Team9_game_balancing_analysis

April 12, 2021

1 "Gold" analysis

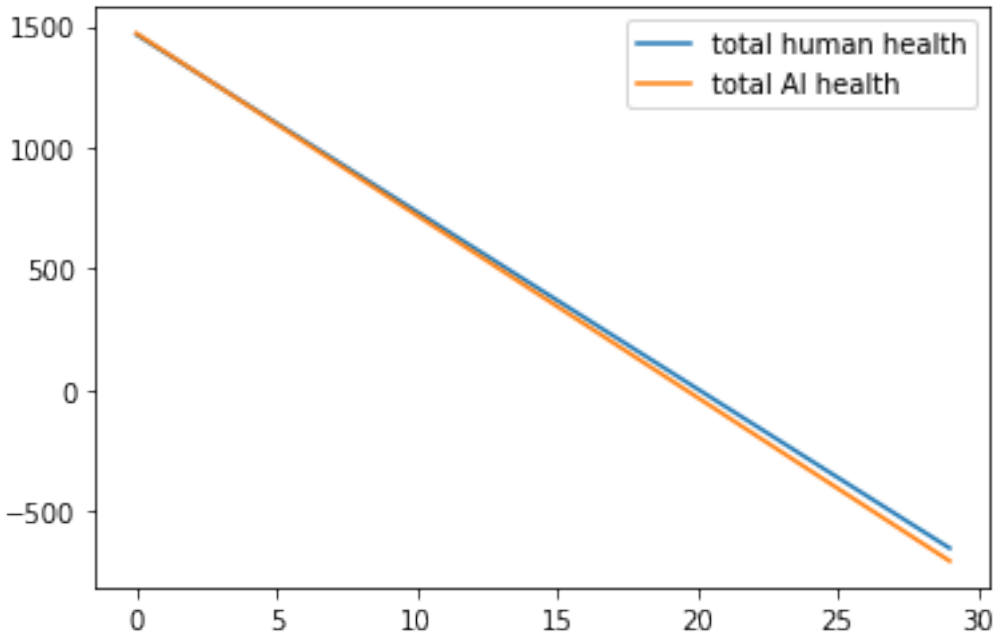
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

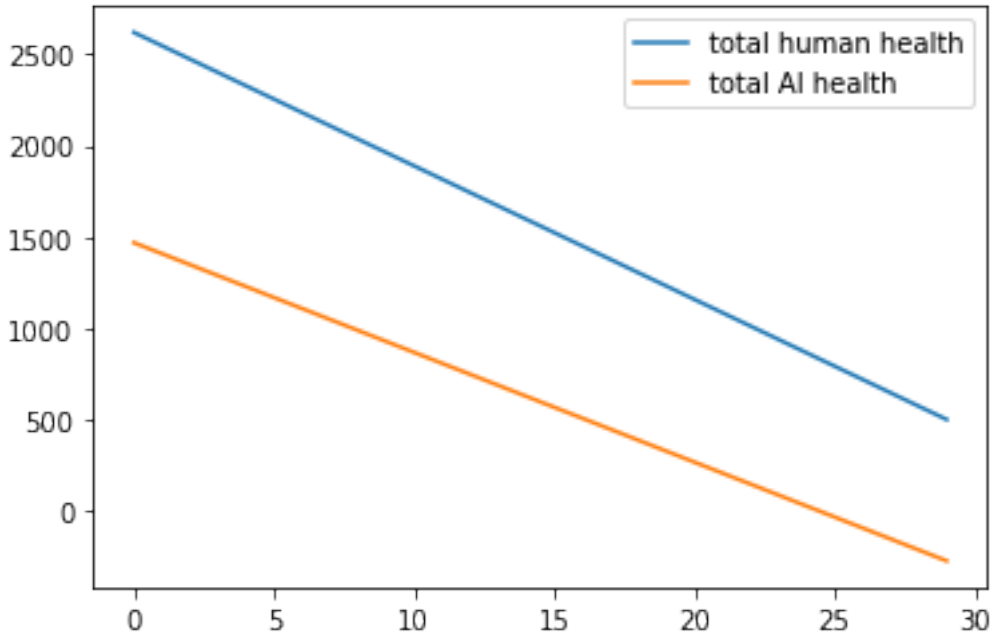
[ ]:
```
# data
time = np.arange(0,30,1)

delta_k=[]
delta_initial_y=[]
gold=[]

#heath, damage, cost constant for each types of unit
human_terminator_damage = 10
human_terminator_hp = 200
human_terminator_cost = 100

human_monitor_damage = 15
human_monitor_hp = 1010
human_monitor_cost = 150

human_archer_damage = 35
human_archer_hp = 75
human_archer_cost = 200

human_healer_damage = 15
human_healer_hp = 180
human_healer_cost = 120

ai_terminator_damage = 12
ai_terminator_hp = 190


ai_monitor_damage = 13
ai_monitor_hp = 1020
```

