CS747 : Programming Assignment 3

Adityaya Dhande 210070005

November 5, 2023

Approach

My solution relies on the availability of the `get_next_state` function. Here is the algorithm that I implemented in `agent.py`,

1. For each ball I calculate the angle by the line joining the ball and hole, for all holes
2. After this I pick a hole which minimises a cost defined as a linear combination of (i)the difference in the angle made by the ball-hole line and ball-cue line, and the (ii)distance between ball and hole.
3. For this I then evaluate the angle at which the cue has to be shot using geometric calculations.
4. I then take 9 values of force in [0.2, 0.3 . . . 1] and simulate using `get_next_state` function. For each `next_state` I take the minimum distance of the targeted ball from all holes. I store the minimum of these minimum distances and the corresponding force in separate lists, `min_dists` and `forces` respectively.
5. I then select the angle and force corresponding of the ball which corresponds to the minimum of the `min_dists` list.

   Though in some calls of agent.py I am using the `get_next_state` function more than 20 times, I'm using it less than 10 times on other ocassions. I had to tune the linear combination in step 2 for good performance.

Code explanation

```python
def cue_dist(self, ballx, bally, cuex, cuey):
    dist = numpy.square(ballx - cuex) + numpy.square(bally - cuey)
    dist = numpy.sqrt(dist)
    return dist

def hole_dists(self, ballx, bally):
    hole_x, hole_y = self.holes[:,0], self.holes[:,1]
    d = numpy.sqrt((hole_x - ballx)**2 + (hole_y - bally)**2)
    return d

def hole_angles(self, ballx, bally):
    hole_x, hole_y = self.holes[:,0], -self.holes[:,1]
    theta_BH = -numpy.arctan2(hole_y + bally, hole_x - ballx) + PI/2
    theta_BH = wrap_angle(theta_BH)
    return theta_BH
```

