Authors:

Michael Rotondo
- Undergraduate, Computer Science, University of Rochester
- mr001m@mail.rochester.edu

Jon Ruskin
- Undergraduate, Computer Science, University of Rochester
- jr003m@mail.rochester.edu

David Sloan
- Undergraduate, Computer Science, University of Rochester
- ds002m@mail.rochester.edu

Agent creation through the use of production systems lends itself to agents of all abilities. Our group decided to create a mapping quagent to scout, explore and map areas for the future use of itself and other bots. These maps can also be useful to human observers as well. We started our mapper using a discrepancy system which proved to be too difficult to debug. From this we went to converting the map into a cellular environment like Wumpus World. Using this cellular environment the quagent explored the area using a depth-first state space search. Each state represented a cell in the world that the quagent could be in. After every subsequent scan the quagent stores its current image in a .jpg file for the viewer to look at.

Keywords:

Artificial Intelligence, Production Systems, Quagents, Agent Manipulation, QuakeWorld, Mapping, State Space, Depth-First Search

To begin the project, the first thing we did was to create a suitable test environment to observe our created quagents in action. To accomplish this task we used Quark, a Quake level creator/modifier. We created a basic square room with a few

distinguishing features. First is that around the top of the room there is a “balcony”, an area that human players can climb to (all the walls have ladder characteristics and are climbable) and see the entire open area below. This will be used as an observation deck to discern the patterns of the quagents and monitor their performance. For consistency, the upper area will be called the observation deck, and the lower area the arena. Another feature is that walls are placed down the center of each wall creating a “+” through the arena. These walls cut the arena into 4 smaller rooms that are connected by holes in the walls, the “doors”. Each door is at a different place in its respective wall for variety.

This environment is large enough that the walls in the arena can be changed to turn it into a maze or many other desirable configurations of walls and objects. This can be done with ease and will not hinder the quagents ability to move around. Basically we have created an environment that is an excellent debugging tool for monitoring quagents.

Our next task was to map our environment. To accomplish this task we loaded up our map into Quake and ran around to the four corners of each of the rooms. We also ran to each of the inner and outer points for the doors and marked their coordinates. Finally, we climbed to the observation deck and plotted the corners of this area. This led us to develop a nice map of the area. An important idea to note at this point is that we control the environment that the quagent operates in. While we do not control the quagent itself, at times during creation and testing we stack things in favor of the quagent. We want the little guy to succeed.

The results of the plotting are:

Areas			Upper-Left	Upper-Right	Bottom-Left	Bottom-Right
	Arena		(-495, 495)	(495, 495)	(-495, -495)	(495, -495)
	Observation Deck		(-591, 623)	(591, 623)	(-591, -623)	(591, -623)
Doors			Inner	Outer		
	Top Door		(0, 208)	(0, 271)		
	Bottom Door		(0, -304)	(0, -366)		
	Left Door		(-176, 0)	(-239, 0)		
	Right Door		(112, 0)	(175, 0)		

The first thing we did in creating a new quagent was to look at the example quagent files provided on CSUG. In particular we looked at the DrunkenMasterII files, the QuagentDemoII file and the BasicQuagent file. This was particularly important because for the DrunkenMasterII Jess file, it was our first real experience looking at Jess code in action. This was a great help in understanding how Jess worked.

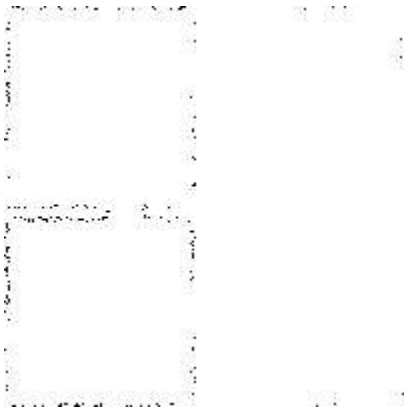
Our next action was to expand the abilities that the quagents could perform from Jess. Mainly this involved creating new java classes for all the new query and response actions, adding methods in the quagent's java class to include these abilities, and modifying the BasicQuagent's getResponse() method to return the new response types. For our first quagent we chose to take the BasicQuagent and DrunkenMasterII files and build off them. This helped us focus on expanding the quagent's abilities and creating our own quagent, which we lovingly called ColumbusBot (CB). It is more effective to build up skill than to jump directly into the thick of code.

