G54DIA
Coursework 2 :
Multi-Agent System

Final Report

Adam Weisen Goh

psyawg@nottingham.ac.uk

4235057

Table of Contents

INTRODUCTION

In this report, the task environment is revisited in context of multi-agent environment. The characteristics and design of agents are explained and discussed in relation to the task environment. Then, the design of the multi-agent system is discussed in detail, covering fundamental concepts such as task sharing, control structure, and communication. In support of the design, the implementation of the multi-agent system will be explained and how it complements the design choices. Finally, the performance of the multi-agent system will be evaluated for different number of agents in the environment. Future works will also be discussed and possible issues that may emerge with the complexity of multi-agent system design.

BACKGROUND MATERIALS AND LITERATURE REVIEW

Multi-agent system

In an environment where multiple agents co-exist, a multi-agent system is an agent program of a single agent designed as a collection of autonomous sub-agents (Russell, Norvig, & Intelligence, 1995). This definition will be used throughout the whole report whenever multi-agent system is mentioned.

Intelligent agents

The definition of intelligent agents operating in a task environment varies, but in this report definition of an agent follows Russell and Norvig's definition of an agent program, "a concrete implementation, running on the agent architecture".(Russell et al., 1995)

SPECIFICATION

This section describes the specification of the agents in the environment and important characteristics of a multi-agent system that must be considered and implemented in context of the task environment. The task environment, task specification, constraints, and percepts and actions are identical to what was described in coursework 1. A full description of the standard task environment and specifications can be referred to the Coursework description in the appendices. For recall purpose, the task environment was described to be sequential, discrete, partially observable, dynamic, and deterministic. The task environment is updated to add multi-agent as one of its characteristics.

Project Specification

Agent specification

Looking from a multi-agent system's perspective, agents can be designed in terms of its degree of specialisation and the number of agents available in the environment. Agents may have common capabilities (*totipotent*) or possesses specific skill set (*specialisation*). With a deterministic and discrete environment, agents have limited percepts and actions within the environment. Hence, specialized agents are deemed unnecessary decoupling of agent's ability. For example, limiting an agent's capability to only search the environment for tasks may be wastage of its potential to complete those tasks. With this in mind, all agents are designed to be *totipotent*, having same actions that can be performed in the environment.

This however, will lead to redundancy of agents, where all agents are capable of many actions and each action can be performed by each agent. Despite that, redundancy of totipotent agents can be resolved by having a task-sharing strategy, method of coordination and an appropriate organisation strategy to increase its specialisation and efficiency.

All agents also applied identical hybrid architecture, where tasks planning are still a major importance to completing the tasks in a microscopic view. Hybrid architecture's control system also allows easy implementation of multi-agent system with minimum modification to agent's original ability. Having different architectures between agents are not suitable for totipotent agents that do not embrace specialisation.

Open Multi-Agent System specification

In a partially observable, multi-agent environment, agents face with the dilemma of focusing on its self-interest or cooperate with other agents in terms of multiagent planning. (Russell et al., 1995). This problem is tackled in a distributive manner with the *open system* approach: Agents, although may not be designed to share common goals, are free to enter or leave an open, pre-defined organisation (Zambonelli, Jennings, & Wooldridge, 2001). This method ensures that agents have its own locus of control, while maintaining cooperation

between agents when necessary. Agents that join an open system must adhere to the organisational rules (norm) when completing tasks with other agents in the open systems. Open system in multi-agent systems heavily mimics human organisations in resolving conflicts, competitions and cooperation. Ganguilhem, a French philosopher, described the organisational rules (norms), which is a core concept of open system, as “a possible mode of unifying diversity, resolving a difference, settling a disagreement”. (Canguilhem, 1991) .

An agent should decide if acting on its self-interest or sharing a common goal with other agents will maximises the system’s performance. A common way of solving this is implementing rules that are predefined deciding if an agent should cooperate by joining the open system, or decline and act upon its own interest by leaving or not joining the system. In the case of cooperation between agents, a strategy of sharing available tasks must be made clear and understood in unison between agents. An interaction protocol dictates the way agent treats other agents and how do they coordinate between them to avoid goal conflicts.

DESIGN

Open multi-agent system allows a clear separation between an agent acting on its own self-interest and agents interacting with each other to increase system’s efficiency. The decoupling also allows flexibility in design of the system by assigning organisation rules, structures, and agents interactions between each other. In this section, the Architecture design is described, Organisational rules are explained, and Organisational Structure, Mutual Belief, and Task sharing are explored. Finally, mechanism design using Games Theory explains how organisational rules are defined.

