МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РФ

ФГБОУ ВО «Кубанский государственный   
аграрный университет имени И.Т. Трубилина»

Кафедра системного анализа и обработки информации

**Курсовая работа**

по дисциплине: Программирование

на тему Разработка игрового приложения «Тетрис»

выполнил студент группы: БИ1602

Белоусов Юрий Эдуардович

Допущен к защите

Руководитель проекта Крамаренко Татьяна Анатольевна

Нормоконтролер Крамаренко Т.А.

(подпись, расшифровка подписи)

Защищен \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Члены комиссии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата, расшифровка подписи)

Краснодар - 2017

МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РФ

ФГБОУ ВО «Кубанский государственный   
аграрный университет имени И.Т. Трубилина»

Кафедра системного анализа и обработки информации

**УТВЕРЖДАЮ:**

Зав. кафедрой Барановская Т.П.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ЗАДАНИЕ**

**на курсовую работу**

Студенту: БИ1602 группы 1 курса

Факультета прикладной информатики а Специальности: 38.03.05 – Бизнес-информатика

Белоусова Юрия Эдуардовича ы

(ФИО)

Тема проекта: Разработка игрового приложения «Тетрис»

Содержание задания: Разработка игрового приложения «Тетрис» с искусственным интеллектом

Объем работы:

а) пояснительная записка к работе\_\_\_\_\_\_35\_\_\_\_\_\_\_\_\_\_листа формата А4

б) графическая часть\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_лист формата А4

Рекомендуемая литература: Мурлин А.Г., Ефанова Н.В., Иванова Е.А., Гончар О.М. – Программирование (С\С++) – Краснодар, 2012. – 117 с.

Срок выполнения проекта: с “ 1 ” марта по “5” июня 2017 г.

Срок защиты: “24” июня 2017 г.

Дата выдачи задания: “1” марта 2017 г.

Дата сдачи проекта на кафедру: “5” июня 2017 г.

Руководитель проекта \_Крамаренко Татьяна Анатольевна, к.п.н. .

(подпись, Ф.И.О., звание, степень)

Задание принял студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Краснодар – 2017

**РЕФЕРАТ**

35 стр., 16 рис.

ИГРА, ТЕТРИС, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ГРАФИКА, МУЛЬТИМЕДИА, C++, SFML, Visual Studio

Целью работы является разработка игрового приложения «Тетрис» в среде Visual Studio.

Объект исследования – создание обучаемого искусственного интеллекта на основе оптимизации параметров с помощью генетического алгоритма для игры в тетрис.

Предмет исследования – объектно-ориентированные средства языков программирования и работа с внешней библиотекой, работающая с мультимедией. Разработанная программа позволяет играть в классическую игру «Тетрис».

Также в программе присутствуют дополнительные режимы игры: игра для двух людей, игра против искусственного интеллекта и режим для его обучения. Программа работает в оконном режиме с поддержкой спрайтов, музыки и звуков.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc484703952)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc484703953)

[1.1 Инструменты разработки 6](#_Toc484703954)

[1.2 Принцип работы генетических алгоритмов 6](#_Toc484703955)

[1.3 Подключение и работа с библиотекой SFML 9](#_Toc484703956)

[2 ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ 12](#_Toc484703957)

[2.1 Структура и описание классов. 12](#_Toc484703958)

[2.2 Примеры и описание функций классов 17](#_Toc484703959)

[2.3 Принцип работы алгоритма искусственного интеллекта 18](#_Toc484703960)

[3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 21](#_Toc484703961)

[ЗАКЛЮЧЕНИЕ 24](#_Toc484703962)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc484703963)

[ПРИЛОЖЕНИЕ 26](#_Toc484703964)

# **ВВЕДЕНИЕ**

Тетрис - компьютерная игра, изобретённая в СССР Алексеем Пажитновым и представленная общественности 6 июня 1984 года. Идею «Тетриса» ему подсказала купленная им игра в пентамино. Название игры происходит от геометрических фигур «тетрамино», которые состоят из четырёх квадратов.

Целью данной работы является разработка игрового приложения Тетрис с искусственным интеллектом работающим на генетическом алгоритме в среде разработки Visual Studio c использованием мультимедийном библиотекой SFML.

Задачи данного проекта:

* написать логику игры;
* Нарисовать текстуры;
* Найти подходящую музыку и звуки;
* Придумать реализацию генетического алгоритма;
* Создать удобный интерфейс;
* Разработать разные режимы игры;

1. **ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

Игровое приложение будет разработано на объектно-ориентированном языке программирования С++ в среде разработки Visiual Studio, с применением внешней мультимедийной библиотеки SFML

* 1. **Инструменты разработки**

С++ компилируемый строго типизированный язык программирования общего назначения. Поддерживает разные парадигмы программирования: процедурную, обобщённую, функциональную; наибольшее внимание уделено поддержке объектно-ориентированного программирования.

Для работы используется среда разработки - Microsoft Visual Studio 2015 — это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию.

Для создания пользовательского интерфейса используется специальная мультимедийная библиотека SFML(Simple and Fast Multimedia Library) -простая и быстрая мультимедийная библиотека. Написана на С++, но доступно также для C, D, Java, Python, Ruby, .Net и Go. Представляет собой объектно-ориентированный аналог SDL.

* 1. **Принцип работы генетических алгоритмов**

Генетический алгоритм (ГА) - это простая модель эволюции в природе, реализованная в виде компьютерной программы. В нем используются как аналог механизма генетического наследования, так и аналог естественного отбора. Представим себе искусственный мир, населенный множеством существ (особей), причем каждое существо - это некоторое решение нашей задачи. Будем считать особь тем более приспособленной, чем лучше соответствующее решение (чем большее значение целевой функции оно дает). Тогда задача максимизации целевой функции сводится к поиску наиболее приспособленного существа. Конечно, мы не можем поселить в наш виртуальный мир все существа сразу, так как их очень много. Вместо этого мы будем рассматривать много поколений, сменяющих друг друга. Теперь, если мы сумеем ввести в действие естественный отбор и генетическое наследование, то полученный мир будет подчиняться законам эволюции. Заметим, что, в соответствии с нашим определением приспособленности, целью этой искусственной эволюции будет как раз создание наилучших решений.

