**Российская федерация**

**Ханты-Мансийский автономный округ - Югра**

**Департамент образования и науки**

**Сургутский государственный университет ХМАО**

Инженерно-физический факультет

Кафедра Автоматики и компьютерных систем

**Пояснительная записка**

к курсовому проекту

по дисциплине структурное программирование

Выполнил: Гущина М.А.

студент группы 609-31

Проверил: Гришмановский П. В.

доцент кафедры автоматики и компьютерных систем

**Сургут**

**2024 год**

**Задание**

Разработать игру «Змейка».

**Аннотация**

Тема курсового проекта: игра «Змейка»

Выполнила: Гущина М.А.

В курсовом проекте представлена разработка программного обеспечения игры «Змейка», предназначенного для управления движущиеся змейки, с целью собрать еду. Работа состоит из введения, анализа, проектирования, кодирования, тестирования, сопровождения и заключения.

Во введении описаны задачи, цель и актуальность разработки игры «Змейка».

В разделе анализ проведен обзор существующих игр, выполнен сравнительный анализ их функциональности и удобства использования, а также определены требования к разрабатываемому ПО.

В разделе проектирование получены проектные решения, на основе которых осуществлена реализация продукта.

В разделе кодирование описан код программного обеспечения, написанный в соответствии с проектом, а также используемые библиотеки и функции.

В разделе тестирования описаны выбранные методы и стратегии тестирования, разработанные тестовые сценарии, проведенное тестирование и полученные результаты.

В разделе сопровождение разработано руководство пользователя.

В заключении подведены итоги проекта, оценено достижение поставленных целей и задач.

**Содержание**

[Введение 4](#_Toc170172807)

[Анализ 5](#_Toc170172808)

[Проектирование 9](#_Toc170172809)

[Кодирование и отладка 16](#_Toc170172810)

[Тестирование 25](#_Toc170172811)

[Сопровождение 30](#_Toc170172812)

[Заключение 31](#_Toc170172813)

[Список использованной литературы 32](#_Toc170172814)

[Приложения 33](#_Toc170172815)

# Введение

Игра «Змейка» - это классическая компьютерная игра, в которой игрок управляет «головой» растущей линии (змеи) и не должен позволять ей столкнуться с препятствиями, в том числе со своим «хвостом». Первые подобные игры возникли в 1970-х годах [1]. На данный момент, «Змейка» стала одной из самых популярных игр во всем мире, перейдя на различные платформы, от консолей до мобильных устройств.

Актуальность игры «Змейка» заключается в том, что она до сих пор остается значительной и популярной игрой среди пользователей всех возрастов.

Целью данной работы является разработка игры «Змейка», которая позволит пользователю управлять змеей, собирать еду и избегать столкновений с краями поля и самой собой.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ существующих версий игры.
2. Разработать структуру и алгоритмы игры.
3. Реализовать консольный интерфейс игры.
4. Провести тестирование и отладку игры.
5. Разработать документацию на программный продукт.

Ожидаемым результатом данной работы является разработанная игра «Змейка», которая будет соответствовать требованиям задания и предоставлять пользователю возможность играть в данную игру.

# Анализ

Для разработки игры «Змейка» необходимо провести глубокий анализ существующих версий игры и выделить ее ключевые элементы и функциональные особенности. Это позволит определить структуру игры, необходимые алгоритмы и данные.



Рисунок 2.1. Классическая игра «Змейка».

Классическая игра «Змейка» (рис. 2.1.) обладает рядом плюсов, которые делают ее привлекательной для многих игроков. Во-первых, она доступна на любой платформе, где есть консоль, что делает ее удобной для игры. Во-вторых, она приносит ностальгические воспоминания о детстве, что часто ценится игроками. В-третьих, классическая «Змейка» отличается минималистичным дизайном и фокусом на игровом процессе, без лишних отвлекающих элементов.

Однако у этой игры также есть свои минусы. Некоторые игроки могут посчитать отсутствие качественной графики недостатком, так как игра выглядит достаточно просто и устаревшим. Также игра может показаться недостаточно сложной для опытных игроков и не предлагать достаточно интерактивности.

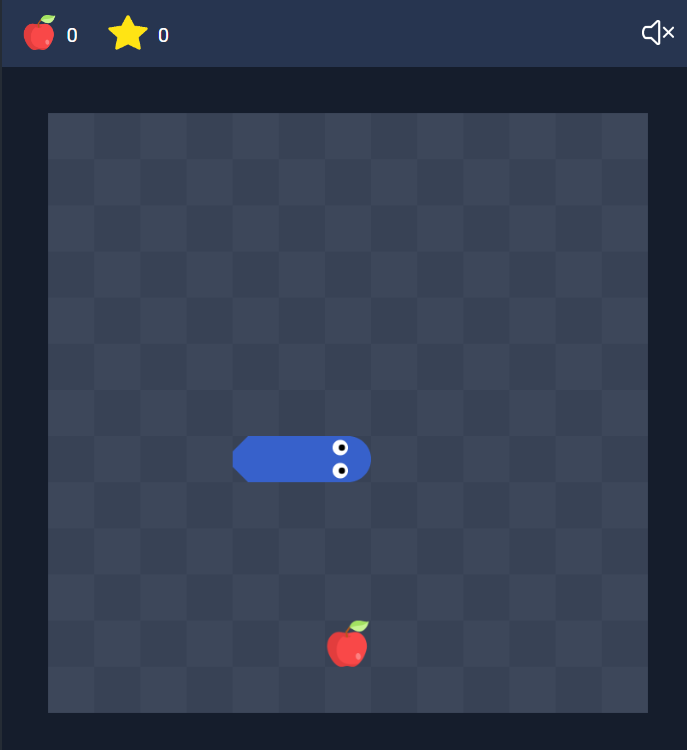


Рисунок 2.2. Современная игра «Змейка».

