Звіт

Програмування інтелектуальних інформаційних систем

Лабораторна робота №2

“Алгоритм мінімакс та альфа-бета обрізки”

Студентки групи ІП-01

Галько Міли Вячеславівни

Лабораторна робота №2

“Алгоритм мінімакс та альфа-бета обрізки”

Завдання:

1. Додати генерацію ворогів з поведінкою. При генерації ви вказуєте кількість згенерованих ворогів одного з двох типів. Перший тип шукає дорогу до гравця, другий тип рухається випадково.
2. Реалізувати алгоритми перемоги за вашою грою - minimax з alpha-beta pruning та expectimax. Функція оцінки має оцінювати “силу” поточної позиції - чим більше число, тим краща позиція.

Реалізація:

Для відтворення гри «Пекмен» маємо реалізувати повноцінну структуру. Отже структуруємо необхідні елементи на групи: алгоритми мінімаксу, гра Пекмен, алгоритми пошуку.

Перш за все, попрацюємо над алгоритмами Minimax – створимо інтерфейс із загальним методом GetBestMove(State, depth). Реалізуємо інтерфейс класами MinimaxStandard та MinimaxAlphaBetaPruning.

Код MinimaxStandard:

```
public class MiniMaxStandard : IMiniMax
{
    public int Apply(State state, int depth, bool maxPlayer)
    {
        if (depth == 0 || state.IsTerminal)
        {
            return state.GetScore();
        }

        int bestScore = maxPlayer ? Int32.MinValue : Int32.MaxValue;
        foreach (var adjacent in state.GetAdjacents(!maxPlayer))
        {
            int score = Apply(adjacent, depth - 1, !maxPlayer);
            bestScore = maxPlayer ? Math.Max(bestScore, score) :
Math.Min(bestScore, score);
        }

        return bestScore;
    }
```

```
    public Cell GetBestMove(State state, int depth)
    {
        var adjacents = state.GetAdjacents(false);
        State bestState = null;
        var bestScore = Int32.MinValue;
        foreach (var adj in adjacents)
        {
            var adjScore = Apply(adj, depth, false);
            if (bestScore < adjScore)
            {
                bestScore = adjScore;
                bestState = adj;
            }
        }

        return bestState.Enemy;
    }
}
```

Код MinimaxAlphaBetaPruning:

```
public class MinimaxAlphaBetaPruning : IMiniMax
{
    public Cell GetBestMove(State state, int depth)
    {
        var adjacents = state.GetAdjacents(false);
        State bestState = null;
        var bestScore = Int32.MinValue;
        foreach (var adj in adjacents)
        {
            var adjScore = Apply(adj, depth, false);
            if (bestScore < adjScore)
            {
                bestScore = adjScore;
                bestState = adj;
            }
        }

        return bestState.Enemy;
    }

    private int Apply(State curr, int depth, bool maxPlayer, int alpha =
Int32.MinValue, int beta = Int32.MaxValue)
    {
        if (depth == 0 || curr.IsTerminal)
        {
            return curr.GetScore();
        }

        if (maxPlayer)
        {
            int maxEvaluation = Int32.MinValue;
            foreach (var child in curr.GetAdjacents(!maxPlayer))
            {
                int evaluation = Apply(child, depth - 1, false, alpha,beta);
```

```
                maxEvaluation = Math.Max(maxEvaluation, evaluation);
                alpha = Math.Max(alpha, evaluation);
                if (beta<=alpha)
                {
                    break;
                }
            }
            return maxEvaluation;
        }
        else
        {
            int minEvaluation = Int32.MaxValue;
            foreach (var child in curr.GetAdjacents(!maxPlayer))
            {
                int evaluation = Apply(child, depth - 1, true, alpha,beta);
                minEvaluation = Math.Min(minEvaluation, evaluation);
                beta = Math.Min(beta, evaluation);
                if (beta<=alpha)
                {
                    break;
                }
            }
            return minEvaluation;
        }
    }
}
```

По-друге, з попередньої лабораторної про пошук найкоротшого шляху, візьмемо реалізацію алгоритму AStar.

