UNIVERSITY OF NOTTINGHAM

G54DIA COURSEWORK 1 REPORT

NAME: Sheng Wang

STUDENT ID: 4207954

EMAIL: psysw1@nottingham.ac.uk

1. Introduction

This coursework is aimed at designing a single agent which can collect and deliver water to stations (customers) in a 2D environment. Tasks (Water Requirement) are generated by stations periodically. The environment also contains several wells which water can be collected. The expectation for this agent is to deliver as much water to as many stations as possible in a given fixed period of time.

First of all, this report will introduce the implementation of this agent system from specification of the task environment, architecture design to software implementation. Then it will give analysis of the performance of this agent and illustrate the relationship between the selected architecture and the system performance.

2. Specification

Environment

Russell and Norvig suggest the following classification of environment properties (Russell and Norvig, 1995).

Accessible versus **inaccessible**.

In accessible environment, agents do not need to maintain internal state to keep track of the world as they can obtain complete, accurate, up-to-date information about the environment's state. However, unfortunately, most real-world environments are not accessible in this sense.

Deterministic versus **non-deterministic**.

If an agent can determine the next state of the environment by given its current state and the action it will perform, then this environment is determinisitic. Non-deterministic is a common property of real-world environment, as nearly every action can fail in real case.

Episodic versus **nonepisodic**.

In an episodic environment, all actions an agent can perform are independent. This means all actions will consistently produce the same effect to the environment. People always try to maintain this property of machines in order for machines to produce reliable and consistent performance.

Static versus **dynamic**.

An environment can be assumed to remain unchanged if there is no performance of actions by the agent, while dynamic environment is also operated by other existing processes and therefore has changes beyond the agent's control.

Discrete versus **continuous**.

If there are a limited number of clearly defined percepts and actions, then the environment is discrete.

The task environment given in this coursework is **inaccessible**, **deterministic**, **episodic**, **dynamic** and **discrete**.

1. **Inaccessible**

   First of all, the agent has limited view range and the environment contains randomly distributed stations and wells. This means that the agent cannot know the location of a station or well unless it has ever seen this station or well before. In addition, as the agent can know whether a station has a task or not only if it can see this station.

Therefore, in order to optimize the process of water delivery in general, it is highly recommended to implement an 'explore' function for the agent. Understanding complete location information can help the agent to plan the route more effectively.

Then, as the agent cannot get the location of itself from the environment, geting itself localized is quite necessary. One common way to stay localized is to use odemetry or path intergration. The accuracy of the localization can be very hard to guarantee for physical agent, e.g. Robots, however, for this agent system, it is quite easy and the result can be pretty accurate.

2. **Deterministic**

   Software environments are usually deterministic which means the action performed will never fail, as they are constructed by precise rules. For this coursework, for example, with an initial position of (x,y) and an action of MoveAction(North), the next position of the agent must be (x,y+1). This property also maintains for all other actions.

3. **Episodic**

   Actions the agent can perform will not affect each other and will produce the same result no matter how often an action is performed. This is different from human body, which may have muscle fatiguability and therefore cause that arms can no longer work as expected.

4. **Dynamic**

   The environment for the coursework seems static based on the fact that locations of stations, wells and fuel pump fixed in the map or the fact that tasks persist until they are achieved, but it is dynamic in general because of the fact that tasks are generated randomly by stations. Therefore, the agent must perform information gathering actions to determine the state of the environment in order for it to select an appropriate action to perform (Moore,1990).

5. **Discrete**

   The agent movement is like the game of chess, which has finite number of states. Therefore, the environment for this coursework is discrete.

Percept

The informations an agent can obtain in this coursework are :

About Environment :

1. **Type of a cell**

The agent can determine the type of a cell it discovers, station, well, fuel pump or empty.

2. **Relative position**
   The agent can know the relative position of a cell based on its current position.
3. **Task**
   The agent can get the requirement of a task that a station generates as long as the station is visible.
4. **Remaining Time**
   The agent can get the current remaining timesteps from the environment.

About Agent Itself :

1. **Fuel level**
   The agent can know how much fuel is left currently.
2. **Water Level**
   The agent can know how much water is left currently.
3. **Score**
   The agent can know how much scores and tasks it has got and completed currently.

Actions

Actions avaliable for the agent are listed below :

1. **Move**
2. **Load Water**
   The agent can take water from a well.
3. **Deliver Water**
   The agent can deliver water to a station.
4. **Refuel**
   The agent can refuel at the fuel pump (In this coursework, it is at the centre of the environment).

3. Design

Which Architecture?

Reactive architecture becomes one of the most commonly used methods for robot control. It works well on robots because it tightly connects the robot's sensors and effectors. This architecture allows robots to operate on a short time-scale and react to changes of the environment rapidly. However, purely reactive systems do not use any internal representations of the environment and do not look ahead the outcomes of their actions, which makes that planning for water delivery in general is very hard. In addition, as purely reactive systems produce intelligent behaviours by emerging from the interaction of the agent with the environment, parallel programming technique may be necessary for implementing agents based on this kind of architecture. This increases the difficulty for building such system. Therefore, in all, reactive architecture can be good for some cases, but may not be suitable for this task.

As for **hybrid architecture**, one of the same problems with reactive architecture is the implementation difficulty. This architecture is perfect in theory, but may be too hard to implement in the real world.

Therefore, **deliberative architecture** will be adopted for this coursework. Reasons for this are listed below:

1. As the task environment is **inaccessible** (Current position of the agent in the environment cannot be obtained from the environment), maintaining internal representation is necessary for this coursework. In addition, with internal representation about the environment, it would be much easier to make optimization of water delivery in general.

2. Although usually deliberative architecture requires a lot of time for computation, this does not matter for this coursework because computation time does not consume time step. This means the agent can think pretty hard before make any decision **without worrying about time**. Besides, even assuming that computation needs to consume time step, the simplicity (deterministic, episodic and discrete) of the environment of this coursework still makes deliberative architecture work well.

3. Deliberative architecture is comparatively **easy** to implement. It can be achieved by just using sequential programming technique.

What Decision Making Model?

Practical reasoning works much closer to how human reasons compared to deductive reasoning, which does reasoning by proving theorems. Based on Bratman (1990), practical reasoning is a matter of comparing and selecting different considerations about competing options, where the relevant

considerations are from what the agent desires about and what the agent believes. It explains how to map percept to actions comprehensively in theory.

