G54DIA

Final Report

Adam Weisen Goh

psyawg@nottingham.ac.uk

4235057

Table of Contents

INTRODUCTION

In this report, the properties of the given task environment is explored and explained. Agent architecture design in regards to the given task environment is also discussed with each component being reason about the environment's characteristics. The design of each component in the agent architecture is explained in details along with the algorithms used. The implementation methodologies shows how these design are applied in the actual code and finally, the results are shown with evaluation on its performances and finally discussion about some of the problems that were encountered throughout the project.

BACKGROUND MATERIALS AND LITERATURE REVIEW

Backward chaining

Backward-chaining is an algorithm that works backwards from the goal, chaining through rules to find known facts that will support the proof. The Figure 1.1 shows the pseudo code of a simple backward chaining algorithm and Figure 1.2 shows an example of an action schema.

```
BACKWARD-CHAINING(GOAL)

if GOAL is satisfied by the current state then
      return true
end if

for every action schema R with an effect of satisfying GOAL do
      if for all Preconditions of action schema R,
            BACKWARD-CHAINING(R) = true then
                  return true
      end if
end for

Return false
```

Figure 1.1 *: Backward-chaining algorithm pseudocode*

```
Action schema for DELIVER

Pre-conds :
      Goal(HAVEWATER)
      Goal(ATTASK)

Action :
      LoadWaterAction();

Effects:
      Goal(COMPLETETASK)
```

Diagram 4.1 *: An example of action schema*

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, Norvig, & Intelligence, 1995)

TouringMachines

TouringMachines uses a multi-layered integrated architecture that manages features and behaviours that are sophisticated or simplified in a concurrent manner that allows it to simultaneously behave robustly with flexibility in a dynamic, complex, real-world domain. To do so, it should exhibit such behaviours:

1) Reactive to deal with urgent tight situations with limited resources to consider
2) Capable of planning from a start location to a target location, goal oriented.
3) Capable of reasoning about events that take place in the environment and what effects it could have on its own goals, as well as predicting near future to be better informed for subsequent actions.

SPECIFICATION

This section describes the specifications of the project in details. It describes the environments' characteristics, available percepts and legal actions, as well as constraints of the environment. A full description of the standard task environment can be referred to the Coursework description in the appendices.

Project Specification

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.

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.

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.

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.

DESIGN

This section describes how the agent was designed to suit the given environment. The chosen agent architecture and its significance is explored and explained in details. Each components of the architecture are discussed as well as the algorithms that determine the behaviour of the agent in the task environment. Diagrams and pseudo-codes will be used for simplicity and readability, emphasizing the agent's workflow and underlying the methodologies used by the agent to perform tasks in the environment.

Agent Architecture

With any intelligent agent, the provided percepts are only meaningful if it produces agent behaviours. Agent behaviour, as quoted to mean, "the action that is performed after any given sequence of percepts". (Russell et al., 1995) These high level behaviours are only achievable by having an underlying agent function mapping the percepts into a given output action. In that sense, the program that controls this mapping activity can be called architecture, and the kind of behaviours produced is dependent on its design.

The design of the architecture must suit the characteristics of the environment correctly to achieve optimal behaviour. In regards to the project's task environment being partially observable, the architecture must be able to receive the percepts and in its way keep track of the states of the environment. The agent must also consider the environment being dynamic, and able to present the best action with regards to any changes perceived from its percepts. For example, if a new task appears when the agent is attempting to achieve a task, it should be able to consider if abandoning its current task will benefit or sticking to the current task will be a better solution.

Hybrid Architecture

The agent architecture designed for the given task environment is a hybrid architecture design. Hybrid architecture combines the perspective of both reactive approach and deliberative approach into a single architecture. It understands how crucial it is for swift, urgent responses without any complex reasoning or decision making as well as the need to plan ahead in abstract using symbolic representation to produce the optimal action with the given percepts. In the context of the given task environment, hybrid architecture is able to react purely on the urgency of refuelling (*REFUEL*) and getting water (*LOADWATER*), while generating plans to complete tasks that the agent perceived. However, for hybrid architectures to perform well, the characteristics of the environment must be understood properly to apply the correct condition-action rule or aiming for reasonable goals.

For recall, the environment's characteristics are sequential, discrete, partially observable, dynamic and deterministic. The deliberative approach is well suited for a discrete, deterministic and sequential environment. A discrete environment means an abstract internal representation of the environment can be constructed and stored for planning. With the constructed representation, the agent is able to plan ahead the actions to be performed, expects its effects and keeping track the state of the environment with those effects. This is possible because of a deterministic and sequential environment. On the flip side, the reactive approach is well suited for partially observable, dynamic environments where quick responses from the agent have the highest priority. In the context of the given environment, time is not the biggest issue. The means to explore the partially observable environment proves vital to the performance of the agent in some situations that will be further discussed below. A dynamic environment also means getting to the nearest well have different references with each time-step.

Architecture Design

The hybrid architecture that was implemented draws inspiration and has striking similarities from the **Ferguson's "*TouringMachines*"**. Ferguson described how TouringMachines architecture "combines capabilities for producing a range of deliberative and reactive behaviours in dynamic, unpredictable domains." (Ferguson, 1992) Although the unpredictable part of the environment is not in the given task environment, the architecture still provides a solid foundation for storing internal representation state, combining reactive and deliberative approach, along with a control system that manages the percepts and actions of the agent. Figure 1.0 shows Ferguson's original architecture design diagram, and Figure 1.1 shows the hybrid architecture that was implemented.