Прежде всего генетический алгоритм это, метод многомерной оптимизации, т.е. метод поиска минимума многомерной функции Потенциально этот метод можно использовать для глобальной оптимизации, но с этим возникают сложности,.

Сама суть метода заключается в том, что мы модулируем эволюционный процесс: у нас есть какая-то популяция (набор векторов), которая размножается, на которую воздействуют мутации и производится естественный отбор на основании минимизации целевой функции. Рассмотри подробнее эти процессы.

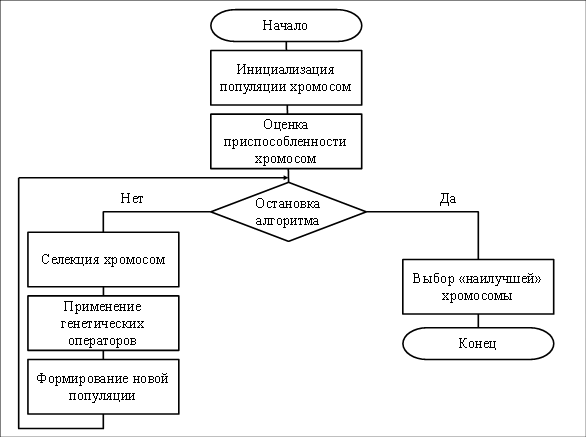
Итак, прежде всего наша популяция должна размножаться. Основной принцип размножения - потомок похож на своих родителей. Т.е. мы д задать какой­то механизм наследования. И лучше будет, если он будет включать элемент случайности. Но скорость развития таких систем очень низкая - разнообразие генетическое падает, популяция вырождается. Т.е. значение функции перестает минимизироваться.

Для решения этой проблемы был введен механизм мутации, который заключается в случайном изменении каких­то особей. Этот механизм позволяет привнести что­то новое в генетическое разнообразие.

Следующий важный механизм -селекция. Как было сказано, селекция - отбор особей (можно из только родившихся, а можно из всех - предков показывает, что это не играет решающую роль), которые лучше минимизируют функцию. Обычно отбирают столько особей, сколько было до размножения, чтобы из эпохи в эпоху у нас было постоянное количество особей в популяции. Также принято отбирать «счастливчиков» - как число особей, которые, возможно, плохо минимизируют функцию, но зато внесут разнообразия в последующие поколения.

Этих трех механизмов чаще всего недостаточно, чтобы минимизировать функцию. Так популяция вырождается - рано или поздно локальный минимум забивает своим значением всю популяцию. Когда такое происходит, проводят процесс, называемый встряской (в природе аналогии глобальные катаклизмы), когда уничтожается почти вся популяция, и добавляются новые (случайные) особи.

Вот описание классического генетического алгоритма, он прост в реализации и есть место для фантазии и исследований. На рисунке 1 представлена блок-схема ГА.

  
Рисунок 1 – Блок-схема ГА

* 1. **Подключение и работа с библиотекой SFML**

Для начала необходимо скачать SFML SDK с официального сайта. Скаченный пакет должен соответствовать версии Visual C++. Например, библиотека, скомпилированная с помощью VC++ 10 (Visu Studio 2010) не будет совместима с VC++ 12 (Visual Studio 2013).

Далее нужно распаковать архив с SFML в любую удобную директорию. Лучше всего держать библиотеки в отдельном месте, особенно если нужно использовать несколько версий одной библиотеки или несколько компиляторов.

Теперь нужно добавить в свойства проекта следующее: Путь до заголовочных файлов SFML (<путь­к­установке­SFML> / include) в C/C++ » General » Additional Include Directories Путь до библиотек SFML (<путь­к­установке­SFML> / lib) в Linker » General » Additional Library Directories (Рисунок 1 «Подключение SFML»)

Следующий шаг - компоновка вашего приложения с библиотеками SFML (файлы с расширением .lib). SFML состоит из пяти модулей (system, window, graphics, network и audio) и библиотек для каждого из них. Библиотеки должны быть добавлены в свойства проекта в Linker » Input » Additional Dependencies. После этого нужно добавить библиотеки SFML. Например «sfml­graphics.lib», «sfml­window.lib» и «sfmlsystem.lib». Также необходимо определить макрос SFML\_STATIC в опциях препроцессора проекта. На рисунке 2 представлен пример подключения.

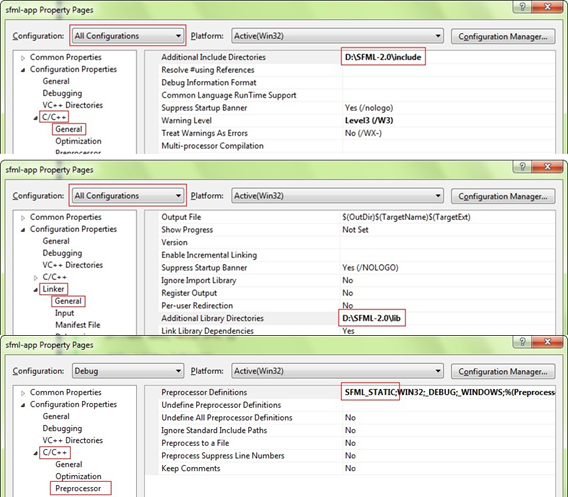


Рисунок 2 - Подключение SFML

После того как библиотека была подключена нужно проверить, что все работает правильно. Пример кода для работы программы.

#include <SFML/Graphics.hpp>

int main()

{

sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");

sf::CircleShape shape(100.f);

shape.setFillColor(sf::Color::Green);

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

}

window.clear();

window.draw(shape);

window.display();

}

return 0;

}

Вывод тестовой программы можно увидеть на рисунке 3.

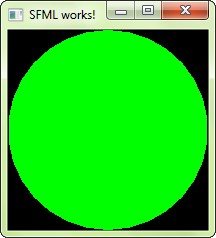


Рисунок -3 Тест SFML

1. **ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ**
   1. **Структура и описание классов.**

**CTetris** – класс запускающий логику игры с статическими функциями управления для дебага. Fields: game – статическое поле класса CGame. Methods: clearConsole – очищает консоль, mainSleep – временно останавливает поток, waitPUSH – останавливает поток пока не будет нажата любая клавиша.

