**6МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет по лабораторной работе №6**

по курсу «Объектно ориентированное программирование»

**«Создание графического**

**интерфейса#»**

Выполнил:

студент гр. КТбо2-6

Коновалов Н.Д.

Таганрог 2020

Оглавление

Техническое задание3

Выполнение задания3

1.Спецификация классов3

2.Диаграмма классов8

3.Листинг8

**Вариант № 15**

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Разработайте графический интерфейс для проекта из первой или второй работы. Язык и шаблон приложения выберите сами.

ВЫПОЛНЕНИЕ ЗАДАНИЯ

1. Спецификация классов.

Для реализации разных типов игроков использовался абстрактный класс **Player**. Данный класс имеет два публичных метода: **MakeADecision** и **ResetPlayerMemory**. Первый определяет действие игрока: довериться или обмануть. Игроки разных классов принимают решения по-разному. Второй метод нужен для сброса состояния игрока до начального.

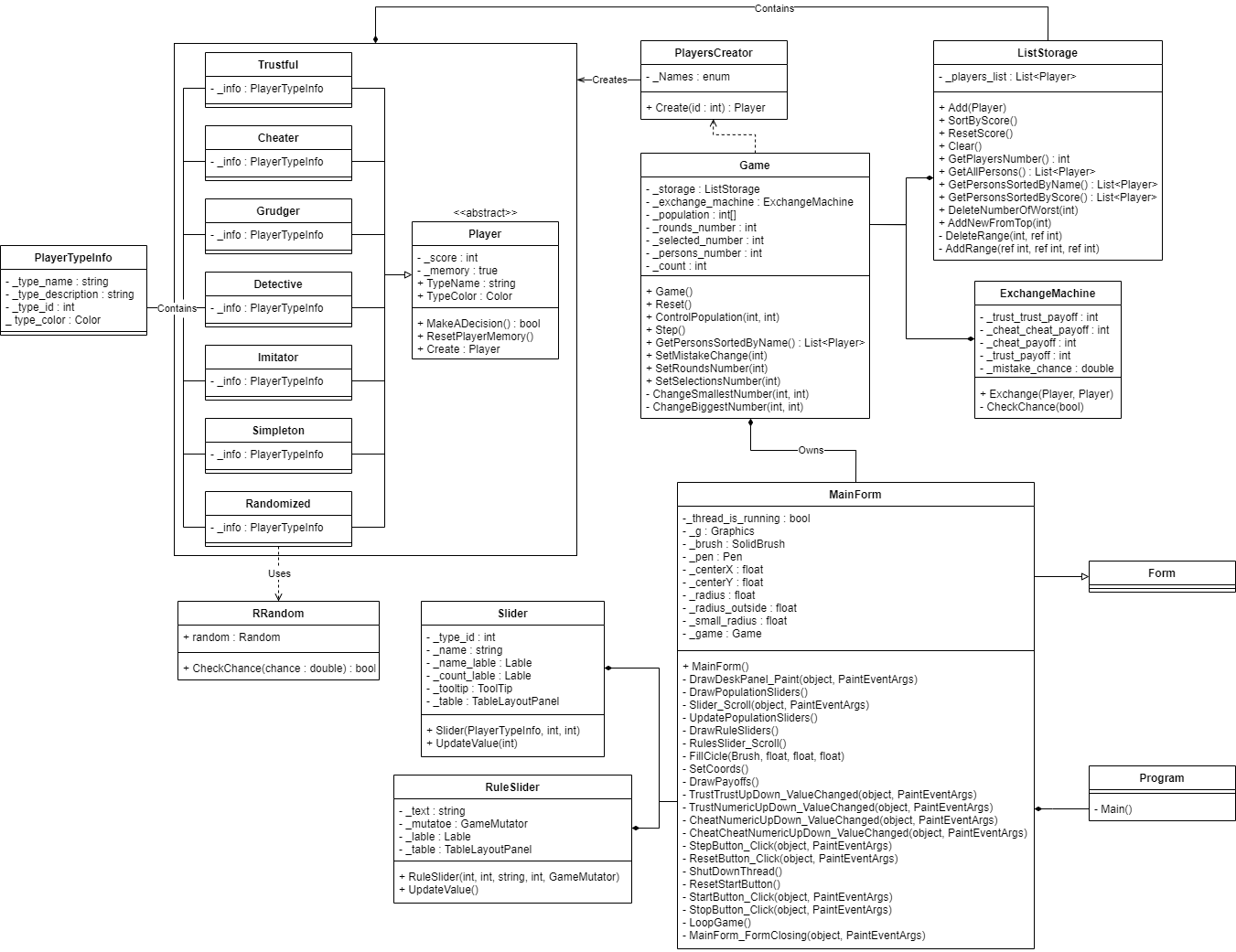
Для реализации обмена используется класс **ExchangeMachine**. В данном классе содержится информация об выплатах и вероятности ошибки игрока при принятии решения.

