BLG435E

Artificial Intelligence

Assigment 1

Ozan Arkan CAN
040090573

November 4, 2012

Question 1

Home Service Robot

Performance: Well serving, happiness of owner
Environment: Home, People, Stuffs
Actuators: Robotic arm
Sensors: Motion sensor, optic sensor, infrared sensor etc.

Partially Observable: Robot cannot see the whole world.
Multi Agent: If people are evaluated as agent than the task has multi agents.
Stochastic: There are possible actions that can be done by robot in definete situation. Robot might act different in same situation.
Sequential: Actions of robot effect the next situations.
Dynamic: Task environment is real world and real world is changing continously.
Continous: The actions are executed in continous time.
Goal based agent: Aim of the robot is serving to people and its actions are not rule based. Also, robot cannot see the whole world and it might be act depend on its aim. So, robot behaves to achive its goal.

A dynamic personalized e-mail organizer for e-mail services

Performance: High percentage on choosing right category
Environment: Web server
Actuators: File organizing commands or operating system
Sensors: e-mail input

Fully Observable: The organizer can see all e-mails.
Single Agent: There is only one agent, the organizer.
Stochastic/Deterministic: If the organizer is a rule based agent than environment is deterministic, else if organizer decides based on some probabilistic parameters than the environment is stochastic.
Episodic: Organizer takes an e-mail and organizes it in one episode. This situation does not effect the next episodes.
Static: E-mails are stable.
Discrete: Organizer makes an action in periodic time (a few cpu cyles).
Model based reflex agent: Organizer might manage the e-mail by rule based or probabilistic model.

Autonomous car driver

Performance: Quicker, Safer, Cheaper

Environment: Road, Cars, People
Actuators: Wheel, gas pedal, break
Sensors: Optic sensor, infrared sensor etc.

Partially Observable: Driver can see only its near circle.
Multi Agent: There are another cars.
Stochastic: There are many options in any situation and driver can chose one of them.
Sequential: Actions of driver are done by an order.
Dynamic: Task environment is real world and real world is changing continously.
Continous: The actions are executed in continous time.
Utility based agent: There are many goals such as time, money, safety. Driver can act according these goals to maximize the outcome.

Activity recognition and anomaly detection software agent

Assumption: The task is observing air traffic and detecting anomaly.

Performance: Detecting any anomaly
Environment: Radar system
Actuators: communication system
Sensors: Radars, flight data

Fully Observable: Data of every flights can be reachable digitally.
Single Agent: Planes are object in this task.
Stochastic/Deterministic: Agent might have model to act but if its rule based than the environment is deterministic else the environment is stochastic.
Sequential: Task is holding in an order like observe, detect, act.
Dynamic: Task is detecting anomaly, so environment must be changeble.
Continous: The actions are executed in continous time.
Model based reflexagent: Agent needs model to detect anomaly. It could be probabilistic or rule based.

Question 2

If algorithm is consistent, for every node n and every successor n' of n generated by any action a, than the estimated cost of reaching the goal from n less or equals than the step cost getting to n' plus the estimated cost of reaching the goal from n'.

$h(n) < c(n, a, n') + h(n')$ (Triangle Inequality)
$f(n') = g(n') + h(n')$
$f(n') = g(n) + c(n, a, n') + h(n')$
$f(n') \geq g(n) + h(n)$
$f(n') \geq f(n)$

$f(n)$ is nondecreasing on any path.

Let n is start node, n' is the successor and g is the goal node.
Assume that:

$c(n, a, n') = 1$, $c(n, a, g) = 10$, $c(n', a, g) = 10$
(Actual costs with Triangle Inequality, Admissible) and

$h(n) = 8$, $h(n') = 6$
$h(n) > h(n') + c(n, a, n')$ Inconsistency

Question 3

Problem Definition

Initial State: Given board configuration is initial state.
Actions: Vertical blocks can move along y axis, horizontal blocks can move along x axis. They are moved while there is no another block or it is not near the border.
Transition Model: After an action has done, the result is new board configuration.
Goal Test: The goal is moving the target block to gate which is at 3rd row, 6th column. Goal test is controlling the whether the target block is at gate or not.
Path Cost: Path cost is number of moves for sliding a block in each action.

Analysis of Uninformed Search Algorithm

In this chapter, the worst case senario is given belove for the analysis:
-All cells that are on the path which is between target block and gate are occupied. This means there are 4 blocks.
-There are anothor blocks on those blocks. This adds 4 blocks more.

So, 8 blocks must be moved to target block can move.
According this information number of branch is number of total blocks. The maximum depth of tree is $m = 8 + 1 = 9$ (The one comes for moving the target block to gate.)

Breadth First Search

The BFS algorithm searches the tree layer by layer. In this problem this facility means in each action a block that has not been moved in layer is moved. This cause if node that is on the first places belongs to solution path, searching the remained nodes in the layer slowdowns the algoritm, but algorithm is available for finding optimal solution. Because the path costes are nondecreasing.
So, time complexity is $O(b^d)$, space complexity is $O(b^d)$.

