**Российская федерация  
Ханты-Мансийский автономный округ – Югра  
Департамент образования и науки  
Сургутский государственный университет ХМАО**

Политехнический институт  
Кафедра Автоматики и компьютерных систем

**Пояснительная записка**  
к курсовому проекту   
по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы 609-31

Гаврилов Е.Е.

Принял: старший преподаватель кафедры АиКС

Назаров Е.В.

Сургут

2025 г.

**Задание**

Реализовать класс «Хеш-таблица».

**Аннотация**

В данной пояснительной записке описаны этапы написания курсового проекта по дисциплине «Алгоритмы и структуры данных» в процессе которого будет написан класс «Хеш-таблица». Записка содержит в себе этапы анализа изучаемой темы, проектирования структур данных и алгоритмов и описание процесса разработки конечного программного продукта.

# **Содержание**

[**Содержание** 4](#_Toc169637886)

[1. Введение 6](#_Toc169637887)

[2. Анализ 7](#_Toc169637888)

[2.1 Анализ предметной области данной задачи 7](#_Toc169637889)

[2.2. Обзор существующих решений 7](#_Toc169637890)

[2.3. Требования к разработанной программе 11](#_Toc169637895)

[3. Проектирование 13](#_Toc169637896)

[3.1 Формальное описание задачи 13](#_Toc169637897)

[3.2 Поведенческая модель программного продукта 13](#_Toc169637898)

[3.3 Представление данных в памяти при их обработке 14](#_Toc169637899)

[3.4 Разработка алгоритмов 14](#_Toc169637900)

[3.5 Разработка пользовательского интерфейса. 17](#_Toc169637910)

[4. Кодирование 19](#_Toc169637911)

[4.1 Определение языка и типов данных. 19](#_Toc169637912)

[4.2 Реализация функций. 19](#_Toc169637913)

[5.Тестирование 29](#_Toc169637914)

[5.1. Планирование тестирования. 29](#_Toc169637915)

[5.2. Разработка тестов. 29](#_Toc169637916)

[5.3. Испытание ПО. 31](#_Toc169637917)

[5.4 Итоги тестирования. 33](#_Toc169637918)

[6. Сопровождение. 34](#_Toc169637919)

[6.1. Требования к программно-аппаратной платформе 34](#_Toc169637920)

[6.2 Инструкция по установке и использованию 34](#_Toc169637921)

[6.3. Сведенья о недостатках программного продукта 34](#_Toc169637922)

[7.Заключение 35](#_Toc169637923)

[**Приложения** 37](#_Toc169637924)

[Приложение 1. Листинг программы 37](#_Toc169637925)

[Приложение 2. Блок схема алгоритма 52](#_Toc169637926)

1. Введение

Хеш-таблица – структура данных, реализующая интерфейс ассоциативного массива, то есть позволяет хранить пары ключ-значение и выполнять три операции: добавление новой пары, удаление и поиск по ключу. Важное свойство хеш-таблиц состоит в том, что все три операции в среднем выполняются за время O(1). При этом не гарантируется, что время выполнения каждой из операций само по себе мало. [ИСТОЧНИК]

1. Анализ
   1. Анализ предметной области

Хеш-функция (англ. hash function от hash — «превращать в фарш», «мешанина»), или функция свёртки — функция, преобразующая массив входных данных произвольного размера в выходную битовую строку определённого (установленного) размера в соответствии с определённым алгоритмом. Преобразование, выполняемое хеш-функцией, называется хешированием. Исходные (входные) данные называются входным массивом, «ключом», «сообщением». [ИСТОЧНИК]

Хеш-таблица является массивом, элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица со списками). Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа.   
Получающееся хеш-значение играет роль индекса в массиве. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива. [ИСТОЧНИК]

Число хранимых элементов, делённое на размер массива (число возможных значений хеш-функции), называется коэффициентом заполнения хеш-таблицы (load factor) и является важным параметром, от которого зависит среднее время выполнения операций. [ИСТОЧНИК]

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией.

Существует несколько способов разрешения коллизий:

1. Метод списков. Каждая ячейка массива является связным списком пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии в таком случае приводят к тому что появляются списки длинной более одного элемента.
2. Метод открытой адресации. В массиве хранятся сами пары ключ-значение. В случае коллизии необходимо проверить ячейки массива в некотором порядке, до тех пор, пока не будет найдена необходимая ячейка.

Хеш-функция, использующаяся для хеш-таблицы должна обладать следующими свойствами:

1. Детерминированность. Один и тот же ключ всегда должен быть преобразован в один и тот же хеш.
2. Равномерность распределения. Индексы должны быть распределены равномерно, чтобы минимизировать количество незаполненных ячеек массива и коллизий. Добиться действительно равномерного распределения на случайных данных нельзя, поэтому хеш-функция должна хотя бы стремиться к равномерному распределению.
   1. Обзор существующих решений
      1. Класс std::unordered\_map в стандартной библиотеке C++

Класс std::unordered\_map – это ассоциативный контейнер, является частью стандартной библиотеки шаблонов (STL) [ИСТОЧНИК] языка C++ (с стандарта C++11). Класс реализует метод списков для разрешения коллизий. Порядок элементов не соблюдается. Дубликаты ключей запрещены. Как и прочие стандартные контейнеры, позволяет работать с любыми типами данных (как ключей, так и значений), поддерживает работу с итераторами[ИСТОЧНИК], позволяет управлять выделяемой памятью с помощью аллокаторов[ИСТОЧНИК]. Размер таблицы меняется автоматически.

* + 1. Класс HashMap в стандартной библиотеке Java

Класс HashMap реализует интерфейс Map на основе хеш-таблицы. Для разрешения коллизий использует метод списков, однако вместо списков использует бинарные деревья (начиная с Java8) [ИСТОЧНИК]. Порядок элементов не соблюдается. Дубликаты ключей запрещены. Позволяет работать с любыми типами данных (ключей и значений). Позволяет задавать изначальную емкость таблицы и коэффициент загрузки, начиная с которого таблица будет перестроена. Как и все стандартные коллекции, поддерживает работу с итераторами. [ИСТОЧНИК]

## 3. Проектирование

## 3.1 Формальное описание задачи

Игра тетрис состоит из двух ключевых объектов – поле и активная фигура. Поле – это массив 10 на 20, который хранит информацию о неподвижных блоках и само по себе не изменяется. Фигура – это массив 4 на 4, но она существует «вне» поля. Поэтому для описания необходимо ввести понятия «глобальных» координат поля и «локальных» координат в рамках активной фигуры. Для описания фигуры будут использованы координаты ее левого верхнего угла, в системе глобальных координат поля. Для перевода локальных координат в глобальные, достаточно к локальным координатам прибавить координаты фигуры. Во время игры фигура будет перемещаться по полю и вращаться. При этом будет создаваться новая «черновая» фигура с измененным строением или измененными координатами. После этого необходимо проверить, не накладываются блоки фигуры на блоки, находящиеся на поле, если нет, старой фигуре присваиваются свойства «черновой». Если под каким-либо блоком фигуры находится блок на поле или нижняя граница поля, то есть фигура не может двигаться вниз, ее блоки из локальных координат переносятся в глобальные на поле и появляется новая фигура под управлением пользователя. Каждый раз, когда фигура переносится на поле, его необходимо проверить снизу вверх, на наличие полностью заполненных горизонтальных линий поля. Если такие линии найдены, их необходимо удалить, сместив все блоки поля вниз на одну клетку. Если фигура не может двигаться вниз, а ее координаты y меньше нуля, то есть, она находится вне поля, игра завершается.