Код AStar:

```
public class AStar : IPathSearch
{
    private readonly int[] _rowNum = { -1, 0, 0, 1 };
    private readonly int[] _colNum = { 0, -1, 1, 0 };

    public int FindPath(Field maze, Cell startPoint, Cell destPoint)
    {
        if (maze[startPoint] != 1 || maze[destPoint] != 1)
        {
            return -1;
        }

        bool[,] visitedNodes = new bool[maze.Height, maze.Width];
        visitedNodes[startPoint.X, startPoint.Y] = true;

        PriorityQueue<PathSearchNode, int> queue = new();
        PathSearchNode startPathSearchNode = new PathSearchNode(startPoint,
0);
        queue.Enqueue(startPathSearchNode, GetHeuristic(startPathSearchNode,
destPoint));

        while (queue.Count != 0)
        {
```

```
            PathSearchNode current = queue.Dequeue();
            Cell cell = current.Cell;

            if (cell.X == destPoint.X && cell.Y == destPoint.Y)
            {
                return current.Distance;
            }

            AddAdjToQueue(queue, current, maze, destPoint, visitedNodes);
        }

        return -1;
    }

    private void AddAdjToQueue(PriorityQueue<PathSearchNode, int> nodeQueue,
PathSearchNode current, Field maze,
        Cell destPoint, bool[,] visitedNodes)
    {
        for (int i = 0; i < 4; i++)
        {
            Cell adjCell = new Cell(current.Cell.X + _rowNum[i],
current.Cell.Y + _colNum[i]);

            if (maze.CellIsValid(adjCell) && maze[adjCell] == 1 &&
                !visitedNodes[adjCell.X, adjCell.Y])
            {
                visitedNodes[adjCell.X, adjCell.Y] = true;

                PathSearchNode adjPathSearchNode = new
PathSearchNode(adjCell, current.Distance + 1);
                nodeQueue.Enqueue(adjPathSearchNode,
GetHeuristic(adjPathSearchNode, destPoint));
            }
        }
    }

    private int GetHeuristic(PathSearchNode current, Cell destPoint)
    {
        return current.Distance + Math.Abs(destPoint.X - current.Cell.X) +
Math.Abs(destPoint.Y - current.Cell.Y);
    }
}

public class PathSearchNode
{
    public int Distance { get; }
    public Cell Cell { get; }
    public PathSearchNode(Cell cell, int distance)
    {
        Cell = cell;
        Distance = distance;
    }
}
```

І на кінець, сама реалізація гри. Потребуємо визначити такі класи як: PacmanGame (запуск гри з Program, ходи гравців із заданим алгоритмом для

ворога та ручною грою пекмена), State (утримання даних: поле, координати гравців і фініша; методи: отримання цінності стану, отримання спадкоємців-станів, перевірка на закінченість гри), Field (поле гри із додатковими методами перевірок) та Cell (координата).

Код PacmanGame:

```
public class PacmanGame
{
    private State _currState;
    private IMiniMax _minimax;
    private bool _playerTurn;

    public PacmanGame(Field maze, IMiniMax algorithm, (Cell pacman, Cell
enemy, Cell destination) coordinates)
    {
        _minimax = algorithm;
        _currState = new State(maze, coordinates.pacman, coordinates.enemy,
coordinates.destination);
        _playerTurn = true;
    }

    public void Start()
    {
        Console.WriteLine(_currState);
        while (!_currState.IsTerminal)
        {
            if (_playerTurn)
            {
                PlayerMove();
            }
            else
            {
                EnemyMove();
            }
            Console.Clear();
            Console.WriteLine(_currState);
            _playerTurn = !_playerTurn;
        }

        Console.WriteLine($"{(_playerTurn ? "Enemy" : "Player")} won!");
    }

    private void EnemyMove()
    {
        Thread.Sleep(500);
        var enemyMove = _minimax.GetBestMove(_currState, 3);
        _currState.Enemy = enemyMove;
    }

    private void PlayerMove()
    {
        while (!TryMakeMove(Console.ReadKey().Key))
        {
```

```
        }

        bool TryMakeMove(ConsoleKey key)
        {
            var choice = new Cell(_currState.Pacman);
            switch (key)
            {
                case ConsoleKey.UpArrow:
                    choice.X--;
                    break;
                case ConsoleKey.DownArrow:
                    choice.X++;
                    break;
                case ConsoleKey.RightArrow:
                    choice.Y++;
                    break;
                case ConsoleKey.LeftArrow:
                    choice.Y--;
                    break;
                default:
                    return false;
            }

            bool moveIsValid = _currState.Field.CellIsValid(choice);
            if (moveIsValid)
            {
                _currState.Pacman = choice;
            }

            return moveIsValid;
        }
    }
}
```

