ÖZYEĞİN UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

CS 333

2023 Fall

Project 1

Hasbro: Connect-4

By
Umut Çırak

Supervised By
Prof. Olcay Taner Yıldız

1. Introduction

1.1 Project Description

The aim of this project is to simulate Hasbro's "Connect-4" game. In the implementation part, a 6x7 grid is created. A node object is created for each grid cell. The class that manages all nodes is NodeManager. Breadth First Search was preferred for the Pathfinding algorithm. The setup of the program is shown in the figure below.

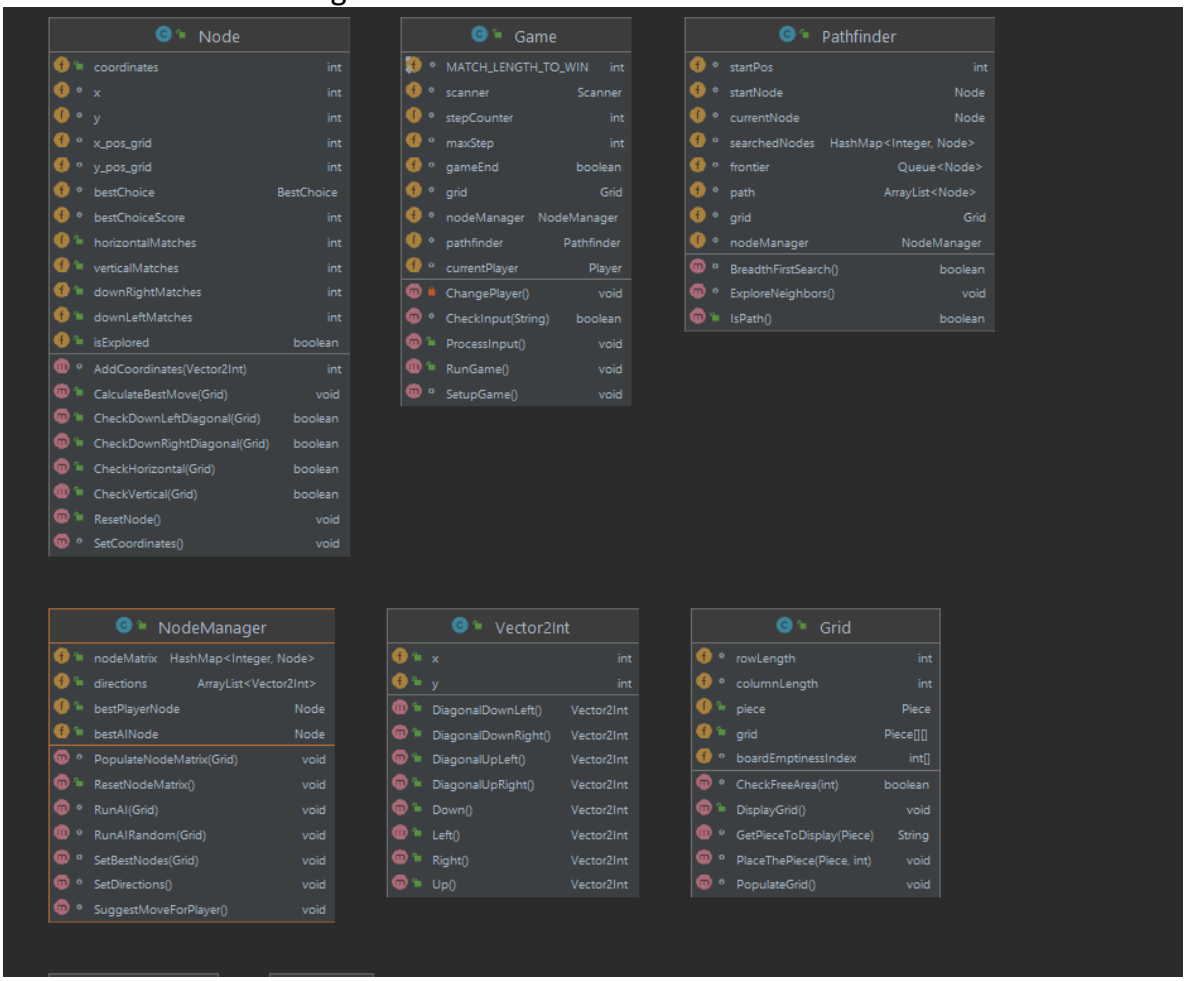
```
public static void main(String[] args) {

    Grid grid = new Grid( rowLength: 6, columnLength: 7);
    NodeManager nodeManager = new NodeManager(grid);
    Pathfinder pathfinder = new Pathfinder(nodeManager,grid);
    Game game = new Game(grid, nodeManager, pathfinder);

    game.RunGame();

}
```

All classes built in the figure below:



2. Methodology

2.1 Grid

The grid object contains the pieces placed on the board. The boardEmptinessIndex array keeps the information in which column there is a space.

