Prateek Garg
20D070060
prateekg@iitb.ac.in

Programming Assignment 3
CS747: Foundations of Intelligent and Learning Agents, Autumn 2023

2023-11-05

Task 1

```python
def anglexy(x,y):
    """returns angle(-1,1) of point x,y"""
    if x<=0 and y<=0: theta = np.arctan(x/y)
    elif x<=0 and y>0:theta = np.pi - np.arctan(-x/y)
    elif x>0 and y>0:theta =  - np.pi + np.arctan(x/y)
    elif x>0 and y<=0:theta = - np.arctan(-x/y)
    return theta/np.pi

def length_xy(v):
    return np.linalg.norm(v)

class Agent:
    def action(self, ball_pos=None):
        vec_w = np.array(ball_pos["white"]) # getting the coordinates of cue ball

        # removing the cue ball from position list
        del ball_pos["white"]
        del ball_pos[0]

        ball_vecs = np.array(list(ball_pos.values())) # Getting the vectors of balls
        hole_vecs = np.array(self.holes)              # Getting the vectors of holes

        hole2ball_vecs = ball_vecs[:,None,:] - hole_vecs[None,:,:] # pairwise vectors between ball and holes

        hit_points_vecs = ball_vecs[:,None,:]+2*self.ball_radius*(hole2ball_vecs/np.linalg.norm(hole2ball_vecs, axis=-1, keepdims=True))
        cue2hit_points_vecs = hit_points_vecs - vec_w[None,None,:]
        cue2ball_vecs = ball_vecs[:,None,:] - vec_w[None,None,:]
        # print(.shape)

        DOT_PRODUCT_TOL = 0
        cos_similarity = (-hole2ball_vecs * cue2ball_vecs).sum(axis=-1)/(np.linalg.norm(hole2ball_vecs, axis=-1) * np.linalg.norm(cue2ball_vecs, axis=-1))
        hittable_holes = cos_similarity > DOT_PRODUCT_TOL

        valid_holes_dict = {}
        best_b_so_far = 0
        best_h_so_far = 0
        best_dist_so_far = 100000
        LAMBDA = 0
        for b in range(ball_vecs.shape[0]):
            ball_h = []
            ball_h_dist = []
            for h in range(hole_vecs.shape[0]):
                if hittable_holes[b,h]:
                    ball_h.append(h)
                    ball_h_dist.append(np.linalg.norm(hole2ball_vecs[b,h])+LAMBDA*(1 - cos_similarity[b,h]))
            if len(ball_h_dist) > 0:
                bmindist = min(ball_h_dist)
                ball_h_i = ball_h_dist.index(bmindist)
                if bmindist < best_dist_so_far:
                    best_dist_so_far = bmindist
                    best_b_so_far = b
                    best_h_so_far = ball_h_i
                valid_holes_dict[b] = ball_h_i
            else:
                valid_holes_dict[b] = None
        EPS = 0.95
        coin_flip = np.random.binomial(1,EPS)
        if coin_flip == 1:
            b = best_b_so_far
            h = best_h_so_far
        else:
            b = np.random.randint(ball_vecs.shape[0])
            h = valid_holes_dict[b]
            if h == None:
                h = np.random.randint(hole_vecs.shape[0])

        b_vec = ball_vecs[b]
        h_vec = hole_vecs[h]

        hit_vec = b_vec+2*self.ball_radius*(b_vec - h_vec)/np.linalg.norm((b_vec - h_vec))
        target_x,target_y =hit_vec - vec_w
        angle = anglexy(target_x,target_y)

        FORCE_FACTOR = 1.1
```

```
75         force = FORCE_FACTOR*(length_xy(hit_vec - vec_w) + (length_xy(h_vec - b_vec)/config.
               ball_coeff_of_restitution))/(960*1.414)
76         return (angle, force)
```

1.a Basic Idea

The basic idea is to hit a selected ball at an angle that it gets potted in the selected hole. To achieve this we employ basic vector calculus to find the unit vector from the hole to ball and then scale it by 2*ball_radius, this will give us the vector of cue ball –with respect to the target ball– when it collides with the target ball. Adding the vector of target ball will give us the vector w.r.t origin and to find the angle to be specified we can get the vector w.r.t initial position of cue ball.

1.b Selecting Ball and Hole

Not every hole will be directly coverable by the cue ball. In particular, any hole which subtends a acute angle at the target w.r.t to line joing target ball and cue ball. So we filter out these holes(line 30-32). This also gives rise to a tunable paramter `DOT_PRODUCT_TOL` which by the argument above should be 0 but in general it is difficult to pot ball at hole at near-right angles.
For every call to `action()`, we search for best hole-ball pair(lines 39-50) using a metric `dist(h,b)+LAMBDA*(1-`$\cos\theta$`)` where θ is the angle described above. For `LAMBDA=0` this reduces to finding ball hole pair which are closest to each other. `LAMBDA>0` emphasis that the hole should nicely lined up as well and balances between the 2 notions of goodness of hole-ball pair.
Additionally, we also track best hole per ball in a dictionary. With probability `1-EPS` we randomly samplpe a ball instead of using best hole-ball pair found, and use it's best hole. This balances exploration and exploitation.
Finally, we have an additional parameter `FORCE_FACTOR` which scales the force which is directly proportional to the distance between cue ball and target plus the distance between target and hole scaled by coefficient of restitution.