РОССИЙСКАЯ ФЕДЕРАЦИЯ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и компьютерных наук

Кафедра программного обеспечения

КУРСОВАЯ РАБОТА

по курсу: «Языки программирования»

на тему: «Разработка приложения «Игра «Две лисы и двадцать кур»

Выполнил:

Студент 168-1 группы

Жомов Юрий Андреевич

Проверил:

Старший преподаватель

Безруков Андрей Сергеевич

Оглавление

[ВВЕДЕНИЕ 3](#_Toc502328388)

[Глава 1. Описание предметной области 4](#_Toc502328389)

[1.1 Рекурсия 4](#_Toc502328390)

[1.2 Основные моменты программирования логических игр 5](#_Toc502328391)

[1.3 Аналитическая функция оценки 6](#_Toc502328392)

[1.4 Работа с коллекцией игровых объектов 7](#_Toc502328393)

[1.5 Игровой процесс 8](#_Toc502328394)

[Глава 2. Разработка программного приложения 9](#_Toc502328395)

[2.1 Подробное описание классов 9](#_Toc502328396)

[2.2 Описание используемых компонентов форм 21](#_Toc502328397)

[2.3 Описание разработанных алгоритмов 24](#_Toc502328398)

[Глава 3. Описание пользовательского интерфейса 26](#_Toc502328399)

[ЗАКЛЮЧЕНИЕ 31](#_Toc502328400)

[СПИСОК ЛИТЕРАТУРЫ 32](#_Toc502328401)

[ПРИЛОЖЕНИЯ 33](#_Toc502328402)

# ВВЕДЕНИЕ

Программирование логических игр, подобных шахматам и шашкам, позволяет создать иллюзию искусственного интеллекта. На самом деле навыки машины полостью зависят от того как хорошо «научил» её играть программист. В простых логических играх, таких как «крестики нолики», не трудно просчитать все возможные варианты ходов и выбрать наилучшие, получив программу всегда выигрывающую или сводящую партию в ничью. Но как быть, если количество возможных ходов бесконечно большое? В таком случае придется рассчитывать «силу» каждого потенциального хода после каждого хода игрока. Целью курсовой работы является программирования логической игры «две лисы и двадцать кур». Основываясь на описанном принципе выше, с помощью рекурсивного алгоритма оценки силы хода, реализован искусственный интеллект для игры «за лис». Он соблюдает правила игры, просчитывать цепочки «поедания» кур и выбирать наиболее длинную из возможных. Для программирования была использована среда разработки «Microsoft Visual Studio 2015», язык программирования C#.



Рисунок . Вид готового приложения

# Глава 1. Описание предметной области

Программирование логических игры было начато еще в середине ХХ века и потому имеется множество примеров различных алгоритмов, ставших типичными для логических игр. При работе над приложением была использована книга «Программирование шахмат и других логических игр» за авторством Евгения Корнилова. Данная книга содержит описание множества алгоритмов, которые применяются или применялись при программирование различных логических игр. Важнейшие разделы книги, которые нашли наиболее сильное отражение в реализованной программе, это посвященные рекурсии, основным моментам программирования логических игр и аналитической функции оценки.

# **1.1 Рекурсия**

Рекурсией называется вызов функции самой себя. При этом все формальные параметры функции задаются заново. Алгоритмы, основанные на данном принципе – рекурсивными. Рекурсия очень удобна для решения задач, основанных на многократном выполнении определенного алгоритма, который при этом идентичен как для первого, так и для некоторого i-го раза выполнения. Глубиной рекурсии называют количество раз вызова функции самой себя в ходе выполнения алгоритма. Она, чаще всего, фиксируется одним из параметров, передаваемых в функцию, либо же условиями внутри самого алгоритма. Это необходимо так как глубина не может быть бесконечной. Пример рекурсивной функции «Foo» на языке С++ (Листинг 1. Пример рекурсивной функции):

void Foo(int callNumber)

{

if(callNumber <= 0) return;

Foo(callNumber-1);

}

Листинг . Пример рекурсивной функции

Здесь целочисленный параметр «callNumber» являет собой глубину рекурсии.

Каждый раз при вызове внутри себя, функция будет получать параметр «callNumber» уменьшенный на 1, и когда «callNumber-1» станет равен 0, новый вызов функции не произойдет, а функция будет завершена.

# **1.2 Основные моменты программирования логических игр**

Существует огромное множество различных логических игр. Они обладают разной степенью сложности, в некоторых присутствует элемент случайности, а в некоторых нет, количество игроков также варьируется. При всём данном разнообразии существует и общие черты, присущее данным играм: чаще всего есть игровое поле, поделенное на «игровые поля», строгий набор правил, описывающих все аспекты игрового процесса, а также таким играм присуще жесткое разделение на ход каждого игрока, и когда ходит определённый игрок, другие не как не могут влиять на процесс игры.

При программировании не сложных логических игр, которые содержат не большое, ограниченное количество возможных ходов, не составит труда описать все возможные варианты развития игровой партии. К таким играм можно отнести «Крестики-Нолики». Но при программировании более сложных игр, такой подход становится невозможным, более того такие игры как «Шахматы» допускают возможность многократного повторении ходов, тем самым сводя количество вариантов развития игровой партии к огромным числам. Здесь приходится прибегать к рекурсивным алгоритмам. Разные игры имеют различные правила, где-то есть игровые фигуры, где-то нет, поэтому положим, что компьютер должен выбрать наилучший из ограниченного количества возможных ходов. Для этого, в зависимости от правил игры, необходимо предоставить оценку для каждого хода, некое число, величина которого являлась бы «силой» данного хода. Затем сделать наиболее «сильный» ход, дождаться хода игрока и вновь провести оценку.

Рисунок . Схема выбора оптимального хода

Данный алгоритм является основой искусственного интеллекта большинства логических игр.

# 1.3 Аналитическая функция оценки

При описании алгоритма выше, было введено понятие аналитической функции, или же функции оценки. Это такая функция, которая используется для расчета «силы» каждого возможного хода. Реализовать эту функцию можно по-разному, в зависимости от правил игры. В общем случае реализация подобной функции выглядит как оценка изменения обстановки после совершения оцениваемого хода, но этого бывает недостаточно, и функция оценки вызывается вновь, но уже для тех ходов что появятся после совершения оцениваемого хода и так далее, до определенной глубины рекурсии. Из этого можно сделать вывод, что аналитическая функция является основным элементом искусственного интеллекта для логической игры. Но расчет в сложных игра, к примеру в шахматах, всех возможных ходов до победы, является очень долгим процессом, а необходимость делать такие расчеты довольно часто, накладывает рамки. Такими рамками является глубина расчета, а также необходимость отбрасывания повторяющихся ходов. Но при написании искусственного интеллекта для более простых игр, можно упростить расчеты до глубины 1, а там, где возможно, считать до конца.

# 1.4 Работа с коллекцией игровых объектов

Логические игры практически всегда подразумевают взаимодействие игрока с набором игровых объектов (фигур, карт и т.п.), аналогичные взаимодействия приходится проводить и искусственному интеллекту. Для таких взаимодействий можно использовать циклы, но это плохо сказывается на производительности такого алгоритма и громоздко при написании самой программы. Более удобным и производительным решением будет использование LINQ (Language-Integrated Query) – удобного языка запроса к источнику данных, то есть, в контексте данного приложения, коллекции игровых объектов. LINQ позволяет оперировать с объектами реализующими интерфейс IEnumerable. Данный инструмент позволит быстро и удобно производить выборки, к примеру по определенным признакам выбрать клетки которые являются потенциальным вариантами для хода (Листинг 2. код пример LINQ).

var selectedTeams = from t in teams // определяем каждый объект из teams как t

                    where t.ToUpper().StartsWith("Б") //фильтрация по критерию

                    orderby t  // упорядочиваем по возрастанию

                    select t; // выбираем объект

Листинг . код пример LINQ

Отдельно стоит рассмотреть тип «var» — это анонимный тип. Данный тип данных позволяет легко инкапсулировать несколько свойств, представляя из себя их коллекцию. Свойства в такой коллекции доступны только для чтения. Такой тип подходит для сравнения и/или вывода информации результата выборки LINQ запроса.

