To be corrected by Daniel Rinaldi

COMP 476-N

Kaustubha Mendhurwa

George Mavroeidis

Q1A)

Declaration of variables

```
pc = np.array([3, 6]) #current position of character
vc = np.array([2, 3]) #current velocity of character
pt = np.array([5, 4]) #current position of target (stationary)
t = 0.25 # tick time
vm = 3.6 # maximum velocity of character
am = 12.25 # maximum acceleration of character
```

Steps explained through code

```
steps = 0
while (steps < 5):
    vd = pt - pc # get velocity direction
    vd_normalized = vd/np.linalg.norm(vd) # normalize velocity direction
    v_ksv = vm * vd_normalized # kinematic seek velocity
    pc = pc + (v_ksv* t) # update position
    steps += 1
```

Results of next five updates:

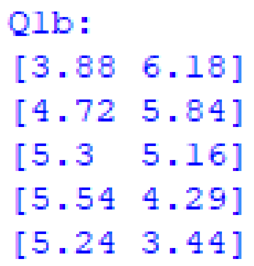
```
Q1a:
[3.64 5.36]
[4.27 4.73]
[4.91 4.09]
[5.55 3.45]
[4.91 4.09]
```

Q1B)

Steps explained through code

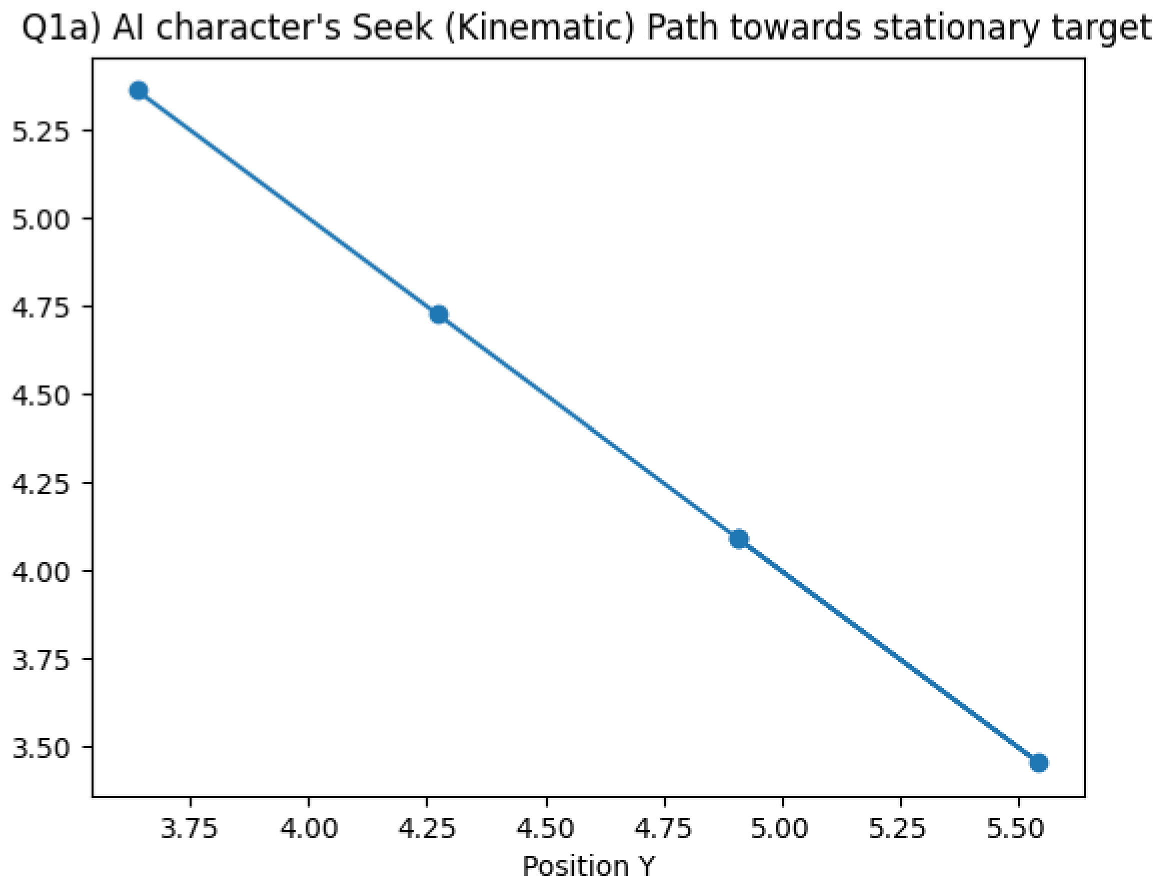
```
steps = 0
while (steps < 5):
    vd = pt - pc # get velocity direction
    vd_magnitude = math.sqrt((vd[0]*vd[0]) + (vd[1]*vd[1]))  # find magnitude
    vd_normalized = vd/vd_magnitude # normalize velocity direction
    a_ssa = am * vd_normalized # steering seek acceleration
    vc = vc + (a_ssa * t) # steering seek velocity
    vc_magnitude = math.sqrt((vc[0]*vc[0]) + (vc[1]*vc[1]))  # find magnitude
    if vc_magnitude > vm: # if it surprasses maximum velocity
        vc = (vc/vc_magnitude) * vm # normalize and multiply by maximum
    pc = pc + (vc * t) # update position
```

