Intelligent Autonomous Agents in Quake

David Ganzhorn, William de Beaumont
University of Rochester
February 27th, 2004

Abstract

For several weeks we researched and developed agents that interact with Quake2UR, a version of Quake2 that has been modified by the University of Rochester graduate student Bo Hu to allow for a socket connection directly to a bot. This connection allows for external programs to command a bot around, thus allow the intelligent decision making to be off-loaded from the bot and into the external controller. Our research has been to develop intelligent and autonomous controllers in JESS, the Java Expert Systems Shell. We were supplied with an agent that randomly walked at right angles, did not interact with the environment, and did not listen for messages from the bot that it had been stopped short of the goal. We have developed two agents from this basis, both of which are capable of locating and acquiring nearby objects in the environment if they are reachable, reacting appropriately to being stopped, and moving freely through quake space. The first agent is a random walker, and the second is a wall follower. Also, we used the perceptual capabilities of the agents to make detailed and accurate maps of the world they were exploring.

Motivation

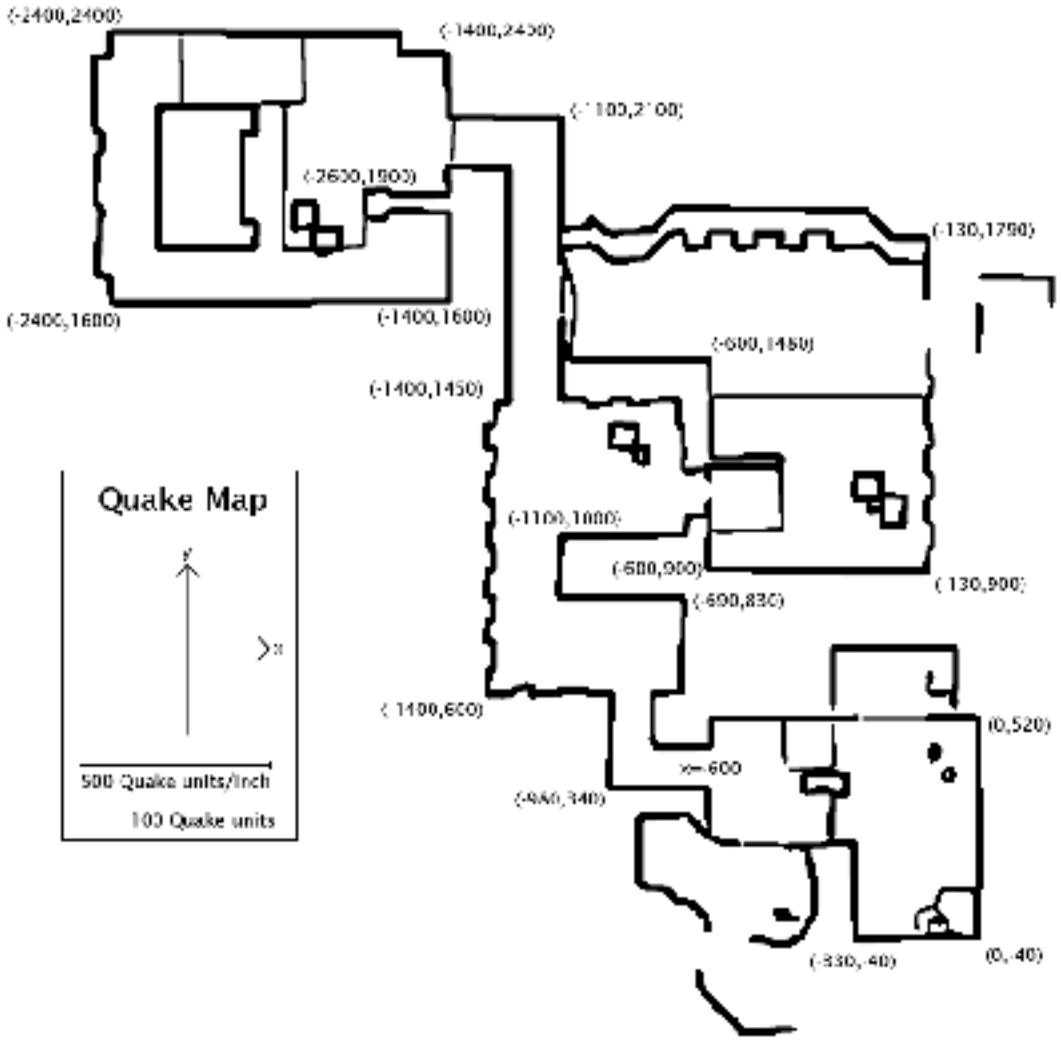
The primary motivation for this project was to explore expert

systems as a means of generating intelligent behavior for an agent that had to operate through complex 3d environments. Jess and Quake2UR were the only available tools to experiment with, and so the decision to use them was obvious. Our goal of creating intelligent navigational and item acquiring behavior through rules seemed challenging and proved to be so. The reason for mapping the Quake world was to both become more familiar with the environment and to have an accurate means of determining coordinates of different positions in quake world, so that we could spawn agents and items in the desired locations with ease.