# 1.5 Игровой процесс

На экране отображается игровая доска, состоящая из клеток, крестообразной формы (два на крест наложенных друг на друга прямоугольника, размером 7х3 клетки). Каждая клетка может быть занята курицей, лисой или может быть пустой. Клетки делятся на два типа, клетки с текстурой травы, и клетки с текстурой доски. На вторых необходимо разместить куриц для победы, это клетки с 1 по 6 и с 9 по 11, если вести отсчет от верха игровой доски. При старте игры, на клетках с 14 по 33 размещаются курицы, а на клетках 9 и 11 лисы. Курицы могут ходить на одну клетку вверх, вправо и влево, лисы на одну клетку вверх, вправо, влево и вниз. Лиса должна съесть курицу если они находятся на смежных клетках, и за курицей есть пустая клетка. При поедании курицы, лиса перепрыгивает через неё, и если после этого она может есть вновь (снова оказывает на клетке, на смежной которой расположена курица с пустой клеткой за ней), то лиса вновь ест, не передавай ход другому игрок. Лиса должна поедать наиболее длинную цепочку из куриц, если есть выбор. Эта модель очень сильно подобна поведению фигур в «шашках». Ход делается по очереди, начинают курицы. Если куриц осталось меньше 9ти, то засчитывается поражение, ибо оставшихся не хватит для победы.

# Глава 2. Разработка программного приложения

Приложение разработано на основе библиотеки Windows Forms, архитектура его состоит из набора классов, несколько для различных игровых объектов: Fox, Field, Chicken, GameObj, а также класса для реализации коллекции игровых объектов: Fields. Gameobj – это абстрактный класс, наследуемый от встроенного в библиотеку класса PictureBox. Тип класса является абстрактным, так как он используется для объявления общих элементов (методов, полей, свойств), которые будут использоваться и реализовываться во всех класса игровых объектов. Сам PictureBox являет собой класс объекта, представляющего из себя картинку. Он позволяет обрабатывать большинство событий, как и прочие компоненты Windows Forms, содержит свойства присущие объектам на плоскости: размеры и координаты; а также адрес изображения на жестком диске, что будет присвоено этому объекту. Этот адрес может быть как абсолютным, так и относительным каталогу из которого осуществляется запуск приложения.

Также существует класс Form1 – он является классом объекта формы, то есть в нем происходит создание игровых объектов и объектов реализующих интерфейс игры, перерисовка их на экране, а также вызов свойств этих объектов, при возникновении советующих событий (Рисунок 16. Диаграмма классов).

# 2.1 Подробное описание классов

Класс GameObj – это абстрактный класс, наследуемый от класса PictureBox. Он содержит поля «id», «loca», «selected». Все свойства используются для инкапсуляции полей. Методы являются абстрактными и не реализуются, но имеют свое предназначение.



Рисунок Схема класса GameObj

Таблица . Поля и свойства класса GameObj

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Описание** |
| id | int | «id» - целочисленное поле, используется для хранения порядкового номера игрового объекта. |
| loca | System.Drawing.Point | Представляет собой точку на плоскости, состоит из вещественных переменны X и Y, используется для хранения положения в координатах игровой доски. |
| selected | bool | «selected» - логическая переменная, хранит информацию о том, был ли выбран данный объект игроком. |

Таблица . методы и конструкторы класса GameObj

|  |  |
| --- | --- |
| **Название** | **Описание** |
| **[1]** | **[2]** |
| SelMy | Метод «SelMy» вызывается тогда, когда происходит выделение объекта, параметром является объект типа «Fields», то есть игровая доска, этот метод имеет тип «Void» - он ничего не возвращает. |
| NotSelMy | Метод «NotSelMy» вызывается для снятия выделения, параметром является объект типа «Fields», то есть игровая доска, этот метод имеет тип «Void» - он ничего не возвращает. |
| toClik | Метод «toClik» вызывается при нажатии кнопкой мыши по объекту, принимает два параметра: параметр типа «Fields», то есть игровая доска, параметр типа «GameObj» то есть выделенный в данный момент игровой объект. Параметры передаются по ссылке - метод изменяет передаваемые ей объекты. Тип метода «Void» - он ничего не возвращает. |
| myPower  **[1]** | Метод «myPower» - единственный целочисленный метод, параметром является коллекция «List<>» игровых объектов типа «GameObj».  **[2]** |
|  | Она так же является игровой доской. Метод рассчитывает и возвращает «силу» текущей клетки для искусственного интеллекта. |
| Step | Метод «Step» вызывается если искусственный интеллект использует данный игровой объект для совершения хода. Параметрами является объект типа «Fields» - исходное положение на игровой доске, измененяемая методом коллекция «List<>» игровых объектов типа «GameObj» - отображает текущее положение на игровой доске, и два изменяемых логических параметра: первый отвечает за возможность совершения хода, второй за то является ли данный ход первым. Тип метода «Void» - он ничего не возвращает. |

Класс «Field» - класс, используемый для описания объекта «Поле».



Рисунок . Схема класса Field

Таблица . Методы и конструкторы класса Field

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Field | Конструктор используемый для создания объекта класса (Листинг 3 конструктор "Field"). |
| SelMy | Смотри «Таблица 2. методы и конструкторы класса GameObj». |
| NotSelMy |
| toClik | Реализует перемещение куриц и искусственный интеллект лис, если есть выбранная курица. |
| myPower | Пустая реализация, ничего не выполняют. |
| Step |

public Field(System.Drawing.Point L/\*Координата на форме\*/, System.Drawing.Point l/\*Игровая координата\*/, System.Drawing.Size S/\*Размер\*/, string I/\*Путь к файлу изображения\*/, int id)

{

SizeMode = PictureBoxSizeMode.StretchImage;

Location = L;

Loc = l;

Size = S;

Image = new System.Drawing.Bitmap(I);

ImageLocation = I;

Visible = true;

ID = id;

}

Листинг конструктор "Field"

В конструктор передаются все основные параметры для создания объекта: координаты на форме; игровые координаты; размеры; путь к файлу с изображением; порядковый номер. Конструктор только сопоставляет переданные параметры со соответствующими свойствами.

Реализация метода «SelMy» заключается в поиске выделенного игрового объекта. Далее если такой элемент существует, то выделение произойдет только в том случае если координаты данного объекта позволяют выделенной курице совершить ход в данное поле. Если такой курицы нет, то выделение происходит вне зависимости от координат данного поля. Само выделение представляет собой изменение текстуры данного поля, на аналогичную, но выделенную. Адрес текстуры берётся из параметра объекта Field.

public override void SelMy(Fields desk) {

var se = from t in desk.F where t.Sell == true select t.Loc;

if (se.Count() > 0) {

if (((se.First().X - Loc.X == 1 || se.First().X - Loc.X == -1) && (se.First().Y == Loc.Y)) || ((se.First().X == Loc.X) && (se.First().Y - Loc.Y == 1)))

{

if (ImageLocation == desk.Desk) { Image = new System.Drawing.Bitmap(desk.DeskSEL); ImageLocation = desk.DeskSEL; }

else if (ImageLocation == desk.Grass) { Image = new System.Drawing.Bitmap(desk.GrassSEL); ImageLocation = desk.GrassSEL; }