Results of next five updates:

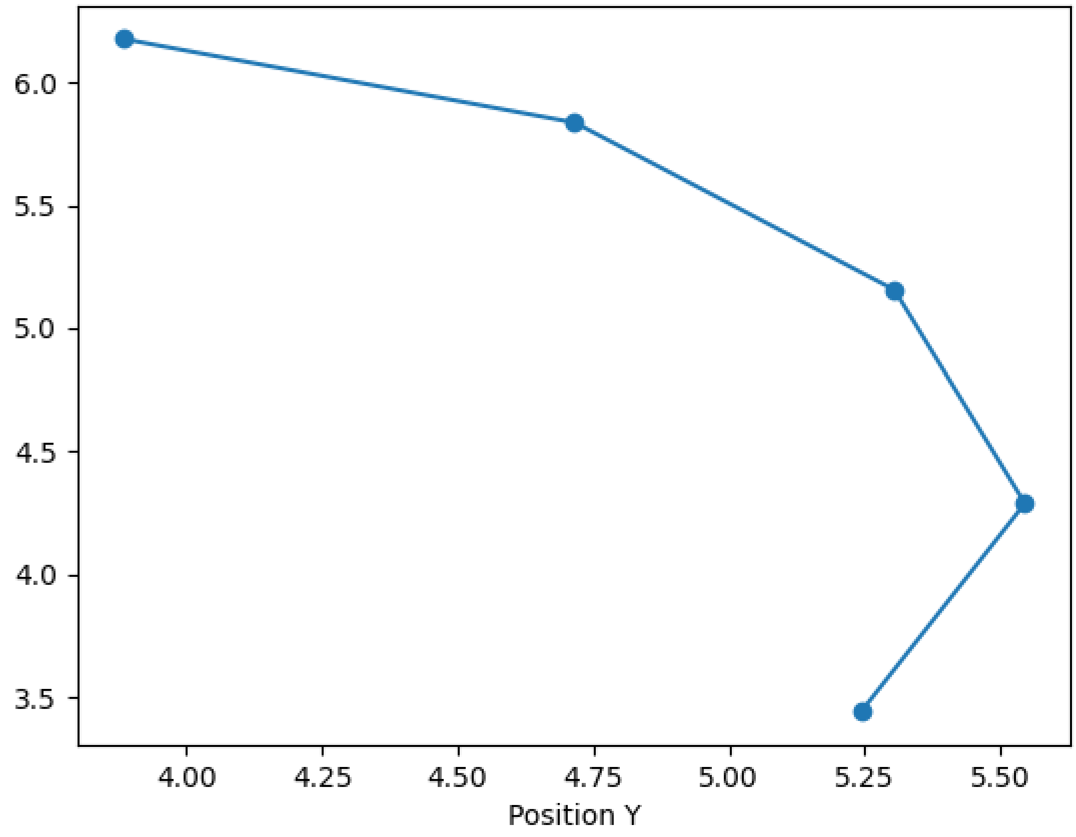


```
Q1b:
[3.88 6.18]
[4.72 5.84]
[5.3  5.16]
[5.54 4.29]
[5.24 3.44]
```

Q1C)