Practical reasoning model consists of two distinct activities, goal deliberation and means-ends reasoning. The former involves deciding what an agent wants to achieve (Desires), while the latter involves deciding how an agent achieves that (Plans).

Beliefs

Beliefs store the knowledge of an agent about the environment. Particularly for this coursework, main useful information about the environment is as follow:

About agent itself
1. Current position of the agent
2. Current cell the agent stands on
3. Current fuel level
4. Current water level

About environment
5. A list of positions to explore
6. A list of locations of stations
7. A list of locations of wells
8. A list of tasks an agent have seen

Why agent information?

The reason for storing the agent's current position is to calculate the position of a station or a well in the environment based on its current position and the relative position information from percepts. The rest information is stored in order to be passed to planner to make decisions.

Why not duplicate the environment?

Besides, in order to build the internal knowledge about the environment, some people would try to nearly duplicate the same map with the real environment. However, as empty cell in this coursework nearly contributes nothing to getting higher score and searching a whole map just for a subset of map information (e.g. Searching a whole map to find a station or well costs more time than searching corresponding station set or well set) is a waste of time, here separate lists containing different useful information are used.

Desires (Deliberation)

This agent system only needs two desires to meet the requirement of this coursework, explore the environment and finish tasks.

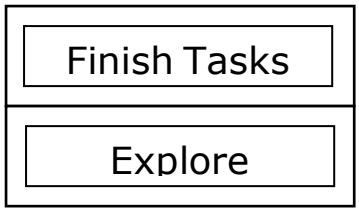


Figure 1 priority of desires in general

Why priority of desires?

In order to get as high scores as possible, the agent should go to finish tasks as long as it has tasks at hand. Therefore, desire 'finish tasks' should have higher priority than 'explore' in general. This is illustrated by figure 1.

However, before the whole map is explored, it will be pretty hard to work out an excellent plan about in what order these tasks should be finished. Therefore, if the environment has been explored, then desire 'explore' should have higher priority than 'finish tasks'. Otherwise, 'finish tasks' should have higher priority.

Plans (Means-ends reasoning)

Means-ends reasoning is the process of deciding how to achieve a goal using available actions. It will take a goal or task and current knowledge about the environment (beliefs) into account, then produce an action available to the agent.

What components of plans?

An action will be decided by three components: Desire Selection, Destination Selection and Action Selection and in this order (figure 2).

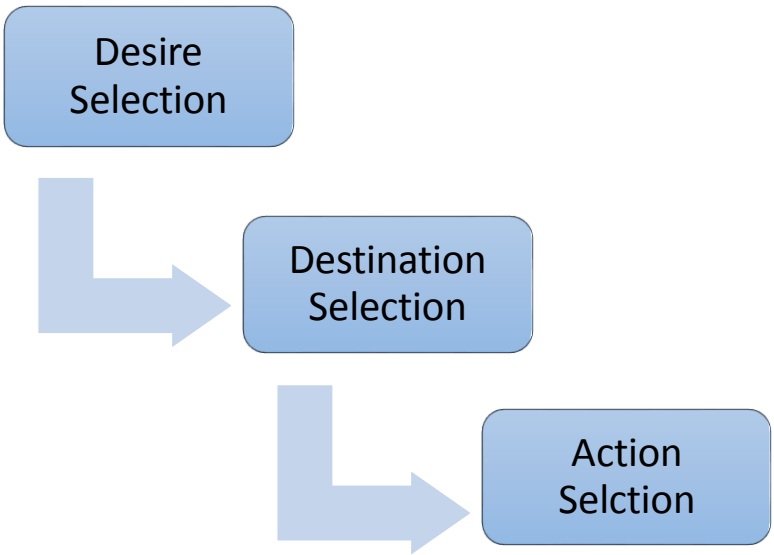


Figure 2 hierarchy of three components

Desire Selection (What goal?)

Desire selection is based on the agent's desires, which in this case 'finish tasks' and 'explore'. This decides what the agent should do at the moment.

Destination Selection (Which task or exploration point?)

If the goal of the agent is 'explore', the destination is one of points in the environment to explore, otherwise, the destination is the location of a station which has a task currently.

Action Selection (How to perform task or exploration?)

Although the destination has been selected previously, the agent may not be able to move towards that destination directly (not enough fuel, for example). Therefore, a final feasible decision will be made at this step. Detailed algorithm for deciding the final action will be introduced in 'Algorithms' part.

How to Map Percept to Actions?