Depth First Search

The DFS searches the tree following branches step by step. According the problem this algorithm can be stack a branch because, algorithm can move a block right to left, left to right or top to down, down to top forever. But if we create a definite search space(give a tree the algorithm in order to creating running time) the algorithm can reach the solution. The running time of algorithm is $O(b^m)$ and the space complexity is $O(bm)$.

Uniform-cost Search

The Uninform-cost search algorithm has same strategy with BFS algorithm, but there is a difference to struct the queue. The algorithm expands the nodes by their path cost as ascending order. So, algorithm chooses node that has smallest path cost in that layer. By this strategy, algorithm reaches the solution faster than BFS.
The running time and the space complexity are same as complexities of BFS, but this analysis is for worst case. In average case, complexties decrease by path cost.

Implementation of A* Algorithm

Algorithm Definition

In this program, A* algorithm is implemented to solve the program. It is a kind of best-search algorithm. Node selection depends on the total cost. A heuristic algorithm like A* needs two function which are cost for reaching current node and the cost of heuristic value for reaching the goal node. In this program, two different heuristic functions are used to calculate the heuristic cost.
Heuristic Function1: Number of moves for moving target block to gate.
Heuristic Function2: Number of blocks on the path that is from target block to gate.
Path Cost ($g(n)$): Path cost is number of moves for sliding a block in that node.

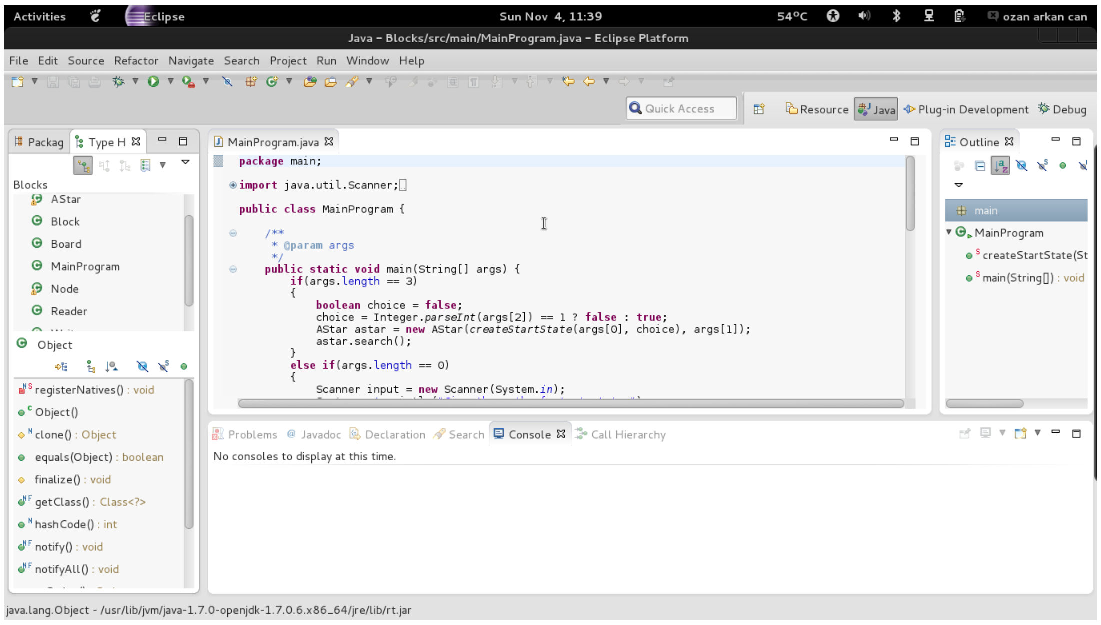
textbfTotal Cost: $f(n) = g(n) + h(n)$

In this implementation a priority queueu is used as frontier and the nodes are sorted depends on their total cost in ascending order.

Development Environment

The program has been develop with *Eclipse* development environment on Fedora17 64 bit system. It has been build and run from this ide. Program can be run with non parameters or with 3 tree parameters which are path for input file, path for outputfile and number of herustic function that is used to calculate cost.
There is also a folder that contains all java files with a makefile. Files can be build and run without ide.

To build: make
To run: java Mainprogram or java Mainprogram inputFilePath outputFilePath choiceForHeuristicFunction
To clean: make clean