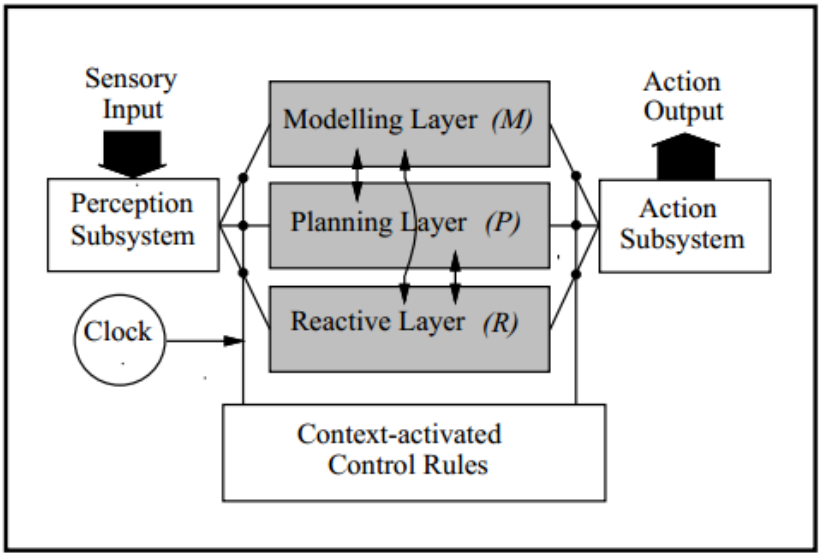


Diagram 4.1 : Ferguson's "TouringMachines" architecture diagram

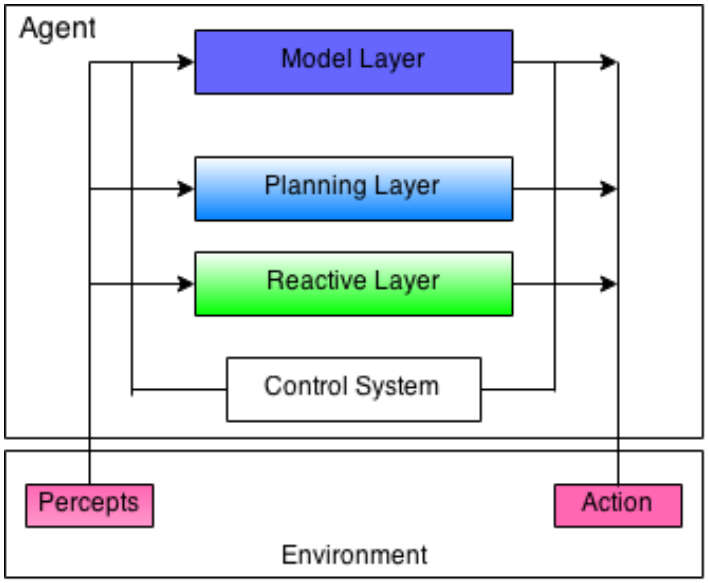


Diagram 4.2 : the Implemented hybrid architecture diagram

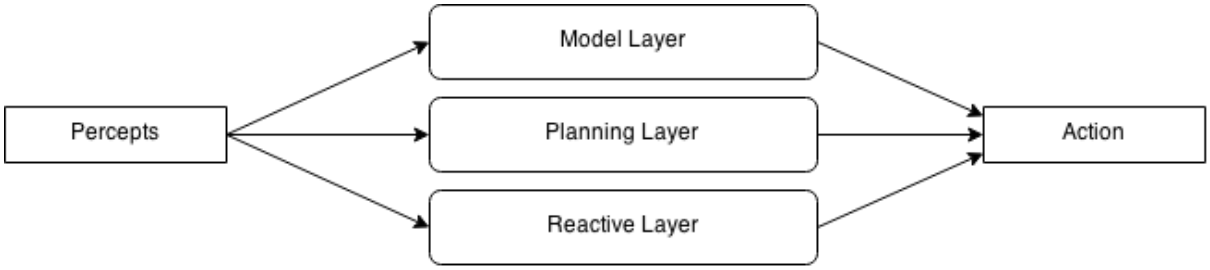


Diagram 4.3 : Horizontal layering, where all layers are connected to percepts and action

The architecture **has three layers of subsystems: the reactive layer, the planning layer, and the modelling layer**. The architecture implements *horizontal layering*, where all layers are connected to the percepts and action output. Diagram 2.1 above shows an example of horizontal layering. However, in this design of hybrid architecture, similar to the TouringMachines' architecture, a *Control System* layer is put in place in between layers as a link that controls what percepts the layer can receive and chooses what the outputs of the agent are.

Control System

The Control System implemented uses explicit control rules and hierarchical arrangements to control the percepts and chosen output. Ferguson implemented the Control System, quoted, "as filter between agent's sensors and its internal layers" (Ferguson, 1992). The *TouringMachines* run in synchronous fashion and at times requires immediate responses for certain circumstances such as avoiding an oncoming object instead

of planning an optimal route to dodge the object. Being able to censor percepts allows the agent to provide action outputs that are much more appropriate in a specific situation, which may be described by the control rules.

The control system is able to censor percepts from layers because all layers can only access the gathered information via the control system. Although the implementation of control system should be of **hierarchical control** just like the TouringMachines, as a single layer decisively controls the percepts and chooses the output of the agent, the implemented architecture has each layer working independently without any communications between them. This allow each layer to work, in a **decentralised control** manner, without interfering each other produce an output which, ultimately, will be chosen by the Control System.

The control system also chooses the final action output that will be executed. Control rules specify specific conditions that a certain output of the layer should be prioritised before the other. This partially relates to the sequential characteristics of the environment, as the action chosen at a given time may lead to agent's failure if not chosen correctly for a certain specific condition.

Model Layer

The model layer keeps an internal representation of the current state of the environment. It keeps track of all objects discovered along with its calculated positions, either relative to the agent's position or its absolute position. The model layer **can be accessed by both the Reactive Layer and Deliberative Layer via the Control System**. However, to reduce the complexity of the Control System, the agent is able to directly access it in order to store any discovered objects in its vision. The agent does not need to go through the control system to store any discovered objects, as no censoring are done towards storing the representation of the current state of the environment. This can be argued that the Model layer should be kept as close and accurate to the actual environment as possible.

