"""

This is an example of a pro bot.

"""

from pirates import \*

tactic = False

def i\_have\_capsule(game):

for pirate in game.get\_my\_living\_pirates():

if pirate.has\_capsule():

return pirate

return False

def enemy\_have\_capsule(game):

for pirate in game.get\_enemy\_living\_pirates():

if pirate.has\_capsule():

return pirate

return False

def best\_pushing\_loc(target,place,game):

r = target.location.row + (target.location.row - place.location.row)

c = target.location.col + (target.location.col - place.location.col)

return Location(r,c)

def send(pirates,dest,aggresive,game):

def pushed(pirate,game):

enemies = game.get\_enemy\_living\_pirates()

for enemy in enemies:

if pirate.can\_push(enemy):

pirate.push(enemy,best\_pushing\_loc(enemy,game.get\_enemy\_capsule(),game))

return True

return False

if aggresive:

for pirate in pirates:

if not pushed(pirate,game):

pirate.sail(dest)

else:

for pirate in pirates:

pirate.sail(dest)

def can\_push\_twice(pirates,target,game):

for pirate in pirates:

if not pirate.can\_push(target):

return False

return True

def push\_twice(pirates,target,game):

for pirate in pirates:

pirate.push(target,best\_pushing\_loc(target,game.get\_enemy\_capsule(),game))

def get\_nearest\_to\_target(pirates,target,game):

choice = pirates[0]

min = choice.distance(target)

for pirate in pirates:

if pirate.distance(target) < min:

min = pirate.distance(target)

choice = pirate

return choice

def eliminate\_threat(pirates,threat,aggresive,game):

for pirate in pirates:

if pirate.can\_push(threat):

pirate.push(threat,best\_pushing\_loc(threat,game.get\_enemy\_capsule(),game))

else:

send([pirate],threat,aggresive,game)

def arrange(group1,group4,game):

for pirate in group1:

if pirate.has\_capsule():

return group1, group4

return group4, group1

def is\_city\_unreachable(game):

counter = 0

enemies = []

for enemy in game.get\_enemy\_living\_pirates():

if enemy.distance(game.get\_my\_mothership()) < 1500:

counter += 1

enemies.append(enemy)

if counter > 2:

return enemies

return False

def average\_place(pirates,game):

global tactic

row = 0

col = 0

for pirate in pirates:

row += pirate.location.row

col += pirate.location.col

row /= len(pirates)

col /= len(pirates)

return Location(row,col)

def do\_turn(game):

group1 = []

group2 = []

group3 = []

group4 = []

core = game.get\_my\_capsule()

mother = game.get\_my\_mothership()

emother = game.get\_enemy\_mothership()

for pirate in game.get\_all\_my\_pirates():

if pirate.is\_alive():

if pirate.id == 0 or pirate.id == 1:

group1.append(pirate)

elif pirate.id == 2 or pirate.id == 3:

group2.append(pirate)

elif pirate.id == 4 or pirate.id == 5:

group3.append(pirate)

else:

group4.append(pirate)

### helpers

carrier = i\_have\_capsule(game)

if carrier: # decide who's the carriers and the scouts

carriers, scouts = arrange(group1,group4,game)

enemy\_guards = is\_city\_unreachable(game)

if enemy\_guards and carrier.distance(mother) < 1500 and not tactic: # if my city is guarded, go like this

tactic = True

enemy\_pos = average\_place(enemy\_guards,game):

elif not tactic: # just go home

send(carriers,mother,True,game)

threat = get\_nearest\_to\_target(game.get\_enemy\_living\_pirates(),carrier,game)

eliminate\_threat(scouts,threat,False,game)

else: # no carrier, go to get capsule

for group in [group1,group4]:

send(group,core,True,game)

### interruptors

target = enemy\_have\_capsule(game)

"""

if target:

for group in [group2,group3]:

if can\_push\_twice(group,target,game):

push\_twice(group,target,game)

else:

send(group,target,False,game)

else:

for group in [group2,group3]:

send(group,emother,True,game)

"""

if target and not carrier:

for group in [group2,group3]:

if can\_push\_twice(group,target,game):

push\_twice(group,target,game)#?

else:

send(group,target,False,game)

elif not target and not carrier:

for group in [group2,group3]:

send(group,emother,True,game)

elif target and carrier:

for group in [group2]:

if can\_push\_twice(group,target,game):

push\_twice(group,target,game)

else:

send(group,target,False,game)

for group in [group3]:

send(group,carrier,True,game)

else:

for group in [group2,group3]:

send(group,emother,True,game)