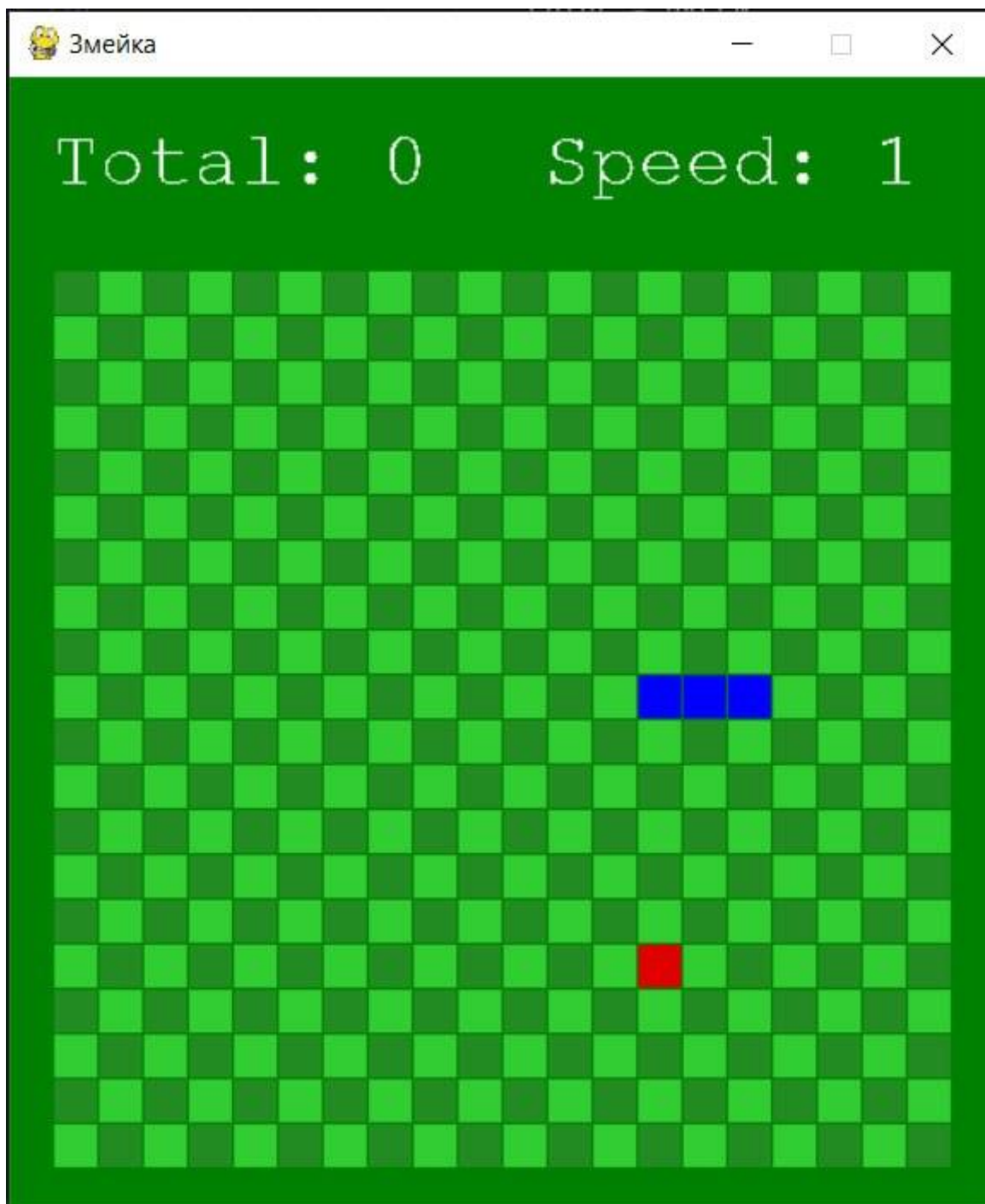


Рисунок 5 – Активный игровой процесс

Взаимодействие с игровым процессом осуществляется на основе следующих правил:

- **Навигация (управление движением):** Изменение пространственного вектора перемещения змейки осуществляется с помощью клавиатуры посредством нажатия клавиш навигационного блока (стрелки):

- Стрелка вверх — движение объекта вверх по координатной сетке;
- Стрелка вниз — движение объекта вниз по координатной сетке;
- Стрелка влево — поворот объекта в левую сторону;
- Стрелка вправо — поворот объекта в правую сторону.
- *Примечание:* Программа аппаратно блокирует некорректные действия пользователя, запрещая совершать мгновенный разворот головы змейки на 180 градусов в противоположную сторону.

