# ДОКУМЕНТАЦИЯ

# АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

* 1. Анализ прототипов

Для успешной реализации программного средства необходимо ознакомиться с существующими аналогами в данной сфере. Анализ их достоинств и недостатков позволяет сформировать требования к проектируемому средству, учитывающие опыт существующих разработок, и внести в них улучшения или изменения. В качестве исследуемых аналогов были выбраны следующие программные продукты: Classic Tetris, Electronic Arts Tetris®, HotFix Tetris.

Classic Tetris – одна из ранних версий Тетриса . Пользовательский интерфейс приложения представлен на рисунке 1.1.1. На данный момент игра представлена веб-версией и приложениями для различных операционных систем. Отличительной особенностью Classic Tetris является простота и близость к первоначальной задумке игры . К достоинствам данного приложения можно отнести быстродействие, универсальность и классическое музыкальное сопровождение. [Classic](mailto:Почта@Mail.ru) Tetris спроектирован на языке Assembler.

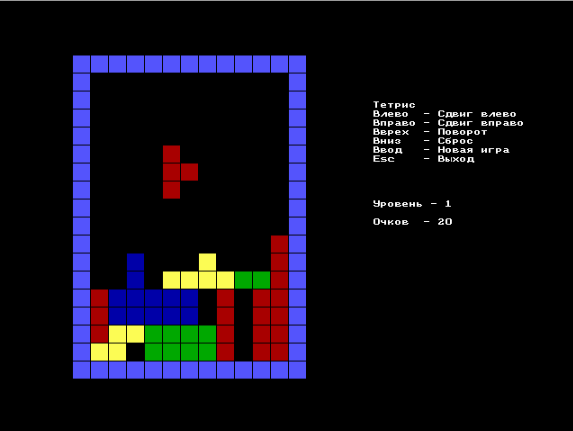


Рисунок 1.1.1 – Скриншот интерфейса Classic Tetris

Electronic Arts Tetris® – на данный момент самый популярный экземпляр Тетриса. Интерфейс программного средства представлен на рисунке 1.1.2. Достоинствами данного сервиса являются продуманная аркадность, современная графика и стилистика, а так же немалое количество режимов игры, включая сетевой режим на двоих. Помимо вышеперечисленных достоинств данное приложение имеет один минус в виде переработанной системы начисления очков – начисление происходит в случае любой активности на сетке (в классическом Тетрисе очки начисляются исключительно за составления 10 колонок стакана в ряд ).

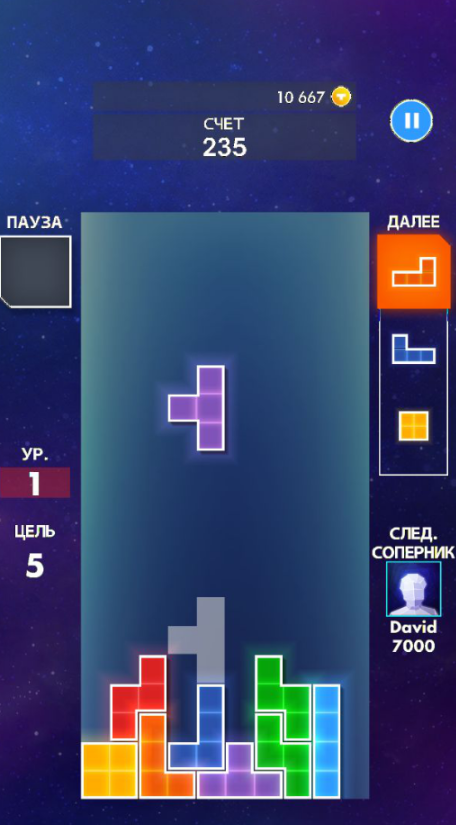


Рисунок 1.1.2 – Скриншот интерфейса программы Electronic Arts Tetris®

HotFix Tetris – игровое приложение, обладающий всеми необходимыми базовыми функциями: классический и соревновательный режимы, сетевая игра. Пользовательский интерфейс программы представлен на рисунке 1.1.3. Достоинства данного программного средства: соблюдение классического дизайна, соревнования в сетевом режиме в реальном времени, гибкая настройка управления.



Рисунок 1.1.3 – Скриншот интерфейса программы HotFix Tetris

* 1. Анализ литературных источников

При написании программного средства была выбрана платформа .NET Framework и язык программирования C#. В печатном издании Герберта Шилдта “C# 4.0 Полное руководство” детально рассмотрены все основные средства языка [1]. В книге подробно описываются возможности C#, даются профессиональные рекомендации и приводятся сотни примеров программ, охватывающие все аспекты программирования на C#. К недостаткам этой книги можно отнести часто встречающиеся синтаксические ошибки, связанные с проблемами перевода оригинального издания, как в коде программ, так и в объяснениях автора.

В книге Э.Джонса и Д.Оланда “Программирование в сетях Windows” [2] детально описан интерфейс прикладного программирования Winsock, рассмотрены семейства адресов, включая протоколы IP, IPX, NetBIOS, AppleTalk и ATM. Основным достоинством данной книги является подробное описание сетевой модели OSI и клиент-серверной архитектуры.

В качестве протокола передачи данных в разрабатываемом ПС используется UDP (User Datagram Protocol). В электронном ресурсе metanit.com представлены основные концепции работы с сетями на платформе .NET [3]. В разделе “работы с сетями в C# и .NET” продемонстрированы основные классы и методы, используемые для отправки дейтаграм на языке C#. Основным достоинством данного ресурса является практическое руководство по применению описанных классов, а также удобная навигационная панель, позволяющая быстро переключаться между разделами.

Для разработки пользовательского интерфейса выбрана подсистема WF (Windows Forms), которая является частью .NET Framework 4.7.2. Электронный ресурс metanit.com является одним из лучших для изучения данной технологии [4]. В нем подробно рассмотрены основные концепции языка XAML: элементы управления, свойства зависимостей, команды, кисти, ресурсы, привязка, триггеры, документы, работа с графикой и другие.

Печатное издание “Совершенный код” Стива Макконнелла является практическим руководством по разработке программного обеспечения [5]. В

данной книге обсуждаются такие детали конструирования, как этапы создания классов, использование данных и управляющих структур, отладка, рефакторинг, написание самодокументирующегося кода, методики оптимизации и характеристики качества программы. В конце каждого раздела представлен список контрольных вопросов, позволяющий оценить качество архитектуры ПС, подход к проектированию, качество классов и методов, имена переменных и многое другое. Описанные в книге методики конструирования не привязаны к какому-либо конкретному языку программирования или платформе, но эффективнее всего использовать их при написании кода на C++, JAVA, C# и Microsoft Visual Basic. Издание содержит почти 500 примеров хорошего и плохого кода, позволяющих читателю не только правильно применять алгоритмы, но и избегать часто встречающихся ошибок.

Веб-сайт msdn.microsoft.com представляет собой документацию для разработчиков, использующих программные продукты компании Microsoft. В разделе .NET есть подробное описание всех инструкций и особенностей их поведения [6]. Преимущества данного ресурса: наличие большого количества примеров, посвященных каждой теме, все описание излагается на английском языке, что не приводит к искажению оригинальной информации, удобная навигационная панель, позволяющая, например, выбрав какой-либо класс, просмотреть все его доступные методы и свойства.

* 1. Формирование требований к проектируемому программному средству
     1. **Назначение разработки**

Функциональным назначением программного средства является предоставление возможности организации соревнований между двумя людьми. В качестве конечного пользователя может выступать любой клиент.

* + 1. **Состав выполняемых функций**

Программное средство должно обеспечивать возможность выполнения следующих функций:

* функция «режим одиночной игры»;
* функция «режим соревнования»;
* функция передачи данных;
* функция ведения рекордов игрока.
  + 1. **Входные данные**

Входными данными для программного средства в режиме соревнования являются: сетка фигурок, отправленная противником и его счёт.

* + 1. **Выходные данные**

В качестве выходных данных выступает падающая фигурка, отправленная на компьютер противника, счёт, а также сетка уже установленных игроком фигурок – для режима соревнования, для режима одиночной игры выходными данными является сетка закрепленных фигурок, а так же счёт игры.

* + 1. **Обоснование выбора языка и сред разработки**

Язык программирования C# лежит в основе разработки программного средства, целевой платформой которой является .NET Framework 4.7.2. Данный язык предоставляет набор библиотек, позволяющих формировать игровые события, взаимодействие с удаленным компьютером, а так же широкие возможности в графике даже в стандартных библиотеках Windows Forms. Достоинствами C# являются простота использования, объектная ориентированность, “сборка мусора”, поддержка совместимости версий и многое другое. Унифицирование системы типов позволяет рассматривать каждый тип как объект. Таким образом, используя класс, структуру, массив или встроенный тип, обращаться к ним можно как к объекту. Все объекты собраны в пространства имен (namespaces). Это позволяет вместо списка подключаемых файлов указывать, какие пространства имен, для доступа к объектам и классам внутри них, будут использованы в программе.  Например, пространство имен System содержит несколько классов, в том числе и Console. Достаточно писать либо название пространства имен перед каждым обращением к классу, либо использовать ключевое слово using. Стоит также отметить, что при создании языка C#, его авторы учитывали достижения многих других языков программирования: C++, C, Java, SmallTalk, Delphi, Visual Basic и т.д.

Для разработки графического интерфейса программы используется класс Windows Forms, который является основополагающей частью приложений в языке C#. Графической технологией является  [GDI](https://ru.wikipedia.org/wiki/GDI)/[GDI+](https://ru.wikipedia.org/wiki/GDI%2B).