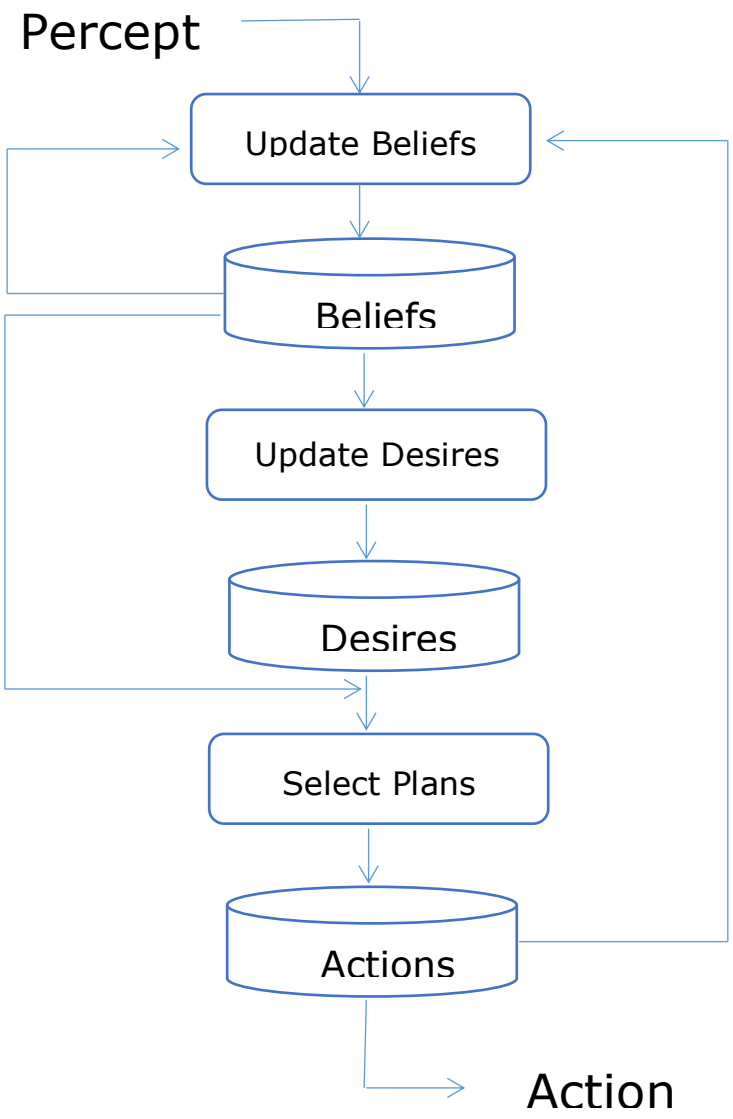


Figure 3 general procedure of mapping percept to actions

Update Beliefs

As shown in figure 3, percept collected by the agent will first update the agent's beliefs (e.g. station, well and task). For some case, beliefs will need to update themselves. For example, as long as the agent has explored all the areas of the environment, it can start to generate a new good (As TSP problem is an NP problem, optimal solution is nearly impossible.) route for later map traverse. Same works for task list optimization. Detailed algorithm will be introduced in 'Algorithms' part.

Update Desires

If the agent checks its beliefs and finds that there is no task left, then it will start to 'explore'. Otherwise, it will start to finish tasks. However, if the map has not been fully explored, then the only desire is to explore the map.

Select Plans

In order to make a plan, the agent needs its beliefs and desires as input. Some actions can update the agent's beliefs (e.g. move action will update the current position of the agent).

Algorithms

How to explore the environment?

Exploration Points Optimization
Input: locations of all stations seen by previous exploration
(x1,y1), (x2,y2)…… (xn,yn)

1. Find the largest x and y coordinate, named as max_x and max_y
2. Find the smallest x and y coordinate, named as min_x and min_y
3. A rectangle area with (max_x,max_y), (max_x,min_y), (min_x,max_y) and (min_x,min_y) as its four vertices can cover all the stations
4. Divide this rectangle area into several small square areas, of which the side length is (View_Range*2 + 1)
5. For every small square area, test if it covers any stations. As long as it covers one, record the central position of that area as a exploration point
6. Divide these new exploration points into four groups based on the quadrant they lie on
7. Sort these four groups in descending order based on the number of points they contains
8. Add all points in one group to final list in turn

Traverse Map
Input: a list of exploration points L, agent's current position and current fuel level

1. For every point in list L, test if the agent can go to that point and still be able to return to the origin point (fuel pump)
2. If there is one point the agent can reach, return this point
3. Otherwise, return the origin point

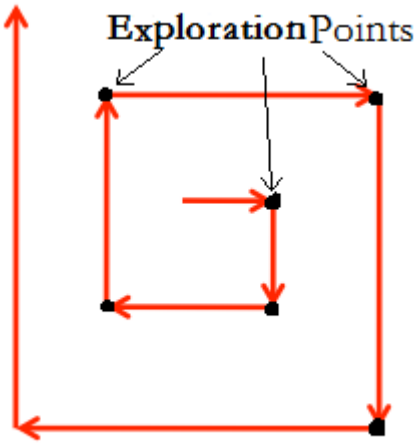


Figure 4 exploration approach selected

First, the agent will use the approach shown in figure 4 to explore the environment. Compared to other approaches, this approach is the easiest one to cover the whole map without missing any point. After the exploration is finished, the agent can get all useful location information (e.g. stations, wells) and all tasks generated at the moment.

However, as stations and wells are not located evenly in the environment, the agent does not need to explore the whole map again after the first exploration. Therefore, the algorithm 'Exploration Points Optimization' is used to generate new exploration points.

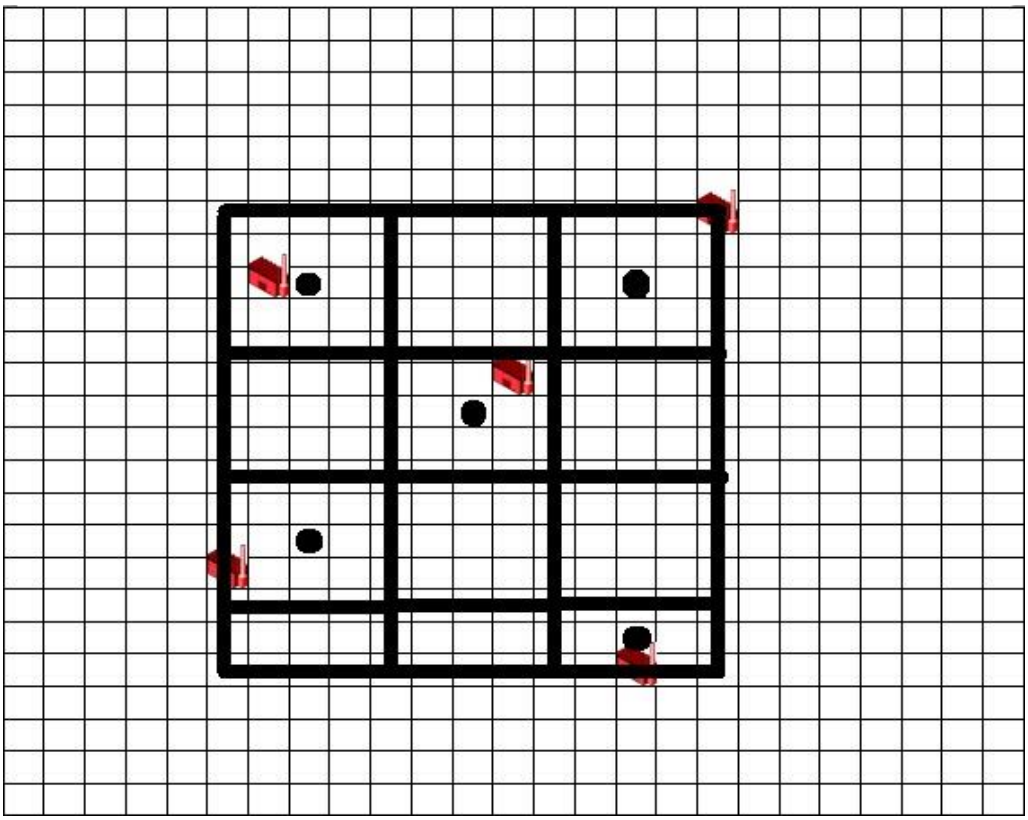


Figure 5 exploration point optimization

After optimization, the agent does not need to explore the whole map. Instead it will only need to explore 5 points as shown in figure 5.