## 3.2 Поведенческая модель программного продукта

При запуске программы пользователь начинает взаимодействовать с меню, в котором может выбрать из следующих пунктов: запустить игру, посмотреть управление, посмотреть таблицу лидеров, выйти из программы.   
При выборе пункта просмотра управления, программа выведет сообщение о том, с помощью каких клавиш происходит управления игрой.  
При выборе пункта просмотра таблицы лидеров, программа прочтет файл с записями о рекордах, или, если такого файла нет, или его структура не соответствует шаблону, создаст новый файл, согласно шаблону и после прочитает, и выведет список рекордных записей.  
Во время игры программа отрисовывает поле игры и дополнительную информацию (счет и следующую фигуру) в консоли. Пользователь взаимодействует с игрой при помощи пяти клавиш: для паузы, движения фигуры вправо, влево, вниз и вращения фигуры. После завершения игры программа выводит сообщение о том, что игра окончена и показывает финальный счет. Если этот счет больше чем один из рекордных показателей, программа сообщит о новом рекорде и предложить ввести имя из 5 символов. Для ввода допускаются цифры, прописные и строчные буквы латинского алфавита, а также несколько знаков препинания и математических символов. Вводимые символы вне этого списка будут проигнорированы.

### 3.3 Представление данных в памяти при их обработке

Размер игрового поля – 10 на 20 клеток, поэтому в памяти оно представляет собой массив из 200 целых чисел.

Для описания подвижной фигуры используется структура, хранящая данные о положении на поле (с помощью координат x и y левого верхнего угла фигуры), цвете, типе фигуры, а также хранящая поклеточно саму фигуру в двумерном массиве 4 на 4 с помощью единиц и нулей. Далее для обозначения такого двумерного массива целых чисел будет использован термин «строение фигуры».  
Например, строение фигуры «т» (см. рис. 2.3) будет следующее:

[[0,1,0,0],  
 [1,1,1,0],  
 [0,0,0,0],  
 [0,0,0,0]]

Для хранения данных о рекордных записях во время действия программы используется структура содержащая количество очков – целое число и имя – массив из 6 символов.

При сохранении файла с записями рекордов записываются 5 пар: 5 символов и число. Каждая пара записывается с новой строки.

### 3.4 Разработка алгоритмов

### 3.4.1 Создание фигуры

Для правильного создания фигуры необходимо случайным образом определить один из пяти возможных цветов и один из семи возможных типов фигуры. В зависимости от типа присвоить строение фигуры.

### 3.4.2 Вращение фигуры

Вращение фигуры зависит от ее типа. Прямую линию необходимо зеркально отразить, поменяв координаты блоков (x;y) на (y;x). Блок 2 на 2 вращать нет необходимости. Остальные фигуры вращаются по часовой стрелке, координаты блоков фигуры (x;y) меняются на (y-2; x);

3.4.3 Вывод в консоль одного кадра (см рис. 3.1)

В цикле пройти все клетки поля, если клетка заполнена или если этой клетке соответствует заполненная клетка фигуры, то вывести открытую и закрытую квадратные скобки через пробел, иначе вывести пробел, точку и пробел. На пятой строке, если включена пауза, после вывода строки поля, вывести сообщение, что пауза включена. На 7 строке таким же образом вывести счет. На строке 10 вывести слово “next”. На строках 11-14 вывести следующую фигуру.



Рис 3.1 Главное меню

# 3.4.4 Проверка возможности движения вниз, в сторону или поворота.

Для проверки возможности движения вниз, в сторону или поворота необходимо проверить глобальные координаты каждого блока фигуры. Если координаты меньше нуля или больше размеров поля, или если по таким координатам (x;y) на уже есть блок, значит движение туда запрещено.

## 3.4.5 Наложение фигуры на поле.

В цикле перенести каждый блок фигуры на поле, преобразуя локальные координаты в глобальные.

#### 3.4.6 Проверка поля на заполненные горизонтальные линии.

В цикле пройтись по всем строкам поля снизу вверх, если в стоке обнаружена пустая клетка, перейти к следующей строке. Иначе сместить все строки на одну клетку вниз и проверить строку еще раз. Для каждой найденной заполненной строки, увеличивать количество возвращаемых очков в 2 раза и прибавлять 100.

3.4.7 Алгоритм игры (прил. 2)

Вначале создается три фигуры: активная, черновая и следующая. Черновая фигура идентичная активной. Вся игра представляет собой бесконечный цикл. В начале каждой итерации цикла определяется время начала этой итерации (в конце итерации фигура сместится на 1 блок вниз, если сможет). В каждой такой итерации определяется время начала кадра и 1 раз печатается кадр. Пока системное время не превышает время начала кадра на фиксированное время (0.03 секунды), считывается значение нажатой клавиши. Так, в конце этого цикла, будет записан сигнал последней нажатой клавиши. Если была нажата клавиша паузы, логическое значение, показывающая, активна ли пауза, меняется на противоположный. Если пауза неактивна, происходит проверка прочих клавиш управления. Если нажата клавиша поворота, черновая фигура поворачивается, если нажаты клавиши движения, координаты черновой фигуры соответственно меняются. После этого проходит проверка возможности движения вниз, в сторону или поворота. Если проверка успешна пройдена, активной фигуре присваиваются свойства черновой. Этот цикл совершается, пока время, прошедшее с время начала итерации, не превысит время падения блока на 1 клетку (по умолчанию 0.25 секунд). Если активна пауза цикл начнет новую итерацию досрочно. Если нет, черновая фигура опускается на 1 клетку вниз, если проверка возможности движения вниз успешно пройдена, активной фигуре присваиваются свойства черновой. В ином случае, если координаты фигуры меньше нуля (фигура вне поля) игра завершается, возвращая количество набранных очков. Иначе, фигура накладывается на поле, активной фигуре присваиваются свойства следующей, следующая фигура создается заново. Количество очков увеличивается на значение, полученное алгоритмом проверки поля на заполненные горизонтальные линии. Время падения блока на 1 клетку становится равно (25 - (количество очков / 1000)) / 100.0).

# 3.4.8 Сохранение записей рекордов.

В файл записываются 5 пар значений строка из 5 символов и целое число через пробел. Каждая пара записывается с новой строки.