It is important that the internal representation is constantly updated each timestep, with the chosen output in a deterministic task environment. If the model is not updated correctly to represent the environment, the agent will not perform as well and as expected. The model layer is also responsible for **updating the representation every timestep**. For each timestep, the model layer must update the position, relative to tanker or absolute, of all stored representation of objects. Any completed tasks are also removed from the internal representation as it does not exist in the actual environment anymore.

Reactive Layer

The Reactive Layer deals with lower level goals that are related with constraints of the environment. These constraints are vital to any planning or higher level goals, but will complicate plans and may not be as effective if implemented directly with Deliberative

Layer. The behaviours from Reactive Layer are Foraging, going to well, refilling water, going to fuel pump, and refuelling.

Foraging is one of the most important behaviour of the agent in a partially observable environment. As tasks only appear when stations are within the vision of the agent, **an effective foraging means better discovery rate for available tasks**. The implemented forage technique emphasises on discovering as far out as possible. This maximises not just the amount of available tasks, but also the number of wells. This is possible as the environment spawns more stations and wells up to a maximum density level. Having many wells are helpful to reduce distances to well for refill.

Reactive layer does not concern itself with any goals that are related to tasks, but providing only fine-grained actions that are essential to the agent such as going to well to refill or going to fuel pump to refuel. These behaviours are compulsory to have as an agent in the given task environment, yet aren't relatively as important to the deliberative layer.

Deliberative Layer

The deliberative layer of the architecture creates a plan for the agent to pursue. A plan consists of several sub-goals that should be achieved. **Each sub goal is a backward-chained list of actions that results in completing the goal.** Backward-chaining a sub-goal deals with the complexity of having water and fuel constraints by grouping the list of actions needed to complete a task as preconditions.

The generated plan will be a set of sub-goals to the available tasks that the agent should complete in an ascending order with minimum fuel. It uses the internal representation of objects from the model layer to create a plan. As each sub-goals in a plan may contain many actions, the total distance of the plan are calculated each "future time-step" with the list of actions. In the calculation process, constraints such as water level and fuel level are projected to get the total distance of the plan.

Deliberative approach to the agent architecture is often scrutinized for the amount of time needed to produce an optimal plan due to its computational complexity. However, in the context of the given task environment **time will not be an issue as time-steps are not used up until an action is actually performed**. This gives the agent an infinite amount of time to generate the best plan without concerning about time.

When new available tasks are discovered, the deliberative layer makes a decision to consider adding it to the plan or not. Similar to a utility-based agent, the deliberative layer will generate a new plan with the new tasks included in it, and using a utility function measures its score. The score allows the agent to compare current and newly generated plan, and decide if switching to a new plan will be beneficial.

The reactive layer, as well as the deliberative layer, always refills water whenever possible. This is because the deliberative layer does not concern its plan with scores but distance, having to refill water opportunistically allows better chance to complete big tasks whenever necessary.

Algorithms

Foraging

The agent's foraging pattern follows a "fan" pattern as shown in Figure 2.3. It travels in diagonal shapes towards each four directions and ends up back at fuel pump to refuel. The pattern chosen for foraging intends to go as far as possible with the biggest discovery area. Comparing it to other alternatives such as spiralling and "bee movement", both shown as Figure 2.4 and Figure 2.5, the fan pattern is able to achieve the furthest distance, while achieving reasonably well discovery areas.

Spiralling suffers from the waste of fuel and time-steps to discover a certain radius size around the centre of the forage. It is ideal if exploring a cluster of stations waiting to

spawn new tasks is the priority. “Bee movement” forage is theoretically similar to the “fan” foraging pattern, but the initial directions of the forage are more focused and emphasized that their adjacent directions. That is, “bee movement” may focuses first on spreading far east and west instead of north and south, leading to possible unexplored area of the directions in the lower priority when the initial directions suffice.