1. АНАЛИЗ ТРЕБОВАНИЙ К ПС И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ
   1. Описание функциональности ПС

Пользователь имеет следующие возможности:

1. Найти игрока для игры:
   * Игрок ищется в автоматическом режиме: если кто-либо еще в локальной сети запустит игру, то установится флаг видимости заданного порта.
   * В случае отсутствия игроков в сети пользователю будет предложено провести игру в одиночном режиме.
2. Сохранить свой результат в таблицу рекордов, либо просмотреть существующие по окончанию игрового сеанса.
3. Играть в одиночном режиме.
   1. **Спецификация функциональных требований**

Ниже приведен полный список функций, предоставляемых клиенту. Пользователь должен иметь следующие возможности:

* + 1. Вводить своё имя после завершения игрового сеанса;
    2. Управлять падающими фигурками;
    3. Ставить игру на паузу и возобновлять;
    4. Включать/выключать режим соревнования;
    5. Удалять рекорды из таблицы;
    6. Просматривать рекорды, установленные в соревнованиях, а также в одиночных играх;
    7. Играть в сетевом режиме.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

* 1. Разработка алгоритмов ПС и отдельных модулей

На рисунке 3.1.1 представлена схема алгоритма работы программного средства, а ниже – ее словесное описание.



Рисунок 3.1.1 – Схема работы программы

Программа начинается с инициализации игровой модели: формируются сетки фигурок, массив начальных фигурок, события игровой модели, таймер и т.д., после чего производится отрисовка графического интерфейса. После отрисовки запускается поток поиска игроков, периодически отправляя запросы используя широковещательные пакеты. Если игрок найден, то ему отправляется сгенерированный порт, на который в последствии будут отправляться дейтаграммы с игровым событием, в свою очередь удаленный игрок, получив порт с идентификатором(идентификатором выступает значение, определяющее, что прислал другой игрок), второй повторяет действия первого, генерируя порт и высылая его. Игровой цикл запускается. По истечению интервала таймера игрок получает данные о игровой сессии и состояние игры на экране обновляется, если игра происходит по сети, данные сериализуются и отправляются второму игроку. В случае ошибки клиент получит соответствующее сообщение, иначе – поле соперника обновится. Если сетка фигурок заполнена, игра завершается. Программное средство устойчиво к любым действиям со стороны пользователя, соответственно последний может играть пока не получит сообщение о поражении, либо победе.

Метод ExtractPiece() предназначена для генерации фигурки по числу в диапазоне [0,6]. В качестве параметров функция принимает число. Предварительно в класс этого метода передается массив полученных от другого игрока чисел, либо, если игра одиночная, сгенерированного на фазе инициализации игрового модели. Метод возвращает объект фигурки таким образом:

public Piece ExtractPiece()

{

Piece currentPiece = null;

switch (Pieces[tempPiece])

{

case 0:

currentPiece = new PieceT();

break;

case 1:

currentPiece = new PieceO();

break;

. . .

case 6:

currentPiece = new PieceZ();

break;

}

return currentPiece;

}

Подпрограмма TetrisGrid() создает и инициализирует сетку фигурок. Изначально создается два обьекта: сетка локального клиента, и удаленного. После, сетки очищаются следующим образом:

public void EmptyGrid()

{

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

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

grid[i, j] = 0;

}

Также метод EmptyGrid() используется в реиницилизации игровой сессии:

public void InitializeGame()

{

currentPiece = null;

nextPiece = null;

g1.EmptyGrid();

g1.EmptyGrid2();

inPlay = true;

Score = 0;

Difficulty = 4;

ExtractPieces();

}

Набор bool методов CanMove(),FixPiece(),CanRotate() служит для определения возможности движения фигурки в заданном направлении, закрепляет на сетке и определяет, можно ли повернуть фигурку в заданных координатах соответственно. Первый метод сравнивает координаты фигурки с координатами сетки:

for (int i = y, row = 0; i < (y + RowsCurrentPiece); i++, row++)

for (int j = x, column = 0; j < (x + ColumnsCurrentPiece); j++, column++)

if ((g1.Grid1[i, j] != 0) && (currentPiece.Pattern[row, column] !=0))

canMove = false;

Второй метод проверяет, не заняты ли координаты требуемого места и располагает фигурку на сетке:

for (int i = currentPiece.Y, rowPiece = 0; i < (currentPiece.Y + RowsCurrentPiece); i++, rowPiece++)

for (int j = currentPiece.X, columnPiece = 0; (j < (currentPiece.X + ColumnsCurrentPiece)); j++, columnPiece++)

if (currentPiece.Pattern[rowPiece, columnPiece] != 0)

g1.Grid1[i, j] = currentPiece.Pattern[rowPiece, columnPiece];

Третий метод определяет, не находится ли фигурка у края сетки, либо возле зафиксированного блока:

int[,] rotationTemp = currentPiece.NextRotation(currentPiece.Rotation);

// обновление х и у вращения

int xNext = currentPiece.XNextRotation(),

yNext = currentPiece.YNextRotation();

// Проверяет, выходит ли следующее вращение из сетки

if ((yNext + rotationTemp.GetLength(0)) > g1.Grid1.GetLength(0))

canRotate = false;

if ((xNext + rotationTemp.GetLength(1)) > g1.Grid1.GetLength(1))

canRotate = false;

if (xNext < 0)

canRotate = false;

int i, j;

if (canRotate)

for (i = yNext; (i < (yNext + rotationTemp.GetLength(0))); i++)

for (j = xNext; (j < (xNext + rotationTemp.GetLength(1))); j++)

if (g1.Grid1[i, j] != 0)

canRotate = false;

Метод MoveDown() является основополагающим методом в игровом цикле. Он осуществляет движение фигурки вниз при истечении интервала таймера, либо при нажатии стрелки вниз. Его схема представлена на рисунке 3.1.2.



Рисунок 3.1.2 – блок-схема алгоритма движения фигурки вниз

Методы DrawPiece(), DrawNextPiece(), DrawBlocks() перерисовывают текущий блок, сетку и провайдер(область с подсказкой о следующей фигурке). Метод DrawPiece() создает область, ограниченную размерами фигурки и заданную её координатами, а после – рисует её на сетке:

pieceContainer = new Rectangle(p.X \* blockSize,

p.Y \* blockSize,

p.Sprite.Width,

p.Sprite.Height);

e.Graphics.DrawImage(p.Sprite, p.X \* 25, p.Y \* 25);

Метод DrawNextPiece() обновляет фигуру, отображаемую в боковой панели. Замечание: блок «I» имеет нестандартный по высоте размер, а значит, не поместится в области, выделенной для отрисовки, поэтому был предусмотрен вариант выбора координаты высоты области:

e.Graphics.DrawImage(p.Sprite,

p.Name=='I'? 290:280,

50);

Метод DrawBlocks() занимается перерисовкой сетки фигур. Вызывается после каждой «Фиксации» фигуры на сетке. Сетка поделена на области по 25х25 пикселей. Метод перебирает массив-сетку и по каждому значению перерисовывает фигуру:

switch (p.Name)

{

case 'I':

e.Graphics.DrawImage(pieceI\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'T':

e.Graphics.DrawImage(pieceT\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

. . .

case 'S':

e.Graphics.DrawImage(pieceS\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

}

e.Graphics.DrawImage(p.Sprite, p.X \* 25 + secondgridmove, p.Y \* 25);

Так же стоит упомянуть, что этот метод дублирован для сетки соперника.

Метод SendRequest() посылает запрос на подключение к удаленному игроку, в дейтаграмме запроса содержится ключевое слово и порт(сгенерированный случайно), по которому слушающий определит, следует ли доверять запросу. В качестве клиента используется UDPclient. Запрос посылается широковещательным пакетом:

ReceiveGridPort = rand.Next(55000, 56000);

try

{

UdpSender = new UdpClient(UdpSendPort, AddressFamily.InterNetwork);

ipUserEndPoint = new IPEndPoint(IpAdressBroadcast,UdpSendPort);

byte[] KeyBytes = Encoding.ASCII.GetBytes(key+ ReceiveGridPort.ToString());

int sendedData = UdpSender.Send(KeyBytes, KeyBytes.Length, ipUserEndPoint);

UdpSender.Close();

}

catch

{

На рисунке 3.1.3 представлена схема алгоритма, реализующего метода GetIpAddressFromUser(). Данная функция вызывается на фазе инициализации программы, работает в дочернем потоке и завершается сразу после получения адреса соперника. Также этот метод ожидает получения массива фигур. В случае, если перед полученными данными находится идентификатор ‘P’, происходит парсинг строки на массив:

Byte[] data = UdpListener.Receive(ref ClientEndPoint);

string GetKey = Encoding.ASCII.GetString(data);

if (GetKey[0] == 'P')

{

string str = GetKey.Substring(1, GetKey.Length - 1);

for (int i = 0; i < Pieces.Length; i++)

try

{

Pieces[i] = int.Parse(str[i].ToString());

}

catch

{

}



Рисунок 3.1.3 – Блок-схема метода GetIpAddressFromUser()

Метод GetGridUdp() слушает ранее полученный порт в ожидании получить сетку фигур. В случае её получения (об этом свидетельствует идентификатор ‘S’) метод десериализует ранее сериализованный массив с помощью JsonConverter:

if (Message.Substring(0, 1) == "S")

{

Score = int.Parse(Message.Substring(1, Message.Length - 1));

}

else

{

int[,] getgrid = JsonConvert.DeserializeAnonymousType<int[,]>(Message, grid.Grid2);

grid.Grid2 = getgrid;

}

* 1. Описание классов и методов

Все реализованные в программе классы обладают хорошей инкапсуляцией, которая, в отличие от абстракции, не позволяет узнать детали реализации.