- **Панель инструментов (Информационная область):** В верхней части окна располагается закрепленная статическая панель инструментов, информирующая пользователя о текущем состоянии сессии:

- Метрика Total отображает текущее количество набранных очков (съеденных яблок);
- Метрика Speed отображает уровень текущей скорости змейки, рассчитываемый низкоуровневой ассемблерной вставкой на каждые 5 набранных очков.

3. Условия завершения игры и возврат в меню

Игровая сессия продолжается непрерывно до момента фиксации одного из двух критических событий (коллизий), приводящих к поражению (рисунок 6).

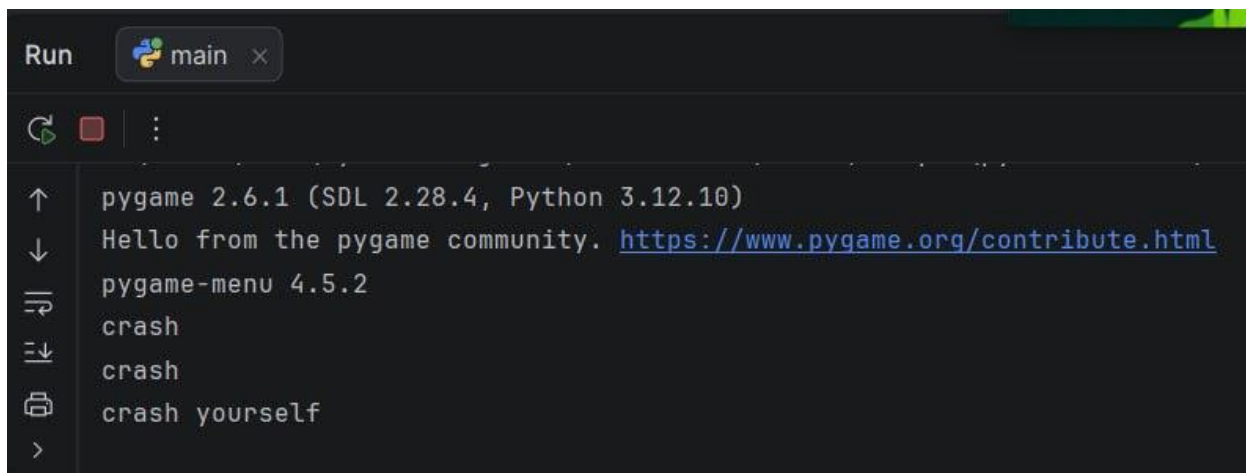


Рисунок 6 – Вывод системных сообщений о коллизиях в консоль отладки

Аварийное завершение сессии происходит в следующих случаях:

1. **Столкновение с внешним ограждением:** Если пользователь вовремя не изменил вектор движения, и голова змейки пересекла крайнюю линию координатного поля (вышла за рамки сетки), фиксируется событие `crash`. Игровой цикл останавливается.

2. **Самопересечение (столкновение с собственным хвостом):** При достижении змейкой большой длины неосторожный маневр может привести к тому, что координата головы совпадет с координатой одного из сегментов её тела. В этом случае фиксируется событие `crash yourself`.

После фиксации любого из этих условий программа автоматически закрывает графический контекст игрового поля, обнуляет параметры сессии и возвращает пользователя на стартовый экран к главному меню (Рисунок X). Пользователь может заново ввести имя, начать новую партию или корректно завершить работу с приложением, нажав кнопку «Выйти».

