Maze Solving AI Using the Genetic Algorithm

December 9, 2022

[83]:

```
from enum import IntEnum
import random
```

[84]:

```
class Directions(IntEnum):
    NORTH = 0
    SOUTH = 1
    EAST = 2
    WEST = 3

direction = {'n': Directions.NORTH, 's': Directions.SOUTH, 'e': Directions.
 ↪EAST, 'w': Directions.WEST}
directionsSet = (Directions.NORTH, Directions.SOUTH, Directions.EAST,␣
 ↪Directions.WEST)
directionChar = {Directions.NORTH : 'N', Directions.SOUTH : 'S', Directions.
 ↪EAST : 'E', Directions.WEST : 'W'}

def DistanceForm(p1, p2):
    x1 = p1[0]
    y1 = p1[1]

    x2 = p2[0]
    y2 = p2[1]

    return (((x2 - x1)**2) + ((y2 - y1)**2))**.5

def pathStr(path):
    res = ""
    res += directionChar[Directions(path[0])]
    for i in range(1, len(path)):
        res += " "
        res += directionChar[Directions(path[i])]
    return res
```

[85]:

```
class Cell:
    def __init__(self, cellNum = 0, isStart = False, isFinish = False):
        self.north = None
        self.south = None
        self.east = None
        self.west = None
        self.isStart = isStart
        self.isFinish = isFinish
        self.cellNum = cellNum

    def setDirections(self, north = None, south = None, east = None, west = None):
        self.north = north
        self.south = south
        self.east = east
        self.west = west

    def getCellNum(self):
        return self.cellNum

    def finishStatus(self):
        return self.isFinish

    def startStatus(self):
        return self.isStart

    def makeStart(self):
        self.isStart = True

    def makeFinish(self):
        self.isFinish = True

    def getNorth(self):
        return self.north

    def getSouth(self):
        return self.south

    def getEast(self):
        return self.east

    def getWest(self):
        return self.west

    def setNorth(self, cell):
        self.north = cell

    def setSouth(self, cell):
        self.south = cell

    def setEast(self, cell):
```

```python
        self.east = cell

    def setWest(self, cell):
        self.west = cell

    def connectCells(self, other, direction):
        match direction:
            case Directions.NORTH:
                self.setNorth(other)
                other.setSouth(self)
            case Directions.SOUTH:
                self.setSouth(other)
                other.setNorth(self)
            case Directions.EAST:
                self.setEast(other)
                other.setWest(self)
            case Directions.WEST:
                self.setWest(other)
                other.setEast(self)

    def hasNorth(self):
        return self.north is not None

    def hasSouth(self):
        return self.south is not None

    def hasEast(self):
        return self.east is not None

    def hasWest(self):
        return self.west is not None

    def getCoords(self, nrows, ncols):
        row = self.cellNum % nrows
        col = ncols - (self.cellNum // ncols)
        return (row, col)

    def hasDir(self, direction):
        match direction:
            case Directions.NORTH:
                return self.hasNorth()
            case Directions.SOUTH:
                return self.hasSouth()
            case Directions.EAST:
                return self.hasEast()
            case Directions.WEST:
                return self.hasWest()
            case _:
                raise ValueError

    def atDeadEnd(self, prevDir):
        match prevDir:
            case Directions.NORTH:
                return not(self.hasSouth()) and not(self.hasEast()) and␣
↪not(self.hasWest())
            case Directions.SOUTH:
                return not(self.hasNorth()) and not(self.hasEast()) and␣
↪not(self.hasWest())
            case Directions.EAST:
                return not(self.hasNorth()) and not(self.hasSouth()) and␣
↪not(self.hasWest())
            case Directions.WEST:
                return not(self.hasNorth()) and not(self.hasSouth()) and␣
↪not(self.hasEast())
            case _:
                raise ValueError
```