В классе **Game** реализована логика взаимодействия игроков.

Классы **Slider** и **RuleSlider** предназначены для облегчения связывания элементов интерфейса, объединяя в себе несколько объектов.

Класс **MainForm** описывает логику взаимодействия с пользователем и отображает текущее состояние игры.

1. Диаграмма классов.



1. Листинг

**Player.cs**

abstract class Player

{

/// <summary>

/// Игрок принимает решение довериться или обмануть другого игрока.

/// Разные типы игроков принимают решение по-разному.

/// </summary>

public abstract bool MakeADecision();

/// <summary>

/// Вернуть игрока в начальное состояние.

/// </summary>

public abstract void ResetPlayerMemory();

/// <summary>

/// Создать игрока своего типа.

/// </summary>

/// <returns></returns>

public abstract Player Create();

/// <summary>

/// Счёт игрока

/// </summary>

private int \_score = 0;

public int Score

{

get { return \_score; }

set { \_score = value; }

}

/// <summary>

/// Предыдущий ход противника.

/// Многие типы игроков принимают решения

/// операясь на этот поле.

/// </summary>

private bool \_memory = true;

public bool Memory

{

get { return \_memory; }

set { \_memory = value; }

}

public abstract string TypeName { get; }

public abstract Color TypeColor { get; }

}

**Trustful.cs**

class Trustful : Player

{

static Trustful()

{

\_info = new PlayerTypeInfo(

0,

"Trustful",

"\"Давай будем лучшими друзьями!\"\nВсегда доверяется.",

Color.Pink

);

}

public override bool MakeADecision()

{

return true;

}

public override void ResetPlayerMemory() { }

public override Player Create()

{

return new Trustful();

}

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Cheater.cs**

class Cheater : Player

{

static Cheater()

{

\_info = new PlayerTypeInfo(

1,

"Cheater",

"\"Cильные должны есть слабых!\"\nВсегда обманывает.",

Color.Black

);

}

public override bool MakeADecision()

{

return false;

}

public override void ResetPlayerMemory() { }

public override Player Create()

{

return new Cheater();

}

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Copycat.cs**

class Copycat : Player

{

static Copycat()

{

\_info = new PlayerTypeInfo(

2,

"Copycat",

"\"Привет! На первом ходу я доверюсь,\nа потом просто буду копировать твой последний ход.\"",

Color.Blue

);

}

public override bool MakeADecision()

{

return Memory;

}

public override void ResetPlayerMemory()

{

Memory = true;

}

public override Player Create()

{

return new Copycat();

}

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Grudger.cs**

class Grudger : Player

{

static Grudger()

{

\_info = new PlayerTypeInfo(

3,

"Grudger",

"\"Я начну с доверия и буду продолжать доверять,\nно если ты хоть раз меня обманешь,\nЯ БУДУ ЖУЛЬНИЧАТЬ ДО ПОСЛЕДНЕГО.\"",

Color.Yellow

);

}

public override bool MakeADecision()

{

if (!Memory)

{

\_was\_deceived = true;

}

return !\_was\_deceived;

}

public override void ResetPlayerMemory()

{

Memory = true;

\_was\_deceived = false;

}

public override Player Create()

{

return new Grudger();

}

private bool \_was\_deceived = false;

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Detective.cs**

class Detective : Player

{

static Detective()

{

\_info = new PlayerTypeInfo(

4,

"Detective",

"\"Учти: я тебя анализирую.\nМои первые ходы: доверие, обман, доверие, доверие.\nЕсли обманешь, я буду действовать как Имитатор.\nЕсли ни разу не обманешь, я буду всегда жульничать,\nчтобы использовать тебя. Элементарно, мой дорогой Ватсон.\"",

Color.Brown

);

}

public override bool MakeADecision()

{

if (\_count < \_first\_moves.Length)

{

if (!Memory)

{

\_was\_deceived = true;

}

bool ret = \_first\_moves[\_count];

\_count++;

return ret;

}

if (\_was\_deceived)

{

return Memory;

}

return false;

}

public override void ResetPlayerMemory()

{

Memory = true;

\_count = 0;

\_was\_deceived = false;

}

public override Player Create()

{

return new Detective();

}

private static readonly bool[] \_first\_moves = { true, false, true, true };

private int \_count = 0;

private bool \_was\_deceived = false;