Современные версии игры «Змейка» (рис. 2.2.). Главными преимуществами являются улучшенная графика, разнообразный геймплей с новыми механиками и уровнями, а также повышенная сложность, требующая от игрока большей сосредоточенности. Кроме того, современные версии обычно более интерактивны, с добавлением новых элементов взаимодействия с игроком.

Однако у современных версий «Змейка» также есть свои минусы. Например, некоторые игры могут требовать специфического оборудования или операционной системы, что может создать определенные проблемы. Также некоторые игроки могут не оценить отсутствие ностальгии, которую приносит классическая версия игры, а также потерю минимализма в игровом процессе из-за перегруженности дополнительными элементами. [2] [3]

Сравнив «классическую» и современную версию игр, можно сказать, что каждая из них по-своему уникальна и имеют свои отличия, но, что до сих пор остается неизменным – это простые привала, интуитивно понятный интерфейс и захватывающий игровой процесс.

Отсюда, можно выделить основные элементы игры:

* Игровое поле – пространство, ограниченное стенами, в котором происходит игра.
* Змея – главный объект игры, который игрок контролирует.
* Еда – объекты, которые змея должна собирать. Обычно это яблоки, которые появляются случайным образом на игровом поле.
* Очки или баллы, которые игрок зарабатывает за каждое съеденную еду.
* Скорость – движение змеи, которое может увеличиваться с определенного количества очков.

Значит, для разработки программы, необходимо выполнить следующие задачи:

* Создание и отображение двумерного игрового поля размером n на m ячеек.
* Реализация управления змеей с помощью определённых игровых клавиш.
* Генерация случайных координат еды на игровом поле.
* Проверка столкновения змеи с краями поля и собственным телом.
* Ведение счета съеденной еды и увеличение скорости.
* Отображение игрового поля на консоли.

Кроме того, необходимо учитывать следующие требования:

* Программа должна быть удобной в использовании, т.е. иметь простой и интуитивно понятный интерфейс для игрока.
* Программа должна быть относительно быстрой и эффективной.

Для успешной разработки программы необходимо разделить реализацию на отдельные функции, каждая из которых будет отвечать за определенный функционал игры. Такой подход упростит разработку, отладку и поддержку программы в дальнейшем.

# Проектирование

Проектирование игры «Змейка» включает в себя разработку модульной структуры, алгоритмов и структур данных, а также интерфейса.

Цель проектирования – создать четкую архитектуру игры, которая позволит эффективно реализовать все необходимые функции и обеспечит гибкость и удобство в дальнейшей разработке.

Для формирования структуры ПО можно разделить на несколько модулей:

* Игровой цикл – отвечает за запуск и управление игрой, т.е. создает поле, змею, яблоко и следит за состоянием игры: проверяет столкновения, отображает информации о счете и уровне и учитывает скоростью игры.
* Поле – представляет собой двумерный массив, который хранит информацию о состоянии каждой клетки, т.е., какая из клеток будет занимать змея, яблоко или пустое пространство.
* Змея – это список координат, где каждая координата хранит информацию о длине, позиции части тела и направлении движения змеи.
* Яблоко – содержит координаты, которые отвечают за случайную генерацию на поле.

Основные алгоритмы (блок-схемы алгоритмов):

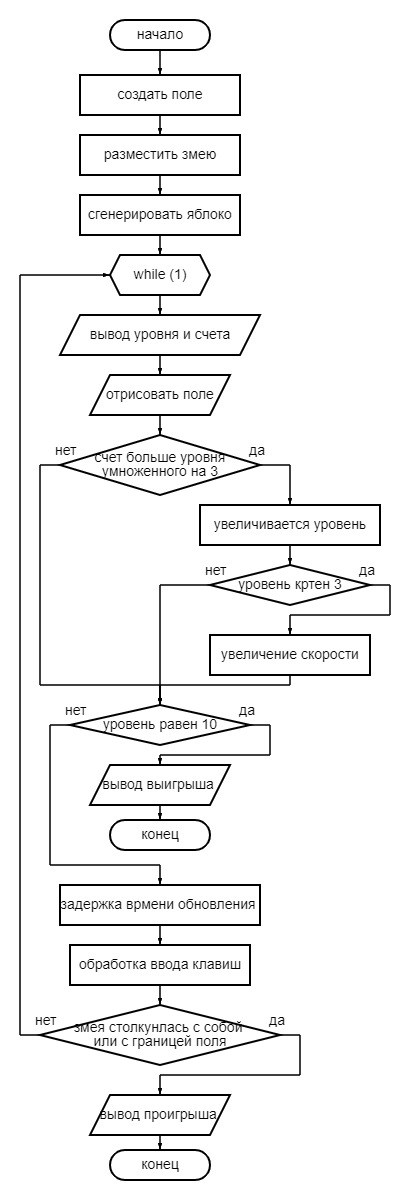


Рисунок 3.1. Алгоритм игрового цикла.

