REPORT MATA KULIAH
PROJECT REINFORCEMENT LEARNING

Disusun oleh:

Kelompok 7

1. Muhamad Iqbal (G1A021073)
2. Fajar Adhitia Suwandhi (G1A021086)
3. Irfan Luthfi (G1A021090)

Dosen Mata Kuliah:

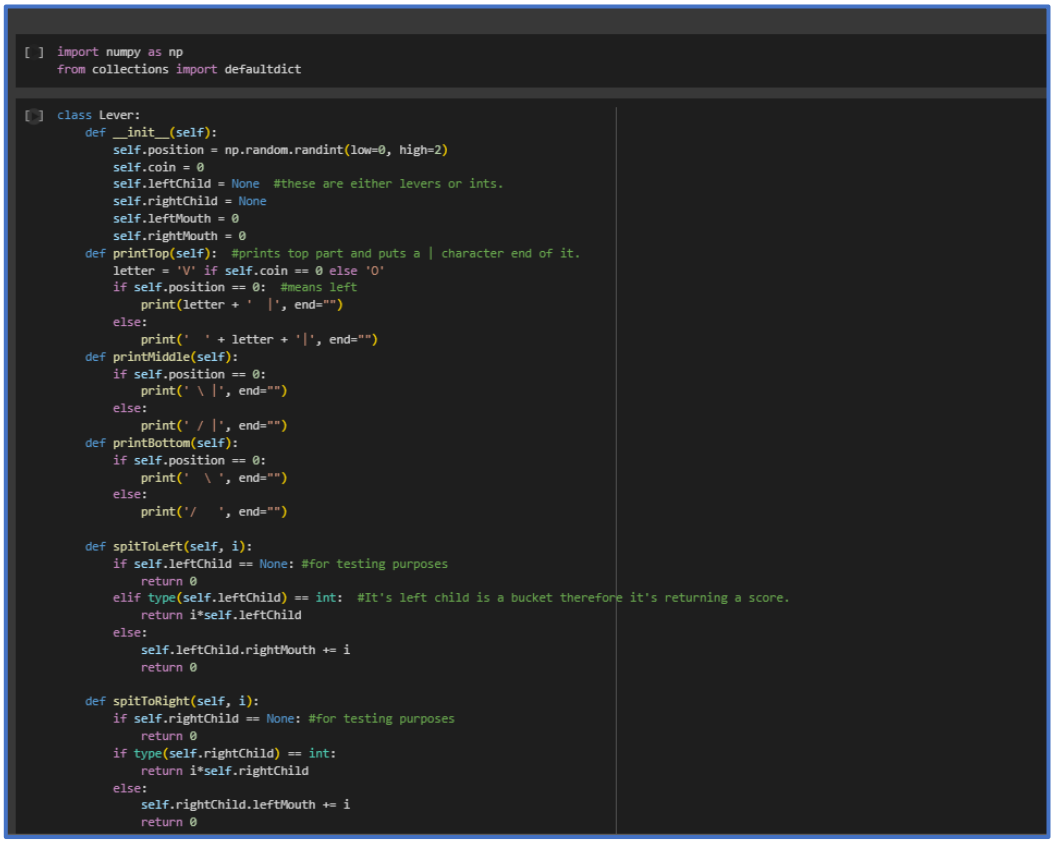
Arie Vatresia, S.T., M.T.I., Ph.D

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS BENGKULU
2024

1. IMPLEMENTASI DEEP Q-LEARNING “TREASURE DROP GAME”

Repository : https://github.com/MuhamadIqbal073/Reinforcement-Learning

Code :

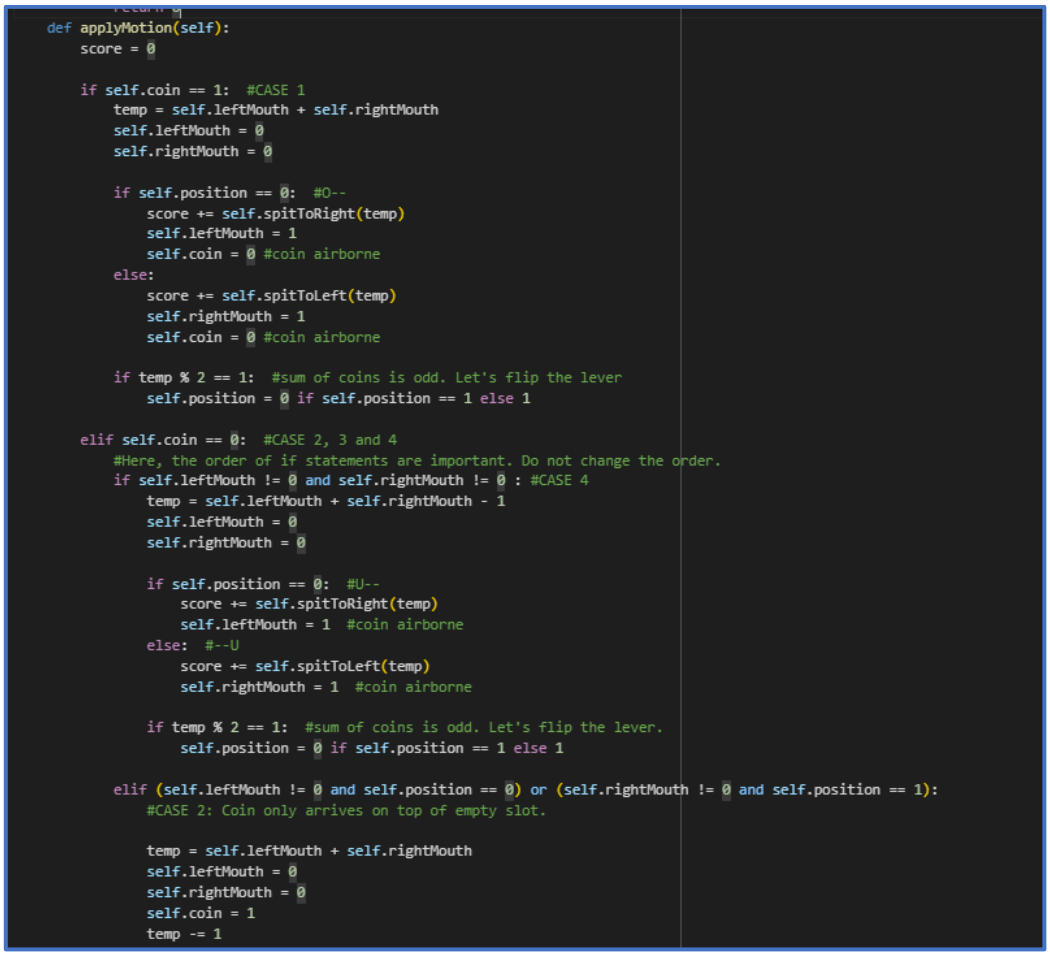


```
[ ] import numpy as np
    from collections import defaultdict
```

```
[ ] class Lever:
        def __init__(self):
            self.position = np.random.randint(low=0, high=2)
            self.coin = 0
            self.leftChild = None  #these are either levers or ints.
            self.rightChild = None
            self.leftMouth = 0
            self.rightMouth = 0
        def printTop(self):  #prints top part and puts a | character end of it.
            letter = 'Y' if self.coin == 0 else 'O'
            if self.position == 0:  #means left
                print(letter + '  |', end="")
            else:
                print('  ' + letter + '|', end="")
        def printMiddle(self):
            if self.position == 0:
                print(' \ |', end="")
            else:
                print(' / |', end="")
        def printBottom(self):
            if self.position == 0:
                print('  \ ', end="")
            else:
                print('/   ', end="")

        def spitToLeft(self, i):
            if self.leftChild == None: #for testing purposes
                return 0
            elif type(self.leftChild) == int:  #It's left child is a bucket therefore it's returning a score.
                return i*self.leftChild
            else:
                self.leftChild.rightMouth += i
                return 0

        def spitToRight(self, i):
            if self.rightChild == None: #for testing purposes
                return 0
            if type(self.rightChild) == int:
                return i*self.rightChild
            else:
                self.rightChild.leftMouth += i
                return 0
```

Penjelasan :