# 3.4.9 Загрузка записей рекордов.

Данные всегда сохраняются и считываются из одного файла с одним, заранее установленным названием, у пользователя нет возможности его изменить. Если файл с заданным именем не существует, он создается и в него записывается пустых 5 строк «EMPTY 0». Если файл существует, считываются 5 пар значений. Если ввод неверный или счет меньше нуля, запись считается некорректной и записывается в массив как «пустая» запись «EMPRY 0». После считывания, массив сортируется по убыванию количества очков в записи

## 3.4.10 Определение и запись нового рекорда.

Для определения нового рекорда, необходимо пройтись по массиву с записями, отсортированному по убыванию количества очков. Если найдена запись, в которой количество очков меньше новой, нужно сообщить об этом пользователю и попросить ввести имя. Имя из 5 букв должно вводиться буфер. Записываться будут только разрешенные символы с номерами от 48 до 126 по ASCII таблице. После этого все последующие элементы массива сдвигаются на одну позицию назад. В поля текущего элемента записываются значение из буфера и новое количество очков. После этого происходит сохранение массива записей в файл.

### 3.5 Разработка пользовательского интерфейса.

Программа имеет консольный ввод – вывод. Взаимодействие с пользователем происходит через меню. Выбор пунктов происходит через ввод числа, соответствующего пункту меню. Пункты меню: начать игру, управление, таблица лидеров, выход (рис 3.2).



Рис 3.2 Главное меню

При запуске игры открывается поле игры, информация о следующей фигуре и количестве набранных очков, как показано на рис 3.1.

Рис 3.3 Интерфейс игры

При выборе пункта «таблица лидеров», программа покажет список из 5 пар строка, число, как показано на рис 3.3.

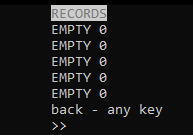


Рис 3.3 Таблица лидеров

При выборе пункта «управление», программа покажет схему управления, как показано на (рис 3.4).

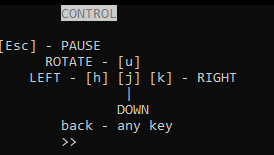


Рис 3.4 Управление

### 4. Кодирование

### 4.1 Определение языка и типов данных.

Программа реализована на языке C стандарта C90 с использованием стандартных библиотек языка, библиотеки conio.h и windows.h для взаимодействия с вводом-выводом системы.  
Для хранения информации используется только статическая память, поскольку все сущности всегда имеют одинаковый объем и изменения размера памяти не требуется.

С помощью директивы препроцессора #define определены вертикальные и горизонтальные размеры игрового поля (20 и 10 соответственно), размер таблицы лидеров (5), путь сохранения таблицы лидеров ("score\_list.txt") и режим консоли для правильного отображения цвета в консоли.

Для работы с цветами определено перечисление где 5 цветам (красному, зеленому, желтому, синему и фиолетовому) присвоены значения от 1 до 5.

Для работы с фигурами определена структура figure, содержащая 4 поля типа char для записи позиции по x и y, обозначения цвета и типа фигур, а также массив char размера 4 на 4 для сохранения самой фигуры. Тип char выбран, так как все поля целочисленные, а их значения могут быть от 0 до 5. Для их хранения достаточно 1 байта.

Для работы с сохранениями определена структура score\_name, которая содержит строку из 6 символов типа char и int для хранения счета. Так как количество очков во время игры меняется от 0 до нескольких тысяч, использования 4 байт оправдано.

# 4.2 Реализация функций.

*Создание случайного целого числа.*  
int rand\_int(int a, int b)

{

return a + (rand()) % (b - a + 1);

}  
Листинг 4.1. Функция для создания случайного числа.  
Для реализации игры необходимо генерировать случайные числа в определенном диапазоне, функция стандартной библиотека rand() не позволяет этого сделать. Функция rand\_int вызывает функцию rand() и с помощью формулы «обрезает» полученное псевдослучайное число до нужного диапазона.

*Создание новой фигуры.*  
Функция принимает указатель на структуру фигуры, которую она заполнит. В функции определяется массив из 7 двумерных массивов размером 4 на 4, содержащих информацию о всех фигурах в виде нулей и единиц. 1 – заполненный блок, 0 – пустота. Случайным образом с помощью функции rand\_int() определяется тип и цвет фигуры. В соответствии с типом в поле blocks фигуры переносится 1 из 7 массивов. Далее координате x присваивается значение 3 (так фигура появится в середине поля). За тем в цикле определяется самая нижняя строка в поле blocks фигуры, где есть хотя бы одна заполненная клетка и в соответствии с этой строкой координате y присваивается значение (фигура всегда будет создаваться вне поля, но выше всего на 1 клетку).

void create\_figure(figure \*a)

{

int figures[7][4][4] = {

{{0, 1, 0, 0},

{0, 1, 0, 0},

{0, 1, 0, 0},

{0, 1, 0, 0}},

{{0, 0, 0, 0},

{0, 1, 1, 0},

{0, 1, 1, 0},

{0, 0, 0, 0}},

{{0, 1, 0, 0},

{1, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{1, 1, 0, 0},

{0, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{0, 1, 1, 0},

{1, 1, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{0, 0, 1, 0},

{1, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{1, 0, 0, 0},

{1, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

};

int i, j;

a->color = rand\_int(1, 5);

a->type = rand\_int(0, 6);

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

a->blocks[i][j] = figures[a->type][i][j];

}

}

a->x = 3;

for (i = 3; i >= 0; i--)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

a->y = j - 5;

return;

}

}

}

}  
Листинг 4.2. Функция создания новой фигуры.

*Поворот фигуры.*  
Функция принимает указатель на структуру фигуры, которую необходимо повернуть. В функции создается буферный массив 4 на 4. Он заполняется повернутой фигурой. Поворот реализован по-разному в зависимости от типа фигуры: прямая из 4 блоков может принимать только два положения и ее необходимо «отразить», поменяв у каждого блока координаты x и y местами; квадрат не нужно поворачивать; остальные фигуры поворачиваются на 90 градусов вправо. После этого массиву фигуры-аргумента присваивается значения буферного массива.

void rotate\_figure(figure \*a)

{

int buf[4][4] = {0};

int i, j;

if (a->type == 0)

{

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

buf[i][j] = a->blocks[j][i];

}

}

}

else if (a->type == 1)

{

return;

}

else if (a->type > 1)

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

buf[i][j] = a->blocks[2 - j][i];

}

}

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

a->blocks[i][j] = buf[i][j];

}

}

}

Листинг 4.3. Функция поворота фигуры.