Заключение

В ходе выполнения данной курсовой работы была успешно спроектирована и разработана интерактивная игровая программа «Змейка» с графическим интерфейсом на языке программирования Python с интеграцией низкоуровневого вычислительного модуля на языке Ассемблер x86-64. Программа реализует полноценный динамический игровой процесс в режиме реального времени, оснащена развитой графической системой на базе библиотеки Pygame, полупрозрачным интерактивным меню пользователя и модулем оптимизации вычислений, который на аппаратном уровне через регистры процессора рассчитывает параметры задержки системного таймера.

В процессе разработки были получены и закреплены следующие практические навыки:

- **Создание многомодульных программ** с разделением исходного кода на изолированные логические компоненты (основной диспетчерский цикл, класс объектов, генерация ресурсов, игровой цикл, графический интерфейс меню);
- **Использование сложных типов данных** и объектно-ориентированных структур: классов (SnakeBlock) для координатного представления ячеек сетки, динамических последовательных списков (list) для хранения сегментов тела змейки, кортежей палитры (tuple) и бинарных буферов оперативной памяти;
- **Работа с графическими подсистемами ЭВМ**, растровыми изображениями заставки (logo.png), шрифтовыми объектами и циклической отрисовкой геометрических примитивов в фиксированном окне отображения;
- **Обработка пользовательского ввода** в реальном времени с валидацией клавиатурных прерываний, отслеживанием фокуса мыши в виджетах и аппаратным блокированием некорректных действий (маневров змейки на 180 градусов);

- **Межязыковое взаимодействие** сред высокого (Python) и низкого (Ассемблер) уровней: динамическое выделение памяти, применение системных вызовов Win32 API (VirtualProtect) для конфигурирования прав доступа дескрипторов в обход аппаратной защиты выполнения данных (DEP), согласование соглашений о вызовах x64 (calling conventions) посредством интерфейса модуля ctypes;
- **Отладка и комплексное тестирование** программного обеспечения с использованием точек останова, пошаговой трассировки алгоритмов коллизий и анализа состояния переменных в среде разработки JetBrains PyCharm.

Разработанное приложение полностью соответствует исходным обязательным техническим требованиям: поддерживает многомодульность, функционирует в графическом режиме видеосистемы с поддержкой мыши и клавиатуры, содержит интерактивное меню настройки сессии, графическую заставку и сложные динамические структуры данных.

В перспективе программа может быть дополнена системой персистентного хранения таблицы рекордов во внешнем файле, добавлением различных типов препятствий на игровом поле или реализацией звукового сопровождения игровых событий. Однако и в текущем виде она служит полноценным примером практического применения знаний и инженерных навыков, полученных в ходе изучения профильных дисциплин.

Список используемых источников

1. Россум, Г. Язык программирования Python / Г. Россум, Ф. Л. Дж. Дрейк, Д. П. Откидач. – Текст : электронный // Официальный сайт Python : [сайт]. – URL: <https://docs.python.org/3/> (дата обращения: 20.05.2026).
2. Документация по библиотеке Pygame : [сайт]. – URL: <https://www.pygame.org/docs/> (дата обращения: 21.05.2026). – Текст : электронный.
3. Юров, В. И. Assembler : учебник для вузов / В. И. Юров. – 2-е изд. – Санкт-Петербург : Питер, 2011. – 640 с. – Текст : непосредственный.
4. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – Санкт-Петербург : Питер, 2021. – 816 с. – Текст : непосредственный.
5. Pygame-menu Documentation. Interface design for Pygame application. – Текст : электронный // GitHub Pages : [сайт]. – URL: <https://pygame-menu.readthedocs.io/> (дата обращения: 25.05.2026).
6. ctypes — A foreign function library for Python. – Текст : электронный // Python Software Foundation Docs : [сайт]. – URL: <https://docs.python.org/3/library/ctypes.html> (дата обращения: 22.05.2026).
7. Microsoft Developer Network (MSDN). VirtualProtect function (memoryapi.h). – Текст : электронный // Microsoft Learn : [сайт portal]. – URL: <https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect> (дата обращения: 24.05.2026).
8. Лутц, М. Изучаем Python : в 2 т. Т. 1 / М. Лутц ; перевод с английского А. Киселева. – 5-е изд. – Санкт-Петербург : Диалектика, 2020. – 832 с. – Текст : непосредственный.
9. Аблязов, Р. З. Программирование на ассемблере x86-64 / Р. З. Аблязов. – Москва : ДМК Пресс, 2021. – 304 с. – Текст : непосредственный.