**CGame** – класс управляющий всей игрой. Fields: computer – искусственный интеллект класса CComputer, factrory – переменная класса CFactory создающая фигурки, gameMode – переменная хранящая номер режима игры, isPause – логическая переменная для паузы, kc – переменная класса CKeyController для управления, listMode – номер окна списка, player1 и player2 – переменные классы CPlayer, printer – пременная класса CPrinter для отображения игры в консоли,window – окно игры (SFML), windowMode – номер режима окна в приложении, wv – переменная класса CWiindowView – для графического отображения игры. На рисунке 4 можно увидеть диаграмму классов СGame и CTetris

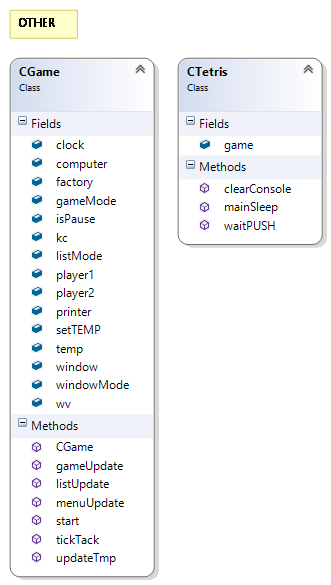


Рисунок – 4 Диаграмма OTHER

**CField –** класс поля игры. Fields: height – высота поля, width – матрица поля, matrix – двумерный массив поля. Methods: addScore – возращает кол-во очков в зависимости от кол-ва заполненных линий за один раз, clearScore – очищает матрицу поля, countHeigth – считает суммы макс высоты каждого столба матрцы, countHoles – считает кол-во дырок в матрице, countLines – считает кол-во заполненных линей, countMonoton – считает кривизну поерхности матрицы, remove – удаляет заполненные линии, setValue – ставит значение в клетку матрицы.

**CFigure -**  класс описывающий фигурки. Fields: index – номер фигурки, matrix – матрица фигурки, r – кол-во поворотов, size – размер фигурки, x и y – координаты фигурки. Methods: countRotate – поворачивает фигурку на заданное кол-во, down – отпускает на одну клетку вниз, downMax – отпускает фигурку вниз пока это возможно, isCurrnetPos – возвращает false если у фгруки не корректная позиция, isMaxRigth – возращает true если у фигурки макс. координаты по x, land – приземляет фигруку в матрицу поля, left – шаг влево, moveToAngel – двигает фигурку в угол, right – шаг в право, rotate – поворот фигурки, up – шаг вверх, upMax – двигает фигурку максимально вверх,

**CFactory** – класс для случайного создания фигурок. Fields: brick\_1…brick\_7 – матрицы фигурок, space – пустая матрица. Methods: crateFigrue – возвращает случайную фигурку, maker – задает параметры для случайной фигурки.

**CGenetic** – класс определяющий всю работу генетического алгоритма для искусственного интеллекта. Fields: bestCondidate – лучшая ДНК для игры в тетрис, bestFigure – лучшая позиция для фигурки, countBirth – кол-во новой популяции, countDNA – кол-во особей, countEVO – колв-во эволюций, countGame – кол-во игр для тестирования особи, countMove – кол-во ходов для тестирования особи, GField- перменная класса CFeld, GFigure – перменная класса GFigure, populations – массив популяций ДНК. Methods: crossover – создает новую популяцию, evolution – начинает эфолюцию, fitness- подсчитывает пригодность каждой особи, generation – создает начальную популяцию, mutation – c вероятностью мутирует новорождённую особь. normalize – нормализует ДНК особи, seacrhMove – подбирает лучшую позицию фигурки, selection – выбирает две особи для создания новой, sort – сортирует особей по их пригодности. DNA – структура описывающая ДНК особи. На рисунке 5 можно увидеть диаграмму выше описанных классов.

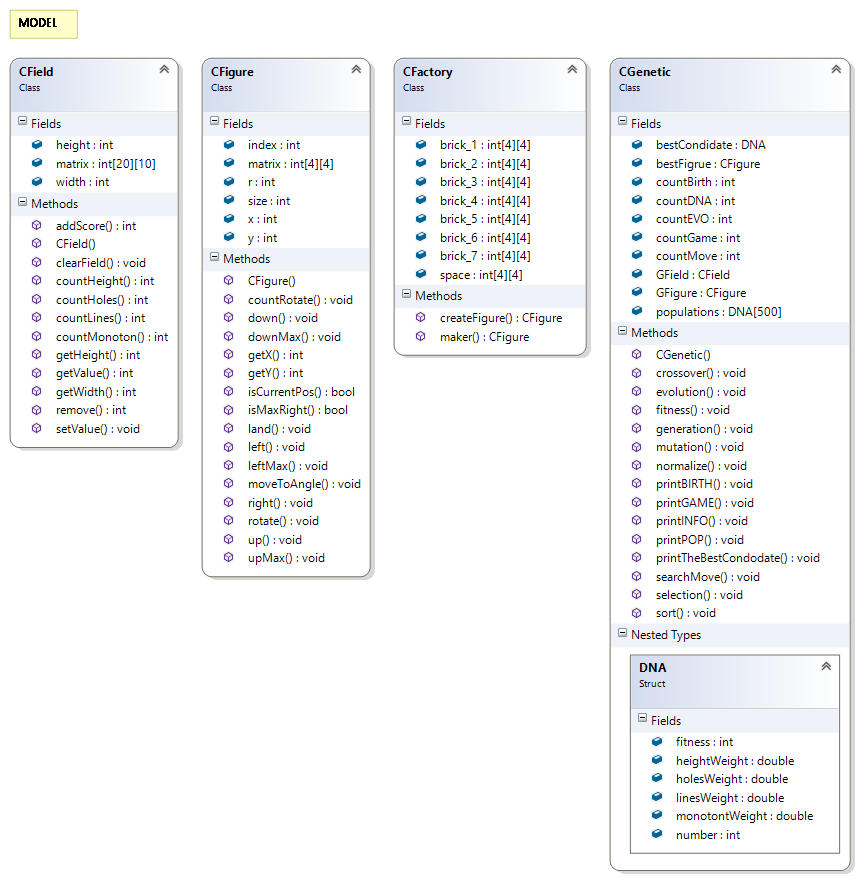


Рисунок – 5 Диаграмма MODEL

**CPrinter** – класс для отображения игры в консоли. Fields: hConsole – переменная для определения консоли, position – позиция курсора в консоли. Methods: printer – отображает игру в консоли.