[92]:

```
class Maze:
    def __init__(self, nrows = 0, ncols = 0):
        self.startCell = None
        self.finishCell = None
        self.cells = []
        self.runningumber = 0
        self.nrows = nrows
        self.ncols = ncols

    def getStartCell(self):
        return self.startCell

    def setRowsCols(self, nrows, ncols):
        self.nrows = nrows
        self.ncols = ncols

    def getRowsCols(self):
        return (self.nrows, self.ncols)

    def __len__(self):
        return len(self.cells)

    def __getitem__(self, index):
        return self.cells[index]

    def __setitem__(self, index, value):
        self.cells[index] = value
    def __contains__(self, index):
        return index in self.cells

    def __iter__(self):
        return self.cells.__iter__()

    '''
    10x10 Maze:
    Start Cell: 0
    Finish Cell: 99
    0  1--2  3  4--5  6--7  8  9
    |  |  |  |  |  |  |     |  |
    10-11 12-13-14 15-16-17-18-19
    |  |  |  |        |     |  |
    20-21 22 23-24-25 26-27 28-29
        |  |     |  |  |  |  |  |
    30-31-32-33 34 35 36 37-38 39
        |     |     |     |
    40 41-42 43 44-45 46-47-48-49
    |        |  |     |     |
    50-51 52-53 54-55-56-57 58-59
    |     |  |  |  |        |
    60-61 62 63-64 65-66-67 68 69
    |     |  |     |     |     |
    70-71 72-73 74-75-76 77 78-79
    |  |     |  |     |  |  |  |
    80 81-82 83-84-85 86 87-88 89
        |  |     |  |     |     |
    90-91 92-93 94 95 96-97-98 99
    '''
    def build10x10Maze(self):
        for i in range(100):
            self.cells.append(Cell(cellNum=i))
        with open("maze_file.txt", "r+") as f:
            for line in f:
                l = line.strip().split()
                for c in l:
                    cleaned = c[1:-1]
                    contents = cleaned.split(",")
                    cellNum = int(contents[0])
                    dirs = contents[1:]
                    for dir in dirs:
                        match dir:
                            case 'N':
                                self[cellNum].connectCells(self[cellNum-10], Directions.NORTH)
                            case 'S':
                                self[cellNum].connectCells(self[cellNum+10],␣
↪Directions.SOUTH)
                            case 'E':
                                self[cellNum].connectCells(self[cellNum+1],␣
↪Directions.EAST)
                            case 'W':
                                self[cellNum].connectCells(self[cellNum-1],␣
↪Directions.WEST)
                            case _:
                                raise KeyError
        self[0].makeStart()
        self.startCell = self[0]
        self[99].makeFinish()
        self.finishCell = self[99]


    '''
    3x3 Maze:
    Start Cell: 0
    Finish Cell: 8
    0-1-2
    |
    3-4-5
    |   |
    6-7 8
    '''
    def build3x3Maze(self):
        cell0 = Cell(0, isStart=True)
        cell1 = Cell(1)
        cell2 = Cell(2)
        cell3 = Cell(3)
        cell4 = Cell(4)
        cell5 = Cell(5)
        cell6 = Cell(6)
        cell7 = Cell(7)
        cell8 = Cell(8, isFinish=True)
        #cell0.south = cell3
        cell0.east = cell1

        cell1.south = cell4
        cell1.east = cell2
        cell1.west = cell0

        #cell2.south = cell5
        cell2.west = cell1
        #cell3.north = cell0
        cell3.east = cell4
        cell3.south = cell6

        cell4.north = cell1
        #cell4.south = cell7
        cell4.east = cell5
        cell4.west = cell3

        #cell5.north = cell2
        cell5.south = cell8
        cell5.west = cell4

        cell6.north = cell3
        cell6.east = cell7

        #cell7.north = cell4
        #cell7.east = cell8
        cell7.west = cell6

        cell8.north = cell5
        #cell8.west = cell7

        self.cells.append(cell0)
        self.cells.append(cell1)
        self.cells.append(cell2)
        self.cells.append(cell3)
        self.cells.append(cell4)
        self.cells.append(cell5)
        self.cells.append(cell6)
        self.cells.append(cell7)
        self.cells.append(cell8)

        self.startCell = cell0
        cell0.makeStart()
        self.finishCell = cell8
        cell8.makeFinish()

    '''
    5x5 Maze:
    Start Cell: 0
    Finish Cell: 24
    0--1--2  3  4
        |  |  |  |
    5--6  7--8--9
    |  |     |
    10-11 12-13-14
    |        |  |
    15-16 17-18 19
    |  |  |  |  |
    20 21-22 23-24
    '''
    def build5x5Maze(self):
        cell0 = Cell(0, isStart=True)
        cell1 = Cell(1)
        cell2 = Cell(2)
        cell3 = Cell(3)
        cell4 = Cell(4)
        cell5 = Cell(5)
        cell6 = Cell(6)
        cell7 = Cell(7)
        cell8 = Cell(8)
        cell9 = Cell(9)
        cell10 = Cell(10)
        cell11 = Cell(11)
        cell12 = Cell(12)
        cell13 = Cell(13)
        cell14 = Cell(14)
        cell15 = Cell(15)
        cell16 = Cell(16)
        cell17 = Cell(17)
        cell18 = Cell(18)
        cell19 = Cell(19)
        cell20 = Cell(20)
        cell21 = Cell(21)
        cell22 = Cell(22)
        cell23 = Cell(23)
        cell24 = Cell(24, isFinish=True)

        #cell0.south = cell5
        cell0.east = cell1

        cell1.south = cell6
        cell1.east = cell2
        cell1.west = cell0

        cell2.south = cell7
        #cell2.east = cell3
        cell2.west = cell1

        cell3.south = cell8