Organisational Design

Architecture Design

The figure below describes the open multi-agent system designed. The outer box encapsulating all agents is an interface that is implemented by all agents, while the inner box is the open system. In the figure below an agent had joined the open system and is within the open system box, while dashed lines between all agents is an acknowledgement of presence of each other in the environment.

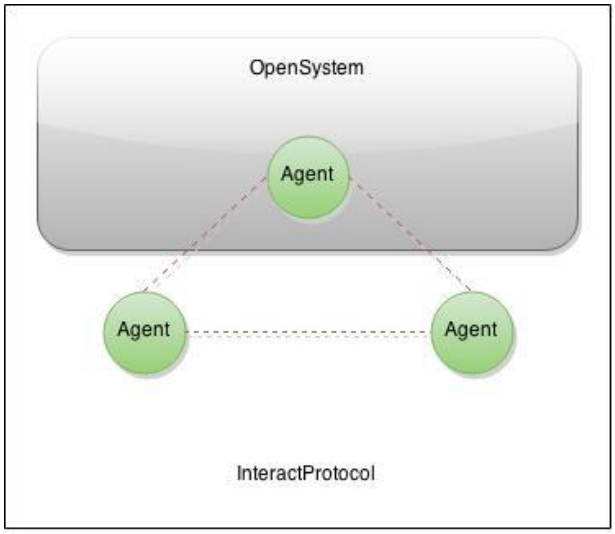


Figure 4.21: An open multi-agent system design

The open multi-agent system can be described by identifying certain types of relationships between agents within it: control, peers, benevolence, dependency, and ownership. These relationships can be identified throughout the explanation of the multi-agent architecture. Agents in the environment do not have direct interaction or control between each other, maintaining a distributive policy of an open system.

However, there is a need in design to avoid competition between agents within the system, between agents outside of the system, and between agent outside of the system and agents inside of the system. To curb this issue, an Interaction Protocol is designed for agent to acknowledge the presence of it and other agents within the environment. Interaction Protocol allows agents that are not interacting via the open system in the environment to share its location for coordination purposes.

Organisational Rules

Before defining the types of behaviours that the agents have within an organisation, a set of constraints that the organisation must respect and execute need to be explicitly specified. Organisational rules maintain control on an organisation's internal coherency within a dynamic environment. Agents acting on its own self-interest may join the organisation, causing conflict of interests between agents in the organisation. Therefore, there is a need to identify global conditions for *1) criteria that the agent must satisfy to enter or leave the system*, and *2) clearly classifying agent's self-interest behaviours and whether any of them must be prevented within the organisation*. (Zambonelli et al., 2001) For example, an agent's planning to complete two given tasks in the most optimal route can be considered as an act of self-interest, but do not generally affect other agent's decisions or their role in the organisation. Hence this self-interest based behaviour can be allowed within the organisation.

With a dynamic task environment, the open multi-agent system is designed in a *distributed fashion* that maintains no control over what the agent chooses to do within the environment. Agents are allowed to leave or join the system at any given timestep under any conditions the agent was predefined to. With such open policy, the organisation must have **control** over the system's traffic to maintain order and its integrity. Hence, the open system was designed to only allow an agent to enter if it accepts any specialized roles that the system assigns it to, and must not abandon its responsibility until its completion. This implementation also resolves redundancy issue that totipotent agents have for being general agents by assigning specific tasks that it must perform under the organisation's influence, reducing agent redundancy.

Organisational rules can be identified to either cover over all roles and protocols in the organisation, or expressing abilities and constraints between agent's role, protocols, or between roles and protocols. (Zambonelli et al., 2001). In the open system designed shown in Figure 4.21, organisation rules must be adhered by all agents within the systems for all roles, and roles do not redefine information sharing among agents via the Interaction Protocol.

Organisational Structure

Organisational structure can be defined as a topology to describe responsibilities within the organisation and their position within the organisation. It also implies a structure of control of the organisation's activities (Zambonelli et al., 2001). In the implemented open multi-agent system, agent within the system that prescribed to different roles does not have any form of **ownership** or relationship with other agents in the system, forming a very flat, distributive organisational structure. Each agent focuses on the role or responsibility that was assigned to it by the organisation without having "opinions" about other agent's roles in the organisation. This allows specialisation as an emergent behaviour within the organisation without any explicit interaction between agents.

Agents that picked up identical roles, however, do not behave with the same structure of control. When acting on the same goal, agents must be able to express and acknowledge the presence of other agents in the organisation with the same responsibility. That is, a hierarchical control structure is put in place to allow communication between agents with identical roles to avoid conflicts and improve coordination. This characterizes an agent being **dependent** on another agent for coordination.