*Вывод в консоль одного кадра.*  
В функции определяется дескриптор, указывающий на стандартный вывод, курсор устанавливается на координаты 0;0, то есть в левый верхний угол консоли (эти структуры и функции из библиотеки windows.h). Функция выводит поле и активную фигуру на ней. На 5-й строке выводится сообщение о паузе, если она включена, на 6-й счет, на 10-13 строках следующую фигуру. Для вывода цвета блоков в консоль используются escape-последовательности. В зависимости от того, какое число стоит в ячейке массива, используется соответствующая последовательность.

void print\_frame(int score, char field[][SCREEN\_X\_SIZE], const figure \*block, const figure \*next\_block, int pause)

{

int i, j;

HANDLE hCon;

COORD cPos;

hCon = GetStdHandle(STD\_OUTPUT\_HANDLE);

cPos.X = 0;

cPos.Y = 0;

SetConsoleCursorPosition(hCon, cPos);

for (i = 0; i < SCREEN\_X\_SIZE \* 3 + 2; i++)

{

printf("=");

}

printf("\n");

for (i = 0; i < SCREEN\_Y\_SIZE; i++)

{

printf("|");

for (j = 0; j < SCREEN\_X\_SIZE; j++)

{

if ((i - block->y >= 0) && (i - block->y < 4) && (j - block->x >= 0) && (j - block->x < 4) && (block->blocks[i - block->y][j - block->x]))

{

printf("\x1B[1;3%dm[ ]\e[0m", block->color);

}

else if (field[i][j] == 0)

{

printf(" . ");

}

else

{

printf("\x1B[1;3%dm[ ]\e[0m", field[i][j]);

}

}

printf("|");

if (pause && i == 5)

{

printf("\tPAUSE");

}

else if (i == 5)

{

printf("\t ");

}

if (i == 6)

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

else if (i == 8)

printf("\tNEXT: ");

else if (i == 9)

{

printf("\t");

for (j = 0; j < 3 \* 4 + 2; j++)

{

printf("=");

}

}

else if (i > 9 && i < 14)

{

printf("\t|");

for (j = 0; j < 4; j++)

{

if (next\_block->blocks[i - 10][j])

{

printf("\x1B[1;3%dm[ ]\e[0m", next\_block->color);

}

else

{

printf(" ");

}

}

printf("|");

}

else if (i == 14)

{

printf("\t");

for (j = 0; j < 3 \* 4 + 2; j++)

{

printf("=");

}

}

printf("\n");

}

for (i = 0; i < SCREEN\_X\_SIZE \* 3 + 2; i++)

{

printf("=");

}

printf("\n");

}

Листинг 4.4. Функция вывода одного кадра в консоль.

*Проверка наложения фигуры на поле.*Функция в цикле проходится по локальным координатам фигуры. Если на соответствующих глобальных позициях поля есть заполненные клетки, считается что наложение невозможна и функция возвращает ложь, в ином случае истину.   
int check\_bottom(figure \*a, char field[][SCREEN\_X\_SIZE])

{

int i, j;

int x, y;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

x = a->x + j;

y = a->y + i;

if (y < 0)

break;

if (y > SCREEN\_Y\_SIZE - 1 || field[y][x])

{

return 0;

}

}

}

}

return 1;

}

Листинг 4.5. Функция проверки наложения фигуры на поле.

*Проверка наложения фигуры на поле (при действиях пользователя)*.  
Функция аналогична предыдущей, однако, кроме наложения на блоки на поле проверяет выход фигуры за границы поля. Такое событие может произойти только в результате действий пользователя.

int check\_turn(figure \*a, char field[][SCREEN\_X\_SIZE])

{

int i, j;

int x, y;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

x = a->x + j;

y = a->y + i;

if (x > SCREEN\_X\_SIZE - 1 || x < 0 || y > SCREEN\_Y\_SIZE - 1 || field[y][x])

{

return 0;

}

}

}

}

return 1;

}

Листинг 4.6. Функция проверки наложения фигуры на поле (при действиях пользователя).

*Функция наложения фигуры на поле.*Функция в цикле проходится по локальным координатам фигуры и переносит на поле заполненные клетки в виде номера цвета фигуры.

void overlay(figure \*a, char field[][SCREEN\_X\_SIZE])

{

int i, j;

int x, y;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

x = a->x + j;

y = a->y + i;

if (y >= 0)

{

field[y][x] = a->color;

}

}

}

}

}

Листинг 4.7. Функция наложения фигуры на поле.

*Проверка поля на заполненные линии.*  
Функция в цикле проходится по строкам массива снизу вверх, если обнаружена заполненная строка, все строки выше смещаются на один блок вниз, число возвращаемых очков увеличивается в 2 раза + 100. Функция возвращает целое число -количество очков, полученных при отчистке поля от заполненных линий.

int check\_field(char field[][SCREEN\_X\_SIZE])

{

int i, j, c, f, k;

c = 0;

for (i = SCREEN\_Y\_SIZE - 1; i >= 0; i--)

{

f = 1;

for (j = 0; j < SCREEN\_X\_SIZE; j++)

{

if (!field[i][j])

{

f = 0;

break;

}

}

if (f)

{

c = c \* 2 + 100;

for (k = i; k >= 1; k--)

{

for (j = 0; j < SCREEN\_X\_SIZE; j++)

{

field[k][j] = field[k - 1][j];

}

}

i++;

}

}

return c;

}

Листинг 4.7. Функция проверки поля.

*Игра.*  
Сначала создаются и определяются переменные и массив поля. Создаются три фигуры: актуальная, буферная (аналогичная актуальной) и следующая. В бесконечном цикле начинается цикл, который длится change\_frame\_time секунд, в котором происходит еще один цикл, длинною 0.03 секунды, в котором если клавиша нажата, она считывается в переменную input. Функции \_getch() и kbhit() определены в библиотеке conio.h. В соответствии нажатой клавише меняется свойство буферной фигуры (изменяются координаты или совершается поворот) или меняется состояние паузы на противоположное. Далее проверяется допустимость такого изменения. Если изменение допустимо, свойства приписываются актуальной фигуре. Если была нажата пауза, нажатия считываются, но обрабатываются только случай повторного нажатия на паузу. В конце цикла, который длится change\_frame\_time секунд, буферная фигура опускается на 1 блок вниз, проверяется допустимость этого действия. Если допустимо, актуальная фигура опускается на 1 блок вниз, цикл завершается и начинается новая итерация. Если нет, актуальная фигура накладывается на поле, совершается проверка на заполненные линии и результат прибавляется к переменной счета. Если при наложении, координата y фигуры меньше нуля (фигура частично или полностью вне поля), функция завершается, возвращая количество очков. Если нет, актуальная фигура копирует следующую, следующая создается, время смены кадра change\_frame\_time вычисляется по формуле (25 - (score / 1000)) / 100.0, то есть время смещения фигуры на 1 блок вниз меняется от 0.25 секунд до 0, каждые 1000 очков уменьшаясь на 0.01 секунду. Технически возможна ситуация, когда предполагаемый промежуток времени станет отрицательным, однако для этого пользователь должен набрать 1000 очков со скоростью падения фигуры 100 блоков в секунду, что представляется невозможным.

int game()

{

clock\_t start\_time, frame\_start\_time;

int score;

figure block;

figure next\_block;

figure buf\_block;

char input;

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

int i;

int pause;

double change\_frame\_time;

create\_figure(&block);

create\_figure(&next\_block);

buf\_block = block;

change\_frame\_time = 0.25;

pause = 0;

input = 0;

score = 0;

while (1)