The abilities that were added for the quagents were to get the inventory, get world location and information, stand, pickup and drop objects, use rays, and get the wellbeing of the quagent. These are most of the standard abilities for quagents though. We also decided that it would be useful to have a few extra abilities. We created a method that

allows a quagent to develop a limited knowledge base of its surroundings based on rays. Each object gathered from a ray has coordinates which are put on a chart and later displayed as a jpg. Making a quagent walk around and turn full circles every now and then turns a quagent using this mapping utility into a mapping quagent. This might be useful for later models to discern where walls are and creating its own working knowledge of its environment. Another useful ability added to the quagent is the ability to turn to a specific heading to face an object knowing only the quagents x and y coordinates and the objects x and y coordinates. This brings the four step walk-turn-walk-turn aspect of the DrunkenMasters into a two step process of turning and walking. The only hindrance to this movement is that the quagent must have the second set of coordinates to turn to for this method to work.

At this point there was a decision to be made in the group, namely what our first quagent's function would be. Since the mapping ability was already ready to be used we decided to create a mapping agent as our first quagent. The first step to create this quagent was to first make sure that at a given point the quagent could map an area properly. Simply we wrote an algorithm that had the quagent turn (360/ # of rays) degrees, one degree at a time, so that every angle would be mapped. At this point we had a quagent that would properly map it surroundings once, but it was stupid and did not know what to do afterwards.

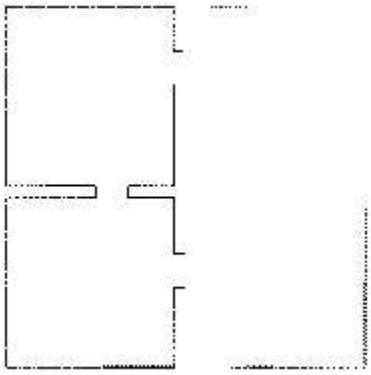
This is the first real map that our quagent returned. Each dot is a returned ray, fairly outlining the walls. Since the quagent was located on the left hand side of the map, it was unable to see most of the right hand side, but a few rays did manage to get through, so you can infer that the total shape of the map is a square. During this trial we controlled the movement of the quagent and directed where it would shoot rays at.



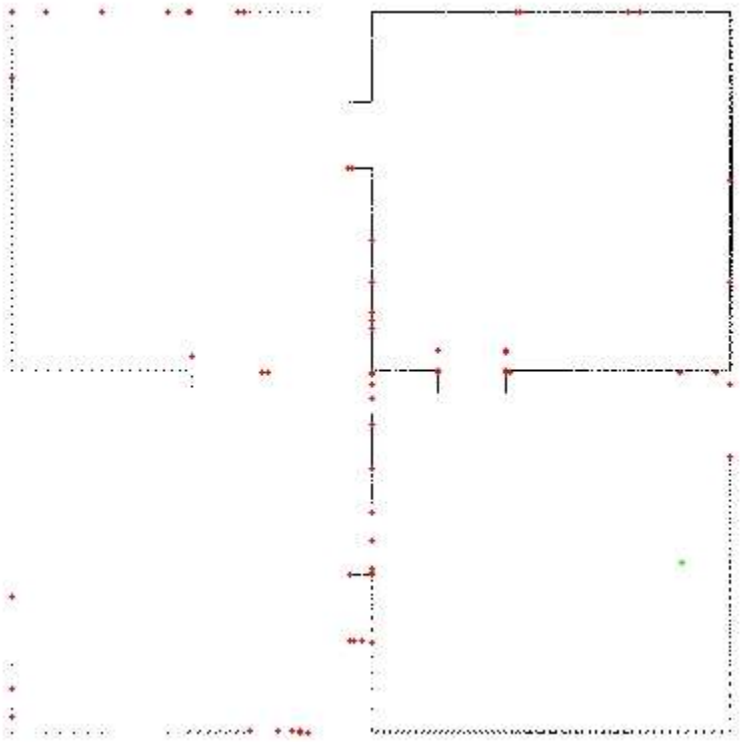
The next step involved created discrepancies which in this context we defined as rays that were sent out an angle apart returned points on walls that had great differences in their x or y coordinates. A discrepancy class was created for each of these that found an absolute angle and distance to walk such that after turning to that angle from 0 degrees and walking the set distance would put the quagent into a position to map again. This class was essentially created to find doorways that the quagent could go through and say how the quagent could get to them. This solution created an agent that would travel from discrepancy to discrepancy and thus from doorway to doorway until it found none more and map its surroundings each time. This method of exploration however proved too difficult for us to solve, as it involved the quagent running headfirst into walls repeatedly. We could not figure out a way to successfully avoid this bug, so we decided to scrap this process altogether.