**CWindowView** – класс для отображения игры в окне. Fields – sprites – спрайты изображений, textures – текстуры игры. Methods: checkCloseWindow – проверка закрытия окна, loader – загружает текстуры, moveBackGround – двигает фон окна, printField – риует поле игры, printFigure – рисует фигурку, setBackGround – устанавливает поле, showEvolution – показывает информацию при завершении эволюции ГА, showGame – показывает игру в зависимости от режима, showList – показывает список меню в зависимости от игры, showLoading – показывает окно загрузки, showMenu – показывает главное меню игры.

**CMediaPlayer** - класс с статическими методами для управлением музыкой и звуками. Methods: loader – загружает звуки и музыку, mute -вкл/выкл музыку. На рисунке 6 можно увидеть диаграмму выше описанных классов.

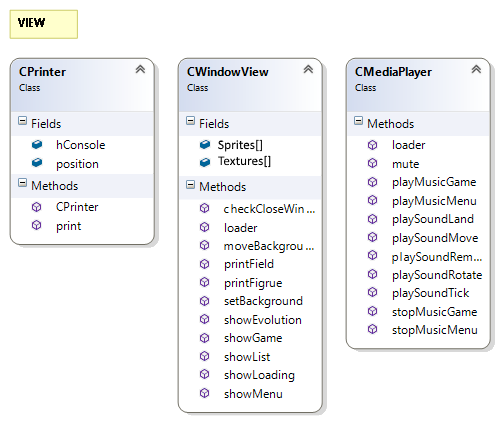


Рисунок – 6 Диаграмма VIEW

**CKeyController –** класс для управления. Methods: checkClick – проверяет был ли клик по заданному месту, exit – выход в главное меню, keyGame – проверка нажатий в игре в зависимости от режима, keyList - проверка нажатий в списках в зависимости от режима, keyMenu – проверка нажатий в главном меню, playerControl – проверка нажатия клавиш для управления фигуркой.

**CPlayer**  – класс описывающий игрока. Fields: field -поле игрока, figure – фигурка игрока, isGameOver – логическая переменная для обазначения конца игры, nextFIgure – следущая фигурка игрока, score – очки игрока. Methods: controller – управление для режима в консоли, initialize – задает начальные параметры игрока, step – делает один шаг.

**CComputer** – класс описывающий искусственный интеллект. Fields: AIBestFigure – лучшая позиция фигурки, AIGenetic – переменная класса CGenetic для запуска генетического алгоритма. Methods: AIController – передвигает фигурку к лучшей позиции, AIStartEvolution – запускает эволюцию, AISeacrhMove- ищет лучшую позицию для фигурки. На рисунке 7 можно увидеть диаграмму выше описанных классов.

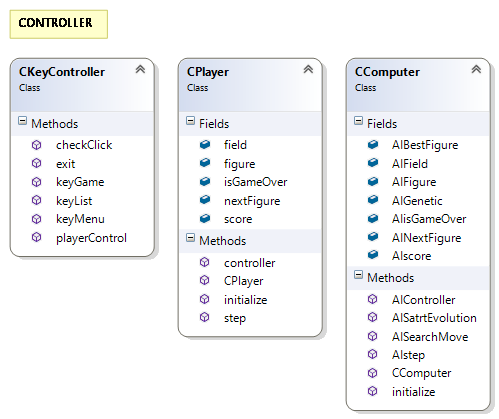


Рисунок – 7 Диаграмма CONTROLLER

* 1. **Примеры и описание основных функций**

Функция **seacrhMove** класса **CGenetic**:

void CGenetic::searchMove(DNA &candidate)

{

double bestScore = -9999;

double tmpScore = 0;

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

{

GFigure.x = 0;

GFigure.y = 0;

GFigure.countRotate(1);

GFigure.moveToAngle(GField);

while (true)

{

GFigure.downMax(GField);

CField landField = GField;

GFigure.land(landField);

candidate.fitness += landField.countLines();

tmpScore = (candidate.heightWeight \* landField.countHeight()) + (candidate.linesWeight \* landField.countLines()) +

(candidate.holesWeight \* landField.countHoles()) + (candidate.monotontWeight \* landField.countMonoton());

if (tmpScore > bestScore) {

bestScore = tmpScore;

bestFigrue = GFigure;

bestFigrue.r = i+1;

}

landField.clearField();

GFigure.upMax(GField);

GFigure.x++;

if (!GFigure.isMaxRight(GField))

break;

}

}

}

На вход функции приходит ДНК особи по величинам который мы будим подбирать лучший ход для падания фигурки, для каждого поворота фигурки мы перебираем все её возможные падения, каждое падение фигурки мы оцениваем с помощью ДНК особи. Перебирая падения фигурки с разными поворотами мы берем ту которая набрала самое большое кол-во очков.

Функция **isCurrentPos** класса **CFigure**

bool CFigure::isCurrentPos(CField &field)

{

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (matrix[i][j] == 1) {

if (y + i >= field.getHeight() || y + i < 0) return false;

if (field.matrix[i + y][x + j] >= 1 && field.matrix[i + y][x + j] <= 7) return false;

if (x+j < 0 || x+j > field.getWidth()-1) return false;

}

}

}

return true;

}

На вход функции поступает поле на котором находится вызываемая фигурка, функция проверяет матрицу фигурки и её координаты. Функция возвращает значение false если фигурка выходит за границы поля или пересекается с другими фигурками.