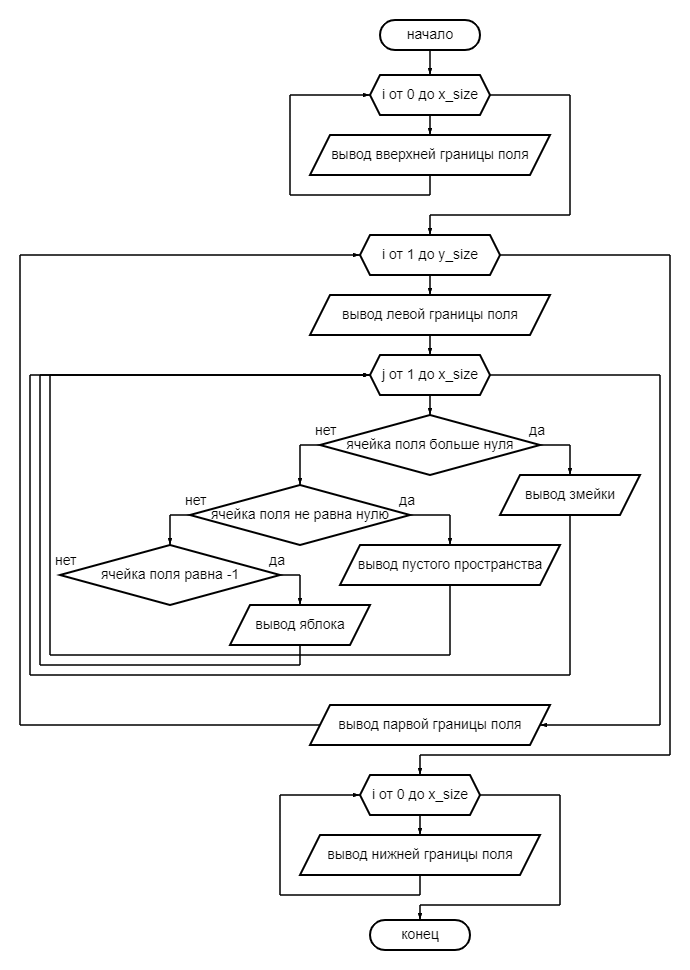


Рисунок 3.2. Алгоритм поля.

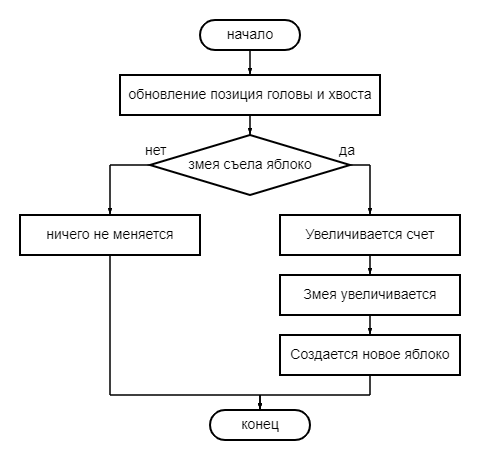


Рисунок 3.3. Алгоритм змеи.

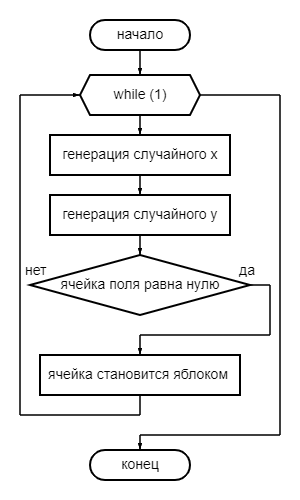


Рисунок 3.4. Алгоритм яблока.

Теперь разработаем интерфейс игры.

Первым делом будет показано меню, где пользователь может запустить игру, просмотреть правила или выйти с программы (рис 3.5).

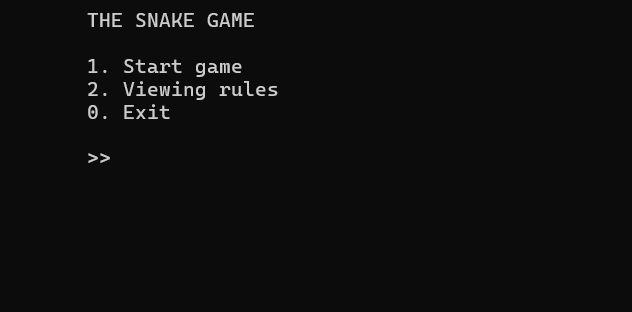


Рисунок 3.5. Меню программы.

При выборе операции «Старт игры» на экран будет выводится информация о текущем состоянии игры, а именно игровое поле, змейка, яблоко, счет и уровень (рис 3.6).

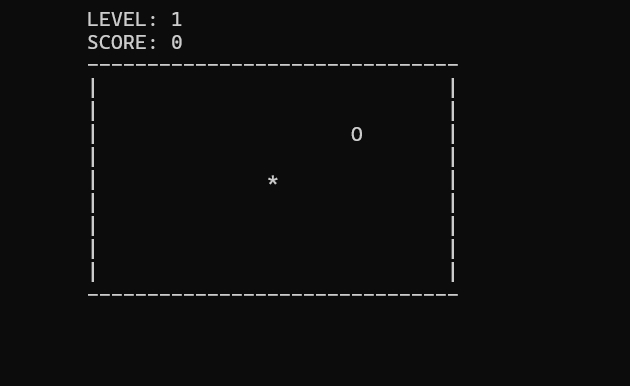


Рисунок 3.6. Начало игры.

При проигрыше программа выводит текст «GAME OVER» и предоставляет возможность вернуться в меню (рис 3.7).



Рисунок 3.7. Проигрыш.

При выигрыше программа выводит текст «YOU WIN!» и тоже возвращает пользователя в меню (рис 3.8).



Рисунок 3.8. Выигрыш.

