Hirano, Derek
Munar, Mark
Ogata, Branden
12/17/10
ICS 461

Project: Our-World

Overview: For the final project, our group chose to build on the AIMA code. The base code comes from the wumpus files, along with inspirations from modifying the vacuum files. The intended product is a room where multiple agents interact among themselves. There are multiple agent types, with each type having special characteristics and abilities. Simulations are done on various environment setups, and their results were compiled and analyzed.

Description: In the wumpus world, the hero had to retrieve gold somewhere in the world. Only armed with a single arrow, the hero had to avoid falling into pits and being eaten by a stationary wumpus.

In our world, the hero is a selfish elf called C, who has a limited number of arrows. C is joined by a human with a sword called A. C has a long range forward attack, while A can either target one or multiple adjacent tiles (left, right, forward) at once. A and C have supposedly superior weapons to forcibly take the gold away. Ideally, A and C should work together to find the gold.

The stationary wumpus is now replaced by multiple mobile dwarves who call themselves B. B's may start off with alcohol and will merrily drink away. The magical alcohol is a special mixture that may allow a B to respawn if killed. Once they run out of liquor, they will wander around, until they see A or C. B's can steal the life of A or C (even both at once) if an alive B can get into the same cell as A or C. B's like shiny objects and are called G once they possess gold.

A simulation terminates when either an alive A or an alive C retrieves the gold, when A and C both are incapacitated, or the number of maximum steps is reached. For each simulation step, a score is assigned to each agent. All agents start at 0. Each simulation step automatically incurs a -1 penalty. Picking up the gold adds 1000, but losing the gold removes that 1000. Death gives a -10000 penalty, but respawning gives back 10000.

Procedure:

1. Unzip .zip file, load up terminal, and change to code directory.
2. Start up LISP (preferably clisp, though Allegro has been tested on also).
3. (load “aima”)
4. (aima-load 'abcg)
   - may have to ignore error about kill function in a wumpus.lisp file
5. (load “proj.lisp”)
6. (start)

Tests: There are multiple variables for our-world, and each one needed to be tested. During development, patterns and tendencies were observed. Some, but not all of these, were formally tested. The following list is a compilation of the formal tests:

- Respawn rate for B
- Alcohol versus no alcohol for B
- Number of arrows for C
- Different room specifications
  - Square versus rectangular corridor
  - Small (3x3 → 1x1)
    - One A, one B, one C, and gold in same tile
  - medium (8x8 → 6x6)
    - Standard case with one A, one C, and multiple B's
  - larger (9x9 → 7x7)
  - Obstacles and pits
    - random placement
    - placement in a line across room
      - continuous versus one hole in line (bottleneck)
- Operation precedence
  - Grab gold first or attack/evade first
- Simulation mechanics
  - Effect of design choices (e.g. A and C with range but no zero-range attacks)
  - Effect of actions executing "at the same time"
  - Study of simulation shortcomings and bugs (e.g. limited percepts, how tiles are handled)

Analysis:

- Respawn rates for B: B wins if the respawn rate is high enough (closer to 1), and B loses if the rate is too low (approaching 0). (See Branden folder for raw data and visual representations) For the standard 6x6 room, it appears that around a respawn rate of 1/6 to 1/8 is a tipping point.
- Alcohol versus no alcohol: At the beginning of the simulation, if A or C is next to B's, alcohol makes the difference. B's will choose to drink first if they have alcohol, giving A or C the chance to attack. If B's don't have alcohol, they are able to kill A or C first because the same-tile-kill check comes before A or C's actions.
- Number of arrows for C: For higher respawn rates for B, C's arrows are quickly used up. Since C can only use and kill one agent at a time, there is no need for more arrows than the number of simulation steps. C does not evade that well when it runs out of arrows. For many, many observed trials C ended up killing A.
- Different room specifications:
  - Room size: For the 3x3 room, since A or C have no same-tile ability, B is able to kill A and C. For larger rooms, A and C still have a difficult time. Very large rooms were not tested.
  - Obstacles: For the 8x8 room, it appeared that having 25% or less of the remaining 6x6 space being obstructed did not change the outcome much. When the frequency was half of remaining space, movement was severely hindered. Random pits did kill agents when trials stated. An interesting case was when a wall and a B was in the same tile. A could only perceive the wall and did not attack. The continuous line of walls worked as expected (no deaths) since A and C were separated from B's. A did not defend bottlenecks correctly, and it resulted in B's getting through and killing A or C. For one test, A and C ended up moving to the side where all the B's were.

  - Pits: The pits were intended to prevent movement, but not prevent ranged attacks. C could be able to shoot across pits. Due to how the code was structured, C could not "see" beyond pits, as the forward sight function was structured similar to the shoot function (keep going until it encounters an object). The result was that pits were just like walls.
- Operation precedence: One room was setup so that B's were setup in A's swing path, with gold in A's cell. Another room was setup so that B was right in front of C, with gold in C's cell. In the original A and C cond block, picking up the gold has a higher precedence than attacking/evading. Changing the cond to attack first then pick up the gold did not make a difference because of simulation mechanics (order of action execution).
- Simulation mechanics: Since A and C end up dying fairly quickly, there was no real need to extend max simulation steps beyond 10. The order that checks and actions are executed end up favoring B. update-fn is executed before agent actions. update-fn includes the check to kill A or C if B is in the current cell. The tiles appear to use a queue, and the effect is that actions such as swing or shoot only affect objects at the front of the queue. C can only kill one B in one tile per step. A can kill at max exactly one B per tile per step. However, B can kill two agents per tile per step. The limited percepts led to situations were A or C and a B end up in the same tile. Once B gets into the same tile, neither A nor C can prevent death. Due to the nature of the AIMA code, there is the "zombie" effect, where an agent moves one tile after being killed. The reason is that for each simulation step, the actions are executed without regard to what other agents may have done. For example, A chooses to swing at a B, and that B chooses to move to another tile. The result is a dead B moving to another tile.

Conclusion: In the design of our-world, our group originally thought A and C would dominate B because of C's range and A's multiple tile attacks. We countered these advantages by giving B multiple instances and the ability to respawn. As the results of our testing has shown, B's are actually overpowered because of respawning, limited percepts for all agents, number of kills per tile, and luck. A is at the most disadvantage because it must get close enough to B's and risk getting killed by B's in its blind spot and respawning B's. The respawning ability also reduces the effectiveness of C's limited arrows. A and C only seem to win when B's do not respond as much or are busy drinking their alcohol. For the simulation, how the code was written led to various unintended and unexpected behaviors. It would certainly have helped if A and C had the code to cooperate with each other, to prevent friendly fire incidents, but our group was not able to implement this feature in time.