For example, an agent may be foraging on one area of the environment; another agent that picks up the same foraging role has form of interaction with the other agent by becoming its child role and forages on another area of the environment.

In summary, the inter-agent relationships existing in the system are distributive in a macro-level, and hierarchical in a role-level.

Mutual Belief

To achieve coordination among agents in the environment, whether they are within the open multi-agent system or acting on its self-interest, a means of interaction must be put in place to avoid conflicts and to share resources **benevolently** among each other. Agents within the open system share non-conflicting resources such as well locations and the each agent's position, but not each agent's belief state. This is to optimize route that an agent can generate to complete a task by having a larger knowledge of resources such as wells, whilst maintaining a healthy distributive relationship between agents without any invasive interaction towards an agent's belief state.

Although agents within the environment are not allowed to maintain or edit other agents' belief state, each agent always checks for any conflicting belief states. If an agent currently has a task in its belief state, and a check reveals that another agent has the same task in its belief state, the agent will delete its belief of the task's existence from its belief state. The justification for such an action is that *1) Both agents are operating in close proximity, and requires another agent to forfeit the task to explore other areas for tasks*, and *2) to avoid both agents' conflicting self-interest regarding that task.* Note that modifying agent's own belief state does not have any effect on the belief state of other agents.

Task-sharing

Whenever an agent have a large amount of tasks that needs cooperation from other agents, it sends a help request to other agents by joining the open system and the organisation will assign portion of tasks as a responsibility to the agent and another portion to an open role for any other agent to accept by joining the system. All agents are permitted to request for help from the open system, and all agents are permitted to accept the task from the open system, ensuring a fair and equal chances between agent and its **peers**.

The belief state of each agent differs among each other, and any agent that is executing its responsibility within the organisation also has its own unique scope of role's belief state. When in an organisation, an agent abandons acting upon its own belief state and generates action based on the role's belief state. This separation of belief states ensures that there will be no conflict of interest between an agent's belief state and the organisation's belief state.

This approach of task sharing resembles *distributed problem solving* where the problem is divided into smaller sizes that can be solved by other agents whenever necessary. Whenever an agent's belief state is lacking tasks to complete, and there are no available tasks in the open system to be completed, the agent is still allowed to join the open system but will be assigned responsibility to explore the environment for more tasks. Agent is allowed to leave the system with any available tasks that can be completed alone, or request for help from the open system if needed.

Game Theory: Decisions within a Multiple-Agent System

In an open multi-agent system, the conditions for joining or leaving the open system to pick up roles are extremely important and sensitive in the context of optimising the open system's efficiency. To aid the design of these conditions, fundamental aspects of Game Theory can be used for **mechanical design** purposes. Game theory is useful for analysing games with partial observability, the same characteristics that that the task environment possesses, and is often used to construct intelligent multi-agent system that solves complex problems in a distributive manner. (Russell et al., 1995)

Players

Players or agents are those who will be making decisions. In this context, to reduce the complexity of the analysis only two players are involved: agent that shared tasks from its belief state, and agent that may or may not join the system to complete the shared tasks. These players will be referred as Harvester and Recruit throughout the game theory analysis.

Actions

Actions are defined actions that the players can make. Action is closely related to pure strategy, where single action is assigned to a player. The actions defined in this analysis are *1) pick up/share tasks by joining the system*, and *2) Leave or do not join the system to pick up/share tasks*.

Payoff functions

The matrix below shows situations where Harvester and Recruit may perform either of the action. Action ACCEPT refers to sharing tasks by joining the system for Harvester, or accepting shared tasks by joining the system for Recruit. Action REJECT refers to refusing to share tasks by joining the system for Harvester, or leaving or not joining the open system to complete the shared task for Recruit.

	Recruit *(R)* : ACCEPT	Recruit *(R)* : DECLINE
Harvester *(H)* : ACCEPT	H : +1 , R : +1	H : +1 , R : 0
Harvester *(H)* : DECLINE	H : -1 , R : 0	H : -1 , R : -1

Table 4.1: Payoff function for a two player game in open multi-agent system

The direst situation that must be avoided is Harvester refusing to share tasks, and Recruit refusing to accept any tasks (H: -1, R: -1). This will lead to the obsolete of the open system and no cooperation will occur. Situation that is greatly discouraged is Harvester refusing to share tasks and Recruit joins the system to accept shared task, when no tasks were shared by anyone(H: -1, R: 0).