The following map was created during the time when the quagent was moving on its own, but due to some rounding errors and some bugs was not able to fully determine the correct angle to travel at to explore the next area. Often times the quagent had to be

spawned in positions that made it easier to get to the center of a discrepancy. This was before we realized it would be too difficult to get the discrepancies system to work.

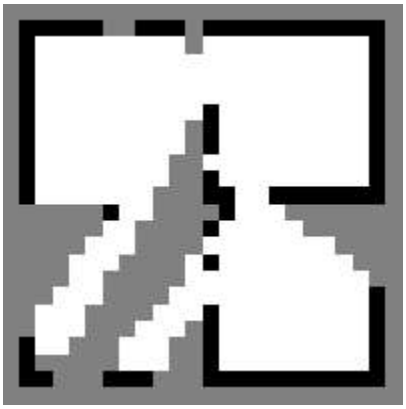


The next map shows the use of colored “flags” on the map to show the points of discrepancy. Again, since this is entirely based on discrepancies, it does not entail our final solution map. This was one of our final maps created using the ColumbusBot.



Before we scrapped the direct-movement-to-exploration idea we decided to try another approach involving turning the Quakeworld into a cellular world like that of Wumpus World. The idea was that if a quagent could move directly at a target without getting into a cell that contained a wall, then he should be safe in avoiding walls. This too did not work out terribly well. While the idea to move directly to a point was discontinued and the ColumbusBot was put out of commission, the cellular environment gave way to our next generation of explorers, the VespucciBot.

The following map shows the cellular environment seen during its use by ColumbusBot. The granularity depicts that if a spot inside the cell was seen, then the entire cell was known.

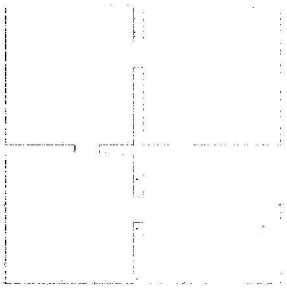


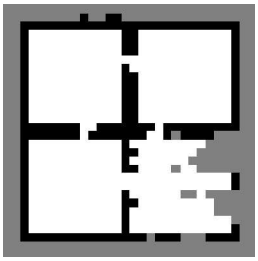
Before we discuss the VespucciBot however, there is another important aspect of the ColumbusBot that carried over. When we started to extend the quagent's discrepancy-to-exploration ability we needed to have some way not only to get the quagent to scan, move and scan again, but to remember the places that were scanned previously and then go to them afterwards. Treating each discrepancy as points on a tree we created a depth first search traversal of the tree, where new branches are constructed on the tree when a node is evaluated. When a branch is exhausted then the agent goes back to the position it was at previously and explores its next discrepancy. When all

discrepancies have been explored, the quagent considers its task completed. This state traversal was an essential idea behind the VespucciBot.

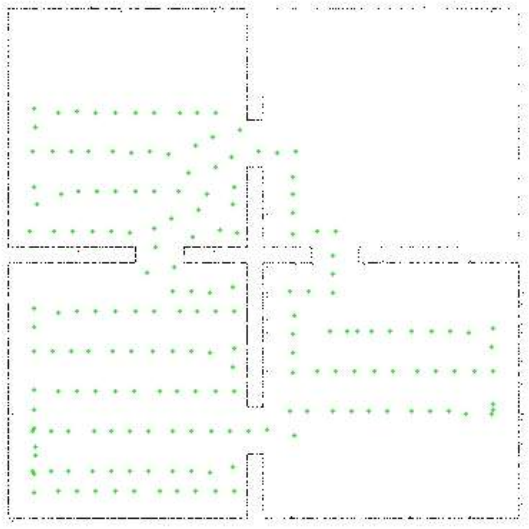
The main idea of the VespucciBot was this – if our world was now cellular and we could interact with it like the agent in Wumpus World, why not act like we were in Wumpus World and perform a depth-first state space search. This way we could ensure that we would go to every cell and spamming out rays at these cells could tell where all the walls were. Also, by taking a single movement at a time and scanning only a few directions then the transitioning between the movements would be swift and easy and it would just seem as if the quagent were walking and briefly looking around. VespucciBot only uses 4 rays at any given time showing that even with a very limited ability to gather information much information can be gathered.