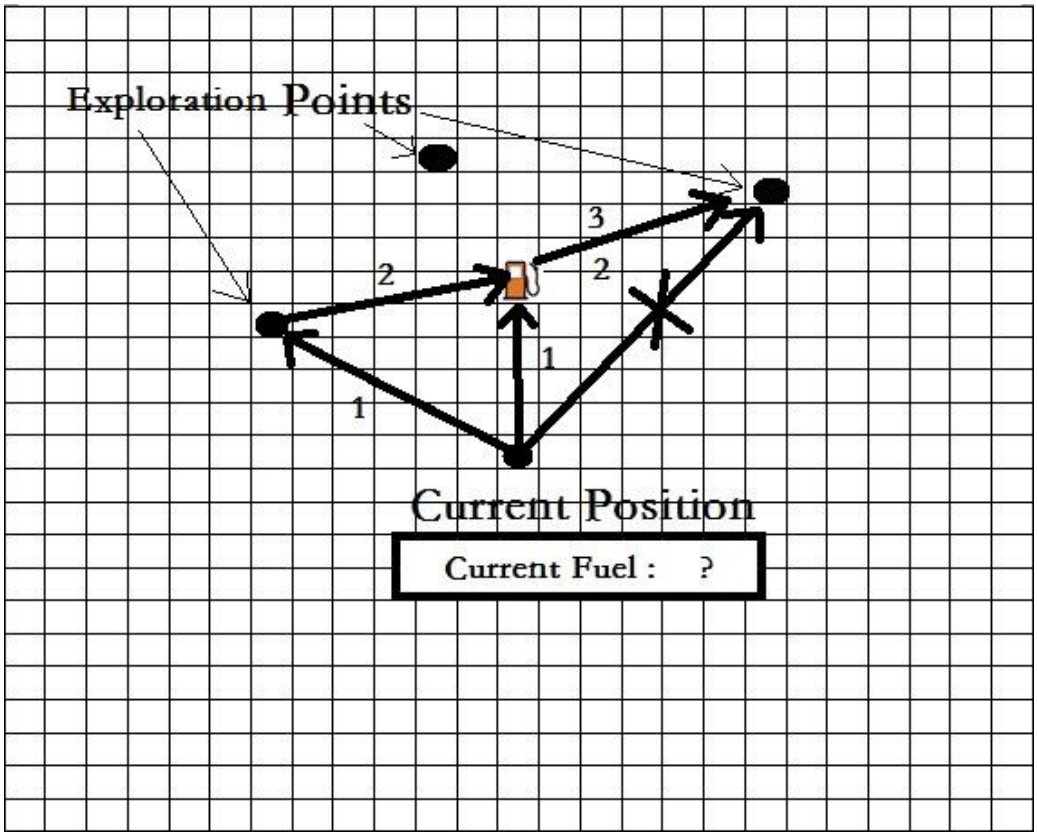


Figure 6 Traverse Map algorithm

Figure 6 illustrates the 'Traverse Map' algorithm. As there is not enough fuel to safely (i.e. have enough fuel left to return to the fuel pump) go to the top right point, one approach is first go to the fuel pump and then go to the target.

However, imaging if the agent currently has, for example, 99 fuels, then refuelling would be a waste of fuel. In order to make good use of any fuel, the agent will not go to the fuel pump unless there is no other option. The agent will select an option and go to that point.

This strategy is also embedded in the algorithm 'Task Execution', where 'try other options' apply this strategy.

How to finish tasks?

Task List Optimization
Input: current task list and locations of all stations and all wells

1. Find the nearest well for each station
2. Put all tasks with the same nearest well in to a list
3. Sort all the lists created by step 2 in descending order based on the number of tasks each list has
4. Add all points in one list to final task list in turn

Task Execution
Input: a list of tasks, agent's current position, fuel level and water level, current cell the agent stands on and a table containing station-well pair where each well is the nearest well to that station

1. If current cell is a station and it has a task T, then set current task to be T
2. Otherwise, if last task stored in the planner is complete, then set current task to be the first task in task list. Otherwise continue last task
3. Get station position based on task at hand
4. Get the nearest well of this station based on the station-well table
5. Get current position of the agent
6. If agent has reached the target station, Yes -> step 7, No -> step 9
7. if it has enough water,
   Yes -> return Deliver, No -> step 8
8. If it can go to a well directly to load water,
   Yes -> return Move to well, No-> try other wells or tasks or exploration points
9. If it has enough water,
   Yes -> step 10, No-> step 11
10. If the agent can go to the station directly
    Yes -> return Move to station, No-> try other tasks or exploration points
11. If it can go to the well directly
    Yes -> return Move to well, No -> try other wells or tasks or exploration points

Before the exploration of the environment is finished, the agent will just 'collect' tasks it has seen. Then after finishing the exploration, tasks collected will be sorted first and then executed.