Методы, реализованные в классах программы, позволяют повысить удобочитаемость кода, его надежность, а также облегчить изменение. Имена методов полностью описывают все выходные данные и побочные эффекты.

Класс Game содержит следующие методы и свойства:

* public TetrisGrid g1 – свойство, инициализирующее сетку фигур;
* private PieceProvider provider – свойство, возвращающее текущую фигуру;
* private Piece currentPiece, nextPiece – возвращают падающую фигуру и следующую;
* public void InitializeGame() – инициализирует игровой сеанс;
* private bool CanMove(int x, int y) – проверяет можно ли двигаться фигуре влево и/или вправо;
* private void FixPiece() – фиксирует падающую фигуру на сетке;
* private bool CanRotate() – проверяет, можно ли повернуть фигуру;
* public void MoveLeft() – метод перемещения фигуры влево;
* public void MoveRight() – метод перемещения фигуры вправо;
* public bool MoveDown() – метод перемещения фигуры вниз;
* public bool RotatePiece() – метод поворота фигуры;
* private bool CheckCompletedLine() – проверяет собранные строки.

Класс UdpListenSend обеспечивает работу приложения с сетью, позволяя пользователю играть в соревновательном режиме. Класс реализует следующие методы:

* public void SendRequest() – отправляет широковещательный запрос на подключение к другому игроку;
* public void GetIpAddressFromUser(bool Start) – ожидает запроса на проведение совместной игры, а также отправляет ответ на запрос в виде порта и/или массива фигур (зависит от первоочередности инициализации игровой модели);
* public void SendGrid(TetrisGrid grid) – посылает текущее состояния сетки фигур, используя JSON сериализацию;
* public void GetGridUdp(TetrisGrid grid) – принимает дейтаграммы с сеткой фигур от второго игрока. Работает в отдельном потоке. Поток запускается только в случае получения адреса соперника;
* public void SendPieces() – посылает массив сгенерированных фигур;

# СОЗДАНИЕ (КОНСТРУИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА

* 1. Конструирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемого ПС должен содержать такие возможности как: приостановить/возобновить процесс игры, отображать игровую сетку фигур и следующую падающую фигуру. Также должны быть предусмотрены события при сворачивании игры, а именно приостановка игрового процесса. Важную роль в восприятии любого игрового приложения играет дизайн, поэтому ему должно быть уделено должное внимание. В разрабатываемом ПС следует учесть контраст фигурок, оформление основной формы и дизайну кнопок.

После соблюдения всех требований к ПС было спроектировано приложение Tetris, его внешний вид представлен на рисунке 4.1.1.



Рисунок 4.1.1 – интерфейс программного средства

* 1. Конструирование контроллера ПС

Для управления игровым процессом были использованы события KeyEventHandler и EventHandler. В контроллере содержится 4 группы событий, происходящие при: нажатии клавиш, обновлении формы, нажатии на кнопки и потери/приобретения фокуса на приложение.

Описание событий:

* Tetris\_Form\_KeyDown – выполняется при нажатии клавиши. Отслеживаются клавиши стрелка вниз, верх, вправо и влево. Событие определяет, какая именно клавиша была нажата и вызывает соответствующий ей метод.
* Tetris\_Form\_ShowRanking – выполняется при нажатии на кнопку «Show high scores». После нажатия вызывается форма в виде модального окна, в которую выводится список установленных рекордов игрока.
* Tetris\_Form\_PauseResumeMinimize – вызывается при изменении размеров игровой формы, в таком случае игра продолжаться не может, а значит, её необходимо приостановить. Вызывается метод, приостанавливающий игровой процесс.
* Tetris\_Form\_PauseResumeGotLostFocus – вызывается, если окно с игрой было свернуто, либо развернуто пользователем, либо потеряло фокус.
* Tetris\_Form\_Draw – вызывается по истечению интервала таймера. В нем вызываются методы перерисовки обновления игрового состояния.
* Tetris\_Form\_PauseResume – вызывается при нажатии на кнопку паузы, либо кнопку возобновления игры.

# ТЕСТИРОВАНИЕ И ОБЕСПЕЧЕНИЕ КАЧЕСТВА

* 1. Тестирование и анализ полученных данных

Тестирование является самой популярной методикой повышения качества ПО. Существует множество видов тестирования, однако работа данного программного средства проверялась на корректность с помощью функционального метода. Данный подход используется с целью проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям.

Таблица 1 – Тестирование функциональных требований

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № тест-кейса | № тестируемой  функции | Тестируемая функциональность | Последовательность действий | Ожидаемый результат |
| 1 | 2.2.1 | Набор своего игрового имени. | Ввести имя, нажать на клавишу «Ок». | Отображение обновленной таблицы рекордов. |
| 2 | 2.2.2 | Управление падающими фигурами | Нажатие на клавиатуре стрелок вверх, вниз, вправо, влево. | Фигура движется в заданном пользователем направлением, а при нажатии «вверх» меняет ориентацию. |
| 3 | 2.2.3 | Приостановка игрового процесса. | Нажать на клавишу Pause-Resume. | В случае игры – остановить игру, в случае паузы – возобновить. |
| 4 | 2.2.4 | Включение/выключение режима соревнования. | Play Offline. | В случае игры с удаленным игроком – разорвать соединение и продолжить играть в одиночном режиме. |
| 5 | 2.2.5 | Удалить рекорды из таблицы рекордов. | Нажать кнопку “Show High scores”, затем нажать “Delete” . | Таблица и сериализованный файл с данными в корневой папке игры очиститься. |
| 6 | 2.2.6 | Отображение текущих рекордов. | Нажать кнопку “Show High scores”. | Отобразится модальное окно с таблицей рекордов. |
| 7 | 2.2.7 | Игра по сети. | Запустить игру и ожидать подключения. | Если игрок найден – игра предложит нажать кнопку “Play”, если нет – кнопку “Play Offline”. |

Успешное прохождение тестов доказывает устойчивость программного средства к любым действиям со стороны пользователя. Программа уведомит клиента при отсутствии подключения к интернету, а также при попытке ввода некорректных входных данных.

* 1. Обеспечение качества

Качество ПО – это систематичная программа действий, призванная гарантировать, что система обладает желательными характеристиками. Контроль качества осуществляется на протяжении всего процесса разработки. Качество ПО имеет внешние и внутренние характеристики. К внешним относятся свойства, которые осознает пользователь: корректность, практичность, эффективность, надежность, правильность, целостность. К внутренним характеристикам относятся удобство сопровождения, гибкость, портируемость, удобочитаемость и тестируемость. Ниже описаны основные концепции, принятые для повышения качества ПС.

Архитектура – это высокоуровневая часть приложения. Для доказательства того, что спроектированная архитектура ПС является качественной, приведен следующий список тезисов:

* Определены основные компоненты программы, их области ответственности и взаимодействие с другими компонентами;
* Все функции, указанные в спецификации требований, реализуются не слишком большим и не слишком малым числом компонентов;
* Приведено описание всех классов и их обоснование;
* Приведено описание организации данных и ее обоснование;
* Пользовательский интерфейс является модульным;
* Приведено описание стратегий ввода-вывода данных и ее обоснование;
* Определена согласованная стратегия обработки ошибок;
* Определен подход к реализации избыточной функциональности.

Класс – это набор данных и методов, имеющих общую целостность, хорошо определенную сферу ответственности. Интерфейс высококачественного класса должен представлять хорошую абстракцию, скрывающую детали реализации, группу методов, четко согласующихся друг с другом. Большинство методов имеют соответствующие противоположные им методы. Например, одна операция добавляет файл в список вложений, другая удаляет, одна создает адресную книгу, другая удаляет и т.д. Следующий список тезисов доказывает качественность реализованных в программе классов:

* Интерфейсы классов формируют согласованную абстракцию;
* Интерфейсы классов ясно описывают их использование;
* При изменении классов целостность интерфейсов сохраняется;
* Члены классов сделаны минимально доступными;
* Классы являются независимыми друг от друга;
* Классы не содержат более семи элементов данных-членов;
* Все данные-члены инициализируются в конструкторе.

Метод – это отдельная функция или процедура, выполняющая одну задачу. Следующий список утверждений подтверждает то, что методы в программе являются качественными:

* имена методов описывают все выполняемые операции;
* методы имеют высокую функциональную связность;
* методы выполняют только ту задачу, для которой были созданы;
* длина методов обусловлена их ролью и логикой, а не искусственным стандартом кодирования;
* список параметров метода формирует согласованную абстракцию интерфейса;
* методы принимают не более семи параметров;
* методы используют все входные параметры;
* вызывающая программа использует все выходные параметры методов;
* если методы являются функциями, они возвращают корректные значения во всех возможных случаях.
  1. Повышение производительности

В данном подразделе приведены методы, использованные для оптимизации и повышения производительности программы:

* использование целых чисел вместо чисел с плавающей точкой где это возможно. Математические операции над целочисленными переменными выполняются быстрее.
* Вычисление результатов предварительно. Такой подход позволяет вычислить результат выражения до выполнения программы и связать его с константой во время компиляции.
* Остановка циклов сразу после получения ответа.
* Размыкание циклов. Если во время выполнения цикла решение не изменяется, его нужно разомкнуть, приняв решение вне цикла. Как правило для этого достаточно поместить циклы в условный оператор, а не наоборот.
* Развертывание циклов. Целью данного метода является сокращение затрат, связанных с выполнением циклов. При каждой итерации циклы могут обрабатывать не один случай, а два и более.

# РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

При запуске программы появляется главная форма, представленная на рисунке 6.1, содержащая четыре кнопки, которые позволят пользователю взаимодействовать с игрой.

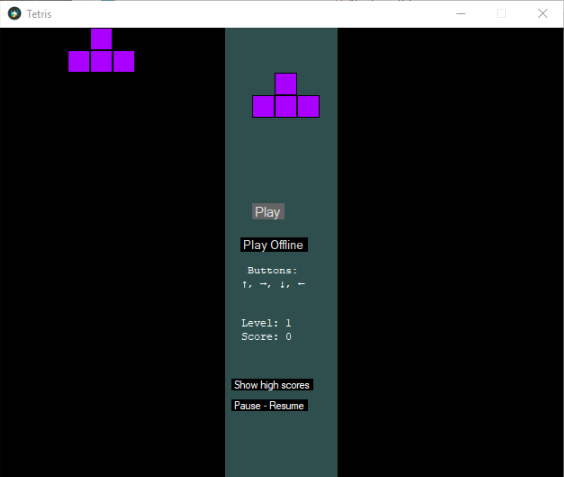


Рисунок 6.1 – Интерфейс программного средства

Для того, чтобы выйти из программы, пользователь должен закрыть окно, нажав на “крестик” в правом углу.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

[1] Шилдт, Г. C# 4.0 – Полное руководство / Г. Шилдт. – Вильямс, 2019. – 1056 c.

[2] Оланд, Д. Программирование в сетях Microsoft Windows / Д. Оланд, Э. Джонс. – Питер, 2002. – 594.

[3] metanit.com [Электронный ресурс]. – Протокол UDP. – Режим доступа: https://metanit.com/sharp/net.

[4] metanit.com [Электронный ресурс]. – Руководство по программированию в Windows Forms. – Режим доступа: https://metanit.com/sharp/windowsforms/

[5] Макконнелл, С. Совершенный код / С. Макконнелл. – БХВ, 2018. – 861с.

[6] msdn.microsoft.com [Электронный ресурс]. – Справочник по C#. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/index

# ПРИЛОЖЕНИЕ А Код программы

Файл Piece.cs:

using System.Drawing;

namespace Tetris.Model

{

public abstract class Piece

{

public abstract int[,] Pattern

{ get; }

public abstract int Rotation

{ get; }

public abstract char Name

{ get; }

public abstract int X

{get; set;}

public abstract int Y

{

get;

set;

}

public abstract Image Sprite

{

get;

}

public abstract int XNextRotation();

public abstract int YNextRotation();

public abstract int[,] NextRotation(int desiredRotation);

public abstract void Rotate();

}

}

Файл PieceI.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceI : Piece

{

private const char name = 'I';

private int x, y;

private int rotation;

private Image immI = Tetris.Properties.Resources.immI;

public override Image Sprite

{

get

{

return immI;

}

}

public PieceI()

{

this.x = 4;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[4, 1]

{

{ 1 },

{ 1 },

{ 1 },

{ 1 }

};

private int[,] pattern90 = new int[1, 4]

{

{ 1, 1, 1, 1 }

};

public override void Rotate()

{

// Цикл 0-3 (4 возможных поворота)

rotation++;

if (rotation % 4 == 0)

rotation = 0;

immI.RotateFlip(RotateFlipType.Rotate90FlipNone);

// Смена координат для централизованного поворота фигуры

switch (rotation)

{

case 0:

y -= 2;

x++;

break;

case 1:

y++;

x--;

break;

case 2:

x += 2;

y--;

break;

case 3:

x -= 2;

y += 2;

break;

}

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] Pattern

{

get

{

int[,] patt = null;

switch (rotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern0;

break;

case 3:

patt = pattern90;

break;

}

return patt;

}

}

public override int[,] NextRotation(int desiredRotation)

{

int[,] patt = null;

desiredRotation++;

if (desiredRotation % 4 == 0)

desiredRotation = 0;

switch (desiredRotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern0;

break;

case 3:

patt = pattern90;

break;

}

return patt;

}

public override int XNextRotation()

{

int xNext = this.x;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

xNext++;

break;

case 1:

xNext--;

break;

case 2:

xNext += 2;

break;

case 3:

xNext -= 2;

break;

}

return xNext;

}

public override int YNextRotation()

{

int yNext = this.y;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

yNext -= 2;

break;

case 1:

yNext++;

break;

case 2:

yNext--;

break;

case 3:

yNext += 2;

break;

}

return yNext;

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл PieceJ.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceJ : Piece

{

private const char name = 'J';

private int x, y;

private int rotation;

private Image immJ = Tetris.Properties.Resources.immJ;

public override Image Sprite

{

get

{

return immJ;

}

}

public PieceJ()

{

this.x = 4;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[3, 2]

{

{ 0, 4 },

{ 0, 4 },

{ 4, 4 }

};

private int[,] pattern90 = new int[2, 3]

{

{ 4, 0, 0 },

{ 4, 4, 4 }

};

private int[,] pattern180 = new int[3, 2]

{

{ 4, 4 },

{ 4, 0 },

{ 4, 0 }

};

private int[,] pattern270 = new int[2, 3]

{

{ 4, 4, 4 },

{ 0, 0, 4 }

};

public override void Rotate()

{

// Cycle through 0-3 (4 possible rotations)

rotation++;

if (rotation % 4 == 0)

rotation = 0;

immJ.RotateFlip(RotateFlipType.Rotate90FlipNone);

// Change of coordinates to rotate the piece centrally

switch (rotation)

{

case 0:

y--;

break;

case 1:

break;

case 2:

x++;

break;

case 3:

y++;

x--;

break;

}

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] Pattern

{

get

{

int[,] patt = null;

switch(rotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern180;

break;

case 3:

patt = pattern270;

break;

}

return patt;

}

}

public override int[,] NextRotation(int desiredRotation)

{

int[,] patt = null;

desiredRotation++;

if (desiredRotation % 4 == 0)

desiredRotation = 0;

switch (desiredRotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern180;

break;

case 3:

patt = pattern270;

break;

}

return patt;

}

public override int XNextRotation()

{

int xNext = this.x;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

break;

case 2:

xNext++;

break;

case 3:

xNext--;

break;

}

return xNext;

}

public override int YNextRotation()

{

int yNext = this.y;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

yNext--;

break;

case 1:

break;

case 2:

break;

case 3:

yNext++;

break;

}

return yNext;

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл PieceL.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceL : Piece

{

private const char name = 'L';

private int x, y;

private int rotation;

private Image immL = Tetris.Properties.Resources.immL;

public override Image Sprite

{

get

{

return immL;

}

}

public PieceL()

{

this.x = 4;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[3, 2]

{

{ 3, 0 },

{ 3, 0 },

{ 3, 3 }

};

private int[,] pattern90 = new int[2, 3]

{

{ 3, 3, 3 },

{ 3, 0, 0 }

};

private int[,] pattern180 = new int[3, 2]

{

{ 3, 3 },

{ 0, 3 },

{ 0, 3 }

};

private int[,] pattern270 = new int[2, 3]

{

{ 0, 0, 3 },

{ 3, 3, 3 }

};

public override void Rotate()

{

// Cycle through 0-3 (4 possible rotations)

rotation++;

if (rotation % 4 == 0)

rotation = 0;

immL.RotateFlip(RotateFlipType.Rotate90FlipNone);

// Change of coordinates to rotate the piece centrally

switch (rotation)

{

case 0:

x++;

break;

case 1:

y++;

x--;

break;

case 2:

y--;

break;

case 3:

break;

}

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] Pattern

{

get

{

int[,] patt = null;

switch (rotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern180;

break;

case 3:

patt = pattern270;

break;

}

return patt;

}

}

public override int[,] NextRotation(int desiredRotation)

{

int[,] patt = null;

desiredRotation++;

if (desiredRotation % 4 == 0)

desiredRotation = 0;

switch (desiredRotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern180;

break;

case 3:

patt = pattern270;

break;

}

return patt;

}

public override int XNextRotation()

{

int xNext = this.x;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

xNext++;

break;

case 1:

xNext--;

break;

case 2:

break;

case 3:

break;

}

return xNext;

}

public override int YNextRotation()

{

int yNext = this.y;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

yNext++;

break;

case 2:

yNext--;

break;

case 3:

break;

}

return yNext;

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл PieceO.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceO : Piece

{

private const char name = 'O';

private int x, y;

private int rotation;

private Image immO = Tetris.Properties.Resources.immO;

public override Image Sprite

{

get

{

return immO;

}

}

public PieceO()

{

this.x = 4;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[2, 2]

{

{ 2, 2 },

{ 2, 2 }

};

public override void Rotate()

{

rotation = 0;

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] NextRotation(int desiredRotation)

{

return pattern0;

}

public override int XNextRotation()

{

return this.x;

}

public override int YNextRotation()

{

return this.y;

}

public override int[,] Pattern

{

get

{

return pattern0;

}

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл PieceS.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceS : Piece

{

private const char name = 'S';

private int x, y;

private int rotation;

private Image immS = Tetris.Properties.Resources.immS;

public override Image Sprite

{

get

{

return immS;

}

}

public PieceS()

{

this.x = 3;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[2, 3]