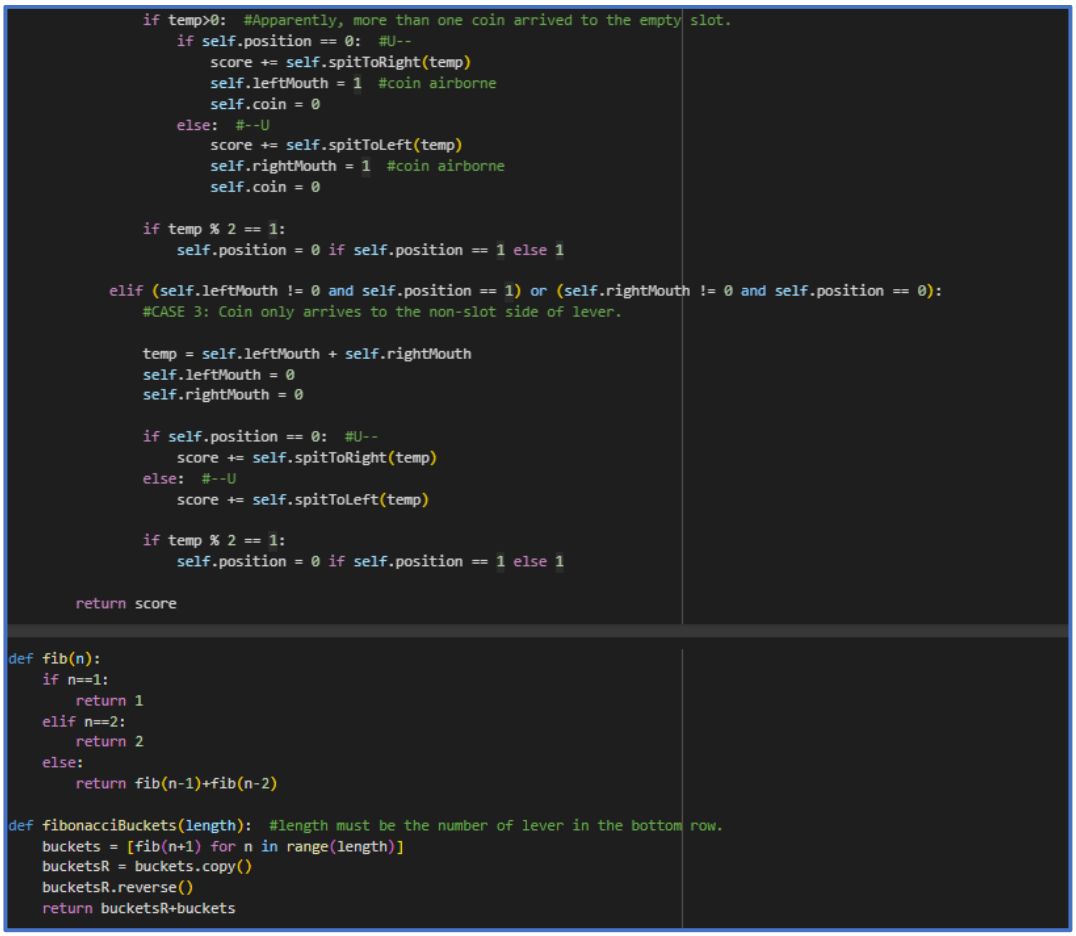
Setiap objek Lever memiliki atribut seperti position, coin, leftChild, rightChild, leftMouth, dan rightMouth. position menentukan arah tuas (0 untuk kiri dan 1 untuk kanan), sementara coin menunjukkan nilai 'Y' atau 'O' untuk merepresentasikan kondisi tertentu. Metode printTop, printMiddle, dan printBottom mencetak representasi visual dari tuas dengan tanda ‘|’ untuk menggambarkan posisi. Metode spitToLeft dan spitToRight digunakan untuk mengatur pergerakan nilai atau objek ke anak tuas di kiri atau kanan, serta memperbarui nilai leftMouth atau rightMouth. Struktur ini memungkinkan pembuatan sistem tuas yang kompleks dengan pergerakan antara objek-objek Lever.



```
    def applyMotion(self):
        score = 0

        if self.coin == 1:  #CASE 1
            temp = self.leftMouth + self.rightMouth
            self.leftMouth = 0
            self.rightMouth = 0

            if self.position == 0:  #O--
                score += self.spitToRight(temp)
                self.leftMouth = 1
                self.coin = 0 #coin airborne
            else:
                score += self.spitToLeft(temp)
                self.rightMouth = 1
                self.coin = 0 #coin airborne

            if temp % 2 == 1:  #sum of coins is odd. Let's flip the lever
                self.position = 0 if self.position == 1 else 1

        elif self.coin == 0:  #CASE 2, 3 and 4
            #Here, the order of if statements are important. Do not change the order.
            if self.leftMouth != 0 and self.rightMouth != 0 : #CASE 4
                temp = self.leftMouth + self.rightMouth - 1
                self.leftMouth = 0
                self.rightMouth = 0

                if self.position == 0:  #U--
                    score += self.spitToRight(temp)
                    self.leftMouth = 1  #coin airborne
                else:  #--U
                    score += self.spitToLeft(temp)
                    self.rightMouth = 1  #coin airborne

                if temp % 2 == 1:  #sum of coins is odd. Let's flip the lever.
                    self.position = 0 if self.position == 1 else 1

            elif (self.leftMouth != 0 and self.position == 0) or (self.rightMouth != 0 and self.position == 1):
                #CASE 2: Coin only arrives on top of empty slot.

                temp = self.leftMouth + self.rightMouth
                self.leftMouth = 0
                self.rightMouth = 0
                self.coin = 1
                temp -= 1
```

Penjelasan :

Gambar ini menunjukkan metode applyMotion dari kelas Lever yang berfungsi untuk mengatur logika gerakan tuas berdasarkan kondisi saat ini. Metode ini menggunakan variabel score untuk menyimpan nilai yang diperoleh dari operasi pergerakan. Terdapat beberapa kondisi utama yang mengatur perilaku tuas: Jika coin bernilai 1, maka tuas akan mengirimkan nilai yang disimpan ke arah kanan atau kiri tergantung pada posisi awalnya dan memperbarui status leftMouth atau rightMouth. Metode ini juga mengubah position tuas jika jumlah total nilai yang dihasilkan (temp) bernilai ganjil, yang akan menyebabkan tuas berganti arah (dari kiri ke kanan atau sebaliknya). Kondisi lain mencakup situasi di mana leftMouth dan rightMouth tidak kosong, serta kondisi di mana coin dapat masuk hanya pada posisi yang kosong, sehingga sistem memperbarui status tuas dan nilai coin yang ditangkap.

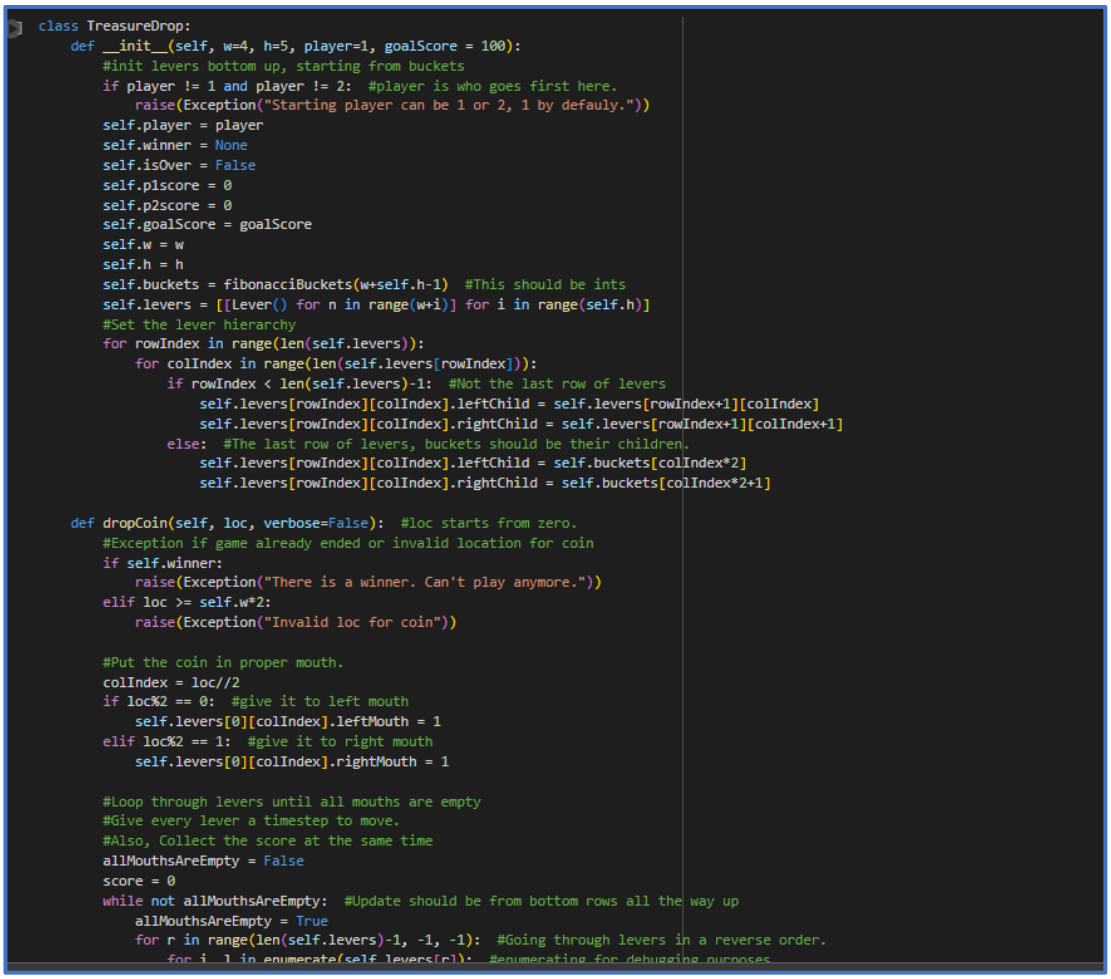


```
            if temp>0:  #Apparently, more than one coin arrived to the empty slot.
                if self.position == 0:  #U--
                    score += self.spitToRight(temp)
                    self.leftMouth = 1  #coin airborne
                    self.coin = 0
                else:  #--U
                    score += self.spitToLeft(temp)
                    self.rightMouth = 1  #coin airborne
                    self.coin = 0

            if temp % 2 == 1:
                self.position = 0 if self.position == 1 else 1

        elif (self.leftMouth != 0 and self.position == 1) or (self.rightMouth != 0 and self.position == 0):
            #CASE 3: Coin only arrives to the non-slot side of lever.

            temp = self.leftMouth + self.rightMouth
            self.leftMouth = 0
            self.rightMouth = 0

            if self.position == 0:  #U--
                score += self.spitToRight(temp)
            else:  #--U
                score += self.spitToLeft(temp)

            if temp % 2 == 1:
                self.position = 0 if self.position == 1 else 1

    return score
```

```
def fib(n):
    if n==1:
        return 1
    elif n==2:
        return 2
    else:
        return fib(n-1)+fib(n-2)

def fibonacciBuckets(length):  #length must be the number of lever in the bottom row.
    buckets = [fib(n+1) for n in range(length)]
    bucketsR = buckets.copy()
    bucketsR.reverse()
    return bucketsR+buckets
```