Methods

The development of the two agents occurred after the creation of the mapping. We were introduced to the idea of using the bots to map quake space by Gregory Briggs and Rob van Dam, fellow students. After learning about this idea for mapping we shared with them the methods of greatly increasing the perceptual abilities of the bots, and thus we both benefited from the exchange. Rather than using the same program to accomplish the mapping, we wrote our own that would also plot the rays of vision of the bots in addition to the points that they saw. This resulted in both an outline of the space and filled regions within this space to more clearly indicate obstacles within the world, as well as to see the extent to which a bot explored a particular region, and from which vantage point. We then set loose several waves of random agents that each created their own maps, and then we composited the maps of

multiple agents to create a more comprehensive map. Eventually, one random agent performed extremely well in exploring the world and we were able to create a detailed map from it's observations alone. The following image is the map that was generated by the agent, and then processed by hand to have clearer outlines, as well as coordinate information. Any areas that do not have complete outlines are regions that are impossible for the agents to enter, but into which they can see. See appendix for the initial map.



Once this was done, we developed the two agents in JESS. The first was created by streamlining the movements of the original agent. The original agent was coded to only move at right angles, so by allowing the agent to move in any direction, our agent become twice as efficient in moving to any given point. Also, we implemented event handling, so that not only would our agent know to not initiate a new movement until its first movement was complete, but it would also be able to initiate a new movement immediately when the previous action ended. This allowed our agent to execute behaviors much faster, particularly when it was stopped immediately upon moving, by encountering a wall. Whereas several seconds would elapse before a new move would be initiated by the original agent, our new agent was capable of initiating new moves as frequently as every tenth of a second. In certain circumstances, our first agent was capable of moving ten to twenty times as fast as the original.

Before implementing item acquisition abilities, we created the second, more sophisticated agent. This agent was designed to implement basic wall-following behavior. Initially the agent consisted primarily of two rules; if the agent encountered a wall, it would turn by thirty degrees to the left until it was capable of moving. Then, the agent would move forwards at ten degrees to the right. This behavior resulted in the agent initially colliding with a wall, and then banging into and off of the wall as it followed it around in quake world. Unfortunately, the configuration of quake world prevented the agent from ever leaving the

initial room; it could not turn sharply enough to make it through doors. We considered making the turning radius much smaller, but this would have lead to enormous inefficiency as the agent collided with the wall every ten or so units, rather than every few hundred units. Instead, we decided upon adding an additional rule to the agent's behavior; whenever the agent had strayed from the wall on its right by too great of a distance, it would turn towards the wall and collide with it, and then continue on its way. This allowed for the agent to perceive doors and to pass through them, as well as to pass through winding tunnels. Once the third rule was implemented, our agent traversed the initial level of the world, where all testing was done, and reached the exit; the agent walked from his initial location, through two tunnels, an intersection of multiple rooms, and up stairs before he came to the end of his journey. Our agent proved robust and was capable of such sophisticated navigation under multiple starting conditions; the agent was deterministic and so the starting location entirely determined the ultimate behavior of the agent.

Lastly, we added item-acquisition behavior to each of the agents. Were they to perceive any items when they were checking out their situation, they would head straight towards the first one perceived and pick it up. If they were unable to directly reach it, then they would go back to regularly moving about.

Once we had implemented item-grabbing behavior, we moved on to comparison tests to see which of the two agents were quantitatively

superior in acquiring the most items in quake world. We randomly spawned 100 items in quake world by using a program that examined our created map, located areas that the agent had explored, and spawned items randomly throughout this space. We ran 8 trials for each of our two agents, where their score was the number of items they acquired during their journey, which lasted for the duration of their lifespan, approximately 500 seconds.

Results

Both agents were severely hindered by points in the quake world that bots are incapable of navigating out of; once reached, the bot is stuck in it for the rest of its lifespan. The random agents tended to be plagued by this problem more than the wall-followers. We assume this is because the random agent is more likely to try to wedge itself into corners. The exact scores of the agents were:

	Random	**Wall-Follower**
Trial 1	10	10
Trial 2	4	9
Trial 3	7	1
Trial 4	7	24
Trial 5	0	30
Trial 6	8	10
Trial 7	3	11
Trial 8	12	18
Total	51	113

Although we did not do test trials with the original agent, we predict that

it would have done extremely poorly in the test, as it is no more sophisticated in navigational abilities than our random walking agent, and it is vastly slower overall. Also, the lack of event handling would likely cause conflicts when the item acquisition behavior was added, as the agent might begin moving in another direction before it had acquired the item it was heading towards.