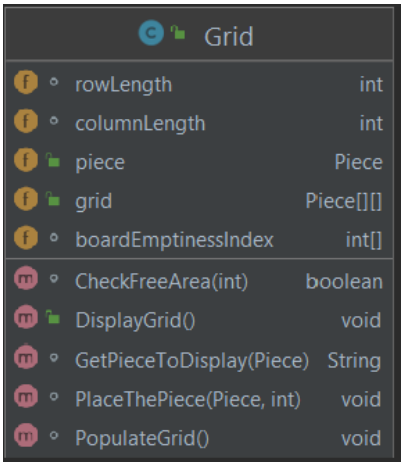
```
public class Grid {

    int rowLength;
    int columnLength;

    enum Piece {X, O, E}   // X: Player Move, O: AI Move E: Empty
    public Piece piece;

    public Piece[][] grid;

    int[] boardEmptinessIndex;
```

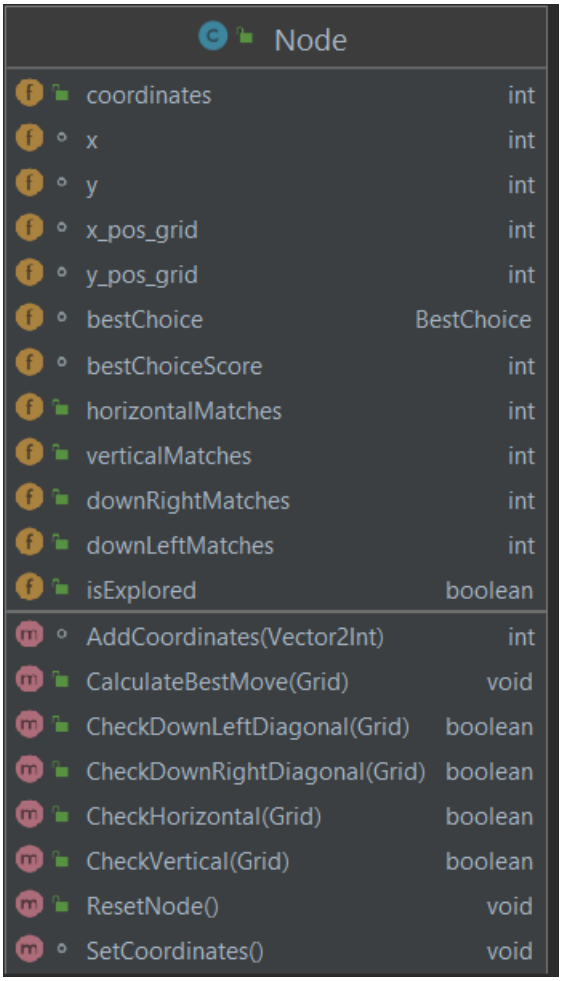


The CheckFreeArea method finds out if there are any empty cells in that column. The Populate Grid method fills the grid with empty cells and the Display Grid method prints the board to the console.

PlaceThePiece method places the pieces into grid with considering the right row position.

2.2 Node

A node is created for each cell in the grid, and this node has x, y coordinates. When the Pathfindings algorithm is run, it is calculated how many horizontal, vertical and diagonal matches each node has. In addition, each node contains information about which move is the best for the next round.



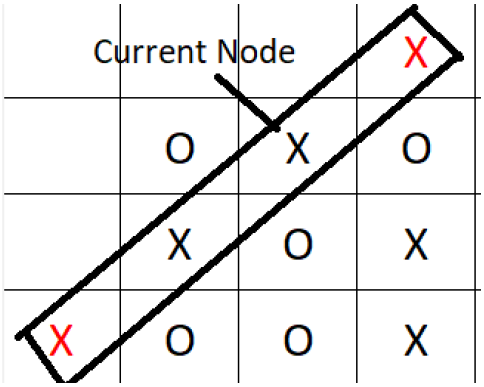
In Horizontal and Diagonal pairings, the next node selection is divided into two as it can be both forward and backward. In vertical match, there is only one probability and the next piece must be on top.

ResetNode method resets all node information to default before the BFS algorithm runs. The Calculate BetsMove method evaluates all possibilities of the node's vertical, horizontal, and diagonal and sets the best choice for the node.

A surplus of grid coordinates is set as position so that each node has a simple key in the hashmap. For example, if the grid coordinate of a node is (0,0), the hashmap key code is 11.

Check methods both calculate how many matches there are and whether there are enough matches to win the game.