* 1. **Принцип работы алгоритма искусственного интеллекта**

Работа генетического алгоритма для тетриса заключается в том, чтобы оптимизировать веса для функции, которая оценивает то на сколько хорошо приземлилась фигурка. Наша цель найти лучший ход перебирая все возможные ходы. Для того чтобы оценить положение фигурки мы будет использовать следующую формулу: A B C D – это весы для оптимизации функции, числа от -1 до 1.(все веса кроме B должны быть отрицательными т.к countLine это положительное качество). Характеристики параметров:

**countHeight** – сумма максимальной высоты каждого столбца на поле, пример представлен на рисунке 8.



Рисунок – 8 Пример countHeight

**countLines** – кол-во заполненных линей на поле, пример представлен на рисунке 9.

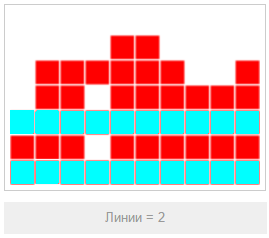


Рисунок – 9 Пример countLines

**countHoles** – кол-во дырок на поле, пример представлен на рисунке 10.



Рисунок – 10 Пример countHoles

**countMonoton** – кривизна поверхности поля. Пример подсчета: наглядный представлен на рисунке 11.



Рисунок – 11 Пример countMonoton

Особь в алгоритме представлять собой вектор ДНК с такими характеристиками как: fitness – оценка пригодности особи, number – номер особи, heightWeight – вес для countHeight (A из формулы для оценки), linesWeight – вес для countLines (B из формулы для оценки), holesWeight – вес для countHoles (C из формулы для оценки), monotonWeight – вес для countMonoton (D из формулы для оценки).

В начале работы алгоритма перед первой эволюцией создаются 30 особей со случайными весами -0.5 до +0.5. Затем мы считает пригодность каждой особи (функция **fitness**). После этого мы сортируем популяцию по пригодности и создаем новое поколение. Выбираем две особи родителя (**selection**), создаем новую особь(**crossover**) и с некоторой вероятностью мутируем её (**mutation**). Посоле этого выбираем лучшую особь, с весами которой наш искусственный интеллект будет подбирать и оценивать положение фигурок.

**Fitness**: для каждой особи мы проводим 5 обычных игр в тетрис (игра длится пока фигурка не окажется на вверху или не привесится лимит ходов, например 200). За каждой ход мы перебираем все возможные положения фигурки оценивая их нашей формулой оценки и используя веса данной особи. После перебора выбираем самое лучшее положение. И увеличиваем fitness фигурки на кол-во заполненных линий. Эти действия мы проводим для всех 5 игр.

**Crossover**: кол-во новых особей будем равно 30% кол-ва всего особей, после выбора родителей мы будет скрещивать их по такому правилу: , где p – это вектор(веса) новой особи, p1 и p2 – родители. После этого мы нормализуем веса новой особи. Таким веса новой особи будут находится между весами родителей. В конце мы берем все новые особи 30% и заменяем их на 30% худших особей, которые находятся в конце массива.

**Selection**: для выбора родителей мы будет брать 2 случайных обособив из первой половины массива особей, т.к. после сортировки вверху у нас будут самые лучшие особи.

**Mutation**: после рождения особь может мутироваться с вероятностью 0.1, во время мутации у особи берется случайный все и увеличивается или уменьшается на случайной число от 0 до 0.5.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

При запуске приложения запускается главное меню игры, скриншот на рисунке 12.



Рисунок – 12 Главное меню

1. «PLAY» - включает список режимов игры
2. «SETTING» - включает список настроек
3. «INFO» - включает список различной информации

При нажатие на спрайт 2 запускается окно настроек игры, скриншот на рисунке 13.

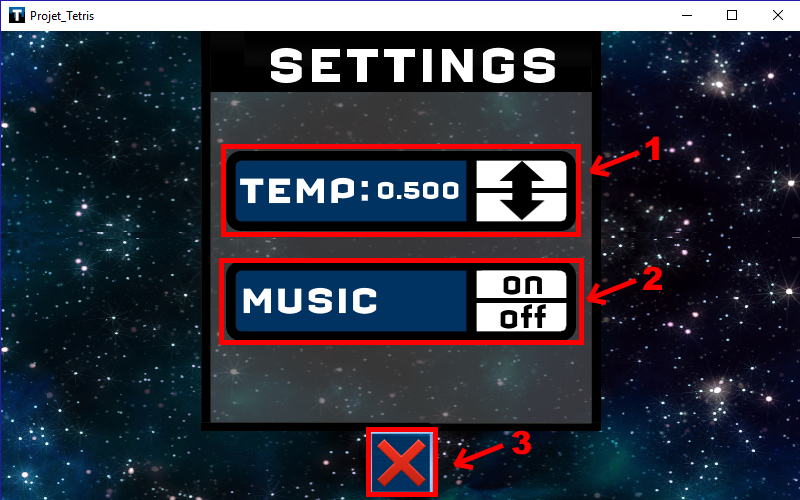


Рисунок – 13 Список настроек

1. «setTEMP» - настройка темпа игры
2. «MUSIC» - вкл/выкл музыки
3. «EXIT» - возврат в главное меню

При нажатии на спрайт 3 запускается окно инфомации, скриншот на рисунке 14.



Рисунок – 14 Список информации

1. «GIT» - ссылка на репозиторий с исходниками проекта
2. «CONTROL» - управление в игре
3. «EXIT» - возврат в главное меню

При нажатие на спрайт 1 запускается окно выбора режима игры, скриншот на рисунке 15.



Рисунок – 15 Список режимов игры

1. «1PLAYER» - режим для одного игрока
2. «2PLAYER» - режим для двух игроков
3. «AIvsPLAYER» - режим игры против искусственного интеллекта
4. «TRAININGAI» - тренировка искусственного интеллекта
5. «EXIT» - возврат в главное меню

Окно игры представлено на рисунке 16.