}

}

else

{

if (ImageLocation == desk.Desk) { Image = new System.Drawing.Bitmap(desk.DeskSEL); ImageLocation = desk.DeskSEL; }

else if (ImageLocation == desk.Grass) { Image = new System.Drawing.Bitmap(desk.GrassSEL); ImageLocation = desk.GrassSEL; }

}

}

Листинг 4 Класс Field, метод SelMy

Метод «NotSelMy» работает аналогично, но не занимается каким-либо поиском, а снимает выделение с данного поля. Для этого аналогично происходит изменение текстуры с выделенной на не выделенную.

public override void NotSelMy(Fields desk)

{

if (ImageLocation == desk.DeskSEL) { Image = new System.Drawing.Bitmap(desk.Desk); ImageLocation = desk.Desk; }

else if (ImageLocation == desk.GrassSEL) { Image = new System.Drawing.Bitmap(desk.Grass); ImageLocation = desk.Grass; }

}

Листинг 5 Класс Field, метод NotSelMy

Метод «toClik» в данном классе играет важную часть реализации модели искусственного интеллекта. Метод проверяет наличие выбранной курицы, если такой нет то не происходит ни каких действий. Если курица есть, и она может осуществить ход в данную клетку, то выделенная курица принимает координаты, как игровые, так и реальные, такие же какие имеет данная клетка, вделанная курица перемещается на фронтовую часть по оси Z, и снимается выделение со всех куриц. Следом происходит реализация искусственного интеллекта:

Вычисляется сила позиций каждой лисы, то есть сколько куриц она может съесть из данного положения, затем, в цикле while, происходит вызов метода совершения наиболее сильного шага для лисы в лучше положении, до тех пор, пока есть возможность поедать куриц, либо если нет, то совершается ход в случайном из допустимых направлений. Далее, в зависимости от количества съеденных куриц, происходит воспроизведение соответствующего звука (Листинг 11 Метод toClik класса Field ).

Методы «myPower» и «Step» имеют пустую реализацию, и ничего не делают, так как это не требуется для данного объекта.

Класс «Chicken» - класс, используемый для описания игрового объекта «Курица».



Рисунок . Схема класса Chicken

Данный класс наследует от класса «GameObj».

Таблица . Методы и конструкторы класса Chicken.

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Chicken | Конструктор используемый для создания объекта класса. |
| SelMy | Смотри «Таблица 2. методы и конструкторы класса GameObj». |
| NotSelMy |
| toClik | Реализует выделение или снятие выделения с данного объекта при нажатии на него кнопкой мыши. |
| myPower | Пустая реализация, ничего не выполняют. |
| Step |

Приводить код методов «SelMy» и «NotSelMy» не имеет смысла, так как они занимаются только сменой текстур, аналогично методу «NotSelMy» класса «Field». Интересным является только метод «toClik», он не меняет текстуры, а занимается только сменой свойства «Sell» у выделенной курицы, и у текущей, а так же воспроизведением звука выбора курицы.

public override void toClik(ref Fields desk, ref GameObj SelectedCh)//выделение курицы

{

if (this.ImageLocation == desk.chDeskSEL || this.ImageLocation == desk.chGrassSEL)

{

if (SelectedCh != null && SelectedCh.ID != ID) //если была выделенна другая

{

SelectedCh.Sell = false;

SelectedCh.NotSelMy(desk);

}

if (Sell)//если выделенна была эта

{

Sell = false;

SelectedCh = null;

}

else//если не была

{

Sell = true;

SelectedCh = this;

}

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"Cud.wav");

simpleSound.Play();

}

}

Листинг 6. Метод toClik, класс Chicken

Класс «Fox» - класс, используемый для описания игрового объекта «Лиса». Наследует от класса «GameObj».



Рисунок . Схема класса Fox

Таблица . Методы и конструкторы класса Fox

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Fox | Конструктор используемый для создания объекта класса. |
| SelMy | Смотри «Таблица 2. методы и конструкторы класса GameObj». |
| NotSelMy |
| toClik | Пустая реализация, ничего не выполняет. |
| myPower | Вызывает функцию HowManyToKill для текущего положения лисы. |
| Step | Вызывает для того чтоб лиса совершила ход. Изменяет передаваемую игровую доску, перемещая на ней лису. Для реализации поедания цепочек куриц, вызывается до тех пор, пока можно есть, за это отвечает одна из передаваемых логических переменных. |
| HowManyToKill | Рассчитывает наибольшее число возможных поеданий для некой клетке на игральной доске |

Приводить код методов «SelMy» и «NotSelMy» не имеет смысла, так как они занимаются только сменой текстур, аналогично методу «NotSelMy» класса «Field». Метод «toClik» имеет пустую реализацию, так как при нажатии на лис ничего не происходит.

Метод «HowManyToKill» является реализацией функции оценки, о которой было сказано в ведении. Она является рекурсивной функцией, и возвращает наибольше количество куриц, которые могут быть съедены, из определенной клетки. Алгоритм работы функции таков:

Она убирает из переданной ей игровой доски, ту курицу которую «съели» на прошлом шаге, если такая есть. Затем ищет куриц на игровой доске вокруг данной, затем пустые клетки через одну от данной, затем куриц и лис через одну от данной. После этих выборок, оставляем только пустые клетки, расположенные за соседними курицами, и для каждой такой клетки вызывает эту же функцию оценки, результаты записываем в массив потенциальных поеданий, увеличив каждое на 1, так как при вызове мы должны учесть, что будет съедена курица, передаваемая параметром функции. Если таких нет, то возвращаем 0, если есть, то наибольшую из полученных оценок (Листинг 12. Аналитическая рекурсиваня функция оценки. Метод HowManyToKill класса Fox).

Метод «Step» выполняет аналогичные выборки, а также тоже вызывает метод «HowManyToKill» для выбора наилучшего хода, при цепочке поеданий. Но кроме этого он содержит отдельный алгоритм на тот случай когда лиса не имеет возможность есть, в таком случае направление для движения выбирается случайно, и если данная лиса не может совершить ход, она его пропускает, сообщая что ход окончен (Листинг 13. метод Step, класс Fox).

Класс «Fields» - класс коллекции игровых объектов.



Рисунок . Схема класса Fields.

Таблица . Поля и свойтсва класса Fields

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Описание** |
| **[1]** | **[2]** | **[3]** |
| chdesk | string | Путь к текстуре курицы на доске. |
| chdeskSEL | Путь к текстуре выделенной курицы на доске. |
| chgrass | Путь к текстуре курицы на траве. |
| chgrassSEL | Путь к текстуре выделенной курицы на траве. |
| desk | Путь к текстуре пустой клетки доски. |
| deskSEL | Путь к текстуре пустой выделенной клетки доски. |
| grass | Путь к текстуре пустой клетки травы. |
| grassSEL | Путь к текстуре пустой выделенной клетки травы. |
| Fdesk | Путь к текстуре лисы на доске. |
| FdeskSEL | Путь к текстуре выделенной лисы на доске. |
| Fgrass | Путь к текстуре лисы на траве. |
| FgrassSEL | Путь к текстуре выделенной лисы на траве. |
| F | List<GameObj> | Коллекция игровых объектов. |

Все свойства представляют собой инкапсуляцию полей.

Таблица . Методы и конструкторы класса Fields

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Fields | Конструктор используемый для создания объекта класса. Метод перегружен. При передаче без параметров создает пустой объект. С параметрами – создает игровое поле. |
| GetEnumerator | Вызывает данный метод от «List<GameObj>F», необходимо для того чтоб можно было обращаться в цикле «foreach» к объекту класса «Fields» как к его полю «F». |

Конструктор с параметрами «Fields» принимает адреса всех тектсур игровых объектов, размеры игрового объекта, (параметр один, так ка все размеры одинаковые) и координаты верхнего левого угла игровой доски (по умолчанию 0,0). Адреса задаются соответствующим свойствам, затем начинается цикл создания игрового поля и объектов. Он разделен на три части: первые две строки, строки с третей по пятую, и строки с 6той по 7мую. В эти три части добавляются пустые поля. Следом после этого к создаются и привязываются к полям курицы, и в конце, два последних шага цикла приходятся на создание и привязку к полям лис. Привязка происходит благодарю тому что все игровые объекты кроме полей, требуют для своего создание передать то пустые поля, на которых они будут расположены, порядковый номер, и адрес изначальной текстуры. Поэтому позиционировать необходимо только объекты «Field» (Листинг 14 Конструктор класса Fields).

