**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 5 з дисципліни

«Проектування алгоритмів»

„**Пошук в умовах протидії, ігри з повною інформацією**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*Спаських Микола Дмитрович ІТ-04 і*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc52291748)

[2 Завдання 4](#_Toc52291749)

[3 Виконання 6](#_Toc52291750)

[3.1 Програмна реалізація алгоритму 6](#_Toc52291752)

[3.1.1 Вихідний код 6](#_Toc52291753)

[3.1.2 Приклади роботи 6](#_Toc52291754)

[3.3 Тестування алгоритму 6](#_Toc52291755)

[Висновок 7](#_Toc52291756)

[Критерії оцінювання 8](#_Toc52291757)

# Мета лабораторної роботи

Мета роботи - вивчити основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Ознайомитися з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією.

# Завдання

Згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм альфа-бета-відсікань.

Реалізувати три рівні складності (легкий, середній, складний) + 1балл.

Зробити узагальнений висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти

|  |  |
| --- | --- |
| **№** | **Варіант** |
| 1 | Баше https://ru.wikipedia.org/wiki/Баше\_(игра) |
| 2 | Нейтріко http://www.iggamecenter.com/info/ru/neutreeko.html |
| 3 | Точки https://ru.wikipedia.org/wiki/Точки\_(игра) |
| 4 | Dots and Boxes https://ru.wikipedia.org/wiki/Палочки\_(игра) |
| 5 | Алемунгула http://www.iggamecenter.com/info/ru/alemungula.html |
| 6 | Snakes http://www.papg.com/show?3AE4 |
| 7 | Cram https://en.wikipedia.org/wiki/Cram\_(game) |
| 8 | Тіко http://www.iggamecenter.com/info/ru/teeko.html |
| 9 | Obstruction 8х8 http://www.papg.com/show?2XMX |
| 10 | Gale http://www.papg.com/show?1TPI |
| 11 | Гомоку https://ru.wikipedia.org/wiki/Гомоку |
| 12 | Нім https://ru.wikipedia.org/wiki/Ним\_(игра) |
| 13 | Клоббер http://www.iggamecenter.com/info/ru/clobber.html |
| 14 | Hackenbush http://www.papg.com/show?1TMP |
| 15 | Лінкідж (більше) http://www.iggamecenter.com/info/ru/linkage.html |
| 16 | Заєць и Вовки (за Зайця) http://www.iggamecenter.com/info/ru/foxh.html |
| 17 | 3D Noughts and Crosses 4 x 4 x 4 http://www.papg.com/show?1TND |
| 18 | Domineering 8х8 http://www.papg.com/show?1TX6 |
| 19 | Транспозиція http://www.iggamecenter.com/info/ru/transposition.html |
| 20 | Заєць и Вовки (за Вовків) http://www.iggamecenter.com/info/ru/foxh.html |
| 21 | Вари http://www.iggamecenter.com/info/ru/oware.html |
| 22 | Калах http://www.iggamecenter.com/info/ru/kalah.html |
| 23 | Реверсі https://ru.wikipedia.org/wiki/Реверси |
| 24 | Лінкідж (менше) http://www.iggamecenter.com/info/ru/linkage.html |
| 25 | Три мушкетери (за мушкетерів) http://www.iggamecenter.com/info/ru/threemusk.html |
| 26 | Сим https://ru.wikipedia.org/wiki/Сим\_(игра) |
| 27 | Col http://www.papg.com/show?2XLY |
| 28 | Snort http://www.papg.com/show?2XM1 |
| 29 | Chomp http://www.papg.com/show?3AEA |
| 30 | Три мушкетери (за гвардейців) http://www.iggamecenter.com/info/ru/threemusk.html |

В мене варіант 18

Domineering 8х8 http://www.papg.com/show?1TX6

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

using System;

using System.Collections.Generic;

namespace lab5

{

class state

{

public static int i = 0;

public static int n = 8;

public static int k = 1;

public int[,] mas;

public int hid;

public state(int hid1)

{

hid = hid1;

mas = new int[n, n];

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

{

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

{

mas[i, j] = 0;

}

}

i++;

}

public int[,] clone ()

{

int[,] rez = new int[n,n];