{

{ 0, 7, 7 },

{ 7, 7, 0 }

};

private int[,] pattern90 = new int[3, 2]

{

{ 7, 0 },

{ 7, 7 },

{ 0, 7 }

};

public override void Rotate()

{

// Cycle through 0-3 (4 possible rotations)

rotation++;

if (rotation % 4 == 0)

rotation = 0;

immS.RotateFlip(RotateFlipType.Rotate90FlipNone);

// Change of coordinates to rotate the piece centrally

switch (rotation)

{

case 0:

break;

case 1:

x++;

break;

case 2:

y++;

x--;

break;

case 3:

y--;

break;

}

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] Pattern

{

get

{

int[,] patt = null;

switch (rotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern0;

break;

case 3:

patt = pattern90;

break;

}

return patt;

}

}

public override int[,] NextRotation(int desiredRotation)

{

int[,] patt = null;

desiredRotation++;

if (desiredRotation % 4 == 0)

desiredRotation = 0;

switch (desiredRotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern0;

break;

case 3:

patt = pattern90;

break;

}

return patt;

}

public override int XNextRotation()

{

int xNext = this.x;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

xNext++;

break;

case 2:

xNext--;

break;

case 3:

break;

}

return xNext;

}

public override int YNextRotation()

{

int yNext = this.y;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

break;

case 2:

yNext++;

break;

case 3:

yNext--;

break;

}

return yNext;

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл PieceT.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceT : Piece

{

private const char name = 'T';

private int x, y;

private int rotation;

private Image immT = Tetris.Properties.Resources.immT;

public override Image Sprite

{

get

{

return immT;

}

}

public PieceT()

{

this.x = 3;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[2, 3]

{

{ 0, 6, 0 },

{ 6, 6, 6 }

};

private int[,] pattern90 = new int[3, 2]

{

{ 6, 0 },

{ 6, 6 },

{ 6, 0 }

};

private int[,] pattern180 = new int[2, 3]

{

{ 6, 6, 6 },

{ 0, 6, 0 }

};

private int[,] pattern270 = new int[3, 2]

{

{ 0, 6 },

{ 6, 6 },

{ 0, 6 }

};

public override void Rotate()

{

rotation++;

if (rotation % 4 == 0)

rotation = 0;

immT.RotateFlip(RotateFlipType.Rotate90FlipNone);

switch (rotation)

{

case 0:

break;

case 1:

x++;

break;

case 2:

x--;

y++;

break;

case 3:

y--;

break;

}

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] Pattern

{

get

{

int[,] patt = null;

switch (rotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern180;

break;

case 3:

patt = pattern270;

break;

}

return patt;

}

}

public override int[,] NextRotation(int desiredRotation)

{

int[,] patt = null;

desiredRotation++;

if (desiredRotation % 4 == 0)

desiredRotation = 0;

switch (desiredRotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern180;

break;

case 3:

patt = pattern270;

break;

}

return patt;

}

public override int XNextRotation()

{

int xNext = this.x;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

xNext++;

break;

case 2:

xNext--;

break;

case 3:

break;

}

return xNext;

}

public override int YNextRotation()

{

int yNext = this.y;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

break;

case 2:

yNext++;

break;

case 3:

yNext--;

break;

}

return yNext;

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл PieceZ.cs:

using System.Drawing;

namespace Tetris.Model

{

class PieceZ : Piece

{

private const char name = 'Z';

private int x, y;

private int rotation;

private Image immZ = Tetris.Properties.Resources.immZ;

public override Image Sprite

{

get

{

return immZ;

}

}

public PieceZ()

{

this.x = 3;

this.y = 0;

this.rotation = 0;

}

private int[,] pattern0 = new int[2, 3]

{

{ 5, 5, 0 },

{ 0, 5, 5 }

};

private int[,] pattern90 = new int[3, 2]

{

{ 0, 5 },

{ 5, 5 },

{ 5, 0 }

};

public override void Rotate()

{

// Cycle through 0-3 (4 possible rotations)

rotation++;

if (rotation % 4 == 0)

rotation = 0;

immZ.RotateFlip(RotateFlipType.Rotate90FlipNone);

// Change of coordinates to rotate the piece centrally

switch (rotation)

{

case 0:

break;

case 1:

x++;

break;

case 2:

x--;

y++;

break;

case 3:

y--;

break;

}

}

public override int Rotation

{

get

{

return rotation;

}

}

public override int[,] Pattern

{

get

{

int[,] patt = null;

switch (rotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern0;

break;

case 3:

patt = pattern90;

break;

}

return patt;

}

}

public override int[,] NextRotation(int desiredRotation)

{

int[,] patt = null;

desiredRotation++;

if (desiredRotation % 4 == 0)

desiredRotation = 0;

switch (desiredRotation)

{

case 0:

patt = pattern0;

break;

case 1:

patt = pattern90;

break;

case 2:

patt = pattern0;

break;

case 3:

patt = pattern90;

break;

}

return patt;

}

public override int XNextRotation()

{

int xNext = this.x;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

xNext++;

break;

case 2:

xNext--;

break;

case 3:

break;

}

return xNext;

}

public override int YNextRotation()

{

int yNext = this.y;

int nextRotation = rotation;

nextRotation++;

if (nextRotation % 4 == 0)

nextRotation = 0;

switch (nextRotation)

{

case 0:

break;

case 1:

break;

case 2:

yNext++;

break;

case 3:

yNext--;

break;

}

return yNext;

}

public override char Name

{

get

{

return name;

}

}

public override int X

{

get

{

return x;

}

set

{

x = value;

}

}

public override int Y

{

get

{

return y;

}

set

{

y = value;

}

}

}

}

Файл Game.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

using System.Windows.Forms;

using System.Xml.Serialization;

using Tetris.Controller;

namespace Tetris.Model

{

class Game

{

// Ссылки на объекты

public int[] pieces = new int[255];

public TetrisGrid g1 = new TetrisGrid();

private PieceProvider provider;

private Piece currentPiece, nextPiece;

// Функциональные переменные

private bool inPlay = false;

private int score;

private int difficulty;

public List<Ranking> scores = new List<Ranking>();

// Конструктор, инициализация игры

public Game()

{

provider = new PieceProvider(pieces);

InitializeGame();

}

// Способ проверки и/или набора во время игры

public bool InPlay

{

get

{

return inPlay;

}

set

{

inPlay = value;

}

}

// Инициализация (или реинициализация) игры

public void InitializeGame()

{

currentPiece = null;

nextPiece = null;

g1.EmptyGrid();

g1.EmptyGrid2();

inPlay = true;

Score = 0;

Difficulty = 4;

ExtractPieces();

if (!File.Exists("HighScores.xml"))

CreateRankingFile();

LoadScores();

}

public void CreateRankingFile()

{

XmlSerializer writer = new XmlSerializer(scores.GetType(), "Tetris.Model");

FileStream file = File.Create("HighScores.xml");

writer.Serialize(file, scores);

file.Close();

}

private void LoadScores()

{

try

{

XmlSerializer serializer = new XmlSerializer(scores.GetType(), "Tetris.Model");

object obj;

using (StreamReader reader = new StreamReader("HighScores.xml"))

{

obj = serializer.Deserialize(reader.BaseStream);

reader.Close();

}

scores = (List<Ranking>)obj;

}

catch(InvalidOperationException)

{

CreateRankingFile();

LoadScores();

}

}

public void SaveScore(string name, int points)

{

var score = new Ranking()

{

Score = points,

Name = name

};

scores.Add(score);

try

{

XmlSerializer writer = new XmlSerializer(scores.GetType(), "Tetris.Model");

FileStream file = File.Create("HighScores.xml");

writer.Serialize(file, scores);

file.Close();

}

catch (UnauthorizedAccessException)

{

System.Windows.Forms.MessageBox.Show("Unable to write in 'HighScores.xml' file.\nIt's probably read-only");

}

catch (IOException)

{

System.Windows.Forms.MessageBox.Show("Unable to save 'HighScores.xml' file.\nProbably the disk is full");

}

}

// получение созданной фигурки

public Piece CurrentPiece

{

get

{

return currentPiece;

}

}

// получение следующей фигурки

public Piece NextPiece

{

get

{

return nextPiece;

}

}

// завершение игры

private void EndGame()

{

InPlay = false;

}

// создание фигурок

public void ExtractPieces()

{

// если это первый старт, обе части должны быть созданы

if (currentPiece == null && nextPiece == null)

{

currentPiece = provider.ExtractPiece();

nextPiece = provider.ExtractPiece();

if (currentPiece == nextPiece)

{

nextPiece = provider.ExtractPiece();

}

}

// если это не первый запуск, он назначает текущую фигурку следующей

else

{

if (nextPiece != currentPiece)

{

currentPiece = nextPiece;

}

else

{

currentPiece = provider.ExtractPiece();

}

currentPiece = nextPiece;

nextPiece = provider.ExtractPiece();

}

}

// возвращает количество строк текущей фигурки

private int RowsCurrentPiece

{

get

{

return currentPiece.Pattern.GetLength(0);

}

}

// возвращает количество столбцов текущей фигурки

private int ColumnsCurrentPiece

{

get

{

return currentPiece.Pattern.GetLength(1);

}

}

// можно ли выполнить движение

private bool CanMove(int x, int y)

{

bool canMove = true;

// может ли кусок существовать в заданных координатах

for (int i = y, row = 0; i < (y + RowsCurrentPiece); i++, row++)

for (int j = x, column = 0; j < (x + ColumnsCurrentPiece); j++, column++)

if ((g1.Grid1[i, j] != 0) && (currentPiece.Pattern[row, column] != 0))

canMove = false;

return canMove;