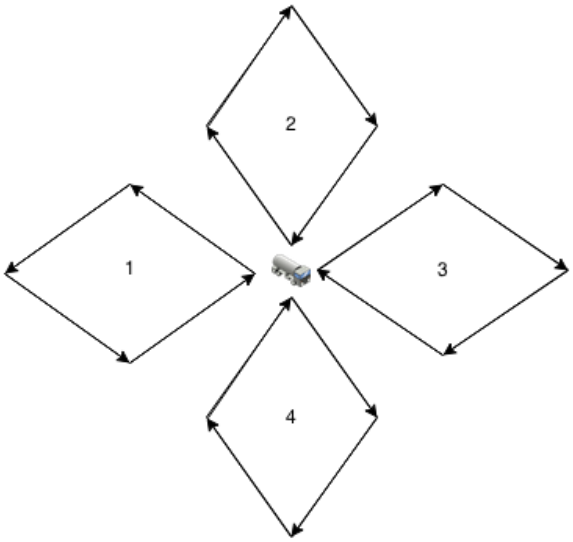


Figure 2.3 : Forage : the “Fan” pattern with its discovery area

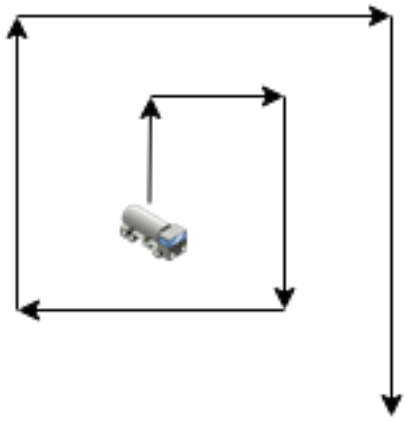


Figure 2.4 : Forage : the “Spiralling” pattern with its discovery area

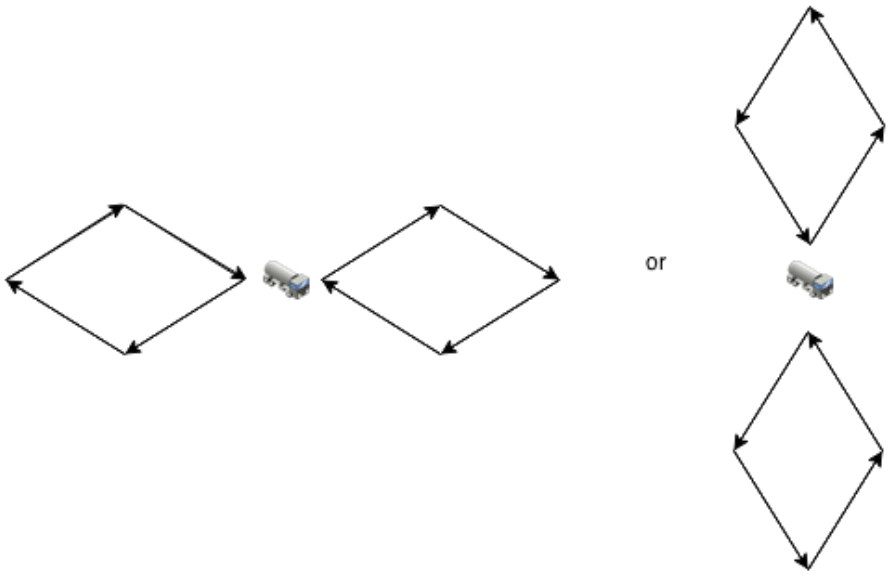


Figure 2.5 : Forage : the “Bee Movement” pattern with its discovery area

Backward-Chaining

The backward-chaining approach works from the intended goal, searching through actions that will satisfy the intended goal. If the actions have preconditions that are not satisfied, the precondition goals are then backward chained to get the actions to achieve them. The result will be a list of actions to satisfy the intended goal. Having known the tasks that should be completed, it is easier to backward-chain to the current state than forward-chain to an unknown goal. Having a deterministic environment means backward-chaining is possible and will not fail to achieve a goal due to probability. It also means the actions can be predefined and the number of actions available can be known. For better and deeper understanding of what is backward chaining and how does it work, please refer to the background materials at the start of the document.

Plan Generation

Given a list of available tasks, deliberative layer must be able to generate a plan that the agent should follow to achieve optimal solution. The list of tasks received from the model will be used to create a plan. Each task is ranked by calculating the total distance covered in order to achieve the task with the given water and fuel level. The distances to refuel and refill water are inclusive in calculating the distance. Once a optimum task is found, it is added to the plan. Then, the next task to be completed will be recalculated and ranked with a given estimated fuel level and water level after completing the chosen task. The result will be a plan similar to a greedy search. While greedy search is not complete to find the shortest path, A better path cannot be found with better heuristics such as A* algorithm

due to the environment being partially observable. With regards, the deliberative aims to exploit the environment by minimising time-steps needed to complete tasks with the given percepts.

Plan Comparing

For each timestep, a new plan will be generated. The plans are then compared for equivalence and optimality. If the current plan and the new plan do not agree, a utility function is used to measure its potential in order for the deliberative layer to decide to change plan. The utility function used for plan comparing is total score of the plan. A new plan with higher score, albeit needing more time-steps, will be prioritized to execute if it passes a certain utility function's threshold. Plan comparing is important in any deliberative approach because it is decisive if an agent should stick to the current plan or switch to a new plan based on its priority.

IMPLEMENTATION

Control System

Percept censoring was designed using fixed control rules, similar to condition-action rules, to determine if certain percepts should be censored. In the context of the given task environment, the planning layer is censored from retrieving an available task from the model layer that was detected if there wasn't any well discovered yet. If there are any discovered wells or available tasks that are beyond the agent's reach due to the fuel constraint (50 cells away), these percepts will be censored from the Planning Layer or the Reactive layer as well. If there are no wells available from the available tasks, the nearest well from the fuel pump will be chosen.

When choosing action output, the control system will always prioritize to follow the deliberative layer's plan unless overwritten by the control rules. Only one control rule was implemented, that is when remaining fuel equates the distance to fuel pump, then the reactive layer's action will be taken instead of the planning layer's action. This ensure that the agent will never run out of fuel. Once an action is decided as the final output, the Model layer will be updated with the given direction of the agent heading to, and all internal representations' positions will be updated.

Model Layer

To keep extra information such as relative and absolute position about perceived percepts which the agent calculated itself, custom classes are used to store these objects. Each object's custom class has functions to update its relative position, and calculate its relative and absolute position when created. These custom classes are stored in an ArrayList that and able to interact with other layers accessor and mutators. Control systems then uses

these accessors and imitate their functions, then adding an extra censoring layer before returning the object to the reactive or deliberative layer

Reactive Layer

Reactive layer uses subsumption architecture to choose which set of behaviour should be activated as action output. As any other subsumption architecture, a hierarchical control between layers is implemented. Behaviour with higher priority, such as refuelling, will be chosen over behaviour with lower higher priority, such as foraging. Despite that, subsumption architecture implemented here does not uses parallel processes, but rather a simple control variable that switches between layers. This is seen at the *layer_state* variable.

Foraging in the reactive layer uses a modulo method to determine if the agent should change direction after a number of steps. That is, if the number of steps mod the set number of steps is 0, the agent changes to the next direction. The patterns are predefined and, when the agent reaches the end of the movement of a certain shape, it changes to the next set of pattern. This repeats itself by constantly changing values and resetting the steps value.

Deliberative Layer

The deliberative layer has one of the more complex implementation of the whole agent architecture. The implementation of backward chaining requires custom classes that support action schemas, goals, plan generation, and plan comparing.