With this, we can infer an optimal strategy, where Harvester always shares tasks and Recruit always picks up tasks. However, Recruit declining to pick up tasks will not be considered as a negative event. This keeps the shared tasks supply in ample quantity while giving Recruit the choice to act on its own interest or cooperate to complete shared tasks.

IMPLEMENTATION

Conflict resolution

To avoid conflicting belief states among each agent, an agent acknowledges the presence of other tankers by joining an interface called *InteractProtocol* that is shared with other agents. The agent is able to access other agent's belief state through an array of agents stored in the interface to perform a check. If there are any conflicting tasks, the task will be removed from the agent's belief state. Besides that, an agent also updates completed tasks in its belief state that may be missed due to other agent completing them through the open system. By ensuring that all agents' belief state has no overlaps and all tasks in the each belief states are incomplete, the conflict for solving the same tasks is resolved.

Organisational Structure

With *InteractProtocol* interface that all agents participated in, the open system can be designed as a static object of the interface to allow all agents a way of accessing it. This allows modularity and encapsulation of the open system object, *OpenSystem*. *OpenSystem* object facilitates open roles that agents may pick up when joining the system, updates each roles' belief state, keeps track of all agent's position in the environment, and maintain a shared array of wells found by agents in the system. Joining the system requires a responsibility to be available in the system. When assigning a responsibility, completing tasks shared by other agents in the system has a higher priority than searching for more tasks.

Agent

OpenSystem keeps track of agents in its system by using the object *Agent*. When an agent joins the system an *Agent* object will be passed to the system and the agent. A*gent* object acts as a "key" for authorization to access the system and contains information given by the system to complete its organisational goals such as its role in the system. *Agent* can be described as an identity object for an agent that separates itself from other agents in the system, maintaining the *flat, distributive organisational structure.*

Role

Role objects define the specification of the responsibility that an agent in the organisation must complete. *Role* may have a "leader" *Role*, where it is able to access the leader Role for information to allow coordination. This is an example of hierarchical control in the organisation structure.

A *Role* also contains the tasks that must be completed. For recall, a Role object is a responsibility that an agent must complete before leaving the open system. To complete a *Role*, all tasks in the *Role's* belief state must be completed and cleared from its belief state. When there are no more tasks in the *Role's* belief state, an agent is allowed to leave the *OpenSystem* by giving the identical *Agent* object back as a source of reference for *OpenSystem* to perform book-keeping. There are two types of Roles extended from the Role object, Harvest and Forage.

Harvest

Harvest is a Role created for the sole purpose of sharing tasks between agents. When an agent requests for help to complete tasks, it creates a Harvest object, inserts tasks that needed help with, and *OpenSystem* posts it up in its array of open *Roles* to be accepted by incoming agents. When an agent joins the *OpenSystem*, the *Harvest* task will be prioritized to be assigned to the agent. When assigned, the agent's tasks belief state will be censored, and its action will be based on the Role's tasks belief state. This is achievable due to hybrid architecture's *Control System*, which adds a layer of control over what the agent may perceive to compute an action.

Harvest object also described a leader Role and child Role relationship. However, the hierarchical relationship is not explored as the task environment does not require the level of complexity in coordination to complete shared tasks.

Forage

Forage object, also an extension of *Role*, dictates the direction of exploration by the agent in the system. *Forage* does not have any tasks that needed completion; neither does it contain any logic for foraging. The *Forage* object is explicitly designed that exploits an agent's ability to perform forage in its Reactive layer (agent applies a hybrid architecture, where simple, low level actions are performed in its Reactive Layer). *Forage* only assigns a direction that the agent can access, and the agent will perform foraging through its own function, but with the Role's given direction. In this case, the agent's Control System does not perform any censoring. However, the Reactive Layer will retrieve the direction information if it detects Forage *Role* in agent's *Agent* object.

In *Forage*, the hierarchical relationship between agents is exploited. That is, a child *Forage* will not explore the same direction with its leader *Forage*. This ensures that the area covered is larger and less overlaps.

Organisational Rules

Organisational Rules are one of the most important features in the open multi-agent system. For each time-step, each agent will perform a "validity check" to determine if it should join the open system because there are no beneficial self-interest based actions to perform. The check is performed in a function called *OpenSysCheck()* in the Control System

of an agent. Performing the check in the control system allows flexibility in what belief state or information does the Planning Layer and Reactive Layer sees, and its modularity allow joining and leaving the *OpenSystem* without affecting any ability of the agent.