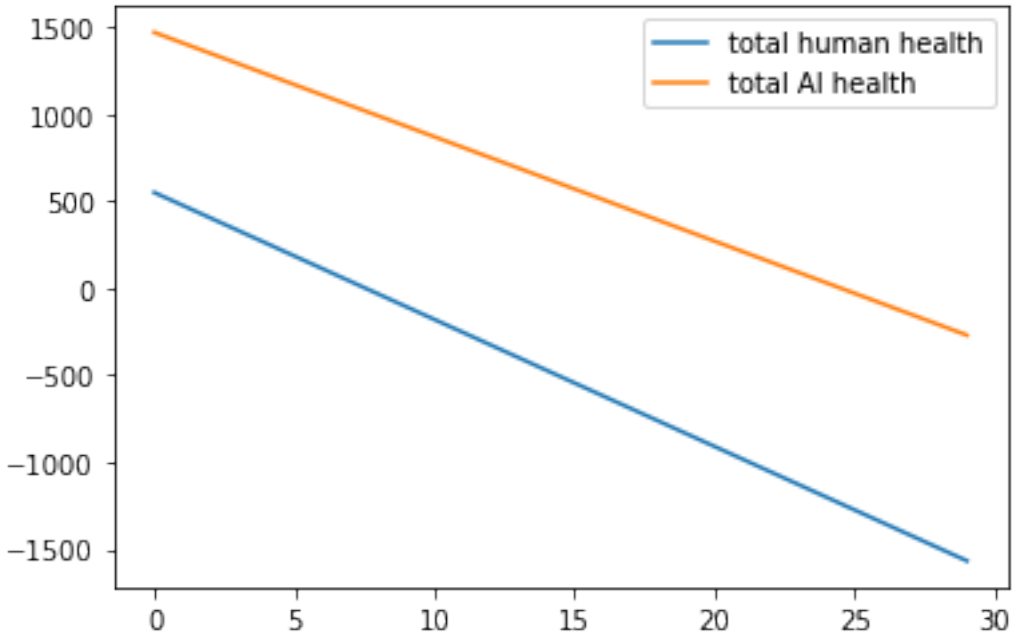
```
ai_archer_damage = 32
ai_archer_hp = 70


ai_healer_damage = 16
ai_healer_hp = 190



# Level1
# 1.Given gold = 600 (two players coorporate, each player has 300)
# Assume each player will place the unit at its optimzed attack range
# Enumerate different types of player will buy different types of units with␣
 ↪the given gold

# (a) tactic player who buy the same types of units as the AI side
init_human_health = human_terminator_hp+ human_monitor_hp+ human_archer_hp+␣
 ↪human_healer_hp
init_ai_health = ai_terminator_hp+ ai_monitor_hp+ ai_archer_hp+ ai_healer_hp
enemy_health = [init_ai_health]
human_health = [init_human_health]

# simulate
enemy_damage = ai_terminator_damage+ ai_monitor_damage+ ai_archer_damage+␣
 ↪ai_healer_damage
human_damage = human_terminator_damage+ human_monitor_damage+␣
 ↪human_archer_damage+ human_healer_damage

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)


# draw the linear line: time vs total human health/ total AI health to see the␣
 ↪similarity of them.
# If these two lines have similar change rate and y value (stick close with␣
 ↪each other), it means the game is balanced.
#  And also indicate that the gold value setting is reasonable,
# so that the player can harness these amount of gold to buy the reasonable␣
 ↪amount of units
# and have a equal chance to win.
plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")
```

```
plt.legend()
plt.show()


delt_change_rate = (human_health[29]-human_health[0])/30 -␣
 ↪(enemy_health[29]-enemy_health[0])/30
delta_y = human_health[0] - enemy_health[0]
delta_initial_y.append(delta_y)
delta_k.append(delt_change_rate)
gold.append(600)

# (b) prefer short distance battle player who buy all the short distance battle␣
 ↪type unit
init_human_health = human_terminator_hp*3+ human_monitor_hp*2
human_health = [init_human_health]
enemy_health = [init_ai_health]
# simulate

human_damage = human_terminator_damage*3+ human_monitor_damage*2

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()

# (c) prefer long distance battle player who buy all the long distance battle␣
 ↪type unit
init_human_health = human_archer_hp*2+ human_terminator_hp*2
human_health = [init_human_health]
enemy_health = [init_ai_health]
# simulate

human_damage = human_terminator_damage*3+ human_monitor_damage*2

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
```

```
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()
```