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Imitator.cs**

class Imitator : Player

{

static Imitator()

{

\_info = new PlayerTypeInfo(

5,

"Imitator",

"\"Привет!\nЯ почти как Имитатор, но жульничаю только если обмануть меня два раза подряд.\nВ конце концов, первый раз мог быть ошибкой!\"",

Color.Purple

);

}

public override bool MakeADecision()

{

if (!Memory) \_count++;

else \_count = 0;

if (\_count == 2)

{

\_was\_deceived\_twice = true;

}

if (\_was\_deceived\_twice)

return Memory;

return true;

}

public override void ResetPlayerMemory()

{

Memory = true;

\_was\_deceived\_twice = false;

\_count = 0;

}

public override Player Create()

{

return new Imitator();

}

private bool \_was\_deceived\_twice = false;

private int \_count = 0;

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Simpleton.cs**

class Simpleton : Player

{

static Simpleton()

{

\_info = new PlayerTypeInfo(

6,

"Simpleton",

"\"превет\nсначала я доверять тебе.\nесли ты тоже доверять мене, я повторять свой ход, даже если это ошибка.\nесли ты обмануть меня, я делать свой ход наоборот, даже если это ошибка.\"",

Color.Green

);

}

public override bool MakeADecision()

{

if (!Memory)

{

\_last\_move = !\_last\_move;

}

return \_last\_move;

}

public override void ResetPlayerMemory()

{

Memory = true;

\_last\_move = true;

}

public override Player Create()

{

return new Simpleton();

}

private bool \_last\_move = true;

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**Randomized.cs**

static Randomized()

{

\_info = new PlayerTypeInfo(

7,

"Randomized",

"Просто жульничает или сотрудничает случайным образом\nс вероятностью 50/50",

Color.Red

);

}

public override bool MakeADecision()

{

return RRandom.CheckChance(\_chance);

}

public override void ResetPlayerMemory() { }

public override Player Create()

{

return new Randomized();

}

private static double \_chance = 0.5;

private static PlayerTypeInfo \_info;

public static PlayerTypeInfo Info { get { return \_info; } }

public override string TypeName { get { return \_info.TypeName; } }

public override Color TypeColor { get { return \_info.TypeColor; } }

}

**PlayerTypeInfo.cs**

class PlayerTypeInfo

{

public PlayerTypeInfo(int id, string name, string description, Color color)

{

\_type\_color = color;

\_type\_id = id;

\_type\_name = name;

\_type\_description = description;

}

private string \_type\_name;

public string TypeName { get { return \_type\_name; } }

private string \_type\_description;

public string TypeDescription { get { return \_type\_description; } }

private int \_type\_id;

public int TypeId { get { return \_type\_id; } }

private Color \_type\_color;

public Color TypeColor { get { return \_type\_color; } }

}

**ExchangeMachine.cs**

class ExchangeMachine

{

/// <summary>

/// Обмен между игроками.

/// </summary>

/// <param name="pers1">Игрок 1.</param>

/// <param name="pers2">Игрок 2.</param>

public void Exchange(Player pers1, Player pers2)

{

bool des1 = CheckChance(pers1.MakeADecision());

bool des2 = CheckChance(pers2.MakeADecision());

int payoff1;

int payoff2;

if (des1 == des2)

{

if (des1)

{

payoff1 = \_trust\_trust\_payoff;

payoff2 = \_trust\_trust\_payoff;

}

else

{

payoff1 = \_cheat\_cheat\_payoff;

payoff2 = \_cheat\_cheat\_payoff;

}

}

else

{

if (des1)

{

payoff1 = \_trust\_payoff;

payoff2 = \_cheat\_payoff;

}

else

{

payoff1 = \_cheat\_payoff;

payoff2 = \_trust\_payoff;

}

}

pers1.Score += payoff1;

pers1.Memory = des2;

pers2.Score += payoff2;

pers2.Memory = des1;

}

/// <summary>

/// Проверка шанса ошибки.

/// Если ошибка произошла, то возвращает значение обратное decision.

/// </summary>

/// <param name="decision">Решение игрока.</param>

/// <returns></returns>

private bool CheckChance(bool decision)

{

return RRandom.CheckChance(\_mistake\_chance) ? !decision : decision;

}

/// <summary>

/// Выплата при одинаковом решении довериться.

/// </summary>

private int \_trust\_trust\_payoff = 2;

public int TrustTrustPayoff

{

get { return \_trust\_trust\_payoff; }

set { \_trust\_trust\_payoff = value; }

}

/// <summary>

/// Выплаты при одинаковом решении обмануть.