# 2.2 Описание используемых компонентов форм

Каждый игровой объект представляет из себя модифицированный объект форм «PictureBox», но помимо игровой доски присутствует еще несколько компонентов форм, используемых для отображения информации и пользовательского интерфейса.

private System.Windows.Forms.MenuStrip menuStrip1;

private System.Windows.Forms.ToolStripMenuItem toolStripTextBox1;

private System.Windows.Forms.ToolStripMenuItem помощьToolStripMenuItem;

private System.Windows.Forms.Label label1;

Листинг . Компоненты форм

Компонент «menuStrip1» это строка меню программы располагаемая под заголовком. «toolStripTextBox1» и «помощьToolStripMenuItem» это элементы данного меню, первое отвечает за начало новой игры, второе за выдачу сообщения с правилами игры.

«label1» — это текст, который не может быть изменен пользователем программы, он расположен в левом нижнем углу формы, отображает текущее количество куриц.

Создание игровых объектов и элементов меню, происходит в методе Form1\_Load (Листинг 15 Генерация элементов форм). Там же происходит задача обработчиков событий.

Большинство обработчиков занимаются только вызовом соответствующего метода игривого объекта

private void MousHover(object sender, EventArgs e)//Мышь наведена

{

((GameObj)sender).SelMy(desk);

}

Листинг Обработчик события наведения курсора на объект

Но обработчик нажатия на игровой объект более сложен, помимо вызова советующего метода он занимается перерисовкой игровых объектов в случае уничтожения данных в игровой коллекции, а также проверяет условия победы и поражения. В случае выполнения одного из условий, выводит советующий звуковой сигнал, и начинает игру заново (Листинг 16 Обработчик события нажатия на игровой объект).

Обработчик нажатия на «toolStripTextBox1» выглядит подобно последним строкам «Обработчика события нажатия на игровой объект».

private void toolStripTextBox1\_Click(object sender, EventArgs e)

{

DialogResult status = MessageBox.Show("Вы уверены?", "Новая игра", MessageBoxButtons.YesNo);

if (status == DialogResult.Yes)

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"NewRoundIn.wav");

simpleSound.Play();

EventArgs ee = new EventArgs();

desk = new Fields();

Controls.Clear();

this.menuStrip1 = new System.Windows.Forms.MenuStrip();

this.toolStripTextBox1 = new System.Windows.Forms.ToolStripMenuItem();

this.помощьToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.Form1\_Load(null, ee);

menuStrip1.Items.Add(помощьToolStripMenuItem);

Controls.Add(label1);

}

}

Листинг . Обработчик нажатия на toolStripTextBox1

Так же, существует событие связанно не с элементом формы, или игровым объектом, а с самой формой. Это событие закрытия формы. Она выдает окно с запросом на подтверждение завершения работы, аналогичное использовалось ранее при досрочном начале новой игры (при нажатии на «toolStripTextBox1»).

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

DialogResult status = MessageBox.Show("Вы уверены?", "Выход", MessageBoxButtons.YesNo);

if(status == DialogResult.No)

{

e.Cancel = true;

}

}

Листинг . Обработчик закрытия формы

# 2.3 Описание разработанных алгоритмов

Алгоритм игрового процесса:

1. Формируется игровое поле, так как было описано в поставленной задаче;
2. Пользователь наводит курсор на игровой объект, он выделяется;
3. Пользователь нажимает на игровой объект;
   1. Если это курица, выделение фиксируется, выделение с остальных кур снимается;
   2. Если это лиса, то ничего не происходит;
   3. Если это пустое поле, то в зависимости от того была ли выбрана курица до этого, и может ли она совершить сюда ход, происходит перемещение курицы, либо не происходит ничего;
4. Если курица совершила ход, то ход делают лисы;
5. Если оставшихся после этого кур меньше чем 9, то игра заканчивается поражением;
6. Если курицы заняли клетки необходимые для победы, то игра заканчивается победой;

Далее этот алгоритм представлен в виде блок схемы на следующей странице (Рисунок 8. Блок схема алгоритма):

А0

Нет

Нет

Нет, значит это поле

начало

Создание игр. полей и элементов форм

Нажатие на игровой объект

Это лиса?

Да

Это курица?

Да

Фиксируем выделение, снимание выделения с прочих

Нет

Есть выбранная курица?

Она может совершить ход в данную клетку?

Да

Да

Перемещаем выделенную курицу

Ход лис

А0

Куриц осталось менее 9ти?

Сообщаем о поражении

Курицы заняли клетки для победы?

А1

Да

Сообщаем о победе

А1

Нет

конец

Рисунок . Блок схема алгоритма

# Глава 3. Описание пользовательского интерфейса

Приложение после запуска готово для начала партии. Имеется игровое поле, и пункты меню (Рисунок 1. Вид готового приложения).

Теперь рассмотрим изменение при наведении курсора на какой-либо игровой объект (Рисунок 9. Пример выделения):



Рисунок . Пример выделения

Теперь рассмотрим изменение после совершения 1 хода (Рисунок 10. Совершен один ход):

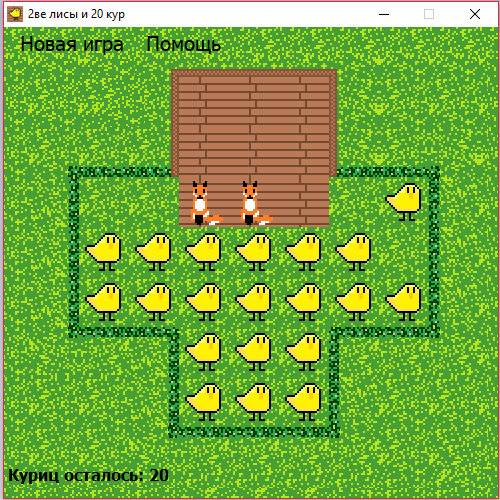


Рисунок . Совершен один ход

Аналогично продолжим игру и рассмотрим более поздний этап партии (Рисунок 11. Поздний этап игры):

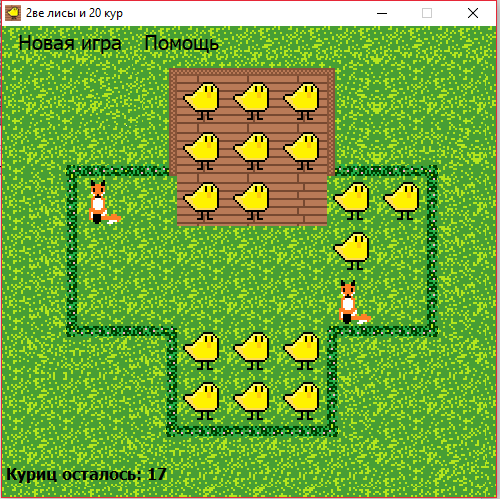


Рисунок . Поздний этап игры

Теперь рассмотрим сообщения о победе (Рисунок 12. Сообщение о победе), поражении (Рисунок 14. Сообщение о поражении), изменение счетчика куриц на игровом поле (Рисунок 13. Куриц осталось 12), а также сообщение выводящее на экран подсказку с правилами игры (Рисунок 15. Сообщение с правилами игры).