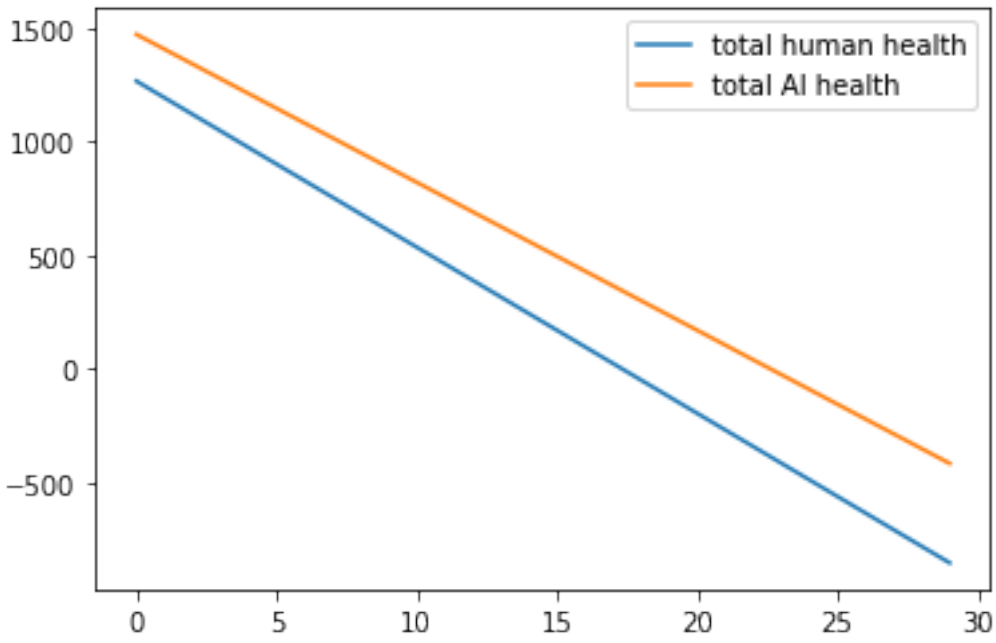


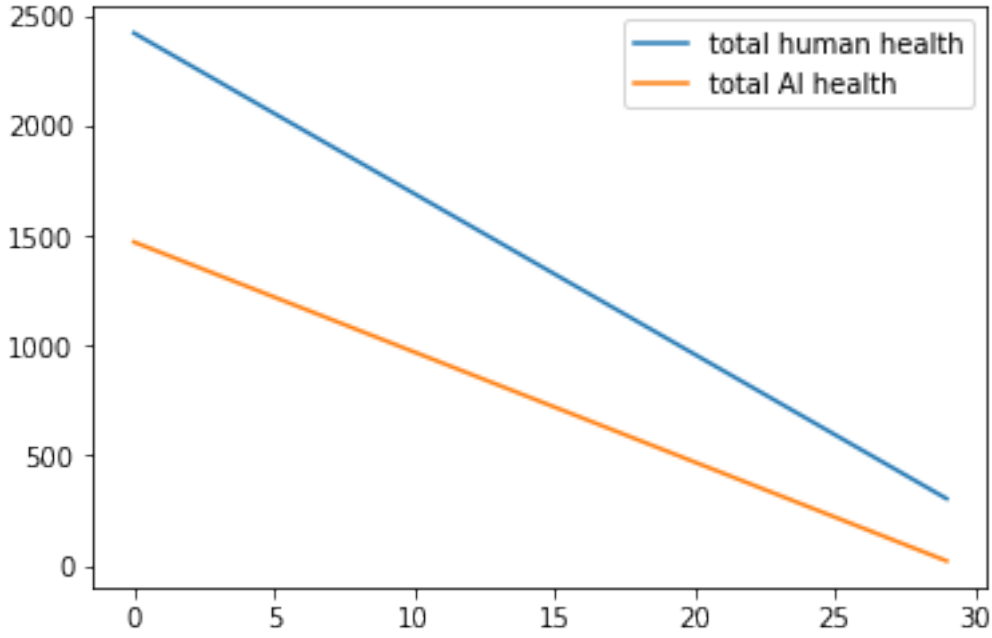
To summarize, given gold 600, we can see tactic player may have 50% chance to win, the change rate of human health and enemy health are exactly the same, as well as their initial value befor battle; while for short-distance-preferred player has a much larger initial human health and faster drecreasing rate than enemy, player has much larger than 50% chance to win. For long-distance-preferred player, he/she has 0% chance to win because of the big gap of the initial value