{

frame\_start\_time = clock();

while (((double)(clock() - frame\_start\_time)) / CLOCKS\_PER\_SEC < change\_frame\_time)

{

start\_time = clock();

// system("cls"); //

print\_frame(score, field, &block, &next\_block, pause);

// printf("%d\n", input);

while (((double)(clock() - start\_time)) / CLOCKS\_PER\_SEC < 0.03)

{

if (kbhit())

{

input = \_getch();

if (!input)

{

input = \_getch();

}

}

}

if (input == 27)

{

pause = !pause;

};

if (!pause)

{

if (input == 'h' || input == 75) // LEFT

{

buf\_block.x--;

if (check\_turn(&buf\_block, field))

{

block.x--;

}

}

if (input == 'u' || input == 72) // ROTATE

{

rotate\_figure(&buf\_block);

if (check\_turn(&buf\_block, field))

{

rotate\_figure(&block);

}

}

if (input == 'j' || input == 80) // DOWN

{

buf\_block.y++;

if (check\_turn(&buf\_block, field))

{

block.y++;

}

}

if (input == 'k' || input == 77)

{

buf\_block.x++;

if (check\_turn(&buf\_block, field))

{

block.x++;

}

}

}

input = 0;

}

if (pause)

{

continue;

}

buf\_block = block;

buf\_block.y++;

if (check\_bottom(&buf\_block, field))

{

block.y++;

}

else

{

if (buf\_block.y < 0)

{

return score;

}

overlay(&block, field);

block = next\_block;

create\_figure(&next\_block);

// return 0;

}

score += check\_field(field);

change\_frame\_time = ((25 - (score / 1000)) / 100.0);

}

}

Листинг 4.8. Функция, реализующая тетрис.

*Сохранение таблицы лидеров.*int save\_score\_board(score\_name \*list, char \*way);  
Функция открывает файл для записи, записывает 5 пар значений имя – счет и закрывает файл. Возвращает ложь в случае неудачного открытия файла.

*Сортировка массива с записями о лучших играх.*void sort\_board(score\_name \*list);

Функция сортирует массив методом пузырьковой сортировки.

*Импорт таблицы лидеров.*

int import\_score\_board(score\_name \*list, char \*way);

Функция открывает файл для чтения, считывает 5 пар значений имя – счет в массив и закрывает файл. Если считываемые данные некорректны, записывает в массив «пустую» запись «EMTPY 0». Если файл не существует, создает файл, открывает для записи и записывает 5 «пустых» пар значений «EMTPY 0». В конце сортирует массив. Возвращает ложь в случае неудачного открытия файла для записи.

*Запись нового рекорда.*  
int new\_records(score\_name \*score\_list, int score, char \*way);  
В функцию подается указатель на список записей, количество очков которое надо проверить и путь до файла. Функция проходится по отсортированному массиву записей. Если найдена запись с меньшим количеством очков чем score, последующие записи сдвигаются вправо. Курсор сдвигается на позицию 2;6, функция считывает 5 символов из допустимого списка (от 48 до 126 согласно таблицы ASCII) в буферную строку. Если символ недопустим, курсор сдвигается влево на 1 позицию. Таким образом ввод не будет закончен, пока не введены 5 допустимых символов. Информация из буферной строки переписывается в актуальную запись в массиве, после этого список сохраняется в файле.

*Основная функция.*В основной функции меню реализовано с помощью бесконечного цикла, в котором печатается пункты меню и запрашивается ввод. При корректном вводе запускается соответствующая функция, при неверной экран отчищается и меню выводится снова. Прерывание цикла и выход из программы предусмотрен соответствующим пунктом меню.

## 5.Тестирование

## 5.1. Планирование тестирования.

Тестирование должно состоять из:  
-проверка каждой отдельной функции на работоспособность и адекватность;  
-проверка корректного поведения программы как одного целого в различных игровых ситуациях: движение фигуры, пауза, изменение скорости и пр.  
-проверка работы с памятью: сохранения и загрузки записей в файл;

# 5.2. Разработка тестов.

*Проверка создания фигуры.*  
Необходимо несколько раз вызвать функцию, подав в нее указатель на тестовую фигуру и убедиться, что полученные значения случайны, а массив заполнен корректно.

*Проверка поворота фигуры.*  
Необходимо создать несколько тестовых фигур различного типа и поворота и убедиться, что поворот работает с каждым типом фигур правильно.

*Проверка отображения одного кадра.*  
Необходимо создать поле (целочисленный двумерный массив размером 10 на 20), заполнить его числами от 0 до 5; создать несколько фигур различного типа, цвета и с разными координатами и вызвать функцию отображения кадра с этими параметрами. Правильно реализованная функция выведет поле с пустыми и заполненными цветными блоками поле и цветные блоки на месте, где должна располагаться фигура.

*Проверка допустимости движения.*  
Для проверки допустимости движения нужно создать поле, заполнить его числами, создать несколько фигур разного типа и положения на поле. Убедиться, что функции безошибочно определяют, когда движение возможно, а когда нет.

*Проверка наложения фигуры на поле.*  
Для этой проверки необходимо создать поле, вывести его, создать фигуру с известными координатами, вызвать функцию, снова вывести поле и убедиться, что фигура наложилась на поле в соответствии со своими координатами.

*Проверка обнаружения заполненных строк поля.*  
Необходимо создать несколько полей с разным количеством заполненных строк (от 0 до 4), проверить, что функция возвращает правильное количество очков и что удаление строк выполняется корректно, со смещением других строк вниз.

*Проверка загрузки.*  
Для этой проверки необходимо создать несколько файлов: заполненного верно, заполненных верно частично, пустой, заполненный мусором. Проверить что загрузка верно обрабатывает все эти ситуации. Проверить ситуацию отсутствия файла. При правильной загрузке, строки, которые по какой-либо причине невозможно считать, записывать в массив как стандартные «пустые» значения.

*Проверка сохранения.*  
Проверить, что данные из массива сохраняются верно.

*Проверка обнаружения и записи нового рекорда.*  
Для проверки этой функции необходимо создать массив с 5 записями с различным счетом. Необходимо проверить ввод нового имени. Должны быть проверены допустимые и недопустимые символы, а также нажатие функциональных клавиш. После ввода необходимо убедиться, что новая запись стоит на своем месте, а данные записаны корректно.

*Проверка игры.*  
Необходимо проверить, что игра ведет себя адекватно и предсказуемо. Нужно проверить различные игровые ситуации, убедиться, что нет ошибок. Для этого нужно сыграть несколько игр.