}

// «закрепить» текущий кусок на сетке

private void FixPiece()

{

for (int i = currentPiece.Y, rowPiece = 0; i < (currentPiece.Y + RowsCurrentPiece); i++, rowPiece++)

for (int j = currentPiece.X, columnPiece = 0; (j < (currentPiece.X + ColumnsCurrentPiece)); j++, columnPiece++)

if (currentPiece.Pattern[rowPiece, columnPiece] != 0)

g1.Grid1[i, j] = currentPiece.Pattern[rowPiece, columnPiece];

}

// Метод проверки возможности вращения

private bool CanRotate()

{

bool canRotate = true;

// получение временной матрицы вращения, следующей за текущей

int[,] rotationTemp = currentPiece.NextRotation(currentPiece.Rotation);

// обновление х и у вращения

int xNext = currentPiece.XNextRotation(),

yNext = currentPiece.YNextRotation();

// Проверяет, выходит ли следующее вращение из сетки

if ((yNext + rotationTemp.GetLength(0)) > g1.Grid1.GetLength(0))

canRotate = false;

if ((xNext + rotationTemp.GetLength(1)) > g1.Grid1.GetLength(1))

canRotate = false;

if (xNext < 0)

canRotate = false;

int i, j;

if (canRotate)

for (i = yNext; (i < (yNext + rotationTemp.GetLength(0))); i++)

for (j = xNext; (j < (xNext + rotationTemp.GetLength(1))); j++)

if (g1.Grid1[i, j] != 0)

canRotate = false;

return canRotate;

}

// Метод перемещения фигуры влево, вызываемый контроллером

public void MoveLeft()

{

// проверка что текущий кусок не находится в левом краю

if (currentPiece.X != 0)

{

// Проверяет, может ли текущий фрагмент двигаться, передав метод CanMove ();

// х, у изменен

if (CanMove(currentPiece.X - 1, currentPiece.Y))

{

// Движение возможно, Х уменьшилось

currentPiece.X--;

}

}

}

// Метод перемещения фигуры вправо, вызываемый контроллером

public void MoveRight()

{

// проверка что текущий кусок не находится на правом краю

if ((currentPiece.X + ColumnsCurrentPiece) < g1.Grid1.GetLength(1))

{

// Проверяет, может ли текущий фрагмент перемещаться с помощью метода CanMove (),

// передавая ему увеличенную координату x, y в неизмененном виде

if (CanMove(currentPiece.X + 1, currentPiece.Y))

{

// можно передвинуть, Х увеличивается

currentPiece.X++;

}

}

}

// Метод перемещения фигуры вниз, вызываемый контроллером

public bool MoveDown()

{

bool fixedPiece = false;

// проверка на то, идет ли игра

if ((g1.Grid1[0, 4] != 0) || (g1.Grid1[0, 3] != 0))

EndGame();

else

{

// текущий кусок не находится на нижнем торце поля

if (((currentPiece.Y + RowsCurrentPiece) < g1.Grid1.GetLength(0)) &&

(CanMove(currentPiece.X, currentPiece.Y + 1)))

currentPiece.Y++;

else

{

// Кусок больше не может двигаться, его необходимо «закрепить» на базовой сетке и убедиться,

// что линия не сформирована полностью, и в этом случае удалить его.

FixPiece();

fixedPiece = true;

CheckCompletedLine();

}

}

return fixedPiece;

}

// Метод поворота текущего фрагмента, вызываемый контроллером

public bool RotatePiece()

{

bool val = false;

if (CanRotate())

{

currentPiece.Rotate();

val = true;

}

return val;

}

private bool CheckCompletedLine()

{

bool completedLine = true;

for (int i = 0; i < g1.Grid1.GetLength(0); i++)

{

completedLine = true;

// Проверяет, есть ли хотя бы один элемент равный 0, если таковой имеется,

// строка не завершена, и будет пропущено следующее условие. Перейти к следующей строке (если есть)

for (int j = 0; (j < g1.Grid1.GetLength(1)) && (completedLine == true); j++)

if (g1.Grid1[i, j] == 0)

completedLine = false;

if (completedLine)

{

// возвращение завершенной линии (i) и всех элементов выше

// строки копируются на неё. То же самое делается, пока не доберется до линии 0, где

// она установлена в 0, потому что все вниз на одну строку

for (int k = i; k > 0; k--)

for (int j = 0; j < g1.Grid1.GetLength(1); j++)

g1.Grid1[k, j] = g1.Grid1[k - 1, j];

for (int j = 0; j < g1.Grid1.GetLength(1); j++)

g1.Grid1[0, j] = 0;

// за каждую строку счет + 10

Score += 10;

// Каждые 100 баллов(10 линий пройдено) сложность увеличивается

if (Score % 100 == 0)

Difficulty++;

}

}

return completedLine;

}

public int Difficulty

{

get

{

return difficulty;

}

private set

{

difficulty = value;

}

}

public int Score

{

get

{

return score;

}

private set

{

score = value;

}

}

public TetrisGrid GridP1

{

get

{

return g1;

}

}

}

}

Файл PieceProvider.cs:

using System;

namespace Tetris.Model

{

class PieceProvider

{

private Random rnd = new Random();

private int tempPiece = 1;

int[] Pieces = new int[255];

public PieceProvider(int[] pieces)

{

Pieces = pieces;

}

public Piece ExtractPiece()

{

Piece currentPiece = null;

switch (Pieces[tempPiece])

{

case 0:

currentPiece = new PieceT();

break;

case 1:

currentPiece = new PieceO();

break;

case 2:

currentPiece = new PieceI();

break;

case 3:

currentPiece = new PieceS();

break;

case 4:

currentPiece = new PieceJ();

break;

case 5:

currentPiece = new PieceL();

break;

case 6:

currentPiece = new PieceZ();

break;

}

tempPiece++;

return currentPiece;

}

}

}

Файл NameForm.cs:

using System;

using System.Windows.Forms;

namespace Tetris.Model

{

public partial class NameForm : Form

{

private static string name = null;

public NameForm()

{

InitializeComponent();

}

public string Name

{

get

{

return name;

}

private set

{

name = value;

}

}

// Click event on "Ok" button

private void bOk\_Click(object sender, EventArgs e)

{

Validate();

}

// KeyDown event: pressing Enter in the TextBox will trigger the Validate method

private void tName\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyData == Keys.Enter)

Validate();

}

// Method used to validate the input. If it's valid it saves the name and closes the form

private void Validate()

{

if ((tName.Text.Length < 13) && (tName.Text.Length > 0))

{

Name = tName.Text;

this.Close();

}

else

MessageBox.Show("Name too long or not valid");

}

private void NameForm\_Load(object sender, EventArgs e)

{

}

}

}

Файл Program.cs:

using Newtonsoft.Json;

using System;

using System.Threading;

using System.Windows.Forms;

using Tetris.Controller;

using Tetris.Model;

namespace Tetris

{

class Program

{

[STAThread]

static void Main(string[] args)

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

TetrisController gameController = new TetrisController();

gameController.StartGame();

}

}

}

Файл Ranking.cs:

using System;

namespace Tetris.Model

{

// Сериализуемый класс, используемый для сохранения результатов и имени в файле "HighScores.xml"

[Serializable()]

public class Ranking

{

public int Score { get; set; }

public string Name { get; set; }

}

}

Файл RankingForm.cs:

using System.Collections.Generic;

using System.Windows.Forms;

namespace Tetris.View

{

public partial class RankingForm : Form

{

private delegate void ButtonEventHandler(object sender, System.EventArgs e);

public RankingForm(List<Model.Ranking> scores)

{

InitializeComponent();

dataGrid.DataSource = scores;

}

private void RankingForm\_Load(object sender, System.EventArgs e)

{

}

}

}

Файл TetrisController.cs:

using System.Drawing;

using System.Windows.Forms;

using Tetris.View;

using Tetris.Model;

using System.IO;

using System.Threading;

using System;

namespace Tetris.Controller

{

class TetrisController

{

// Ссылки на объекты

public Game game;

// private WaitingForm waiting;

private TetrisForm tView;

private NameForm nameForm;

private RankingForm rankingForm;

// Рисование

private Graphics g;

private SolidBrush brush = new SolidBrush(Color.DarkSlateGray);

private Rectangle sidebar = new Rectangle(250, 0, 125, 500);

private Rectangle pieceContainer;

// Изображения 25x25px для просмотра уже исправленных фрагментов

private Image pieceI\_25px = Properties.Resources.immI\_25;

private Image pieceJ\_25px = Properties.Resources.immJ\_25;

private Image pieceL\_25px = Properties.Resources.immL\_25;

private Image pieceO\_25px = Properties.Resources.immO\_25;

private Image pieceS\_25px = Properties.Resources.immS\_25;

private Image pieceT\_25px = Properties.Resources.immT\_25;

private Image pieceZ\_25px = Properties.Resources.immZ\_25;

// Функциональные переменные

private bool fixedPiece = false;

private bool onPause = false;

private int difficulty;

// Constants

private const int blockSize = 25;

UdpListenSend udp;

Thread ReceiveThread = null;

public int ScoreOpponent { get; set; }

Random rand = new Random();

bool Start = false;

public TetrisController()

