**EXTENSION REPORT**

By Le Bao Duy Nguyen – 102449993

**EXTENSION 1 – T03 – TIC TAC TOE**

The goal of the extension of this game is to create a more advanced twist to the game. By letting the user the options to choose who they’d like to face , or see other AI’s play against each other, adds diversity to the game.

from random import randrange

def reset\_game\_data():

'''Resets the game data in the global variables to the defaults'''

global board, current\_player, ai\_choice, ai\_choices, ai\_vs\_ai, players, winner, move, firstTurn, quitting

board = [' '] \* 9

current\_player = '' *# 'x' or 'o' for first and second player*

ai\_choices = {'r': 'Random', 'a': 'Average Ai', 's': 'Smart AI'}

ai\_choice = None

ai\_vs\_ai = False

players = {'x': 'Human', 'o': 'Super AI'}

winner = None

move = None

firstTurn = True

quitting = False

def check\_set\_for\_player(set, player):

count = 0

move = -1

for x, index in enumerate(set):

if board[index] == player:

count += 1

elif board[index] != 'x' and board[index] != 'o':

move = index

if x == 2 and count != 2:

*# if the count is at two when we have ennumerated through the set then we set move*

*# to -1 to indicate to return false*

move = -1

else:

move = -1

return move

*# agent (human or AI) functions*

def get\_human\_move():

'''Get a human players raw input. Returns None if a number is not entered.'''

return input('[0-8] >> ')

def get\_ai\_move():

'''Get the AI's next move '''

*# A simple dumb random move - valid or NOT!*

*# Note: It is the models responsibility to check for valid moves...*

return randrange(9) *# [0..8]*

def get\_average\_ai\_move():

global current\_player

if current\_player == 'x':

otherPlayer = 'o'

else:

otherPlayer = 'x'

for set in WIN\_SET:

*# Check if other player is about to win using check\_set().*

chk = check\_set\_for\_player(set, otherPlayer)

if chk != -1:

return chk

*# Then make the move to stop the other player from winning*

*#else choose a random option*

return randrange(9) *# [0..8]*

def get\_smart\_ai\_move():

global current\_player, firstTurn

if current\_player == 'x':

otherPlayer = 'o'

else:

otherPlayer = 'x'

for set in WIN\_SET:

*# Check if this player is about to win using check\_set()*

chk = check\_set\_for\_player(set, current\_player)

if chk != -1:

return chk

*# Then make the move that allows you to win*

*# Check if other player is about to win using check\_set().*

chk = check\_set\_for\_player(set, otherPlayer)

if chk != -1:

return chk

*# Then make the move to stop the other player from winning*

*# if its the first turn, return the middle*

if firstTurn:

firstTurn = False *# set this to False so it only tries this once.*

return 4

*# If neither condition*

*# Then make a random move from available spaces*

return randrange(9) *# [0..8]*

*#==============================================================================*

*# Standard trinity of game loop methods (functions)*

def process\_input():

'''Get the current players next move.'''

*# save the next move into a global variable*

global move, ai\_choice

if current\_player == 'x':

move = get\_human\_move()

elif ai\_choice == 's':

move = get\_smart\_ai\_move()

elif ai\_choice == 'a':

move = get\_average\_ai\_move()

elif ai\_choice == 'r':

move = get\_ai\_move()

else:

move = get\_ai\_move() *# Defaults to the random AI*

def process\_ai\_vs\_ai\_input():

'''Get the current players next move, where there are two ai battling.'''

*# save the next move into a global variable*

global move, ai\_choice

if current\_player == 'x':

move = get\_smart\_ai\_move() *# Always Smart AI vs another AI*

elif ai\_choice == 's':

move = get\_smart\_ai\_move()

elif ai\_choice == 'a':

move = get\_average\_ai\_move()

elif ai\_choice == 'r':

move = get\_ai\_move()

else:

move = get\_ai\_move() *# Defaults to the random AI*

def run\_human\_vs\_ai\_game():

'''Run a Human Vs AI game'''

show\_human\_help()

*# by default the human player starts. This could be random or a choice.*

global current\_player

current\_player = 'x'

*# show the initial board and the current player's move*

render\_board()

*# Standard game loop structure*

while winner is None:

process\_input()

update\_model()

render\_board()

def run\_ai\_vs\_ai\_game():

'''Run a game between a Smart AI and a selected AI'''

*# by default 'x' starts*

global current\_player

current\_player = 'x'

*# Standard game loop structure*

while winner is None:

process\_ai\_vs\_ai\_input()

update\_model()

*# Render the Final Board State*

render\_board()

if \_\_name\_\_ == '\_\_main\_\_':

*# Welcome ...*

print('Welcome to the amazing+awesome tic-tac-toe! \n')

while not quitting:

*# Choose to play or have the AI fight it out*

print('Do you want the smart AI to fight on your behalf?')

choice = input ('[Y/N] -> ')

if choice == 'Y' or choice == 'y':

ai\_vs\_ai = True

else:

ai\_vs\_ai = False

*# Select the AI opponent playing second*

print('\nSelect the opponent')

for key in ai\_choices.keys():

print(key, ai\_choices[key])

ai\_choice = input('>> ')

if ai\_vs\_ai:

run\_ai\_vs\_ai\_game()

else:

run\_human\_vs\_ai\_game()

print(HR)

if winner == 'tie':

print('TIE!')

elif winner in players:

print('%s is the WINNER!!!' % players[winner])

print(HR)

print('Play Again?')

tmp = input('[Y/N] -> ')

if tmp == 'Y' or tmp == 'y':

reset\_game\_data()

else:

quitting = True

print('Goodbye, Thank you for playing.')

**RESULT**

**A close up of text on a black background

Description automatically generated**

**A close up of text on a black background

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA close up of text on a black background

Description automatically generatedA picture containing drawing

Description automatically generated**

**EXTENSION 2 – T08 – LAB – STEERING #1 – SEEK ARRIVE FLEE**

Pursuit mode is added. Predicting where an agent will be in time T and seeks towards that point to intercept it.

def pursuit(self, evader):

self.toEvader = evader.pos - self.pos

self.relHeading = Vector2D.dot(evader.heading, self.heading)

if(Vector2D.dot(self.toEvader, self.heading) > 0 and self.relHeading < -0.95):

return self.seek(evader.pos)

lookAheadTime = Vector2D.length(self.toEvader) / (self.max\_speed + evader.speed())

return self.seek(evader.pos + evader.vel \* lookAheadTime)

A picture containing shirt

Description automatically generated

**EXTENSION 3 – T14 – AGENT MARKSMANSHIP**

Shotgun mode is added where it sent out waves of controllable number of bullets out. Slow but deadly (guaranteed to hit ).

if self.world.hunter.aim is True:

bullet\_speed = 20 if self.mode in ['Shotgun'] else 10

target\_pos = self.aim()

elif self.mode == 'Shotgun':

for i in range(5):

self.world.add(ShotgunBullet(self.init\_pos, enemy\_pos))

class ShotgunBullet(Bullet):

def \_\_init\_\_(self, firing\_pos, target\_pos):

Bullet.\_\_init\_\_(self, firing\_pos, target\_pos +

Vector2D(randrange(-100, 100), randrange(-100, 100)))

self.radius = 5

self.velocity = randrange(18, 22)

**A close up of a logo

Description automatically generated**

**EXTENSION 4 – T16 - GOAP**

A menu is made so that user can choose between options. Each option will guide them on what is their possibility to defeat Thanos. This increases the liveliness of an AI.

print(

"""--------------------------------

A: Do you want see the possibility of you saving the universe?

B: Quit --------------------------------""")

option = input("Enter option:")

if option.lower() == "a":

print("""--------------------------------

A: Steal Gem

B: Steal Gauntlet

C: Kill Thanos

D: Too hard ! Get me out --------------------------------""")

option = input("Enter your choice:")

if option.lower() == "a":

goal\_state = 'Has Gem'

path = agent.plan(goal\_state)

print('Goal: ' + goal\_state + "\n")

for i in range(len((path['Actions']))):

print(str(i+1) + ') ' + path['Actions'][i].name +' (' + str(path['Actions'][i].cost) + ')')

print('Win ratio: 1/' + str(path['Ratio']))

pass

elif option.lower() == "b":

goal\_state = 'Has Gauntlet'

path = agent.plan(goal\_state)

print('Goal: ' + goal\_state + "\n")

for i in range(len((path['Actions']))):

print(str(i+1) + ') ' + path['Actions'][i].name + ' (' + str(path['Actions'][i].cost) + ')')

print('Win ratio: 1/' + str(path['Ratio']))

pass

elif option.lower() == "c":

goal\_state = 'Defeat Thanos'

path = agent.plan(goal\_state)

print('Goal: ' + goal\_state + "\n")

for i in range(len((path['Actions']))):

print(str(i+1) + ') ' + path['Actions'][i].name +' (' + str(path['Actions'][i].cost) + ')')

print('Win ratio: 1/' + str(path['Ratio']))

pass

elif option.lower() == "d":

print("--------------------------------")

print("Understandable choice!")

pass

elif option.lower() == "b":

print("--------------------------------")

print("Understandable choice!")

pass

else:

print("You can only selct A or B")

print("Please try again")

pass

A screenshot of a cell phone

Description automatically generated