        #cell3.east = cell4
        #cell3.west = cell2
        cell4.south = cell9
        #cell4.west = cell3

        #cell5.north = cell0
        cell5.south = cell10
        cell5.east = cell6

        cell6.north = cell1
        cell6.south = cell11
        #cell6.east = cell7
        cell6.west = cell5

        cell7.north = cell2
        #cell7.south = cell12
        cell7.east = cell8
        #cell7.west = cell6

        cell8.north = cell3
        cell8.south = cell13
        cell8.east = cell9
        cell8.west = cell7

        cell9.north = cell4
        #cell9.south = cell14
        cell9.west = cell8

        cell10.north = cell5
        cell10.south = cell15
        cell10.east = cell11

        cell11.north = cell6
        #cell11.south = cell16
        #cell11.east = cell12
        cell11.west = cell10

        #cell12.north = cell7
        #cell12.south = cell17
        cell12.east = cell13
        #cell12.west = cell11

        cell13.north = cell8
        cell13.south = cell18
        cell13.east = cell14
        cell13.west = cell12

        #cell14.north = cell9
        cell14.south = cell19
        cell14.west = cell13

        cell15.north = cell10
        cell15.south = cell20
        cell15.east = cell16

        #cell16.north = cell11
        cell16.south = cell21
        #cell16.east = cell17
        cell16.west = cell15

        #cell17.north = cell12
        cell17.south = cell22
        cell17.east = cell18
        #cell17.west = cell16

        cell18.north = cell13
        cell18.south = cell23
        #cell18.east = cell19
        cell18.west = cell17

        cell19.north = cell14
        cell19.south = cell24
        #cell19.west = cell18

        cell20.north = cell15
        #cell20.east = cell21

        cell21.north = cell16
        cell21.east = cell22
        #cell21.west = cell20

        cell22.north = cell17
        #cell22.east = cell23
        cell22.west = cell21

        cell23.north = cell18
        cell23.east = cell24
        #cell23.west = cell22

        cell24.north = cell19
        cell24.west = cell23

        self.cells.append(cell0)
        self.cells.append(cell1)
        self.cells.append(cell2)
        self.cells.append(cell3)
        self.cells.append(cell4)
        self.cells.append(cell5)
        self.cells.append(cell6)
        self.cells.append(cell7)
        self.cells.append(cell8)
        self.cells.append(cell9)
        self.cells.append(cell10)
        self.cells.append(cell11)
        self.cells.append(cell12)
        self.cells.append(cell13)
        self.cells.append(cell14)
        self.cells.append(cell15)
        self.cells.append(cell16)
        self.cells.append(cell17)
        self.cells.append(cell18)
        self.cells.append(cell19)
        self.cells.append(cell20)
        self.cells.append(cell21)
        self.cells.append(cell22)
        self.cells.append(cell23)
        self.cells.append(cell24)

        self.startCell = cell0
        cell0.makeStart()
        self.finishCell = cell24
        cell24.makeFinish()

    def defineCellNum(self):
        self.runningumber += 1
        self.cellNumbers.add(self.runningumber)
        return self.runningumber

    def buildMazeCell(self, cell, directions):
        north = None
        south = None
        east = None
        west = None
        for dir in directions:
            cellNum = self.defineCellNum()
            match dir:
                case Directions.NORTH:
                    north = Cell(cellNum)
                    cell.connectCells(north, Directions.NORTH)
                case Directions.SOUTH:
                    south = Cell(cellNum)
                    cell.connectCells(south, Directions.SOUTH)
                case Directions.EAST:
                    east = Cell(cellNum)
                    cell.connectCells(east, Directions.EAST)
                case Directions.WEST:
                    west = Cell(cellNum)
                    cell.connectCells(west, Directions.WEST)
        cell.setDirections(north, south, east, west)

    def enterMaze(self):
        return self.startCell

    def getFitness(self, currentCell):
        curCoords = currentCell.getCoords(self.nrows, self.ncols)
        endCoords = self.finishCell.getCoords(self.nrows, self.ncols)
        return DistanceForm(curCoords, endCoords)

    def testValidPath(self, path):
        cell = self.startCell
        for i in path:
            dir = Directions(i)
            match i:
                case Directions.NORTH:
                    if(cell.hasNorth()):
                        cell = cell.getNorth()
                    else:
                        return False
                case Directions.SOUTH:
                    if(cell.hasSouth()):
                        cell = cell.getSouth()
                    else:
                        return False
                case Directions.EAST:
                    if(cell.hasEast()):
                        cell = cell.getEast()
                    else:
                        return False
                case Directions.WEST:
                    if(cell.hasWest()):
                        cell = cell.getWest()
                    else:
                        return False
        if(cell.getCellNum() == self.finishCell.getCellNum()):
            return True
        else:
            return False
```