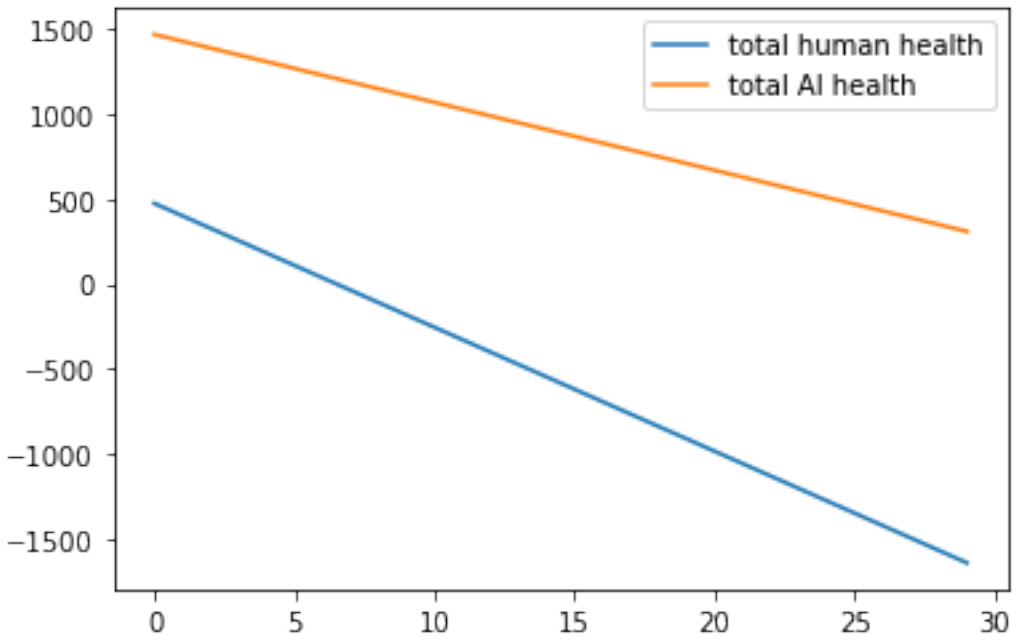
of human health's faster decreasing rate. To average, given gold 600, player (human) can harness these amounts of gold to arrang the unit and will have balanced equal chance (compared to enemy)to win.

[ ]:
```
# Level1
# 2.Given gold = 500 (two players coorporate, each player has 250)
# Assume each player will place the unit at its optimzed attack range
# Enumerate different types of player will buy different types of units with␣
 ↪the given gold

# (a) tactic player who buy the same types of units as the AI side
init_human_health = human_monitor_hp+ human_archer_hp+ human_healer_hp
init_ai_health = ai_terminator_hp+ ai_monitor_hp+ ai_archer_hp+ ai_healer_hp
enemy_health = [init_ai_health]
human_health = [init_human_health]

# simulate
enemy_damage = ai_terminator_damage+ ai_monitor_damage+ ai_archer_damage+␣
 ↪ai_healer_damage
human_damage = human_monitor_damage+ human_archer_damage+ human_healer_damage

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()

delt_change_rate = (human_health[29]-human_health[0])/30 -␣
 ↪(enemy_health[29]-enemy_health[0])/30
delta_k.append(delt_change_rate)
delta_y = human_health[0] - enemy_health[0]
delta_initial_y.append(delta_y)
gold.append(500)
# (b) prefer short distance battle player who buy all the short distance battle␣
 ↪type unit
init_human_health = human_terminator_hp*2+ human_monitor_hp*2
human_health = [init_human_health]
enemy_health = [init_ai_health]
# simulate
```

```
human_damage = human_terminator_damage*2+ human_monitor_damage*2

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()

# (c) prefer long distance battle player who buy all the long distance battle␣
 ↪type unit
init_human_health = human_archer_hp*1+ human_terminator_hp*2
human_health = [init_human_health]
enemy_health = [init_ai_health]
# simulate

human_damage = human_terminator_damage*1+ human_monitor_damage*2

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()
```