Goal and Schema

A goal is a custom class with a label on what kind of action that the goal should produce with that specific action that goes along with it. The implementation of backward chaining requires knowing what kind of goal that that it should look for. An example of a goal is ATWELL. A goal is able to check if an object can satisfy the specified goal, with type safe checking implemented with inheritance using *instanceof*. Goal checks is important for backward chaining to know if a goal had been satisfied or should be tracked backwards to find appropriate actions that satisfy the goal.

A schema, as detailed like the goal schemas in backward-chaining algorithm, has a list of precondition goals and effect goals. These schemas are predefined and may grow with the number of actions available in the task environment. When backward chaining a goal, the object targeted is first checked if it satisfies the high level goal. If it is not satisfied, the algorithm then looks for a schema that has the goal as an effect and adds its action as the final output. Backward chaining then looks for the precondition goals from the schema recursively and check if their goals are satisfied. If there are any goals that are not satisfied, its action will be prioritized and chosen as the final output. Diagram 2.1 shows the dataflow of the backward chaining algorithm for clearer explanation, while Figure 2.1 shows an example of a schema and a goal.

Plan Generation

A Plan class then builds on the notion of goals being able to be chained as a set of actions, and produces a list of goals that the agent should execute in a sequential manner. A Plan is an *ArrayList* of goals that are sorted in an ascending order based on its total distances needed. A method call *calcATask* takes a task, perceived water and fuel level, as well as perceived distance to pump to calculate total distance needed to achieve the goal. For example, the total distance required to complete task X will depend if it can be reached with the current fuel level and water level. If no, distance to the fuel pump or well and back to the task must be added to the total distance. The total distance must also take into account the fuel required to return back to the fuel pump. Once the final plan is decided, the first action in the plan will be executed by backward chaining it. Once backward-chaining returns true, the action will be deleted from the plan and the successive goal will be pursued. A plan is generated each time-step and will be used to compare with the current plan.

Plan Comparing

The generated plan that differs from the current plan will be decided by the deliberative layer if it should switch by comparing its measured score value. The score value of a plan is measured by the following algorithm:

$$ScoreMeasure = \frac{PlanScore_{NewPlan} - PlanScore_{CurrentPlan}}{PlanScore_{NewPlan} + PlanScore_{CurrentPlan}}$$

If the *ScoreMeasure* is larger than 0.5, the agent will decide that the new plan, with a certain amount of distance increased, is worth switching as it increases the scores obtained by more than 50%.

EVALUATION

Table 5.1 shows results from 10 consecutive run with its average. A screenshot of it can be reviewed from the Appendices 1.1. The Highest score is 16.2 billion with 1798 delivered tasks, with its screenshot can be referred from the Appendices 1.2. The lowest score is 5.1 billion with only 1016 tasks, with its screenshot can be referred from Appendices 1.3

The average score is 11 billion with 1517 number of tasks delivered. An earlier version of the agent that only implements backward chaining had only an average score of 10 billion with 1401 number of tasks delivered, as shown in Table 5.2 the results in a table form. The improvement means that having a deliberative layer increases the performance of the agent, making it reasonably better within the given task environment.

recent

Round no	Tasks Delivered	Score
1	1499	11081326021
2	1720	14713485440
3	1313	8742917742
4	1183	682367178
5	1503	11541956337
6	1816	16592960888
7	1490	11206260200
8	1564	12163669048
9	1372	9237980584
10	1713	14544207757
Average	1517.3	11050713120

Table 4.1 : Results from 10 consecutive run with only Backward-chaining with average score

old implementation

Round no	Task delivered	Score
1	1253	7790682872
2	1168	6870348864
3	1505	11443518835
4	1352	9231923792
5	1800	15959638800
6	1282	7943911718
7	1548	12160029168
8	1132	6662364492
9	1569	12406701186
10	1409	9635522586
Average	1401.8	10010464231

Table 4.2 : Results from 10 consecutive run with only Backward-chaining with average score

DISCUSSION AND CONCLUSIONS

Problems faced

Although planning forward does increases its performance, the algorithm can only act in greedy search behaviour, choosing only the best of a certain criteria. The plan cannot be guaranteed to be complete nor optimal because of the partially observable environment. Hence, pure deliberative or deliberative may have decent performance on the given task environment, but may not produce the best agent performance.

Plan comparing could have used a better performance measures that takes distance into account. Although the plan generated is considered a shortest route to complete the tasks, having a distance measure between plans involved with the score measure will produce a more balanced measure value. However, if the difference of distance is too small between plans, the measure value will diminishes the huge increment of total score gained in the new plan and performance may drop. This behaviour shows that the performance of deliberative approach have a tight dependency on the quality of the chosen performance measure.

One of the implementation problem faced was having backward chaining producing a list of action. This may be due to the structure of the classes and its dependency on recursively looking through preconditions and effects to create the list of action. This however was solved with an action being the output each time a run of backward chaining returns false (where a goal is not satisfied).