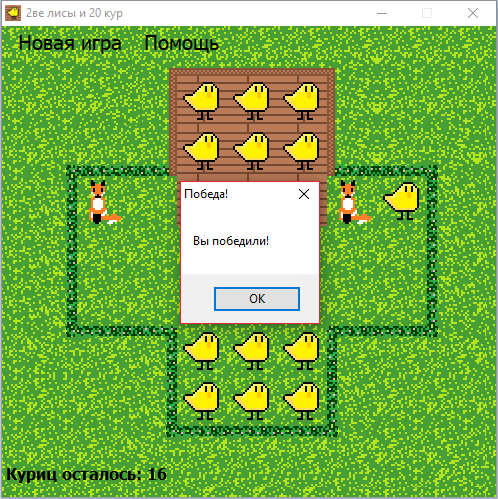


Рисунок . Сообщение о победе

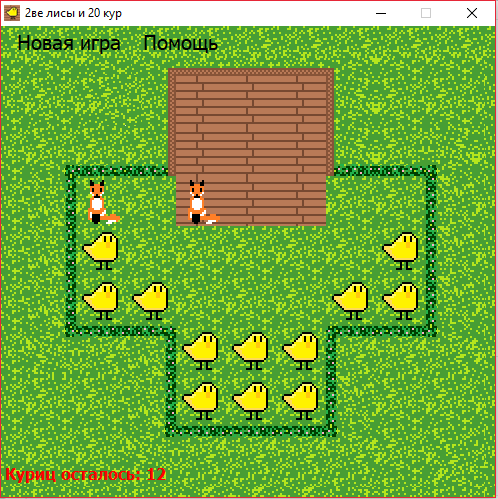


Рисунок . Куриц осталось 12

Когда куриц остается 12 или меньше, цвет счетчика меняется с черного на красный (Рисунок 13. Куриц осталось 12).



Рисунок . Сообщение о поражении

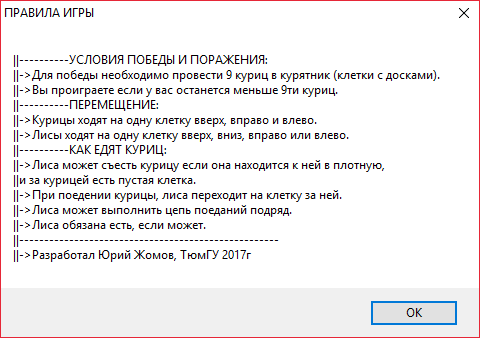


Рисунок . Сообщение с правилами игры

# ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является приложение «Логическая игра «Две лисы и двадцать кур», в которой реализован простой искусственный интеллект, способный играть за лис в соответствии с целями, поставленными при разработке приложения. За время разработки приложения была изучена работа с Microsoft Forms, различные алгоритмы программирования искусственного интеллекта в логических играх, применение рекурсивных алгоритмов. Рассмотрено применение объектно-ориентированного программирования на практике, а также особенности языка программирование C# и его составляющих элементов таких как LINQ запросы. Изучена документация и ГОСТы по разработке блок схем и алгоритмов, а также оформления документации и научных работы.

Основной проблемой было составление алгоритма работы искусственного интеллекта, и применение рекурсивных алгоритмов при его реализации.

Все поставленные цели и задачи полностью реализованы:

* изучена научная литература в предметной области данной курсовой работы, необходимая для решения поставленных задач;
* разработан алгоритм игрового процесса, а также интерфейс для взаимодействия пользователя и приложения;
* реализованы все необходимые алгоритмы на языке программирования C#;
* приложение скомпилировано, отлажено и прошла тестирование;
* отчет о проделанной работе составлен.

# СПИСОК ЛИТЕРАТУРЫ

1. Корнилов, Е.Н. Программирование шахмат и других логических игр [Текст] / Корнилов Е.Н.: ПХВ-Петербург, 2005. – 267 с.

2. Троелсен, Э. Языки программирования C# 5.0 и платформа .NET 4.5,6-е изд. [Текст] / Троелсен, Эндрю; перевод с англ. Н.Ю. Артеменко – М.: ООО “И.Д. Вильямс”, 2013. – 1312 с.: ил. – Парал. тит. англ.

3. ГОСТ 7.1—2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления [Текст] – Взамен ГОСТ 7.1—84, ГОСТ 7.16—79, ГОСТ 7.18—79, ГОСТ 7.34—81, ГОСТ 7.40—82; введ. 2004–07–01. – Изд-во стандартов, cop. 2002. — 17 с.

4. ГОСТ Р 7.0.5—2008 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая ссылка. Общие требования и правила составления [Текст]: офиц. изд. – введ. 2009–01–01. – Изд-во стандартов, cop. 2002. — 15 с.

5. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения [Текст] – Взамен ГОСТ 19.002-80, ГОСТ 19.003-80; введ. 1992-01-01. – Гос. комитет СССР по управлению качеством продукции и стандартам от 26.12.90 N 3294 - М.: Стандартинформ, 2010. – 22 с.

# ПРИЛОЖЕНИЯ

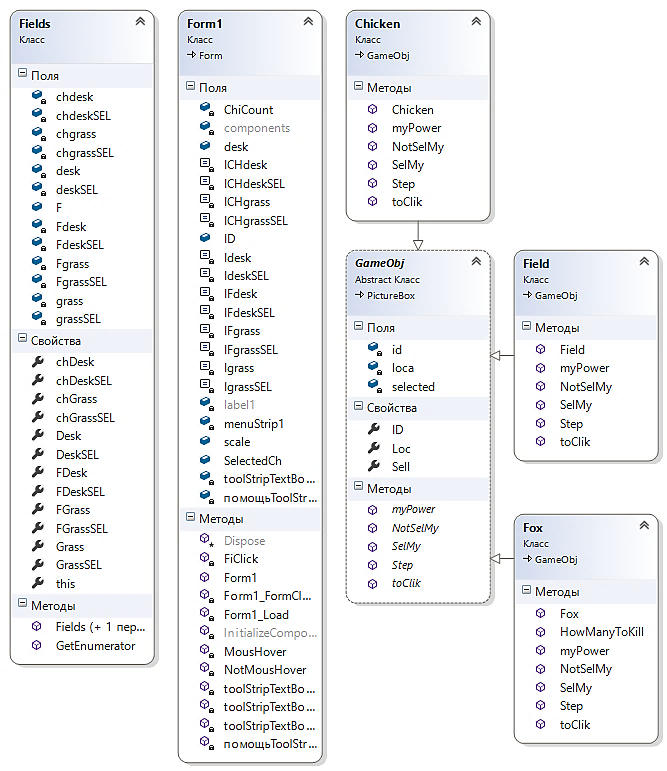


Рисунок 16. Диаграмма классов

public override void toClik(ref Fields desk, ref GameObj SelectedCh)

{

if (SelectedCh != null)//если курица выбранна

{

Point SelectedField = SelectedCh.Loc;//получаем её игровые координаты

if (((Loc.X == SelectedField.X) && (Loc.Y - SelectedField.Y == -1)) || (Math.Abs(Loc.X - SelectedField.X) == 1 && Loc.Y == SelectedField.Y))

{

if (desk.F[desk.F.IndexOf(SelectedCh)].Sell == true)

{

desk.F[desk.F.IndexOf(SelectedCh)].Sell = false;

if (ImageLocation == desk.DeskSEL || ImageLocation == desk.Desk)

{

desk.F[desk.F.IndexOf(SelectedCh)].Image = new System.Drawing.Bitmap(desk.chDesk);

desk.F[desk.F.IndexOf(SelectedCh)].ImageLocation = desk.chDesk;

}

else

{

desk.F[desk.F.IndexOf(SelectedCh)].Image = new System.Drawing.Bitmap(desk.chGrass);

desk.F[desk.F.IndexOf(SelectedCh)].ImageLocation = desk.chGrass;

