Государственный Университет Молдовы  
Факультет Математики и Информатики  
Департамент Информатики

**Лабораторная работа №1**

По курсу “Теория вероятности”

Выполнил студент группы I1902:  
Чобану Артём

Кишинэу, 2020

**Задание 1**

Посчитать вероятность того, что кони побьют друг друга через n ходов.

**Код программы:**

(так же можно получить по ссылке: https://github.com/ArtiomCiobanu/ProbabilityTheory\_Horses)

using System;  
using ProbabilityTheory\_Horses.API;  
using ProbabilityTheory\_Horses.API.Static\_Classes;  
  
namespace ProbabilityTheory\_Horses  
{  
 internal static class Program  
 {  
 private static double Probability =>

(double) SuccessfullExperiments / AllExperiments;  
 private static int SuccessfullExperiments { get; set; }  
 private static int AllExperiments { get; set; }  
  
 private static int LastMove { get; set; }  
  
 private static void Main()  
 {  
 HorseTools.Width = 8;  
 HorseTools.Height = 8;  
 LastMove = 3; //Количество ходов, за которые кони

//должны побить друг друга  
  
 SuccessfullExperiments = 0; //Удавшиеся эксперименты  
 AllExperiments = 1000000; //Все эксперименты  
  
 var firstHorse = new Horse();  
 var secondHorse = new Horse();  
  
 var random = new Random();  
  
 for (int experimentIndex = 1; experimentIndex <= AllExperiments;

experimentIndex++)  
 {  
 //Получаем случайную клетку  
 firstHorse.CurrentCell = CellTools.GetRandomCell();

//Получаем случайную клетку кроме ранее полученной  
 secondHorse.CurrentCell =

CellTools.GetRandomCellExcept(firstHorse.CurrentCell);  
   
 for (int i = 0; i <= LastMove; i++)  
 {  
 //Проверка того, побили ли кони друг друга  
 if (firstHorse.HorseCellOptions.Contains(secondHorse.CurrentCell))  
 {  
 SuccessfullExperiments++;  
 continue;  
 }  
  
 //Ходящий конь определяется чётной/нечётной итерацией  
 var movingHorse = i % 2 == 0 ? firstHorse : secondHorse;  
   
 //Сам ход выбранного коня  
 int nextMoveCellIndex =

random.Next(0, movingHorse.CellOptionsAmount);  
 movingHorse.CurrentCell =

movingHorse.HorseCellOptions[nextMoveCellIndex];  
 }  
 }  
  
 Console.WriteLine($"Количество успешных экспериментов: {SuccessfullExperiments} из {AllExperiments}");  
 Console.WriteLine($"Приблизительный шанс: {Probability}");  
 }  
 }  
}

Для лучшей реализации проекта я написал небольшой API.

Класс шахматной клетки:

using System;  
  
namespace ProbabilityTheory\_Horses.API  
{  
 public class Cell : IEquatable<Cell>  
 {  
 public int X { get; }  
 public int Y { get; }  
  
 public Cell(int x, int y)  
 {  
 X = x;  
 Y = y;  
 }  
  
 public bool Equals(Cell other)  
 {  
 if (ReferenceEquals(null, other)) return false;  
 if (ReferenceEquals(this, other)) return true;  
 return X == other.X && Y == other.Y;  
 }  
  
 public override string ToString()  
 {  
 return $"{(X, Y)}";  
 }  
 }  
}

Класс коня:

using System.Collections.Generic;  
using System.Linq;  
using ProbabilityTheory\_Horses.API.Static\_Classes;  
  
namespace ProbabilityTheory\_Horses.API  
{  
 public class Horse  
 {  
 private Cell \_cell;  
  
 public Cell CurrentCell  
 {  
 get => \_cell;  
 set  
 {  
 \_cell = value;  
 HorseCellOptions = HorseTools.GetOptions(\_cell).ToList();  
 }  
 }  
  
 public List<Cell> HorseCellOptions { get; private set; }  
  
 public int CellOptionsAmount => HorseCellOptions.Count;  
  