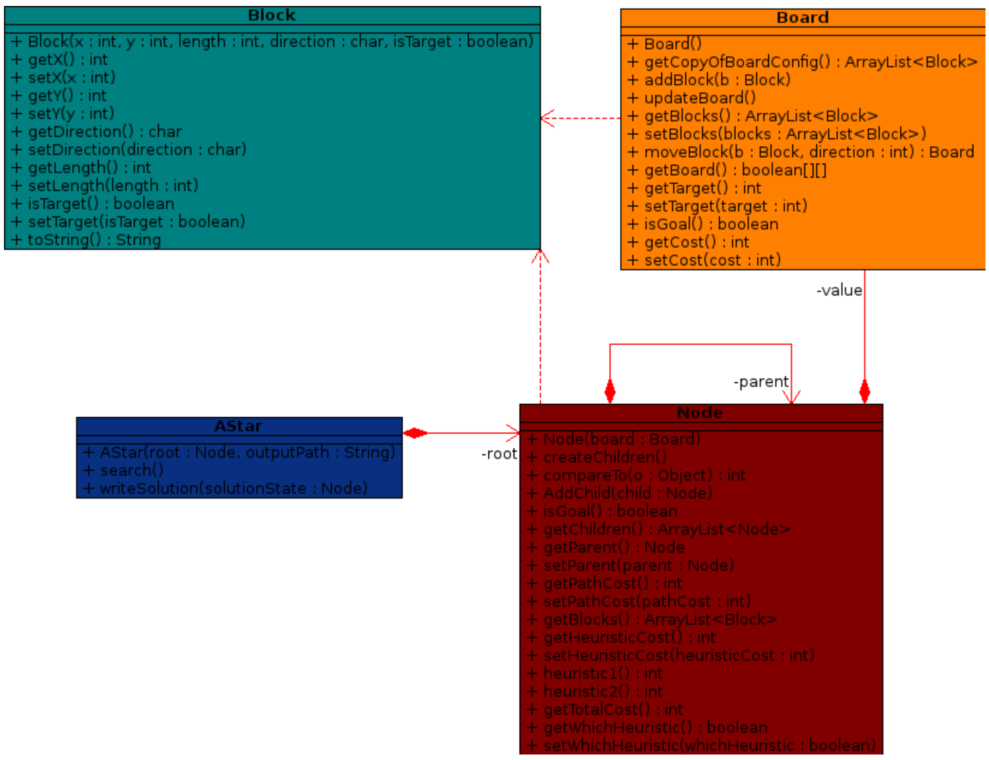
```java
package main;

import java.util.Scanner;

public class MainProgram {

    /**
     * @param args
     */
    public static void main(String[] args) {
        if(args.length == 3)
        {
            boolean choice = false;
            choice = Integer.parseInt(args[2]) == 1 ? false : true;
            AStar astar = new AStar(createStartState(args[0], choice), args[1]);
            astar.search();
        }
        else if(args.length == 0)
        {
            Scanner input = new Scanner(System.in);
```

No consoles to display at this time.

java.lang.Object - /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.6.x86_64/jre/lib/rt.jar

UML Diagram

The UML diagram of the program is given belove.


Block
+ Block(x : int, y : int, length : int, direction : char, isTarget : boolean)
+ getX() : int
+ setX(x : int)
+ getY() : int
+ setY(y : int)
+ getDirection() : char
+ setDirection(direction : char)
+ getLength() : int
+ setLength(length : int)
+ isTarget() : boolean
+ setTarget(isTarget : boolean)
+ toString() : String
Board
+ Board()
+ getCopyOfBoardConfig() : ArrayList<Block>
+ addBlock(b : Block)
+ updateBoard()
+ getBlocks() : ArrayList<Block>
+ setBlocks(blocks : ArrayList<Block>)
+ moveBlock(b : Block, direction : int) : Board
+ getBoard() : boolean[][]
+ getTarget() : int
+ setTarget(target : int)
+ isGoal() : boolean
+ getCost() : int
+ setCost(cost : int)
-value
-parent
AStar
+ AStar(root : Node, outputPath : String)
+ search()
+ writeSolution(solutionState : Node)
-root
Node
+ Node(board : Board)
+ createChildren()
+ compareTo(o : Object) : int
+ AddChild(child : Node)
+ isGoal() : boolean
+ getChildren() : ArrayList<Node>
+ getParent() : Node
+ setParent(parent : Node)
+ getPathCost() : int
+ setPathCost(pathCost : int)
+ getBlocks() : ArrayList<Block>
+ getHeuristicCost() : int
+ setHeuristicCost(heuristicCost : int)
+ heuristic1() : int
+ heuristic2() : int
+ getTotalCost() : int
+ getWhichHeuristic() : boolean
+ setWhichHeuristic(whichHeuristic : boolean)

Algorithm Analysis

Implemented A* algorithm run with two different heuristic functions. Strategy for the first function is choosing the node that has nearest target block to gate. But this function misses the number of moves that for cleaning the path. So there are a lot of nodes that has same heuristic value and decision for choosing the node effects the algorithm complexity.
Second heuristic function aims to remove the blocks that are between target block and gate. Algorithm slides a block if the block on the path or the block must be moved for removing the obstacle block. Algorithm cannot visit any node again because the costs are nondecreasing. In conclusion, A* algorithm is faster than uninformed algortihms and second heuristic function works better than other.

Conclusion

This assigment helps to exercise the following issues:

- To define a problem
- To define properties of environment and PEAS components
- To define an agent
- Uninformed and heuristic algorithms
- Data Structures