Penjelasan :

Kode ini melanjutkan logika applyMotion dari kelas Lever, dengan lebih banyak kondisi untuk menangani situasi di mana beberapa koin tiba di slot kosong. Metode ini memastikan bahwa jika lebih dari satu koin datang, tuas akan mengatur arah gerakan berdasarkan posisi saat ini (spitToRight atau spitToLeft). Setelah koin dipindahkan, posisi tuas dapat diubah jika jumlah total koin yang dihitung (temp) adalah ganjil, mirip dengan bagian sebelumnya. Kode ini juga berisi fungsi fib(n) yang menghitung bilangan Fibonacci secara rekursif, dan fibonacciBuckets(length) yang membuat daftar jumlah koin berdasarkan urutan Fibonacci, menggabungkan urutan asli dan versi terbalik. Fungsi ini tampaknya digunakan untuk menentukan distribusi awal koin dalam sistem tuas yang lebih besar.



```python
class TreasureDrop:
    def __init__(self, w=4, h=5, player=1, goalScore = 100):
        #init levers bottom up, starting from buckets
        if player != 1 and player != 2:  #player is who goes first here.
            raise(Exception("Starting player can be 1 or 2, 1 by defauly."))
        self.player = player
        self.winner = None
        self.isOver = False
        self.p1score = 0
        self.p2score = 0
        self.goalScore = goalScore
        self.w = w
        self.h = h
        self.buckets = fibonacciBuckets(w+self.h-1)  #This should be ints
        self.levers = [[Lever() for n in range(w+i)] for i in range(self.h)]
        #Set the lever hierarchy
        for rowIndex in range(len(self.levers)):
            for colIndex in range(len(self.levers[rowIndex])):
                if rowIndex < len(self.levers)-1:  #Not the last row of levers
                    self.levers[rowIndex][colIndex].leftChild = self.levers[rowIndex+1][colIndex]
                    self.levers[rowIndex][colIndex].rightChild = self.levers[rowIndex+1][colIndex+1]
                else:  #The last row of levers, buckets should be their children.
                    self.levers[rowIndex][colIndex].leftChild = self.buckets[colIndex*2]
                    self.levers[rowIndex][colIndex].rightChild = self.buckets[colIndex*2+1]

    def dropCoin(self, loc, verbose=False):  #loc starts from zero.
        #Exception if game already ended or invalid location for coin
        if self.winner:
            raise(Exception("There is a winner. Can't play anymore."))
        elif loc >= self.w*2:
            raise(Exception("Invalid loc for coin"))

        #Put the coin in proper mouth.
        colIndex = loc//2
        if loc%2 == 0:  #give it to left mouth
            self.levers[0][colIndex].leftMouth = 1
        elif loc%2 == 1:  #give it to right mouth
            self.levers[0][colIndex].rightMouth = 1

        #Loop through levers until all mouths are empty
        #Give every lever a timestep to move.
        #Also, Collect the score at the same time
        allMouthsAreEmpty = False
        score = 0
        while not allMouthsAreEmpty:  #Update should be from bottom rows all the way up
            allMouthsAreEmpty = True
            for r in range(len(self.levers)-1, -1, -1):  #Going through levers in a reverse order.
                for i, l in enumerate(self.levers[r]):  #enumerating for debugging purposes
```

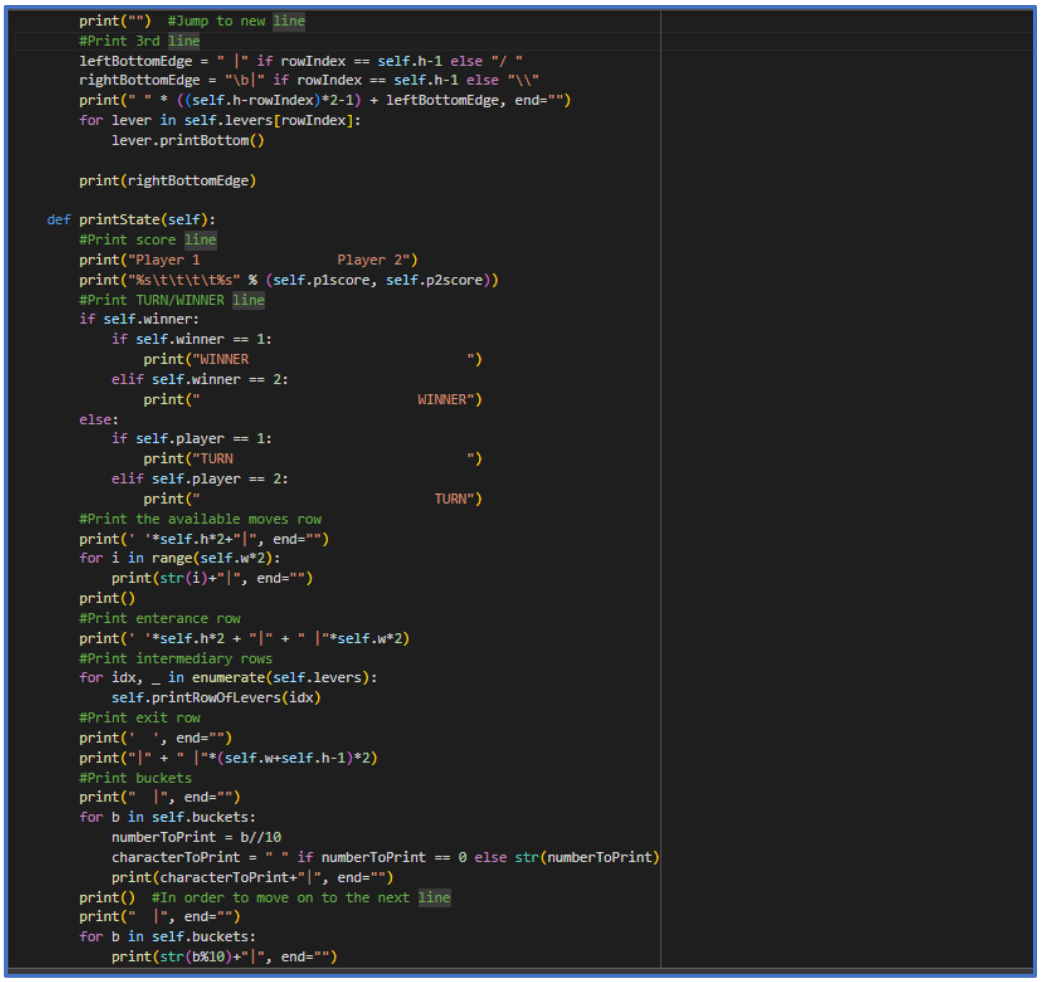
Penjelasan :

Ini mendefinisikan kelas TreasureDrop yang mensimulasikan permainan di mana koin dijatuhkan melalui serangkaian tuas dengan tujuan mencapai skor tertentu. Konstruktornya (__init__) menginisialisasi permainan dengan beberapa parameter, seperti ukuran permainan (w untuk lebar dan h untuk tinggi), pemain pertama yang bergerak (player), dan skor tujuan (goalScore). Metode ini membuat struktur tuas dalam bentuk hierarki menggunakan objek Lever, yang diatur dalam beberapa baris dan kolom. Fungsi dropCoin digunakan untuk menjatuhkan koin pada posisi tertentu, mengubah status tuas berdasarkan gerakan koin ke kiri atau kanan. Setelah koin ditempatkan, kode ini memastikan bahwa semua tuas diperiksa dari bawah ke atas untuk memperbarui status hingga tidak ada lagi pergerakan yang tersisa (allMouthsEmpty). Hal ini dilakukan dengan iterasi terbalik untuk memastikan setiap perubahan pada posisi tuas dipertimbangkan dalam urutan yang benar, memungkinkan simulasi pergerakan koin yang akurat dalam permainan ini.

```
            allMouthsAreEmpty = True
            for r in range(len(self.levers)-1, -1, -1):  #Going through levers in a reverse order.
                for i, l in enumerate(self.levers[r]):  #enumerating for debugging purposes.
                    if l.leftMouth != 0 or l.rightMouth != 0:
                        allMouthsAreEmpty = False
                        score += l.applyMotion()

        #Give score to player
        if self.player == 1:
            self.p1score += score
        elif self.player == 2:
            self.p2score += score

        #It's other player's turn
        if self.player == 1:
            self.player = 2
        elif self.player == 2:
            self.player = 1

        #Check if the game ends
        if self.p1score >= self.goalScore:
            self.player = None
            self.winner = 1
            self.isOver = True
        elif self.p2score >= self.goalScore:
            self.player = None
            self.winner = 2
            self.isOver = True

        #If verbose, print the game state. OR, do this job somewhere else. I'm not sure.
        if verbose:
            self.printState()

        return score


    def printRowOfLevers(self, rowIndex):  #Helper function of printState
        #Print 1st line
        print(" " * (self.h-rowIndex)*2 + '|', end="")
        for lever in self.levers[rowIndex]:
            lever.printTop()
        print("")  #Jump to new line
        #Print 2nd line
        print(" " * (self.h-rowIndex)*2 + '|', end="")
        for lever in self.levers[rowIndex]:
            lever.printMiddle()
        print("")  #Jump to new line
        #Print 3rd line
```

