Data Structures 2023-2

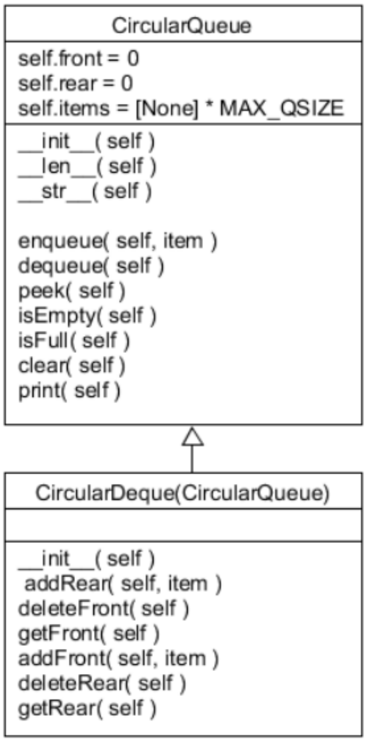
Lab 04: Queue Abstract Data Type

1. **Task-1: Implement Deque Data Structure**

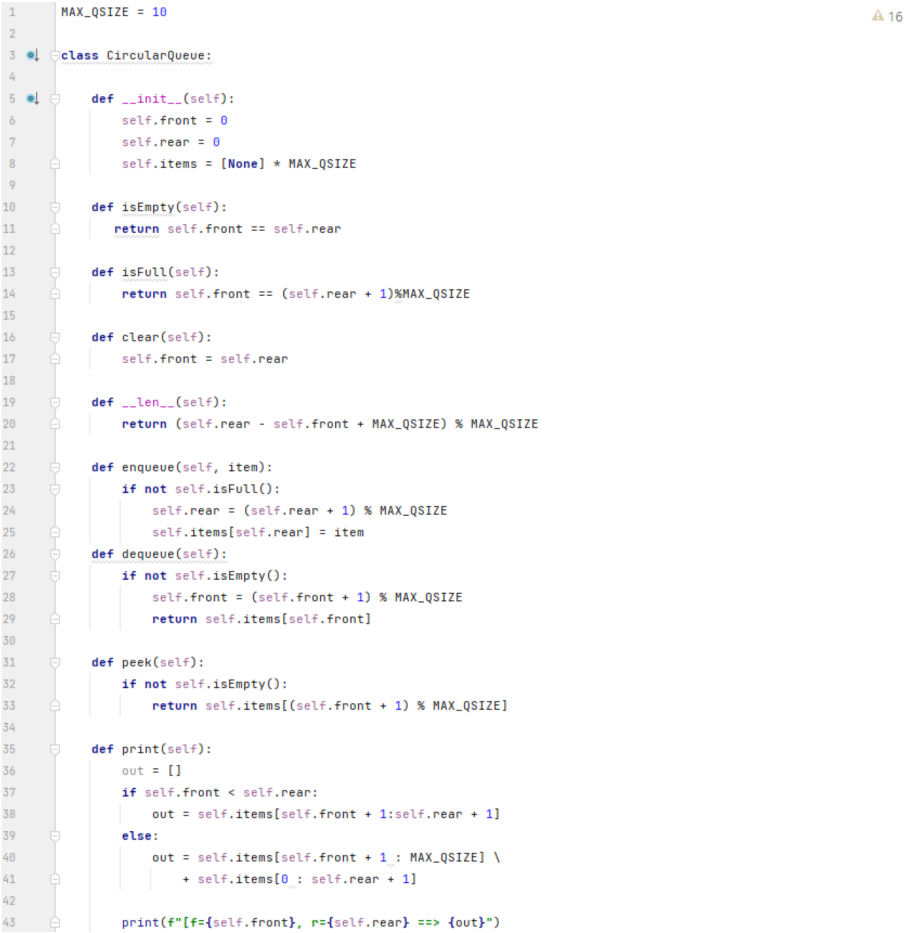
   Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends. Write code for Deque abstract data type and test it.

   Deque Operations: The following basic operations are performed on the deque:

   - ✓ addFront(): Adds an item at the front of Deque.
   - ✓ addRear(): Adds an item at the rear of Deque.
   - ✓ deleteFront(): Deletes an item from front of Deque.
   - ✓ deleteRear(): Deletes an item from rear of Deque.
   - ✓ getFront(): Gets the front item from queue.
   - ✓ getRear(): Gets the last item from queue.
   - ✓ isEmpty(): Checks whether Deque is empty or not.
   - ✓ isFull(): Checks whether Deque is full or not.

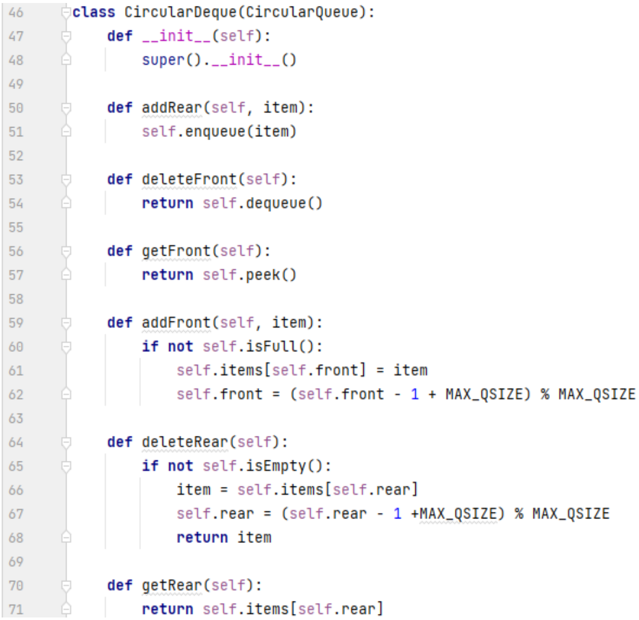


CircularQueue Code



```python
MAX_QSIZE = 10

class CircularQueue:

    def __init__(self):
        self.front = 0
        self.rear = 0
        self.items = [None] * MAX_QSIZE

    def isEmpty(self):
        return self.front == self.rear

    def isFull(self):
        return self.front == (self.rear + 1)%MAX_QSIZE

    def clear(self):
        self.front = self.rear

    def __len__(self):
        return (self.rear - self.front + MAX_QSIZE) % MAX_QSIZE

    def enqueue(self, item):
        if not self.isFull():
            self.rear = (self.rear + 1) % MAX_QSIZE
            self.items[self.rear] = item
    def dequeue(self):
        if not self.isEmpty():
            self.front = (self.front + 1) % MAX_QSIZE
            return self.items[self.front]

    def peek(self):
        if not self.isEmpty():
            return self.items[(self.front + 1) % MAX_QSIZE]

    def print(self):
        out = []
        if self.front < self.rear:
            out = self.items[self.front + 1:self.rear + 1]
        else:
            out = self.items[self.front + 1 : MAX_QSIZE] \
                + self.items[0 : self.rear + 1]

        print(f"[f={self.front}, r={self.rear} ==> {out}")
```

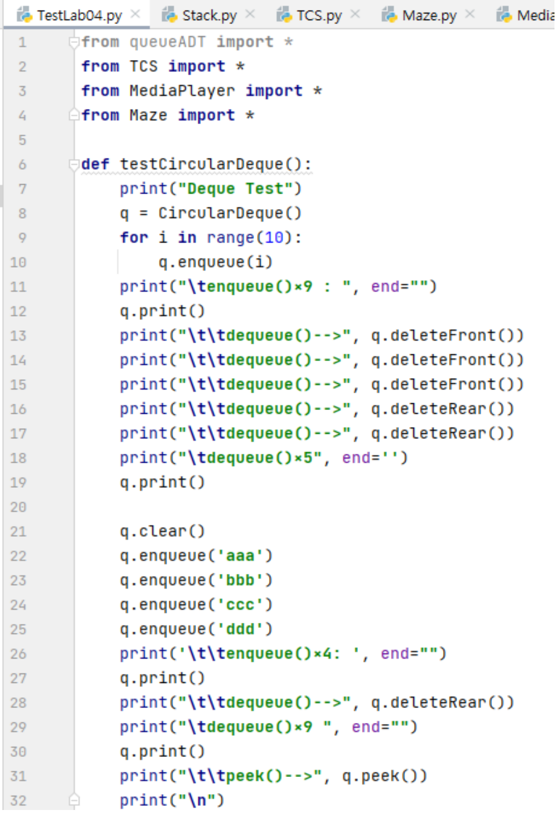
CirculatrDeque Code



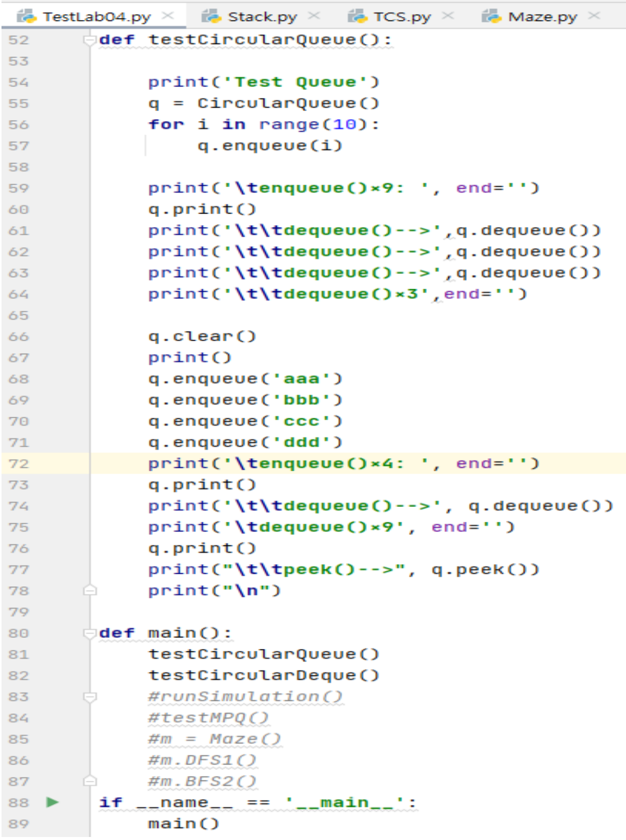
```python
class CircularDeque(CircularQueue):
    def __init__(self):
        super().__init__()

    def addRear(self, item):
        self.enqueue(item)

    def deleteFront(self):
        return self.dequeue()

    def getFront(self):
        return self.peek()

    def addFront(self, item):
        if not self.isFull():
            self.items[self.front] = item
            self.front = (self.front - 1 + MAX_QSIZE) % MAX_QSIZE

    def deleteRear(self):
        if not self.isEmpty():
            item = self.items[self.rear]
            self.rear = (self.rear - 1 +MAX_QSIZE) % MAX_QSIZE
            return item

    def getRear(self):
        return self.items[self.rear]
```