Приложение А.

Листинг программы

```
import pygame
import sys
import random
import pygame_menu
import ctypes
pygame.init()

bg_image = pygame.image.load('logo.png')
SIZE_BLOCK = 20
FRAME_COLOR = (0, 128, 0)
GREEN = (34, 139, 34)
LIME = (50, 205, 50)
RED = (224, 0, 0)
WHITE = (255, 255, 255)
HEADER_COLOR = (0, 128, 0)
SNAKE_COLOR = (0, 0, 255)
COUNT_BLOCKS = 20
HEADER_MARGIN = 70
MARGIN = 1

size = [SIZE_BLOCK * COUNT_BLOCKS + 2 * SIZE_BLOCK +
MARGIN * COUNT_BLOCKS,
SIZE_BLOCK * COUNT_BLOCKS + 2 * SIZE_BLOCK + MARGIN *
COUNT_BLOCKS + HEADER_MARGIN]

screen = pygame.display.set_mode(size)
pygame.display.set_caption('Змейка')
timer = pygame.time.Clock()
```

```

courier = pygame.font.SysFont('courier', 36)

class SnakeBlock:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def is_inside(self):
        return 0<=self.x<COUNT_BLOCKS and 0<=self.y<COUNT_BLOCKS

    def __eq__(self, other):
        return isinstance(other, SnakeBlock) and self.x ==
other.x and self.y == other.y

    def draw_block(color,row, column):
        pygame.draw.rect(screen, color, [SIZE_BLOCK + column *
SIZE_BLOCK + MARGIN * (column + 1),
HEADER_MARGIN + SIZE_BLOCK + row * SIZE_BLOCK + MARGIN *
(row + 1),
SIZE_BLOCK, SIZE_BLOCK])

    def start_the_game():

    def get_random_empty_block():
        x = random.randint(0, COUNT_BLOCKS - 1)
        y = random.randint(0, COUNT_BLOCKS - 1)
        empty_block = SnakeBlock(x, y)
        while empty_block in snake_blocks:
            empty_block.x = random.randint(0, COUNT_BLOCKS - 1)
            empty_block.y = random.randint(0, COUNT_BLOCKS - 1)

```

```

return empty_block

asm_bytes =
b"\x89\xc8\xb9\x05\x00\x00\x00\x99\xf7\xf9\xff\xc0\xc3"

dst = ctypes.create_string_buffer(asm_bytes)
buf_addr = ctypes.addressof(dst)
ctypes.windll.kernel32.VirtualProtect.argtypes = [
ctypes.c_void_p,
ctypes.c_size_t,
ctypes.c_ulong,
ctypes.c_void_p
]

PAGE_EXECUTE_READWRITE = 0x40
ctypes.windll.kernel32.VirtualProtect(
buf_addr,
len(asm_bytes),
PAGE_EXECUTE_READWRITE,
ctypes.byref(ctypes.c_ulong())
)

asm_calculate_speed = ctypes.CFUNCTYPE(ctypes.c_int,
ctypes.c_int)(buf_addr)

snake_blocks = [SnakeBlock(9, 8), SnakeBlock(9, 9),
SnakeBlock(9, 10)]
apple = get_random_empty_block()
d_row = buf_row = 0
d_col = buf_col = 1

```

```

total = 0
speed = 1

while True:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            print('exit')
            pygame.quit()
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP and d_col!=0:
                buf_row = -1
                buf_col = 0
            elif event.key == pygame.K_DOWN and d_col!=0:
                buf_row = 1
                buf_col = 0
            elif event.key == pygame.K_LEFT and d_row!=0:
                buf_row = 0
                buf_col = -1
            elif event.key == pygame.K_RIGHT and d_row!=0:
                buf_row = 0
                buf_col = 1

    screen.fill(FRAME_COLOR)
    pygame.draw.rect(screen, HEADER_COLOR, [0,0,size[0],HEADER
_MARGIN])

    text_total = courier.render(f"Total: {total}",0, WHITE)
    text_speed = courier.render(f"Speed: {speed}", 0, WHITE)