Penjelasan :

Ini melanjutkan logika permainan dalam kelas TreasureDrop dengan menerapkan mekanisme pemberian skor dan pergantian giliran pemain. Setelah memproses pergerakan koin di setiap tuas menggunakan applyMotion, permainan mengecek apakah ada perubahan posisi tuas dengan memeriksa apakah ada mulut (mouth) yang tidak kosong (allMouthsAreEmpty). Jika tidak, pergerakan koin dilanjutkan. Skor yang dihasilkan kemudian diberikan kepada pemain yang sesuai (p1Score atau p2Score), dan giliran pemain berganti setelah setiap langkah. Jika salah satu pemain mencapai skor target (goalScore), permainan berakhir dengan menetapkan pemenang (isOver menjadi True dan self.winner menunjuk ke pemain yang menang). Jika mode verbose diaktifkan, status permainan akan dicetak menggunakan metode printState, yang mencetak status tuas di setiap baris. Fungsi printRowOfLevers membantu menampilkan representasi visual dari setiap baris tuas dengan mengakses dan mencetak setiap bagian dari tuas tersebut (top, middle, dan bottom).



```
        print("")  #Jump to new line
        #Print 3rd line
        leftBottomEdge = " |" if rowIndex == self.h-1 else "/ "
        rightBottomEdge = "\b|" if rowIndex == self.h-1 else "\\"
        print(" " * ((self.h-rowIndex)*2-1) + leftBottomEdge, end="")
        for lever in self.levers[rowIndex]:
            lever.printBottom()

        print(rightBottomEdge)

    def printState(self):
        #Print score line
        print("Player 1                Player 2")
        print("%s\t\t\t\t\t%s" % (self.p1score, self.p2score))
        #Print TURN/WINNER line
        if self.winner:
            if self.winner == 1:
                print("WINNER                          ")
            elif self.winner == 2:
                print("                          WINNER")
        else:
            if self.player == 1:
                print("TURN                            ")
            elif self.player == 2:
                print("                           TURN")
        #Print the available moves row
        print(' '*self.h*2+"|", end="")
        for i in range(self.w*2):
            print(str(i)+"|", end="")
        print()
        #Print enterance row
        print(' '*self.h*2 + "|" + " |"*self.w*2)
        #Print intermediary rows
        for idx, _ in enumerate(self.levers):
            self.printRowOfLevers(idx)
        #Print exit row
        print('  ', end="")
        print("|" + " |"*(self.w+self.h-1)*2)
        #Print buckets
        print("  |", end="")
        for b in self.buckets:
            numberToPrint = b//10
            characterToPrint = " " if numberToPrint == 0 else str(numberToPrint)
            print(characterToPrint+"|", end="")
        print()  #In order to move on to the next line
        print("  |", end="")
        for b in self.buckets:
            print(str(b%10)+"|", end="")
```

Penjelasan :

Ini melanjutkan implementasi dari metode printState di kelas TreasureDrop, yang digunakan untuk mencetak status permainan secara visual dan informasi lainnya, seperti skor pemain, giliran, dan posisi tuas. Kode ini mencetak garis-garis yang memisahkan bagian visual, termasuk baris skor pemain, menampilkan skor p1Score dan p2Score, serta status pemenang atau pemain yang sedang bermain. Bagian dari kode ini juga menampilkan gerakan yang tersedia bagi pemain, menunjukkan baris yang berisi nomor-nomor kolom tempat koin bisa dijatuhkan. Selanjutnya, kode ini memanggil printRowOfLevers untuk mencetak representasi setiap baris tuas, serta menampilkan status ember atau bucket di bagian bawah. Penataan dan pengaturan format output diatur sedemikian rupa agar status permainan mudah dibaca oleh pengguna, dengan menggunakan loop dan kondisi untuk mengatur tampilan baris demi baris, termasuk simbol-simbol khusus untuk menunjukkan batas atau tepi visual dari tuas dan ember.

```
        for b in self.buckets:
            print(str(b%10)+"|", end="")
        print()
        print()

    def getState(self):  #state returns as a string
        state = []
        for r in range(len(self.levers)):
            for l in self.levers[r]:
                state.append(l.position)
                state.append(l.coin)
        return str(state), self.w, self.h

    def setState(self, state, w, h):  #adjust the states of levers according to the given state
        if self.w != w or self.h != h:
            print(self.w)
            print(w)
            print(self.h)
            print(h)
            raise(Exception("width and height not compatible for setState()."))

        s = list(state)
        for r in range(len(self.levers)):
            for l in self.levers[r]:
                l.position = s.pop(0)
                l.coin = s.pop(0)

    def nextStatesAndRewards(self):  #Returns next states as strings and associated rewards.
        currentState = self.getState()  #preserving the current state
        nextStatesAndRewards = []
        #because nextState[1] is width and nextState[2] is height
        for i in range(self.w*2):
            td = TreasureDrop(w=self.w, h=self.h)
            td.setState(currentState[0], currentState[1], currentState[2])
            reward = td.dropCoin(i, verbose=False)
            nextState = td.getState()[0]
            nextStatesAndRewards.append((nextState, reward))
        return nextStatesAndRewards

    def playable(self):
        return not self.isOver
```

Penjelasan :

Ini melanjutkan implementasi dari kelas TreasureDrop, dengan menambahkan metode-metode yang berkaitan dengan pengambilan dan pengaturan status permainan. Metode getState menghasilkan representasi status permainan saat ini dalam bentuk string, dengan menyimpan posisi dan status koin dari setiap tuas. Metode setState memungkinkan untuk mengatur ulang status tuas berdasarkan input tertentu, memastikan bahwa lebar (w) dan tinggi (h) dari status yang diberikan sesuai dengan pengaturan permainan. Jika tidak cocok, akan memunculkan exception. Metode nextStatesAndRewards mengembalikan daftar status berikutnya yang mungkin beserta reward (skor) yang diperoleh dari menjatuhkan koin pada posisi yang berbeda. Metode ini menciptakan instansi baru dari TreasureDrop untuk setiap kemungkinan gerakan, menjalankan simulasi gerakan menggunakan dropCoin, dan menyimpan hasil status berikutnya beserta rewardnya. Metode playable memeriksa apakah permainan masih berlangsung (isOver bernilai False). Dengan metode ini, permainan dapat dijalankan dalam simulasi untuk mengeksplorasi berbagai kemungkinan hasil berdasarkan tindakan pemain.

```
td = TreasureDrop(w=4, h=5)

state = None
# Added variable definition of w
w = 4
while not td.isOver:
    reply = input("Make a move: ")
    if reply == "q":
        break
    move = int(reply)
    td.dropCoin(move, verbose=True)
    td.dropCoin(np.random.randint(low=0, high=2*w-1), verbose=True)  #The random Agent
```

Penjelasan :

Kode diatas menunjukkan bagian interaksi pengguna untuk permainan TreasureDrop. Sebuah objek permainan td dibuat dengan lebar (w=4) dan tinggi (h=5). Permainan berjalan dalam loop while selama td.isOver bernilai False, yang berarti permainan belum berakhir. Di dalam loop, pemain diminta untuk memasukkan gerakan melalui input (reply), dan jika pemain memasukkan "q", permainan akan berhenti. Input pemain kemudian diubah menjadi bilangan bulat dan digunakan sebagai gerakan (move) yang kemudian diteruskan ke metode dropCoin dari objek td untuk menjatuhkan koin pada posisi yang dipilih. Setelah itu, simulasi agen acak menjatuhkan koin pada posisi acak menggunakan fungsi np.random.randint untuk memilih gerakan berikutnya. Kedua aksi (pemain dan agen) memiliki parameter verbose=True untuk mencetak status permainan setelah setiap gerakan, sehingga pemain dapat melihat perkembangan permainan secara visual.

Output :

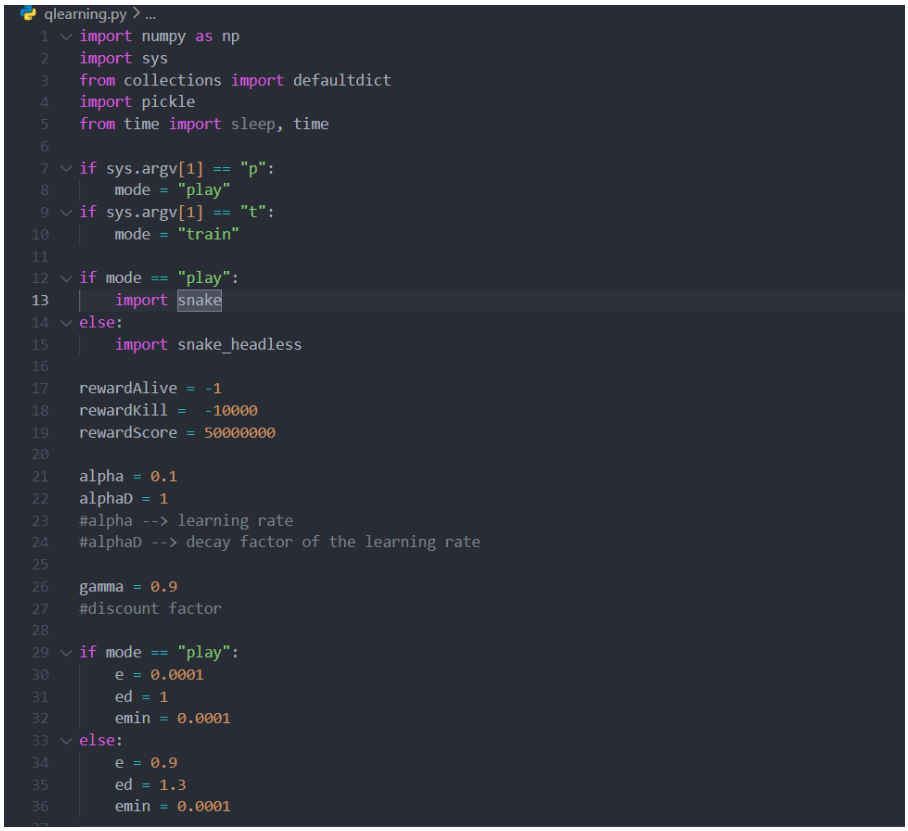
Penjelasan:

Output ini menunjukkan hasil dari permainan "Treasure Drop" antara dua pemain. Di bagian atas, terlihat skor masing-masing pemain: "Player 1" memiliki 38 poin, sedangkan "Player 2" memiliki 100 poin. Teks "WINNER" di sebelah "Player 2" menunjukkan bahwa pemain kedua memenangkan permainan, karena telah mencapai atau melebihi batas 100 poin yang ditentukan sebagai syarat kemenangan.