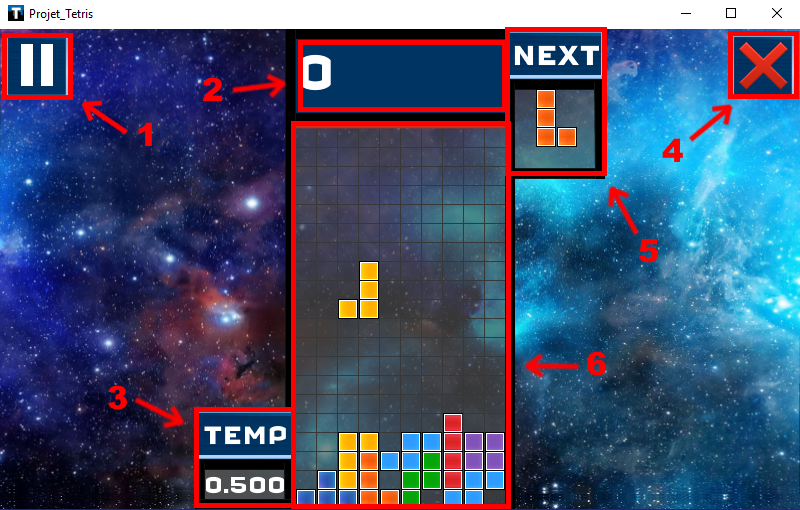


Рисунок – 16 Процесс игры

1. «PAUSE» - пауза
2. «SCORE» - кол-во очков
3. «TEMP» - текущий темп игры
4. «EXIT» - возврат в главное меню
5. «NEXT» - следующая фигурка
6. «FIELD» - поле игры

# **ЗАКЛЮЧЕНИЕ**

Генетические алгоритмы - реализация одной из наиболее популярных идей такого рода. Эта популярность вызвана, по-видимому, исключительной красотой подхода и его близостью к природному механизму. Подобным образом популярность нейросетевой технологии подогревается во многом ее сходством с работой мозга. По-настоящему активное развитие эвристических подходов, как мы видим, непосредственно связано с развитием свободного рынка и экономики в целом

Изложенный подход является эвристическим, т. е. показывает хорошие результаты на практике, но плохо поддается теоретическому исследованию и обоснованию. Естественно задать вопрос - следует ли пользоваться такими алгоритмами, не имеющими строгого математического обоснования? Как и в вопросе о нейронных сетях, здесь нельзя ответить однозначно. С одной стороны, в математике существует достаточно большой класс абсолютно надежных (в смысле гарантии получения точного решения) методов решения различных задач. С другой стороны, речь идет о действительно сложных практических задачах, в которых эти надежные методы часто неприменимы. Нередко эти задачи выглядят настолько необозримыми, что не предпринимается даже попыток их осмысленного решения.

В данной курсовой работе можно выделить такие достоинства как:

* Приятный пользовательский интерфейс;
* Грамотная структура классов;
* Разнообразность приложения;
* Работа искусственного интеллекта;

Таким образом были выполнены следующие выше поставленные цели:

* Разработано игровое приложение Тетрис;
* Разработан искусственный интеллект для игры;
* Освоены принципы работы генетических алгоритмов;
* Создан приятный пользовательский интерфейс;

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Айвор Хортон. Visual C++ 2010. Полный курс. : Пер. с англ. – М. : Издательский дом «Вильямс», 2012. – 624 с.
2. Литвиненко Н.А. технология программирования на С++. Win32 API-приложения. – СПб. : БХВ-Петербург, 2013. – 288 ил.
3. Сидорина Т.Л. Самоучитель Microsoft Visual Studio C++ и MFC. – СПб.: БХВ-Петербург, 2013. – 848 ил.
4. Глушаков С.В. Программирование на С++ /, Т.В. Дуравкина. – изд. 2-е, доп. и переработ. – М. : АСТ, 2015. – 685 с.
5. Шилдт Г. C++ базовый курс. : Пер. с англ. – М. : Издательский дом «Вильямс», 2012. – 624 с.
6. Павловская Т. А. C/C++. Структурное программирование : Практикум / Т. А. Павловская, Ю.А.Щупак. – СПб. : Питер, 2011. – 240 с.
7. Подбельский В.В. Язык С++: Учебное пособие. – 5-е изд. -М. : Финансы и статистика, 2015. -560 с.
8. Лафоре Р ОБЪЕКТНО - ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В C++ / – СПб. : Питер, 2016. – 240 с.
9. Гладков Л.А. Генетические алгоритмы / Курейчик В.В., Курейчик В.М – СПб. : Питер, 2014. – 240 с.

# **ПРИЛОЖЕНИЕ**

**Листинг исходного модуля**

**Class CGenetic**

#include "Player.h"

#include "stdafx.h"

#include "Game.h"

#include "Factory.h"

#include "Tetris.h"

#include "iostream"

#include <conio.h>

#include <windows.h>

#include "AI.h"

#include "Genetic.h"

#include "time.h"

#include "fstream"

using namespace std;

CGenetic::CGenetic(int width, int height) :

GField(width, height),

bestFigrue(width / 2 - 1, 0, 4, 0, CTetris::game.factory.space),

GFigure(width / 2 - 1, 0, 4, 0, CTetris::game.factory.space)

{

countDNA = 30;

countEVO = 3;

countGame = 5;

countMove = 200;

countBirth = countDNA / 3;

}

void CGenetic::evolution()

{

cout << "Genetic Algorithm start..." << endl;

generation();

for (int i = 0; i < countEVO; i++)

{

cout << "EVO." << i + 1 << endl;

fitness();

crossover();

}

cout << "Genetic Algorithm end..." << endl;

bestCondidate = populations[0];

printTheBestCondodate();

Beep(1000, 1000);

CTetris::waitPUSH();

CTetris::clearConsole();

}

void CGenetic::fitness()

{

for (int i = 0; i < countDNA; i++)

{

DNA &candidate = populations[i];

candidate.fitness = 0;

for (int j = 0; j < countGame; j++)

{

GField.clearField();

bool isGameOver = false;

int countLineScore = 0;

GFigure = CTetris::game.factory.createFigure();

int k = 0;

while (!isGameOver && k < countMove)

{

searchMove(candidate);

bestFigrue.land(GField);

countLineScore += GField.remove();

isGameOver = bestFigrue.getY() <= 0;