### 5.3. Испытание ПО.

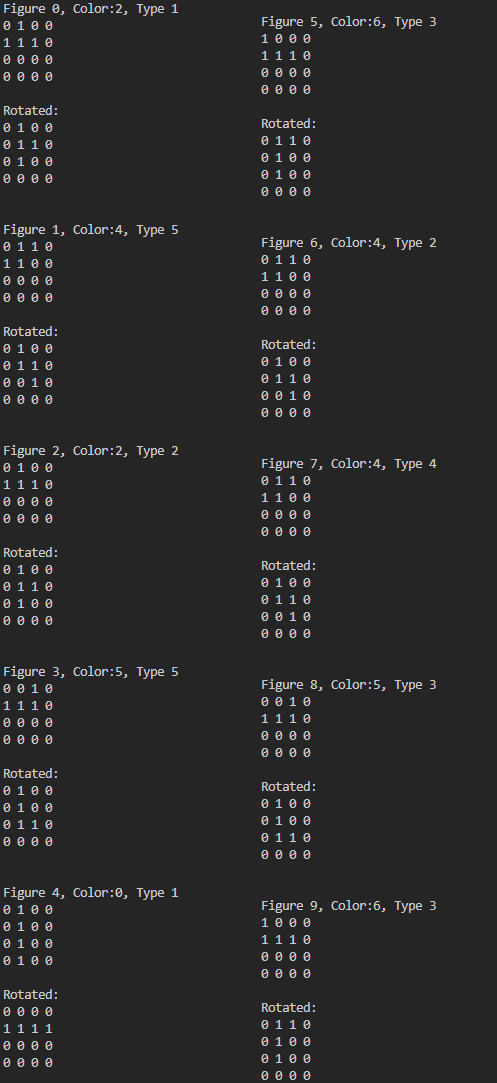
*Проверка создания и поворота фигуры.*  
На рис. 5.1 результат создания нескольких фигур и их поворотов. Функции работают предсказуемо.  


Рис 5.1. Проверка создания и поворота фигуры

*Проверка отображения кадра.*  
Согласно разработанному теста несколько раз созданы объекты и вызвана функция. На рис. 5.2 результат одного из такого вызова. Этот и прочие тесты ошибок не обнаружили.

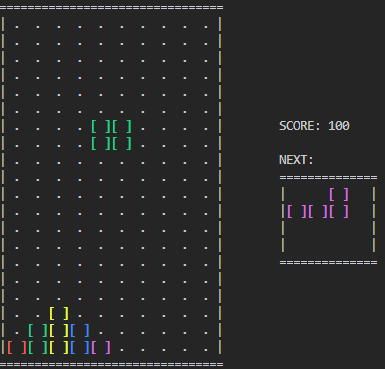


Рис.5.2. Проверка отображения кадра

*Проверки допустимости движения, наложения фигуры на поле, обнаружения заполненных строк поля .*  
Проверки функций допустимости движения, наложения фигуры на поле и обнаружения заполненных строк успешно прошли через тесты. Ошибок не обнаружено.

*Проверка загрузки и сохранения.*Для проверки использовались 4 файла, 1 с правильным вводом, 3 с неправильным, 1 файл не существовал. После чтения, массив был сохранен в файл. Результаты чтения в массив и его записи представлены в таблице 5.3. Как и ожидалось, прочитаны будут только правильно собранные строки из пар слов из 5 допустимых символов и натуральных чисел. Из чего можно сделать вывод, что функции работает правильно.

|  |  |  |
| --- | --- | --- |
| Содержимое файла | Данные, записанные в массив | Данные, записанные в файл |
| aaaaa 1000  bbbbb 900  ccccc 400  ddddd 300  eeeee 0 | aaaaa 1000  bbbbb 900  ccccc 400  ddddd 300  eeeee 0 | aaaaa 1000  bbbbb 900  ccccc 400  ddddd 300  eeeee 0 |
| bbbbb 900  ddddd 300  eeeee 100 | bbbbb 900  ddddd 300  eeeee 100  EMPTY 0  EMPTY 0 | bbbbb 900  ddddd 300  eeeee 100  EMPTY 0  EMPTY 0 |
| asdsfsafqwe 1000  bbbbasdb 900  ccccc 400  ddd 300  e 0 | EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0 | EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0 |
| абвгд 1000  ☺☻♥♦♣ 900  ccccc 400.0  ddddd 300 | ccccc 400  EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0 | ccccc 400  ddddd 300  EMPTY 0  EMPTY 0  EMPTY 0 |
| Файл не существует | EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0 | EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0  EMPTY 0 |

Таблица 5.3. Результат тестирования ввода

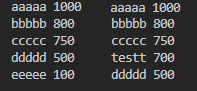
*Проверка обнаружения и записи нового рекорда.*Был создан массив типа score\_name из 5 элементов. Функция вызвана со значением score = 700, имя (“test”) введено, запись успешно помещена в массив (см. Рис. 5.4) . Так же проверен ввод случайных символов и нажатие функциональных клавиш, программа считывает только допустимые символы, функция работает правильно.  


Рис 5.4 Массив до и после введения новой записи

*Проверка игры.*  
В рамках теста было сыграно несколько игр. Все функции игры работают согласно заданным требованиям. Единственное замечание – это легкая неотзывчивость управления.

5.4 Итоги тестирования.  
При прохождении тестов, ошибок в функциях или алгоритмах не выявлены. Поведения всех функций соответствуют ожиданиям.

## 6. Сопровождение.

## 6.1. Требования к программно-аппаратной платформе. Из-за используемых библиотек, программа может запускаться только на операционной системе Windows. Минимальное место на диске – 17Кб, определено размером исходного файла и файла сохранения. Размер исполняемого файла может меняться в зависимости от системы.

## 6.2 Инструкция по установке и использованию. Для компиляции исходного кода потребуются нестандартные библиотеки <conio.h> и <windows.h>. При взаимодействии с меню от пользователя ожидается ввод числа, соответствующий пункту меню. Для управления игрой используются клавиши U, H, J, K или стрелочки. Для включения или выключения паузы используется Ecs. Информацию об управлении также можно посмотреть в соответствующим разделе меню.

## 6.3. Сведенья о недостатках программного продукта. Управление игрой является несколько неотзывчивым, это может принести легких дискомфорт у пользователя. Так же из-за особенностей вывода данных в консоль, программа не может быть запущена на операционных системах семейства Linux.

## 7.Заключение. В ходе выполнения курсового проекта был создан продукт, соответствующий заданию. Была реализована игра «тетрис» и возможность сохранять результат лучших игр. Достоинства: -Простая и нетребовательная реализация; -Возможность сохранять результаты лучших игр;

Недостатки:  
-Жесткая привязка к операционной системе;  
-Тугое управление.

Перспективы дальнейшего развития:  
-Улучшение визуальной составляющей;  
-Изменение управления на более удобное;  
-Адаптация под другие операционные системы;  
-Оптимизация алгоритмов, уменьшение объемов требуемой памяти;  
-Добавление возможности менять управление;  
-Добавление возможности играть двум игрокам одновременно.