2. IMPLEMENTASI Q-LEARNING “SNAKE GAME”

Repository : https://github.com/MuhamadIqbal073/Kelomok-7-Q-Learning-SnakeGame

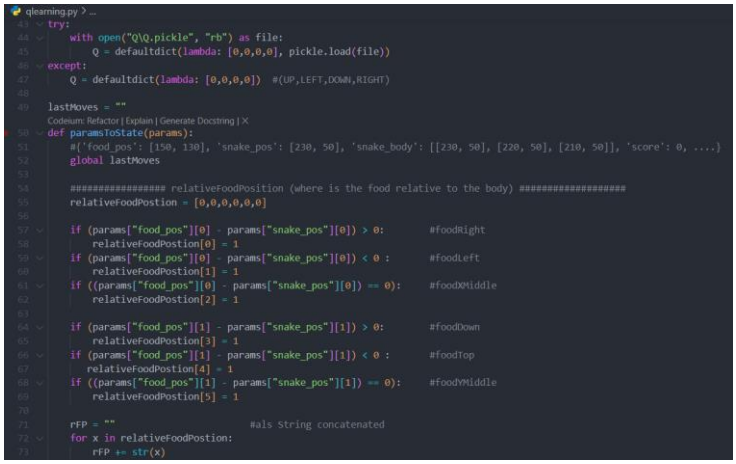
Code ;



```python
import numpy as np
import sys
from collections import defaultdict
import pickle
from time import sleep, time

if sys.argv[1] == "p":
    mode = "play"
if sys.argv[1] == "t":
    mode = "train"

if mode == "play":
    import snake
else:
    import snake_headless

rewardAlive = -1
rewardKill =  -10000
rewardScore = 50000000

alpha = 0.1
alphaD = 1
#alpha --> learning rate
#alphaD --> decay factor of the learning rate

gamma = 0.9
#discount factor

if mode == "play":
    e = 0.0001
    ed = 1
    emin = 0.0001
else:
    e = 0.9
    ed = 1.3
    emin = 0.0001
```

Penjelasan :

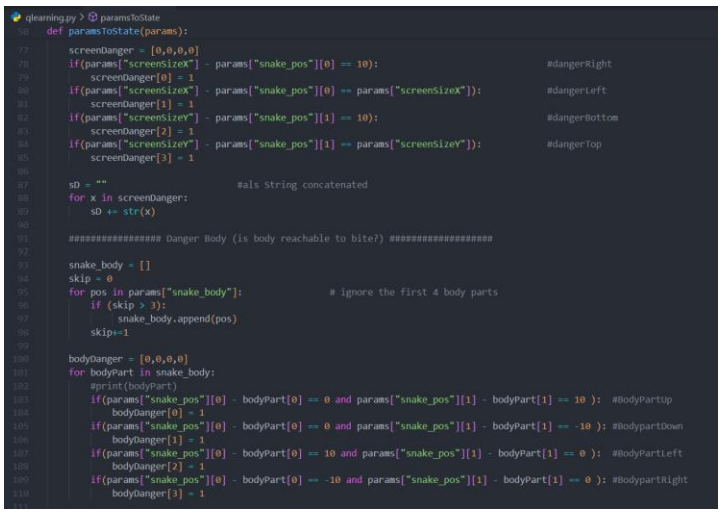
Kode pada gambar adalah bagian awal dari implementasi Q-learning untuk game Snake, dengan dua mode: play dan train, yang diatur berdasarkan input pengguna (p untuk play atau t untuk train). Dalam mode play, program mengimpor modul snake, sedangkan mode train menggunakan snake_headless, memungkinkan pelatihan tanpa visualisasi. Beberapa parameter pembelajaran seperti rewardAlive, rewardKill, dan rewardScore digunakan untuk mengatur reward dan penalti, membantu agen belajar dari aksinya. Parameter alpha (learning rate) dan gamma (discount factor) mengatur pembaruan nilai Q, sementara e (epsilon) mengontrol keseimbangan antara eksplorasi dan eksploitasi, dengan nilai yang lebih tinggi saat pelatihan untuk meningkatkan eksplorasi.



```
qlearning.py > ...
try:
    with open("Q\Q.pickle", "rb") as file:
        Q = defaultdict(lambda: [0,0,0,0], pickle.load(file))
except:
    Q = defaultdict(lambda: [0,0,0,0])  #(UP,LEFT,DOWN,RIGHT)

lastMoves = ""
Codeium: Refactor | Explain | Generate Docstring | ×
def paramsToState(params):
    #{'food_pos': [150, 130], 'snake_pos': [230, 50], 'snake_body': [[230, 50], [220, 50], [210, 50]], 'score': 0, ....}
    global lastMoves

    ################ relativeFoodPosition (where is the food relative to the body) ##################
    relativeFoodPostion = [0,0,0,0,0,0]

    if (params["food_pos"][0] - params["snake_pos"][0]) > 0:        #foodRight
        relativeFoodPostion[0] = 1
    if (params["food_pos"][0] - params["snake_pos"][0]) < 0 :       #foodLeft
        relativeFoodPostion[1] = 1
    if ((params["food_pos"][0] - params["snake_pos"][0]) == 0):     #foodXMiddle
        relativeFoodPostion[2] = 1

    if (params["food_pos"][1] - params["snake_pos"][1]) > 0:        #foodDown
        relativeFoodPostion[3] = 1
    if (params["food_pos"][1] - params["snake_pos"][1]) < 0 :       #foodTop
        relativeFoodPostion[4] = 1
    if ((params["food_pos"][1] - params["snake_pos"][1]) == 0):     #foodYMiddle
        relativeFoodPostion[5] = 1

    rFP = ""                        #als String concatenated
    for x in relativeFoodPostion:
        rFP += str(x)
```

Penjelasan :

Ini memuat implementasi bagian dari Q-learning untuk game Snake, di mana nilai Q (representasi dari tabel Q) dimuat dari file Q0.pickle jika tersedia, atau diinisialisasi sebagai defaultdict jika file tersebut tidak ada. Variabel lastMoves menyimpan gerakan terakhir ular. Fungsi paramsToState mengubah posisi makanan relatif terhadap posisi ular menjadi representasi status. Parameter params berisi informasi seperti posisi makanan (food_pos), posisi ular (snake_pos), dan tubuh ular (snake_body). Fungsi ini menghasilkan vektor relativeFoodPosition yang menandai posisi makanan relatif terhadap kepala ular (misalnya, di kanan, kiri, atas, atau bawah). Setiap kondisi diperiksa untuk memperbarui vektor ini, dan hasilnya digabungkan menjadi string rFP, yang merepresentasikan status yang dipakai untuk pembelajaran. Hal ini memungkinkan agen untuk memahami lingkungan sekitar dan menentukan tindakan terbaik berdasarkan posisi relatif makanan.



```
qlearning.py > paramsToState
def paramsToState(params):
    screenDanger = [0,0,0,0]
    if(params["screenSizeX"] - params["snake_pos"][0] == 10):                       #dangerRight
        screenDanger[0] = 1
    if(params["screenSizeX"] - params["snake_pos"][0] == params["screenSizeX"]):    #dangerLeft
        screenDanger[1] = 1
    if(params["screenSizeY"] - params["snake_pos"][1] == 10):                       #dangerBottom
        screenDanger[2] = 1
    if(params["screenSizeY"] - params["snake_pos"][1] == params["screenSizeY"]):    #dangerTop
        screenDanger[3] = 1

    sD = ""                     #als String concatenated
    for x in screenDanger:
        sD += str(x)

    ################ Danger Body (is body reachable to bite?) ##################

    snake_body = []
    skip = 0
    for pos in params["snake_body"]:            # ignore the first 4 body parts
        if (skip > 3):
            snake_body.append(pos)
        skip+=1

    bodyDanger = [0,0,0,0]
    for bodyPart in snake_body:
        #print(bodyPart)
        if(params["snake_pos"][0] - bodyPart[0] == 0 and params["snake_pos"][1] - bodyPart[1] == 10 ):  #BodyPartUp
            bodyDanger[0] = 1
        if(params["snake_pos"][0] - bodyPart[0] == 0 and params["snake_pos"][1] - bodyPart[1] == -10 ): #BodypartDown
            bodyDanger[1] = 1
        if(params["snake_pos"][0] - bodyPart[0] == 10 and params["snake_pos"][1] - bodyPart[1] == 0 ):  #BodyPartLeft
            bodyDanger[2] = 1
        if(params["snake_pos"][0] - bodyPart[0] == -10 and params["snake_pos"][1] - bodyPart[1] == 0 ): #BodypartRight
            bodyDanger[3] = 1
```

Penjelasan :

Ini merupakan bagian dari fungsi paramsToState yang mengubah parameter lingkungan game Snake menjadi status yang lebih detail untuk algoritma Q-learning. Bagian ini memetakan situasi bahaya di sekitar ular, seperti bahaya menabrak tepi layar (screenDanger) atau menabrak tubuhnya sendiri (bodyDanger). Vektor screenDanger digunakan untuk menandai jika kepala ular berada dekat dengan tepi layar di keempat arah (kanan, kiri, bawah, dan atas). Posisi tubuh ular (snake_body) juga diproses untuk menandai apakah ada bagian tubuh di dekat kepala ular, mengindikasikan bahaya jika ular bergerak ke arah tersebut. Setelah menentukan kondisi bahaya, informasi ini dikonversi menjadi string sD yang merepresentasikan status yang lebih komprehensif dari lingkungan sekitar ular. Representasi ini memungkinkan agen untuk membuat keputusan yang lebih baik dengan mempertimbangkan risiko di sekitar setiap langkah.