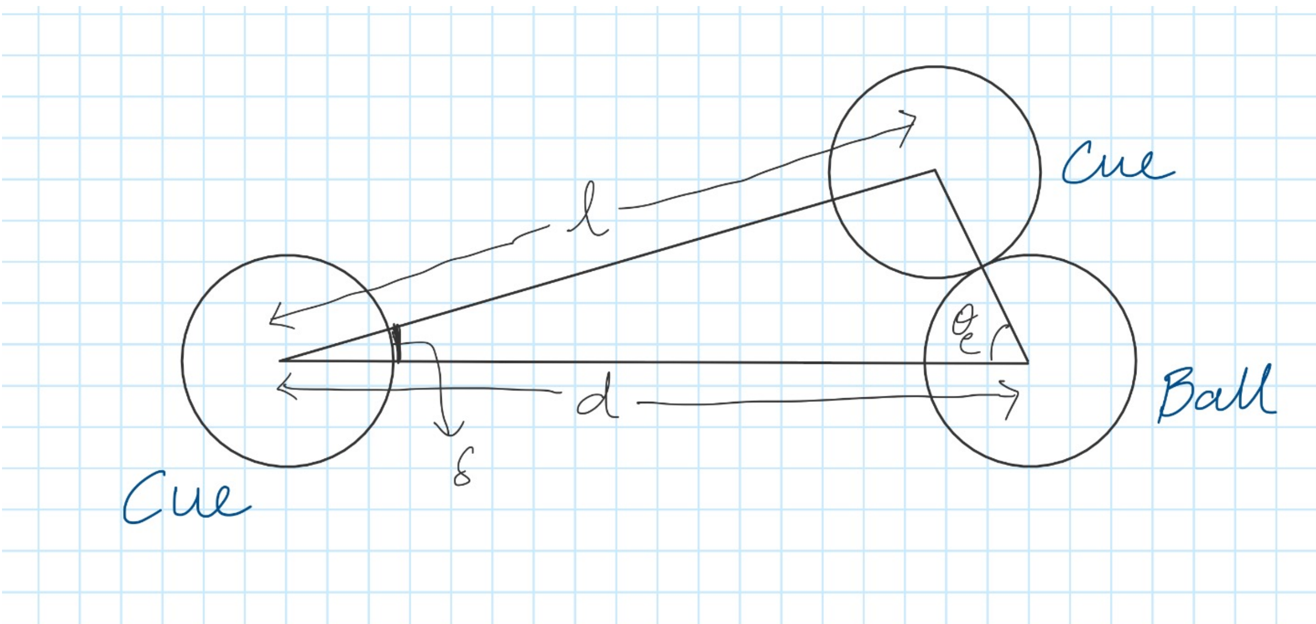
```python
def get_delta(self, ballx, bally, hole, d, cue_angle):
    hole_angles = self.hole_angles(ballx, bally)
    hole_angle = hole_angles[hole]
    theta_m = PI/2 - math.asin(self.ball_radius * 2/d)
    angle_error = numpy.clip(hole_angle - cue_angle, -theta_m, theta_m)
    l = numpy.sqrt(d**2 + 4*(self.ball_radius**2) - 4 * d * self.ball_radius * numpy.cos(angle_error))
    shot_angle = numpy.arcsin(2 * self.ball_radius * numpy.sin(angle_error) / l)
    return shot_angle

def simulate(self, ball, delta, current_state):
    forces = numpy.linspace(0.2, 1, 9)
    no_balls = len(current_state.keys())
    curr_min_dist = 2000
    optimal_force = 0.5
    for force in forces :
        next_state = self.ns.get_next_state(current_state, [-delta,force], seed=10)
        if len(next_state.keys()) < no_balls :
            return force, 0
        ballx = next_state[ball][0]
        bally = next_state[ball][1]
        hole_dists = self.hole_dists(ballx, bally)
        min_dist = numpy.min(hole_dists)
        if min_dist < curr_min_dist :
            optimal_force = force
            curr_min_dist = min_dist
    return optimal_force, min_dist


def action(self, ball_pos):
    balls = list(ball_pos.keys())
    balls.remove('white')
    if 0 in balls :
        balls.remove(0)
    balls.sort()
    cue_x, cue_y = ball_pos['white']
    X = [ball_pos[i][0] for i in balls]
    Y = [ball_pos[i][1] for i in balls]
    dist = self.cue_dist(X, Y, cue_x, cue_y)
    theta_CB = -numpy.arctan2(cue_y - Y, X - cue_x) + PI/2
    theta_CB = wrap_angle(theta_CB)
    shooting_angles = []
    forces = []
    min_dists = []
    for i, ball in enumerate(balls) :
        alpha = 0.8
        cost = alpha*self.hole_dists(X[i], Y[i])/600 + (1-alpha)*numpy.absolute(self.hole_angles(X[i], Y[i])-theta_CB[i])
        hole = numpy.argmin(cost)
        delta = self.get_delta(X[i], Y[i], hole, dist[i], theta_CB[i])
        delta = theta_CB[i] - delta
        force, closest_dist = self.simulate(ball, delta/PI, ball_pos)
        shooting_angles.append(delta)
        forces.append(force)
        min_dists.append(closest_dist)
    action = numpy.argmin(numpy.array(min_dists))
    return -shooting_angles[action]/PI, forces[action]
```

1. `cue_dist` : Returns the distance between
2. `hole_dists` : Returns the distances between the ball and all the holes in an array.
3. `hole_angles` : Returns an array containing the angles made by the lines joining the ball with all the holes.
4. `get_delta` : Returns the δ angle, as described in angle calculation
5. `simulate` : Returns the best force and the minimum final distance from the closest hole, as described in step 4, for a given ball
6. `action` : Evaluates and returns the best action by calling functions to implement the entire algorithm.

Angle calculation



The unknowns in the figure are l and δ. We know θ_e and d and r. We can first find l using cosine rule as,

$$l^2 = d^2 + (2r)^2 - 2 \times (2r) \times (d) \times \cos(\theta_e)$$

We can then use sine rule to find δ as,

$$\sin(\delta) = \frac{(2r) \times \sin(\theta_e)}{l}$$