**Список литературы**

1. Тетрис [Электронный курс]//Википедия[сайт] URL: <https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%82%D1%80%D0%B8%D1%81>

2. Инструкция по запуску игры Tetris на экране паяльника TS-100 [Электронный ресурс]//ixbt [сайт] URL: <https://www.ixbt.com/live/nemoi13/instrukciya-po-zapusku-igry-tetris-na-ekrane-payalnika-ts-100.html>

3. Онлайн-тетрис [Электронный ресурс]//online-tetris [сайт] URL: <https://online-tetris.ru/>

4. История тетриса [Электронный ресурс]// vadim.oversigma URL: <https://vadim.oversigma.com/Tetris.htm>

5. BrickGame [Электронный курс]// Википедия [сайт] URL: <https://ru.wikipedia.org/wiki/Brick_Game>

6.Tetris(GameBoy)[Электронный курс] // tetris.wiki [сайт] URL: <https://tetris.wiki/Tetris_(Game_Boy)>

# **Приложения**

# Приложение 1. Листинг программы

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <conio.h>

#include <time.h>

#define SCREEN\_X\_SIZE 10

#define SCREEN\_Y\_SIZE 20

#define SCORE\_BOARD\_SIZE 5

#define SAVE\_FILE "score\_list.txt"

#ifndef ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING

#define ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING 0x0004

#endif

enum \_colors

{

RED = 1,

GREEN = 2,

YELLOW = 3,

BLUE = 4,

PURPLE = 5

};

int rand\_int(int a, int b)

{

return a + (rand()) % (b - a + 1);

}

typedef struct figure

{

char x;

char y;

char color;

char type;

char blocks[4][4];

} figure;

typedef struct score\_name

{

char name[6];

int score;

} score\_name;

void rotate\_figure(figure \*a)

{

int buf[4][4] = {0};

int i, j;

if (a->type == 0)

{

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

buf[i][j] = a->blocks[j][i];

}

}

}

else if (a->type == 1)

{

return;

}

else if (a->type > 1)

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

buf[i][j] = a->blocks[2 - j][i];

}

}

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

a->blocks[i][j] = buf[i][j];

}

}

}

void create\_figure(figure \*a)

{

char figures[7][4][4] = {

{{0, 1, 0, 0},

{0, 1, 0, 0},

{0, 1, 0, 0},

{0, 1, 0, 0}},

{{0, 0, 0, 0},

{0, 1, 1, 0},

{0, 1, 1, 0},

{0, 0, 0, 0}},

{{0, 1, 0, 0},

{1, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{1, 1, 0, 0},

{0, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{0, 1, 1, 0},

{1, 1, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{0, 0, 1, 0},

{1, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

{{1, 0, 0, 0},

{1, 1, 1, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}},

};

int i, j;

a->color = rand\_int(1, 5);

a->type = rand\_int(0, 6);

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

a->blocks[i][j] = figures[a->type][i][j];

}

}

a->x = 3;

for (i = 3; i >= 0; i--)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

a->y = -i-1;

return;

}

}

}

}

void print\_frame(int score, char field[][SCREEN\_X\_SIZE], const figure \*block, const figure \*next\_block, int pause)

{

int i, j;

HANDLE hCon;

COORD cPos;

hCon = GetStdHandle(STD\_OUTPUT\_HANDLE);

cPos.X = 0;

cPos.Y = 0;

SetConsoleCursorPosition(hCon, cPos);

for (i = 0; i < SCREEN\_X\_SIZE \* 3 + 2; i++)

{

printf("=");

}

printf("\n");

for (i = 0; i < SCREEN\_Y\_SIZE; i++)

{

printf("|");

for (j = 0; j < SCREEN\_X\_SIZE; j++)

{

if ((i - block->y >= 0) && (i - block->y < 4) && (j - block->x >= 0) && (j - block->x < 4) && (block->blocks[i - block->y][j - block->x]))

{

printf("\x1B[1;3%dm[ ]\e[0m", block->color);

}

else if (field[i][j] == 0)

{

printf(" . ");

}

else

{

printf("\x1B[1;3%dm[ ]\e[0m", field[i][j]);

}

}

printf("|");

if (pause && i == 5)

{

printf("\tPAUSE");

}

else if (i == 5)

{

printf("\t ");

}

if (i == 6)

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

else if (i == 8)

printf("\tNEXT: ");

else if (i == 9)

{

printf("\t");

for (j = 0; j < 3 \* 4 + 2; j++)

{

printf("=");

}

}

else if (i > 9 && i < 14)

{

printf("\t|");

for (j = 0; j < 4; j++)

{

if (next\_block->blocks[i - 10][j])

{

printf("\x1B[1;3%dm[ ]\e[0m", next\_block->color);

}

else

{

printf(" ");

}

}

printf("|");

}

else if (i == 14)

{

printf("\t");

for (j = 0; j < 3 \* 4 + 2; j++)

{

printf("=");

}

}

printf("\n");

}

for (i = 0; i < SCREEN\_X\_SIZE \* 3 + 2; i++)

{

printf("=");

}

printf("\n");

}

int check\_bottom(figure \*a, char field[][SCREEN\_X\_SIZE])

{

int i, j;

int x, y;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

x = a->x + j;

y = a->y + i;

if (y < 0)

break;

if (y > SCREEN\_Y\_SIZE - 1 || field[y][x])

{

return 0;

}

}

}

}

return 1;

}

int check\_turn(figure \*a, char field[][SCREEN\_X\_SIZE])

{

int i, j;

int x, y;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

x = a->x + j;

y = a->y + i;

if (x > SCREEN\_X\_SIZE - 1 || x < 0 || y > SCREEN\_Y\_SIZE - 1 || field[y][x])

{

return 0;

}

}

}

}

return 1;

}

void overlay(figure \*a, char field[][SCREEN\_X\_SIZE])

{

int i, j;

int x, y;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++)

{

if (a->blocks[i][j])

{

x = a->x + j;

y = a->y + i;

if (y >= 0)

{

field[y][x] = a->color;

}

}

}

}

}

int check\_field(char field[][SCREEN\_X\_SIZE])

{

int i, j, c, f, k;

c = 0;

for (i = SCREEN\_Y\_SIZE - 1; i >= 0; i--)

{

f = 1;

for (j = 0; j < SCREEN\_X\_SIZE; j++)

{

if (!field[i][j])

{

f = 0;

break;

}

}

if (f)

{

c = c \* 2 + 100;

for (k = i; k >= 1; k--)

{

for (j = 0; j < SCREEN\_X\_SIZE; j++)

{

field[k][j] = field[k - 1][j];

}

}

i++;

}

}

return c;

}

int game()

{

clock\_t start\_time, frame\_start\_time;

int score;

figure block;

figure next\_block;

figure buf\_block;

char input;

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

int i;

int pause;

double change\_frame\_time;

create\_figure(&block);

create\_figure(&next\_block);

buf\_block = block;

change\_frame\_time = 0.25;

pause = 0;

input = 0;

score = 0;

while (1)

{

frame\_start\_time = clock();

while (((double)(clock() - frame\_start\_time)) / CLOCKS\_PER\_SEC < change\_frame\_time)

{

start\_time = clock();

// system("cls"); //

print\_frame(score, field, &block, &next\_block, pause);

// printf("%d\n", input);

while (((double)(clock() - start\_time)) / CLOCKS\_PER\_SEC < 0.03)