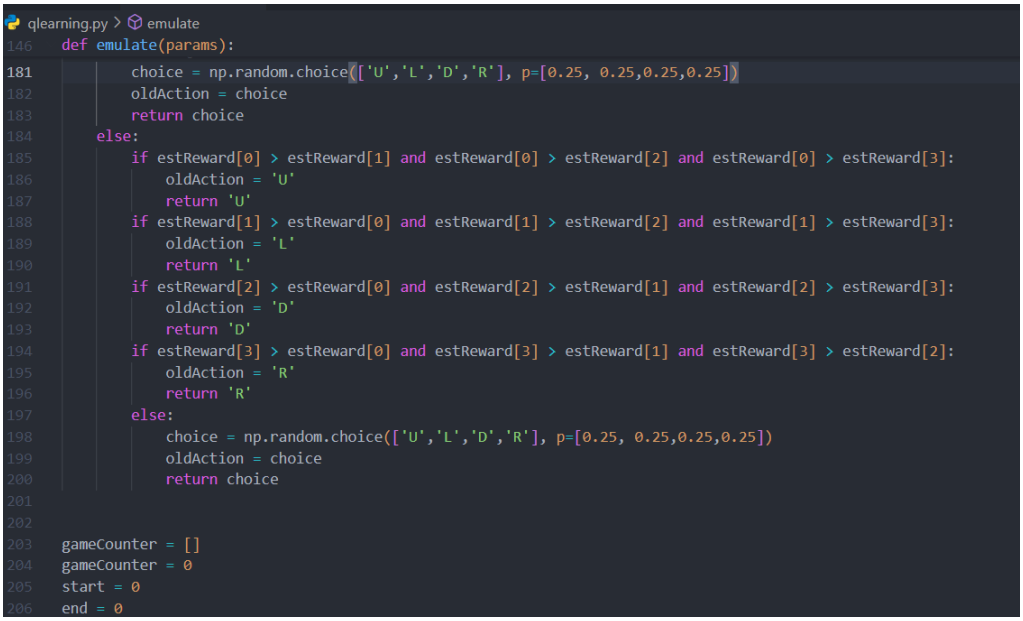
Penjelasan :

Ini melengkapi fungsi paramsToState yang mengubah parameter game Snake menjadi representasi status yang digunakan untuk pembelajaran Q-learning. Kode ini menentukan arah gerakan ular berdasarkan perbedaan posisi antara bagian pertama dan kedua tubuhnya (dx dan dy). Arah tersebut diwakili oleh nilai numerik: 0 untuk kanan, 1 untuk kiri, 2 untuk atas, dan 3 untuk bawah. Variabel bb menggabungkan informasi bahaya yang dihasilkan dari posisi tubuh ular, dan direction merepresentasikan arah gerakan saat ini. Status akhir (state) adalah kombinasi dari informasi posisi makanan (rFP), bahaya layar

(sD), bahaya tubuh (bb), dan arah gerakan (direction). Status ini digunakan untuk menentukan langkah optimal dalam algoritma Q-learning. Variabel tambahan seperti oldState, oldAction, dan gameCounter tampaknya digunakan untuk melacak status permainan sebelumnya, tindakan terakhir, dan skor selama permainan.

Penjelasan :

Ini menunjukkan fungsi emulate yang digunakan dalam algoritma Q-learning untuk game Snake. Fungsi ini mengonversi parameter permainan menjadi status menggunakan paramsToState dan mengambil nilai estimasi reward (estReward) dari tabel Q untuk status tersebut. Variabel prevReward mengambil nilai reward dari status sebelumnya (oldState). Berdasarkan aksi sebelumnya (oldAction), indeks yang sesuai dalam prevReward diperbarui menggunakan formula Q-learning, yang mempertimbangkan reward yang diterima, nilai diskonto (gamma), dan nilai maksimal dari status berikutnya (estReward). Reward dihitung secara negatif berdasarkan jumlah gerakan sejak skor terakhir, sehingga mendorong agen untuk mencapai makanan lebih cepat. Setelah memperbarui nilai reward, status saat ini disimpan ke oldState, dan keputusan selanjutnya dibuat: berdasarkan probabilitas e, agen dapat memilih tindakan secara acak (eksplorasi) atau berdasarkan tabel Q (eksploitasi). Jika eksplorasi dipilih, aksi dipilih secara acak dari pilihan arah (U, L, D, R) dengan probabilitas yang sama. Pendekatan ini membantu agen belajar untuk memilih tindakan terbaik dengan menyeimbangkan eksplorasi dan pemanfaatan pengetahuan yang diperoleh.



Penjelasan :

Ini adalah lanjutan dari fungsi emulate dalam algoritma Q-learning untuk permainan Snake. Di sini, agen memutuskan arah gerakan berikutnya berdasarkan estimasi reward (estReward) yang disimpan di tabel Q. Jika eksplorasi dipilih, gerakan acak dipilih dari daftar arah (U, L, D, R) dengan probabilitas yang sama (25% untuk masing-masing). Namun, jika agen memilih untuk mengeksploitasi, ia akan memilih arah yang memiliki reward tertinggi dari estReward untuk setiap arah. Arah dengan nilai estimasi tertinggi menentukan gerakan (U untuk atas, L untuk kiri, D untuk bawah, R untuk kanan), dan variabel oldAction diperbarui dengan pilihan tersebut. Jika ada kondisi yang sama atau ambigu di antara reward, agen kembali memilih secara acak. Pendekatan ini memungkinkan agen untuk belajar memilih tindakan yang lebih baik seiring berjalannya waktu, mengoptimalkan gerakan menuju makanan sekaligus menghindari rintangan. Variabel seperti gameCounter, start, dan end tampaknya digunakan untuk melacak kemajuan atau durasi dari setiap iterasi permainan.

```
qlearning.py > ...
    Codeium: Refactor | Explain | Generate Docstring | X
def onGameOver(score, moves):
    global oldState
    global oldAction
    global gameCounter
    global alpha, e, ed
    global start, end



    gameScores.append(score)

    #update Q of previous state (state which lead to gameOver)
    prevReward = Q[oldState]

    if oldAction == None:
        index = 0
    if oldAction == 'U':
        index = 0
    if oldAction == 'L':
        index = 1
    if oldAction == 'D':
        index = 2
    if oldAction == 'R':
        index = 3

    prevReward[index] = (1 - alpha) * prevReward[index] + \
                        alpha * rewardKill

    Q[oldState] = prevReward

    oldState = None
    oldAction = None

    #save Q as pickle
    if gameCounter % 200 == 0:
        with open("Q/" + "Q" + ".pickle","wb" ) as file:
```

Penjelasan :

Ini menunjukkan fungsi onGameOver, yang menangani pembaruan status Q-learning ketika permainan Snake berakhir. Fungsi ini menggunakan beberapa variabel global seperti oldState, oldAction, gameCounter, alpha, e, start, dan end untuk mengatur dan menyimpan status pembelajaran. Saat permainan berakhir, skor permainan (score) ditambahkan ke daftar gameScores, dan tabel Q diperbarui berdasarkan status terakhir yang menyebabkan game over. Indeks reward diperoleh dari oldAction (misalnya, U untuk gerakan ke atas atau L untuk kiri) dan digunakan untuk memperbarui nilai reward menggunakan formula Q-learning, dengan menambahkan penalti besar (rewardKill) untuk aksi yang menyebabkan kegagalan. Setelah pembaruan, oldState dan oldAction disetel ke None untuk menandakan akhir dari sesi pelatihan ini. Selain itu, setiap 200 iterasi, tabel Q disimpan ke file Q0.pickle, memungkinkan agen untuk melanjutkan pelatihan dari titik terakhir. Proses ini memungkinkan agen untuk memperbaiki perilaku dalam menghadapi situasi serupa di masa depan.

```
qlearning.py > onGameOver
def onGameOver(score, moves):
    with open("Q/" + "Q" + ".pickle","wb" ) as file:
        pickle.dump(dict(Q), file)
    print("+++++++++ Pickle saved +++++++++")

#show some stats
if gameCounter % 100 == 1:
    end = time()
    timeD = end - start
    print (str(gameCounter)+ " : " + "\t" + 'meanScore: ' +  str(np.mean(gameScores[-100:])) + "| HighScore: " + str(np.max(gameScores)) + \
           '| moves: ' + str(np.mean(moves[-100:])) + "| time for 10 games: " + str(round(timeD*10)/100))
    start = time()

#print hyperparameters
if gameCounter % 100 == 0:
    print ("a:", alpha)
    print ("e:", e)
    print ("g:", gamma)

#decrease alpha / e over time (moves)
if gameCounter % 100 == 0:
    alpha = alpha * alphaD
    if e > emin:
        e = e / ed

gameCounter += 1
```