Чтобы поставить игру на паузу, необходимо нажать клавишу «p». На экране отобразиться сообщение о паузе и информация о дальнейшем развитии игры, т.е. пользователь может продолжить игру, нажав на клавишу «c», или выйти, нажав клавишу «e» (рис 3.9).



Рисунок 3.9. Пауза.

При выборе в меню второй операции «Просмотр правил», на экране выводится соответственно подробное описание привал игры. (рис 3.10).

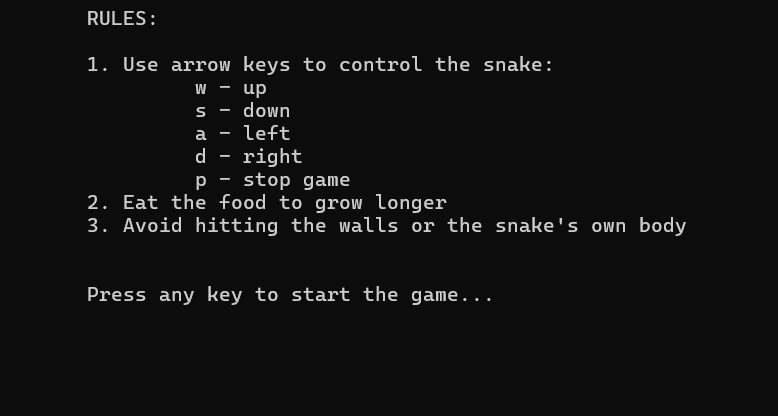


Рисунок 3.10. Правила игры.

При выборе операции «Выход», программа завершает свою работу.

Таким образом, данное проектирование предоставляет общую картину структуры и алгоритмов игры «Змейка». Для дальнейшей разработки необходимо перейти к этапу кодирования.

# Кодирование и отладка

На данном этапе преобразовываем проектные решения в программный код, значит, можно сказать, что целью данного этапа является создание работоспособного программного продукта.

Выбор библиотек:

Библиотека stdio.h используется для ввода и вывода данных в программе. Эта библиотека необходима для взаимодействия с пользователем, вывода информации о игре и отображения игрового поля.

Библиотека stdlib.h используется для работы с случайными числами в программе. В ней содержатся функции, такие как rand(), которая используется для генерации случайных чисел, в нашем случае – это генерация случайных координат для появления яблока в игре.

Библиотека windows.h используется для работы с операционной системой Windows в программе. В ней содержатся функции, такие как system("cls"), которая используется для очистки консоли перед каждой отрисовкой игрового поля.

Библиотека conio.h используется для ввода и вывода данных с клавиатуры в программе. В ней содержатся функции, такие как kbhit(), которая используется для проверки нажатия клавиш на клавиатуре, и \_getch(), которая используется для получения символов с клавиатуры без их отображения на экране.

Библиотека time.h используется для работы с временем в программе. В ней содержится функция clock(), которая используется для получения текущего времени в миллисекундах. Эта библиотека необходима для управления скоростью игры, создания задержки между шагами змеи и обеспечения правильной игровой динамики.

Выбор типов данных:

Тип int используется для хранения целых чисел, таких как координаты на игровом поле, счет игрока, уровень сложности игры или размер поля. Он предоставляет возможность хранить целочисленные значения без дробной части.

Тип char используется для хранения отдельных символов, например, символов, вводимых пользователем с клавиатуры. Этот тип данных позволяет работать с отдельными символами в текстовом формате.

Enum direction используется для представления определенных направлений движения, какие могут быть заданы конкретными значениями, например, UP (вверх), DOWN (вниз), LEFT (влево), RIGHT (вправо). Он упрощает работу с направлениями, делает код более читаемым и обеспечивает безопасное использование различных направлений.

Тип double используется для хранения чисел с плавающей точкой, таких как значение скорости игры. Этот тип данных необходим для управления скоростью игровых процессов, где значения могут изменяться не целочисленно. double предоставляет возможность хранить значения с дробной частью и управлять ими в игровых сценариях.

Описание функций:

1. main() – главный модуль, запускает игру и содержит цикл обработки событий.

int main()

{

int key;

do{

system("cls");

printf("\n\tTHE SNAKE GAME\n\n");

printf("\t1. Start game\n"

"\t2. Viewing rules\n"

"\t0. Exit\n\n");

while (printf("\t>> "), fflush(stdin),

scanf("%d", &key) != 1 || !(key >= 0 || key <= 2))

printf("Unknown command\n");

switch (key){

case 1:

system("cls");

game();

break;

case 2:

system("cls");

print\_rules();

break;

case 0:

return 0;

break;

}

}

while (key);

system("pause");

return 0;

}

Функция main() является точкой входа в программу и управляет запуском всей игры. Она отвечает за отображение меню игры, обрабатывает ввод данных от пользователя, а также инициирует необходимые функции, включая основный игровой цикл, вывод правил игры, с помощью функции print\_rules() и возможность завершить программу. Она обеспечивает главную цель игры – предоставить игроку возможность запустить игру, ознакомиться с правилами и управлять процессом игры.