[87]:

```
class Player:
    def __init__(self, startCell = Cell(isStart = True)):
        self.current_cell = startCell
        self.fitness = 0
        self.path = []
        self.maze = None

    def enterMaze(self, maze):
        self.current_cell = maze.getStartCell()
        self.maze = maze

    def goNorth(self):
        if(self.current_cell.getNorth() is not None):
            self.current_cell = self.current_cell.getNorth()
        else:
            print("Cannot go North")

    def goSouth(self):
        if(self.current_cell.getSouth() is not None):
            self.current_cell = self.current_cell.getSouth()
        else:
            print("Cannot go South")

    def goEast(self):
        if(self.current_cell.getEast() is not None):
            self.current_cell = self.current_cell.getEast()
        else:
            print("Cannot go East")

    def goWest(self):
        if(self.current_cell.getWest() is not None):
            self.current_cell = self.current_cell.getWest()
        else:
            print("Cannot go West")

    def move(self, direction):
        match direction:
            case Directions.NORTH:
                self.goNorth()
            case Directions.SOUTH:
                self.goSouth()
            case Directions.EAST:
                self.goEast()
            case Directions.WEST:
                self.goWest()
            case _:
                raise ValueError

    def checkFinish(self):
        if self.current_cell.finishStatus():
            print("Reached Finish")
            return True
        else:
            return False

    def checkStart(self):
        if self.current_cell.startStatus():
            print("You're at the start")
            return True
        else:
            return False

    def hasNorth(self):
        return self.current_cell.hasNorth()

    def hasSouth(self):
        return self.current_cell.hasSouth()

    def hasEast(self):
        return self.current_cell.hasEast()

    def hasWest(self):
        return self.current_cell.hasWest()

    def lookAround(self):
        msg = "You can go "
        if(self.hasNorth()):
            msg += "North "
        if(self.hasSouth()):
            msg += "South "
        if(self.hasEast()):
            msg += "East "
        if(self.hasWest()):
            msg += "West"
        print(msg)

    def getCurrentCell(self):
        return self.current_cell

    def setFitness(self, fitness):
        self.fitness = fitness

    def getFitness(self):
        return self.fitness

    def setPath(self, path):
        self.path = path.copy()

    def getPath(self):
        return self.path

    def runPath(self, maze):
        for i in range(len(self.path)):
            dir = Directions(self.path[i])
            if(i > 0):
                prev = Directions(self.path[i-1])
                if(self.current_cell.atDeadEnd(prev)):
                    self.fitness = .00000000001
                    return .00000000001
            if(not(self.current_cell.hasDir(dir))):
                self.fitness = .00000000001
                return .00000000001
            self.move(dir)
            if(self.checkFinish()):
                self.fitness = 1
                self.path = self.path[:i+1]
                current_cellnum = self.current_cell.getCellNum()
                return 1
        perf = maze.getFitness(self.current_cell)
        if(perf == 0):
            self.fitness = 1
            return 1
        self.fitness = 1/perf
        return 1/perf
```