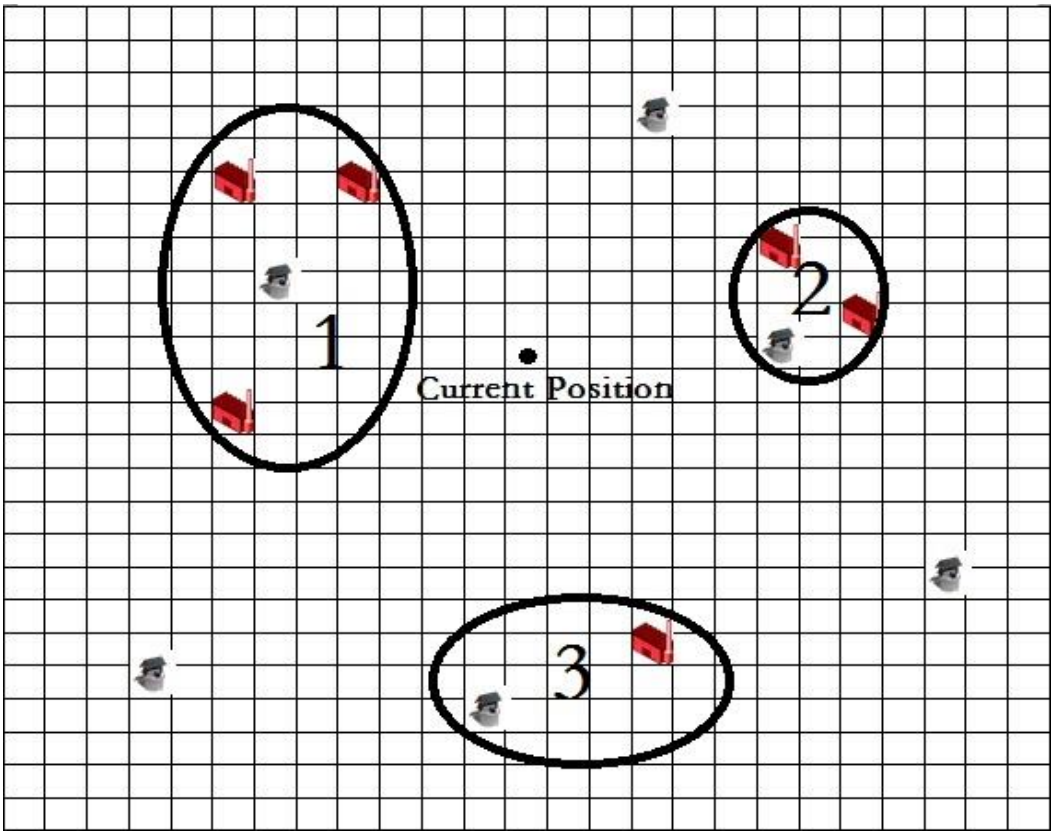


Figure 7 Task List Optimization algorithm

Generally speaking, 'Task List Optimization' algorithm sorts tasks by task density (illustrated by figure 7). Stations with the same nearest well will be grouped. The number of group members (tasks) decides the order they are executed.

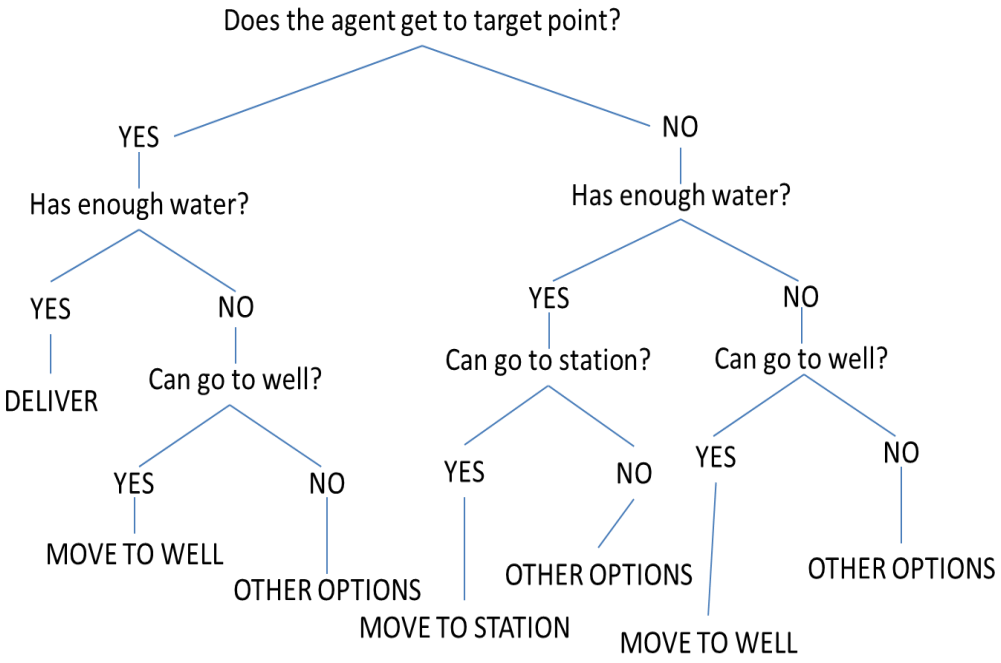


Figure 8 Task Execution algorithm

Figure 8 illustrates the 'Task Execution' algorithm. However, there are two actions not included in the figure, Refuel and Load Water. As long as the agent is on the fuel pump or well, and meanwhile it does not have full fuel or water, refuel or load water. Therefore, these two actions have the highest priority that no matter what condition the agent currently faces, refuel or load water as long as it satisfies the precondition of these two actions.

