CIG Coursework Crib Sheet

Board Designer

- You can use this to design your own environments. Double click on the exe file to start. Drag cells to the board to add walls, difficult terrain or power-pills
- The power cost of each type of cell can be set by right-clicking on the cell type. Power-pills (indicated by lightning flash) should have negative values
- The cell type can be set to blocking if desired to prevent the robot traversing it.
- The visibility (depth to which the robot can 'see') can set to a value between 0 and 3.
- Wall cost in the left hand refers to the outside walls of the board
- Boards can be saved/loaded using the file menu (as xml files)
- You are supplied with 5 test environments but you can create your own as well

Writing a Pathfinding Algorithm

- Go to the VS_Project folder and double click on PathFinderDemo.sln
- A demo console application and a demo GUI application are provided. In these demos, the robot simply selects a random move.
- Start the GUI code - press Load when the application opens and select one of the test environments provided. Press Run.

How to write your own code:

Modify the MyAlgorithmClass. Your code needs to be inserted in the run() method inside the loop:

```
while (Board.isBoardRunning())
          {
                // write your code here
                // your code needs to determine what move to make
                // val represents the direction of the move
                // you have selected

                Board.move(val);
                Cell cell = Board.getCurrentCell();
                System.Console.WriteLine(cell.ToString());
          }
```

Some hints/tip regarding the code are given below

The Board object

The game environment is represented by the Board object. All instructions to move the robot are methods of the Board object.

Cell Objects

Squares on the Board are represented as Cell objects. Cells have column and row properties to represent their position.

- int cell.powerCost – positive values indicate the energy cost associated with moving to a square
- boolean cell.isBlocking : if true, cannot move to this square
- int cell.isVisited returns the number of times a cells has been visited by the robot

The powerCost property represents the cost if moving onto that square (a negative value occurs when the square contains power). Not all cells may be traversed. If the cell represents a terrain type that cannot be traversed the isBlocking() property will be set to true.

Directions

Directions are represented using the enumerated structure Direction, described below:

- `Direction.Undefined = -1,`
- `Direction.North = 0,`
- `Direction.NorthEast = 1,`
- `Direction.East = 2,`
- `Direction.SouthEast = 3,`
- `Direction.South = 4,`
- `Direction.SouthWest = 5,`
- `Direction.West = 6,`
- `Direction.NorthWest = 7,`

Board.getOppositeDirection(Direction) returns the opposite direction to the supplied direction (i.e. north returns south).

Board.getDirectionTo(someCell) returns the direction to someCell from the current cell – but this only works for cells which are ***directly adjacent*** to the cell you are in. Otherwise, the method returns -1.

Moving

Moves can be specified in two ways:

```
public Cell move(Direction direction);
public Cell move(int directionId);
```

- The cost of the move will be deducted from the current energy count. If the move is not allowed (e.g. into a wall) the energy cost is still deducted, but the robot remains in the current cell.
- If move() returns null, then the attempted move was illegal and was not made. Otherwise, the method returns the new cell the robot is at.

setCurrentCell(cell) moves the robot to the cell specified as an argument. Note that you can only 'jump' like this to a cell you have previously visited. This method can only be used if the Board visibility is set to 0 or 1.

Making a sequence of moves

Instead of moving one cell at a time, you can move the robot through a predefined set of moves (represented by a List structure containing Direction objects)

Board.Solve(List<Direction>)

The current location of the robot

Board.getCurrentCel() returns a Cell object representing the cell currently occupied

Distance to the end

- Board.getEuclideanDistancetoEnd(Cell) returns the Euclidean (straight line) distance from the specified cell to the end cell.

Looking around you

You can ask the board to return the set of adjacent cells to the cell you are currently in, in a given direction. By specifying a depth value (visibility), you can increase the extent to which you can 'look ahead'. However, the maximum depth you can look ahead to is 3

Two methods can be used to look around:

- Board.getCell(Direction, depth) returns a single cell in the direction specified at the depth specified . The method returns a cell even if it is a blocked cell you cannot move to
- Board.getCells(Direction, depth) returns an array of *depth* cells in the direction specified, e.g the 3 cells to the North

Note the method does not return any cells beyond a wall: e.g if you specify a depth of 3 to the North and there is a wall at position 1, then the array has null values for positions 2 and 3.

For more in depth information please refer to the Documentation.cfm file.