G53DIA:
Designing Intelligent Agents

Lecture 8: Coursework 1

Brian Logan
School of Computer Science
bsl@cs.nott.ac.uk

Outline of this lecture

- description of the coursework
- project resources
- structure and content of the report
- some hints to help get you started
- tutorial arrangements

First coursework

- the first coursework involves the design and implementation of a single agent

- assessed by a report describing your agent and the associated code

- submissions due **Friday 26th of February**

The problem

- task consists of collecting and delivering water to *stations* (customers)
- environment contains a number of stations which periodically generate *tasks* – requests for a specified amount of water
- environment also contains a number of *wells* from which water can be collected
- the goal of the agent(s) is to deliver as much water to as many stations as possible in the time available

The objective

- objective is to investigate agent architectures and algorithms for water collection and delivery strategies, e.g., the trade-off between:
  - exploring the environment for sources of water and tasks
  - deciding which tasks to perform and how to perform them (the order in which to process requests for water and which wells to use)
- to understand which features of the environment are necessary for a particular architecture to work well
- aim is not just to build a agent that works well, *but to understand why it works well*

Why water delivery?

- the water delivery problem is representative of an important class of agent design problems where agents must try to satisfy competing goals within constraints

- need to decide:

  – how to choose between competing goals (ends)

  – how to achieve each goal (means)

  – while respecting various constraints (on percepts and actions)

Task environment

- the task environment is given as part of the coursework specification and specifies:
  - – properties of the task and environment
  - – percepts and actions available to an agent
- the task environment should not be modified or extended
- all other decisions regarding software design and implementation strategy are up to you
- you will receive guidance and feedback on your project in individual tutorials

Task environment in detail 1

- the environment is discrete and consists of a grid of cells
- the environment contains randomly distributed stations and wells
- stations periodically generate *tasks* – a request for a specified amount of water (max 10,000 litres)
- tasks persist until they are achieved (a station has at most one task at any time)
- wells contain an infinite amount of water
- there is a single fuel station in the centre of the environment that contains an infinite amount of fuel

Task environment in detail 2

- the agent can see any stations and wells within 12 cells of its current position
- if a station is visible, the agent can see if it has a task, and if so, how much water is required
- the agent can carry a maximum of 100 litres of fuel and 10000 litres of water
- the agent moves at 1 cell / timestep and consumes 1 litre of fuel / cell
- filling the fuel and water tanks and delivering water to a station takes one timestep
- if the agent runs out of fuel, it can do nothing for the rest of the run

Task environment in detail 3

- the agent starts out in the centre of the environment (at the fuel station) with 100 litres of fuel and no water

- a run lasts 100,000 timesteps

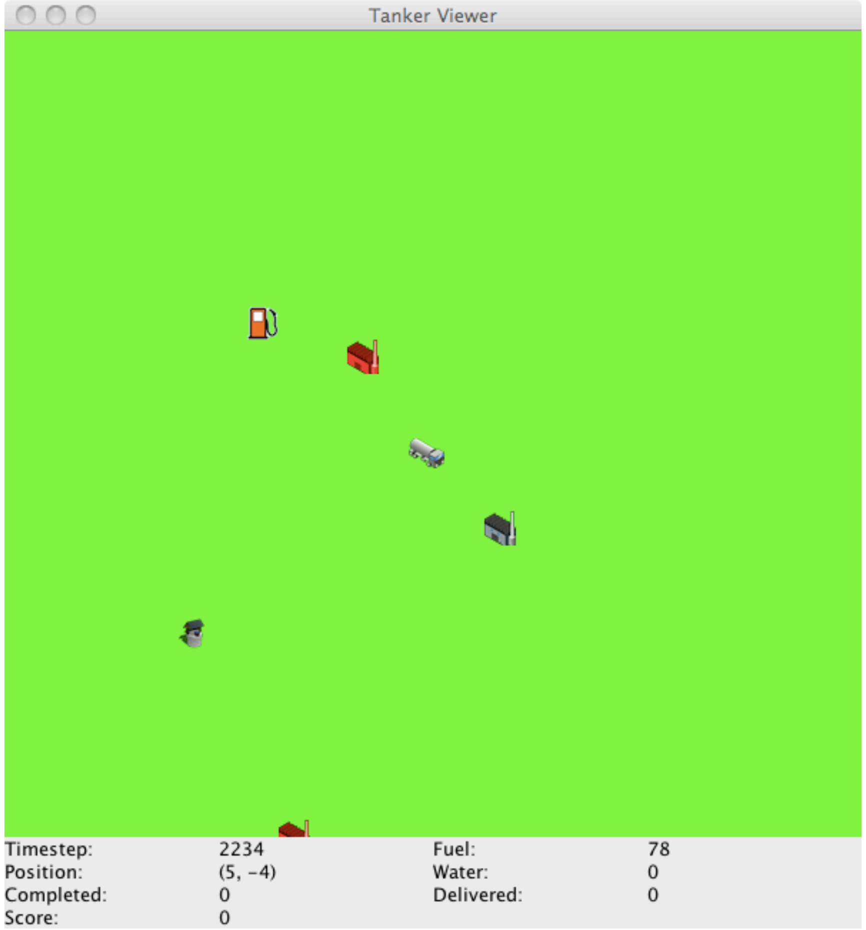
- the success of an agent in the task environment is determined by its score at the end of the run

