Problem Description

The second coursework builds on Coursework 1 and uses the same 2D environment and task. However, in the second coursework, multiple cooperating agents must collect waste from stations and dispose of it in wells. The goal of the agents is to collect as much waste as possible in a fixed period of time.

Introduction

In order to build the multi-agent system, work from the previous coursework was adapted. A fleet of agents are created and operate in the world. Each agent within the fleet aims to maximise the amount of waste it collects in the given timeframe, in this case $10,000$ time steps. All the agents have individual constraints; they can only carry up to 1000 units of waist, and their range is limited by their fuel level. They must also allow for the other agents to function within the world. If any of the agents run out of fuel the run is stopped.

The environment is a discrete, infinite grid with wells, fuel-pumps and stations randomly distributed throughout. The agent at any one point can see a 20×20 grid of the world around it. As an agent moves around the world it discovers the location of the aforementioned points and updates about their state, e.g. if a station has a task.

Agent Design

Due to the fact that the agent has multiple goals to balance, collecting waist, not running out of fuel, and is rewarded for doing so as efficiently as possible a reactive architecture with state was chosen. The agent must also collaborate with other agents working in the world. Doing so enables the agent to react to strict goals, such as never running out of fuel, whilst also having a memory and plan in order to perform tasks efficiently.

The agent use the *MoveAction* over the *MoveTowardsAction*. Although it uses twice as much fuel per step, it is guaranteed to succeed enabling the agent know whether a target point can successfully be reached with 100% certainty.

2.1 Word Representation

As each agent moves around the world they add to the shared internal representation, the "world map". Storing a representation of the world allows the agents to ask questions and plan referring back to points outside of its current view. Additionally, since the world map is shared by all agents, they can use it to communicate with one another. Internally the world map is comprised of a 2D array and is filled in and updated as points are visited.

2.1.1 Implementation

The map exists as a Java class and is updated by each agent once every time step with its new view. In addition to containing the "world map," the map class also contains a reference to the agent fleet. Using the reference to the agent fleet, the map class can provide agents to plan collaboratively. The map class also exposes various utility functions the agent can use. The core functions:

- **Find closest of type** – Will find the closest know point that is of the given type (e.g. Well, FuelPump)
- **Calculate distance** – Will calculate the distance between two points.
- **Find plants with a task** – Find the closest station with a task.

The class also exposes other functions built to provide more efficient access to often requested points e.g. closest plant with a task. The implementation was built to be easily extendable and loosely coupled to the agent as such it is not the most efficient implementation.

2.2 Core Reactive Layer

The agent is split into two sections, the core reactive layer and the task selection layer. The core reactive layer is responsible for the survival and base functionality of the agent, it remains unchanged from the previous work. Within the layer, there are two classes of actions. Functional actions perform basic core functions and survival action stop the agent from entering an undesirable sate (e.g. running out of fuel).

Functional actions consist of collecting waist when over a plant, empty waist when over a well, and refuelling when over a fuel pump. These actions use no fuel and so are always performed with the highest priority when the required conditions are met.

The survival actions act ensure continued functionality of the agent, they are most often trigged when the agent is exploring the world. They consist of a refuel and an empty action. The refuel action is trigged if the agent will move out of range of the nearest know FuelPump[1]. The agent will enter a refuel state signifying it is strongly committed to moving towards the nearest FuelPump and continue to be so until fuel is collected (which is taken care of by the functional actions). The empty action is trigged when the agent no longer has capacity to collect more waist. Although not necessary for survival the waist must be emptied in order for the agent to continue to function. Similar to the refuel action, the agent will enter an empty state signifying it is strongly committed to moving towards the nearest Well. However it will only do so provided it can make it to the closest know well without triggering a refuel, otherwise it will refuel and then empty itself.

Finally if there are no tasks to perform, the agent will enter an exploration state. In this case it chooses a random diagonal and will drive in the chosen direction. Diagonals are chose to maximise the exploration of the world, each time step 17 new points are seen compared to 10 for a cardinal direction. Since the initial sate of the agent is exploration, the first diagonal is assigned to be different from all other agents when the fleet size is less than 4. The agent will stop exploring if a higher priority action is triggered, either a refuel or a task is located.

2.2.1 Implementation

The core of the reactive layer exists as a method in the abstract agent class and is called every time step. It returns an action if the following conditions are met (conditions are checked in the given order):

State	Action
Over a fuel pump	Refuel the agent unless fuel is full
Over a well	Empty the waist if the agent is carrying any
Over station	If the agent has capacity and the station has a task collect waist
Moving out of range of fuel	Strongly commit to refuelling, moving the agent towards the fuel pump
Waist level is max	Strongly commit to emptying if well within range otherwise refuel

If no action is returned the agent will call the task selection layer. If the agent has no task it will use the time step to explore, continuing on its chosen diagonal if the pervious step was used to explore or otherwise randomly select a new diagonal.

2.3 Task Selection Layer

The task selection layer is responsible for achieving the higher level goals of the agent, in this case maximising the waist collected. It does so by setting target locations and working towards them.