Results/Output

[Test code]

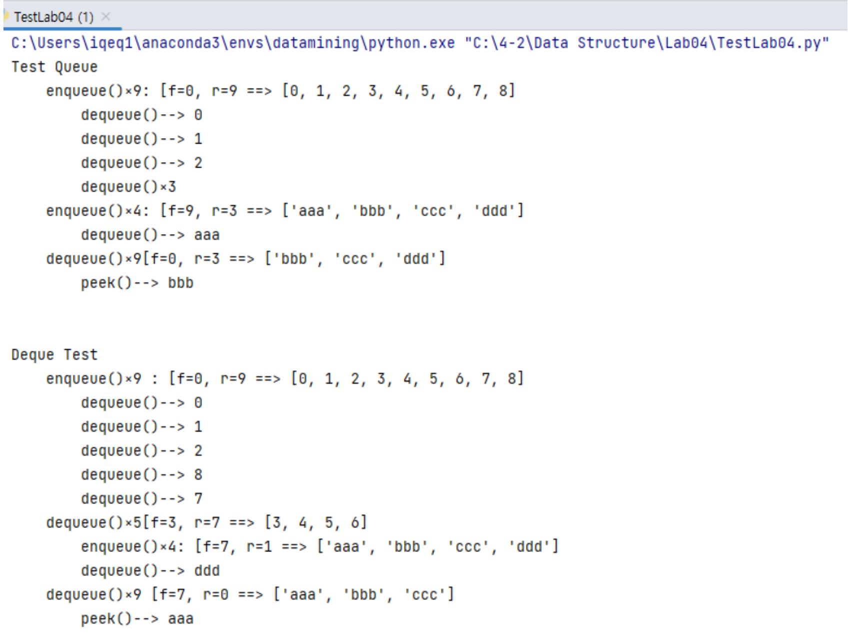


```python
from queueADT import *
from TCS import *
from MediaPlayer import *
from Maze import *

def testCircularDeque():
    print("Deque Test")
    q = CircularDeque()
    for i in range(10):
        q.enqueue(i)
    print("\tenqueue()×9 : ", end="")
    q.print()
    print("\t\tdequeue()-->", q.deleteFront())
    print("\t\tdequeue()-->", q.deleteFront())
    print("\t\tdequeue()-->", q.deleteFront())
    print("\t\tdequeue()-->", q.deleteRear())
    print("\t\tdequeue()-->", q.deleteRear())
    print("\tdequeue()×5", end='')
    q.print()

    q.clear()
    q.enqueue('aaa')
    q.enqueue('bbb')
    q.enqueue('ccc')
    q.enqueue('ddd')
    print('\t\tenqueue()×4: ', end="")
    q.print()
    print("\t\tdequeue()-->", q.deleteRear())
    print("\tdequeue()×9 ", end="")
    q.print()
    print("\t\tpeek()-->", q.peek())
    print("\n")
```



```python
def testCircularQueue():

    print('Test Queue')
    q = CircularQueue()
    for i in range(10):
        q.enqueue(i)

    print('\tenqueue()×9: ', end='')
    q.print()
    print('\t\tdequeue()-->',q.dequeue())
    print('\t\tdequeue()-->',q.dequeue())
    print('\t\tdequeue()-->',q.dequeue())
    print('\t\tdequeue()×3',end='')

    q.clear()
    print()
    q.enqueue('aaa')
    q.enqueue('bbb')
    q.enqueue('ccc')
    q.enqueue('ddd')
    print('\tenqueue()×4: ', end='')
    q.print()
    print('\t\tdequeue()-->', q.dequeue())
    print('\tdequeue()×9', end='')
    q.print()
    print("\t\tpeek()-->", q.peek())
    print("\n")

def main():
    testCircularQueue()
    testCircularDeque()
    #runSimulation()
    #testMPQ()
    #m = Maze()
    #m.DFS1()
    #m.BFS2()
if __name__ == '__main__':
    main()
```

[Test Queue and Test Deque]



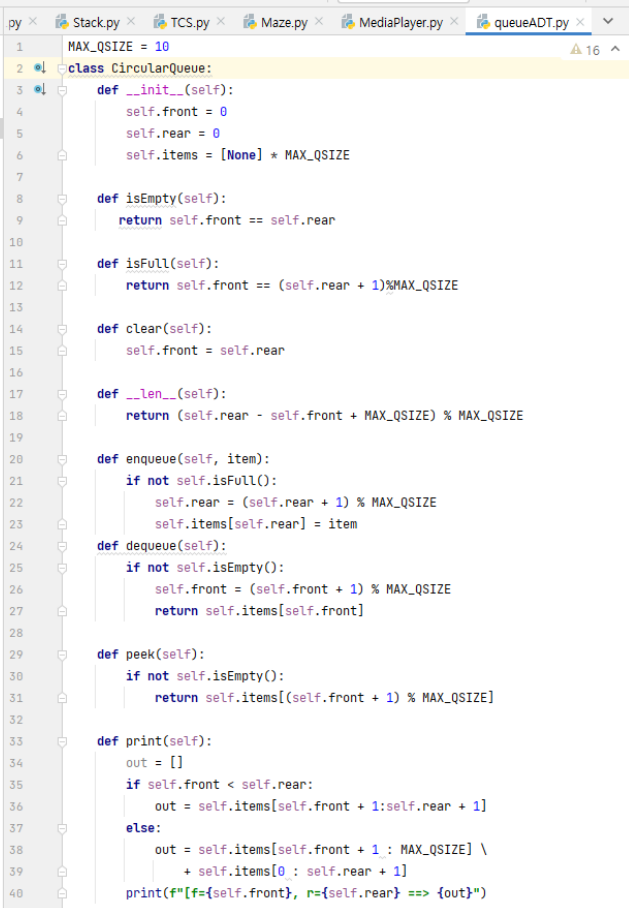
```
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab04\TestLab04.py"
Test Queue
    enqueue()×9: [f=0, r=9 ==> [0, 1, 2, 3, 4, 5, 6, 7, 8]
        dequeue()--> 0
        dequeue()--> 1
        dequeue()--> 2
        dequeue()×3
    enqueue()×4: [f=9, r=3 ==> ['aaa', 'bbb', 'ccc', 'ddd']
        dequeue()--> aaa
    dequeue()×9[f=0, r=3 ==> ['bbb', 'ccc', 'ddd']
        peek()--> bbb


Deque Test
    enqueue()×9 : [f=0, r=9 ==> [0, 1, 2, 3, 4, 5, 6, 7, 8]
        dequeue()--> 0
        dequeue()--> 1
        dequeue()--> 2
        dequeue()--> 8
        dequeue()--> 7
    dequeue()×5[f=3, r=7 ==> [3, 4, 5, 6]
        enqueue()×4: [f=7, r=1 ==> ['aaa', 'bbb', 'ccc', 'ddd']
        dequeue()--> ddd
    dequeue()×9 [f=7, r=0 ==> ['aaa', 'bbb', 'ccc']
        peek()--> aaa
```

2. **Task2:** Ticketing Counter system (Simulation)
   A computer simulation can be developed to model this Ticketing Counter system using the Queue data structure.

An object-oriented solution with multiple classes.

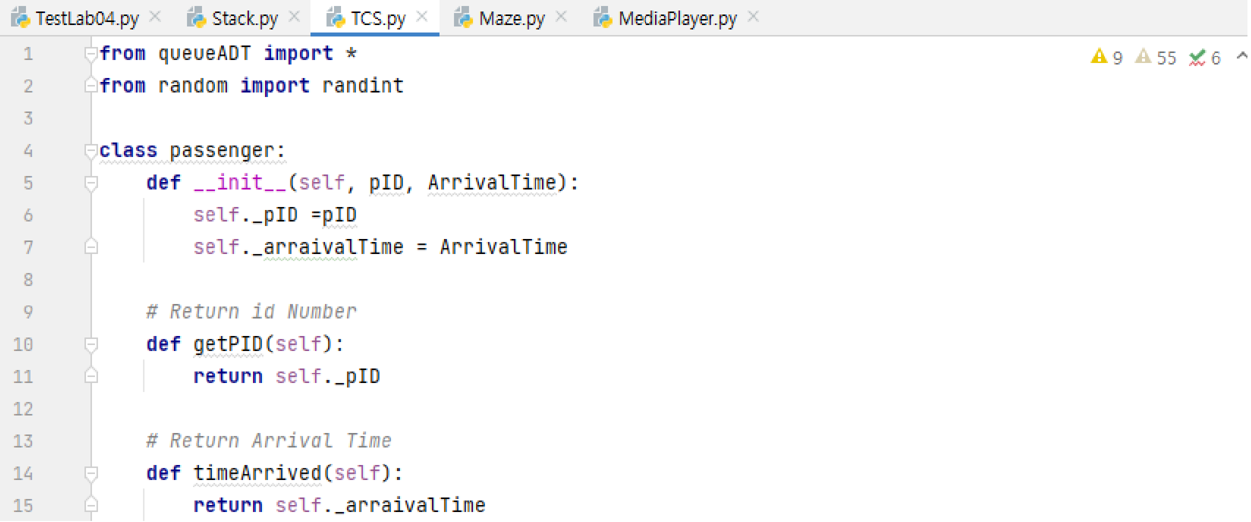
- ✓ CircularQueue : Data structure to hold the passengers.
- ✓ Passenger : store info related to a passenger.
- ✓ TicketAgent : store info related to an agent.
- ✓ TicketCounterSimulation : manages the actual simulation.