/// </summary>

private int \_cheat\_cheat\_payoff = 0;

public int CheatCheatPayoff

{

get { return \_cheat\_cheat\_payoff; }

set { \_cheat\_cheat\_payoff = value; }

}

/// <summary>

/// Выплата за обман.

/// </summary>

private int \_cheat\_payoff = 3;

public int CheatPayoff

{

get { return \_cheat\_payoff; }

set { \_cheat\_payoff = value; }

}

/// <summary>

/// Выплата за доверие.

/// </summary>

private int \_trust\_payoff = -1;

public int TrustPayoff

{

get { return \_trust\_payoff; }

set { \_trust\_payoff = value; }

}

/// <summary>

/// Шанс ошибки.

/// </summary>

private double \_mistake\_chance = 0.05;

public double MistakeChance

{

get { return \_mistake\_chance; }

set { \_mistake\_chance = value; }

}

}

**Game.cs**

class Game

{

public Game()

{

Reset();

}

/// <summary>

/// Вернуть количество игроков начальное состояние.

/// </summary>

public void Reset()

{

\_storage.Clear();

\_count = 0;

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

{

for (int j = 0; j < \_population[i]; j++)

{

\_storage.Add(PlayersCreator.Create(i));

}

}

\_persons\_number = \_storage.GetPlayersNumber();

}

/// <summary>

/// Именить

/// </summary>

/// <param name="value"></param>

/// <param name="id"></param>

public void ControlPopulation(int value, int id)

{

if(value == \_population[id])

{

return;

}

int dif = \_population[id] - value;

\_population[id] = value;

if(dif < 0)

{

ChangeBiggestNumber(id, dif);

return;

}

ChangeSmallestNumber(id, dif);

}

private void ChangeSmallestNumber(int index, int dif)

{

for(; dif!=0 ; dif--)

{

int smallest = \_persons\_number;

int index\_smallest = 0;

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

{

if (\_population[i] == 0)

{

continue;

}

if (i == index)

{

continue;

}

if (smallest > \_population[i])

{

index\_smallest = i;

smallest = \_population[i];

}

}

\_population[index\_smallest]++;

}

}

private void ChangeBiggestNumber(int index, int dif)

{

for (; dif != 0; dif++)

{

int biggest = 0;

int index\_biggest = 0;

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

{

if (\_population[i] == 0)

{

continue;

}

if (i == index)

{

continue;

}

if (biggest < \_population[i])

{

index\_biggest = i;

biggest = \_population[i];

}

}

\_population[index\_biggest]--;

}

}

public void Step()

{

if (\_count == \_persons\_number)

{

\_storage.SortByScore();

\_storage.DeleteNumberOfWorst(\_selected\_number);

\_count = -1;

\_persons\_number = \_storage.GetPlayersNumber();

return;

}

if (\_count == -1)

{

\_storage.AddNewFromTop(\_selected\_number);

\_storage.ResetScore();

\_persons\_number = \_storage.GetPlayersNumber();

\_count = 0;

return;

}

for (; \_count < \_persons\_number; \_count++)

{

for (int i = \_count + 1; i < \_persons\_number; i++)

{

PlayRounds(\_storage[\_count], \_storage[i]);

}

}

}

private void PlayRounds(Player pers1, Player pers2)

{

for (int i = 0; i < \_rounds\_number; i++)

{

\_exchange\_machine.Exchange(pers1, pers2);

}

pers1.ResetPlayerMemory();

pers2.ResetPlayerMemory();

}

/// <summary>

/// Получить список всех игроков отсортированный по именам.

/// </summary>

/// <returns>Список игроков.</returns>

public List<Player> GetPersonsSortedByName()

{

return \_storage.GetPersonsSortedByName();

}

/// <summary>

/// Установить шанс ошибки игрока.

/// </summary>

/// <param name="value">Процент вероятности.</param>

public void SetMistakeChange(int value)

{

\_exchange\_machine.MistakeChance = (double)value / 100.0;

}

/// <summary>

/// Хранилище игроков.

/// </summary>

private ListStorage \_storage = new ListStorage();

private ExchangeMachine \_exchange\_machine = new ExchangeMachine();

public ExchangeMachine ExchangeMachine { get { return \_exchange\_machine; } }

/// <summary>

/// Массив количества типов игроков.

/// </summary>

private int[] \_population = { 4, 4, 4, 4, 4, 4, 3, 3 };

public int[] Population { get { return \_population; } }

/// <summary>

/// Количество раундов.

/// </summary>

private int \_rounds\_number = 10;

/// <summary>

/// Установить количество раундов.