}

desk.F[desk.F.IndexOf(SelectedCh)].Loc = Loc;

desk.F[desk.F.IndexOf(SelectedCh)].Location = Location;

desk.F[desk.F.IndexOf(SelectedCh)].BringToFront();

SelectedCh = null;

}

#region ИИ лис

int BeforeKills = desk.F.Count;

bool tomove = true; //флаг для нескольких поеданий подряд

bool first = true; //флаг если поеданий еще не было

//Вычисления силы позиций на данный момент

int P1 = desk.F[desk.F.Count - 2].myPower(desk.F);

int P2 = desk.F[desk.F.Count - 1].myPower(desk.F);

if(P1 > P2) //оценка позиий лис

{

while (tomove) //выполнение нескольких или одного поедания

{

desk.F[desk.F.Count - 2].Step(desk,ref desk.F, ref tomove, first);

first = false;

}

}

else if (P1 < P2)//оценка позиий лис

{

while (tomove) //выполнение нескольких или одного поедания

{

desk.F[desk.F.Count - 1].Step(desk, ref desk.F, ref tomove, first);

first = false;

}

}

else //если позиции равносильны

{

Random rand = new Random();

int r = rand.Next(0,2);

if (r == 1)

{

while (tomove)

{

System.Drawing.Point Before = desk.F[desk.F.Count - 2].Loc;

desk.F[desk.F.Count - 2].Step(desk,ref desk.F, ref tomove, first);

//если выбранная лиса не изменила положения и не ела (то есть не ходила)

if (P1 == 0 && desk.F[desk.F.Count - 2].Loc == Before) desk.F[desk.F.Count - 1].Step(desk, ref desk.F, ref tomove, first);

first = false;

}

}

else

{

while (tomove)

{

System.Drawing.Point Before = desk.F[desk.F.Count - 1].Loc;

desk.F[desk.F.Count - 1].Step(desk, ref desk.F, ref tomove, first);

//если выбранная лиса не изменила положения и не ела (то есть не ходила)

if (P2 == 0 && desk.F[desk.F.Count - 1].Loc == Before) desk.F[desk.F.Count - 2].Step(desk, ref desk.F, ref tomove, first);

first = false;

}

}

}

#region Звук

switch (BeforeKills - desk.F.Count)

{

case 0: break;

case 1:

{

if (BeforeKills == 55)

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"firstblood.wav");

simpleSound.Play();

}

break;

}

case 2:

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"doublekill.wav");

simpleSound.Play();

break;

}

case 3:

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"multikill.wav");

simpleSound.Play();

break;

}

case 4:

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"megakill.wav");

simpleSound.Play();

break;

}

case 5:

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"ultrakill.wav");

simpleSound.Play();

break;

}

case 6:

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"monsterkill.wav");

simpleSound.Play();

break;

}

default:

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"rampage.wav");

simpleSound.Play();

break;

}

}

#endregion

}

}

#endregion

}

Листинг 11 Метод toClik класса Field

public int HowManyToKill(List<GameObj> G/\*Игральная доска\*/, System.Drawing.Point SLoc/\*Коодинаты клетки для оценки\*/, GameObj FirstBlood/\*Курица которую придется съесть\*/, int Kill = 0/\*Кооличеств ранее "убитых" куриц при проведении оценки\*/)

{

List<GameObj> F = new List<GameObj>();

F.AddRange(G);

if (FirstBlood != null) F.Remove(FirstBlood);//Убираем "съеденную" курицу на данном ходу

Kill++;

List <int> Kills = new List<int>();//создаем массив из потенциальных "поеданий" для потенциальных ходов

var ChickensAround = from t in F

where

((((t.Loc.X - SLoc.X == 1 || t.Loc.X - SLoc.X == -1) && (t.Loc.Y == SLoc.Y) && (t.ID > 32 && t.ID < 53))) ||

(((t.Loc.Y - SLoc.Y == 1 || t.Loc.Y - SLoc.Y == -1) && (t.Loc.X == SLoc.X) && (t.ID > 32 && t.ID < 53))))

select t; //Ищем куриц вокруг данной клетки в радиусе 1

List<GameObj> FieldsNearChickens = new List<GameObj>();

var FieldsNearChickenss = from t in F

where

((((t.Loc.X - SLoc.X == 2 || t.Loc.X - SLoc.X == -2) && (t.Loc.Y == SLoc.Y))) ||

(((t.Loc.Y - SLoc.Y == 2 || t.Loc.Y - SLoc.Y == -2) && (t.Loc.X == SLoc.X)))) && (t.ID < 33)

select t; //Ищем пустые клетки вокруг данной в радиусе 2

FieldsNearChickens.AddRange(FieldsNearChickenss.ToArray());

var ChickensAround2 = from t in F

where

((((t.Loc.X - SLoc.X == 2 || t.Loc.X - SLoc.X == -2) && (t.Loc.Y == SLoc.Y) && (t.ID > 32))) ||

(((t.Loc.Y - SLoc.Y == 2 || t.Loc.Y - SLoc.Y == -2) && (t.Loc.X == SLoc.X) && (t.ID > 32))))

select t; //Ищем куриц и лис вокруг данной клетки в радиусе 2

for (int i = 0; i < ChickensAround2.Count(); i++) //убираем занятые клетки в радиусе 2

{

var bedField = from t in FieldsNearChickens

where (t.Loc == ChickensAround2.ToArray()[i].Loc)

select t;

if (bedField.Count() > 0) FieldsNearChickens.Remove(bedField.First());

}

for (int i = 0; i < ChickensAround.Count(); i++)

{

var AbleToMove = from t in FieldsNearChickens

where ((t.Loc.X - ChickensAround.ToArray()[i].Loc.X == 1 || t.Loc.X - ChickensAround.ToArray()[i].Loc.X == -1) && (t.Loc.Y == ChickensAround.ToArray()[i].Loc.Y)) ||

((t.Loc.Y - ChickensAround.ToArray()[i].Loc.Y == 1 || t.Loc.Y - ChickensAround.ToArray()[i].Loc.Y == -1) && (t.Loc.X == ChickensAround.ToArray()[i].Loc.X))

select t; //Выбираем для данной курицы пустую клетку за ней из пустых клеток выбранных до этого

if (AbleToMove.Count() > 0) Kills.Add(HowManyToKill(F, AbleToMove.First().Loc, ChickensAround.ToArray()[i], Kill)+1); //если у данной курицы есть пустая клетка для хода, то вызываем оценучную функцию для нее

}//повторяем для каждой курицы вокруг данной клетки

if (Kills.Count > 0) return Kills.Max();//возвращаем наибольшее число вохможных убийств (оценка окончена для данной клетки)

else return 0;

}

Листинг . Аналитическая рекурсиваня функция оценки. Метод HowManyToKill класса Fox

public override void Step(Fields desk,ref List<GameObj> F/\*Игральная доска\*/, ref bool CanMove/\*Флаг цепочки поеданий\*/, bool first = true/\*Первый ли это шаг\*/)

{

var ChickensAround = from t in F where

((((t.Loc.X - Loc.X == 1 || t.Loc.X - Loc.X == -1) && (t.Loc.Y == Loc.Y) && (t.ID > 32 && t.ID < 53))) ||