- the agent's score is given by the amount of water delivered × number of (completed) deliveries

Project resources

- *Java agent package* (Package uk.ac.nott.cs.g53dia) as a starting point for your project work
  - implementation of the task environment which generates a random set of stations & wells for each run, and periodically generates tasks
  - an abstract agent class which provides methods for sensing and acting
  - a concrete 'demo agent', that chooses actions at random
  - all you have to do is (re)write the action selection function …

Task environment



Timestep:	2234	Fuel:	78
Position:	(5, −4)	Water:	0
Completed:	0	Delivered:	0
Score:	0		

Submissions

- submission consists of two parts:
  - a report describing your agent
  - the code implementing your agent

Suggested report structure

- your name, email address, student id and “G53DIA coursework 1 report”
- introduction
- relevant background material
- software design
- software implementation
- evaluation (average score over at least 10 runs)
- discussion/conclusions
- references

Software design

- the *software design* states how you are going to achieve the project specification
- it is an *abstract* description of how you are going to solve the problem:
  - high level: what sort of architecture your agent has
  - low level: how the agent decides which tasks to perform in which order, and which wells to use for each task
- it should *not* be a list of classes and methods

Documenting your design

- you need to describe your design and the *reasons for each design decision* clearly in your report
- a good approach is first to say which general type of architecture your agent has (and why)
- then explain the main components or steps in its operation in outline
- then describe each component in detail

Software implementation

- high level description of the implementation
  - which data structures were used
  - how the algorithms were implemented etc.
  - why the approach adopted was chosen
- try to focus on the ‘interesting’ bits of the implementation
- *do not* include a full code listing in the report

Evaluation & discussion

- how well does your agent work, e.g.,
  - what score does it achieve (on average)
  - how does this compare to other agents (e.g., simpler versions of the same agent)
- why is your solution appropriate for the task environment
- which features of the task environment are critical – how would you expect your agent to perform in other task environments?

How it will be assessed

- aim of the module is to understand the relationship between and agent’s task environment and its architecture
- to do well, you need to develop an agent that works well and demonstrate that you *understand why your agent works well*
- marking is therefore based on:
  - the capabilities of the implemented agent, including the quality of the specification, design and implementation
  - the degree to which the specification, design and implementation are clearly documented in the report
  - clarity of presentation in general (including grammar, spelling and punctuation)

Assessment guidelines

- very broadly, a basic implementation of the requirements (and corresponding report) will gain a pass mark

- extra credit will be given for submissions that demonstrate a clear understanding of the relationship between the task environment specified for the coursework and the architecture of your agent

- full assessment guidelines on the moodle page …

Some hints …

Classifying the task environment

- a good way to start is by classifying the task environment given as part of the problem

- what properties do the agent’s task, percepts and actions have?

- hint: see lecture 2 …

Designing the architecture

- once you understand the features of the problem, think about their implications for the agent's architecture
- use the features of the task environment to help you make *and justify* high-level decisions about the design of your agent
- e.g., is the environment observable? – if so, what does this mean for the architecture of your agent?

Low-level design

- once you have made the high-level decisions, think about how each aspect of the agent could/should be implemented

- e.g., how will the agent search the environment, or decide what what to do next

- will your agent always use the same action selection function, or will the action selection function vary with time, etc.

- you can use algorithms from agent case studies in the lectures, from previous AI courses or AI textbooks, or invent your own solution

Initial implementation

- once you have a design you will want to start thinking about how to implement it
- one way to start is by implementing a simple agent that can collect water from the nearest well
- basic capability that will be required by almost any design
- ensures that you are familiar with the toolkit, and gives you some practice using it

Collecting water from the nearest well

- starting from the fuel station, your agent must be able to:
  - find the nearest well
  - collect water from the well
  - return to the fuel station
  - without running out of fuel

Finding the nearest well

- finding the nearest well requires:
  - finding a well
  - somehow ensuring that the selected well is the closest to the fuel station
- several approaches are possible:
  - design the search for wells so that the first one found must be the closest; or
  - having found a well, check to ensure that there is no other well(s) closer to the fuel station

Collecting water

- collecting water requires navigating to the selected well (easy)
- returning to the fuel station (also easy)
- not running out of fuel in the process (see the demo agent)

Tutorials

- the project work is supported by group and individual tutorials
- **group tutorials** cover the use of the Java agent package
  - the first group tutorial is on Friday the 12th of February at 11:00 in C60
- **individual tutorials** cover the design and implementation of your agent
  - individual tutorials are scheduled for 11:00-12:00 and 15:00-16:00 on Mondays, 15:00-16:00 on Tuesdays and 11:00-12:00 on Fridays (starting 15th of February) in C34
  - email me to make an appointment