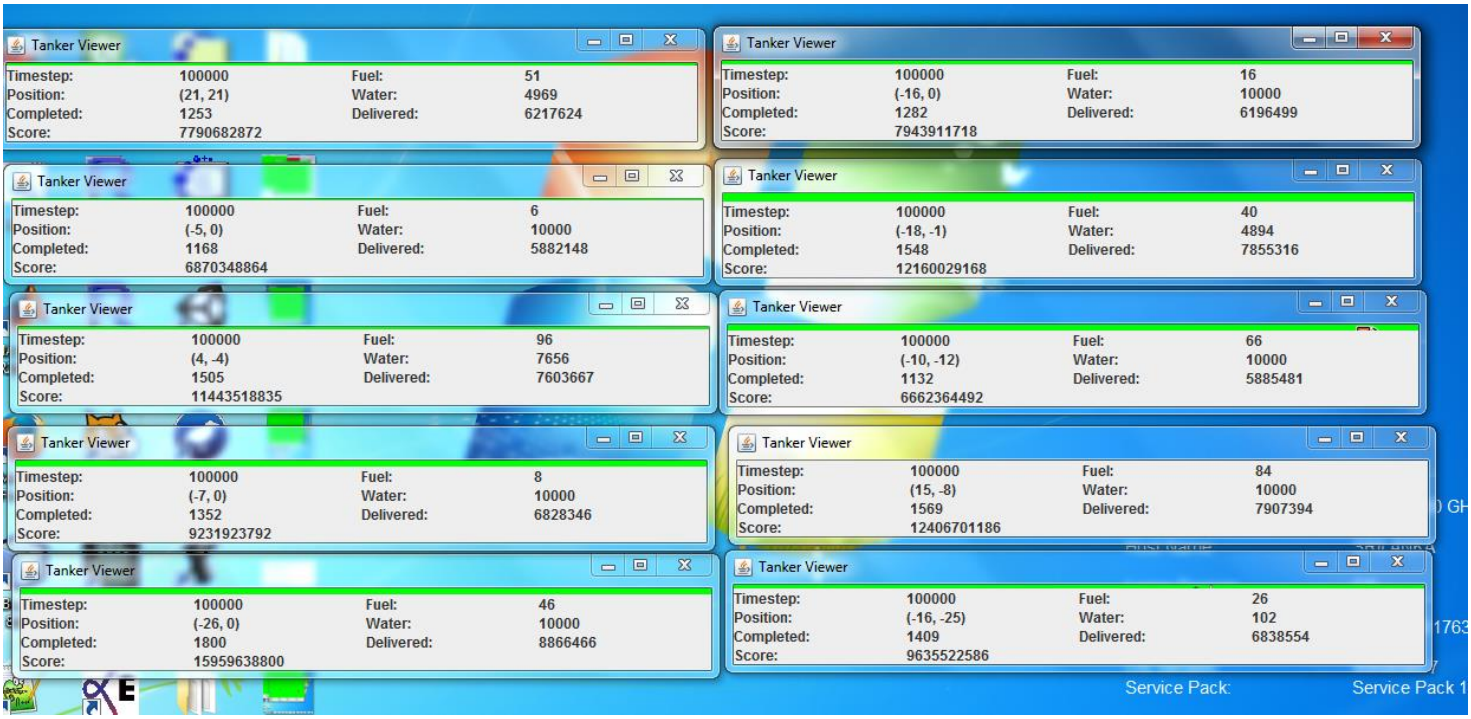
In conclusion, a combination of deliberative approach with reactive approach depends heavily on the control rules, deliberative algorithms, and what are the standards that the plans are built upon. The hybrid agent architecture performs decently well in the given task environment, but may not just be the best approach without any optimization.

REFERENCES

Ferguson, I. A. (1992). Touring machines: Autonomous agents with attitudes. *Computer, 25*(5), 51-55.

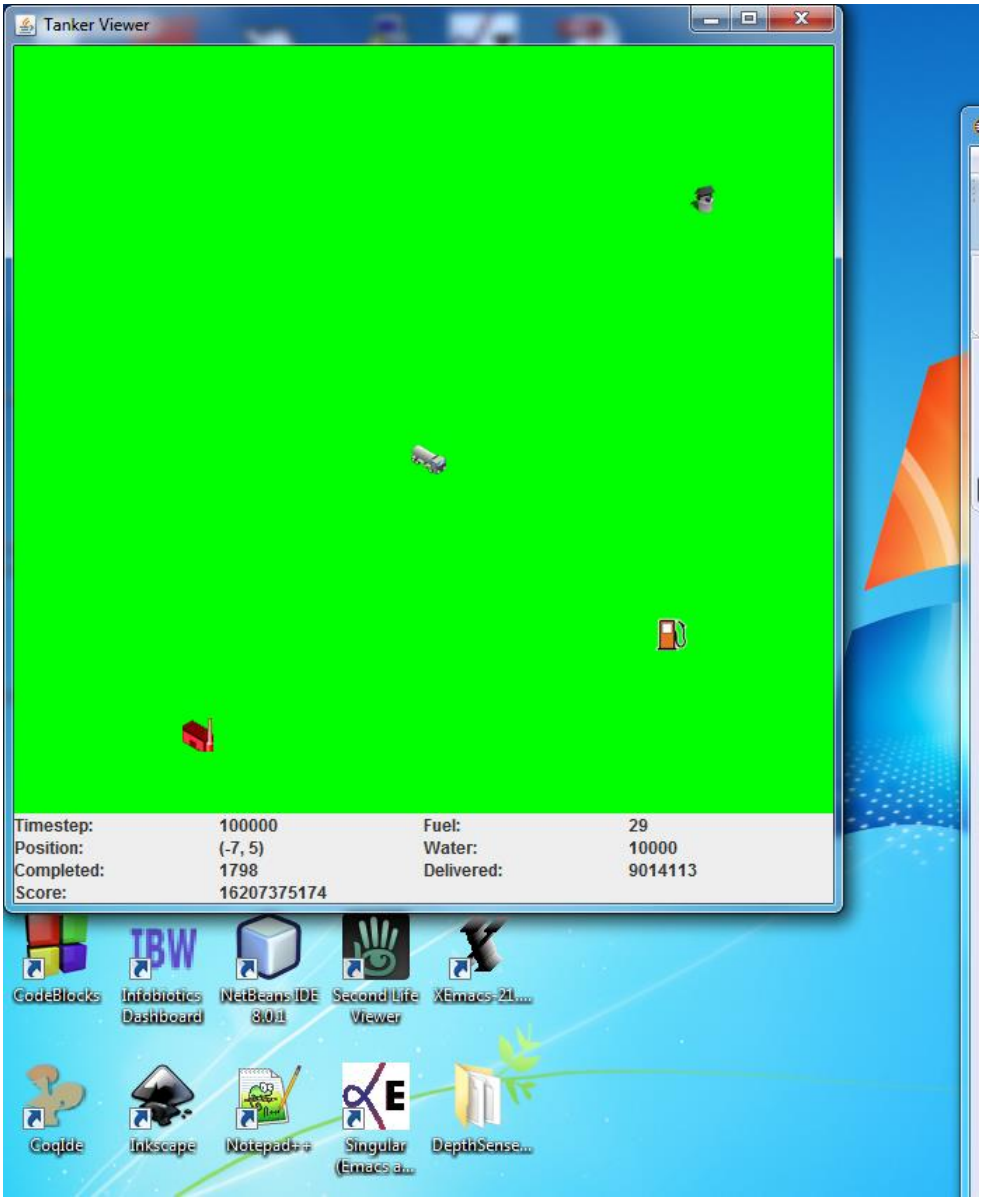
Russell, S., Norvig, P., & Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs, 25*.