Код State:

```
public class State
{
    public Field Field;
    public Cell Pacman;
    public Cell Enemy;
    public Cell Destination;
    private IPathSearch _searchAlgo;

    public State(Field field, Cell pacman, Cell enemy, Cell destination)
    {
        Field = field;
        Pacman = pacman;
        Enemy = enemy;
        Destination = destination;
        _searchAlgo = new AStar();
    }

    public bool IsTerminal => Pacman == Destination || Enemy == Pacman;

    public int GetScore() => _searchAlgo.FindPath(Field, Pacman, Destination)
```

```
                            - _searchAlgo.FindPath(Field, Enemy, Pacman) *
10;

    public IEnumerable<State> GetAdjacents(bool isPacman)
    {
        Cell curr = isPacman ? Pacman : Enemy;
        List<State> states = new List<State>();
        for (int i = -1; i <= 1; i++)
        {
            for (int j = -1; j <= 1; j++)
            {
                if (i == 0 || j == 0)
                {
                    Cell adj = new Cell(curr.X + i, curr.Y + j);
                    if (Field.CellIsValid(adj))
                    {
                        if (isPacman)
                        {
                            states.Add(new State(Field, adj, Enemy,
Destination));
                        }
                        else
                        {
                            states.Add(new State(Field, Pacman, adj,
Destination));
                        }
                    }
                }
            }
        }

        return states;
    }

    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        string[,] stringMaze = new string[Field.Height, Field.Width];
        for (int i = 0; i < Field.Height; i++)
        {
            for (int j = 0; j < Field.Width; j++)
            {
                stringMaze[i, j] = Field[i, j] == 0 ? "▯" : "  ";
            }
        }

        stringMaze[Pacman.X, Pacman.Y] = "😇";
        stringMaze[Enemy.X, Enemy.Y] = "😈";
        stringMaze[Destination.X, Destination.Y] = "▯";

        sb.AppendLine("▯▯▯▯▯▯▯▯▯▯");
        for (int i = 0; i < Field.Height; i++)
        {
            sb.Append("▯");
            for (int j = 0; j < Field.Width; j++)
            {
```

```
                sb.Append(stringMaze[i, j]);
            }

            sb.AppendLine("▯");
        }

        sb.AppendLine("▯▯▯▯▯▯▯▯▯▯");
        return sb.ToString();
    }
}
```

Код Field:

```
public class Field
{
    private int[,] _maze;
    public int Height => _maze.GetLength(0);
    public int Width => _maze.GetLength(1);

    public int this[int x, int y] => _maze[x, y];
    public int this[Cell cell] => _maze[cell.X, cell.Y];

    private Field(int[,] maze)
    {
        this._maze = maze;
    }

    public static Field GetDefaultField() => new Field(new[,]
    {
        { 1, 1, 1, 1, 1, 1, 1 },
        { 1, 1, 1, 0, 1, 1, 1 },
        { 0, 0, 1, 0, 1, 0, 0 },
        { 1, 1, 1, 0, 1, 1, 1 },
        { 1, 1, 1, 1, 1, 1, 1 },
    });

    public bool CellIsValid(Cell cell)
    {
        return cell.X < Height && cell.Y < Width && cell.X >= 0 && cell.Y >=
0 && this[cell] != 0;
    }


    /*
    ▯▯▯▯▯▯▯▯▯▯
    ▯😈▪▪▪▪▪▪▯
    ▯▪😇▪▯▪▪▪▯
    ▯▯▯▪▯▪▯▯▯
    ▯▪▪▪▯▪ ✕ ▪▯
    ▯▪▪▪▪▪▪▪▯
    ▯▯▯▯▯▯▯▯▯▯
     */
};
```

