COS 402 – P5

1.
   In general the mouse appears to be behaving intelligently by successfully avoiding the cat for all three types of cats. For the oblivious cat the mouse stays at least one square away from the cat and goes after the cheese while maintain this distance. Additionally, when there are no cheese on the board the mouse will tend to stay on or near (depending on the cat's position) the cheese square closest to the mouse hole because the mouse "knows" it is the safest cheese square. For the unrelenting cat, the mouse stays inside the mouse hole until the cat is sufficiently far away enough (for the mouse to reach a cheese square and return) before venturing outside the mouse hole. For the sporting the mouse is less conservative, but will go back to the mouse hole if it appears the cat is chasing it.
2.
   The mouse's intelligence comes from the algorithm finding the policy that optimizes utility. In other words, the algorithms will dictate the move that puts the agent (the mouse) in the best position possible. In general, this means the algorithm will dictate the mouse move in such a way that always avoids the cat, but gets the cheese if it is possible, given what the cat will do. The mouse itself does not possess any "true" intelligence, since it is only making moves that have the highest payoff which is computed by a mathematical formula. The mouse isn't necessarily "thinking" or deliberating about its decisions available, instead, it knows that a certain choice should give the best expected utility.
3.
   No, it is would not be possible on average for me to do a better job of getting cheese while not being eaten. It is important to emphasize on average because, in practice, I might be able to luckily guess when the cheese appears and where and go after it, thereby getting a better payoff than the algorithm (since the algorithm doesn't guess). However, on average the best I can do is do as well as the algorithm, but not better. This is because the algorithm has inherently computed the policy that improves the expected utility. Therefore, following the algorithm in the long term will statistically yield the highest utility. I can easily choose actions that avoids getting eaten, but maybe not choose the best actions that yield the most cheese since mentally I might not be able to compute the probabilities correctly, whereas the algorithms do.
4.
   In the real world the cheese would not appear on only three squares and a cheese piece would disappear once the mouse lands on the square with it. Additionally, in the simulated world the mouse knows exactly where the cat is which is not necessarily true in a real-world situation. The mouse also knows which squares the cheese can appear on. In a real-world situation the mouse might only be able to perceive the cat in certain situations (e.g. a few squares away). Additionally, the state-space and action-space could change on given situations. For example, some squares might not be reachable (a fence/barrier could be put up), or the mouse might be able to choose to move two squares or one (if it's on surface which it can get a good grip). There may also be some kind of progression the state space. For example, the cat might learn where the mouse hole is or where cheese tends to appear, in which case the cat tends to gravitate

towards those areas, making certain state unreachable. In general a real-world situation could have random occurrences that change the state or action space.

The main difficulty to "scale up" the MDP-based approach to more realistic situations would first be memory limits to store the transition probabilities and the next reachable states in the cases where adding more states resolves the situation. For random occurrences that change the state space it becomes much more difficult. For example, let's say normally the mouse can more up to three spaces at a time (one, two, or three). But, with some random probability there is a spill and the mouse can only move one space at a time. This would be a random change in which actions are possible. A simple change would be to add two distinct states (with some random probability a state will contain a spill) this allows for a distinct action-space for each state. The problem with this is it increases the memory usage dramatically. The other major problem is a situation in which the cat learns. The main difficulty is then the MDP-based approach needs some way of keeping track of time (e.g. the cat might begin to learn after the mouse has gotten five pieces of cheese). The problem is then trying to figure out how to encode a change in the cat's behavior based on the mouse's behavior.

5.

Using a sporting cat, ValueIteration uses 368 iterations to converge while PolicyIteration uses 10 iterations with a combined 3842 PolicyEvaluation iterations to converge. In general it seems both algorithms are relatively similar in this regard. This is because ValueIteration iterates over all actions for each state whereas PolicyEvaluation does not need to iterate over the possible actions (since the actions are passed in as an argument). When this is taken into account (by multiplying by the 9 actions at each state) ValueIteration does a very similar amount of calculations. In this case ValueIteration does slightly better. When the discount rate is decreased ValueIteration has 12 (9*12=108) iterations and PolicyIteration has 75. In this case it would appear that PolicyIteration is slightly better. To generalize, it appears ValueIteration performs better for higher discount rates, while PolicyIteration performs better for lower discount rates. Based on the algorithms themselves ValueIteration is more complete than PolicyIteration. ValueIteration computes a policy by converging utilities, whereas PolicyIteration does so by converging utilities based on a policy and improving the policy from the converged utilities. If ValueIteration ends early (before it converges), the computed policy can be completely off while PolicyIteration is constantly improving the best policy. In this sense, early termination would be better for PolicyIteration. However, ValueIteration is more complete in the sense that it computes the exact utility and calculates a policy based on this.