Multi-Agent Systems Summative Assignment

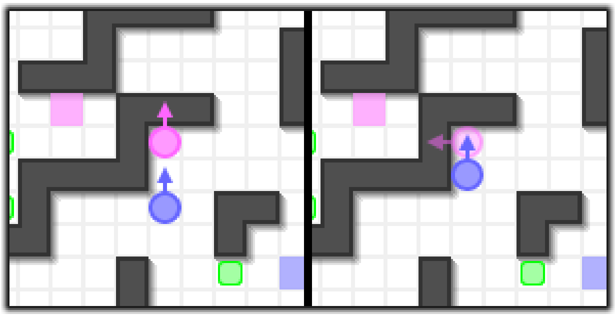
James King

November 24, 2013

1 Application Specification

1.1 Overview

World The task is a simple turn-based game within a tile based world, where two or more teams of bots compete for survival. Each team starts with one or more *home* tiles, upon which a bot belonging to that team begins. Every remaining tile in the world may be a solid *wall*, a *resource* item, a *bot*, or *empty*. The world is a two dimensional grid of fixed dimensions, and has the topology of the surface of a torus (travelling off one side of the grid will bring you to the opposite side). The world is initially unknown to players of the game, but at the start of each turn every player is given the state of previously undiscovered tiles that are within a certain fixed *vision radius* of any agent on their team. The locations of resources, along with the positions, team affiliation and direction of agents within the vision radius is also given before each turn.


(a) The blue bot moves to eliminate the pink bot.

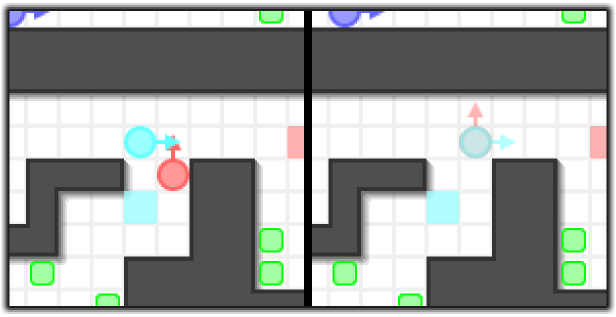
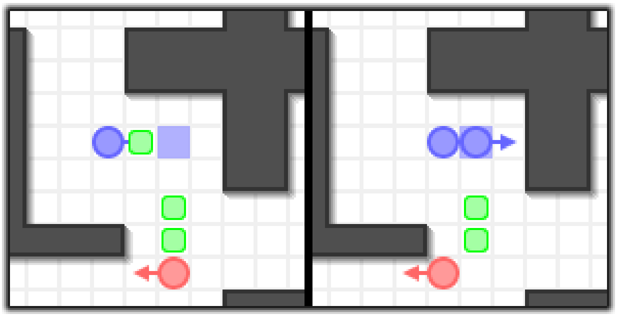

(b) Mutual elimination when two bots occupy the same tile.

Figure 1: Demonstrations of the two rules for bot elimination.

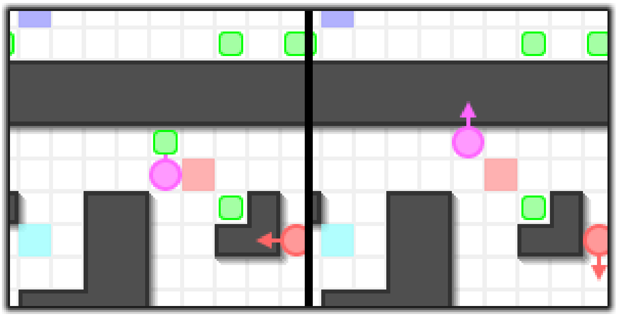
Bots Bots have a *direction* property, which may be any one of the cardinal directions (*north*, *east*, *south*, or *west*). Each turn, each team decides a single action to perform with each bot belonging to them. This action may be to rotate the bot *left* or *right* (so a bot that was facing east which turns left will now face north), to *move* one tile in the direction it is currently facing (unless a wall is in the destination tile, in which case it does not move), or to *pass* and do nothing. After each team decides on a set of moves, all the bots commit to their assigned instruction simultaneously. If any two or more bots occupy the same tile they are removed from the world before the next turn. Also, a bot is removed if it is neighbouring another bot of a different team that is facing it.