In preceding order, an agent will first check if had completed its role if it had joined the open system. If the role had been completed, the agent will leave OpenSystem and Agent object will be destroyed. Then the agent will check if it has any tasks or wells in its belief state. If there isn't any, the agent will join the open system in an attempt to seek help by cooperating with other agents within the system. However, there can be a situation where all agents do not have tasks and wells in its belief states, and there are no available *Harvest Roles* in the *OpenSystem*. This is first satisfied by all agents in the beginning. All agents then joined the organisation, and *OpenSystem* will assign each of the joined agents a *Forage Role*. The first agent will be assigned the leader *Forage* while the subsequent will be a child with its previous agent as the leader. This is an example of **emergent behaviour** where all agents worked together to achieve a common goal: to search for more tasks and wells in different direction. In accordance to joining the open system, the agent is able to retrieve wells from *OpenSystem's* list of wells by presenting its *Agent* object.

Whenever a Harvest Role is present in the OpenSystem while an agent is in its Forage Role, the open system allows the agent to change from Forage to Harvest Role due to priority given to Harvest Role.

After dealing with Forage conditions, the agent then checks its belief state through *RequestHarvest()* to see if there are more than three incomplete tasks that need help from other agents. If there are three or more tasks available, the agent will always share its tasks by performing task-sharing in the form of *Roles*. This and the previous Role changing are both in accordance with the **optimum strategy** inferred through Game Theory analysis, where agent always share tasks if possible, and agent choose to accept shared tasks unless stated otherwise.

One condition that an agent may not opt for optimum strategy and choose to decline completing a *Harvest Role* is when the agent checks if its belief state for available tasks and infer that it does not need external help. If the agent has 1 or two incomplete tasks whilst participating in the open system, the agent will attempt to leave as soon as the Role assigned is completed.

Mutual Belief

All agents in the system share the same belief state of the locations of well. When joining OpenSystem, all agents' belief about locations of wells will merge with the array list of wells seen by all previous agents. Note that leaving the system does not remove these wells'

belief state, as they are considered as information offered for joining the system and completing an organisational goal.

Task-Sharing

Task sharing decisions are mostly made in the agent's side of code. If an agent shares tasks if it had completed its *Role* or not in the open system, and its belief state contains three or more incomplete tasks. The agent will first divide the tasks into two portions, joins the open system and assigns one of the portions as its own Role in the system. The other portion will be a Role inserted into *OpenSystem's* array list of *Roles* available for any other incoming agents to pick up. This is an example of *distributed problem solving* in a multi-agent system.

EVALUATION

Table 6.1, Table 6.2, Table 6.3, and Table 6.4 all shows results from 10 consecutive run with its average score and average score per agent for 2 agents, 3 agents, 4 agents and 5 agents respectively. Screenshots of all runs can be found in the appendices as proof of execution. The highest average score achieved is 33.9 billion with 5 agents, but with only 6.7 billion per agent. The lowest score is 18 billion, but with an average score of 9.4 billion per agent.

2

Run Number	Score	Average Score per Agent
1	19517104050	9758552025
2	20072697942	10036348971
3	18429104688	9214552344
4	17158705215	8579352608
5	15547725765	7773862883
6	17478616216	8739308108
7	10284587682	5142293841
8	28391475119	14195737560
9	22492262960	11246131480
10	19195162892	9597581446
Average	**18856744253**	**9428372126**

Table 6.1: Results from 10 consecutive runs for 2 agents in the environment.

3

Run Number	Score	Average Score per Agent
1	26227064904	8742354968
2	29836137512	9945379171
3	36339628470	12113209490
4	32621602932	10873867644
5	18239342895	6079780965
6	32514563489	10838187830
7	18517048918	6172349639
8	37419637382	12473212461
9	25132716308	8377572103
10	19349296576	6449765525
Average	**27619703939**	**9206567980**

Table 6.2: Results from 10 consecutive runs for 3 agents in the environment.

4

Run Number	Score	Average Score per Agent
1	29767576800	7441894200
2	22547394216	5636848554
3	13464156984	3366039246
4	33473780879	8368445220
5	23902160565	5975540141
6	23715483000	5928870750
7	25873763350	6468440838
8	25393500921	6348375230
9	33800862078	8450215520
10	27772866114	6943216529
Average	**25971154491**	**6492788623**

Table 6.3: Results from 10 consecutive runs for 4 agents in the environment.

5

Run Number	Score	Average Score per Agent
1	27143160660	5428632132
2	25258340820	5051668164
3	41575241730	8315048346
4	41845945520	8369189104
5	26905123895	5381024779
6	26378119138	5275623828
7	29761942590	5952388518
8	41864231815	8372846363
9	39753519120	7950703824
10	38892938964	7778587793
Average	**33937856425**	**6787571285**

Table 6.4: Results from 10 consecutive runs for 5 agents in the environment.