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

{

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

{

rez[i, j] = mas[i, j];

}

}

return rez;

}

public void vivod()

{

Console.Write(" ");

for (int i = 0; i < n; i++) Console.Write(i);

Console.WriteLine();

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

{

Console.Write(i+" ");

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

{

if (mas[i, j] == 0) Console.Write(mas[i, j]);

else

{

Console.Write("#");

}

//Console.BackgroundColor = ConsoleColor.Black;

}

Console.WriteLine();

}

}

public List<state> Successors()

{

List<state> rez = new List<state>();

if (hid == 1)

{

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

{

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

{

if (mas[i, j] == 0 && mas[i + 1, j] == 0)

{

state st\_add = new state(2);

st\_add.mas = clone();

st\_add.mas[i, j] = 2;

st\_add.mas[i + 1, j] = 4;

rez.Add(st\_add);

//if (state.i % 10000000 == 0) st\_add.vivod(); // test ------------------

}

}

}

}

else if (hid == 2)

{

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

{

for (int j = 0; j < n - 1; j++)

{

if (mas[i, j] == 0 && mas[i, j + 1] == 0)

{

state st\_add = new state(1);

st\_add.mas = clone();

st\_add.mas[i, j] = 1;

st\_add.mas[i, j + 1] = 3;

rez.Add(st\_add);

//if (state.i % 10000000 == 0) st\_add.vivod(); // test ------------------

}

}

}

}

else Console.WriteLine("Error hid!=1,2");

return rez;

}

public bool terminal\_test()

{

return Successors().Count == 0;

}

}

class tree

{

public static state top;

public static state current;

public static bool flag = true;

public tree()

{

top = new state(1);

current = top;

}

public static state mini\_max\_desidion(state current\_state)

{

int rez\_i = -1;

int max = 0;

state.i = 1;

List<state> list\_nashadkiv = current\_state.Successors();

if (list\_nashadkiv.Count == 0) { flag = false; Console.WriteLine("your win"); return null; }

else

{

for (int i = 0; i < list\_nashadkiv.Count; i++)

{

if (max < min\_value(list\_nashadkiv[i]))

{

max = min\_value(list\_nashadkiv[i]);

rez\_i = i;

}

if (state.i > 1000000\* state.k)

{

state.i = 1;

return otsinka(current\_state);

}

}

return list\_nashadkiv[rez\_i];

}

return null;

}

public static int max\_value(state st)

{

if (state.i > 1000000 \* state.k) return 0;

List<state> list\_nashadkiv = st.Successors();

if (list\_nashadkiv.Count==0) return st.hid;

int rez = 0;

for (int i=0; i< list\_nashadkiv.Count; i++)

{

if (rez < min\_value(list\_nashadkiv[i])) rez = min\_value(list\_nashadkiv[i]);

}

return rez;

}

public static int min\_value(state st)

{

if (state.i > 1000000 \* state.k) return 0;

List<state> list\_nashadkiv = st.Successors();

if (list\_nashadkiv.Count==0) return st.hid;

int rez = 3;

for (int i = 0; i < list\_nashadkiv.Count; i++)

{

if (rez > max\_value(list\_nashadkiv[i])) rez = max\_value(list\_nashadkiv[i]);

}

return rez;

}

public static void hid()

{

List<state> list\_nashadkiv = top.Successors();

current = null;

int t = top.hid;

if (t == 1)

{

if (list\_nashadkiv.Count > 0)

{

Console.Write("i=");

int i = int.Parse(Console.ReadLine());

Console.Write("j=");

int j = int.Parse(Console.ReadLine());

for (int k = 0; k < list\_nashadkiv.Count; k++)

{

if (list\_nashadkiv[k].mas[i, j] == 2)

{

current = list\_nashadkiv[k];

}

}

}

else { Console.WriteLine("game over"); flag = false; return; }

}

else

{

if (list\_nashadkiv.Count > 0)

{

Console.Write("i=");

int i = int.Parse(Console.ReadLine());

Console.Write("j=");

int j = int.Parse(Console.ReadLine());

for (int k = 0; k < list\_nashadkiv.Count; k++)

{

if (list\_nashadkiv[k].mas[i, j] == 1)