 public Horse()  
 {  
 \_cell = null;  
 }  
  
 public Horse(Cell currentCell)  
 {  
 CurrentCell = currentCell;  
 }  
 }  
}

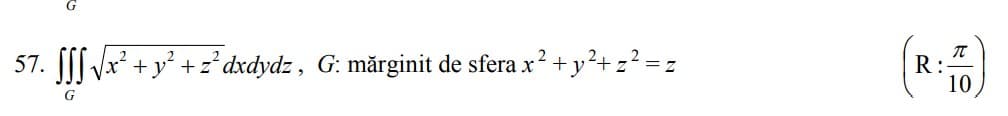
Статический класс с методами, помогающими в разных вычислениях:

using System.Collections.Generic;  
using System.Linq;  
using ProbabilityTheory\_Horses.API.Other;  
  
namespace ProbabilityTheory\_Horses.API.Static\_Classes  
{  
 public static class HorseTools  
 {  
 public static int Width { get; set; }  
 public static int Height { get; set; }  
  
 public static bool CellIsValid(Cell cell)  
 {  
 return cell.X >= 1 && cell.X <= Width && cell.Y >= 1 && cell.Y <= 10;  
 }  
 public static IEnumerable<Cell> GetAllHorseCells(Cell cell)  
 {  
 int x = cell.X;  
 int y = cell.Y;  
  
 return new List<Cell>  
 {  
 new Cell(x - 1, y - 2),  
 new Cell(x + 1, y - 2),  
 new Cell(x + 2, y + 1),  
 new Cell(x + 2, y - 1),  
 new Cell(x - 2, y - 1),  
 new Cell(x - 2, y + 1),  
 new Cell(x - 1, y + 2),  
 new Cell(x + 1, y + 2),  
 };  
 }  
 public static IEnumerable<Cell> GetOptions(Cell horseCell) =>  
 CellTools.GetValidCells(GetAllHorseCells(horseCell));  
 public static bool HorsesHaveCommonCells(Horse first, Horse second)  
 {  
 return GetCommonHorseCells(first, second).Any();  
 }  
 public static IEnumerable<Cell> GetCommonHorseCells(Horse first, Horse second)  
 {  
 return first.HorseCellOptions.Intersect(second.HorseCellOptions, new CellEqualityComparer());  
 }  
 }  
}

Подобный класс для операций с клетками:

using System;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace ProbabilityTheory\_Horses.API.Static\_Classes  
{  
 public static class CellTools  
 {  
 private static Random Random { get; } = new Random();  
  
 public static Cell GetRandomCell()  
 {  
 return new Cell(Random.Next(1, HorseTools.Width + 1), Random.Next(1, HorseTools.Height + 1));  
 }  
  
 public static Cell GetRandomCellExcept(Cell cell)  
 {  
 Cell result;  
 do  
 {  
 result = GetRandomCell();  
 } while (Equals(result, cell));  
  
 return result;  
 }  
  
 public static IEnumerable<Cell> GetValidCells(IEnumerable<Cell> cells)  
 {  
 return cells.Where(HorseTools.CellIsValid);  
 }  
 }  
}

**Задание 2**



Код программы:

using System;  
namespace MonteCarloMethod  
{  
 internal static class Program  
 {  
 private static Random R { get; } = new Random();  
 private static double Square(double value)  
 {  
 return Math.Pow(value, 2);  
 }  
 private static double GetValue(double x, double y, double z) =>

Square(x) + Square(y) + Square(z);  
 private static void Main()  
 {  
 int experimentAmount = 1000;  
 double sum = 0;  
  
 for (int i = 1; i <= experimentAmount; i++)  
 {  
 Console.WriteLine($"Эксперимент No.{i}");  
 double x;  
 double y;  
 double z;

y = R.NextDouble();  
 z = R.NextDouble();

x = Math.Sqrt(z + Square(y) + Square(z));  
 Console.WriteLine($"x={x}, y={y}, z={z}");  
 var value = GetValue(x, y, z);  
 sum += value;  
 }  
 Console.WriteLine(sum / experimentAmount);  
 }  
 }  
}