CircularQueue Code : Data structure to hold the passengers



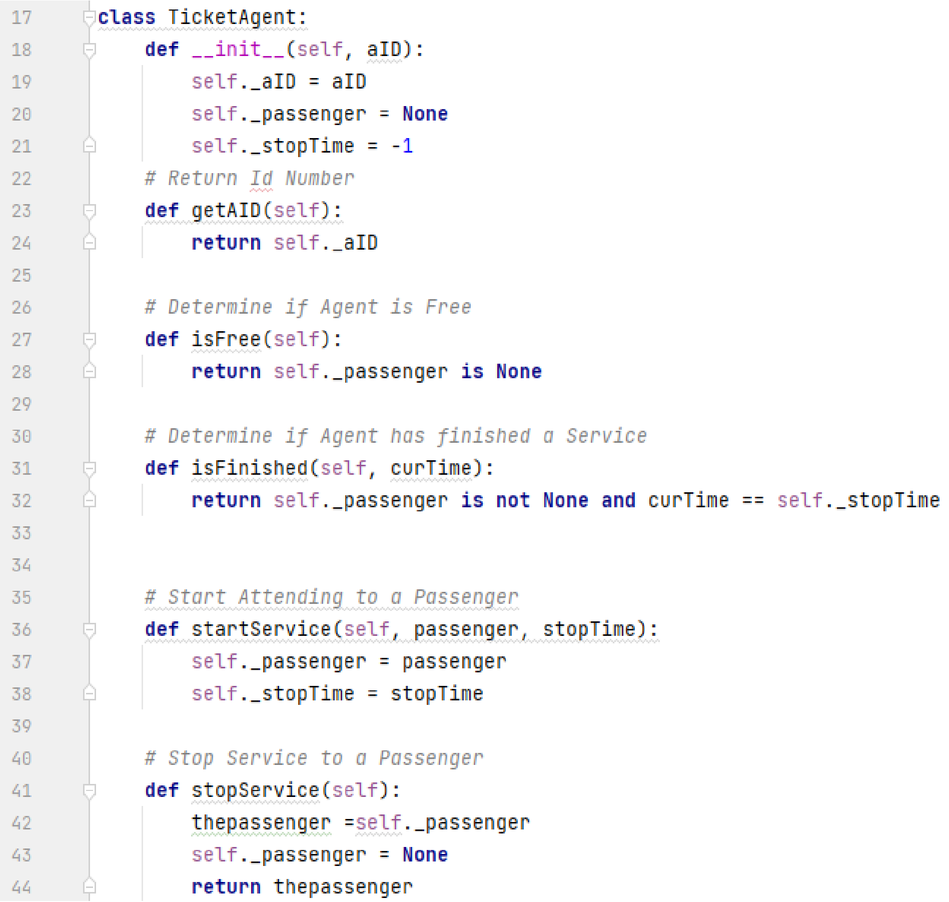
```python
MAX_QSIZE = 10
class CircularQueue:
    def __init__(self):
        self.front = 0
        self.rear = 0
        self.items = [None] * MAX_QSIZE

    def isEmpty(self):
       return self.front == self.rear

    def isFull(self):
        return self.front == (self.rear + 1)%MAX_QSIZE

    def clear(self):
        self.front = self.rear

    def __len__(self):
        return (self.rear - self.front + MAX_QSIZE) % MAX_QSIZE

    def enqueue(self, item):
        if not self.isFull():
            self.rear = (self.rear + 1) % MAX_QSIZE
            self.items[self.rear] = item
    def dequeue(self):
        if not self.isEmpty():
            self.front = (self.front + 1) % MAX_QSIZE
            return self.items[self.front]

    def peek(self):
        if not self.isEmpty():
            return self.items[(self.front + 1) % MAX_QSIZE]

    def print(self):
        out = []
        if self.front < self.rear:
            out = self.items[self.front + 1:self.rear + 1]
        else:
            out = self.items[self.front + 1 : MAX_QSIZE] \
                + self.items[0 : self.rear + 1]
        print(f"[f={self.front}, r={self.rear} ==> {out}")
```

passenger Code : store info related to a passenger.



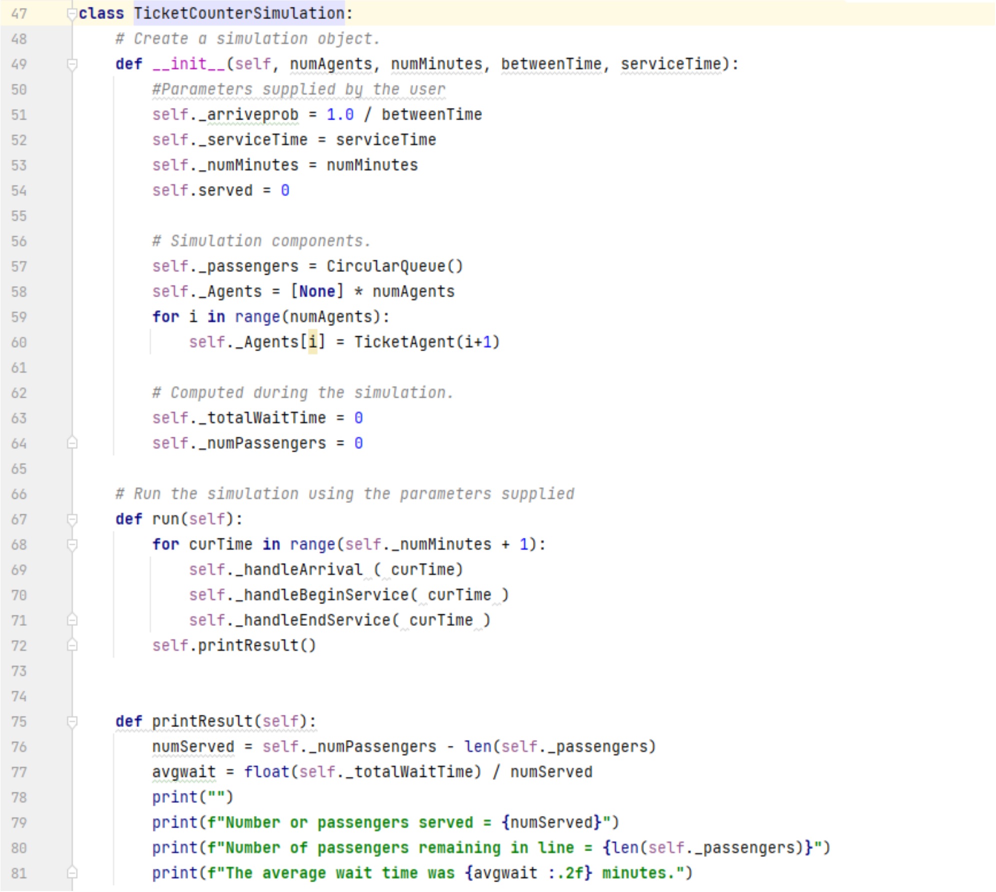
```python
from queueADT import *
from random import randint

class passenger:
    def __init__(self, pID, ArrivalTime):
        self._pID =pID
        self._arraivalTime = ArrivalTime

    # Return id Number
    def getPID(self):
        return self._pID

    # Return Arrival Time
    def timeArrived(self):
        return self._arraivalTime
```

TicketAgent Code : store info related to an agent.

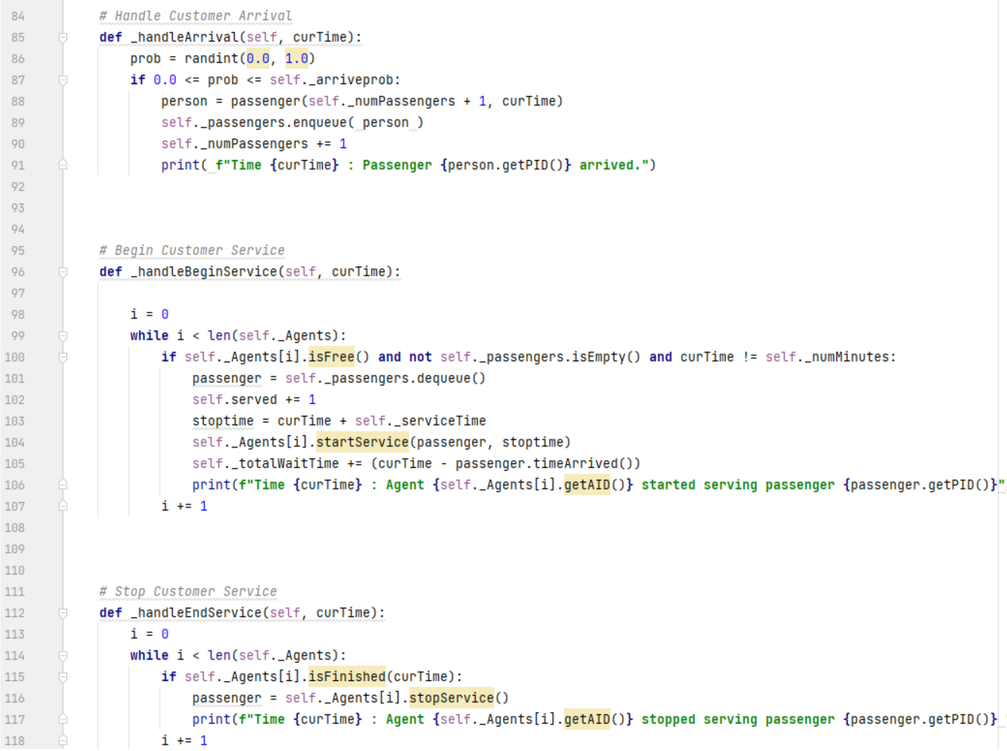


```python
class TicketAgent:
    def __init__(self, aID):
        self._aID = aID
        self._passenger = None
        self._stopTime = -1
    # Return Id Number
    def getAID(self):
        return self._aID

    # Determine if Agent is Free
    def isFree(self):
        return self._passenger is None

    # Determine if Agent has finished a Service
    def isFinished(self, curTime):
        return self._passenger is not None and curTime == self._stopTime


    # Start Attending to a Passenger
    def startService(self, passenger, stopTime):
        self._passenger = passenger
        self._stopTime = stopTime

    # Stop Service to a Passenger
    def stopService(self):
        thepassenger =self._passenger
        self._passenger = None
        return thepassenger
```

- ✓ **TicketCounterSimulation Code** : manages the actual simulation.