{

this.tView = new TetrisForm();

this.tView.KeyDown += new KeyEventHandler(Tetris\_Form\_KeyDown);

this.tView.Paint += new PaintEventHandler(Tetris\_Form\_Draw);

this.tView.bShowRanking.Click += new EventHandler(Tetris\_Form\_ShowRanking);

this.tView.bPauseResume.Click += new EventHandler(Tetris\_Form\_PauseResume);

this.tView.Resize += new EventHandler(Tetris\_Form\_PauseResumeMinimize);

this.tView.GotFocus += new EventHandler(Tetris\_Form\_PauseResumeGotLostFocus);

this.tView.LostFocus += new EventHandler(Tetris\_Form\_PauseResumeGotLostFocus);

this.tView.Playlbl.Click += new EventHandler(PlayLabel);

this.tView.PlayOfflineLbl.Click += new EventHandler(PlayOffline);

}

public void StartGame()

{

// Инициализация игрового объекта и начало формы

this.game = new Game();

udp = new UdpListenSend(game.GridP1, ScoreOpponent, game.pieces);

for (int i = 0; i < game.pieces.Length; i++)

game.pieces[i] = rand.Next(0, 7);

this.tView.t.Tick += new EventHandler(Tetris\_Form\_Tick);

this.difficulty = this.game.Difficulty;

this.tView.t.Enabled = false;

udp.SendRequest();

ReceiveThread = new Thread( ( ) => { udp.GetIpAddressFromUser(); } );

ReceiveThread.Start();

this.tView.ShowDialog();

}

public void CountForStart()

{

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

{

tView.TimerLbl.Text = i.ToString();

Thread.Sleep(500);

tView.Update();

}

tView.TimerLbl.Text = "GO";

Thread.Sleep(500) ;

tView.Update();

tView.TimerLbl.Text = "";

StartTimer();

}

public void PlayOffline(object sender, System.EventArgs e)

{

ReceiveThread.Abort();

CountForStart();

ResumeGame();

}

public void StartTimer()

{

tView.t.Enabled = true;

}

public void PlayLabel(object sender, System.EventArgs e)

{

if (udp.Start)

{

CountForStart();

tView.t.Enabled = true;

ResumeGame();

}

}

private void Tetris\_Form\_KeyDown(object sender, KeyEventArgs e)

{

//while (!Start) { }

//CountForStart();

if (!onPause)

{

udp.SendGrid(game.GridP1);

switch (e.KeyData)

{

// Для каждой кнопки выполнить движение и обновить

// положение фрагмента, где он находится, и следующий

case Keys.Up:

game.RotatePiece();

tView.Invalidate(new Rectangle(game.CurrentPiece.X \* blockSize,

game.CurrentPiece.Y \* blockSize,

game.CurrentPiece.Sprite.Width,

game.CurrentPiece.Sprite.Height));

tView.Invalidate(pieceContainer);

tView.Update();

break;

case Keys.Down:

MoveDown();

break;

case Keys.Left:

game.MoveLeft();

tView.Invalidate(pieceContainer);

pieceContainer.X -= blockSize;

tView.Invalidate(pieceContainer);

tView.Update();

break;

case Keys.Right:

game.MoveRight();

tView.Invalidate(pieceContainer);

pieceContainer.X += blockSize;

tView.Invalidate(pieceContainer);

tView.Update();

break;

}

}

}

private void Tetris\_Form\_PauseResumeGotLostFocus(object sender, System.EventArgs e)

{

if (tView.Focused == true)

{

//CountForStart();

// ResumeGame();

}

else if (tView.Focused == false)

PauseGame();

}

// временнОе событие: часть перемещена вниз

private void Tetris\_Form\_Tick(object sender, System.EventArgs e)

{

MoveDown();

udp.SendGrid(game.GridP1);

udp.SendScore(game.Score);

}

private void Tetris\_Form\_PauseResumeMinimize(object sender, System.EventArgs e)

{

if (tView.WindowState == FormWindowState.Minimized)

PauseGame();

else if (tView.WindowState == FormWindowState.Normal)

ResumeGame();

}

private void Tetris\_Form\_ShowRanking(object sender, System.EventArgs e)

{

PauseGame();

rankingForm = new RankingForm(game.scores);

rankingForm.bEmpty.Click += new System.EventHandler(RankingForm\_Empty\_Click);

rankingForm.ShowDialog();

ResumeGame();

}

private void RankingForm\_Empty\_Click(object sender, System.EventArgs e)

{

rankingForm.dataGrid.DataSource = null;

game.scores.Clear();

game.CreateRankingFile();

rankingForm.dataGrid.DataSource = game.scores;

}

// приостановить игру

private void PauseGame()

{

onPause = true;

tView.t.Enabled = false;

}

// возобновить игру с паузы

private void ResumeGame()

{

onPause = false;

tView.t.Enabled = true;

}

// переключения с паузы и возобновления игры

private void PauseResumeGame()

{

if (tView.t.Enabled == true)

PauseGame();

else

ResumeGame();

}

// Метод, который обрабатывает событие click формы на Label bPauseResume

private void Tetris\_Form\_PauseResume(object sender, System.EventArgs e)

{

PauseResumeGame();

}

// Метод, который занимается рисованием фигур в форме

private void Tetris\_Form\_Draw(object sender, PaintEventArgs e)

{

// Управление событием Paint

e.Graphics.FillRectangle(brush, sidebar);

DrawBlocks(g, e);

DrawPiece(game.CurrentPiece, g, e);

// DrawPiece2(game.CurrentPiece, g, e);

// DrawPiece2(game.CurrentPiece, g, e);

DrawNextPiece(game.NextPiece, g, e);

}

private const int secondgridmove = 375;

// Метод для рисования блоков, уже зафиксированных в сетке

private void DrawBlocks(Graphics g, PaintEventArgs e)

{

for (int y = 0; y < game.GridP1.Grid1.GetLength(0); y++)

for (int x = 0; x < game.GridP1.Grid1.GetLength(1); x++)

{

switch (game.GridP1.Grid1[y, x])

{

case 1:

e.Graphics.DrawImage(pieceI\_25px,

x \* 25,

y \* 25);

break;

case 2:

e.Graphics.DrawImage(pieceO\_25px,

x \* 25,

y \* 25);

break;

case 3:

e.Graphics.DrawImage(pieceL\_25px,

x \* 25,

y \* 25);

break;

case 4:

e.Graphics.DrawImage(pieceJ\_25px,

x \* 25,

y \* 25);

break;

case 5:

e.Graphics.DrawImage(pieceZ\_25px,

x \* 25,

y \* 25);

break;

case 6:

e.Graphics.DrawImage(pieceT\_25px,

x \* 25,

y \* 25);

break;

case 7:

e.Graphics.DrawImage(pieceS\_25px,

x \* 25,

y \* 25);

break;

}

switch (game.GridP1.Grid2[y, x])

{

case 1:

e.Graphics.DrawImage(pieceI\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

case 2:

e.Graphics.DrawImage(pieceO\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

case 3:

e.Graphics.DrawImage(pieceL\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

case 4:

e.Graphics.DrawImage(pieceJ\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

case 5:

e.Graphics.DrawImage(pieceZ\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

case 6:

e.Graphics.DrawImage(pieceT\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

case 7:

e.Graphics.DrawImage(pieceS\_25px,

x \* 25 + secondgridmove,

y \* 25);

break;

}

}

}

public Thread draw = null;

// обрабатывает рисунок текущего куска

private void DrawPiece(Piece p, Graphics g, PaintEventArgs e)

{

pieceContainer = new Rectangle(p.X \* blockSize,

p.Y \* blockSize,

p.Sprite.Width,

p.Sprite.Height);

//e.Graphics.DrawImage(p.Sprite,pieceContainer);

e.Graphics.DrawImage(p.Sprite, p.X \* 25, p.Y \* 25);

}

private void DrawPiece2(Piece p, Graphics g, PaintEventArgs e)

{

switch (p.Name)

{

case 'I':

e.Graphics.DrawImage(pieceI\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'O':

e.Graphics.DrawImage(pieceO\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'L':

e.Graphics.DrawImage(pieceL\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'J':

e.Graphics.DrawImage(pieceJ\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'Z':

e.Graphics.DrawImage(pieceZ\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'T':

e.Graphics.DrawImage(pieceT\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

case 'S':

e.Graphics.DrawImage(pieceS\_25px,

p.X \* 25 + secondgridmove,

p.Y \* 25);

break;

}

e.Graphics.DrawImage(p.Sprite, p.X \* 25 + 625, p.Y \* 25);

// e.Graphics.DrawImage(p.Sprite, p.X \* 25+ secondgridmove, p.Y \* 25);

}

// обрабатывает рисование следующего куска на боковой панели

private void DrawNextPiece(Piece p, Graphics g, PaintEventArgs e)

{

e.Graphics.DrawImage(p.Sprite,

p.Name=='I'? 290:280,

50);

}

// Метод, который работает с интервальным таймером в форме

private void DecreaseTimerInterval()

{

// Интервал таймера не может быть <200

// Шаги уменьшения 50

if (tView.t.Interval >= 200)

tView.t.Interval -= 50;

}

// сдвинуть текущий кусок вниз

private void MoveDown()

{

// действует ли игра еще

if (game.InPlay)

{

// деталь сдвинута ли вниз

if (!fixedPiece)

{

fixedPiece = game.MoveDown();

if (fixedPiece)

{

tView.Invalidate();

// Если часть была исправлена,

// извлечь новые части и повторно инициализировать фиксированную переменную

game.ExtractPieces();

fixedPiece = false;

