Report of Question 1

Zekun Zhang (zz364)

zekunzhang.1996@gmail.com

1. Compute a strategy for controlling your dog bots that attempts to minimize the number of rounds needed to corner the sheep. How did you formulate this, and how did you solve it?

Data Representation:

In this project, it would be quite difficult to use a 2-dimensional array to represent the whole playground. So, I'm trying to use a position value (from 0 to 63) to represent an 8 * 8 array. The advantages to use a single value to represent the point of a 2-dimensional array:

1) 2-dimensional array is kind of space-consuming. Because my whole map only contains three points for one sheep and two sheep dogs, the rest space of the map would be wasted.

2) It would simplify the calculation process if I utilize this format of representation. Before I introduce the main strategy of my algorithm, it's quite clear that I should let the sheep to the cell (0, 0) which could be represented as 0. So, the main idea should be how to make the position value of the sheep decrease to 0.

Make a Move:

It would be much easier to make a move by simply adding specific values (-8, -1, 1, 8 represent up, left, right, down) to the position value. Because these three can't step on each other, when I'm trying to move the sheep or the sheep dogs, I have to make sure every single move they make would be valid by using the check function.

How to Make a Decision:

Because the movement of the sheep is random, if I want the position value of the sheep decrease, I should remove the direction choice 1(represent right) and 8(represent down) from the random move list. So, what I'm going to do is to make the sheep dogs to reach the positions where are just 1 cell below and 1 cell right and the whole process is called **chase process**. Then after 2 dogs arriving, the sheep can only move upwards or leftwards which can be seen as the process that the position value decreasing. When the sheep's position value decreases to 0, game over. This process is called **corner process**.

The corner process is quite easy to understand, because the rounds taken in this process is determined by the starting position of the corner process. The key problem is in the chase process. In this process, I first assign the destinations for two dogs, which one is the cell below the sheep and the other is the cell right the sheep. Then the dogs would move towards to the destinations. If the position value of dogs is smaller than the position value of the destinations,

the dogs would move rightwards or downwards, which makes the position value increase, and vice versa. But there would be several corner conditions I would meet. Then I would give these decisions below and I would introduce **pos** as the position value now and **des** as the destination:

If pos < des:

If pos mod 8 <= des mod 8:

pos = pos + 1 (Move right)

Else if pos mod 8 > des mod 8:

pos = pos + 8 (Move down)

Else if pos > des:

If pos mod 8 < des mod 8:

pos = pos + 1 (Move right)

Else if pos mod 8 >= des mod 8:

pos = pos + (-8) (Move up)

(Improvement with a random flip: pos = pos + (-1) (Move left))

Else:

pos = pos + 0 (Hold)

The main decision-making process is above, but when I finish the coding part and find that there are small mistakes, which would lead a dead loop. So, what I'm going to do is trying to flip the dead loop, I use a random function to make the random choice in moving upwards or leftwards. But due to the random flip, it makes my dogs take more rounds to reach the sheep and I tested for 300000 times and compared two different methods. It shows that if I don't use a flip the fail times (stuck in the dead loop) is around 72396 times and the average round is 11. As we can see from the Table 1, the algorithm with random flip performs much better in fail times, but it takes more rounds on average. So, I can make a conclusion that based on my algorithm, there's a trade-off between average round and fail times.

Table 1

	Average Round	Fail times
Without Flip	11	72396
Flip	19	441

2. Given the initial state in the above example, how many rounds do you need (on average) to corner the sheep? Answer as precisely as possible.

I test 100000 times using the given points where sheep is in (4, 0), sheep dogs is in (0, 7), (6, 3). The overall rounds of the 100000 units run is 1462054. So, the average rounds it takes is 14.