DISCUSSION AND CONCLUSIONS

From the given tables in evaluation, it shows that adding more agents in the environment does not translate to better individual agent performance. In fact, the number of agent is inversely proportional to the average score of single agent: The more agent in the environment, the lower the score of individual agents. This can be argued that having more agents in the environment meant more overlapping areas and tasks, hence creating more competition and increasing redundancy. However, in the angle of multi-agent system, agents that cooperate with each other increases the total score that the system can achieve by having more agents in it to execute more tasks that a single agent is not able to achieve. A better direction to improvise the performance of multi-agent system is to incorporate better cooperation and reduce competition between agents.

Future Improvements

With the modularity and distributive manner of open multi-agent systems, the system can be expanded to suffice more complicated environment. That is, more open systems with different organisational rules and structures can be constructed, and agents can choose which organisation it should join to maximise the system's performance. Having multiple organisations with different structures and rules means that these organisations must adhere to a global rule that complements the task environment to avoid any possible conflict of interest between organisations.

REFERENCES

Canguilhem, G. (1991). The normal and the pathological.

Russell, S., Norvig, P., & Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs, 25*.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). *Organisational abstractions for the analysis and design of multi-agent systems.* Paper presented at the Agent-oriented software engineering.

APPENDICES

Appendices 1.1:Task specification

The task environment contains certain goals that the agent must aim to achieve. Russell and Norvig described task environments as “problems” to which rational agents are the “solutions”. Russell et al. (1995)Tasks or goals can be considered as a performance measure for how well the agents are doing in the specified environment. Different types of goals have different causes and effects towards the environment. In our environment, there are several goals with constrains on how to achieve them. An example of a task that an agent is expected to complete is delivering water without running out of fuel within a given number of timestep. . The agent needs to complete “tasks” from available stations with limited resources like agent’s water level and fuel level. Such tasks are called an *achievement goal*, where the agent is needed to complete “tasks” to gain score.

Appendices 1.2:Environment specification

The task environment also displays certain characteristics that should be taken into consideration when designing the agent. The project’s task environment is sequential, discrete, partially observable, dynamic, and deterministic. A sequential environment means that the current actions will affect future decisions. For example, choosing to refill water level or not will affects future decisions to complete a certain task or not. The environment is discrete as it shows finite clear defined states with fixed grid of cells. The position of available tasks, wells, stations, fuel pump, and tanker are at a fixed value and does not change throughout each time-step.

As the agent is only able to perceive vision of the environment within a limited range of 25x25, the environment is only partially observable to the agent. The task environment is also dynamic, as new stations or wells may appear upon discovery, and new tasks may be available within the vision of the agent. These dynamic events are also not under direct influence from agent’s action. However, if an agent’s action always produces guaranteed expected output, we can say that the environment is deterministic, because there is no probability of the action producing an unexpected output.