APPENDICES

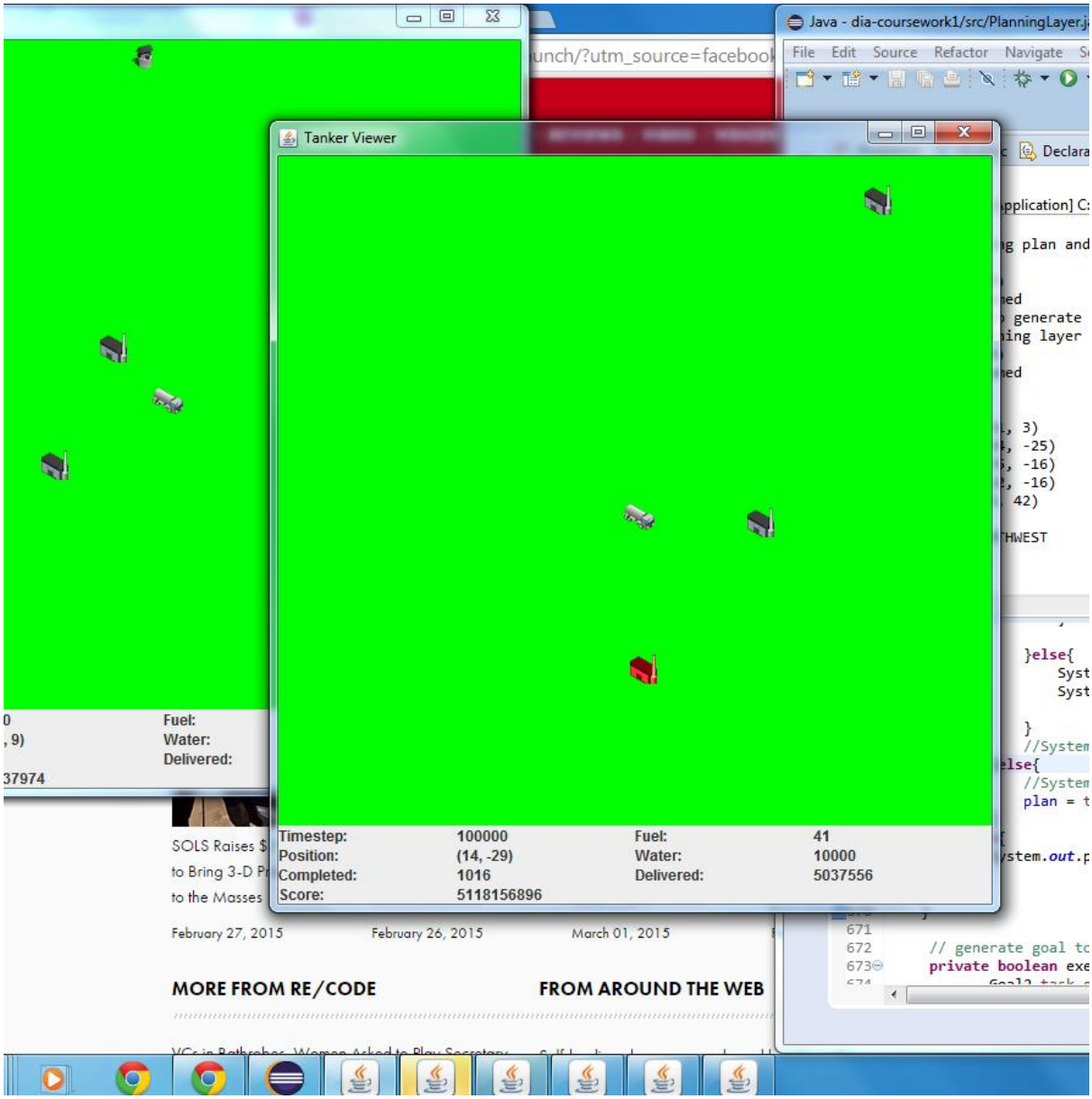


Appendices 1.1 *: Screenshot from 10 consecutive run from current implementation*