Calculating Best Move

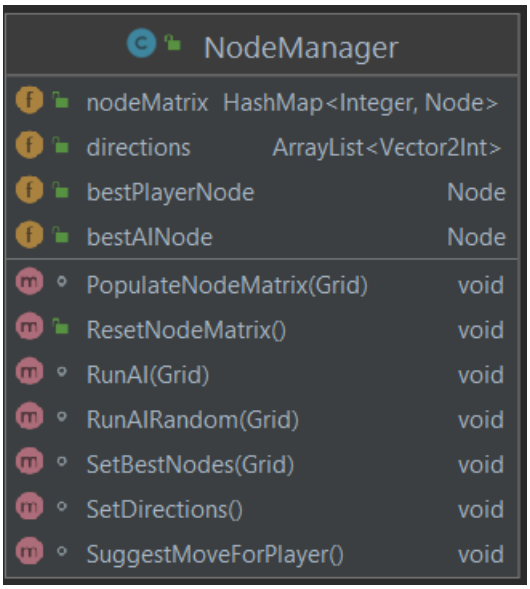


A lot of information is needed to keep track of what the next best move is within a node. First we need to know if the most matches are horizontal, vertical or diagonal. Then we need to know where the next piece should be placed in order for this match to progress, and whether that cell is full or not, and whether there is enough height in that column for it to be placed there. As we can see in the following example, the best move for the node at the two ends of the diagonal line. When the best move is more than one, the first one is selected in the Calculator method. The best selection score and the best selection enum value are then set for this node.

```
enum BestChoice {HorizontalForward, HorizontalBackward, Vertical, DownRightForward, DownRightBackward
    , DownLeftForward, DownLeftBackward, None}
BestChoice bestChoice;

int bestChoiceScore;
```

2.3 Node Manager



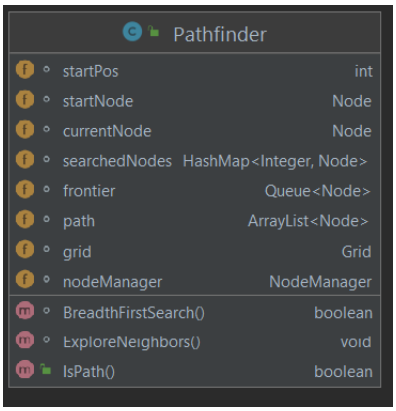
Node Manager keeps all nodes in a HashMap with their positions as keys. The SetBestNodes method updates the nodes with the best score for the next move, both on behalf of the AI and the user. It simply looks at all the nodes in the nodeMatrix and gets the highest scoring node.

The SetDirections method sets the traversal order during Breadth First Search. The RunAI method takes the AI node with the highest score and makes the recorded move when it is the AI's turn to place.

The SuggestMovePlayer method hints the best move while in the user's turn. As seen in the example, the best move for the user is the 4th column.

```
GAME:
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  X  -  -  -  -
-  O  O  -  -  -  -
O  X  X  -  -  -  -
--------------------
Player 1 is moving to place 'X'
pick a move (a column number between 1-7)
Best Move for Player is 4. column to complete HorizontalForward
```

2.4 Path Finding



It works with Breadth First Search. First, the start node is determined and set to the current node. Then, neighboring nodes are visited with the directions we get from the Node Manager object. If neighbors have not been visited before, they are added into frontier and searchedNodes.

```
void ExploreNeighbors()
{
    ArrayList <Node> neighbors = new ArrayList<~>();

    for (Vector2Int direction: nodeManager.directions)
    {
        int neighborCoordinate = currentNode.AddCoordinates(direction);

        if(nodeManager.nodeMatrix.containsKey(neighborCoordinate))
        {
            neighbors.add(nodeManager.nodeMatrix.get(neighborCoordinate));
        }
    }

    for (Node neighbor: neighbors)
    {
        if(!searchedNodes.containsKey(neighbor.coordinates))
        {
            searchedNodes.put(neighbor.coordinates,neighbor);
            frontier.add(neighbor);
        }
    }
}
```

Every loop, the current node is changed and its neighbors are traversed again. The algorithm terminates if the current node has 4 matches.

```
boolean BreadthFirstSearch()
{
    frontier.clear();
    searchedNodes.clear();

    var nodeMatrix :HashMap<Integer, Node> = nodeManager.nodeMatrix;

    boolean isRunning = true;
    frontier.add(startNode);
    searchedNodes.put(startNode.coordinates,startNode);

    while (isRunning && frontier.size() > 0)
    {
        currentNode = frontier.poll();
        currentNode.isExplored = true;
        ExploreNeighbors();

        if(currentNode.CheckHorizontal(grid) || currentNode.CheckVertical(grid)
                || currentNode.CheckDownLeftDiagonal(grid) || currentNode.CheckDownRightDiagonal(grid))
        {
            isRunning = false;
            return true;
        }
```

2.5 Game Cycle

The game loop continues until the maximum number of steps is reached or any match is found. In each step, user input is taken and the part is placed on the grid. If it's AI's turn, it moves the highest scoring AI node from the most Node Manager. After the piece is placed, the board is printed on the console. Then it is checked whether there is a match and the scores of the nodes on the board are updated again.

```
public void RunGame()
{

    while (this.stepCounter < 42 && !gameEnd)
    {
        this.ProcessInput();
        grid.DisplayGrid();
        this.gameEnd = pathfinder.IsPath();
        nodeManager.SetBestNodes(grid);

        if(this.gameEnd == true)
        {
            System.out.println("!!! " + currentPlayer.toString()  +" IS WON !!!");
        }
        ChangePlayer();
    }
```