/// </summary>

/// <param name="value">Количество раундов.</param>

public void SetRoundsNumber(int value)

{

\_rounds\_number = value;

}

public int RoundsNumber

{

get { return \_rounds\_number; }

set { \_rounds\_number = value; }

}

/// <summary>

/// Количество отбираемых игроков.

/// </summary>

private int \_selected\_number = 5;

/// <summary>

/// Уставновить количество отбираемых игроков.

/// </summary>

/// <param name="value">Количество отбираемых игроков.</param>

public void SetSelectionsNumber(int value)

{

\_selected\_number = value;

}

public int SelectionsNumber

{

get { return \_selected\_number; }

set { \_selected\_number = value; }

}

/// <summary>

/// Количество игроков.

/// </summary>

private int \_persons\_number;

public int PopulationNumber { get { return \_persons\_number; } }

/// <summary>

/// Состояние игры.

/// </summary>

private int \_count = 0;

}

**ListStorage.cs**

class ListStorage

{

/// <summary>

/// Добавление игрока в хранилище.

/// </summary>

/// <param name="pers">Добавляемый игрок.</param>

public void Add(Player pers)

{

\_players\_list.Add(pers);

}

/// <summary>

/// Отсортировать хранилище по счёту.

/// </summary>

public void SortByScore()

{

\_players\_list.Sort((x, y) => x.Score.CompareTo(y.Score));

}

/// <summary>

/// Обнуление счёта игроков.

/// </summary>

public void ResetScore()

{

foreach (var item in \_players\_list)

{

item.Score = 0;

}

}

/// <summary>

/// Очищение хранилища.

/// </summary>

public void Clear()

{

\_players\_list.Clear();

}

/// <summary>

/// Получить количество игроков.

/// </summary>

/// <returns>Количество игроков.</returns>

public int GetPlayersNumber()

{

return \_players\_list.Count();

}

/// <summary>

/// Получить список всех игроков.

/// </summary>

/// <returns>Список игроков.</returns>

public List<Player> GetAllPersons()

{

return new List<Player>(\_players\_list);

}

/// <summary>

/// Получить список всех игроков отсортированный по именам.

/// </summary>

/// <returns>Список игроков.</returns>

public List<Player> GetPersonsSortedByName()

{

return \_players\_list.OrderBy(u => u.TypeName).ToList();

}

/// <summary>

/// Получить список всех игроков отсортированный по счёту.

/// </summary>

/// <returns>Список игроков.</returns>

public List<Player> GetPersonsSortedByScore()

{

return \_players\_list.OrderBy(u => u.Score).ToList();

}

/// <summary>

/// Получение игркоа по индексу.

/// </summary>

/// <param name="index">Индекс.</param>

/// <returns>Игрок.</returns>

public Player this[int index]

{

get { return \_players\_list[index]; }

}

/// <summary>

/// Удаление указанного количества игроков с наименьшим счётом.

/// Если у игроков одинаковые счета, то удаляются случайные из них.

/// </summary>

/// <param name="number">Количество удаляемых игроков.</param>

public void DeleteNumberOfWorst(int number)

{

if(\_players\_list.Count < number)

{

Clear();

return;

}

int lowest\_score = \_players\_list[0].Score;

int i = 0;

for (; number > 0 && i < \_players\_list.Count; i++)

{

if (lowest\_score == \_players\_list[i].Score)

{

continue;

}

DeleteRange(i, ref number);

i = 0;

lowest\_score = \_players\_list[i].Score;

}

if (number > 0)

{

DeleteRange(i, ref number);

}

}

/// <summary>

/// Удаление игроков из указанного промежутка.

/// Если промежуток больше числа удаляемых игроков, то выбираются случайные игроки из этого промежутка.

/// Начало промежутка всегда 0.

/// Игроки удаляются из промежутка [0, end).

/// </summary>

/// <param name="end">Конец промежутка.</param>

/// <param name="number">Количество удаляемых игроков.</param>

private void DeleteRange(int end, ref int number)

{

if (end <= number)

{

\_players\_list.RemoveRange(0, end);

number -= end;

end = 0;

return;

}

while (number != 0)

{

int index = RRandom.random.Next(end);

\_players\_list.RemoveAt(index);

number--;

end--;

}

}

/// <summary>

/// Добавление указанного количества игроков.

/// Новые игроки создаются из игроков с лучшим счётом.

/// Если у игроков одинаковые счета, то создаются по случайным из них.