(((t.Loc.Y - Loc.Y == 1 || t.Loc.Y - Loc.Y == -1) && (t.Loc.X == Loc.X) && (t.ID > 32 && t.ID < 53))))

select t; //Ищем куриц вокруг данной клетки в радиусе 1

List<GameObj> FieldsNearChickens = new List<GameObj>();

var FieldsNearChickenss = from t in F

where

((((t.Loc.X - Loc.X == 2 || t.Loc.X - Loc.X == -2) && (t.Loc.Y == Loc.Y))) ||

(((t.Loc.Y - Loc.Y == 2 || t.Loc.Y - Loc.Y == -2) && (t.Loc.X == Loc.X)))) && (t.ID < 33)

select t; //Ищем пустые клетки вокруг данной в радиусе 2

FieldsNearChickens.AddRange(FieldsNearChickenss.ToArray());

var ChickensAround2 = from t in F where

((((t.Loc.X - Loc.X == 2 || t.Loc.X - Loc.X == -2) && (t.Loc.Y == Loc.Y) && (t.ID > 32))) ||

(((t.Loc.Y - Loc.Y == 2 || t.Loc.Y - Loc.Y == -2) && (t.Loc.X == Loc.X) && (t.ID > 32))))

select t; //Ищем куриц и лис вокруг данной клетки в радиусе 2

for (int i = 0; i < ChickensAround2.Count(); i++) //убираем занятые клетки в радиусе 2

{

var bedField = from t in FieldsNearChickens

where (t.Loc == ChickensAround2.ToArray()[i].Loc)

select t;

if (bedField.Count() > 0) FieldsNearChickens.Remove(bedField.First());

}

List<int> Kills = new List<int>();//создаем массив из потенциальных "поеданий" для потенциальных ходов

List<GameObj> MustDie = new List<GameObj>();//массив из куриц которые могут быть съеденны на данном шаге

List<GameObj> ToMove = new List<GameObj>(); //массив из клеток за курицами куда переметится лиса после поедания

foreach (GameObj Chicken in ChickensAround)

{

var AbleToMove = from t in FieldsNearChickens

where ((t.Loc.X - Chicken.Loc.X == 1 || t.Loc.X - Chicken.Loc.X == -1) && (t.Loc.Y == Chicken.Loc.Y)) ||

((t.Loc.Y - Chicken.Loc.Y == 1 || t.Loc.Y - Chicken.Loc.Y == -1) && (t.Loc.X == Chicken.Loc.X))

select t; //Выбираем для данной курицы пустую клетку за ней из пустых клеток выбранных до этого

if (AbleToMove.Count() > 0 &&(Chicken.Loc.X != 2 || Chicken.Loc.Y != 0))//если за данной курицей есть пустая клетка вносим данные в листы

{

MustDie.Add(Chicken);//курицу в лист к возможным для поедания

ToMove.Add(AbleToMove.First());//клетку за курицей к возможным клеткам для перемещния лисы

Kills.Add(HowManyToKill(F, AbleToMove.First().Loc, Chicken)); // для этого потенцильного шага вызываем функцию оценки

}

}//повторяем для каждой курицы вокруг данной клетки

if (Kills.Count() > 0) //если есть шаги с поеданиями

{

//выбираме максимальные по силе вариант хода

this.Loc = ToMove[Kills.IndexOf(Kills.Max())].Loc; //смена игровых координат

this.Location = ToMove[Kills.IndexOf(Kills.Max())].Location;//смена реальных

//смена текстур если необходима

if (ToMove[Kills.IndexOf(Kills.Max())].ImageLocation == desk.Desk) { Image = new System.Drawing.Bitmap(desk.FDesk); ImageLocation = desk.FDesk; }

else if (ToMove[Kills.IndexOf(Kills.Max())].ImageLocation == desk.Grass) { Image = new System.Drawing.Bitmap(desk.FGrass); ImageLocation = desk.FGrass; }

F.Remove(MustDie[Kills.IndexOf(Kills.Max())]);//удалиние убитой курицы

}

else if (ChickensAround.Count() != 4 && first == true) // если лиса не окружена и может ходить, но не может есть, и до этого не было поеданий

{

var ChickensAndFoxAround = from t in F

where

((((t.Loc.X - Loc.X == 1 || t.Loc.X - Loc.X == -1) && (t.Loc.Y == Loc.Y) && (t.ID > 32))) ||

(((t.Loc.Y - Loc.Y == 1 || t.Loc.Y - Loc.Y == -1) && (t.Loc.X == Loc.X) && (t.ID > 32))))

select t;//ищем куриц и лис вокруг

var FieldsAroundd = from t in F

where

((((t.Loc.X - Loc.X == 1 || t.Loc.X - Loc.X == -1) && (t.Loc.Y == Loc.Y) && (t.ID > -1 && t.ID < 33))) ||

(((t.Loc.Y - Loc.Y == 1 || t.Loc.Y - Loc.Y == -1) && (t.Loc.X == Loc.X) && (t.ID > -1 && t.ID < 33))))

select t; //Ищем пустые клетки

List <GameObj> FieldsAround= new List<GameObj>();

FieldsAround.AddRange(FieldsAroundd.ToArray());

for (int i = 0; i < ChickensAndFoxAround.Count(); i++)

{

var bedField = from t in FieldsAround

where (t.Loc == ChickensAndFoxAround.ToArray()[i].Loc)

select t;

if (bedField.Count() > 0) FieldsAround.Remove(bedField.First());

}//убираем занятые клетки

if (FieldsAround.Count > 0) //если есть не занятые клетки - совершаем ход

{

Random rand = new Random();

int r = rand.Next(0,FieldsAround.Count());

Loc = FieldsAround.ToArray()[r].Loc;

Location = FieldsAround.ToArray()[r].Location;

if (FieldsAround.ToArray()[r].ImageLocation == desk.Desk) { Image = new System.Drawing.Bitmap(desk.FDesk); ImageLocation = desk.FDesk; }

else if (FieldsAround.ToArray()[r].ImageLocation == desk.Grass) { Image = new System.Drawing.Bitmap(desk.FGrass); ImageLocation = desk.FGrass; }

}

CanMove = false;//ход окончен

}

//если лиса полностью окружена и не может сходить

else CanMove = false;//ход окончен

}

Листинг 13. метод Step, класс Fox

public Fields(string DeskImg, string DeskSELImg, string GrassImg, string GrassSELImg, string chDeskImg, string chDeskSELImg, string chGrassImg, string chGrassSELImg, string FDeskImg, string FDeskSELImg, string FGrassImg, string FGrassSELImg/\*Адреса текстур\*/, System.Drawing.Size S/\*Размеры полей\*/, int StartX = 0, int StartY = 0/\*Начальные координаты\*/)

{

F = new List<GameObj>();//Выделяем память для хранения данных полей

Grass = GrassImg;

GrassSEL = GrassSELImg;

Desk = DeskImg;

//Добавляем куриц

chGrass = chGrassImg;

chGrassSEL = chGrassSELImg;

chDesk = chDeskImg;

chDeskSEL = chDeskSELImg;

//Добавляем лис

FGrass = FGrassImg;

FGrassSEL = FGrassSELImg;

FDesk = FDeskImg;

FDeskSEL = FDeskSELImg;

int i = 0;

DeskSEL = DeskSELImg;//запоминает текстуры

int X = StartX + S.Width\*2;//Создаем точки для начала отрисовки (Y не используется, за ненадобностью)

System.Drawing.Point L = new System.Drawing.Point(X, StartY);//Координата на форме

System.Drawing.Point l = new System.Drawing.Point(2,0);//Игровая координата

for (i = 0; i < 55; i++)//Задача парметров для полей игрового поля в соответствии с заданием курсовой работы

{

if (i < 6)

{

if (i == 3) { L.Y += S.Height; L.X = X; l.Y += 1; l.X = 2;}

else if (i != 0)

{

L.X += S.Width;

l.X += 1;

}

F.Add(new Field(L, l, S, Desk,i));

}

else if (i > 5 && i < 27)

{

if (i == 6 || i == 13 || i == 20) { L.Y += S.Height; L.X = X-S.Width\*2; l.Y += 1; l.X = 0; }

else

{

L.X += S.Width;

l.X += 1;