For the current problem the goals are constrained in a few main ways. The target location must be in range for the agent i.e. the agent has enough fuel to get there. The agent must have capacity to collect, at least in part, waist from the target. No other agent is already working towards the task.

In order to function effectively, the agent must balance exploitation, collecting waist, and exploration of the environment, discovering Plants (with or without tasks), Wells and FuelPumps. The agents must also be aware of each other, if two agents work too closely together they can cause one another to operate less efficiently.

[1] A point is considered to be in range if the distance between the agent and the point plus the distance between the point and the nearest know FuelPump is less than the range of the agent.

One approach is for task selection is for the agent to create a full plan, an ordered set of points to visit, that maximise the waist collected. Calculating the true optimal solution is highly computationally expensive. Employing approaches such as evolutionary algorithms it is possible to find a plan but these approaches are still expensive. Additionally, Since the environment is dynamic and changes in non-deterministic ways, e.g. new tasks can be generated every time step, in order to be optimal truly the plan needs to be re-evaluated every time new information is discovered.

Rather than build a full plan, the agent makes use of simple heuristics in order to choose the next target. This enables each agent to balance exploration and exploitation of the world.

2.3.1 Implementation

At each time step the task selection layer sets a soft target, a point to move towards. If the agent already has a target it remains the same, if there is none one is calculated. The soft target is removed from memory if a refuel action is triggered or the agent reaches the target location.

The agent uses simple heuristics to select its target. Since the main goal is to collect waist, the agent targets the closest know plant with a task that is not targeted by another agent and that there is no free agent closer. Additionally in order to avoid interfering with other agents, When an agent selects a target, an exclusion region 15 tiles around the target is created stopping other agents choosing targets within this range. This is done to encourage agents to working in different parts of the map whenever possible.

If an appropriate target exits the agent will move towards it, if the closed appropriate target is out of range a refuel action is trigged. If no target exits an exploration action occurs. If the waist capacity is below a given threshold, the agent will target a well if it is closer than the nearest target, otherwise it will move toward the plant and attempting to collect the waist relying on the core reactive layer to empty the agent when it is full.

Results

There are a few hyper parameters to tune for the state approach, the feet size and the exclusion region size. For the sake of simplicity a simple grid search with a fleet size of 4 was used to find an optimal exclusion size of 15. This value was used for all further experiments.

The table below contains results for fleet sizes one to four. The submitted code uses a fleet of size 4.

Run	**Fleet Size**			
	1	2	3	4
1	158,000	118,500	122,200	91,710
2	126,100	131,800	115,600	125,500
3	134,000	132,800	134,000	130,200
4	108,300	120,900	103,500	112,200
5	158,300	141,900	120,100	117,700
6	118,100	104,100	109,000	94,260
7	150,800	147,700	143,400	126,800
8	148,800	126,800	108,400	129,000
9	162,900	149,900	138,300	128,900
10	129,400	114,400	117,700	114,300
Average	139,470	128,880	121,220	117,057
Std	18,858	14,865	13,420	14,203

Conclusion

Overall the proposed solution performs poorer than expected. As can be clearly seen in the results, each additional agent added to the fleet causes a marginal decrees in performance. The author spouses this is caused by a few issues.

The primary reason for the reduced performance is that the effective exclusive range of each agent is reduced. Since

communication between agents is limited, all discovered tasks are effectively shared between all agents. Attaching the exclusion zone to the target location was an attempt to reduce agents stealing tasks from one another but it has proven to be a less than optimal solution. Other solution such as scaling the size relative to task distance and amount value of tasks in a region could have proved to be more effective.

Additionally, since all agent start at the same location, each agent takes more time to find is own area to work in. The exploration approach used was almost identical to that of the single agent, the only difference being the initial direction the agents take. As such, the agents can randomly choose to explore the same area of them map. This overlap of exploration can cause them to time competing for tasks close to one another. Better communication between agents about regions to explore or even subdividing the task environment up for each agent to have their own are to work in could deal with these issues.

The individual agents have the same strengths and weaknesses of those stated in the previous work

- Non-determinism in movement actions, would reduced performance as the agent would have difficulty determining if a target location was in range.
- if the move fail probability for the *MoveTowardsAction* was less than 0.5 the current implementation would be unable to take advantage.
- If compute time was a factor the current architecture would perform well.
- highly adaptable to changes in the view size and distribution of items in the world

Collectively the fleet worked well but in its current configuration and task environment scales with a factor of ≈ 0.9. With the current scoring system this renders the multi agent approach somewhat redundant. If the goal was to collect as much waste in total or reduce the amount of time a plant must wait for waist to be collect this approach would prove to be more effective. It is also worth noting that all agents in the proposed solution both collect waist and map the environment. This made the most sense as all agents factor into the score, however if agents were valued differently, it could be beneficial to have some agents map the environment whist other perform tasks.

It is clear that in order to scale better than linearly more intra-agent communication is needed and perhaps use of deliberation over simple heuristics for task selection and assignment to avoid agent competition causing inefficiencies.