Q1b) AI character's Steer (Kinematic) Path towards stationary target

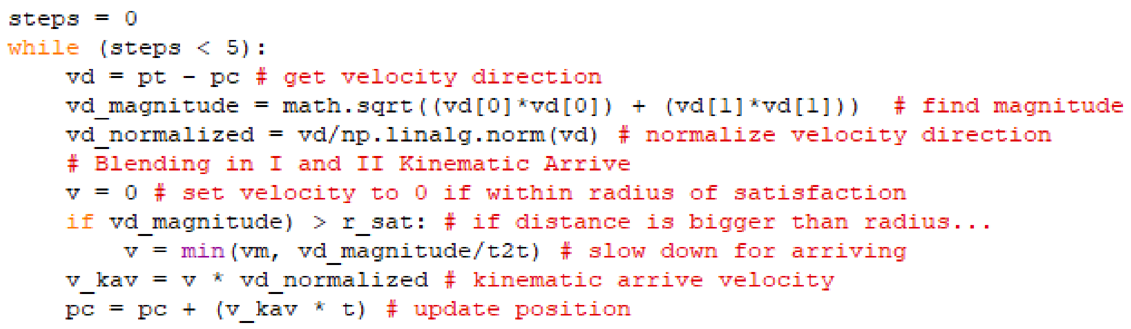


Q1D)

Declaration of variables

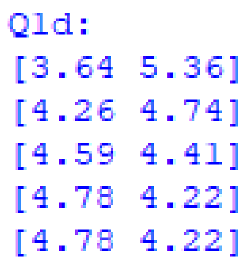
```
pc = np.array([3, 6]) #current position of character
vc = np.array([2, 3]) #current velocity of character
pt = np.array([5, 4]) #current position of target (stationary)
t = 0.25 # tick time
vm = 3.6 # maximum velocity of character
am = 12.25 # maximum acceleration of character
r_sat = 0.5 # satisfaction radius
t2t = 0.55 # time-to-target
```

Steps explained through code



```
steps = 0
while (steps < 5):
    vd = pt - pc # get velocity direction
    vd_magnitude = math.sqrt((vd[0]*vd[0]) + (vd[1]*vd[1]))  # find magnitude
    vd_normalized = vd/np.linalg.norm(vd) # normalize velocity direction
    # Blending in I and II Kinematic Arrive
    v = 0 # set velocity to 0 if within radius of satisfaction
    if vd_magnitude) > r_sat: # if distance is bigger than radius...
        v = min(vm, vd_magnitude/t2t) # slow down for arriving
    v_kav = v * vd_normalized # kinematic arrive velocity
    pc = pc + (v_kav * t) # update position
```

Results of next five updates:



```
Q1d:
[3.64 5.36]
[4.26 4.74]
[4.59 4.41]
[4.78 4.22]
[4.78 4.22]
```

Q1E)

Declaration of variables

```
pc = np.array([3, 6]) #current position of character
vc = np.array([2, 3]) #current velocity of character
pt = np.array([5, 4]) #current position of target (stationary)
t = 0.25 # tick time
vm = 3.6 # maximum velocity of character
am = 12.25 # maximum acceleration of character
r_arr = 0.2 # radius of arrival
r_sat = 0.5 # satisfaction radius
t2t = 0.55 # time-to-target
```