/// </summary>

/// <param name="number">Количество удаляемых игроков.</param>

public void AddNewFromTop(int number)

{

if(\_players\_list.Count == 0)

{

return;

}

if (\_players\_list.Count < number)

{

number = \_players\_list.Count;

}

int best\_score = \_players\_list[\_players\_list.Count - 1].Score;

int i = \_players\_list.Count - 2;

int start = i + 1;

for (; number > 0 && i >= 0; i--)

{

if (best\_score == \_players\_list[i].Score)

{

continue;

}

AddRange(ref start, ref i, ref number);

best\_score = \_players\_list[i].Score;

start = i;

}

AddRange(ref start, ref i, ref number);

}

/// <summary>

/// Добавление игроков из указанного промежутка.

/// Если промежуток больше числа добавляемых игроков, то выбираются случайные игроки из этого промежутка.

/// Игроки добавляются из промежутка [start, end].

/// </summary>

/// <param name="start">Начало промежутка.</param>

/// <param name="end">Конец промежутка.</param>

/// <param name="number">Количество удаляемых игроков.</param>

private void AddRange(ref int start, ref int end, ref int number)

{

if (start - end <= number)

{

for (int j = end + 1; j <= start; j++)

{

Add(\_players\_list[j].Create());

}

number -= start - end;

return;

}

while (number != 0)

{

int index = RRandom.random.Next(end + 1, start + 1);

Add(\_players\_list[index]);

Add(\_players\_list[index].Create());

\_players\_list.RemoveAt(index);

start--;

number--;

}

}

/// <summary>

/// Хранилище игроков.

/// </summary>

private List<Player> \_players\_list = new List<Player>();

}

**PlayersCreator.cs**

class PlayersCreator

{

public static Player Create(int id)

{

switch (id)

{

case (int)\_Names.Trustful:

return new Trustful();

case (int)\_Names.Cheater:

return new Cheater();

case (int)\_Names.Copycat:

return new Copycat();

case (int)\_Names.Grudger:

return new Grudger();

case (int)\_Names.Detective:

return new Detective();

case (int)\_Names.Imitator:

return new Imitator();

case (int)\_Names.Simpleton:

return new Simpleton();

case (int)\_Names.Randomized:

return new Randomized();

default:

throw new ArgumentException();

}

}

private enum \_Names

{

Trustful,

Cheater,

Copycat,

Grudger,

Detective,

Imitator,

Simpleton,

Randomized

}

}

**RRandom.cs**

static class RRandom

{

public static bool CheckChance(double chance)

{

return chance >= random.NextDouble(); ;

}

public static Random random = new Random();

}

**Slider.cs**

class Slider : TrackBar

{

public Slider(PlayerTypeInfo info, int value, int max)

{

\_name = info.TypeName;

\_type\_id = info.TypeId;

\_table.ColumnCount = 1;

\_table.RowCount = 3;

\_table.RowStyles.Add(new RowStyle());

\_table.RowStyles.Add(new RowStyle());

\_table.RowStyles.Add(new RowStyle(SizeType.Percent, 100F));

\_name\_lable.Text = \_name;

\_name\_lable.ForeColor = info.TypeColor;

\_name\_lable.Font = new Font("Arial", 12, FontStyle.Bold);

\_name\_lable.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

\_name\_lable.Dock = DockStyle.Fill;

\_table.Controls.Add(\_name\_lable, 0, 0);

this.Minimum = 0;

this.Maximum = max;

this.Value = value;

this.LargeChange = 1;

this.Dock = DockStyle.Fill;

\_table.Controls.Add(this, 0, 2);

\_count\_lable.Text = this.Value.ToString();

\_count\_lable.Dock = DockStyle.Fill;

\_count\_lable.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

\_table.Controls.Add(\_count\_lable, 0, 1);

\_toolTip.AutoPopDelay = 5000;

\_toolTip.InitialDelay = 100;

\_toolTip.ReshowDelay = 500;

\_toolTip.ShowAlways = true;

\_toolTip.ToolTipTitle = info.TypeName;

\_toolTip.UseFading = true;

\_toolTip.SetToolTip(\_name\_lable, info.TypeDescription);

}

public void UpdateValue(int value)

{

this.Value = value;

\_count\_lable.Text = this.Value.ToString();

}

private int \_type\_id;

public int TypeId { get { return \_type\_id; } }

private string \_name;

private Label \_name\_lable = new Label();

private Label \_count\_lable = new Label();

private ToolTip \_toolTip = new ToolTip();

public Label Counter

{

get { return \_count\_lable; }

}

private TableLayoutPanel \_table = new TableLayoutPanel();

public TableLayoutPanel Table

{

get { return \_table; }

}

}

**RuleSlider.cs**

class RuleSlider : TrackBar

{

public RuleSlider(int min, int max, string text, int value, GameMutator mutator)

{

\_table.ColumnCount = 1;