Penjelasan :

Ini melengkapi fungsi onGameOver dengan logika untuk menyimpan data, menampilkan statistik, dan menyesuaikan parameter pembelajaran Q-learning. Setiap 200 iterasi permainan, tabel Q disimpan dalam file Q0.pickle menggunakan modul pickle, sehingga progres pembelajaran bisa dilanjutkan. Selain itu, kode ini menampilkan statistik seperti skor rata-rata (meanScore) dari 100 game terakhir, skor tertinggi (HighScore), rata-rata jumlah gerakan, dan waktu yang dibutuhkan untuk 10 game terakhir. Ini membantu memantau performa agen selama pelatihan. Ada juga mekanisme untuk mengurangi nilai alpha (learning rate) dan e (epsilon, yang mengontrol tingkat eksplorasi), sehingga agen menjadi lebih fokus pada eksploitasi strategi optimal seiring bertambahnya iterasi. Jika e mencapai nilai minimal (emin), penurunan berhenti. Kode ini memastikan bahwa agen Q-learning terus belajar dan memperbaiki strategi dengan penyesuaian otomatis berdasarkan progresnya.

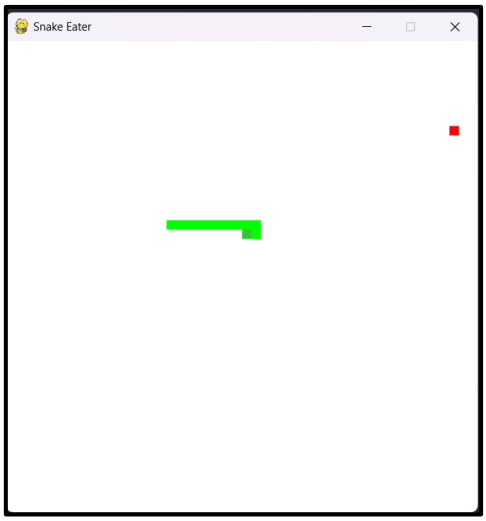
```
Codeium: Refactor | Explain | Generate Docstring | ×
def onScore(params):
    global oldState
    global oldAction
    global gameCounter

    state = paramsToState(params)

    estReward = Q[state]

    prevReward = Q[oldState]

    if oldAction == 'U':
        index = 0
    if oldAction == 'L':
        index = 1
    if oldAction == 'D':
        index = 2
    if oldAction == 'R':
        index = 3

    prevReward[index] = (1 - alpha) * prevReward[index] + \
                        alpha * (rewardScore + gamma * max(estReward) )

    Q[oldState] = prevReward



if mode == "play":
    snake.main(emulate, onGameOver, onScore)
else:
    snake_headless.main(emulate, onGameOver, onScore)
```

Penjelasan :

Kode ini mendefinisikan fungsi onScore, yang digunakan untuk memperbarui tabel Q saat agen mendapatkan skor dalam permainan Snake. Fungsi ini menggunakan status saat ini (state) yang diperoleh dari paramsToState dan mencari estimasi reward (estReward) untuk status tersebut. Berdasarkan aksi sebelumnya (oldAction), indeks yang sesuai diperbarui dalam prevReward menggunakan formula Q-learning, yang mengombinasikan reward langsung (rewardScore) dengan nilai diskonto (gamma) dari reward masa depan yang

diestimasi. Nilai reward ini kemudian disimpan kembali ke dalam Q[oldState]. Kode juga menentukan tindakan yang harus diambil ketika mode play atau train dipilih. Dalam mode play, game dijalankan menggunakan snake.main, sementara dalam mode train, game dijalankan menggunakan snake_headless.main, yang mungkin berjalan tanpa tampilan visual. Fungsi onScore dan onGameOver memungkinkan agen untuk belajar dari setiap skor yang diperoleh dan dari kondisi akhir permainan, sehingga secara bertahap mengoptimalkan kebijakannya untuk mendapatkan skor lebih tinggi di masa depan.

Output ;



Penjelasan :

Output ini menunjukkan tampilan dari permainan Snake yang sedang berjalan, seperti yang diatur oleh kode Q-learning yang dibahas sebelumnya. Di dalam jendela permainan yang berjudul "Snake Eater", ular (berwarna hijau) bergerak di layar putih, dan target makanan ditandai dengan kotak merah. Ular perlu bergerak untuk memakan kotak merah agar bertambah panjang dan mencetak poin. Agen Q-learning, seperti yang diimplementasikan dalam kode, akan membuat keputusan untuk menggerakkan ular ke arah yang paling menguntungkan berdasarkan status lingkungan saat ini. Melalui pelatihan, agen belajar untuk menghindari menabrak dinding atau tubuhnya sendiri sambil bergerak menuju makanan untuk mendapatkan poin. Tampilan ini menunjukkan visualisasi dari mode play di mana agen mungkin telah dilatih sebelumnya dan sekarang menjalankan strategi untuk mencapai makanan.

3. IMPLEMENTASI (MC) MONTE CARLO “SNAKE GAME”

Repository : https://github.com/MuhamadIqbal073/Kelompok-7-MonteCarlo-SnakeGame

Code :

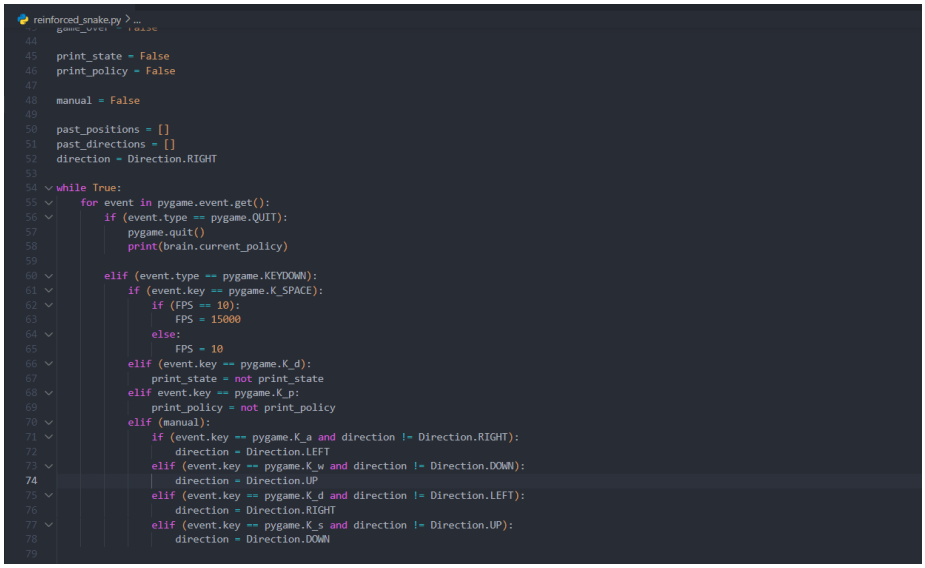
reinforced_snake.py

```python
import pygame
from montecarlo.brain import Brain
from montecarlo.state import State
from montecarlo.game.grid import Grid
from montecarlo.game.items import *
from random import choices

pygame.init()


canvas = pygame.display.set_mode((800, 800))

sheep_sprite = pygame.image.load("sprites/apple.png")
shepperd_sprite = pygame.image.load("sprites/snake.png")
cheese_sprite = pygame.image.load("sprites/apple.png")

shepperd_sprite = pygame.transform.scale(shepperd_sprite, (50, 50))
sheep_sprite = pygame.transform.scale(sheep_sprite, (50, 50))
cheese_sprite = pygame.transform.scale(cheese_sprite, (50, 50))

FPS = 10


Codeium: Refactor | Explain | Generate Docstring | ×
def update_screen():
    pygame.display.update()
    pygame.time.Clock().tick(FPS)

Codeium: Refactor | Explain | Generate Docstring | ×
def show_game_over():
    text = pygame.font.Font(None, 36).render("Game Over", True, (255, 255, 255))
    canvas.blit(text, (250, 250))




grid = Grid(16, 800)

shepperd = Shepperd(0, 5, grid)
current_sheep = Sheep(grid.random_cell(), grid.random_cell())

brain = Brain(gamma=0.78)

game_over = False
```

Penjelasan :

Kode ini adalah bagian dari permainan berbasis reinforcement learning yang menggunakan pustaka pygame untuk visualisasi. Kode ini menginisialisasi jendela permainan dengan ukuran 800x800 piksel dan memuat beberapa sprite untuk objek seperti ular (shepperd_sprite), domba (sheep_sprite), dan keju (cheese_sprite), kemudian mengubah ukurannya menjadi 50x50 piksel. Fungsi update_screen bertanggung jawab untuk memperbarui tampilan permainan pada setiap frame dengan kecepatan yang ditentukan oleh variabel FPS (frame per second), yang diatur menjadi 10. Fungsi show_game_over menampilkan teks "Game Over" di layar ketika permainan berakhir. Kode juga

menginisialisasi objek grid sebagai arena permainan, serta objek Shepperd (sepertinya adalah agen ular) dan Sheep di posisi acak dalam grid. Variabel game_over digunakan untuk mengontrol status permainan, dan sebuah objek Brain digunakan untuk mengatur proses pembelajaran dengan parameter gamma sebesar 0.78, yang mungkin berfungsi sebagai faktor diskonto dalam algoritma reinforcement learning.