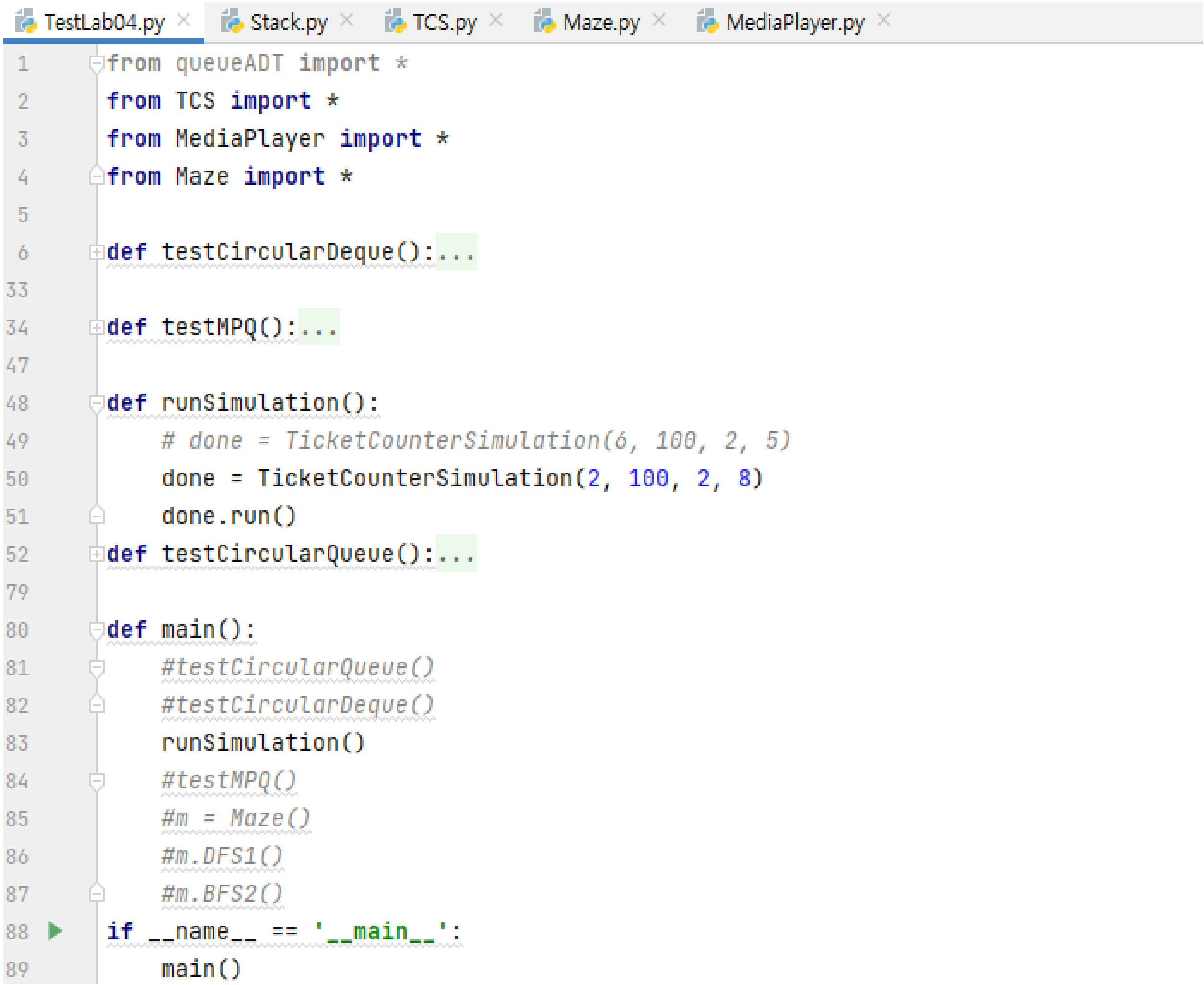


```python
class TicketCounterSimulation:
    # Create a simulation object.
    def __init__(self, numAgents, numMinutes, betweenTime, serviceTime):
        #Parameters supplied by the user
        self._arriveprob = 1.0 / betweenTime
        self._serviceTime = serviceTime
        self._numMinutes = numMinutes
        self.served = 0

        # Simulation components.
        self._passengers = CircularQueue()
        self._Agents = [None] * numAgents
        for i in range(numAgents):
            self._Agents[i] = TicketAgent(i+1)

        # Computed during the simulation.
        self._totalWaitTime = 0
        self._numPassengers = 0

    # Run the simulation using the parameters supplied
    def run(self):
        for curTime in range(self._numMinutes + 1):
            self._handleArrival ( curTime)
            self._handleBeginService( curTime )
            self._handleEndService( curTime )
        self.printResult()


    def printResult(self):
        numServed = self._numPassengers - len(self._passengers)
        avgwait = float(self._totalWaitTime) / numServed
        print("")
        print(f"Number or passengers served = {numServed}")
        print(f"Number of passengers remaining in line = {len(self._passengers)}")
        print(f"The average wait time was {avgwait :.2f} minutes.")
```



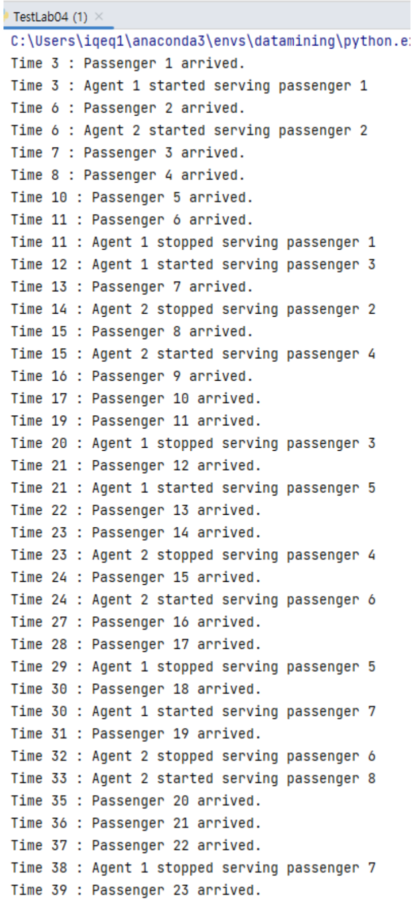
```python
    # Handle Customer Arrival
    def _handleArrival(self, curTime):
        prob = randint(0.0, 1.0)
        if 0.0 <= prob <= self._arriveprob:
            person = passenger(self._numPassengers + 1, curTime)
            self._passengers.enqueue( person )
            self._numPassengers += 1
            print( f"Time {curTime} : Passenger {person.getPID()} arrived.")



    # Begin Customer Service
    def _handleBeginService(self, curTime):

        i = 0
        while i < len(self._Agents):
            if self._Agents[i].isFree() and not self._passengers.isEmpty() and curTime != self._numMinutes:
                passenger = self._passengers.dequeue()
                self.served += 1
                stoptime = curTime + self._serviceTime
                self._Agents[i].startService(passenger, stoptime)
                self._totalWaitTime += (curTime - passenger.timeArrived())
                print(f"Time {curTime} : Agent {self._Agents[i].getAID()} started serving passenger {passenger.getPID()}")
            i += 1



    # Stop Customer Service
    def _handleEndService(self, curTime):
        i = 0
        while i < len(self._Agents):
            if self._Agents[i].isFinished(curTime):
                passenger = self._Agents[i].stopService()
                print(f"Time {curTime} : Agent {self._Agents[i].getAID()} stopped serving passenger {passenger.getPID()} ")
            i += 1
```

Results/Output Code

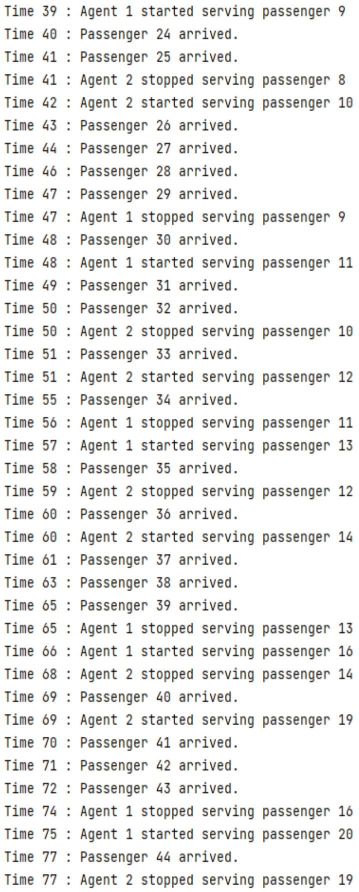


```python
from queueADT import *
from TCS import *
from MediaPlayer import *
from Maze import *

def testCircularDeque():...

def testMPQ():...

def runSimulation():
    # done = TicketCounterSimulation(6, 100, 2, 5)
    done = TicketCounterSimulation(2, 100, 2, 8)
    done.run()
def testCircularQueue():...

def main():
    #testCircularQueue()
    #testCircularDeque()
    runSimulation()
    #testMPQ()
    #m = Maze()
    #m.DFS1()
    #m.BFS2()
if __name__ == '__main__':
    main()
```