Discussions

The results make it clear that the wall-following agent is much more capable of navigating and acquiring items in quake world than the random agent. We believe this is due to the greater range of the wall-follower's travel in the same amount of time. It is easy to make an analogy from quake world to the real world, as both are continuous 3d spaces, and to see the value of more sophisticated navigational abilities for autonomous agents. If the agents were cleaning robots that were attempting to pick up as much trash as possible in an indoor area, it is probably safe to assume that the wall-following agent's behaviors would be much more efficient than the random agents behaviors.

With modifications to the item-grabbing behavior, the wall-follower could most likely do comparatively even better. One such modification would be for the agent to move to and grab the nearest item, rather than the item which is initially perceived. This would result in a smaller disturbance in the movement of the bots, and this would be of primary advantage to the wall-follower, who's movements are

organized, rather than the the random agent, who's movements are done randomly, without any goal. However, this only becomes a significant factor for situations in which more than one item at a time is perceived, which was rare in our simulations. Another modification that we think would make a much larger change would be for the agent to return to it's original facing and position after it had acquired an item, or even a string of several items. This behavior could easily be implemented with a stack of locations, and would allow for the navigational patterns of the agents to be unperturbed by the behavior of acquiring items. Again, this would only be of help to the wall-following agent. This methodology would allow for any new behavior to be added to the agents, without impairing their current navigating habits, although it would result in a substantial amount of backtracking. We would like to continue researching better navigational patterns that would both me more effective in and of themselves, and that would be capable of being unperturbed by other behaviors of the agent.

Appendix

A1. JESS rules for the random-walking quagent:

Causes the bot to randomly select a new direction and distance to walk.

```
(defrule find-new-behavior-walk
        (idle)
        =>
        (retract ?*current-goal*)
        (bind ?*current-goal*           (assert (goto)))
        (bind ?*goto-d* (?*quake* randInt))
        (bind ?*goto-a* (?*quake* randInt360))
)
```

If the agent perceives an item within 100 units, then activate item-getting behavior. Otherwise just move again.

```
(defrule look-around
        (get-situation)
        =>
        (bind ?*current-situation* (?*quake* probe 100))
        (retract ?*current-goal*)
        (if (= (?*quake* interestingItems ?*current-situation*) 0)
                then (bind ?*current-goal* (assert (idle)))
                else (bind ?*current-goal* (assert (go-find))))
        (printout t "-scan-" crlf)
)
```

Determine which item is preferable (currently implemented to be the closest) and calculate the angle and distance to the item, then go get the item.

```
(defrule find-items
        (go-find)
        =>
        (bind ?*item-name* (?*quake* bestItemSeen ?*current-situation*))
        (bind ?*Location* (?*quake* whereAmI))
        (bind ?*goto-a* (?*quake* angleToItem ?*current-situation*
                                                ?*item-name* ?*Location*))
        (bind ?*goto-d* (?*quake* distToItem ?*current-situation*
                                ?*item-name*))
        (retract ?*current-goal*)
        (bind ?*current-goal* (assert (getitem)))
)
```

Move to the item and pick it up if it is reachable, if not then become idle (which leads to moving).

```
(defrule get-item
        (getitem)
        =>
        (bind ?*return-value* (?*quake* turn ?*goto-a*))
        (bind ?*return-value* (?*quake* walk ?*goto-d*))
        (bind ?*stopped-distance* (?*quake* stopped))
        (if (< (- ?*stopped-distance* ?*goto-d*) -15)
                then    (retract ?*current-goal*)
                        (bind ?*current-goal* (assert (idle)))
                        (printout t "Stupid wall!" crlf)
```

```
                else    (retract ?*current-goal*)
                        (bind ?*current-goal* (assert (get-situation)))
                        (bind ?*return-value* (?*quake* pickUp ?*item-name*))
                        (printout t "I picked up " ?*item-name* crlf)
                        (bind ?*item-count* (?*quake* countInventory))
                        (printout t "I now have " ?*item-count* " items!"
                                                                        crlf)
        )
)
```