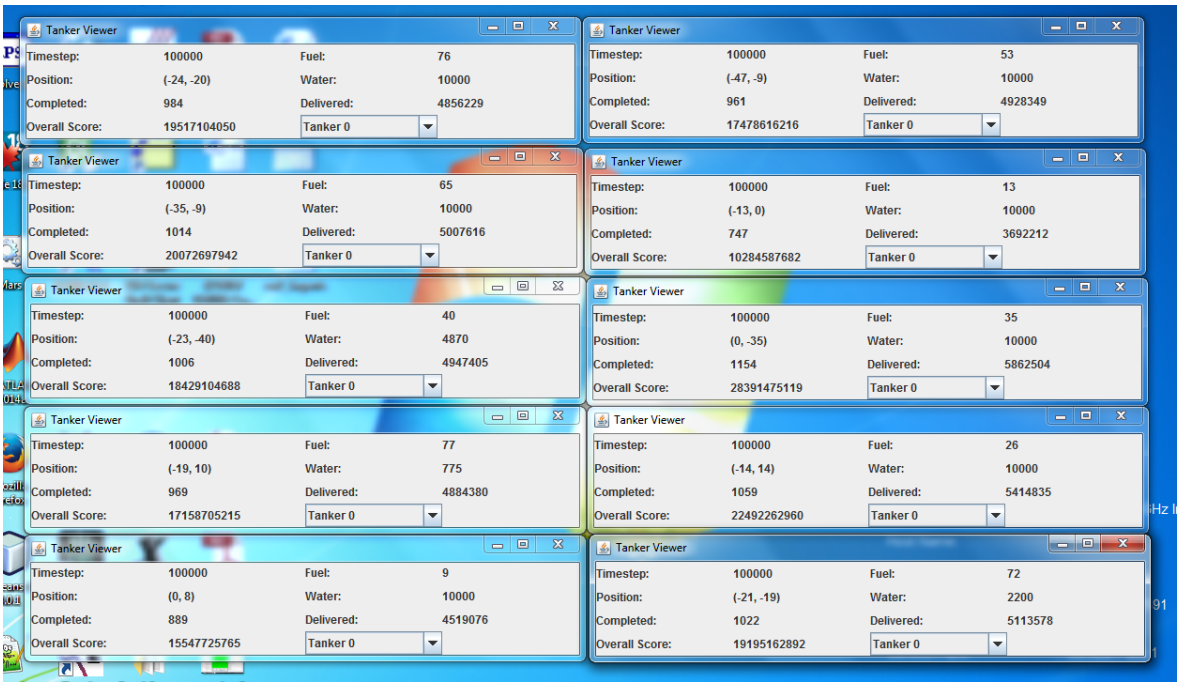
Appendices 1.3:Percepts and Actions

There are several percepts and actions that allow the agent to communicate or to interact with the environment. The agent is able to retrieve basic information of objects such as wells, stations, or available tasks that are within its visions. Although the agent has access to the objects within its vision, it is not able to access any perceivable object’s absolute position in the environment. The agent also keeps track of its fuel and water level. There are also allowed actions performed by the agent that can change the state of the environment or the agent’s state in the environment.

The agent can *REFUEL*, restoring its fuel level to the maximum fuel level, or LOADWATER, which in turn restores the water level to the maximum water level. Being part of the task of the environment, the agent must be allowed an action that complete available tasks. This can be done by DELIVERWATER, completing the tasks if the agent has enough water. To allow the agent to navigate about the environment, the MOVETOWARDS and MOVE actions help with updating agent's position in the environment.

Appendices 1.4:Constraints

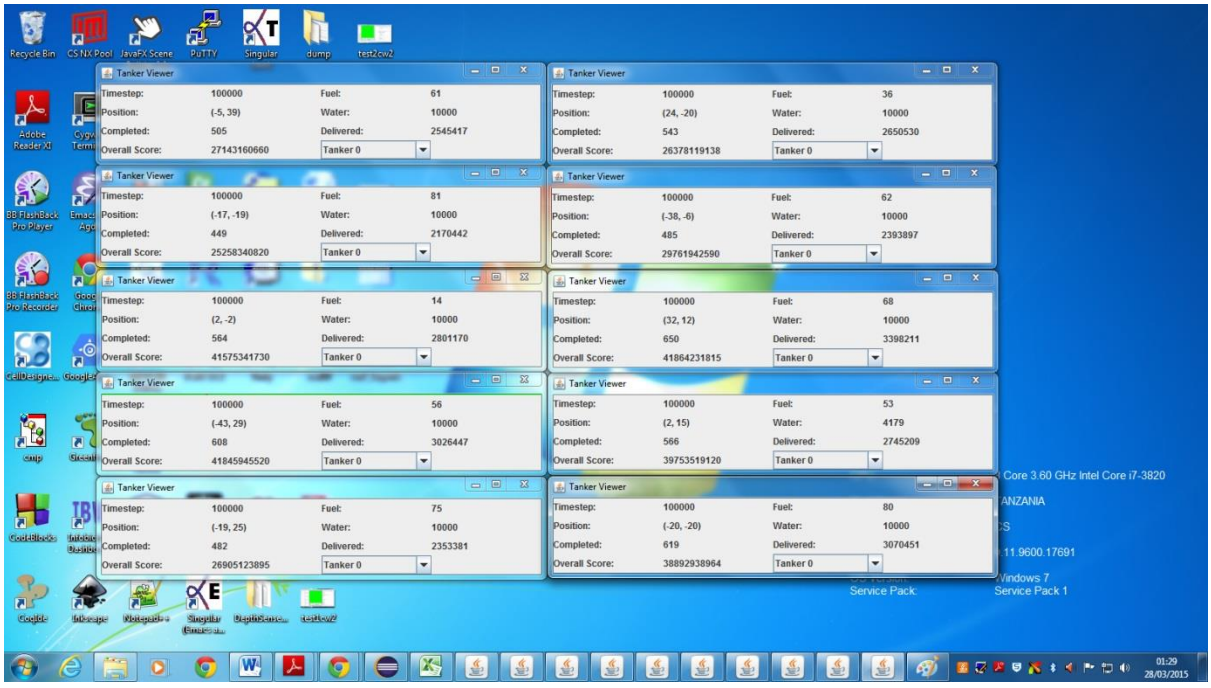
There are constraints that limits the agent's behaviour in the given task environment. The tanker is limited to perceive only items within its vision range of 25x25 cells. To perceive any further would need exploration of the environment. This implies that the view of the environment is processed each timestep that the tanker had moved in the environment. Besides that, the tanker must be able to sustain its fuel level throughout the whole process. This means going back to the fuel pump for replenish its fuel. With that in mind, the tanker can only move 50 cells away from the fuel pump. Combining this knowledge with the agent's partial vision, the available space is only 50 cells in radius with vision of 62 in radius.