{

if (kbhit())

{

input = \_getch();

if (!input)

{

input = \_getch();

}

}

}

if (input == 27)

{

pause = !pause;

};

if (!pause)

{

if (input == 'h' || input == 75) // LEFT

{

buf\_block.x--;

if (check\_turn(&buf\_block, field))

{

block.x--;

}

}

if (input == 'u' || input == 72) // ROTATE

{

rotate\_figure(&buf\_block);

if (check\_turn(&buf\_block, field))

{

rotate\_figure(&block);

}

}

if (input == 'j' || input == 80) // DOWN

{

buf\_block.y++;

if (check\_turn(&buf\_block, field))

{

block.y++;

}

}

if (input == 'k' || input == 77)

{

buf\_block.x++;

if (check\_turn(&buf\_block, field))

{

block.x++;

}

}

}

input = 0;

}

if (pause)

{

continue;

}

buf\_block = block;

buf\_block.y++;

if (check\_bottom(&buf\_block, field))

{

block.y++;

}

else

{

if (buf\_block.y < 0)

{

return score;

}

overlay(&block, field);

block = next\_block;

create\_figure(&next\_block);

// return 0;

}

score += check\_field(field);

change\_frame\_time = ((25 - (score / 1000)) / 100.0);

}

}

int save\_score\_board(score\_name \*list, char \*way)

{

FILE \*f = fopen(way, "w");

int i;

if (!f)

return 0;

for (i = 0; i < SCORE\_BOARD\_SIZE; i++)

{

fprintf(f, "%s %d\n", list[i].name, list[i].score);

}

fclose(f);

return 1;

}

void sort\_board(score\_name \*list)

{

int i, j;

score\_name temp;

for (i = 0; i < SCORE\_BOARD\_SIZE; i++)

{

for (j = i + 1; j < SCORE\_BOARD\_SIZE; j++)

{

if (list[i].score < list[j].score)

{

temp = list[i];

list[i] = list[j];

list[j] = temp;

}

}

}

}

int import\_score\_board(score\_name \*list, char \*way)

{

FILE \*f = fopen(way, "r");

char chbuf[100];

int i, j, error;

if (!f)

{

f = fopen(way, "w");

if (!f)

return 0;

for (i = 0; i < SCORE\_BOARD\_SIZE; i++)

{

fprintf(f, "EMPTY 0\n");

}

fclose(f);

}

fclose(f);

f = fopen(way, "r");

for (i = 0; i < SCORE\_BOARD\_SIZE; i++)

{

error = 0;

if (fscanf(f, "%s %d", chbuf, &list[i].score) != 2 || list[i].score < 0 || strlen(chbuf) != 5)

{

strcpy(list[i].name, "EMPTY");

list[i].score = 0;

continue;;

}

printf("(%d)",list[i].score);

for (j = 0; j < 5; j++)

{

if (chbuf[j] < 48 || chbuf[j] >= 127)

{

printf("(%d)",chbuf[j]);

strcpy(list[i].name, "EMPTY");

printf("(%s)",list[i].name);

list[i].score = 0;

error = 1;

break;

}

}

list[i].name[5] = 0;

if (error) continue;

strncpy(list[i].name, chbuf, 5);

}

fclose(f);

sort\_board(list);

return 1;

}

int new\_records(score\_name \*score\_list, int score, char \*way)

{

HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_CURSOR\_INFO structCursorInfo;

GetConsoleCursorInfo(console, &structCursorInfo);

structCursorInfo.bVisible = FALSE;

SetConsoleCursorInfo(console, &structCursorInfo);

COORD cPOS;

int i, j;

char b;

char buf[6] = {0};

for (i = 0; i < SCORE\_BOARD\_SIZE; i++)

{

if (score > score\_list[i].score)

{

printf("\t!!!NEW RECORD!!!\n"

"\tEnter your name:\n"

">>\_\_\_\_\_");

cPOS.X = 2;

cPOS.Y = 6;

SetConsoleCursorPosition(console, cPOS);

for (j = 0; j < 5; j++)

{

b = \_getche();

if (b >= 48 && b < 127)

{

buf[j] = b;

}

else

{

if (!b)

{

\_getch();

}

cPOS.X = 2 + j;

SetConsoleCursorPosition(console, cPOS);

printf("\_");

SetConsoleCursorPosition(console, cPOS);

j--;

}

}

for (j = SCORE\_BOARD\_SIZE - 1; j > i; j--)

{

score\_list[j].score = score\_list[j - 1].score;

strcpy(score\_list[j].name, score\_list[j - 1].name);

}

strcpy(score\_list[i].name, buf);

score\_list[i].name[6] = 0;

score\_list[i].score = score;

save\_score\_board(score\_list, SAVE\_FILE);

break;

}

}

}

int main()

{

srand(time(NULL));

HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);

DWORD consoleMode;

GetConsoleMode(console, &consoleMode);

consoleMode |= ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING;

SetConsoleMode(console, consoleMode);

CONSOLE\_CURSOR\_INFO structCursorInfo;

GetConsoleCursorInfo(console, &structCursorInfo);

structCursorInfo.bVisible = FALSE;

SetConsoleCursorInfo(console, &structCursorInfo);

COORD cPOS;

clock\_t start\_time;

int score;

int key;

int i, j;

char b;

score\_name score\_list[SCORE\_BOARD\_SIZE];

FILE \*f;

import\_score\_board(score\_list, SAVE\_FILE);

while (1)

{

system("cls");

printf("\x1b[1;30;47m\tTETRIS\x1b[0m\n\n");

printf("\tMenu\n"

"\t1.Start game\n"

"\t2.Control\n"

"\t3.Leaderboard\n"

"\t0.Exit\n"

"\t>> ");

scanf("%d", &key);

fflush(stdin);

switch (key)

{

case 1:

system("cls");

score = game();

system("cls");

printf("\x1b[1;30;47m\tGAME OVER\x1b[0m\n\n");

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

start\_time = clock();

while (((double)(clock() - start\_time)) / CLOCKS\_PER\_SEC < 2)

{

\_getch();

}

new\_records(score\_list, score, SAVE\_FILE);

break;

case 2:

system("cls");

printf("\x1b[1;30;47m\tCONTROL\x1b[0m\n\n"

"[Esc] - PAUSE\n"

" ROTATE - [u]\n"

" LEFT - [h] [j] [k] - RIGHT \n"

"\t\t|\n"

"\t DOWN \n"

"\tback - any key\n"

"\t>> ");

\_getch();

break;

case 3:

system("cls");

printf("\x1b[1;30;47m\tRECORDS\x1b[0m\n");

for (i = 0; i < SCORE\_BOARD\_SIZE; i++)

{

printf("\t%s %d\n", score\_list[i].name, score\_list[i].score);

}

printf("\tback - any key\n\t>> ");

\_getch();

break;

case 0:

return 0;

default:

printf("Unknown command\n");

break;

}

}

return 0;

}

Приложение 2. Блок схема алгоритма игры