[111]:

```
# Agent
def generatePlayers(k, _maze):
    players = []
    for i in range(k):
        newPlayer = Player()
        newPlayer.enterMaze(maze=_maze)
        players.append(newPlayer)
    return players


def generateStarts(_players, lengthOfPath):
    k = len(_players)
    for org in range(k):
        path = []
        for i in range(lengthOfPath):
            path.append(random.randint(0,3))
        _players[org].setPath(path)
```

```
def runGeneration(_players, maze):
    fitnesses = []
    for player in _players:
        player.enterMaze(maze)
        player.runPath(maze)
        fitnesses.append(player.getFitness())
        if(fitnesses[-1] == 1):
            break
    string = ""
    string += str(fitnesses[0])
    return fitnesses

def selectParents(_players, fitns):
    fitnesses = fitns.copy()
    indices = [i for i in range(len(_players))]
    ip1 = random.choices(indices, weights=fitnesses)[0]
    ip2 = random.choices(indices, weights=fitnesses)[0]
    while(ip1 == ip2):
        index = indices.index(ip2)
        del indices[index]
        del fitnesses[index]
        ip2 = random.choices(indices, weights=fitnesses)[0]
    p1 = _players[ip1]
    p2 = _players[ip2]

    assert(len(p1.getPath()) == len(p2.getPath()))
    slicePoint = random.randint(0, len(p1.getPath()) - 1)

    childPath1 = p1.getPath()[:slicePoint] + p2.getPath()[slicePoint:]
    childPath2 = p2.getPath()[:slicePoint] + p1.getPath()[slicePoint:]

    return (childPath1, childPath2)

def mutate(numMutations, player):
    muts = set()
    while(len(muts) < numMutations):
        muts.add(random.randint(0, len(player.getPath()) - 1))
    for mut in muts:
        player.getPath()[mut] = random.randint(0, 3)

def generateChildren(_players, fitnesses, numberOfMutations):
    k = len(_players)
    children = []
    while(len(children) < k):
        children_i = selectParents(_players, fitnesses)
        children.append(children_i[0])
        children.append(children_i[1])
    for i in range(len(_players)):
        player = _players[i]
        child = children[i]
        player.setPath(child)
    for chld in _players:
        mutate(numberOfMutations, chld)

def geneticAlgorithm(maze, numPlayers, numberOfMutations, lengthOfPath):
    players = generatePlayers(numPlayers, maze)
    generateStarts(players, lengthOfPath)
    i = 0
    while(True):
    #for i in range(numGenerations):
        fitnesses = runGeneration(players, maze)
        if(1 in fitnesses):
            ind = fitnesses.index(1)
            solnPath = players[ind].getPath()
            if(len(solnPath) == 1):
                print("stop here")
            print("Found solution with path: " + pathStr(solnPath) + "\nAfter " + str(i) + " generations")
            break
        generateChildren(players, fitnesses, numberOfMutations)
        i += 1


def Solve3x3Maze():
    maze = Maze(3, 3)
    maze.build3x3Maze()
    numberOfStates = 100
    numberOfMutations = 5
    lengthOfPath = 25
    geneticAlgorithm(maze, numberOfStates, numberOfMutations, lengthOfPath)

def Solve5x5Maze():
    maze = Maze(5, 5)
    maze.build5x5Maze()
    numberOfStates = 100
    numberOfMutations = 50
    lengthOfPath = 100
    geneticAlgorithm(maze, numberOfStates, numberOfMutations, lengthOfPath)

def Solve10x10Maze():
    maze = Maze(10, 10)
    maze.build10x10Maze()
    numberOfStates = 1000
    numberOfMutations = 100
    lengthOfPath = 1000
    geneticAlgorithm(maze, numberOfStates, numberOfMutations, lengthOfPath)
```

[114]:

```
Solve3x3Maze()
```

```
Reached Finish
Found solution with path: E S E S
After 2 generations
```

[110]:

```
Solve5x5Maze()
```

```
Reached Finish
Found solution with path: E E S E S S S E
After 32 generations
```

[ ]:

```
Solve10x10Maze() # takes too long to run
```