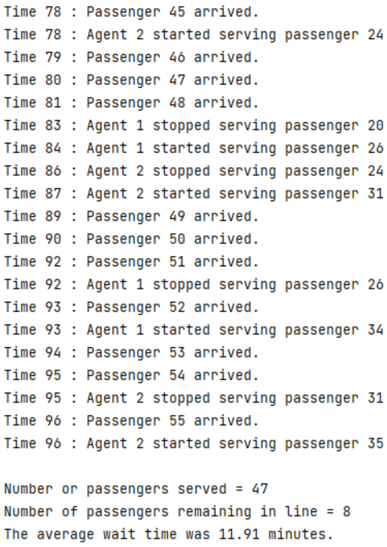
Results/Output



```
C:\Users\iqeq1\anaconda3\envs\datamining\python.e
Time 3 : Passenger 1 arrived.
Time 3 : Agent 1 started serving passenger 1
Time 6 : Passenger 2 arrived.
Time 6 : Agent 2 started serving passenger 2
Time 7 : Passenger 3 arrived.
Time 8 : Passenger 4 arrived.
Time 10 : Passenger 5 arrived.
Time 11 : Passenger 6 arrived.
Time 11 : Agent 1 stopped serving passenger 1
Time 12 : Agent 1 started serving passenger 3
Time 13 : Passenger 7 arrived.
Time 14 : Agent 2 stopped serving passenger 2
Time 15 : Passenger 8 arrived.
Time 15 : Agent 2 started serving passenger 4
Time 16 : Passenger 9 arrived.
Time 17 : Passenger 10 arrived.
Time 19 : Passenger 11 arrived.
Time 20 : Agent 1 stopped serving passenger 3
Time 21 : Passenger 12 arrived.
Time 21 : Agent 1 started serving passenger 5
Time 22 : Passenger 13 arrived.
Time 23 : Passenger 14 arrived.
Time 23 : Agent 2 stopped serving passenger 4
Time 24 : Passenger 15 arrived.
Time 24 : Agent 2 started serving passenger 6
Time 27 : Passenger 16 arrived.
Time 28 : Passenger 17 arrived.
Time 29 : Agent 1 stopped serving passenger 5
Time 30 : Passenger 18 arrived.
Time 30 : Agent 1 started serving passenger 7
Time 31 : Passenger 19 arrived.
Time 32 : Agent 2 stopped serving passenger 6
Time 33 : Agent 2 started serving passenger 8
Time 35 : Passenger 20 arrived.
Time 36 : Passenger 21 arrived.
Time 37 : Passenger 22 arrived.
Time 38 : Agent 1 stopped serving passenger 7
Time 39 : Passenger 23 arrived.
```



```
Time 39 : Agent 1 started serving passenger 9
Time 40 : Passenger 24 arrived.
Time 41 : Passenger 25 arrived.
Time 41 : Agent 2 stopped serving passenger 8
Time 42 : Agent 2 started serving passenger 10
Time 43 : Passenger 26 arrived.
Time 44 : Passenger 27 arrived.
Time 46 : Passenger 28 arrived.
Time 47 : Passenger 29 arrived.
Time 47 : Agent 1 stopped serving passenger 9
Time 48 : Passenger 30 arrived.
Time 48 : Agent 1 started serving passenger 11
Time 49 : Passenger 31 arrived.
Time 50 : Passenger 32 arrived.
Time 50 : Agent 2 stopped serving passenger 10
Time 51 : Passenger 33 arrived.
Time 51 : Agent 2 started serving passenger 12
Time 55 : Passenger 34 arrived.
Time 56 : Agent 1 stopped serving passenger 11
Time 57 : Agent 1 started serving passenger 13
Time 58 : Passenger 35 arrived.
Time 59 : Agent 2 stopped serving passenger 12
Time 60 : Passenger 36 arrived.
Time 60 : Agent 2 started serving passenger 14
Time 61 : Passenger 37 arrived.
Time 63 : Passenger 38 arrived.
Time 65 : Passenger 39 arrived.
Time 65 : Agent 1 stopped serving passenger 13
Time 66 : Agent 1 started serving passenger 16
Time 68 : Agent 2 stopped serving passenger 14
Time 69 : Passenger 40 arrived.
Time 69 : Agent 2 started serving passenger 19
Time 70 : Passenger 41 arrived.
Time 71 : Passenger 42 arrived.
Time 72 : Passenger 43 arrived.
Time 74 : Agent 1 stopped serving passenger 16
Time 75 : Agent 1 started serving passenger 20
Time 77 : Passenger 44 arrived.
Time 77 : Agent 2 stopped serving passenger 19
```

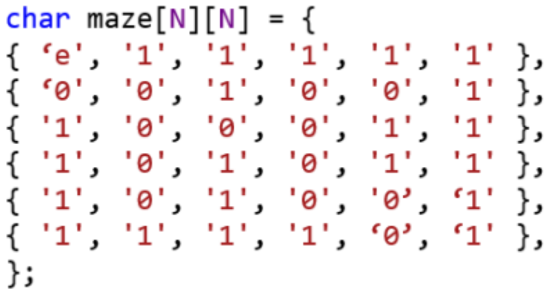


```
Time 78 : Passenger 45 arrived.
Time 78 : Agent 2 started serving passenger 24
Time 79 : Passenger 46 arrived.
Time 80 : Passenger 47 arrived.
Time 81 : Passenger 48 arrived.
Time 83 : Agent 1 stopped serving passenger 20
Time 84 : Agent 1 started serving passenger 26
Time 86 : Agent 2 stopped serving passenger 24
Time 87 : Agent 2 started serving passenger 31
Time 89 : Passenger 49 arrived.
Time 90 : Passenger 50 arrived.
Time 92 : Passenger 51 arrived.
Time 92 : Agent 1 stopped serving passenger 26
Time 93 : Passenger 52 arrived.
Time 93 : Agent 1 started serving passenger 34
Time 94 : Passenger 53 arrived.
Time 95 : Passenger 54 arrived.
Time 95 : Agent 2 stopped serving passenger 31
Time 96 : Passenger 55 arrived.
Time 96 : Agent 2 started serving passenger 35

Number or passengers served = 47
Number of passengers remaining in line = 8
The average wait time was 11.91 minutes.
```

3. **Task-3:** Solve the Maze problem through DFS and BFS using different data structures

A Maze is given as N×N binary matrix of blocks where source block is maze[0][0] and destination block is maze[N-1][N-1].

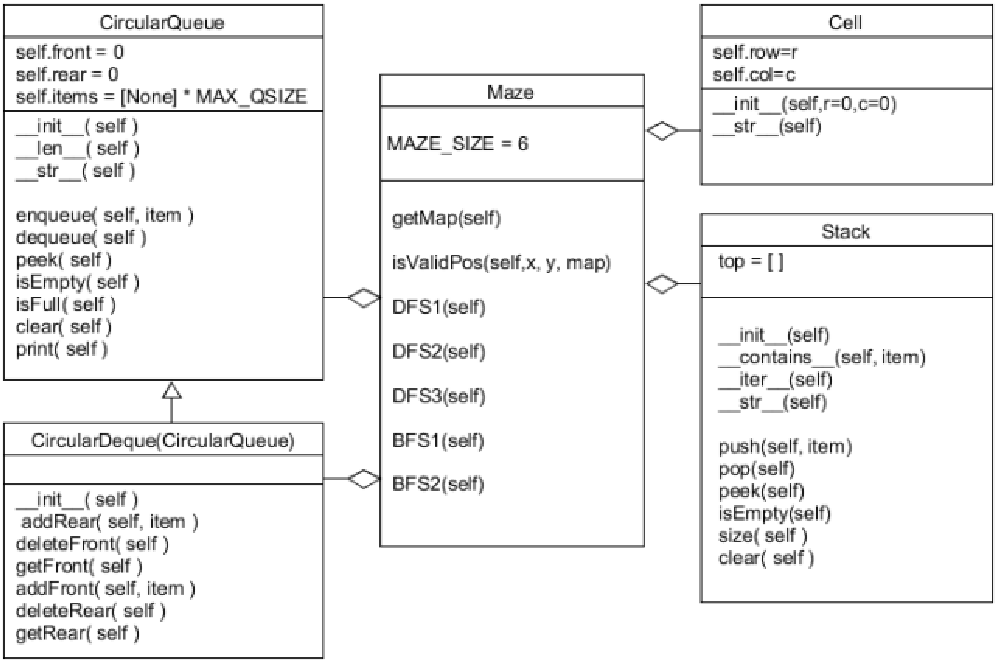
- ✓ A rat starts from the source and has to reach the destination. The rat can move only in four directions: upward, down, forward, and back word.
- ✓ In the maze matrix, 1 means the block is a dead end and 0 means the block can be used in the path from source to destination
- ✓ The task is to check if there exists any path so that the rat can reach at the destination or not.



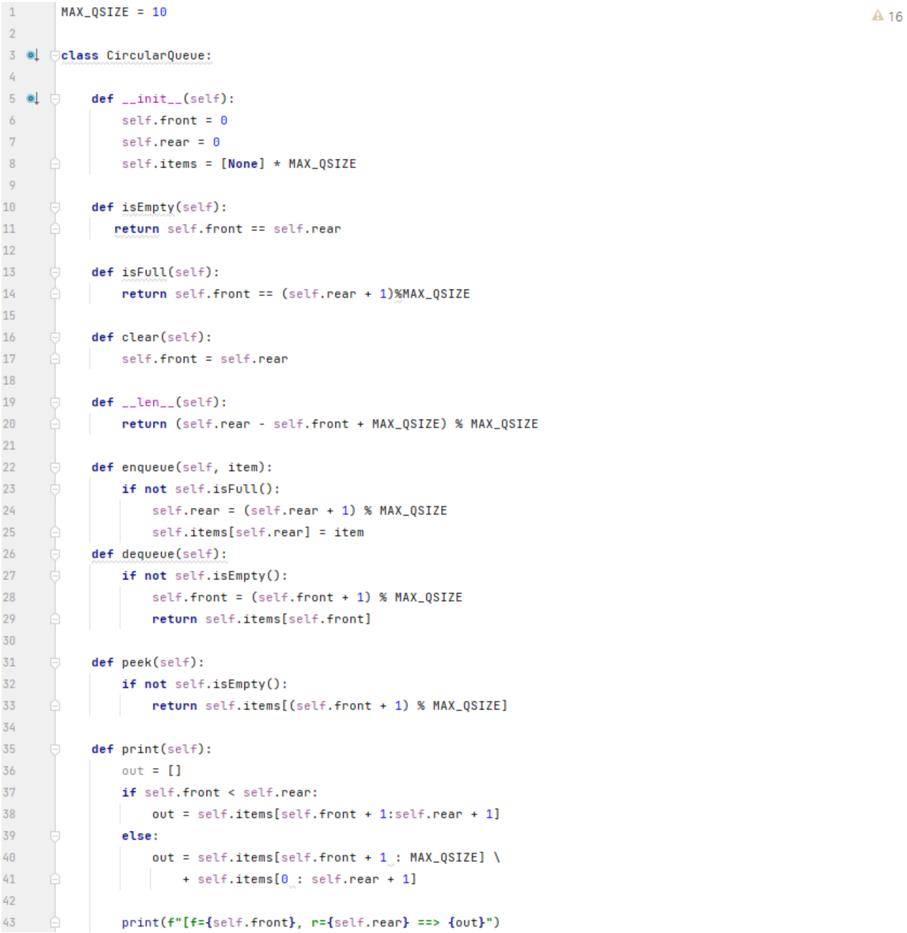
```
char maze[N][N] = {
{ ‘e', '1', '1', '1', '1', '1' },
{ ‘0', '0', '1', '0', '0', '1' },
{ '1', '0', '0', '0', '1', '1' },
{ '1', '0', '1', '0', '1', '1' },
{ '1', '0', '1', '0', '0’, ‘1' },
{ '1', '1', '1', '1', ‘0', ‘1' },
};
```

Write the following functions in Maze class

- ✓ DFS1() : It searches maze using stack from CircularDeque (front end operations)
- ✓ DFS2() : It searches maze using Stack
- ✓ DFS1() : It searches maze using stack from CircularDeque (rear end operation)
- ✓ BFS1() : It searches maze using Queue from CircularDeque
- ✓ BFS2() : It searches maze using Queue from CircularQueue