total human health
total AI health
1500
1000
500
0
−500
0
5
10
15
20
25
30

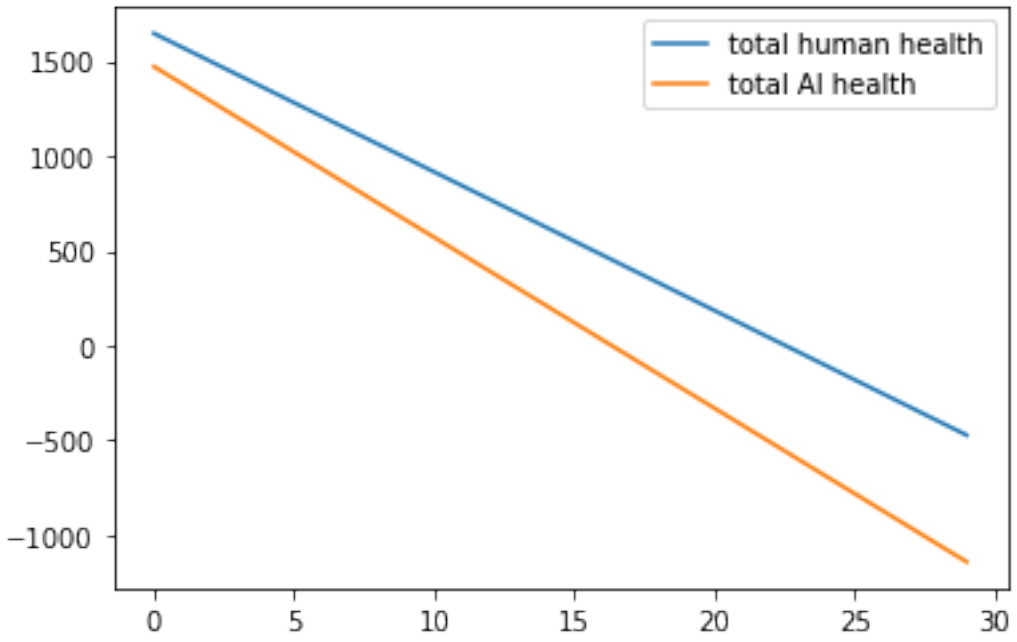

total human health
total AI health
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30

