# Project 3: The Meanest Sheep in the World Who Is Also Near Sighted

You are a sheepdog robot, and you are responsible for corralling the meanest sheep in the world. While most sheep you're able to herd and guide where they're supposed to go, this sheep is big, mean, and hates robots. If the sheep sees you, it will try to charge you, and if it hits you, it will absolutely obliterate you - a combination of poor craftsmanship, perhaps, and the fact that this sheep just works out thinking about ways to destroy robots.

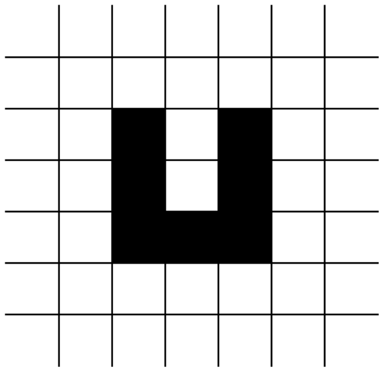
But you have two advantages - one, you're more agile and can move in ways this sheep cannot; two, the sheep is somewhat blind, and if you get outside its field of view it will lose interest in you.

That being so, you come up with the following strategy: perhaps you can trick the sheep into chasing you, and in doing so lead the sheep into the pen. But that is going to require planning, to know what to do (where to move) when the sheep moves, etc. And, being scared of this sheep, you'd like to get it into pen as fast as you reasonably can.

## 1 Implementation

### 1.1 The Field

The grazing field for this project is a 51 x 51 square grid. The robot and the sheep can occupy cells in the grid and move between them. The middle square in the grid is surrounded by a U of blocked cells, as below:



This U is the pen - the goal is to get the sheep into that center node (at which point your ally will slam the gate, trapping the sheep).

### 1.2 The Sheep

- The sheep occupies one of the cells. It can move up/down/left/right, or stay in place.
- The sheep cannot move out of the 51x51 grid, or into any of the blocked cells.
- The sheep's field of view is the 5x5 block of cells centered on the sheep's position.
- If the robot is not within the 5x5 field of view, the sheep will pick uniformly at random between any of its currently valid actions.

- If the robot /is/ within the 5x5 field of view, the sheep will pick randomly between actions that take it closer to the robot (charging the robot).
- If the sheep enters the center cell, at the middle of the pen, the sheep is trapped and the game is over.

### 1.3 The Sheepdog Bot

- The robot occupies one of the cells. It can move to any of the 8 neighboring cells (if unblocked/empty), or stay in place.

## 2 Sheepdog Bot 1

We can consider the optimal sheepdog bot in the following way: letting *state* be the current configuration of the field (bot and sheep positions), define $T^*(\textit{state})$ to be the expected number of moves an optimal bot would need to catch the sheep. If the sheep cannot be captured for a given state, $T^*(\textit{state}) = \infty$.

What states are easy/immediate to determine $T^*(\textit{state})$ for?

For any other given state *state*, express a formula for $T^*(\textit{state})$ in terms of a) the actions the bot can make, b) how the sheep responds, and c) what state results. What are special cases you ought to consider?

If you could determine $T^*(\textit{state})$ for each state, how would you determine the optimal action for the sheepdog bot to take?

Compute $T^*$ for every state. If the sheep starts in the upper left corner, and the robot starts in the lower right corner, what's the expected number of moves needed to capture the sheep?

If the robot can start at any location, and the sheep will be introduced at a random empty spot - where should the robot start? Why? What is the expected number of moves to catch the sheep, when started in this locaiton?

Based on $T^*$ and the optimal actions, simulate the sheepdog bot / sheepdog interactions. How does the simulated data compare to the compute expected value?

## 3 Sheepdog Bot 2

The Sheepdog Bot 2 is based off Sheepdog Bot 1. Note that for sheepdog bot 1, you need to calculate potentially as many as 2550 different values to fully get $T^*$ and the optimal policy. This is potentially very expensive, memorywise. For this bot, fit a model $\tilde{T}$ to input/output pairs $(\textit{state}, T^*(\textit{state}))$.

How should you represent states as input to the model? What kind of model do you want to consider? What input features are relevant? What kind of error or loss are you using to assess your model? What training algorithm are you using? How can you compare the size of the fit model to the size of the fully computed $T^*$?

How can you assess the quality of your model fit? Is overfitting an issue? If so, how can you avoid it?

Sheepdog Bot 2 functions in the following way: in any state, the bot estimates the value of a given action using $\tilde{T}$ to estimate the number of moves remaining to catch the sheep after taking an action; the bot then chooses what action to take based on the smallest estimated moves to catch the sheep.

Simulate Sheepdog Bot 2, starting the sheep in the upper left corner, and the robot in the lower right corner. Does Sheepdog Bot 2 reliably catch the sheep? How does it compare in its performance to Sheepdog Bot 1?

**Bonus:** Improve the model. Does that improve Sheepdog Bot 2?

## 4 Sheepdog Bot 3

Sheepdog Bot 3 learns by watching. Using Sheepdog Bot 1, you can generate a lot of data, of the form (*input state*, *output action*). Build a model to predict the desired output action from the specified input state.

What is your input space? What is your output space? What is your model space? How are you assessing the error, and how are you reducing that error - i.e., what is your training algorithm?

How can you assess the quality of your model fit? Is overfitting an issue? If so, how can you avoid it?

Simulate Sheepdog Bot 3, starting the sheep in the upper left corner, and the robot in the lower right corner. Does Sheepdog Bot 3 reliably catch the sheep? How does it compare in its performance to Sheepdog Bot 1?

## 5 Analysis

The usual! Write up your algorithms, ideas, results, and the answers to all the above questions.