1. game() – функция, отвечающая за логику игры (обработка ввода, движение змеи, проверка столкновений, управление скоростью и уровнем).

int game()

{

clock\_t start\_time;

int direction = STOP;

int score = 0;

int level = 1;

char input;

double speed = 0.15;

int x\_size = SCREEN\_X;

int y\_size = SCREEN\_Y;

int head\_position\_x = x\_size/2;

int head\_postion\_y = y\_size/2;

int field[SCREEN\_Y][SCREEN\_X] = {0};

field[head\_postion\_y][head\_position\_x] = 1;

spawn\_apple(x\_size, y\_size, field);

while (1)

{

if (score >= level \* 3) {

level++;

if (level % 3 == 0) {

speed \*= 0.8;

}

}

else if (level == 10) {

printf("\n\tYOU WIN!\n");

printf("\n\tPress any key to return to the menu...");

while (1) {

if (kbhit()) {

input = \_getch();

if (input != 'a' && input != 's' && input != 'd' && input != 'w') {

return;

}

}

}

return;

}

system("cls");

printf("\n\tLEVEL: %d", level);

printf("\n\tSCORE: %d \n", score);

print\_frame(x\_size, y\_size, field);

start\_time = clock();

while (((double)(clock() - start\_time)) / CLOCKS\_PER\_SEC < speed) {

if (kbhit()) {

input = \_getch();

}

}

if (input == 'p') {

printf("\n\tThe game is on pause.\n\n"

"\tTo continue, click on 'c'\n"

"\tTo exit, click on 'e'\n");

while (1) {

if (kbhit()) {

input = \_getch();

if (input == 'c') {

break;

}

else if (input = 'e'){

return;

}

}

}

}

if (input == 'a' && direction != RIGHT)

direction = LEFT;

if (input == 'w' && direction != DOWN)

direction = UP;

if (input == 's' && direction != UP)

direction = DOWN;

if (input == 'd' && direction != LEFT)

direction = RIGHT;

if (!snake(direction, &head\_position\_x, &head\_postion\_y, x\_size, y\_size, field, &score) ||

(head\_position\_x == 0 || head\_position\_x == x\_size || head\_postion\_y == 0 || head\_postion\_y == y\_size)){

printf("\n\tGAME OVER\n");

printf("\n\tPress any key to return to the menu...");

while (1) {

if (kbhit()) {

input = \_getch();

if (input != 'a' && input != 's' && input != 'd' && input != 'w') {

return;

}

}

}

}

}

}

Функция game() является основной функцией игры, которая содержит игровой цикл и управляет всеми игровыми процессами. Она создает игровое поле, запускает змею, генерирует яблоко и запускает игровой цикл, в котором обрабатывает движение змеи, столкновения, паузу, повышение уровня и победу. Она обеспечивает последовательность игрового цикла, управляет всеми взаимодействиями и условиями игры, а также создает увлекательный игровой процесс.

1. print\_frame() – функция, отвечающая за отрисовку игрового поля в консоли.

void print\_frame(int x\_size, int y\_size, int field[y\_size][x\_size])

{

int i, j;

printf("\t");

for (i = 0; i < x\_size + 1; i++){

printf("-");

}

printf("\n");

for (i = 1; i < y\_size; i++){

printf("\t|");

for (j = 1; j < x\_size; j++){

if (field[i][j] > 0)

printf("\*");

else if (!field[i][j])

printf(" ");

else if (field[i][j] == -1)

printf("O");

}

printf("|");

printf("\n");

}

printf("\t");

for (i = 0; i < x\_size + 1; i++){

printf("-");

}

printf("\n");

}

void spawn\_apple(int x\_size, int y\_size, int field[y\_size][x\_size])

{

int x, y;

while (1) {

x = rand() % (x\_size - 1) + 1;

y = rand() % (y\_size - 1) + 1;

if (field[y][x] == 0) {

field[y][x] = -1;

return;

}

}

}

Функция print\_frame() отображает игровое поле с змеей, яблоком и границами, что позволяет игроку видеть игровой процесс. Это делает игру более понятной и интересной. Она принимает размер поля x\_size, y\_size и двумерный массив field, содержащий информацию о состоянии каждой ячейки поля.

1. spawn\_apple() – функция, отвечающая за генерацию яблока на случайной клетке.

void spawn\_apple(int x\_size, int y\_size, int field[y\_size][x\_size])

{

int x, y;

while (1) {

x = rand() % (x\_size - 1) + 1;

y = rand() % (y\_size - 1) + 1;

if (field[y][x] == 0) {

field[y][x] = -1;

return;

}

}

}

Функция spawn\_apple() генерирует позицию яблока на игровом поле. Она принимает размер поля x\_size, y\_size и двумерный массив field, чтобы проверить, не находится ли яблоко уже на занятой ячейке. Данная функция обеспечивает постоянный источник пищи для змеи, заставляя ее двигаться и собирая очки, что делает игру более захватывающей.

1. snake() – функция, отвечающая за движение змеи, проверку столкновений с границами и собственным телом, обновление состояния поля.

int snake(int direction, int \*head\_pos\_x, int \*head\_pos\_y, int x\_size, int y\_size, int field[y\_size][x\_size], int \*score)

{

int i, j;

int new\_x = \*head\_pos\_x;

int new\_y = \*head\_pos\_y;

switch (direction){

case UP: new\_y--;

break;

case RIGHT: new\_x++;

break;

case DOWN: new\_y++;

break;

case LEFT: new\_x--;

break;

case STOP:

return 1;

default:

break;

}

if (field[new\_y][new\_x] == -1){

(\*score)++;

field[\*head\_pos\_y][\*head\_pos\_x] = (\*score);

spawn\_apple(x\_size, y\_size, field);

}

else if (field[new\_y][new\_x] > 1){

return 0;

}

else

{

for (i = 0; i < y\_size; i++){

for (j = 0; j < x\_size; j++){

if (field[i][j] > 0){

field[i][j]--;

}

}

}

}

field[new\_y][new\_x] = field[\*head\_pos\_y][\*head\_pos\_x] + 1;

\*head\_pos\_x = new\_x;

\*head\_pos\_y = new\_y;

return 1;

}

Функция snake() управляет движением змеи и имеет конечную цель, т.е. съесть яблоко и увеличить счет, тем самым, игрок может влиять на поведение змеи и наблюдать за последствиями своих решений. Она принимает направление движения direction, текущие координаты головы змеи head\_pos\_x, head\_pos\_y, размер поля x\_size, y\_size, двумерный массив field с информацией о состоянии поля и счет score. Функция реализует основную игровую механику и является ключевой функцией игры.