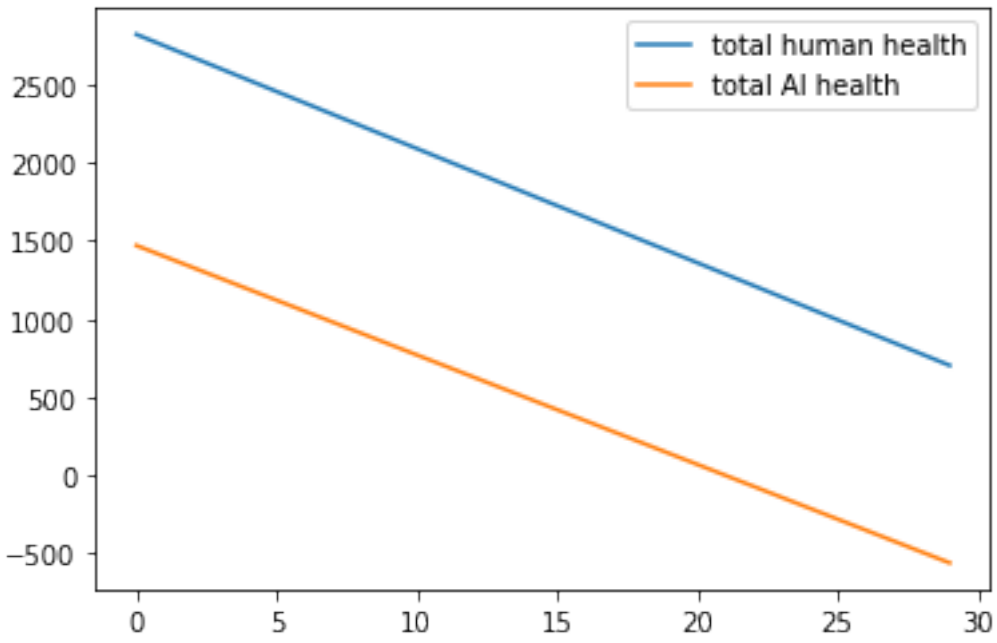


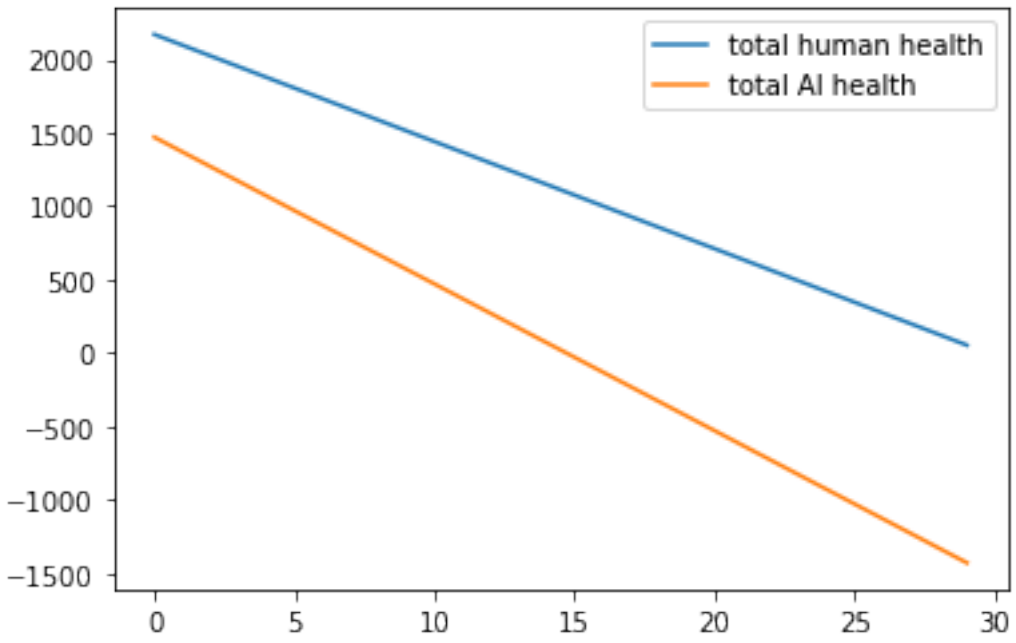
[ ]:
```
# Level1
# 3.Given gold = 700 (two players coorporate, each player has 350)
# Assume each player will place the unit at its optimzed attack range
# Enumerate different types of player will buy different types of units with␣
 ↪the given gold

# (a) tactic player who buy the same types of units as the AI side
init_human_health = human_monitor_hp+ human_archer_hp+ human_healer_hp*2+␣
 ↪human_terminator_hp
init_ai_health = ai_terminator_hp+ ai_monitor_hp+ ai_archer_hp+ ai_healer_hp
enemy_health = [init_ai_health]
human_health = [init_human_health]

# simulate
enemy_damage = ai_terminator_damage+ ai_monitor_damage+ ai_archer_damage+␣
 ↪ai_healer_damage
human_damage = human_monitor_damage+ human_archer_damage+ human_healer_damage*2␣
 ↪+human_terminator_damage

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)
```

```
plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()

delt_change_rate = (human_health[29]-human_health[0])/30 -␣
 ↪(enemy_health[29]-enemy_health[0])/30
delta_k.append(delt_change_rate)
delta_y = human_health[0] - enemy_health[0]
delta_initial_y.append(delta_y)
gold.append(700)
# (b) prefer short distance battle player who buy all the short distance battle␣
 ↪type unit
init_human_health = human_terminator_hp*4+ human_monitor_hp*2
human_health = [init_human_health]
enemy_health = [init_ai_health]
# simulate

human_damage = human_terminator_damage*4+ human_monitor_damage*2

for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()

# (c) prefer long distance battle player who buy all the long distance battle␣
 ↪type unit
init_human_health = human_archer_hp*2+ human_monitor_hp*2
human_health = [init_human_health]
enemy_health = [init_ai_health]
# simulate

human_damage = human_archer_damage*2+ human_monitor_damage*2
```

```
for i in time[1:]:
  new_hh = human_health[i-1] - enemy_damage
  new_eh = enemy_health[i-1] - human_damage
  human_health.append(new_hh)
  enemy_health.append(new_eh)



plt.plot(time, human_health, label="total human health")
plt.plot(time, enemy_health, label="total AI health")


plt.legend()
plt.show()
```