3. What is the worst possible initial state (position of dog bots, sheep), and why? Justify mathematically.

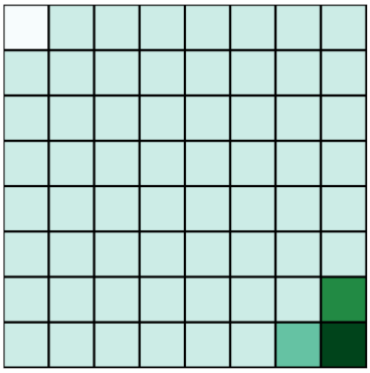

Figure 1 Example for dead loop

In my project, the worst possible initial state must be the one showed in the Figure 1. First, I would use the data after large amounts of test to explain that this must be the worst initial case. I use a function to collect the positions where would cause a dead loop. After huge amounts of test, the result shows that this is the only position which would cause a dead loop. Except that I use a flip function when the sheep is cornered at the position (7,7), there's no possibility for me to jump out of the dead loop based on my algorithm. So, this is one of my drawbacks of my algorithm which means I can't avoid this situation by specific algorithm or operation.

Moreover, I will explain it logically and mathematically. First, position (7, 7) would take 14 rounds to the position (0, 0) if the goat goes to the position (0, 0) directly which is the most. Next, my dogs would move to the destinations below the sheep and right from the sheep. So, being above and left from the sheep would take lots of rounds to get to the destinations and in the meantime, there's still a chance that the sheep would back to the position (7, 7).

So, I could make a summary that position (7, 7) would be the worst initial position.

4. You are allowed to place your dog bots anywhere in the field at the start, then the sheep will be placed uniformly at random in one of the remaining unoccupied cells. Where should you place your dog bots initially, and why? Justify mathematically.

In my analysis, I can model the average rounds taken as:

$$Round_{total} = Round_{chasing} + Round_{cornering}$$

As I've mentioned, if my dogs have already in their positions, it would start the cornering process and the rounds taken are determined by the Manhattan distance from the starting position of cornering process. In another word, the $Round_{cornering}$ almost can't be calculated due to the uncertainty of the movement of the sheep. So, what I'm trying to do is to make sure the dogs can finish the chasing process with as few rounds as possible. First, I just assume that the sheep wouldn't move which would make it easier to model the distance the dogs would travel. I can give a function that contains Manhattan distance as major indicator for calculating the average min rounds:

$$Average\ Distance = \sum\sum P_{ij} \times Distance_{Dog\ between\ Sheep}$$

Then I can figure out that the initial position in the center of the playground is suggested. The positions would be (3, 3), (3, 4), (4, 3), (4, 4). If the dogs are initialized to assign to these positions, it would take an average min rounds to finish the chasing process.

5. Do you think better strategies exist than the one you came up with? Justify.

As far as I'm concerned, I should determine what the best average case is like, so, I try to calculate how many rounds would take if there's no dogs and the sheep would directly go to (0, 0). I find that the best case which make sure there's no any other influence, would be 7.

$$Average\ Min\ Round = \sum_{i=1}^{8} \frac{i \times (i-1)}{8 \times 8} + \sum_{i=9}^{14} \frac{i \times (15-i)}{8 \times 8} = 7$$

As I've mentioned in the last part, I can utilize the equation I given to estimate the average min round in the chasing process. For the best case, if the dogs are initialized to assign to the center of the playground. Then the average min round is around 4. Because this process really contains uncertainty, I just make the sheep still to calculate the average min round – that's why I called it as an estimation.

Then I can figure out the best average round is around 11. As I've mentioned, I use the algorithm without flip, which would give an average output at 11. It's not hard to understand why that algorithm performs such well, because I've cut all the dead loop cases. If there's no dead loop cases exist, the first algorithm I given has the best performance due to it has such a clean logic. That's all of the best cases. The real cases would contain lots of uncertainty and changes.

So, I can make a conclusion that there must be a better strategy existing, because I can make a better algorithm by changing some parameters just like the random flip rate. If I can determine the position and use a specific flip rate, the average round would be smaller. That's only one possible way to improve the performance of my code. So, I believe there's a better algorithm exists.