Move, then become idle.

```
(defrule need-to-move
        (goto)
        =>
        (bind ?*return-value* (?*quake* turn ?*goto-a*))
        (bind ?*return-value* (?*quake* walk ?*goto-d*))
        (bind ?*return-value* (?*quake* stopped))
        (retract ?*current-goal*)
        (bind ?*current-goal* (assert (get-situation)))
)
```

A2. JESS rules for wall-follower agent

Head slightly to the right, which should eventually reach a wall.

```
(defrule approach-wall
        (idle)
        =>
        (retract ?*current-goal*)
        (bind ?*current-goal*           (assert (goto)))
        (bind ?*goto-a* -10)              ;(?*quake* randWander)
        (bind ?*goto-d* 50)              ;(?*quake* randInt)
)
```

Turn to the left thirty degrees, and move forward, in hopes of getting away from the wall.

```
(defrule avoid-wall
        (at-wall)
        =>
        (retract ?*current-goal*)
        (bind ?*current-goal*           (assert (goto)))
        (bind ?*goto-a* 30)
        (bind ?*goto-d* 50)
)
```

Look to see if there are any interestingitems around. If so, initiate item-grabbing behavior. If not, and we are too far awhile from the wall on the right, then go to the wall on the right. Otherwise, just become idle (which leads to moving).

```
(defrule look-around
        (get-situation)
        =>
        (bind ?*current-rays* (?*quake* scan 4))
        (bind ?*dist-right-wall* (?*quake* distToWall ?*current-rays*))
        (retract ?*current-goal*)
        (bind ?*current-situation* (?*quake* probe 100))
        (if (= (?*quake* interestingItems ?*current-situation*) 0)
                then (if (>= ?*dist-right-wall* 50)
```

```
                        then (bind ?*current-goal* (assert (wall-bang)
                        else (bind ?*current-goal* (assert (idle))))
                else (bind ?*current-goal* (assert (go-find))))
        (printout t "-scan-" crlf)
)
```

Determine which item is preferable (currently implemented to be the closest) and calculate the angle and distance to the item, then go get the item.

```
(defrule find-items
        (go-find)
        =>
        (bind ?*item-name* (?*quake* bestItemSeen ?*current-situation*))
        (bind ?*Location* (?*quake* whereAmI))
        (bind ?*goto-a* (?*quake* angleToItem ?*current-situation*
                                                ?*item-name* ?*Location*))
        (bind ?*goto-d* (?*quake* distToItem ?*current-situation*
                                                        ?*item-name*))
        (retract ?*current-goal*)
        (bind ?*current-goal* (assert (getitem)))
)
```

Move to the item and pick it up if it is reachable, if not then become idle (which leads to moving).

```
(defrule get-item
        (getitem)
        =>
        (bind ?*return-value* (?*quake* turn ?*goto-a*))
        (bind ?*return-value* (?*quake* walk ?*goto-d*))
        (bind ?*stopped-distance* (?*quake* stopped))
        (if (< (- ?*stopped-distance* ?*goto-d*) -15)
                then    (retract ?*current-goal*)
                        (bind ?*current-goal* (assert (idle)))
                        (printout t "Stupid wall!" crlf)
                else    (retract ?*current-goal*)
                        (bind ?*current-goal* (assert (get-situation)))
                        (bind ?*return-value* (?*quake* pickUp ?*item-name*))
                        (printout t "I picked up " ?*item-name* crlf)
                        (bind ?*item-count* (?*quake* countInventory))
                        (printout t "I now have " ?*item-count* " items!" crlf)
        )
)
```

Head directly towards the wall on the right, and move as far towards it as it is distant.

```
(defrule go-to-the-wall
        (wall-bang)
        =>
        (retract ?*current-goal*)
        (bind ?*current-goal* (assert (goto)))
        (bind ?*goto-a* -90)
        (bind ?*goto-d* ?*dist-right-wall*)
)
```

Just turn, then assert that we need to walk.

```
(defrule need-to-move
```

```
        (goto)
        =>
        (bind ?*return-value* (?*quake* turn ?*goto-a*))
        (retract ?*current-goal*)
        (bind ?*current-goal* (assert (walk)))
)
```

Walk, and if stopped short, then assert that we are at a wall, otherwise get the situation.

```
(defrule walkin
        (walk)
        =>
        (bind ?*return-value* (?*quake* walk ?*goto-d*))
        (bind ?*stopped-value* (?*quake* stopped))
        (bind ?*stopped-difference* (- ?*stopped-value* ?*goto-d*))
        (retract ?*current-goal*)
        (if ( >= (abs ?*stopped-difference*) 30)
                then (bind ?*current-goal* (assert (at-wall)))
                else (bind ?*current-goal*(assert (get-situation))))
)
```

A3. Unaltered map , generated from the mapping program and bot.