\_table.RowCount = 2;

\_table.RowStyles.Add(new RowStyle(SizeType.Percent, 50F));

\_table.RowStyles.Add(new RowStyle(SizeType.Percent, 50F));

\_table.Dock = DockStyle.Fill;

\_text = text;

\_mutator = mutator;

\_lable.Text = \_text.Replace("{}", value.ToString());

\_lable.Font = new Font("Microsoft Sans Serif", 11);

\_lable.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;

\_lable.Dock = DockStyle.Fill;

\_table.Controls.Add(\_lable, 0, 0);

this.Minimum = min;

this.Maximum = max;

this.Value = value;

this.Margin = new Padding(10);

this.LargeChange = 1;

this.Dock = DockStyle.Fill;

\_table.Controls.Add(this, 0, 1);

}

public void UpdateValue()

{

int value = this.Value;

\_lable.Text = \_text.Replace("{}", value.ToString());

\_mutator(this.Value);

}

private string \_text;

internal delegate void GameMutator(int value);

private GameMutator \_mutator;

public GameMutator Mutator

{

get { return \_mutator; }

set { \_mutator = value; }

}

private Label \_lable = new Label();

public Label Label

{

get { return \_lable; }

}

private TableLayoutPanel \_table = new TableLayoutPanel();

public TableLayoutPanel Table

{

get { return \_table; }

}

}

**MainForm.cs**

public partial class MainForm : Form

{

private List<Slider> \_sliders = new List<Slider>();

private bool \_thread\_is\_running = false;

private Graphics \_g;

private SolidBrush \_brush = new SolidBrush(Color.Black);

private Pen \_pen = new Pen(Color.Red, 1F);

private float \_centerX;

private float \_centerY;

private float \_radius;

private float \_radius\_outside;

private float \_small\_radius = 15F;

private Game \_game = new Game();

public MainForm()

{

InitializeComponent();

\_g = DrawDeskPanel.CreateGraphics();

DrawPopulationSliders();

DrawRuleSliders();

DrawPayoffs();

SetCoords();

}

private void DrawDeskPanel\_Paint(object sender, PaintEventArgs e)

{

StringFormat format = new StringFormat();

Font font = new Font("Arial", 11);

if(\_game.PopulationNumber == 0)

{

return;

}

if(\_game.PopulationNumber == 1)

{

Player player = \_game.GetPersonsSortedByName()[0];

\_brush.Color = player.TypeColor;

FillCicle(\_brush, \_centerX, \_centerY, \_small\_radius);

\_brush.Color = Color.Black;

\_g.DrawString(player.Score.ToString(), font, \_brush, \_centerX + \_small\_radius, \_centerY, format);

return;

}

double angle = 360 / \_game.PopulationNumber;

double current\_angle = 0;

foreach(var player in \_game.GetPersonsSortedByName())

{

double radians = current\_angle \* Math.PI / 180;

float x = \_centerX + \_radius \* (float)Math.Cos(radians);

float y = \_centerY + \_radius \* (float)Math.Sin(radians);

\_brush.Color = player.TypeColor;

FillCicle(\_brush, x, y, \_small\_radius);

x = \_centerX + \_radius\_outside \* (float)Math.Cos(radians);

y = \_centerY + \_radius\_outside \* (float)Math.Sin(radians);

\_brush.Color = Color.Black;

\_g.DrawString(player.Score.ToString(), font, \_brush, x, y, format);

current\_angle += angle;

}

}

private void DrawPopulationSliders()

{

PlayerTypeInfo[] PlayerTypeInfos = {

Trustful.Info,

Cheater.Info,

Copycat.Info,

Grudger.Info,

Detective.Info,

Imitator.Info,

Simpleton.Info,

Randomized.Info

};

int i = 0;

for (int y = 0; y < PopulatioTableLayoutPanel.ColumnCount; y++)

{

for (int x = 0; x < PopulatioTableLayoutPanel.RowCount; x++)

{

Slider slider = new Slider(PlayerTypeInfos[i], \_game.Population[i], \_game.PopulationNumber);

\_sliders.Add(slider);

slider.Scroll += Slider\_Scroll;

PopulatioTableLayoutPanel.Controls.Add(slider.Table, x, y);

i++;

}

}

}

private void Slider\_Scroll(object sender, System.EventArgs e)

{

ShutDownThread();

Slider slider = sender as Slider;

slider.Counter.Text = (sender as Slider).Value.ToString();

\_game.ControlPopulation(slider.Value, slider.TypeId);

\_game.Reset();

UpdatePopulationSliders();

DrawDeskPanel.Refresh();