1. print\_rules() – функция, которая информирует игрока о правилах игры.

void print\_rules() {

printf("\n\tRULES: \n\n");

printf("\t1. Use arrow keys to control the snake:\n"

"\t\t w - up\n"

"\t\t s - down\n"

"\t\t a - left\n"

"\t\t d - right\n"

"\t\t p - stop game\n"

"\t2. Eat the food to grow longer\n"

"\t3. Avoid hitting the walls or the snake's own body\n\n");

printf("\n\tPress any key to start the game...");

\_getch();

}

Данный код представляет собой базовую реализацию игры «Змейка». Важно правильно выбрать типы данных, классы и функции, а также использовать необходимые библиотеки.

# Тестирование

Проведение тестирования является неотъемлемой частью разработки программного обеспечения. Тщательное тестирование поможет выявить и устранить ошибки в работе игры, что позволит создать качественный и увлекательный продукт для пользователей.

1. Тестирование на основе требований.

* Запуск и завершение игры.

Тест 1. Запуск программы.

Ожидаемый результат: программа запускается без ошибок, отображается главное меню с опциями «Старт игры», «Просмотр правил» и «Выход».

Тест 2. Выход из программы.

Ожидаемый результат: программа корректно завершает работу без ошибок и зависаний.

* Игровой процесс.

Тест 3. Управление змеей.

Ожидаемый результат: змея движется в соответствующем направлении (вверх, влево, вниз, вправо), без задержек и рывков.

Тест 4. Сбор яблок и рост змеи.

Ожидаемый результат: длина змеи увеличивается на один сегмент; счет увеличивается на 1 очко; новое яблоко появляется в случайной свободной ячейке игрового поля.

Тест 5. Повышение уровня сложности.

Ожидаемый результат: уровень сложности повышается; скорость игры увеличивается.

Тест 6. Пауза в игре.

Ожидаемый результат: игра ставится на паузу; отображается сообщение о паузе и инструкции по возобновлению игры или выходу в меню.

Тест 7. Продолжение игры после паузы.

Ожидаемый результат: игра воспроизводится с момента паузы, змея продолжает движение с того же направления.

Тест 8. Выход из игры во время паузы.

Ожидаемый результат: возврат в главное меню игры.

Тест 9. Столкновение со стеной.

Ожидаемый результат: игра завершается; отображается сообщение «GAME OVER».

Тест 10. Столкновение с самой собой.

Ожидаемый результат: игра завершается; отображается сообщение «GAME OVER».

Тест 11. Достижение максимального уровня.

Ожидаемый результат игра завершается; отображается сообщение «YOU WIN!».

1. Тестирование на устойчивость.

* Обработка некорректного ввода.

Тест 12. Ввод неверных значений в меню.

Ожидаемый результат: игра не завершается с ошибкой, а выводит сообщение об ошибке и предлагает повторно ввести значение.

Тест 13. Нажатие клавиш, не задействованных в управлении во время игры.

Ожидаемый результат: игра игнорирует нажатия некорректных клавиш, продолжая работать в штатном режиме.

* Производительность.

Тест 14. Оценка производительности при максимальной длине змеи.

Ожидаемый результат: игра не должна зависать при максимальной длине змеи, а скорость отрисовки и отзывчивость управления должны оставаться на приемлемом уровне.

Проведение тестирования игры «Змейка».

1. Тестирование на основе требований.

* Запуск и завершение игры.

Тест 1. Запуск программы.

Ожидаемый результат: программа запускается без ошибок, отображается главное меню с опциями «Старт игры», «Просмотр правил» и «Выход».

Результат: программа успешно запущена, главное меню отображается корректно. Пройден.

Тест 2. Выход из программы.

Ожидаемый результат: программа корректно завершает работу без ошибок и зависаний.

Результат: программа корректно завершила работу. Пройден.

* Игровой процесс.

Тест 3. Управление змеей.

Ожидаемый результат: змея движется в соответствующем направлении (вверх, влево, вниз, вправо), без задержек и рывков.

Результат: змея успешно управляется при помощи клавиш w, a, s, d. Пройден.

Тест 4. Сбор яблок и рост змеи.

Ожидаемый результат: длина змеи увеличивается на один сегмент; счет увеличивается на 1 очко; новое яблоко появляется в случайной свободной ячейке игрового поля.

Результат: длина змеи увеличилась на один сегмент; счет увеличился на 1 очко; новое яблоко появилось в случайном месте. Пройден.

Тест 5. Повышение уровня сложности.

Ожидаемый результат: уровень сложности повышается; скорость игры увеличивается.

Результат: уровень сложности повышен, скорость игры увеличилась. Пройден.

Тест 6. Пауза в игре.

Ожидаемый результат: игра ставится на паузу; отображается сообщение о паузе и инструкции по возобновлению игры или выходу в меню.

Результат: игра поставлена на паузу, сообщение о паузе и инструкции отображаются. Пройден.