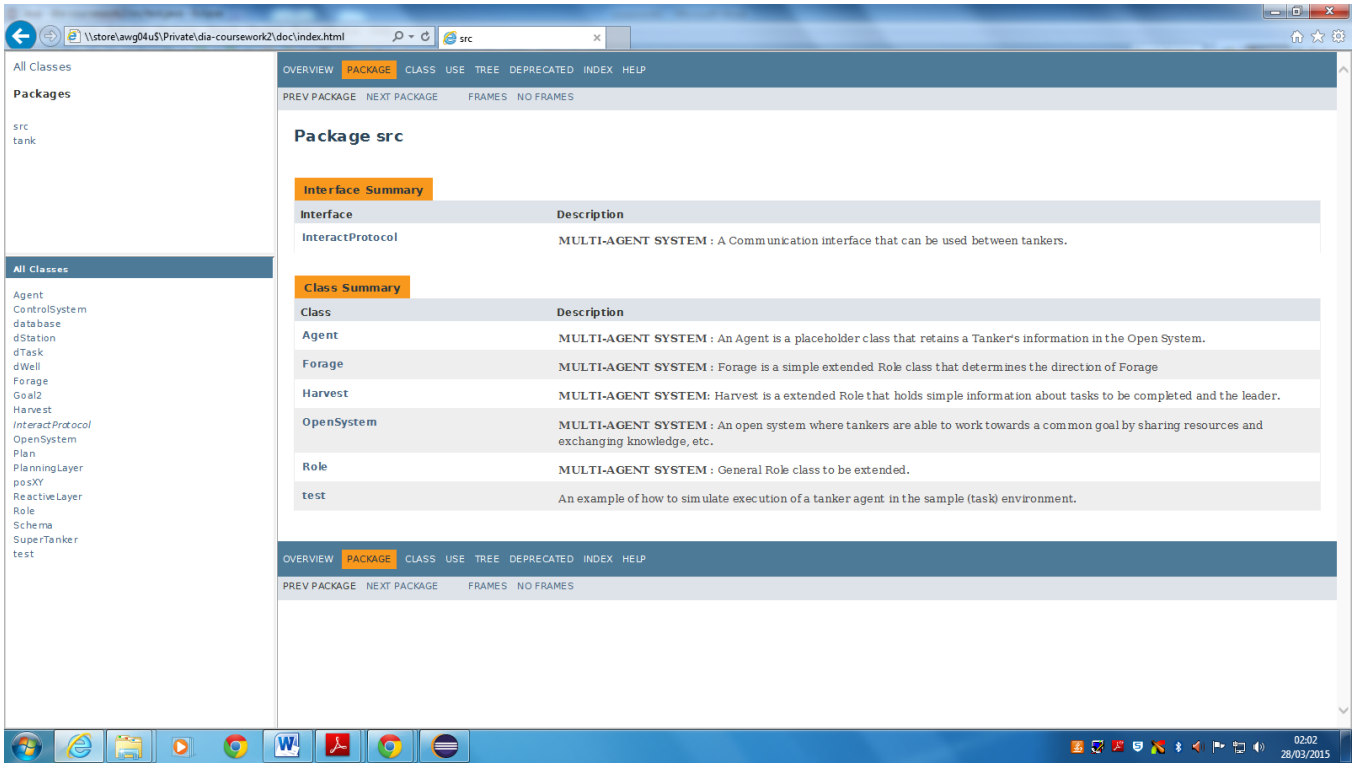
Appendices 1.5: Screenshot for 10 consecutive runs with 2 agents

Appendices 1.6: Screenshot for 10 consecutive runs with 3 agents

Appendices 1.7: Screenshot for 10 consecutive runs with 4 agents



Appendices 1.8: Screenshot for 10 consecutive runs with 5 agents



Appendices 1.9: Screenshot of Javadoc for the multi-agent system package. For full documentation, please access doc/index.html