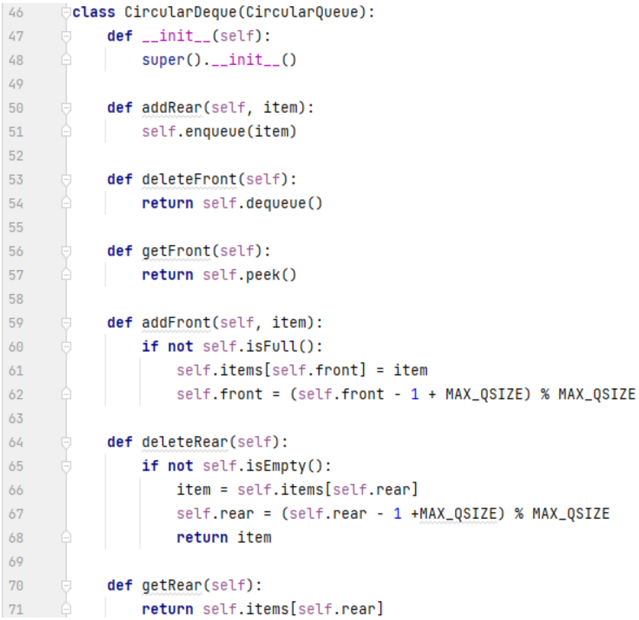


CircularQueue Code



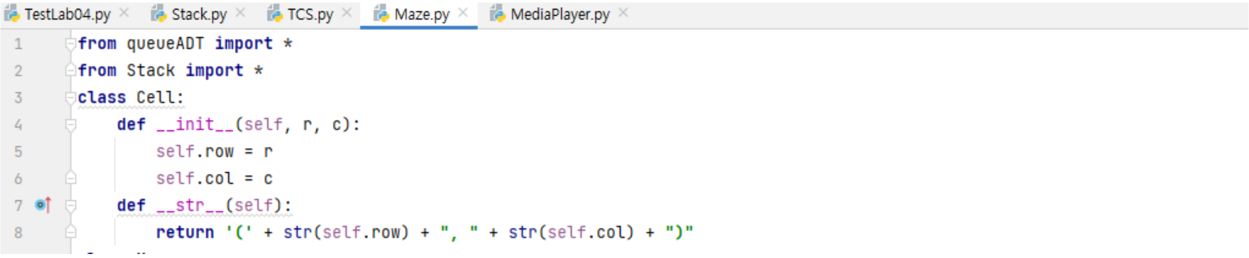
```python
MAX_QSIZE = 10

class CircularQueue:

    def __init__(self):
        self.front = 0
        self.rear = 0
        self.items = [None] * MAX_QSIZE

    def isEmpty(self):
       return self.front == self.rear

    def isFull(self):
        return self.front == (self.rear + 1)%MAX_QSIZE

    def clear(self):
        self.front = self.rear

    def __len__(self):
        return (self.rear - self.front + MAX_QSIZE) % MAX_QSIZE

    def enqueue(self, item):
        if not self.isFull():
            self.rear = (self.rear + 1) % MAX_QSIZE
            self.items[self.rear] = item
    def dequeue(self):
        if not self.isEmpty():
            self.front = (self.front + 1) % MAX_QSIZE
            return self.items[self.front]

    def peek(self):
        if not self.isEmpty():
            return self.items[(self.front + 1) % MAX_QSIZE]

    def print(self):
        out = []
        if self.front < self.rear:
            out = self.items[self.front + 1:self.rear + 1]
        else:
            out = self.items[self.front + 1 : MAX_QSIZE] \
                + self.items[0 : self.rear + 1]

        print(f"[f={self.front}, r={self.rear} ==> {out}")
```

CirculatrDeque Code



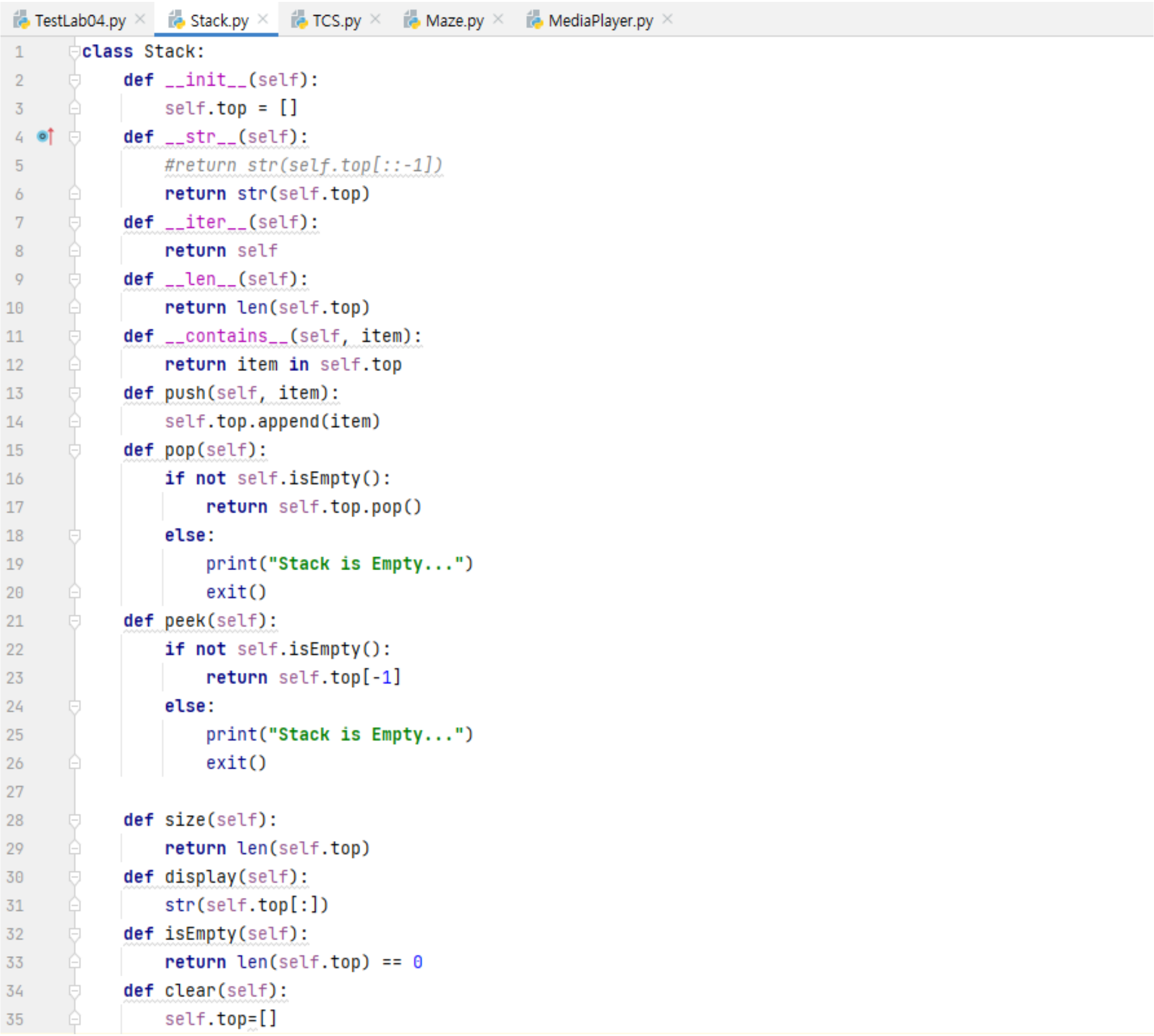
```python
class CircularDeque(CircularQueue):
    def __init__(self):
        super().__init__()

    def addRear(self, item):
        self.enqueue(item)

    def deleteFront(self):
        return self.dequeue()

    def getFront(self):
        return self.peek()

    def addFront(self, item):
        if not self.isFull():
            self.items[self.front] = item
            self.front = (self.front - 1 + MAX_QSIZE) % MAX_QSIZE

    def deleteRear(self):
        if not self.isEmpty():
            item = self.items[self.rear]
            self.rear = (self.rear - 1 +MAX_QSIZE) % MAX_QSIZE
            return item

    def getRear(self):
        return self.items[self.rear]
```

Cell Code



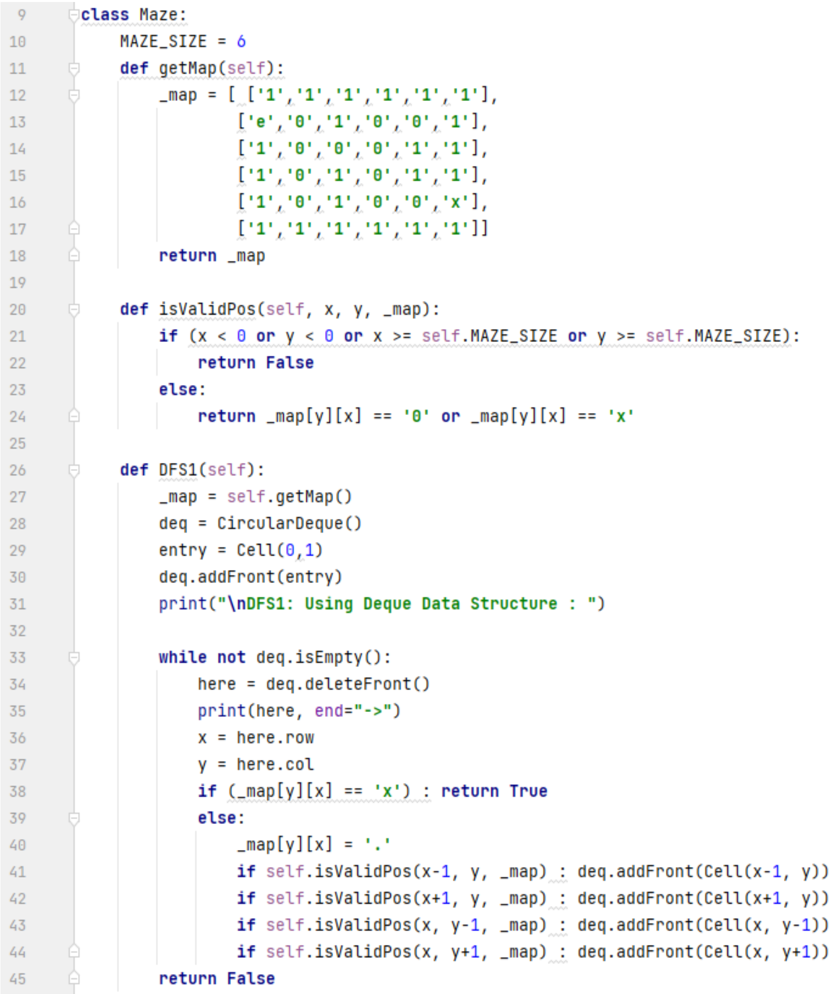
```python
from queueADT import *
from Stack import *
class Cell:
    def __init__(self, r, c):
        self.row = r
        self.col = c
    def __str__(self):
        return '(' + str(self.row) + ", " + str(self.col) + ")"
```