```

```

screen.blit(text_total, [SIZE_BLOCK, SIZE_BLOCK])
screen.blit(text_speed, [SIZE_BLOCK+230, SIZE_BLOCK])

for row in range(COUNT_BLOCKS):
    for column in range(COUNT_BLOCKS):
        if (row + column) % 2 == 0:
            color = GREEN
        else:
            color = LIME
        draw_block(color, row, column)

head = snake_blocks[-1]
if not head.is_inside():
    print('crash')
    break
# pygame.quit()
#
sys.exit()

draw_block(RED, apple.x, apple.y)

for block in snake_blocks:
    draw_block(SNAKE_COLOR, block.x, block.y)

pygame.display.flip()

if apple == head:
    total+=1
    speed = asm_calculate_speed(total)
    snake_blocks.append(apple)

```

```

apple = get_random_empty_block()

d_row = buf_row
d_col = buf_col
new_head = SnakeBlock(head.x + d_row, head.y+d_col)

if new_head in snake_blocks:
    print('crash yourself')
    break
    # pygame.quit()
    # sys.exit()

snake_blocks.append(new_head)
snake_blocks.pop(0)

timer.tick(3+speed)

main_theme = pygame_menu.themes.THEME_DARK.copy()
main_theme.background_color = (40,41,43,130)

menu = pygame\_menu.Menu(title='',
width=250, height=300,
theme=main_theme)

menu.add.text_input('Имя:', default='Игрок 1')
menu.add.button('Играть', start_the_game)
menu.add.button('Выйти', pygame_menu.events.EXIT)

while True:

```

```
screen.blit(bg_image, (0,0))

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

    if menu.isenabled():
        menu.draw(screen)
        menu.update(events)

pygame.display.update()
```

Приложение Б.

Вычислительный модуль на языке Ассемблер

В рамках реализации межязыкового интерфейса разработан изолированный низкоуровневый вычислительный модуль, представленный в виде последовательности машинных инструкций (байт-кода) архитектуры x86-64. Данный модуль демонстрирует возможность интеграции компилируемого машинного кода с интерпретируемой средой высокого уровня Python без использования сторонних динамических библиотек (.dll).

Вычислительный модуль решает задачу динамического пересчета аппаратной задержки игрового таймера на основе текущего прогресса игрока по следующему алгоритму:

1. В качестве входного параметра функция принимает текущее значение общего счета игрока (total), которое передается через регистр процессора RCX (в соответствии с соглашением о вызовах Microsoft x64 calling convention).
2. Значение очков переносится в регистр EAX инструкцией `mov eax, ecx` (в коде: `\x89\xс8`).
3. Задается делитель (значение 5), помещаемый в регистр ECX инструкцией `mov ecx, 5` (в коде: `\xb9\x05\x00\x00\x00`).
4. Инструкция `cdq` (`\x99`) выполняет знаковое расширение регистра EAX до пары EDX:EAX для корректного проведения операции деления.
5. Инструкция `idiv ecx` (`\xf7\xf9`) производит целочисленное деление накопленных очков на 5. Результат деления (количество полных пятачков съеденных яблок) сохраняется в регистре EAX.
6. Инструкция `inc eax` (`\xff\xс0`) инкрементирует полученный результат на единицу, формируя итоговый коэффициент увеличения скорости (speed).

7. Инструкция `ret (\xc3)` возвращает управление в вызывающий Python-скрипт, передавая итоговое значение шага скорости обратно через регистр EAX.