Resources Resources are items that appear at regular intervals in the world on randomly selected empty tiles. These are initially stationary, but if a bot appears on a neighbouring tile and faces the resource, the bot begins to *carry* the resource which will attempt to remain one tile in front of the carrying bot. If the carrying bot moves forward, the resource is *pushed* in the direction the bot moved by one tile. If the bot turns, the resource will move to whichever tile the bot is now facing. However, if the destination tile for the resource when either pushed or turned isn't vacant, the resource is removed from the world. Finally, if the resource is moved to a home tile, a new bot belonging to the home tile's corresponding team is created in its place.

Solution Requirements and Interfaces Each team will have a single program that controls all of the bots on their team. Each program is initially given the dimensions of the world, number of teams, vision radius and allowed time per turn over standard input, and then at the start of each turn they provided with given the tile, resource and bot information as previously specified. Each team's program is then expected to produce an action for each bot over standard output, and upon the arrival of all instructions the game moves to the next turn. If any team's program fails to produce its set of actions before the allotted time has elapsed, all bots belonging to that team are eliminated and its program terminated. Each controller program may be implemented in any language that supports standard input and output operations.



(a) Moving a resource (green) onto a home tile spawns a new bot and consumes the resource.



(b) Pushing a resource into a wall, bot, or another resource will remove it.

Figure 2: Demonstrations of resource consumption and destruction.

End Conditions and Goals The game lasts until either a predetermined turn limit is reached, or only bots belonging to one team remain. When the game ends, whichever team has the most remaining bots is declared the winner. The aim of the game is therefore to try and eliminate as many bots belonging to opposing teams as possible, while attempting to transport resources back to a home tile owned by your team and preventing other teams from doing so.

1.2 Task Environment Identification

A useful initial step when designing any kind of system involving agents is to classify the environment in terms of how it is perceived by and reacts to agents, and to recognise exactly which behaviours should be encouraged in agents. These properties of the task are usually known as the *performance measure*, *environment properties*, *actuators* and *sensors*, or the *PEAS*, as discussed by Norvig and Russel in Artificial Intelligence: A Modern Approach[1].

Performance Measure As this is a cooperative system (at least with respect to the bots belonging to one team), where the cooperating bots reside in the same program, performance may be measured for the team as a whole and not for each bot independently. Each team is attempting to maximise their number of active bots, while reducing the number of bots in opposing teams. This can be expressed as a ratio of active bots belonging to the measured team against the total number of active bots. However, this would suggest that a scenario where 5 bots are split between three teams (A, B, C) using the distribution $(2, 2, 1)$ is equivalent to the distribution $(2, 3, 0)$ for team A. This should not be the case, as in the first scenario team A is tied in first place with team B, but in the second scenario team A is in second place and so should have a lower performance measure. One possible revised performance measure expression that resolves this is the following:

$$\text{performance}\,(\text{team}) = \frac{|\text{bots} \cap \text{team}|^2}{\sum_{t \in \text{teams}} |\text{bots} \cap \text{t}|^2}$$

Figure 3: A performance measure expression that takes into account the number of bots belonging to opposing teams.

This performance measure can differentiate between distributions of agents between teams where the original expression could not, so eliminating bots from a larger (and therefore more threatening) opposing team would provide more utility than doing so from a small one. Maximizing this score will increase the probability of winning the game, and in fact as soon as the performance score for a given team reaches 1.0 that team must win, because all opposing agents have been eliminated.