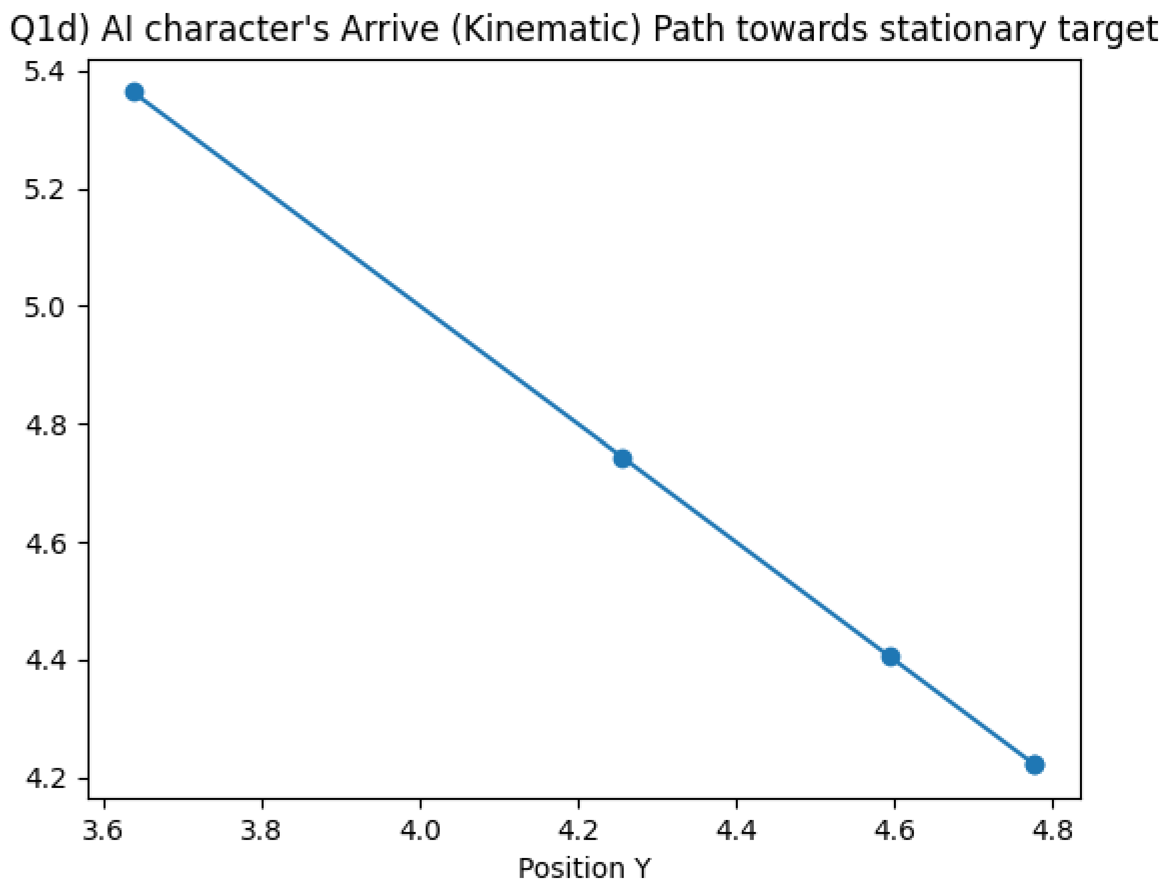
Steps explained through code

```
steps = 0
while (steps < 5):
    vd = pt - pc # get velocity direction
    vd_magnitude = math.sqrt((vd[0]*vd[0]) + (vd[1]*vd[1]))  # find magnitude
    vd_normalized = vd/vd_magnitude # normalize velocity direction
    if vd_magnitude <= r_arr: # if character arrives, stop moving
        continue
    elif vd_magnitude <= r_sat: # if character is in radius, slow down
        speed_target = (vd_magnitude/r_sat) * vm # find target speed
        # find target velocity
        v_target = vd_normalized
        v_target *= speed_target

        # find acceleration
        a = v_target - vc
        a /= t2t
        # keep acceleration within maximum
        a_magnitude = math.sqrt((a[0]*a[0]) + (a[1]*a[1]))
        if np.linalg.norm(a) > am:
            a_norm = (a/np.linalg.norm(a))
            a = a_norm * am

        # find desired velocity
        at = a * t
        v = vc + at

        v_magnitude = math.sqrt((v[0]*v[0]) + (v[1]*v[1]))
        if v_magnitude > vm:
            v = (v/v_magnitude) * vm

        vc = v
        pc = pc + (vc*t) #update position
    else: # if outside of radii, continue steering seek as usual
        a = vd_normalized *am
        v = vc + (a*t)
        v_magnitude = math.sqrt((v[0]*v[0]) + (v[1]*v[1]))
        if v_magnitude > vm:
            v = (v/v_magnitude) * vm
        vc = v
        pc = pc + (vc*t)
```