// Обновить метку счета и проверить

// если сложность увеличилась по сравнению с текущей

// если увеличено, изменитб интервальный таймер и обновить метку

// до текущего уровня

tView.Score = game.Score.ToString() + ": Score : " + udp.Score;

if (difficulty < game.Difficulty)

{

DecreaseTimerInterval();

difficulty = game.Difficulty;

tView.Diff = "Level: " + difficulty.ToString();

}

}

}

}

else

{

// заканчивается игра:

// отключаем объект таймера и показываем MessageBox со счетом

// Нажав кнопку "ОК", игра начинается снова и переменные будут инициализированы заново

this.tView.t.Enabled = false;

if (game.Score == udp.Score)

MessageBox.Show("Ничья довольно редкое событие в этой игре, но высмогли. Набрано очков: " + game.Score);

else

if (game.Score > udp.Score)

MessageBox.Show("Победа за вами! набрано очков: " + game.Score);

else

MessageBox.Show("Увы, вы проиграли:( , счёт: " + game.Score);

InsertScore();

game.InitializeGame();

this.tView.Initialize();

this.tView.Invalidate();

}

tView.Invalidate(pieceContainer);

pieceContainer.Y += blockSize;

tView.Invalidate(pieceContainer);

tView.Update();

//!!!!!! tcp.SendGrid(user, game.g1);

}

// Метод для отображения NameForm

private void InsertScore()

{

nameForm = new NameForm();

this.nameForm.ShowDialog();

game.SaveScore(nameForm.Name, game.Score);

}

}

}

Файл TetrisForm.cs:

using System.Drawing;

using System.Windows.Forms;

namespace Tetris.View

{

public partial class TetrisForm : Form

{

private delegate void KeyEventHandler(object sender, KeyEventArgs e);

private delegate void PaintEventHandler(object sender, PaintEventArgs e);

private delegate void EventHandler(object sender, System.EventArgs e);

public Timer t;

public TetrisForm()

{

InitializeComponent();

this.t = new Timer();

Initialize();

this.DoubleBuffered = true;

this.BackColor = Color.Black;

}

public string Score

{

set

{

this.score.Text = value;

}

}

// Initialization (or re-initialization) of timer and labels

public void Initialize()

{

this.t.Enabled = false;

this.t.Interval = 700;

this.diff.Text = "Level: 1";

this.score.Text = "Score: 0";

}

public string Diff

{

set

{

this.diff.Text = value;

}

}

private int TimerInterval

{

get

{

return t.Interval;

}

set

{

t.Interval = value;

}

}

private void TetrisForm\_Load(object sender, System.EventArgs e)

{

}

private void buttons\_Click(object sender, System.EventArgs e)

{

}

private void bPauseResume\_Click(object sender, System.EventArgs e)

{

}

}

}

Файл TetrisGrid.cs:

using System;

namespace Tetris.Model

{

class TetrisGrid

{

public const int gridColumns = 10;

public const int gridRows = 20;

private static int[,] grid = new int[gridRows, gridColumns];

private static int[,] grid2 = new int[gridRows, gridColumns];

public TetrisGrid()

{

}

public int[,] Grid1

{

get { return grid; }

set { grid = value; }

}

public int[,] Grid2

{

get { return grid2; }

set { grid2 = value; }

}

public void EmptyGrid()

{

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

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

grid[i, j] = 0;

}

public void EmptyGrid2()

{

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

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

grid2[i, j] = 0;

}

}

}

Файл UdpListenSend.cs:

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Tetris.Model;

namespace Tetris

{

class UdpListenSend

{

private UdpClient UdpSender;

private readonly IPAddress IpAdressBroadcast = IPAddress.Parse("192.168.43.255");//IPAddress.Broadcast;

private IPEndPoint ipUserEndPoint;

private UdpClient UdpListener = null;

private readonly string key = "KEY";

public IPAddress UserIP = null;

public int UdpSendPort = 55555;

public int ListenPiecesPort = 48048;

public Random rand = new Random();

public int ReceiveGridPort;

public int SendGridPort;

private Thread RecThread = null;

public TetrisGrid Grid;

public int Score = 0;

public bool Start = false;

public int[] Pieces = new int[255];

public UdpListenSend(TetrisGrid grid, int score,int[] pieces)

{

Grid = grid;

Score = score;

Pieces = pieces;

}

public void SendRequest()

{

ReceiveGridPort = rand.Next(55000, 56000);

try

{

UdpSender = new UdpClient(UdpSendPort, AddressFamily.InterNetwork);

ipUserEndPoint = new IPEndPoint(IpAdressBroadcast,UdpSendPort);

byte[] KeyBytes = Encoding.ASCII.GetBytes(key+ ReceiveGridPort.ToString());

int sendedData = UdpSender.Send(KeyBytes, KeyBytes.Length, ipUserEndPoint);

UdpSender.Close();

}

catch

{

}

}

public void GetIpAddressFromUser()

{

UdpListener = new UdpClient();

try

{

IPEndPoint ClientEndPoint = new IPEndPoint(IPAddress.Any, UdpSendPort);

UdpListener.Client.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

UdpListener.ExclusiveAddressUse = false;

UdpListener.Client.Bind(ClientEndPoint);

while (true)

{

Byte[] data = UdpListener.Receive(ref ClientEndPoint);

string GetKey = Encoding.ASCII.GetString(data);

if (GetKey[0] == 'P')

{

// for (int i=0; i<masstr.Length;i++)

string[] strget = GetKey.Split(' ');

for (int i = 0; i < Pieces.Length; i++)

{

try

{

Pieces[i] = int.Parse(strget[i]);

}

catch { }

}

string str = GetKey.Substring(1, GetKey.Length - 1);

StreamWriter SW = new StreamWriter(new FileStream("FileName.txt", FileMode.Create, FileAccess.Write));

SW.Write(GetKey);

SW.Close();

}

if (GetKey.Substring(0, 3) == key)

{

SendGridPort = int.Parse(GetKey.Substring(3, GetKey.Length - 3));

UserIP = ClientEndPoint.Address;

Start = true;

MessageBox.Show("Opponent was finded. You can play with him");

break;

}

}

UdpListener.Close();

Thread.Sleep(150);

try

{

string SendStr = "";

for (int i = 0; i < Pieces.Length; i++)

{

SendStr = SendStr + " " + Pieces[i].ToString();

}

UdpSender = new UdpClient(UdpSendPort, AddressFamily.InterNetwork);

ipUserEndPoint = new IPEndPoint(UserIP, UdpSendPort);

byte[] KeyBytes = Encoding.ASCII.GetBytes("P" + SendStr);

int sendedData = UdpSender.Send(KeyBytes, KeyBytes.Length, ipUserEndPoint);

Thread.Sleep(100);

ipUserEndPoint = new IPEndPoint(IpAdressBroadcast, UdpSendPort);

KeyBytes = Encoding.ASCII.GetBytes(key + ReceiveGridPort.ToString());

sendedData = UdpSender.Send(KeyBytes, KeyBytes.Length, ipUserEndPoint);

UdpSender.Close();

RecThread = new Thread(() => { GetGridUdp(Grid); } );

RecThread.Start();

}

catch

{

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

finally

{

UdpListener.Close();

}

}

public void SendGrid(TetrisGrid grid)

{

string jsonObject = JsonConvert.SerializeObject(grid.Grid1, Formatting.Indented, new JsonSerializerSettings { });

try

{

if (UserIP != null)

{

UdpSender = new UdpClient(SendGridPort, AddressFamily.InterNetwork);

ipUserEndPoint = new IPEndPoint(UserIP, SendGridPort);

byte[] SendData = Encoding.ASCII.GetBytes(jsonObject);

int sendedData = UdpSender.Send(SendData, SendData.Length, ipUserEndPoint);

UdpSender.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

public void GetGridUdp(TetrisGrid grid)

{

UdpListener = new UdpClient();

try

{

IPEndPoint ClientEndPoint = new IPEndPoint(IPAddress.Any, ReceiveGridPort);

UdpListener.Client.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

UdpListener.ExclusiveAddressUse = false;

UdpListener.Client.Bind(ClientEndPoint);

while (true)

{

Byte[] data = UdpListener.Receive(ref ClientEndPoint);

string Message = Encoding.ASCII.GetString(data);

if (Message.Substring(0, 1) == "S")

{

Score = int.Parse(Message.Substring(1, Message.Length - 1));

}

else

{

int[,] getgrid = JsonConvert.DeserializeAnonymousType<int[,]>(Message, grid.Grid2);

grid.Grid2 = getgrid;

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

finally

{

UdpListener.Close();

}

}

public void SendScore(int Score)

{

try

{

if (UserIP != null)

{

UdpSender = new UdpClient(SendGridPort, AddressFamily.InterNetwork);

ipUserEndPoint = new IPEndPoint(UserIP, SendGridPort);

byte[] SendData = Encoding.ASCII.GetBytes("S"+Score.ToString());

int sendedData = UdpSender.Send(SendData, SendData.Length, ipUserEndPoint);

UdpSender.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

public void SendPieces()

{

for (int i = 0; i < Pieces.Length; i++)

{

Pieces[i] = rand.Next(0, 7);

}

try

{

UdpSender = new UdpClient(SendGridPort, AddressFamily.InterNetwork);

ipUserEndPoint = new IPEndPoint(UserIP, SendGridPort);

byte[] KeyBytes = Encoding.ASCII.GetBytes("P" + Pieces.ToString());

int sendedData = UdpSender.Send(KeyBytes, KeyBytes.Length, ipUserEndPoint);

UdpSender.Close();

}

catch

{

}

}

}

}