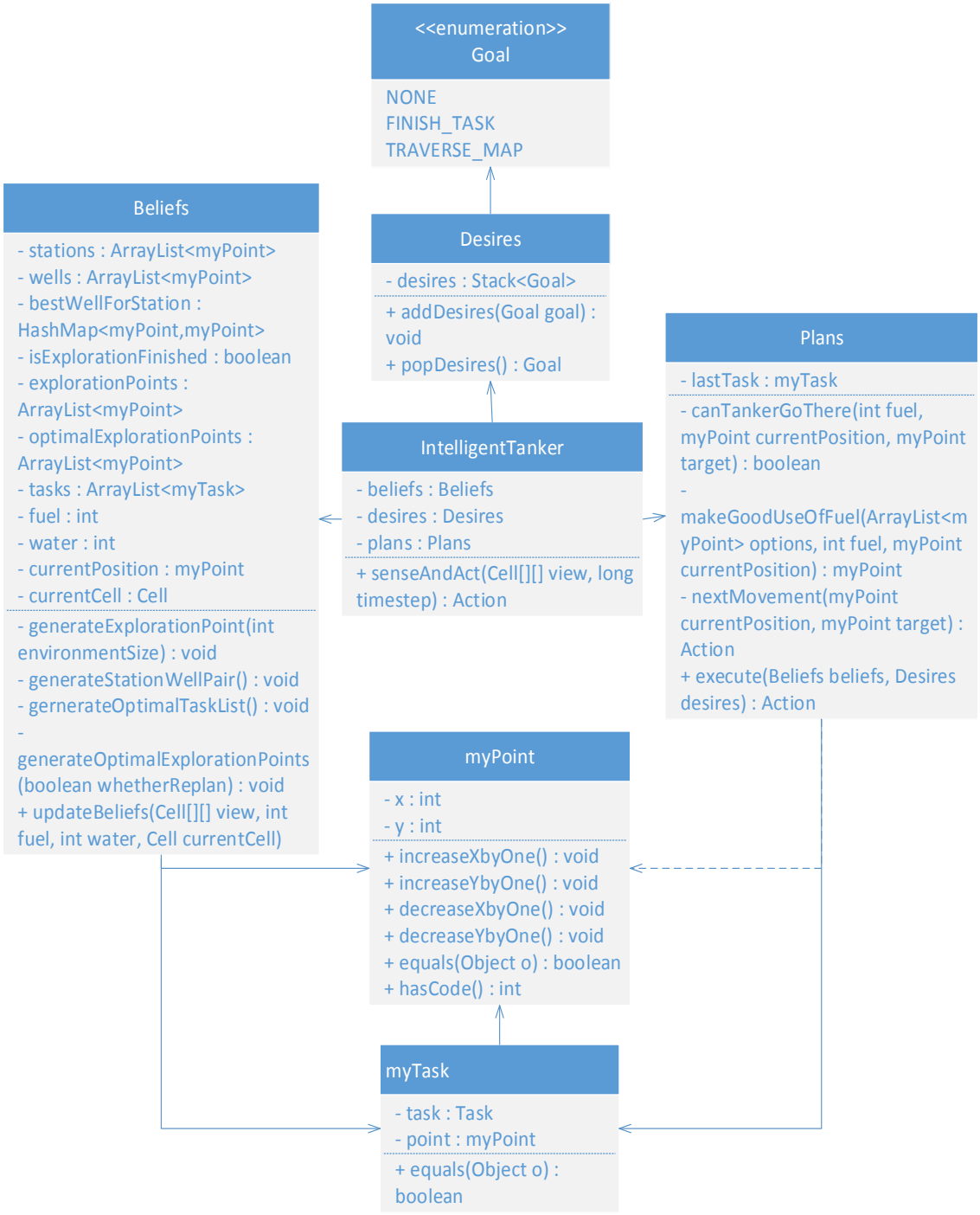
4. Implementation

1. Instance Creation

Tanker t = new IntelligentTanker((Tanker.MAX_FUEL/2)-5);

The agent object needs the environment size as a parameter.

2. Overall Class Diagram



3. Data Structure

There are 4 provided data structures (ArrayList, HashMap, Stack and Enum) and 2 custom data structures (myPoint and myTask) used in this agent system.

- *ArrayList*

  ArrayList is used to store a list of position, for example, a list of stations and wells. It is good for storing something of which the size is unknown.

- *HashMap*

  HashMap is used to store a table recording positions of each station and relevant nearest well. The key of this hash map is the position of the station and value is the position of the well.

- *Stack*

  As introduced above, FINISH_TASK has higher priority than TRAVERSE_MAP. Therefore, a Stack is pretty suitable for this case.

- *Enum*

  Although an enumeration class can simply be replaced by integer, this does improve the readability and elegance of the code.

- *myPoint*

  As x and y element in Point class is inaccessible, similar class which is used to record the position based on odometry is necessary. 'myPoint' class is nearly the same with 'Point' class except operations on x and y are allowed in the former.

  In addition, as this class will be used in HashMap, 'equals' and 'hashCode' functions have to be overridden.

- *myTask*

  The aim of using 'myTask' is the same as 'myPoint' that records where the task is from.

  One tricky point of 'myTask' is that a Task instance after wrap and fetch will not be the same instance as the instance before wrap.

```
Task task = new Task(xxx);
myTask task_wrapper = new myTask(task);
Task task_fetch = task_wrapper.getTask();
System.out.println(task_fetch.equals(task));
-------------------------------------------
False
```

The above code illustrates this case. The reason for this is that after the task instance is wrapped, a new instance of 'Task' class with copied value has been created. Therefore, these two objects do not have the same memory address.

One possible problem this issue will cause is that if the agent only uses 'new DeliverWaterAction(task)' to hand in the task, the instance 'task' is complete but the instance 'task_fetch' is not complete.

4. One tricky problem in percept

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

The above table illustrates what the agent sees in the task environment and each number stands for a cell. The agent lies on Cell 13.

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

The above table illustrates how 'Cell[][] view' represents the environment. Cells of the same row in 'view' actually represent those of the same column in the task environment. Therefore, given the following example (figure 9) and current position of the agent as (x, y), the following equation holds:

View[3][0].getPosition() == (x + 2, y + 1)

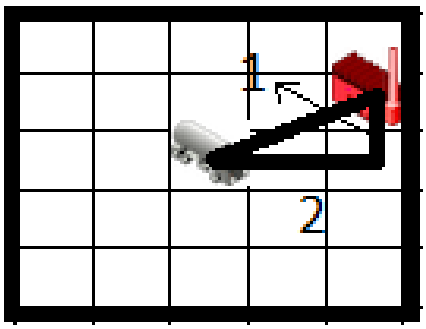


Figure 9 relative position

5. Evaluation

How is Performance?

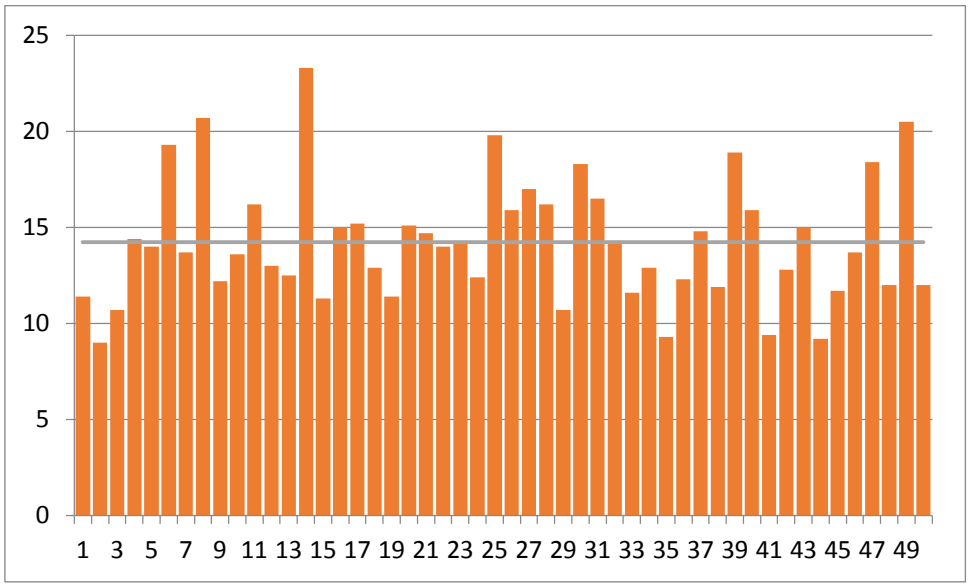


Table 1 Scores of 50 Experiments (unit: billion)

