Programming Assignment 3

CS 747: Foundations of Intelligent & Learning Agents

(Autumn 2023)

Atishay Jain (210050026)

210050026@iitb.ac.in

Contents

PART

I

Optimal Cue-stick Control

Section 1

Basic Implementation

First of all, I started the assignment with trying out different constant angle values to be returned from agent so as to get idea of the convention of angle measurement, and also different constant force values to get idea of hitting power at a particular force. Then, I made some helper functions of the physics of the game, like calculating angle (as per the convention used), distance between two points etc. Now, I proceeded to make my agent hit a particular ball which I want it to hit in a desired direction -

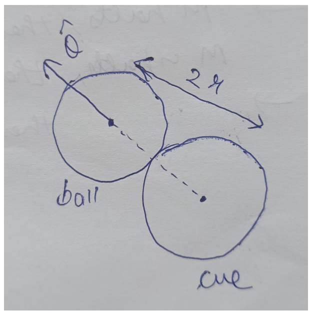
Subsection 1.1

Hitting a ball

I used basic physics for hitting a ball, in which the first step is to find the appropriate direction to which I want to hit the ball, and then hitting the ball with cue such that the ball goes in that direction. Suppose that I have chosen a particular ball (with coordinates (b_x, b_y)) to hit and a hole (with coordinates (h_x, h_y)) towards which I am hitting it. So, the desired direction in which ball should go is -

$$\theta = -\tan^{-1}\left(\frac{h_x - b_x}{b_y - h_y}\right)$$

The direction angle θ's formula is a bit different from the standard one due to the convention of angle being different. Now, say the cue is at coordinates (c_x, c_y) and the radius of both ball and cue is r



From the figure, we see that when the cue comes in contact of ball, the ball will go in the direction pointed by arrow, that is, from center of cue towards center of ball. So, the point P at which we should aim our cue should be the point which is at a distance of $2 * r$ from center of ball, in the direction opposite to θ. So, the coordinate (p_x, p_y) at which I am actually aiming the cue (or in other words, I want my cue to hit at this point) would be (say the direction opposite to θ is at angle ϕ)

$$(p_x, p_y) = (b_x + 2rsin(\phi), b_y + 2rcos(\phi))$$

Subsection 1.2
Angle and Force

Now, suppose I have chosen a ball to hit and the hole to which I am hitting it by some mechanism which I will explain later, then the needed parameters are calculated as -

- angle: the angle between cue and the point (p_x, p_y) towards which I am hitting the cue
- force: A natural feeling was that force should depend on the distance. That is, if the distance to be covered is more, the force should be more. Furthermore, it needs to be mapped to range [0,1]. But, there are several cases, in which it is crucial to decide which distance, i.e. cue-ball, ball-hole, cue-hole or their combination. I experimented for following choices for force -

  1. Say, d1 = distance(cue,ball), d2 = distance(ball,hole), d3 = distance(cue,hole), max_d = maximum distance possible, (= 920)

     $$\text{force} = max(max(d1/max\_d, d2/max\_d), d3/max\_d)$$

     $$\text{force} = min(\text{force}, 1)$$

  2. Taking ratio

     $$\text{force} = max\left(\frac{d1}{d1 + d2}, \frac{d2}{d1 + d2}\right)$$

  3. force = $d3/max\_d$

  After all the experiments, and trying out for different values of max_d as well, the one which performed well (by well I mean atleast passed all public cases for some seeds) was the 2nd case. Ideally, it seems that 2nd case is not the most appropriate one, because as it is a ratio, it will output similar force for small d1,d2 pair as that for large d1,d2 pair. But, maybe because it was mostly an aggresive one (as force was mostly not too low), it performed well. And that's why in my final submission, I am using this only.

Subsection 1.3
Choosing Ball to Hit

This is a very critical decision at a step. There are several ways by which you can decide which ball to hit as well to which hole to hit it. I experimented on following ideas -

1. ChooseBall1: Hit the ball which is nearest to some hole
2. ChooseBall2: Hit the ball for which the total distance cue-ball + ball-hole is least for some hole
3. ChooseBall3: Hit the ball for which the angle between the vectors cue-ball and ball-hole is minimized for some hole

Among these, all were almost equally good, as for some seeds, 3rd one performed better, but for some 1 and 2. But overall, I felt that 3rd one is more ideal because in this case, the hit would be more accurate and the ball chosen would be at a position which is hit-able by the cue

Subsection 1.4

Choosing Hole to Aim the chosen Ball

After choosing a particular ball to hit, there are some metrics for choice of hole also, for which I experimented -

1. ChooseHole: Aim for the hole to which the chosen ball is nearest to
2. ChooseHole1: Aim for the hole which minimizes the angle between vector cue-chosenball and chosenball-hole

Ideally it feels that for ChooseBall3, ChooseHole1 should be taken and for ChooseBall1 and 2, ChooseHole. So I tried with many such combinations, but overall ChooseHole was performing better, maybe because it always performs better if hole is a corner hole

Subsection 1.5

Usage of `get_next_state()` function

Now, that I have so many options of Choosing a Ball and Choosing a Hole, and force also, I thought to use this function such that -

- Choose a ball, Choose a Hole, and then find out the expected next state using this function
- Do this for every combination, and stop when in a particular combination, some ball gets to a hole
- If in none of them this happens, then use a default strategy of ChooseBall3 and ChooseHole.
- But if a ball gets into a hole on some combination, break from the loop of trying and use that combination as final output. Although due to error, the exact desired outcome may not happen, but there are high chances that it may happen.

Now, in the case explained above there would be atmost 6 calls to this function. But actually, when I experimented by removing the Hole choice and only trying different choice of balls, the agent performed better. So, in my final code, I am looping only over the choice of balls and using ChooseHole only. Therefore, making atmost 3 calls to `get_next_state()` in a call of action function

Section 2

Final Code

Here, I am explaining some final modifications to above things explained that I have done in my final agent -

- Added two variables to class Agent
  - `self.curr_ball` : The ball chosen to hit
  - `self.hitting_hole` : The hole towards which this ball is aimed at
- Made some helper functions inside the Agent class
  - `self.get_angle` : for finding angle between two points, as per convention
  - `self.get_dist` : finding distance between two points
  - `self.choose_ball1`, `self.choose_ball2, .choose_ball3, self.choose_hole, self.choose_hole1` : for deciding ball to hit and hole to aim as per the strategy explained before
  - `self.check` : to check if any ball is poked after `get_next_state()`
- Note that in an ideal case, the cue should have been aimed at a point (p_x, p_y) at a distance of $2r$ as explained before. But due to noise, sometimes this might led to cue not even touching the ball, as the distance might increase further due to noise. To resolve this issue, I aimed at the point (p_x, p_y) which is a little nearer, that is, $1.85r$ from center of ball. I tried for many values and found that at $1.85r$, the agent was performing well compared to others. So, actually the expression which I am using is
$$(p_x, p_y) = (b_x + 1.85rsin(\phi), b_y + 1.85rcos(\phi))$$
- Also, as explained before that I am iterating only over choose_ball strategies with `get_next_state()`, and using the strategy of nearest hole for holes and force as described in its 2nd method

Overall, I think there is pretty much amount of randomness involved because my current agent which I am submitting, passes all 10 given public levels for some seeds, but for some seeds it fails some levels as well. The other ideas which I have described have worked for some other seeds, but not for some others similar to this. Finally I just submitted the agent which worked well for most of my trials.