}

if(i > 7 && i <11) F.Add(new Field(L, l, S, Desk,i));

else F.Add(new Field(L, l, S, Grass,i));

}

else if(i > 26 && i < 33)

{

if (i == 27 || i == 30) { L.Y += S.Height; L.X = X; l.Y += 1; l.X = 2; }

else

{

L.X += S.Width;

l.X += 1;

}

F.Add(new Field(L, l, S, Grass,i));

}

else if (i > 32 && i < 53){ F.Add(new Chicken(F[i-20], chGrass, i)); }

else { if (i == 53) F.Add(new Fox(F[8], FDesk,i));

else F.Add(new Fox(F[10], FDesk,i));}

}

}

Листинг 14 Конструктор класса Fields

private void Form1\_Load(object sender, EventArgs e) //Генерация игровых объектов

{

ChiCount = 20;

this.label1.Text = "Куриц осталось: " + ChiCount;

this.label1.ForeColor = Color.Black;

this.menuStrip1 = new System.Windows.Forms.MenuStrip();

this.toolStripTextBox1 = new System.Windows.Forms.ToolStripMenuItem();

this.помощьToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.menuStrip1.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(192)))), ((int)(((byte)(0)))));

this.menuStrip1.BackgroundImage = new System.Drawing.Bitmap(Igrass);

this.menuStrip1.Font = new System.Drawing.Font("Segoe UI", 15F, System.Drawing.FontStyle.Bold);

this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.toolStripTextBox1,

this.помощьToolStripMenuItem});

this.menuStrip1.Location = new System.Drawing.Point(0, 0);

this.menuStrip1.Name = "menuStrip1";

this.menuStrip1.Size = new System.Drawing.Size(484, 32);

this.menuStrip1.TabIndex = 0;

this.menuStrip1.Text = "menuStrip1";

//

// toolStripTextBox1

//

this.toolStripTextBox1.BackColor = System.Drawing.Color.Transparent;

this.toolStripTextBox1.Font = new System.Drawing.Font("Tahoma", 15F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.toolStripTextBox1.ForeColor = System.Drawing.Color.Black;

this.toolStripTextBox1.Name = "toolStripTextBox1";

this.toolStripTextBox1.RightToLeft = System.Windows.Forms.RightToLeft.No;

this.toolStripTextBox1.Size = new System.Drawing.Size(126, 28);

this.toolStripTextBox1.Text = "Новая игра";

this.toolStripTextBox1.Click += new System.EventHandler(this.toolStripTextBox1\_Click);

this.toolStripTextBox1.MouseLeave += new System.EventHandler(this.toolStripTextBox1\_MouseLeave);

this.toolStripTextBox1.MouseHover += new System.EventHandler(this.toolStripTextBox1\_MouseHover);

menuStrip1.Items.Add(toolStripTextBox1);

//

// помощьToolStripMenuItem

//

this.помощьToolStripMenuItem.Font = new System.Drawing.Font("Tahoma", 15F);

this.помощьToolStripMenuItem.ForeColor = System.Drawing.Color.Black;

this.помощьToolStripMenuItem.Name = "помощьToolStripMenuItem";

this.помощьToolStripMenuItem.Size = new System.Drawing.Size(97, 28);

this.помощьToolStripMenuItem.Text = "Помощь";

this.помощьToolStripMenuItem.Click += new System.EventHandler(this.помощьToolStripMenuItem\_Click);

this.помощьToolStripMenuItem.MouseLeave += new System.EventHandler(this.toolStripTextBox1\_MouseLeave);

this.помощьToolStripMenuItem.MouseHover += new System.EventHandler(this.toolStripTextBox1\_MouseHover);

menuStrip1.Items.Add(помощьToolStripMenuItem);

//

this.Controls.Add(menuStrip1);

this.FormBorderStyle = FormBorderStyle.FixedDialog;

this.Text = "2ве лисы и 20 кур";

//создние игровых полей, кур, лис

desk = new Fields(Idesk, IdeskSEL, Igrass, IgrassSEL, ICHdesk, ICHdeskSEL, ICHgrass, ICHgrassSEL, IFdesk, IFdeskSEL, IFgrass, IFgrassSEL, scale, 75, 50);

Controls.AddRange(desk.F.ToArray());

for (int i = 0; i < 33; i++)

{

desk[i].SendToBack();

desk[i].MouseHover += new System.EventHandler(this.MousHover);

desk[i].MouseLeave += new System.EventHandler(this.NotMousHover);

desk[i].Click += new System.EventHandler(this.FiClick);

}

for (int i = 33; i < 55; i++)

{

desk[i].BringToFront();

desk[i].MouseHover += new System.EventHandler(this.MousHover);

desk[i].MouseLeave += new System.EventHandler(this.NotMousHover);

desk[i].Click += new System.EventHandler(this.FiClick);

}

}

Листинг 15 Генерация элементов форм

private void FiClick(object sender, EventArgs e)//Обработка нажатия на клетку

{

((GameObj)sender).toClik(ref desk, ref SelectedCh);

//перерисовка при удалении куриц

GameObj[] A = new GameObj[Controls.Count-2];

if (desk.F.Count != Controls.Count - 2)

{

int j = 0;

for (int i = 0; i < Controls.Count; i++)

{

try

{

A[j] = (GameObj)Controls[i];

j++;

}

catch { }

}

List<GameObj> B = A.ToList();

foreach (GameObj D in desk.F)

{

B.Remove(D);

}

foreach (GameObj D in B)

{

Controls.Remove(D);

}

ChiCount = ChiCount - B.Count;

this.label1.Text = "Куриц осталось: " + ChiCount;

if(ChiCount < 13)this.label1.ForeColor = Color.Red;

}

//проверка условий победы/поражения

var TryFail = from win in desk.F where (win.ID > 32 && win.ID < 53) select win;

var TryWin = from win in desk.F where (win.Loc.Y < 3 && win.Loc.X > 1 && win.Loc.X < 5 && (win.ID > 32 && win.ID < 53)) select win;

if (TryWin.Count() == 9)

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"You\_have\_won\_the\_match.wav");

simpleSound.Play();

if (desk.F.Count == 55)

{

simpleSound = new System.Media.SoundPlayer(@"Flawless\_victory.wav");

simpleSound.Play();

}

DialogResult status = MessageBox.Show("Вы победили!", "Победа!", MessageBoxButtons.OK);

simpleSound = new System.Media.SoundPlayer(@"NewRoundIn.wav");

simpleSound.Play();

EventArgs ee = new EventArgs();

desk = new Fields();

Controls.Clear();

this.menuStrip1 = new System.Windows.Forms.MenuStrip();

this.toolStripTextBox1 = new System.Windows.Forms.ToolStripMenuItem();

this.помощьToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.Form1\_Load(null, ee);

menuStrip1.Items.Add(помощьToolStripMenuItem);

Controls.Add(label1);

}

else if (TryFail.Count() < 9)

{

System.Media.SoundPlayer simpleSound = new System.Media.SoundPlayer(@"You\_have\_lost\_the\_match.wav");

simpleSound.Play();

DialogResult status = MessageBox.Show("Вы проиграли!", "Поражение!", MessageBoxButtons.OK);

simpleSound = new System.Media.SoundPlayer(@"NewRoundIn.wav");

simpleSound.Play();

EventArgs ee = new EventArgs();

desk = new Fields();

Controls.Clear();

this.menuStrip1 = new System.Windows.Forms.MenuStrip();

this.toolStripTextBox1 = new System.Windows.Forms.ToolStripMenuItem();

this.помощьToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.Form1\_Load(null, ee);

menuStrip1.Items.Add(помощьToolStripMenuItem);

Controls.Add(label1);

}

}

Листинг Обработчик события нажатия на игровой объект