Тест 7. Продолжение игры после паузы.

Ожидаемый результат: игра воспроизводится с момента паузы, змея продолжает движение с того же направления.

Результат: игра поставлена на паузу, сообщение о паузе и инструкции отображаются. Пройден.

Тест 8. Выход из игры во время паузы.

Ожидаемый результат: возврат в главное меню игры.

Результат: осуществлен возврат в главное меню. Пройден.

Тест 9. Столкновение со стеной.

Ожидаемый результат: игра завершается; отображается сообщение «GAME OVER».

Результат: игра окончена, сообщение «GAME OVER» отображается. Пройден.

Тест 10. Столкновение с самой собой.

Ожидаемый результат: игра завершается; отображается сообщение «GAME OVER».

Результат: игра окончена, сообщение «GAME OVER» отображается. Пройден.

Тест 11. Достижение максимального уровня.

Ожидаемый результат игра завершается; отображается сообщение «YOU WIN!».

Результат: игра окончена, сообщение «YOU WIN!» отображается. Пройден.

1. Тестирование на устойчивость.

* Обработка некорректного ввода.

Тест 12. Ввод неверных значений в меню.

Ожидаемый результат: игра не завершается с ошибкой, а выводит сообщение об ошибке и предлагает повторно ввести значение.

Результат: игра выводит сообщение об ошибке и предлагает повторно ввести значение. Пройден.

Тест 13. Нажатие клавиш, не задействованных в управлении во время игры.

Ожидаемый результат: игра игнорирует нажатия некорректных клавиш, продолжая работать в штатном режиме.

Результат: игра игнорирует нажатия неиспользуемых клавиш. Пройден.

* Производительность.

Тест 14. Оценка производительности при максимальной длине змеи.

Ожидаемый результат: игра не должна зависать при максимальной длине змеи, а скорость отрисовки и отзывчивость управления должны оставаться на приемлемом уровне.

Результат: каких-либо задержек в отклике не наблюдается. Пройден.

Все запланированные тесты, как функциональные, так и на устойчивость, пройдены успешно. Критических ошибок в работе игры «Змейка» не обнаружено. Игра демонстрирует стабильную работу, корректно обрабатывает некорректный ввод и граничные условия, а также показывает приемлемый уровень производительности.

# Сопровождение

Минимальные требования (для запуска программы):

* Операционная система: Windows XP
* Процессор: Intel или AMD с частотой 1 ГГц.
* Оперативная память: 512 МБ.
* Свободное место на диске: 1 МБ.

Рекомендуемые требования (для комфортной игры):

* Операционная система: Windows 10.
* Процессор: Intel или AMD с частотой 2 ГГц.
* Оперативная память: 1 ГБ.
* Свободное место на диске: 10 МБ.

Инструкция по установке:

1. Установить исполняемый файл «snake.exe».
2. Программа готова к работе.

Правила эксплуатации:

1. Используйте программу по назначению. Игра «Змейка» предназначена исключительно для развлекательных целей. Не используйте ее для каких-либо действий, так как это может привести к непредсказуемым результатам.
2. Не изменяйте исходный код программы. Внесение изменений в исходный код программы может привести к ошибкам и сбоям.
3. Не запускайте программу от имени администратора, если это не требуется, так как это может нанести потенциальный ущерб системе или данным

Сведения о неисправленных недостатках:

На данный момент не выявлено ни одного существенного недостатка, влияющего на работоспособность игры «Змейка».

Предупреждения о возможном неадекватном функционировании:

* Низкая производительность. На очень слабых компьютерах или при максимально возможной длине змеи игра может работать недостаточно плавно.

# Заключение

В рамках курсового проекта была разработана игровая программа «Змейка». В результате создано консольная игра, которая предоставляет пользователю возможность управлять змейкой и увеличивать ее размер, поедая случайно появляющуюся на игровом поле еду. Разработанная программа представляет собой развлекательное приложение, которое позволяет провести время с удовольствием, наслаждаясь игровым процессом и улучшая свои навыки логики и реакции.

Достоинства полученного решения:

* Благодаря интуитивно понятному интерфейсу, пользователю не требуется тратить время на изучение сложных инструкций.
* Программа может быть запущен на различных операционных системах.
* Программа не требует высокой производительности компьютера и может быть запущена на устаревших системах.

Недостатки:

* Программа работает исключительно в консольном режиме, что может быть неудобно для некоторых пользователей, предпочитающих графический интерфейс.
* Реализация содержит базовый функционал игры «Змейка» без дополнительных возможностей, например, выбор уровня сложности или сохранение рекордов.

Перспективы дальнейшего развития:

* Улучшение графического интерфейса программы, что сделает игру более современной и привлекательной.
* Добавление новых игровых элементов, уровней сложности, таблицы рекордов.

Разработка игры «Змейка» позволила получить практические навыки программирования, работы с интерфейсом консоли и алгоритмами игровой логики. Программа может быть использована как основа для дальнейшего развития и создания более сложных и интересных игр.