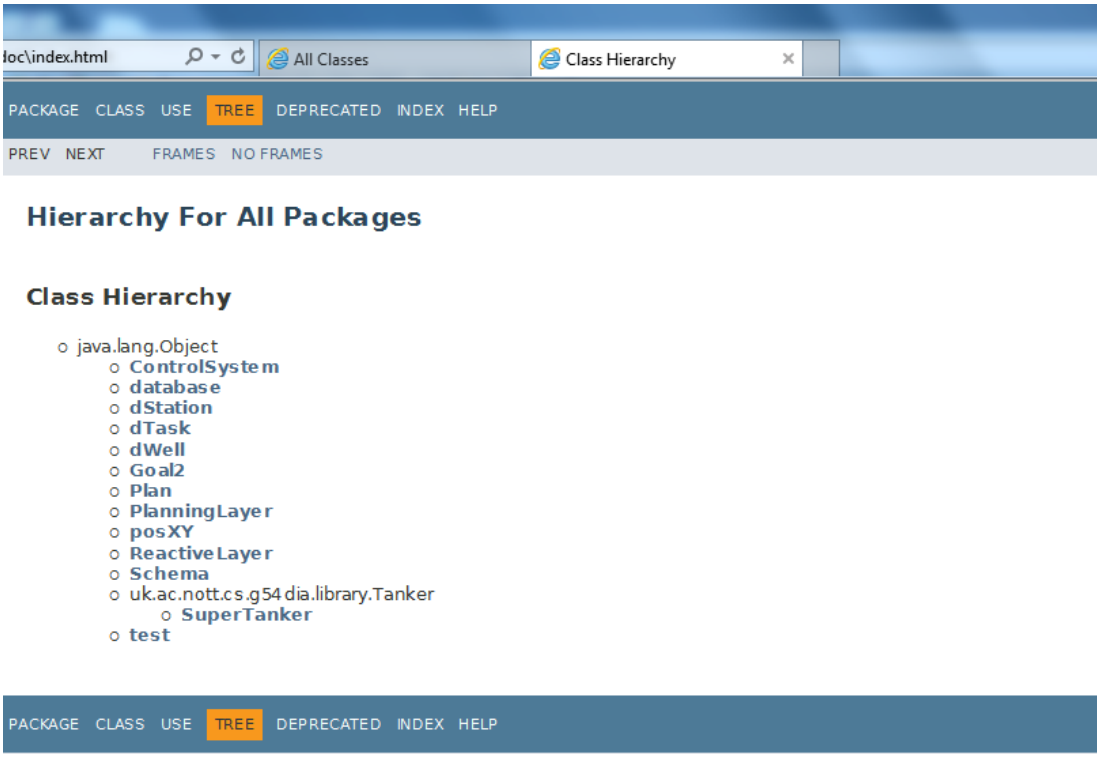
Appendices 1.2 *: Screenshot from 10 consecutive run from older implementation*



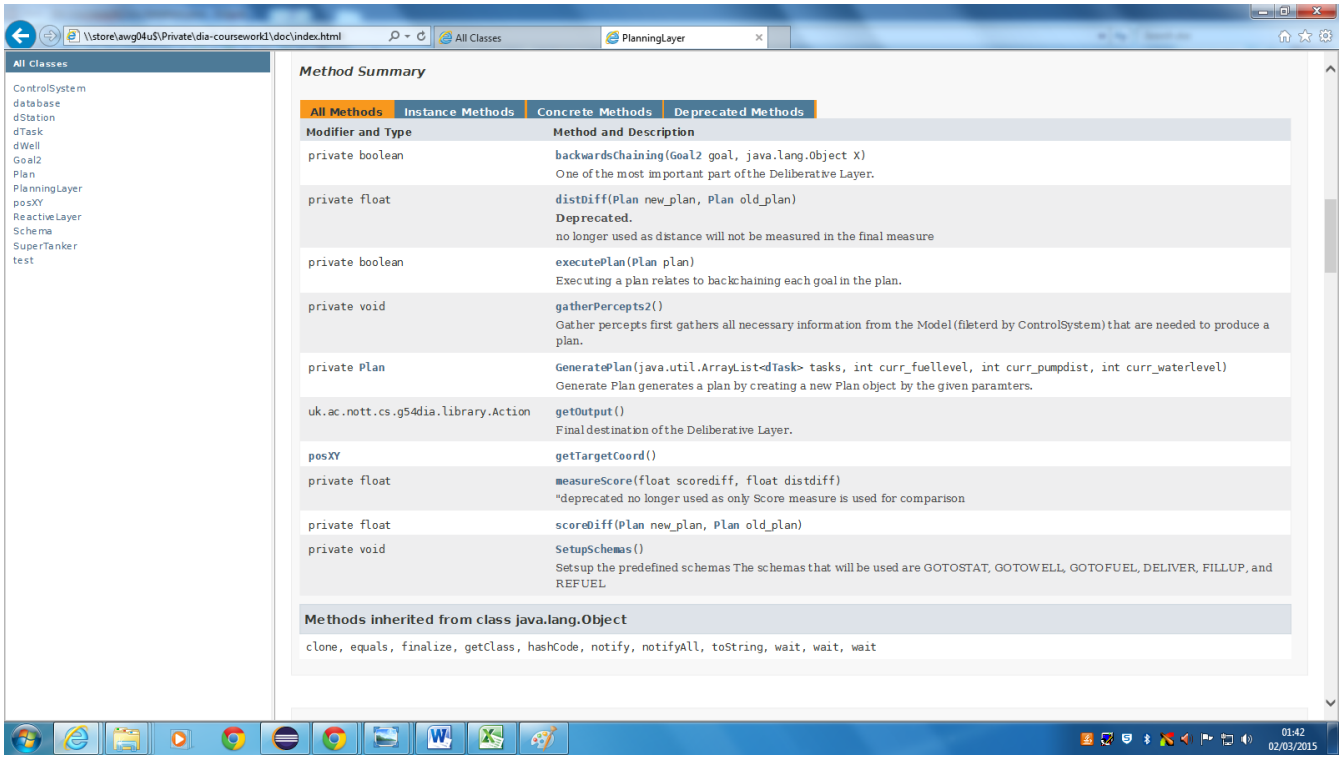
Appendices 1.3 *: Screenshot of the result with the highest achieved score, 16.2 billion with 1798 tasks delivered*



Appendices 1.3 *: Screenshot of the result with the lowest achieved score, 5.1 billion with only 1016 tasks delivered*



Appendices 1.4 *: Screenshot of All classes in the agent. For full documentation, please access src/doc/index.html*



Appendices 1.5 *: Screenshot example of methods available from Planning layer. For full documentation, please access src/doc/index.html*