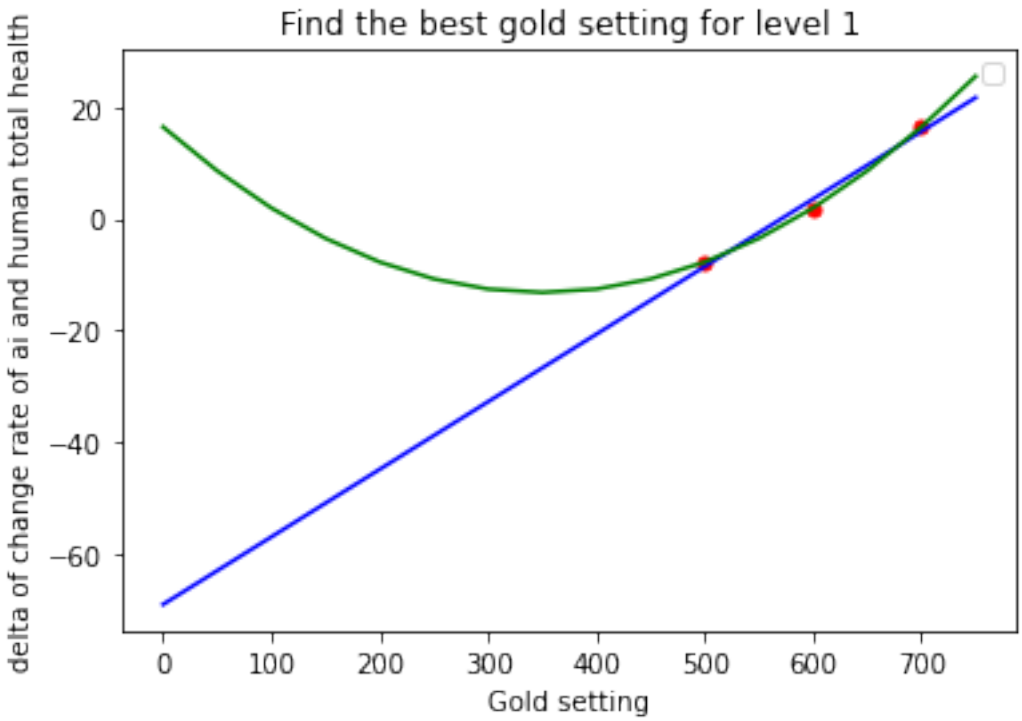


[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
```

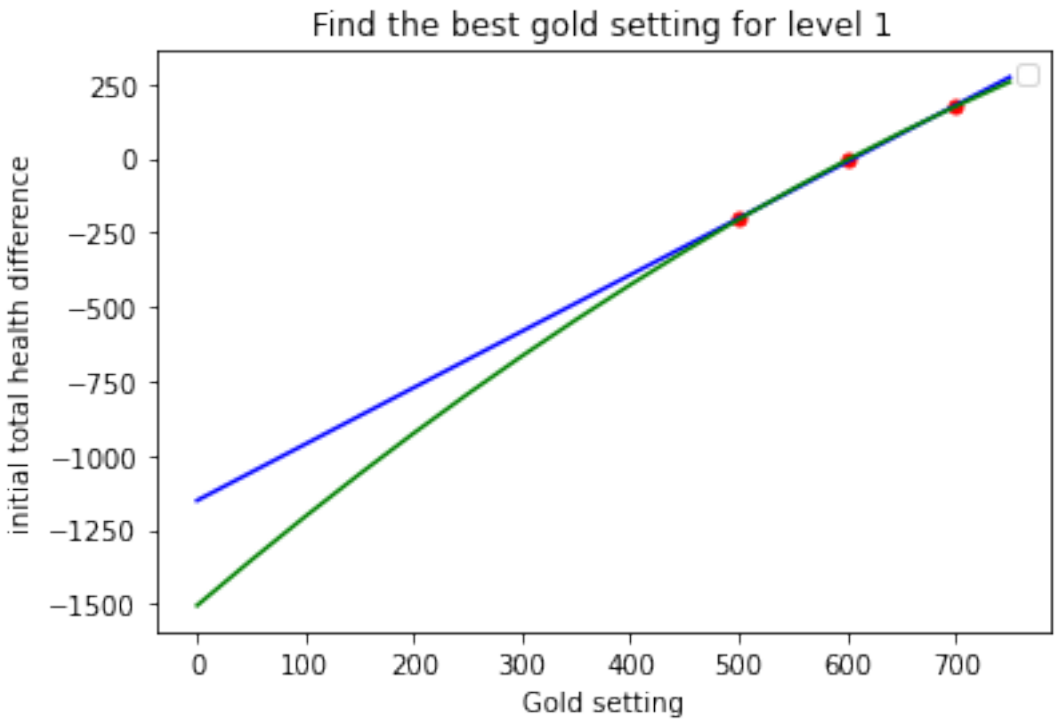
```
#linear equantion
def f_1(x, A, B):
    return A*x + B