Table 1 illustrates the score distribution of 50 experiments and the average score of these 50 experiments is **14.23 billion**. Detailed data is attached in the appendix.

Generally, there are mainly three parts which improve the performance of the agent: exploration point optimization, task list optimization and option list. To examine the contribution of each of these parts to the overall performance, three simpler versions of the agent which replace relevant part with simple approach will be compared with the original agent.

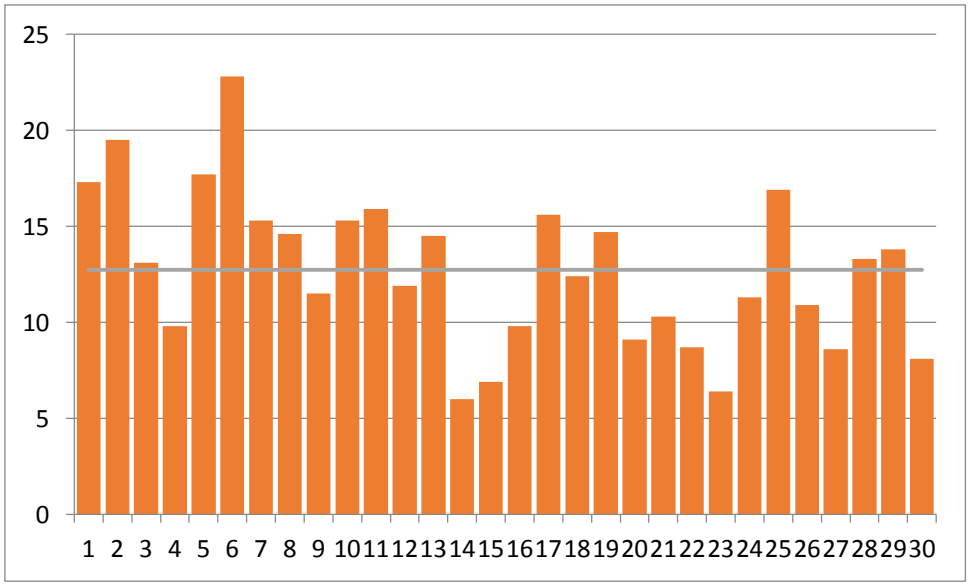


Table 2 Scores achieved without exploration point optimization

Table 2 is the result with the agent that is replaced with a spiral way to explore the environment. The average score of these 30 experiments is **12.73 billion**.

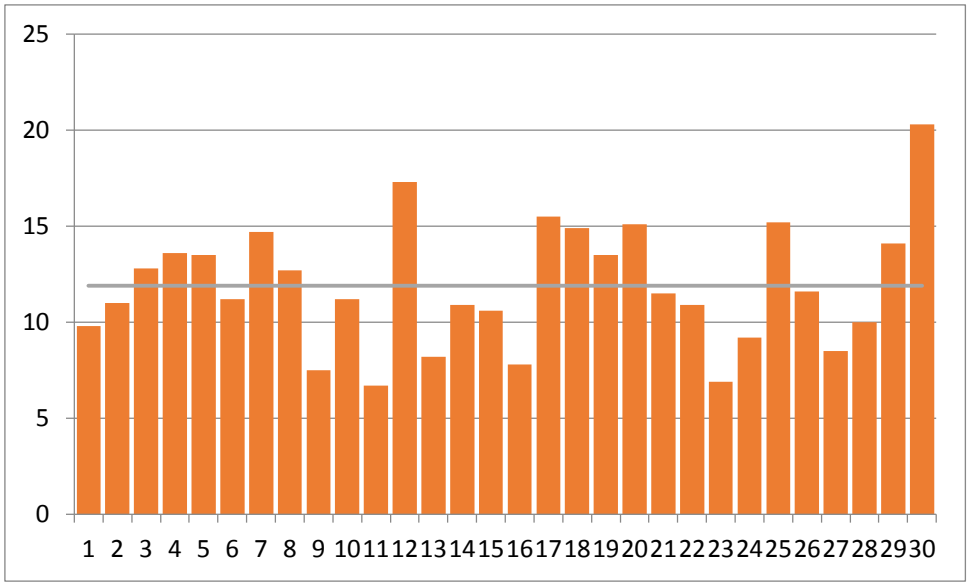


Table 3 Scores achieved without task list optimization

Table 3 illustrates the result with the agent that performs tasks in a 'first come, first serve' manner. The average score of these 30 experiments is **11.89 billion**.