{

current = list\_nashadkiv[k];

}

}

}

else { Console.WriteLine("game over"); flag = false; return; }

}

if (current != null) top = current;

else hid();

}

public static state ab\_vidsikania(state current\_state)

{

int rez\_i = -1;

int max = 0;

state.i = 1;

List<state> list\_nashadkiv = current\_state.Successors();

if (list\_nashadkiv.Count == 0) { flag = false; Console.WriteLine("your win"); return null; }

for (int i = 0; i < list\_nashadkiv.Count; i++)

{

if (max < ab\_min\_value(list\_nashadkiv[i],1,2))

{

max = ab\_min\_value(list\_nashadkiv[i],1,2);

rez\_i = i;

}

if (state.i > 1000000 \* state.k)

{

state.i = 1;

return otsinka(current\_state);

}

}

return list\_nashadkiv[rez\_i];

}

public static int ab\_max\_value(state st, int a, int b )

{

if (state.i > 1000000 \* state.k) return 0;

List<state> list\_nashadkiv = st.Successors();

if (list\_nashadkiv.Count == 0) return st.hid;

int v = 0;

for (int i = 0; i < list\_nashadkiv.Count; i++)

{

if (v < ab\_min\_value(list\_nashadkiv[i],a,b)) v = ab\_min\_value(list\_nashadkiv[i],a,b);

if (v >= b) return v;

if (v > a) a = v;

}

return v;

}

public static int ab\_min\_value(state st, int a, int b)

{

if (state.i > 1000000 \* state.k) return 0;

List<state> list\_nashadkiv = st.Successors();

if (list\_nashadkiv.Count == 0) return st.hid;

int v = 3;

for (int i = 0; i < list\_nashadkiv.Count; i++)

{

if (v > ab\_max\_value(list\_nashadkiv[i],a,b)) v = ab\_max\_value(list\_nashadkiv[i],a,b);

if (v <= a) return v;

if (b > v) b = v;

}

return v;

}

public static state otsinka (state st)

{

List<state> list\_nashadkiv = st.Successors();

int min = 56;

int min\_i=-1;

for (int i = 0; i < list\_nashadkiv.Count; i++)

{

if (min > list\_nashadkiv[i].Successors().Count)

{

min = list\_nashadkiv[i].Successors().Count;

min\_i = i;

}

}

if (min\_i != -1)

{

return list\_nashadkiv[min\_i];

}

else return null;

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Vvedit skladnist");

tree t1 = new tree();

state.k= int.Parse(Console.ReadLine());

while (tree.flag)

{

if (tree.flag)

{

Console.WriteLine("stan:");

tree.top.vivod();

tree.top = tree.mini\_max\_desidion(tree.top);

Console.WriteLine("stan:");

if(tree.top != null) tree.top.vivod();

}

if (tree.top != null) tree.hid();

}

}

}

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

## 

Висновок

В ході лр 5 я написав гру з повною інформацією Domineering 8х8.

В ній застосований аглоритм альфа-бета відсікання і оціночна функція Сопова. Спочатку визначається складність, введенням скільки мільйонів станів буде розглянуто альфа бета відсіканням, перш ніж застосувати оціночну функцію Сопова. Алгоритм альфа бета відсікання рекурентно рахує оцінку користності станів починаючи з термінальних станів і вверх по дереву гри відкидаючи частину дерева, яка не впливає на шуканий стан. АБвідсікання схоже на пошук в глибину з частковим відкиданням гілок пошуку. Оціночна функція Сопова перевіряє який хід залишає противнику найменший вибір ходів, тобто серед нащадків поточного стану обирає нащадка з найменшою кількістю нащадків. Вона дозволяє автоматизувати гру будь яких розмірів і за одну ітерацію знаходить дуже сильний хід. Загальна кількість станів в дереві гри приблизно (7\*8)!=56! варіантів, тому без оціночної функції Сопова алгоритм Мінімакса і альфа бета відсікання не зможе її прорахвати навіть за мільйони років. А оціночна функція Сопова знаходить правильний хід менш ніж за секунду. Ссилка на гідхаб з лр5 https://github.com/kolya-sp/alg\_2