Stack Code

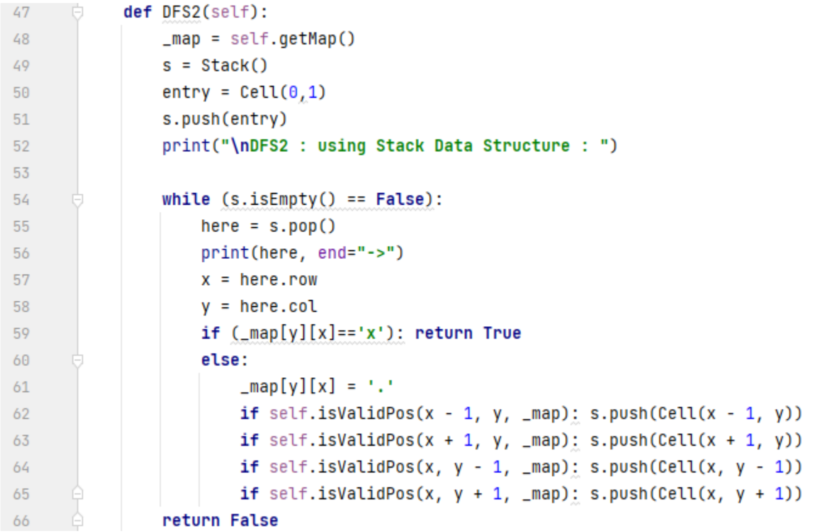


```python
class Stack:
    def __init__(self):
        self.top = []
    def __str__(self):
        #return str(self.top[::-1])
        return str(self.top)
    def __iter__(self):
        return self
    def __len__(self):
        return len(self.top)
    def __contains__(self, item):
        return item in self.top
    def push(self, item):
        self.top.append(item)
    def pop(self):
        if not self.isEmpty():
            return self.top.pop()
        else:
            print("Stack is Empty...")
            exit()
    def peek(self):
        if not self.isEmpty():
            return self.top[-1]
        else:
            print("Stack is Empty...")
            exit()

    def size(self):
        return len(self.top)
    def display(self):
        str(self.top[:])
    def isEmpty(self):
        return len(self.top) == 0
    def clear(self):
        self.top=[]
```

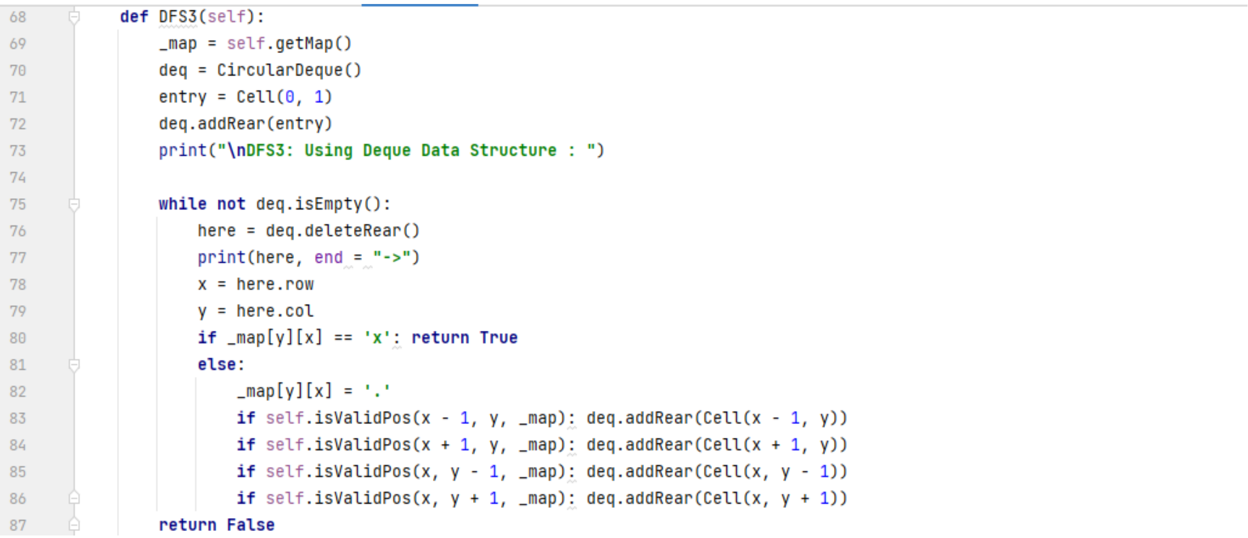
Maze Code



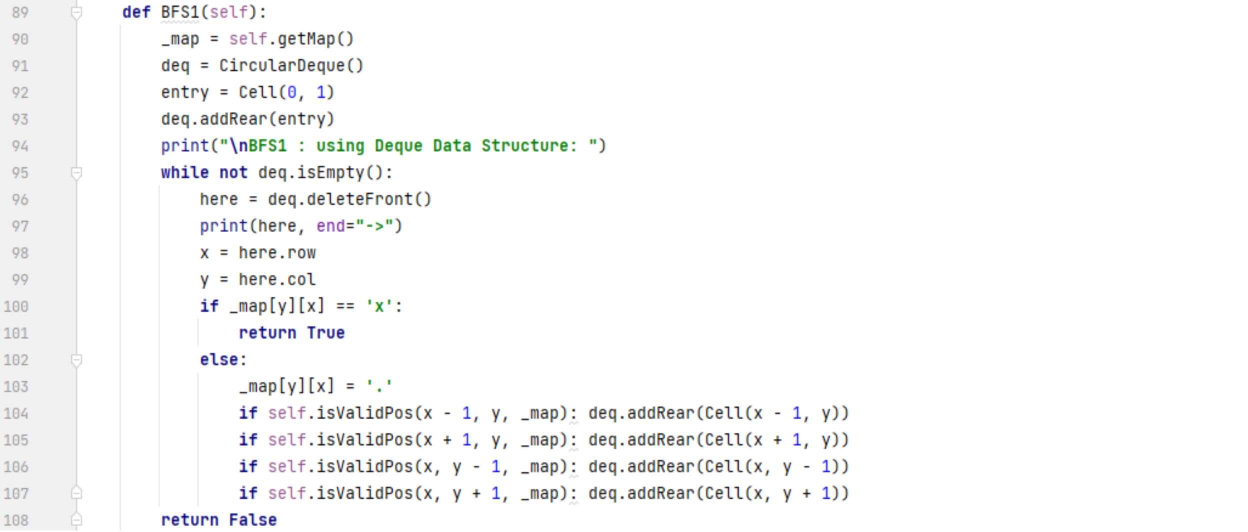
```
class Maze:
    MAZE_SIZE = 6
    def getMap(self):
        _map = [ ['1','1','1','1','1','1'],
                ['e','0','1','0','0','1'],
                ['1','0','0','0','1','1'],
                ['1','0','1','0','1','1'],
                ['1','0','1','0','0','x'],
                ['1','1','1','1','1','1']]
        return _map

    def isValidPos(self, x, y, _map):
        if (x < 0 or y < 0 or x >= self.MAZE_SIZE or y >= self.MAZE_SIZE):
            return False
        else:
            return _map[y][x] == '0' or _map[y][x] == 'x'

    def DFS1(self):
        _map = self.getMap()
        deq = CircularDeque()
        entry = Cell(0,1)
        deq.addFront(entry)
        print("\nDFS1: Using Deque Data Structure : ")

        while not deq.isEmpty():
            here = deq.deleteFront()
            print(here, end="->")
            x = here.row
            y = here.col
            if (_map[y][x] == 'x') : return True
            else:
                _map[y][x] = '.'
                if self.isValidPos(x-1, y, _map) : deq.addFront(Cell(x-1, y))
                if self.isValidPos(x+1, y, _map) : deq.addFront(Cell(x+1, y))
                if self.isValidPos(x, y-1, _map) : deq.addFront(Cell(x, y-1))
                if self.isValidPos(x, y+1, _map) : deq.addFront(Cell(x, y+1))
        return False
```



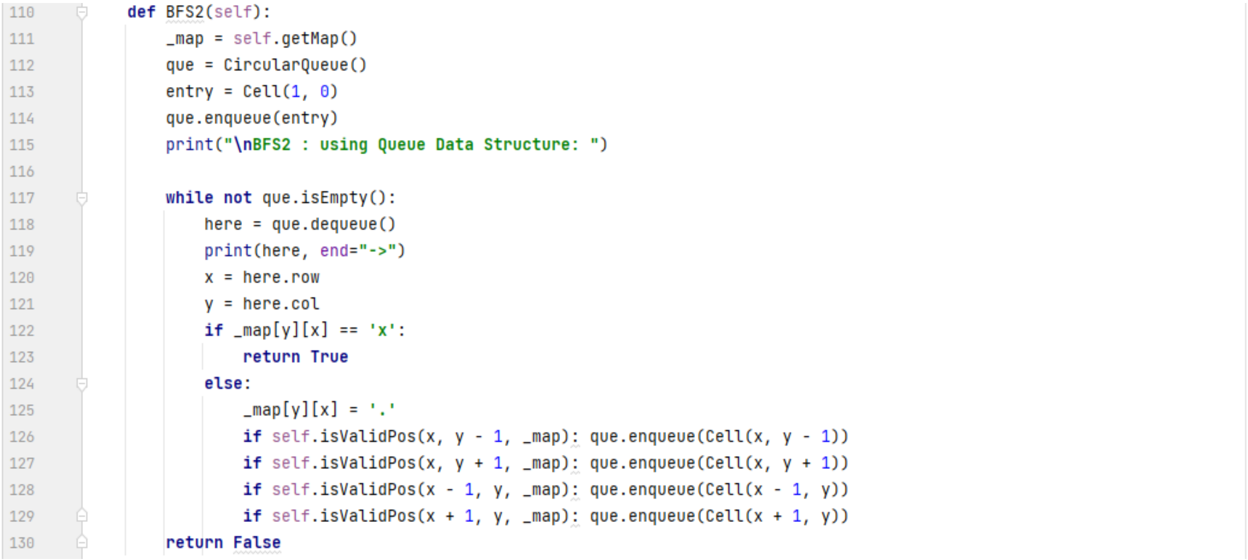
```
    def DFS2(self):
        _map = self.getMap()
        s = Stack()
        entry = Cell(0,1)
        s.push(entry)
        print("\nDFS2 : using Stack Data Structure : ")

        while (s.isEmpty() == False):
            here = s.pop()
            print(here, end="->")
            x = here.row
            y = here.col
            if (_map[y][x]=='x'): return True
            else:
                _map[y][x] = '.'
                if self.isValidPos(x - 1, y, _map): s.push(Cell(x - 1, y))
                if self.isValidPos(x + 1, y, _map): s.push(Cell(x + 1, y))
                if self.isValidPos(x, y - 1, _map): s.push(Cell(x, y - 1))
                if self.isValidPos(x, y + 1, _map): s.push(Cell(x, y + 1))
        return False
```



```python
def DFS3(self):
    _map = self.getMap()
    deq = CircularDeque()
    entry = Cell(0, 1)
    deq.addRear(entry)
    print("\nDFS3: Using Deque Data Structure : ")

    while not deq.isEmpty():
        here = deq.deleteRear()
        print(here, end = "->")
        x = here.row
        y = here.col
        if _map[y][x] == 'x': return True
        else:
            _map[y][x] = '.'
            if self.isValidPos(x - 1, y, _map): deq.addRear(Cell(x - 1, y))
            if self.isValidPos(x + 1, y, _map): deq.addRear(Cell(x + 1, y))
            if self.isValidPos(x, y - 1, _map): deq.addRear(Cell(x, y - 1))
            if self.isValidPos(x, y + 1, _map): deq.addRear(Cell(x, y + 1))
    return False
```