Для интеграции данного модуля в оперативную память средствами встроенной библиотеки `ctypes` выполняется предварительная конфигурация сегментов данных и вызов системной функции ядра `VirtualProtect`, которая перераспределяет права страницы памяти, заменяя стандартные ограничения `PAGE_READWRITE` на исполняемый флаг `PAGE_EXECUTE_READWRITE` (код защиты `0x40`). Это позволяет обойти аппаратную политику предотвращения выполнения данных (DEP) операционной системы Windows и безопасно выполнить машинный код процессором ЭВМ (рисунок Б1).

```
asm_bytes = b"\x89\xc8\xb9\x05\x00\x00\x00\x99\xf7\xf9\xff\xc0\xc3"

dst = ctypes.create_string_buffer(asm_bytes)
buf_addr = ctypes.addressof(dst)
ctypes.windll.kernel32.VirtualProtect.argtypes = [
    ctypes.c_void_p,
    ctypes.c_size_t,
    ctypes.c_ulong,
    ctypes.c_void_p
]

PAGE_EXECUTE_READWRITE = 0x40
ctypes.windll.kernel32.VirtualProtect(
    buf_addr,
    len(asm_bytes),
    PAGE_EXECUTE_READWRITE,
    ctypes.byref(ctypes.c_ulong())
)

asm_calculate_speed = ctypes.CFUNCTYPE(ctypes.c_int, *argtypes: ctypes.c_int)(buf_addr)
```

Рисунок Б1 - Программная реализация внедрения ассемблерного байт-кода в среду Python

Приложение В.

Результаты тестирования программного комплекса

Для подтверждения надежности, отказоустойчивости и соответствия разработанного приложения техническому заданию было проведено комплексное функциональное тестирование. В ходе тестов проверялась корректность обработки координатных коллизий, стабильность межязыкового интерфейса и поведение графической подсистемы при различных сценариях действий пользователя. Результаты проверок сведены в таблицу В1.

Таблица В1 — Протокол функционального тестирования игрового приложения

№ теста	Описание проверяемой ситуации (тест-кейс)	Ожидаемый результат программы	Фактический результат	Статус
1	Запуск исполняемого файла <code>main.py</code>	Создание окна, отрисовка заставки <code>logo.png</code> и инициализация виджетов меню.	Окно создано, меню активно, заставка отображается корректно.	Успешно
2	Ввод алфавитно-цифровых данных в поле «Имя:»	Символы корректно отображаются в поле ввода, строка сохраняется в памяти.	Имя успешно вводится с клавиатуры и фиксируется системой.	Успешно
3	Клик левой кнопкой мыши по кнопке «Выйти»	Закрытие графического окна, освобождение ресурсов, завершение процесса.	Программа корректно закрывается без системных ошибок.	Успешно

Продолжение таблицы В1

4	Клик левой кнопкой мыши по кнопке «Играть»	Скрытие меню, запуск таймера, инициализация змейки из 3 блоков и яблока.	Переход в игровой режим выполнен, объекты отрисованы.	Успешно
5	Нажатие клавиш со стрелками на клавиатуре	Изменение вектора движения змейки в пространстве.	Змейка мгновенно реагирует на нажатия клавиш-стрелок.	Успешно
6	Попытка резкого разворота змейки на 180 градусов (в саму себя)	Блокировка команды, змейка продолжает двигаться в прежнем направлении.	Разворот заблокирован, аварийного самопересечения не произошло.	Успешно
7	Поглощение головой змейки объекта «Яблоко»	Увеличение длины хвоста, инкремент очков <code>total</code> , регенерация яблока.	Счет увеличился, длина выросла, яблоко появилось в новом месте.	Успешно
8	Набор каждые 5 очков в процессе игровой сессии	Передача параметра в Ассемблер, увеличение скорости <code>speed</code> и тактов таймера.	Программа плавно ускоряет игровой цикл через регистровые вычисления.	Успешно
9	Столкновение головы змейки с внешними границами сетки	Остановка цикла, вывод сообщения <code>crash</code> в консоль, возврат в главное меню.	Игровое окно закрылось, пользователь успешно вернулся к экрану меню.	Успешно
10	Столкновение головы змейки с сегментом собственного тела	Остановка цикла, вывод сообщения <code>crash yourself</code> , возврат в меню.	Коллизия зафиксирована, игра прервалась, осуществлен возврат в меню.	Успешно