#quadratic curve
def f_2(x, A, B, C):
    return A*x*x + B*x + C


x_label = 'Gold setting'
y_label = 'delta of change rate of ai and human total health'

def plot_test(x0,y0,x_label,y_label):

    plt.figure()


    #draw the scattered points
    plt.scatter(x0[:], y0[:], 25, "red")

    #Straight line fitting and drawing
    A1, B1 = optimize.curve_fit(f_1, x0, y0)[0]
    x1 = np.arange(0, 800, 50)
    y1 = A1*x1 + B1
    plt.plot(x1, y1, "blue")

    #Conic curve fitting and drawing
    A2, B2, C2 = optimize.curve_fit(f_2, x0, y0)[0]
    x2 = np.arange(0, 800, 50)
    y2 = A2*x2*x2 + B2*x2 + C2
    plt.plot(x2, y2, "green")

    plt.title("Find the best gold setting for level 1")
    plt.xlabel(x_label)
    plt.ylabel(y_label)

    plt.legend()
    plt.show()

    return

plot_test(gold, delta_k,x_label,y_label)
y_label = 'initial total health difference'

plot_test(gold, delta_initial_y,x_label,y_label)
```

```
/usr/local/lib/python3.7/dist-packages/scipy/optimize/minpack.py:808:
```

```
OptimizeWarning: Covariance of the parameters could not be estimated
  category=OptimizeWarning)
No handles with labels found to put in legend.
```




```
/usr/local/lib/python3.7/dist-packages/scipy/optimize/minpack.py:808:
OptimizeWarning: Covariance of the parameters could not be estimated
  category=OptimizeWarning)
No handles with labels found to put in legend.
```



We enumerate some value of gold setting for level 1, and we pick the tactic player's strategy and record the corresponding delta change rate of ai and human health and the initial total health differece of AI and human. We do the linear line fiting and quadratic line fitting. According to the results from the above two fitted graphs, we can come out the best gold setting for level 1 (the most balanced result) should be : 600. Since when gold is around 600, we can see both the initial total health differece of AI and human of the the delta change rate of ai and human health is close to 0 which indicate the enemy units and players' units is balanced.

We then used the same methods to come accross the best gold setting for

Level 2 : 900

Level 3 : 1200

Level 4 : 1600

Level 5 : 1900

[6]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
!cp drive/My\ Drive/Colab\ Notebooks/Team9_game_balancing_analysis.ipynb ./
!jupyter nbconvert --to PDF "Team9_game_balancing_analysis.ipynb"
```

```
Reading package lists... Done
Building dependency tree
```

```
Reading state information... Done
pandoc is already the newest version (1.19.2.4~dfsg-1build4).
texlive is already the newest version (2017.20180305-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
texlive-xetex is already the newest version (2017.20180305-1).
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.7/dist-
packages (1.5)
Requirement already satisfied: wheel>=0.25.0 in /usr/local/lib/python3.7/dist-
packages (from pypandoc) (0.36.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from pypandoc) (54.2.0)
Requirement already satisfied: pip>=8.1.0 in /usr/local/lib/python3.7/dist-
packages (from pypandoc) (19.3.1)
[NbConvertApp] Converting notebook Team9_game_balancing_analysis.ipynb to PDF
[NbConvertApp] Support files will be in Team9_game_balancing_analysis_files/
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Making directory ./Team9_game_balancing_analysis_files
[NbConvertApp] Writing 60128 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
'-quiet']
[NbConvertApp] Running bibtex 1 time: [u'bibtex', u'./notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 198156 bytes to Team9_game_balancing_analysis.pdf
```