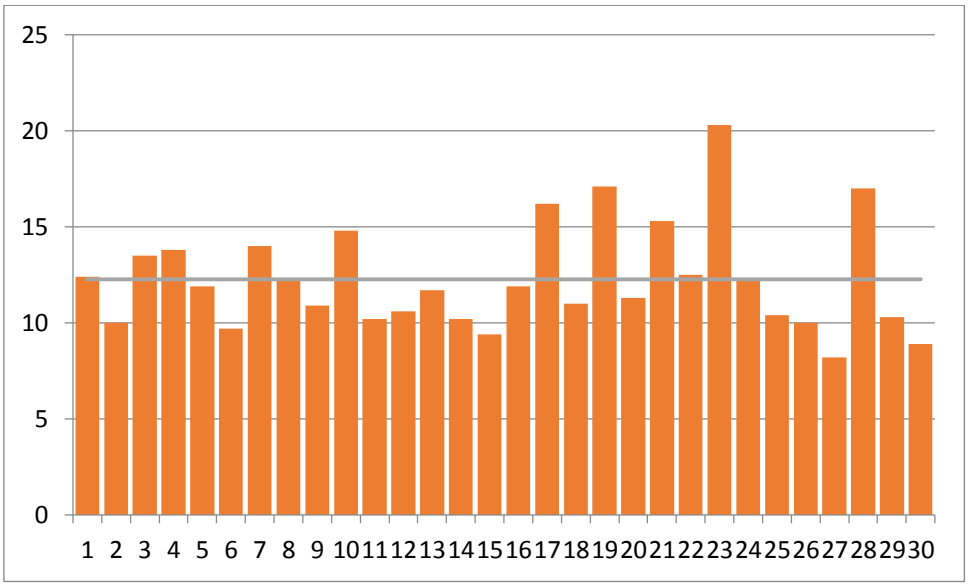


Table 4 Scores achieved without options

Table 4 is the result with the agent that is removed the option selection part. This means that if the agent cannot directly go to a station or a well because of lack of fuel, it will directly head for the fuel pump instead of trying other wells or tasks. The average score of these 30 experiments is **12.27 billion**.

Compared with the overall score, the above three results suggest that these three parts do improve the performance of the agent by working together and the improvement rate is at least **12%**.

Better Performance ≠ Better

Although the performance of the agent designed is good, it is hard to only compare performance of the agent with that of other agents to determine which one is better. As Russell and Norvig pointed out, a rational agent does not mean an omniscient agent and being omniscient is impossible in reality. This works even in this task environment, for example, the agent cannot know which station will generate the best task. Therefore, rationality instead of perfection is the requirement for the agent. And to some extent, this means that as long as an agent does not always make a worst decision, the agent is rational.

Why solution appropriate?

As mentioned above, the task environment given is comparatively simple and is not time-constrained. Therefore, solutions based on deliberation architecture will be good choices which balance the implementation difficulty and performance.

However, if the environment is **totally dynamic**, for example, thinking takes time and environment changes from time to time, then this solution will fail. Too much time spent on thinking and planning will make the final decision out of date and may not be suitable for up-to-date situation.

Moreover, if the environment is **non-deterministic** and **continuous**, this solution will fail as well. The agent works in such environment will have to continuously figure out the actual effect of its action. For example, if it moves one cell to the left, it has to figure out the actual distance it moves by making use of something like landmarks. Otherwise uncertainty about the location accumulates will finally ruin the plan.

6. Conclusion

This report illustrates a BDI model based solution to the water delivery problem under given task environment. Compared to some simpler versions of this agent, it has better performance on average. This solution, however, is limited to environment with some critical features (deterministic, partial dynamic and discrete). Some modifications are needed for dealing with problems under other task environments.

7. Appendix

Index	Score (billion)
1	11.4
2	9
3	10.7
4	14.4
5	14
6	19.3
7	13.7
8	20.7
9	12.2
10	13.6
11	16.2
12	13
13	12.5
14	23.3
15	11.3
16	15
17	15.2
18	12.9
19	11.4
20	15.1
21	14.7
22	14
23	14.3
24	12.4
25	19.8
26	15.9
27	17
28	16.2
29	10.7
30	18.3
31	16.5
32	14.3
33	11.6
34	12.9
35	9.3
36	12.3
37	14.8
38	11.9
39	18.9
40	15.9
41	9.4

42	12.8
43	15
44	9.2
45	11.7
46	13.7
47	18.4
48	12
49	20.5
50	12
Average	14.226

Table 5 Achieved Scores

Index	Score
1	17.3
2	19.5
3	13.1
4	9.8
5	17.7
6	22.8
7	15.3
8	14.6
9	11.5
10	15.3
11	15.9
12	11.9
13	14.5
14	6
15	6.9
16	9.8
17	15.6
18	12.4
19	14.7
20	9.1
21	10.3
22	8.7
23	6.4
24	11.3
25	16.9
26	10.9
27	8.6
28	13.3
29	13.8
30	8.1
Average	12.73

Table 6 Achieved Scores (without exploration point optimization)

Index	Score
1	9.8
2	11
3	12.8
4	13.6
5	13.5
6	11.2
7	14.7
8	12.7
9	7.5
10	11.2
11	6.7
12	17.3
13	8.2
14	10.9
15	10.6
16	7.8
17	15.5
18	14.9
19	13.5
20	15.1
21	11.5
22	10.9
23	6.9
24	9.2
25	15.2
26	11.6
27	8.5
28	10
29	14.1
30	20.3
Average	11.89

Table 7 Achieved Scores (without task list optimization)

Index	Score
1	12.4
2	10
3	13.5
4	13.8
5	11.9
6	9.7
7	14
8	12.2
9	10.9

10	14.8
11	10.2
12	10.6
13	11.7
14	10.2
15	9.4
16	11.9
17	16.2
18	11
19	17.1
20	11.3
21	15.3
22	12.5
23	20.3
24	12.3
25	10.4
26	10
27	8.2
28	17
29	10.3
30	8.9
Average	12.26667

Table 8 Achieved Scores (without options)

8. References

1. Alexander, P., Lars B., Christopher H. and Jan L. 2014. Programming BDI Agents with Pure Java. In: MATES (CONFERENCE), MÜLLER, J. P., WEYRICH, M., & BAZZAN, A. L. C. (2014).*Multiagent system technologies: 12th German Conference, MATES 2014, Stuttgart, Germany, September 23-25, 2014 : proceedings*. Switzerland: Springer, pp. 216-233.

2. BERGENTI, F., GLEIZES, M.-P., & ZAMBONELLI, F. (2004). *Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook*. Boston, [Mass.], Kluwer Academic.

3. LIANG, Y. D. (2001). *Introduction to Java programming*. Upper Saddle River, N.J., Prentice Hall.

4. MATARIĆ, M. J. (2007). *The robotics primer*. Cambridge, Mass, MIT Press.

5. RUSSELL, S. J., & NORVIG, P. (1995). *Artificial intelligence: a modern approach*. Englewood Cliffs, N.J., Prentice Hall.

6. WOOLDRIDGE, M. J. (2002). *An introduction to multiagent systems*. New York, J. Wiley.