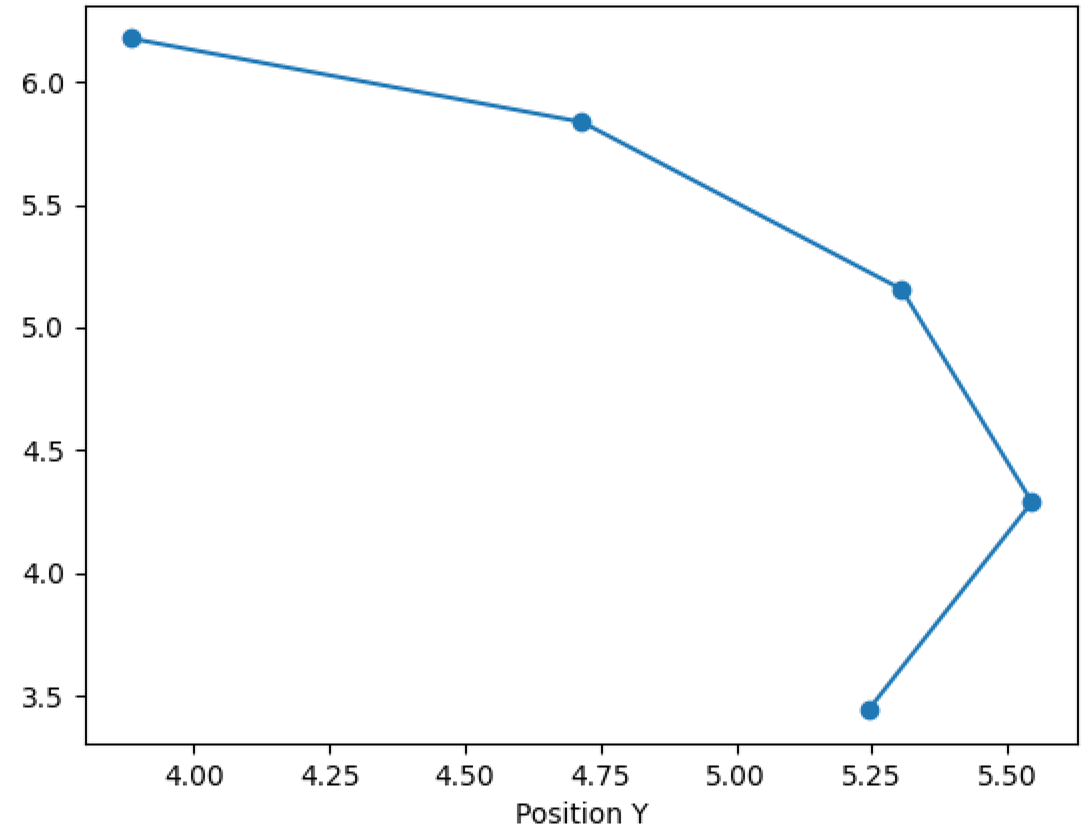
Results of next five updates:

```
Q1e:
[3.88 6.18]
[4.72 5.84]
[5.3  5.16]
[5.51 4.47]
[5.46 3.98]
```

Q1F)



Q1e) AI character's Arrive (Steering) Path towards stationary target



Q2A)

Declaration of variables:

```
c1_asp = np.array([22, 18])
c1_ap = np.array([21, 16])
c1_av = np.array([3, 1])

c2_asp = np.array([6, 13])
c2_ap = np.array([5, 11])
c2_av = np.array([3, 3])

c3_asp = np.array([29, 12])
c3_ap = np.array([28, 9])
c3_av = np.array([6, 5])

k_offset = 1
```

Steps explained through code

```
pc = (c1_ap + c2_ap + c3_ap)/3
vc = (c1_av + c2_av + c3_av)/3
p_anchor = pc + (k_offset * vc)
```

Results:

```
pc = [18. 12.]
vc = [4. 3.]
p_anchor = [22. 15.]
```

Q2B)

```
delta_ps1 = c1_asp - pc
pc1 = p_anchor + delta_ps1

delta_ps2 = c2_asp - pc
pc2 = p_anchor + delta_ps2

delta_ps3 = c3_asp - pc
pc3 = p_anchor + delta_ps3
```

Results:

```
delta_ps1 = [4. 6.]
pc1 = [26. 21.]

delta_ps2 = [-12.   1.]
pc2 = [10. 16.]

delta_ps3 = [11.  0.]
pc3 = [33. 15.]
```

Q2C)

```
pc = (c1_ap + c2_ap)/2
vc = (c1_av + c2_av)/2
p_anchor = pc + (k_offset * vc)

delta_ps1 = c1_asp - pc
pc1 = p_anchor + delta_ps1
delta_ps2 = c2_asp - pc
pc2 = p_anchor + delta_ps2
```

```
[13.  13.5]
[3. 2.]
p_anchor = [16.  15.5]

delta_ps1 = [9.  4.5]
pc1 = [25. 20.]

delta_ps2 = [-7.  -0.5]
pc2 = [ 9. 15.]
```

When C3 is killed, the anchor and the slot positions of C1 and C2 will change only if C3 was killed prior to any of these calculations. If p_anchor was calculated, but then C3 dies, nothing changes.