GFigure = CTetris::game.factory.createFigure();

k++;

}

}

}

}

void CGenetic::searchMove(DNA &candidate)

{

double bestScore = -9999;

double tmpScore = 0;

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

{

GFigure.x = 0;

GFigure.y = 0;

GFigure.countRotate(1);

GFigure.moveToAngle(GField);

while (true)

{

GFigure.downMax(GField);

CField landField = GField;

GFigure.land(landField);

candidate.fitness += landField.countLines();

tmpScore = (candidate.heightWeight \* landField.countHeight()) + (candidate.linesWeight \* landField.countLines()) +

(candidate.holesWeight \* landField.countHoles()) + (candidate.monotontWeight \* landField.countMonoton());

if (tmpScore > bestScore) {

bestScore = tmpScore;

bestFigrue = GFigure;

bestFigrue.r = i+1;

}

landField.clearField();

GFigure.upMax(GField);

GFigure.x++;

if (!GFigure.isMaxRight(GField))

break;

}

}

}

void CGenetic::selection(DNA parents[2])

{

DNA parent0 = populations[rand() % countDNA];

DNA parent1 = populations[rand() % countDNA];

for (int i = 0; i < countDNA; i++)

{

int numberGroup = rand() % 2;

if (numberGroup == 0)

{

if (populations[i].fitness > parent0.fitness)

parent0 = populations[i];

}

if (numberGroup == 1)

{

if (populations[i].fitness > parent1.fitness)

parent1 = populations[i];

}

}

parents[0] = parent0;

parents[1] = parent1;

}

void CGenetic::mutation(DNA &candidate)

{

int random = rand() % 10;

int parametr = rand() % 4;

double quantity = (double)(rand()) / RAND\_MAX - 0.5;

if (random == 1)

{

switch (parametr)

{

case 0: candidate.heightWeight = quantity; break;

case 1: candidate.linesWeight = quantity; break;

case 2: candidate.holesWeight = quantity; break;

case 3: candidate.monotontWeight = quantity; break;

}

}

}

void CGenetic::crossover()

{

DNA parents[2];

DNA birth;

for (int i = 0; i < countBirth; i++)

{

selection(parents);

birth.heightWeight = parents[0].fitness \* parents[0].heightWeight + parents[1].fitness \* parents[1].heightWeight;

birth.linesWeight = parents[0].fitness \* parents[0].linesWeight + parents[1].fitness \* parents[1].linesWeight;

birth.holesWeight = parents[0].fitness \* parents[0].holesWeight + parents[1].fitness \* parents[1].holesWeight;

birth.monotontWeight = parents[0].fitness \* parents[0].monotontWeight + parents[1].fitness \* parents[1].monotontWeight;

birth.fitness = 0;

mutation(birth);

normalize(birth);

sort();

birth.number = populations[countDNA - 1 - i].number;

populations[countDNA - 1 - i] = birth;

}

}

void CGenetic::sort()

{

for (int i = 0; i < countDNA-1; i++)

{

for (int j = i + 1; j < countDNA; j++)

{

if (populations[i].fitness < populations[j].fitness)

{

DNA tmp = populations[i];

populations[i] = populations[j];

populations[j] = tmp;

}

}

}

}

void CGenetic::normalize(DNA &candidate)

{

double norm = sqrt(pow(candidate.heightWeight, 2) +

pow(candidate.linesWeight, 2) +

pow(candidate.holesWeight, 2) +

pow(candidate.monotontWeight, 2));

candidate.heightWeight /= norm;

candidate.linesWeight /= norm;

candidate.holesWeight /= norm;

candidate.monotontWeight /= norm;

}

void CGenetic::generation()

{

srand((unsigned)time(NULL));

for (int i = 0; i < countDNA; i++)

{

populations[i].heightWeight = (double)(rand()) / RAND\_MAX - 0.5;

populations[i].linesWeight = (double)(rand()) / RAND\_MAX - 0.5;

populations[i].holesWeight = (double)(rand()) / RAND\_MAX - 0.5;

populations[i].monotontWeight = (double)(rand()) / RAND\_MAX - 0.5;

populations[i].fitness = 0;

populations[i].number = i + 1;

normalize(populations[i]);

}

}

void CGenetic::printGAME(CField &GField, CFigure &GFigure, int score)

{

CTetris::game.printer.print(GField, GFigure, bestFigrue, 0, score);

}

void CGenetic::printPOP()

{

cout << "POPULATIONS" << endl;

for (int i = 0; i < countDNA; i++)

{

cout << populations[i].number << ". | ";

cout << populations[i].heightWeight << " | ";

cout << populations[i].linesWeight << " | ";

cout << populations[i].holesWeight << " | ";

cout << populations[i].monotontWeight << " | ";

cout << "F = " << populations[i].fitness << endl;

}

cout << endl;

}

void CGenetic::printINFO(CField &GField, DNA &candidate, double &tmpScore, double &bestScore)

{

cout << "=========== INFO." << candidate.number << " ===========" << endl;

cout << "Whight: ";

cout << candidate.heightWeight << " | ";

cout << candidate.linesWeight << " | ";

cout << candidate.holesWeight << " | ";

cout << candidate.monotontWeight << endl;

cout << "Stage: ";

cout << GField.countHeight() << " | ";

cout << GField.countLines() << " | ";

cout << GField.countHoles() << " | ";

cout << GField.countMonoton() << endl;

cout << "Score: " << tmpScore << endl;

cout << "Fitness: " << candidate.fitness << endl;

cout << "Best score: " << bestScore << endl;

cout << "Best figrue x: " << bestFigrue.x << endl << endl;

}

void CGenetic::printBIRTH(DNA &birth, DNA parents[2])