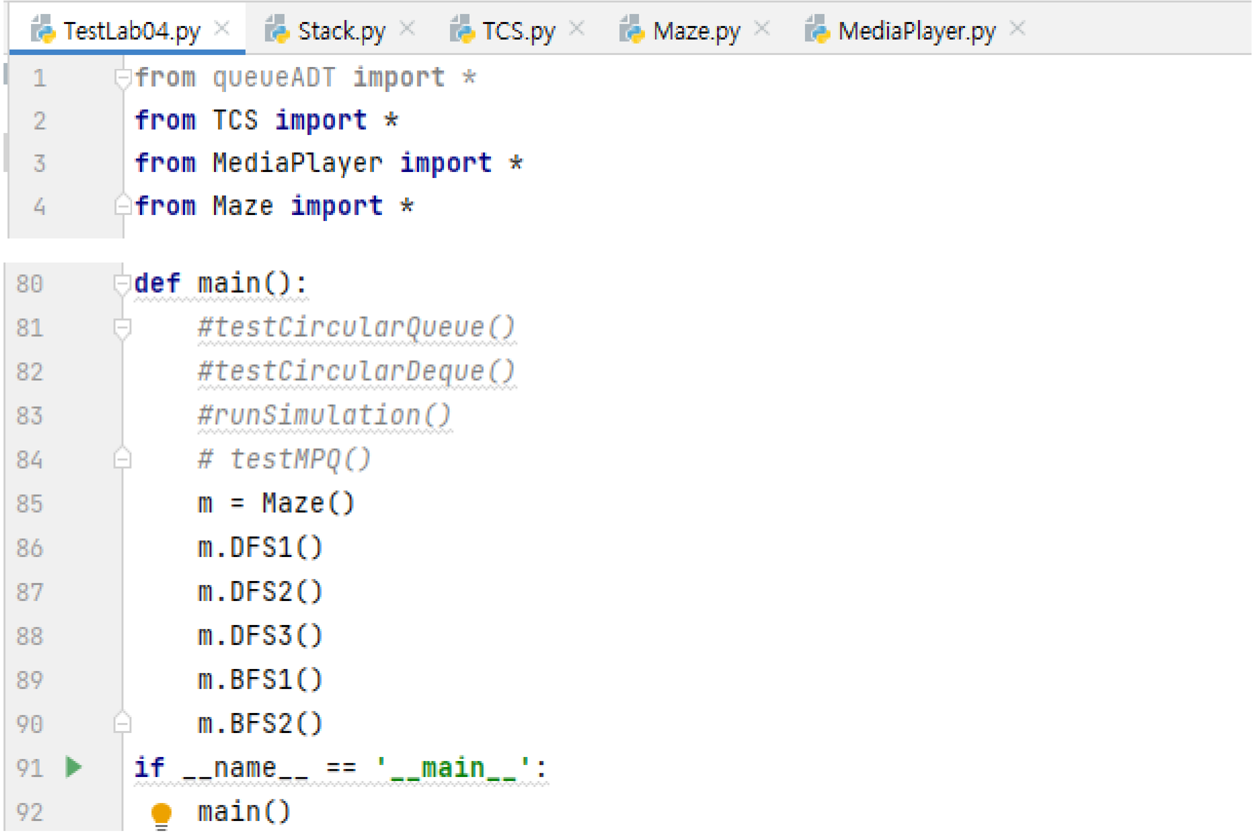


```python
def BFS1(self):
    _map = self.getMap()
    deq = CircularDeque()
    entry = Cell(0, 1)
    deq.addRear(entry)
    print("\nBFS1 : using Deque Data Structure: ")
    while not deq.isEmpty():
        here = deq.deleteFront()
        print(here, end="->")
        x = here.row
        y = here.col
        if _map[y][x] == 'x':
            return True
        else:
            _map[y][x] = '.'
            if self.isValidPos(x - 1, y, _map): deq.addRear(Cell(x - 1, y))
            if self.isValidPos(x + 1, y, _map): deq.addRear(Cell(x + 1, y))
            if self.isValidPos(x, y - 1, _map): deq.addRear(Cell(x, y - 1))
            if self.isValidPos(x, y + 1, _map): deq.addRear(Cell(x, y + 1))
    return False
```



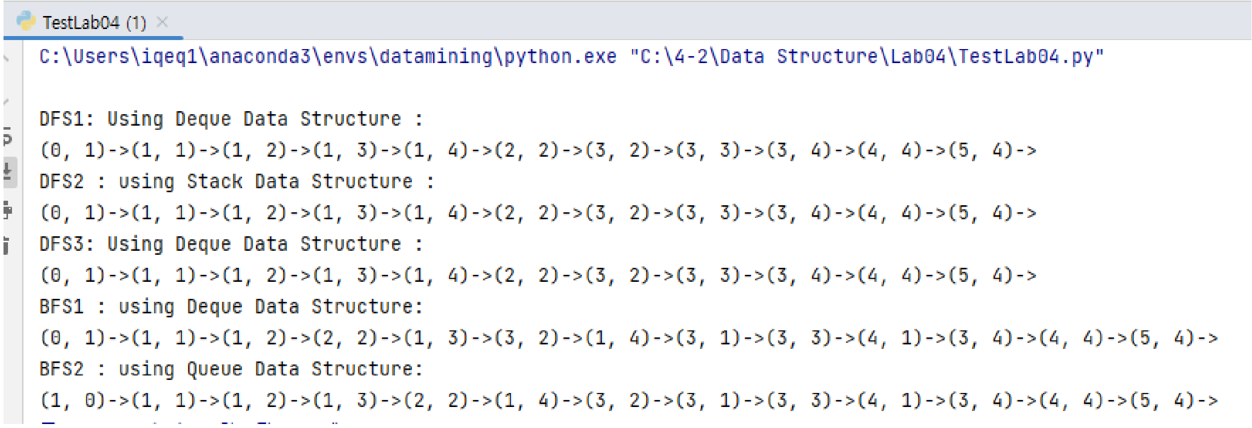
```python
def BFS2(self):
    _map = self.getMap()
    que = CircularQueue()
    entry = Cell(1, 0)
    que.enqueue(entry)
    print("\nBFS2 : using Queue Data Structure: ")

    while not que.isEmpty():
        here = que.dequeue()
        print(here, end="->")
        x = here.row
        y = here.col
        if _map[y][x] == 'x':
            return True
        else:
            _map[y][x] = '.'
            if self.isValidPos(x, y - 1, _map): que.enqueue(Cell(x, y - 1))
            if self.isValidPos(x, y + 1, _map): que.enqueue(Cell(x, y + 1))
            if self.isValidPos(x - 1, y, _map): que.enqueue(Cell(x - 1, y))
            if self.isValidPos(x + 1, y, _map): que.enqueue(Cell(x + 1, y))
    return False
```

Results/Output Code



```python
from queueADT import *
from TCS import *
from MediaPlayer import *
from Maze import *

def main():
    #testCircularQueue()
    #testCircularDeque()
    #runSimulation()
    # testMPQ()
    m = Maze()
    m.DFS1()
    m.DFS2()
    m.DFS3()
    m.BFS1()
    m.BFS2()
if __name__ == '__main__':
    main()
```

Results/Output



```
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab04\TestLab04.py"

DFS1: Using Deque Data Structure :
(0, 1)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(2, 2)->(3, 2)->(3, 3)->(3, 4)->(4, 4)->(5, 4)->
DFS2 : using Stack Data Structure :
(0, 1)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(2, 2)->(3, 2)->(3, 3)->(3, 4)->(4, 4)->(5, 4)->
DFS3: Using Deque Data Structure :
(0, 1)->(1, 1)->(1, 2)->(1, 3)->(1, 4)->(2, 2)->(3, 2)->(3, 3)->(3, 4)->(4, 4)->(5, 4)->
BFS1 : using Deque Data Structure:
(0, 1)->(1, 1)->(1, 2)->(2, 2)->(1, 3)->(3, 2)->(1, 4)->(3, 1)->(3, 3)->(4, 1)->(3, 4)->(4, 4)->(5, 4)->
BFS2 : using Queue Data Structure:
(1, 0)->(1, 1)->(1, 2)->(1, 3)->(2, 2)->(1, 4)->(3, 2)->(3, 1)->(3, 3)->(4, 1)->(3, 4)->(4, 4)->(5, 4)->
```

4. Conclusion

Conclude the Lab. Write your views about it, i.e. what have you learned from this lab? It was helpful or difficult etc

Through this task, I was able to learn the Queue and Deque data structure. He was also able to learn how to create a ticketing center system using him. And I was able to learn how to implement DFS and BFS by comprehensively using the data structures I have learned so far.
It was an opportunity to realize once again that Python is a really good tool in implementing the data structure.