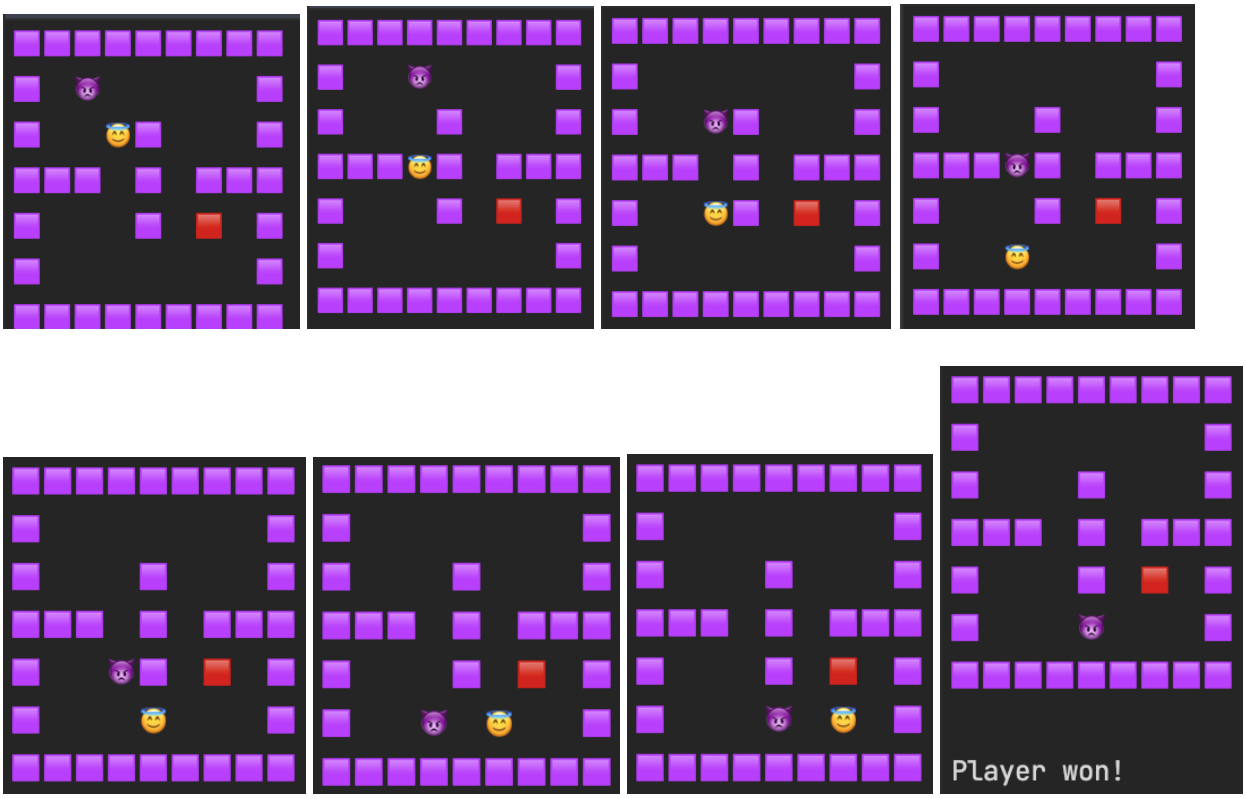
Код Cell:

```
public class Cell
{

    /// <summary>
    /// I index
    /// </summary>
    public int X { get; set; }

    /// <summary>
    /// J index
    /// </summary>
    public int Y { get; set; }

    public Cell(Cell old) : this(old.X, old.Y) {}
    public Cell(int x, int y)
    {
        X = x;
        Y = y;
    }
    private bool Equals(Cell other)
    {
        return X == other.X && Y == other.Y;
    }

    public static bool operator ==(Cell left, Cell right)
    {
        return left.Equals(right);
    }

    public static bool operator !=(Cell left, Cell right)
    {
        return !left.Equals(right);
    }
}
```

Для запуску гри виконуємо код Program:

```
PacmanGame game = new PacmanGame(Field.GetDefaultField(), new
MinimaxAlphaBetaPruning(),
    (new Cell(1, 1), new Cell(0, 0), new Cell(3, 5)));
game.Start();
```

Результати (виграш):



Результати (програш):