Environment Properties Using the definitions provided by Russel and Norvig[1], the environment is *partially observable* due to how only the contents of tiles near to bots are revealed to the team each bot belongs to. Naturally, it is a *multi-agent* system that is both *cooperative* (between agents belonging to the same team) and *competitive* (between agents belonging to distinct teams). The environment is slightly *stochastic* due to the random distribution of resources over time, and *uncertain* because of its stochastic nature, the presence of other agents, and because it is partially observable. Turns are *sequential* rather than episodic because each choice may affect every following one. Environment state and the intervals between percepts are *discrete* since the world is aligned to a grid and time is divided into turns, and the environment is mostly static except for the maximum turn time limit which renders it *semidynamic*. Finally, the outcomes of actions are *known* because they always produce an expected result for some localised aspect of the environment.

Actuators and Sensors Each team can interact with the environment by selecting an action from the set of allowed actions for each of their bots. Percepts include several elements; first is a subset of previously undiscovered tiles revealing their state (either empty, a wall, or a home tile for a specified team), then the locations of visible bots along with their team and the direction they are facing, and finally the locations of visible resources.

2 Agent Roles

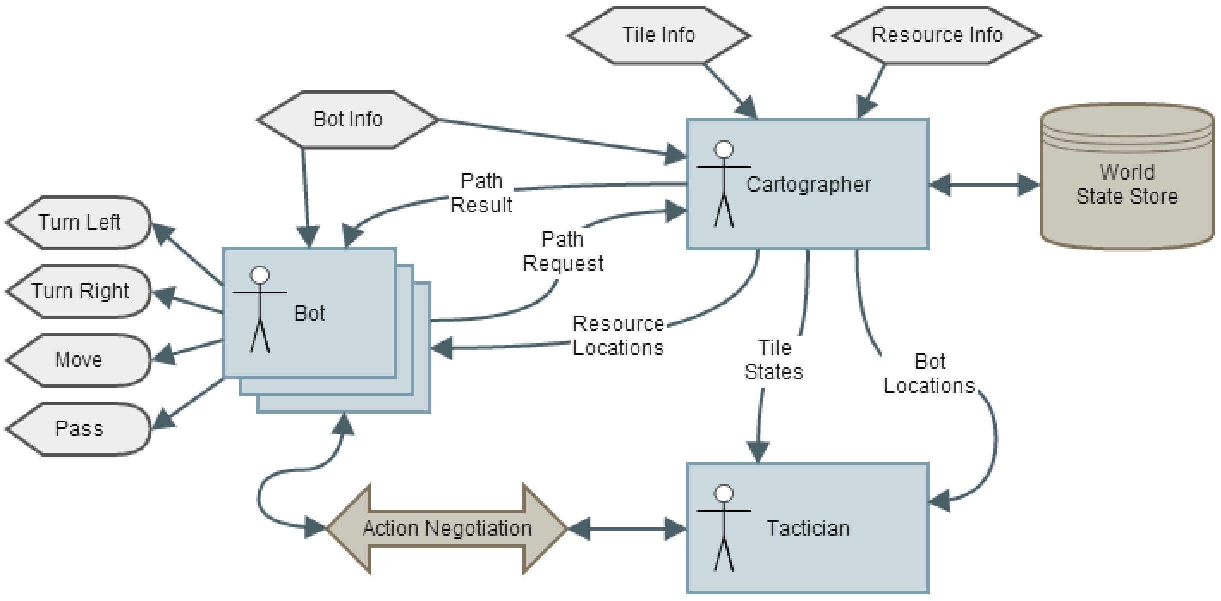


Figure 4: Diagram showing the interactions between the three proposed agent classes.

2.1 Cartographer

As the world is partially observable, its state must be recorded and updated so that static tiles that are no longer visible may be remembered. It would be wasteful for every bot to hold a separate representation of the world, so a distinct agent should be implemented whose main purpose is to receive world information to be stored and recalled. This agent would be able to provide several services for other agents in the system, such as to plan a path between two locations in the world, decide

References

[1] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach.* Pearson Education Inc., 3rd Edition, 2010.