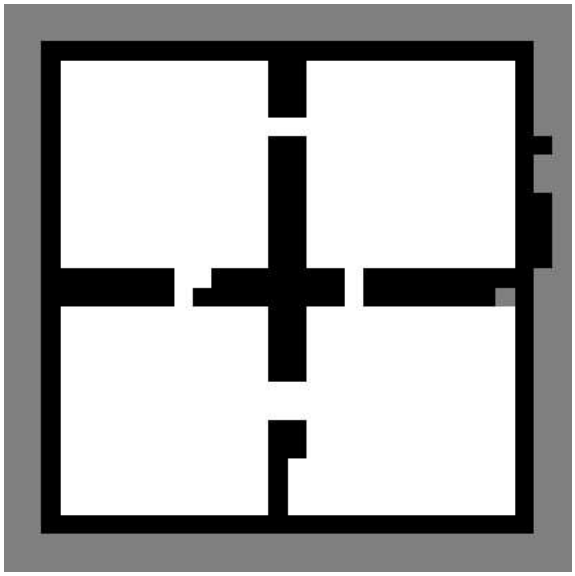
The following two maps are from an early VespucciBot before optimizations. The first shows the dot-map creation which looks nicer but is much harder for the quagent to use to decide where to go next. The second is the cellular map, which while much more granular is easier for the quagent to use as where is legal to move and where is not.





We found that Vespucci took too long to map the world and died in the process. As a result of this we optimized its code for searching so that less actions would be taken and the quagent would survive longer. This allowed the quagent to make a successful run of the environment and get a good set of maps. The following maps are examples of the final revisions on the quagent. In the dot map, the green dots represent where the quagent walked to. You'll notice that its path is very meandering and could be optimized, but in this form the quagent is successful at mapping its environment.





We considered this a valuable accomplishment and decided that the results that were most useful to the quagent, the cellular map, could be extended to other quagents as well. Essentially what we have done now is made a quagent that maintains a knowledge base of its surrounding environment, and acts on that knowledge. It might be a very interesting aspect of quagents if another object were created that contained the quagent's map. A quagent could then sell his knowledge base (his map of the environment) to other quagents. You could also allow a quagent to start its life knowing the environment by setting his internal map to a previously created map.

To complete this project we received help from and thank the following sources:

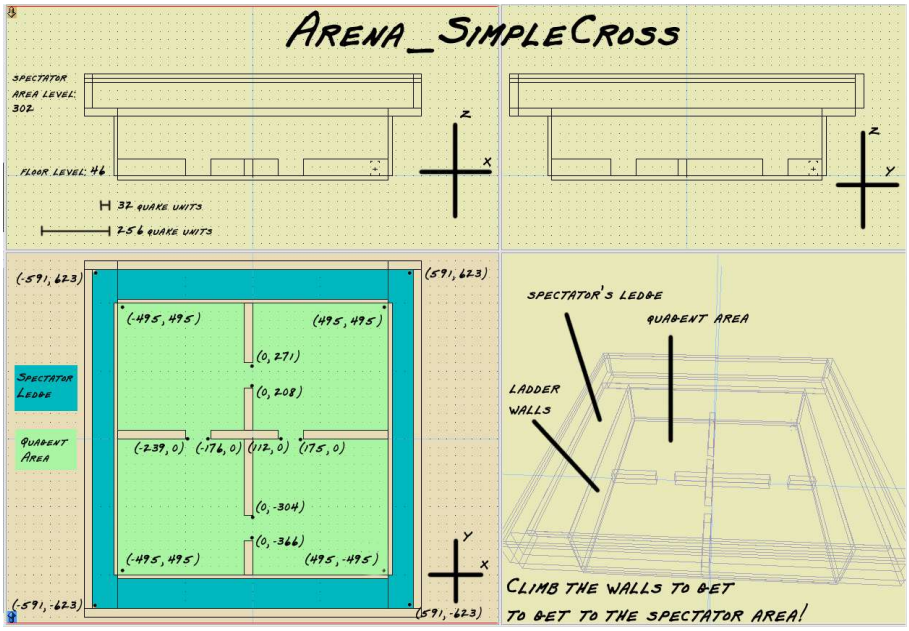
-University of Rochester CSUG for Quagent Protocols, Quagent files, a basic guide to making a quagent and providing the Jess user guide.

-Peter Barnum for a succinct new-user guide to Jess.

-Amerigo Vespucci and Christopher Columbus for lending their names to our exploration quagents.

Addendum A –

"Hey Mike, spawn another one!"



Our environment