{

cout << "Birth:" << endl;

cout << birth.number << ". | ";

cout << birth.heightWeight << " | ";

cout << birth.linesWeight << " | ";

cout << birth.holesWeight << " | ";

cout << birth.monotontWeight << " | ";

cout << "F = " << birth.fitness << endl;

cout << "Parent1:" << endl;

cout << parents[0].number << ". | ";

cout << parents[0].heightWeight << " | ";

cout << parents[0].linesWeight << " | ";

cout << parents[0].holesWeight << " | ";

cout << parents[0].monotontWeight << " | ";

cout << "F = " << parents[0].fitness << endl;

cout << "Parent2:" << endl;

cout << parents[1].number << ". | ";

cout << parents[1].heightWeight << " | ";

cout << parents[1].linesWeight << " | ";

cout << parents[1].holesWeight << " | ";

cout << parents[1].monotontWeight << " | ";

cout << "F = " << parents[1].fitness << endl << endl;

}

void CGenetic::printTheBestCondodate()

{

cout << "THE BEST | ";

cout << bestCondidate.number << ". | ";

cout << bestCondidate.heightWeight << " | ";

cout << bestCondidate.linesWeight << " | ";

cout << bestCondidate.holesWeight << " | ";

cout << bestCondidate.monotontWeight << " | ";

cout << "F = " << bestCondidate.fitness << endl;

}

**Class GField**

#include "stdafx.h"

#include "Game.h"

#include "Tetris.h"

#include "Field.h"

#include <windows.h>

#include <math.h>

#include <iostream>

CField::CField(int width\_c, int height\_c)

{

width = width\_c;

height = height\_c;

}

int CField::getWidth()

{

return width;

}

int CField::getHeight()

{

return height;

}

int CField::addScore(int countLine)

{

int score = 0;

if (countLine == 1)

score += 100;

else if (countLine == 2)

score += 300;

else if (countLine == 3)

score += 700;

else if (countLine == 4)

score += 1500;

return score;

}

void CField::getMatrix(int tmp[20][10])

{

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

tmp[i][j] = matrix[i][j];

}

}

}

int CField::getValue(int x, int y)

{

if (x >= 0 && x < width && y >= 0 && y < height)

return matrix[y][x];

}

void CField::setValue(int x, int y, int value)

{

if (x >= 0 && x < width && y >= 0 && y < height)

matrix[y][x] = value;

}

void CField::clearField()

{

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

if (matrix[i][j] != 0)

{

matrix[i][j] = 0;

}

}

}

}

int CField::remove()

{

int countLine = 0;

for (int i = 0; i < height; i++)

{

int count = 0;

for (int j = 0; j < width; j++) {

if (matrix[i][j] != 0)

count++;

}

if (count == width) {

countLine++;

for (int j = 0; j < width; j++) {

matrix[i][j] = 0;

}

for (int j = i; j > 0; j--) {

for (int k = 0; k < width; k++) {

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

}

}

}

}

return addScore(countLine);

}

int CField::countHeight()

{

int count = 0;

for (int i = 0; i < width; i++)

{

int j = 0;

while (matrix[j][i] == 0 && j < height)

j++;

count += height - j;

}

return count;

}

int CField::countLines()

{

int countLine = 0;

for (int i = 0; i < height; i++)

{

int countB = 0;

for (int j = 0; j < width; j++) {

if (matrix[i][j] != 0)

countB++;

}

if (countB == width) {

countLine++;

}

}

return countLine;

}

int CField::countHoles()

{

int count = 0;

for (int i = 0; i < width; i++)

{

for (int j = 0; j < height; j++)

{

if (matrix[j][i] != 0)

{

while (j < height)

{

if (matrix[j][i] == 0)

count++;

j++;

}

}

}

}

return count;

}

int CField::countMonoton()

{

int count = 0;

int mas[10];

for (int i = 0; i < width; i++)

{

int j = 0;

while (matrix[j][i] == 0 && j < height)

j++;

mas[i] = height - j;

}

for (int i = 0; i < width-1; i++)

{

count += abs(mas[i] - mas[i + 1]);

}

return count;

}

**Class CFigure**

#include "stdafx.h"

#include "iostream"

#include "Figure.h"

#include "Tetris.h"

#include "windows.h"

using namespace std;

CFigure::CFigure(int x\_c, int y\_c, int size\_c, int index\_c, int matrix\_c[4][4])

{

x = x\_c;

y = y\_c;

size = size\_c;

index = index\_c;

r = 0;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

matrix[i][j] = matrix\_c[i][j];

}

}

}

int CFigure::getX()

{

return x;

}

int CFigure::getY()

{

return y;

}

void CFigure::getMatrix(int tmp[4][4])

{

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

tmp[i][j] = matrix[i][j];

}

}

}

void CFigure::up()

{

y--;

}

void CFigure::right(CField &field)

{

x++;

if (!isCurrentPos(field))

x--;

}

void CFigure::left(CField &field)

{

x--;

if (!isCurrentPos(field))

x++;

}

void CFigure::down()

{

y++;

}

void CFigure::downMax(CField &field)

{

while (isCurrentPos(field))

y++;

y--;

}

void CFigure::upMax(CField & field)

{

while (isCurrentPos(field))

y--;

y++;

}

void CFigure::leftMax(CField & field)

{

while (isCurrentPos(field))

x--;

x++;

}

void CFigure::rotate(CField &field)

{

r++;

int tmp[4][4];

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

tmp[i][j] = matrix[i][j];

}

}

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

matrix[j][size - 1 - i] = tmp[i][j];

}

}

if (!isCurrentPos(field)) {

r--;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

matrix[i][j] = tmp[i][j];

}

}

}

}

void CFigure::land(CField &field)

{

if (isCurrentPos(field))

{

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (matrix[i][j] == 1) {

field.matrix[i + y][j + x] = index;

}

}

}

}

}

void CFigure::moveToAngle(CField & field)

{

upMax(field);

leftMax(field);

}

bool CFigure::isCurrentPos(CField &field)

{

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (matrix[i][j] == 1) {

if (y + i >= field.getHeight() || y + i < 0) return false;

if (field.matrix[i + y][x + j] >= 1 && field.matrix[i + y][x + j] <= 7) return false;

if (x+j < 0 || x+j > field.getWidth()-1) return false;

}

}

}

return true;

}

bool CFigure::isMaxRight(CField & field)

{

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (matrix[i][j] == 1) {

if (x + j < 0 || x + j > field.getWidth() - 1) return false;

}

}

}

return true;

}