# Список использованной литературы

[1]<https://ru.wikipedia.org/wiki/Змейка_(игра)>

[2]<https://fishki.net/3269217-zmejka-dlja-nokia-3310-polnaja-istorija-velichajshej-mobilynoj-igry.html>

[3]<https://pikabu.ru/story/istoriya_snake_kak_igra_dlya_nokia_sozdala_novuyu_yeru_v_mobilnoy_industrii_8830285>

# Приложения

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <conio.h>

#include <time.h>

#define SCREEN\_X 30

#define SCREEN\_Y 10

enum direction

{

STOP,

UP,

RIGHT,

DOWN,

LEFT

};

void print\_frame(int x\_size, int y\_size, int field[y\_size][x\_size])

{

int i, j;

printf("\t");

for (i = 0; i < x\_size + 1; i++){

printf("-");

}

printf("\n");

for (i = 1; i < y\_size; i++){

printf("\t|");

for (j = 1; j < x\_size; j++){

if (field[i][j] > 0)

printf("\*");

else if (!field[i][j])

printf(" ");

else if (field[i][j] == -1)

printf("O");

}

printf("|");

printf("\n");

}

printf("\t");

for (i = 0; i < x\_size + 1; i++){

printf("-");

}

printf("\n");

}

void spawn\_apple(int x\_size, int y\_size, int field[y\_size][x\_size])

{

int x, y;

while (1) {

x = rand() % (x\_size - 1) + 1;

y = rand() % (y\_size - 1) + 1;

if (field[y][x] == 0) {

field[y][x] = -1;

return;

}

}

}

int snake(int direction, int \*head\_pos\_x, int \*head\_pos\_y, int x\_size, int y\_size, int field[y\_size][x\_size], int \*score)

{

int i, j;

int new\_x = \*head\_pos\_x;

int new\_y = \*head\_pos\_y;

switch (direction){

case UP: new\_y--;

break;

case RIGHT: new\_x++;

break;

case DOWN: new\_y++;

break;

case LEFT: new\_x--;

break;

case STOP:

return 1;

default:

break;

}

if (field[new\_y][new\_x] == -1){

(\*score)++;

field[\*head\_pos\_y][\*head\_pos\_x] = (\*score);

spawn\_apple(x\_size, y\_size, field);

}

else if (field[new\_y][new\_x] > 1){

return 0;

}

else

{

for (i = 0; i < y\_size; i++){

for (j = 0; j < x\_size; j++){

if (field[i][j] > 0){

field[i][j]--;

}

}

}

}

field[new\_y][new\_x] = field[\*head\_pos\_y][\*head\_pos\_x] + 1;

\*head\_pos\_x = new\_x;

\*head\_pos\_y = new\_y;

return 1;

}

int game()

{

clock\_t start\_time;

int direction = STOP;

int score = 0;

int level = 1;

char input;

double speed = 0.15;

int x\_size = SCREEN\_X;

int y\_size = SCREEN\_Y;

int head\_position\_x = x\_size/2;

int head\_postion\_y = y\_size/2;

int field[SCREEN\_Y][SCREEN\_X] = {0};

field[head\_postion\_y][head\_position\_x] = 1;

spawn\_apple(x\_size, y\_size, field);

while (1)

{

if (score >= level \* 3) {

level++;

if (level % 3 == 0) {

speed \*= 0.8;

}

}

else if (level == 10) {

printf("\n\tYOU WIN!\n");

printf("\n\tPress any key to return to the menu...");

while (1) {

if (kbhit()) {

input = \_getch();

if (input != 'a' && input != 's' && input != 'd' && input != 'w') {

return;

}

}

}

return;

}

system("cls");

printf("\n\tLEVEL: %d", level);

printf("\n\tSCORE: %d \n", score);

print\_frame(x\_size, y\_size, field);

start\_time = clock();

while (((double)(clock() - start\_time)) / CLOCKS\_PER\_SEC < speed) {

if (kbhit()) {

input = \_getch();

}

}

if (input == 'p') {

printf("\n\tThe game is on pause.\n\n"

"\tTo continue, click on 'c'\n"

"\tTo exit, click on 'e'\n");

while (1) {

if (kbhit()) {

input = \_getch();

if (input == 'c') {

break;

}

else if (input = 'e'){

return;

}

}

}

}

if (input == 'a' && direction != RIGHT)

direction = LEFT;

if (input == 'w' && direction != DOWN)

direction = UP;

if (input == 's' && direction != UP)

direction = DOWN;

if (input == 'd' && direction != LEFT)

direction = RIGHT;

if (!snake(direction, &head\_position\_x, &head\_postion\_y, x\_size, y\_size, field, &score) ||

(head\_position\_x == 0 || head\_position\_x == x\_size || head\_postion\_y == 0 || head\_postion\_y == y\_size)){

printf("\n\tGAME OVER\n");

printf("\n\tPress any key to return to the menu...");

while (1) {

if (kbhit()) {

input = \_getch();

if (input != 'a' && input != 's' && input != 'd' && input != 'w') {

return;

}

}

}

}

}

}

void print\_rules() {

printf("\n\tRULES: \n\n");

printf("\t1. Use arrow keys to control the snake:\n"

"\t\t w - up\n"

"\t\t s - down\n"

"\t\t a - left\n"

"\t\t d - right\n"

"\t\t p - stop game\n"

"\t2. Eat the food to grow longer\n"

"\t3. Avoid hitting the walls or the snake's own body\n\n");

printf("\n\tPress any key to start the game...");

\_getch();

}

int main()

{

int key;

do{

system("cls");

printf("\n\tTHE SNAKE GAME\n\n");

printf("\t1. Start game\n"

"\t2. Viewing rules\n"

"\t0. Exit\n\n");

while (printf("\t>> "), fflush(stdin),

scanf("%d", &key) != 1 || !(key >= 0 || key <= 2))

printf("Unknown command\n");

switch (key){

case 1:

system("cls");

game();

break;

case 2:

system("cls");

print\_rules();

break;

case 0:

return 0;

break;

}

}

while (key);

system("pause");

return 0;

}