}

private void UpdatePopulationSliders()

{

foreach (var slider in \_sliders)

{

slider.UpdateValue(\_game.Population[slider.TypeId]);

}

}

private void DrawRuleSliders()

{

RuleSlider[] sliders =

{

new RuleSlider(1, 50, "Играть {} раундов за матч", \_game.RoundsNumber, \_game.SetRoundsNumber),

new RuleSlider(1, 15, "После каждого турнира удалять {} худших игроков и копировать n лучших", \_game.SelectionsNumber, \_game.SetSelectionsNumber),

new RuleSlider(0, 50, "В каждом раунде игрок делает ошибку с вероятностью {}%", (int)(\_game.ExchangeMachine.MistakeChance\*100), \_game.SetMistakeChange),

};

for(int i = 0; i < RulesTableLayoutPanel.RowCount; i++)

{

sliders[i].Scroll += RulesSlider\_Scroll;

RulesTableLayoutPanel.Controls.Add(sliders[i].Table, 0, i);

}

}

private void RulesSlider\_Scroll(object sender, System.EventArgs e)

{

(sender as RuleSlider).UpdateValue();

}

private void FillCicle(Brush brush, float centerX, float centerY, float radius)

{

\_g.FillEllipse(brush, centerX - radius, centerY - radius, radius \* 2, radius \* 2);

}

private void SetCoords()

{

\_centerX = DrawDeskPanel.Width / 2;

\_centerY = DrawDeskPanel.Height / 2;

\_radius = DrawDeskPanel.Width / 2 - 80;

\_radius\_outside = \_radius + \_small\_radius;

}

private void DrawPayoffs()

{

TrustNumericUpDown.Value = (decimal)\_game.ExchangeMachine.TrustPayoff;

TrustTrustUpDown.Value = (decimal)\_game.ExchangeMachine.TrustTrustPayoff;

CheatCheatNumericUpDown.Value = (decimal)\_game.ExchangeMachine.CheatCheatPayoff;

CheatNumericUpDown.Value = (decimal)\_game.ExchangeMachine.CheatPayoff;

}

private void TrustTrustUpDown\_ValueChanged(object sender, EventArgs e)

{

NumericUpDown numeric = sender as NumericUpDown;

if(numeric.Value > numeric.Maximum)

{

numeric.Value = numeric.Maximum;

}

if (numeric.Value < numeric.Minimum)

{

numeric.Value = numeric.Minimum;

}

\_game.ExchangeMachine.TrustTrustPayoff = (int)numeric.Value;

}

private void TrustNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

NumericUpDown numeric = sender as NumericUpDown;

if (numeric.Value > numeric.Maximum)

{

numeric.Value = numeric.Maximum;

}

if (numeric.Value < numeric.Minimum)

{

numeric.Value = numeric.Minimum;

}

\_game.ExchangeMachine.TrustPayoff = (int)numeric.Value;

}

private void CheatNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

NumericUpDown numeric = sender as NumericUpDown;

if (numeric.Value > numeric.Maximum)

{

numeric.Value = numeric.Maximum;

}

if (numeric.Value < numeric.Minimum)

{

numeric.Value = numeric.Minimum;

}

\_game.ExchangeMachine.CheatPayoff = (int)numeric.Value;

}

private void CheatCheatNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

NumericUpDown numeric = sender as NumericUpDown;

if (numeric.Value > numeric.Maximum)

{

numeric.Value = numeric.Maximum;

}

if (numeric.Value < numeric.Minimum)

{

numeric.Value = numeric.Minimum;

}

\_game.ExchangeMachine.CheatCheatPayoff = (int)numeric.Value;

}

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

{

ShutDownThread();

\_game.Step();

DrawDeskPanel.Refresh();

}

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

{

ShutDownThread();

\_game.Reset();

DrawDeskPanel.Refresh();

}

private void ShutDownThread()

{

if (!\_thread\_is\_running)

return;

\_thread\_is\_running = false;

ResetStartButton();

}

private void ResetStartButton()

{

StartButton.Text = "Start";

StartButton.Click += StartButton\_Click;

StartButton.Click -= StopButton\_Click;

}

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

{

Thread loop\_game = new Thread(LoopGame);

\_thread\_is\_running = true;

loop\_game.Start();

StartButton.Text = "Stop";

StartButton.Click -= StartButton\_Click;

StartButton.Click += StopButton\_Click;

}

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

{

ShutDownThread();

}

private void LoopGame()

{

while(\_thread\_is\_running)

{

\_game.Step();

DrawDeskPanel.Refresh();

Thread.Sleep(200);

}

}

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

{

ShutDownThread();

}

}