Report Multi-agent Systems Module I

Tom Jacobs (s0214835), Thomas Uyttendaele (s0215028)

February 27, 2013

Part I

1.

Run 1:	A2, B2, C1, B2, B1, C3, B1, B3, C2, B2, A1, A3.
Run 2:	C2, A2, B3, C3, A1, A3, B1, A3, A1, A2, A1, B2.
Run 3:	A3, C3, A3, C2, B2, B1, A1, A2, C3, A2, B2, C1.

2. The agent produces different behaviour every run, caused by lines 46-47. The first line generates a goal for every possible next location, the second line randomly chooses one of those goals and sets it as the next location to go to.

3.
- Agent percepts the information of the entire map and processes it to beliefs.
- Decides which location it is at.
- Generates all possible goals.
- Uses line 47 to select the next location to go to.
- Processes all perceived information while going to the next location.

4. Based on the current state the next action can be decided. Possible states are: *unknown*, *travelling*, *arrived* and *collided*. Every time the main module is executed, *goTo* is called. The preconditions enforce that new locations can only be chosen when an agent is waiting inside a room, or waiting in front of an occupied room. In the event module percepts of state updates are used to update the beliefs. An agent doesn't know where it is at start-up before it has perceived its environment, and is thus in the *unknown* state.

5. The knowledge of the rooms (line 6) enforces the names of the valid rooms. Only rooms that are in the intersection of these names in the knowledge base and the perceived places on the map will be visited. Thus, arbitrary maps are allowed, but limit the movement options of the agent(s).

6.a The coloured blocks represent assets that should be retrieved with a certain priority. The searching element focuses on locating the blocks in their respective rooms. The rescue component then consists of moving them in the correct color order to the *DropZone*. An agent is the performer of this search and rescue operation.

6.b The analogy fails on the following points:
- The entire map of possible rooms is known at start-up, something not always possible in practical search and rescue cases.
- The priority of rescue tasks is set up front and doesn't change. Real-world examples of course have no such luxury to be able to predetermine the entire operation.

• The world is completely static and is thus only influenced by explicit actions of the agent(s). Again, not very realistic.

7. A possible performance characteristic could be minimizing the amount of so-called **misses**. A miss could here be defined as entering a room without picking up a block, putting down a block anywhere other than the *DropZone*, waiting until a room is no longer occupied or starting a path without actually ending up in the target room. Other options are counting the total number of visited rooms, the average distance travelled per route taken, ...

Part II

4. Using the original codebase, the initialization uses the adopt function to set the goals. As these are never fulfilled, no goals get removed, and the agent keeps looping over all locations. Goals can be removed in two ways: Automatically by just fulfilling the goal, or by dropping them manually.

7. The agent works flawlessly on every map.

8. The syntax is much less intuitive yet programs can be written much more elegantly and concise.

9. **Liked:**
• Both visual and textual feedback
• Queries and action to manually manipulate and question the beliefs and knowledge of the agent.
Can be improved:
• Prolog debugger, some sort of easy stepping within Prolog code.
• Having to manually restart the goal agent and server.

Part III

1. No efficient solving, the agents aren't aware of each other. They both try to fulfil their goals at the same time. The performance can be improved by sharing the discovery of a room and informing the other agent(s) every time a correctly coloured block is picked up.

Note: In the provided solution a few optimizations are implemented. Agents now use a breadth-first shortest path finding algorithm to select the best room to explore or fetch a block at, and let the others know they call dibs on said room or block. This has the effect that as soon as an agent starts fetching a block, the other agent(s) will go on and get the next blocks in the sequence, and if needed will wait with these at the *FrontDropZone* until their own turn(s).

6. As described in the note in section III.1, the agent(s) now work together without any issues, and completely as intuitively expected.