Penjelasan:

Ini merupakan bagian dari loop utama untuk permainan berbasis pygame yang melibatkan pengaturan kontrol dan respons terhadap input pengguna. Beberapa variabel seperti print_state dan print_policy mengatur apakah status dan kebijakan agen akan dicetak ke layar. Variabel manual mengatur apakah kontrol permainan dilakukan secara manual oleh pengguna. Di dalam loop while True, program memeriksa berbagai event pygame seperti QUIT (untuk keluar dari permainan) dan KEYDOWN untuk memeriksa input dari tombol keyboard. Tombol SPACE mengatur kecepatan permainan antara cepat (1500 FPS) dan lambat (10 FPS). Tombol P memungkinkan pemain untuk menampilkan kebijakan agen. Jika mode manual aktif, tombol A, W, D, dan S digunakan untuk menggerakkan objek (misalnya ular) ke kiri, atas, kanan, atau bawah, dengan validasi arah agar tidak berbelok ke arah yang berlawanan secara langsung. Kode ini mengatur bagaimana permainan merespons input dari pengguna untuk mengontrol arah pergerakan dalam permainan Snake.

```
reinforced_snake.py > ...

    if (not manual):
        pass
        state = State(shepperd.get_sheep_direction(current_sheep), shepperd.get_queue_directions(past_positions[1:shepperd.sheeps], direction))
        if (print_state):
            print(state)
        direction = brain.choose_direction(state, direction)



    past_directions.insert(0, direction)
    if (len(past_directions) >= 100):
        past_directions.pop()

    shepperd.move(direction)


    if (len(past_positions) >= 200):
        past_positions.pop()

    if (shepperd.x_cell == current_sheep.x_cell and shepperd.y_cell == current_sheep.y_cell):
        current_sheep = Sheep(grid.random_cell(), grid.random_cell())
        shepperd.sheeps += 1
        brain.add_reward(50)
        #brain.evaluate()
    else:
        brain.add_reward(-1)

    for i in range(shepperd.sheeps):
        if ((shepperd.x_cell, shepperd.y_cell) == past_positions[i]):
            shepperd = Shepperd(0, 5, grid)
            current_sheep = Sheep(grid.random_cell(), grid.random_cell())
            brain.add_reward(-300)
            brain.evaluate()
            break


    past_positions.insert(0, (shepperd.x_cell, shepperd.y_cell))

    if (print_policy):
        print(brain.current_policy)


    canvas.fill((255, 255, 255))
```

Penjelasan:

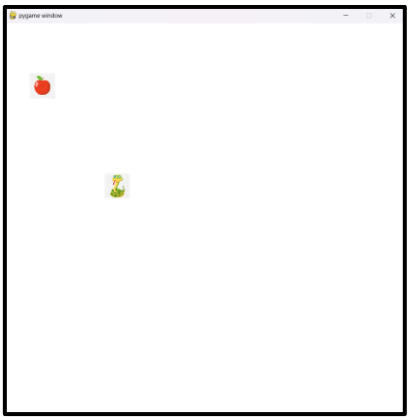
Ini merupakan bagian dari permainan berbasis pygame yang menggunakan reinforcement learning untuk mengontrol agen (shepperd). Jika permainan tidak dalam mode manual, agen menggunakan status saat ini untuk memilih arah berdasarkan kebijakan dari objek brain, yang ditentukan oleh metode brain.choose_direction(). Arah yang dipilih kemudian digunakan untuk menggerakkan agen. Kode ini juga melacak hingga 100 arah sebelumnya (past_directions) dan posisi-posisi sebelumnya hingga 200 posisi (past_positions) untuk menganalisis pergerakan agen. Saat agen bertabrakan dengan objek sheep (domba), agen mendapatkan reward positif (+50), dan posisi domba diacak ulang. Namun, jika tidak terjadi tabrakan, agen mendapatkan penalti kecil (-1). Jika agen bertabrakan dengan posisi sebelumnya (yang bisa berarti menabrak dirinya sendiri), penalti besar diberikan (-300), dan posisi direset. Kondisi ini memungkinkan agen belajar untuk menghindari tabrakan dan lebih efektif mengejar domba. Jika print_policy diaktifkan, kebijakan saat ini dicetak ke layar. Selain itu, tampilan di layar diperbarui dengan membersihkan canvas menggunakan canvas.fill((255, 255, 255)) untuk menggambar ulang objek di posisi baru.

```
canvas.blit(shepperd_sprite, (grid.from_cell(shepperd.x_cell), grid.from_cell(shepperd.y_cell)))
for i in range(0, shepperd.sheeps):
    canvas.blit(cheese_sprite, (grid.from_cell(past_positions[i + 1][0]), grid.from_cell(past_positions[i + 1][1])))



canvas.blit(sheep_sprite, (grid.from_cell(current_sheep.x_cell), grid.from_cell(current_sheep.y_cell)))


update_screen()
```

Penjelasan:

Kode ini menampilkan logika untuk menggambar ulang objek di atas layar permainan menggunakan pygame. Objek shepperd_sprite (yang mungkin mewakili ular atau karakter pemain) digambar pada posisi koordinat yang diperoleh dari shepperd.x_cell dan shepperd.y_cell menggunakan metode from_cell dari objek grid untuk mengubah koordinat sel ke koordinat layar. Dalam loop, beberapa objek cheese_sprite (kemungkinan representasi dari makanan atau jejak) digambar berdasarkan posisi masa lalu yang tersimpan dalam past_positions, memberikan visualisasi jejak pergerakan. Kemudian, sheep_sprite digambar di posisi current_sheep, menunjukkan posisi domba saat ini di grid. Terakhir, fungsi update_screen() dipanggil untuk memperbarui tampilan di layar dan menampilkan semua perubahan yang dilakukan. Kode ini memastikan bahwa setiap pergerakan karakter dan objek lainnya di layar ditampilkan dengan benar setiap frame.

Output :



Penjelasan:

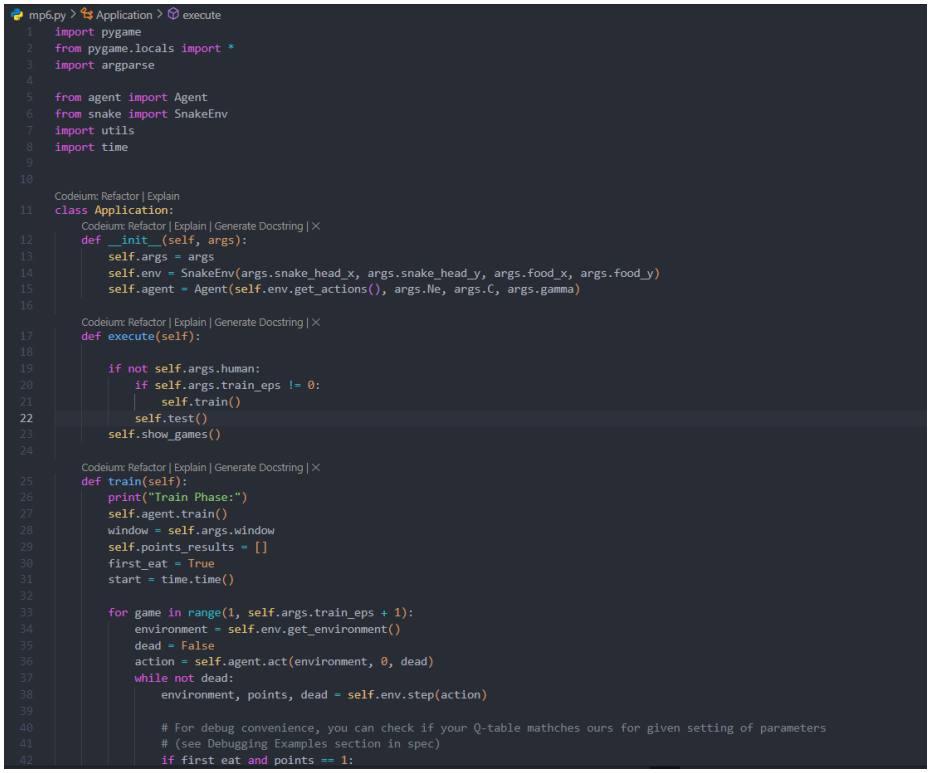
Output ini adalah jendela permainan pygame yang menunjukkan visualisasi dari game berbasis reinforcement learning. Di layar, terlihat dua objek utama: gambar apel (atau mungkin keju) dan ular (atau karakter shepperd). Apel tampaknya berfungsi sebagai target yang harus dicapai oleh ular atau karakter yang dikendalikan oleh agen pembelajaran. Ular akan bergerak di layar untuk mencoba mendekati dan "memakan" apel, yang memberi reward positif dalam proses pelatihan. Posisi apel dan ular diatur dalam grid, dan pergerakan serta interaksi di antara mereka ditampilkan secara visual dalam jendela ini.

Visualisasi ini membantu pengguna melihat bagaimana agen belajar dan bertindak dalam lingkungan, mengikuti kebijakan yang telah dilatihnya untuk mencapai target.

4. IMPLEMENTASI (TD) TEMPORAL DIFFERENCE “SNAKE GAME”

Repository : https://github.com/MuhamadIqbal073/Kelompok-7-Temporal-Difference-snakeGame

Code :

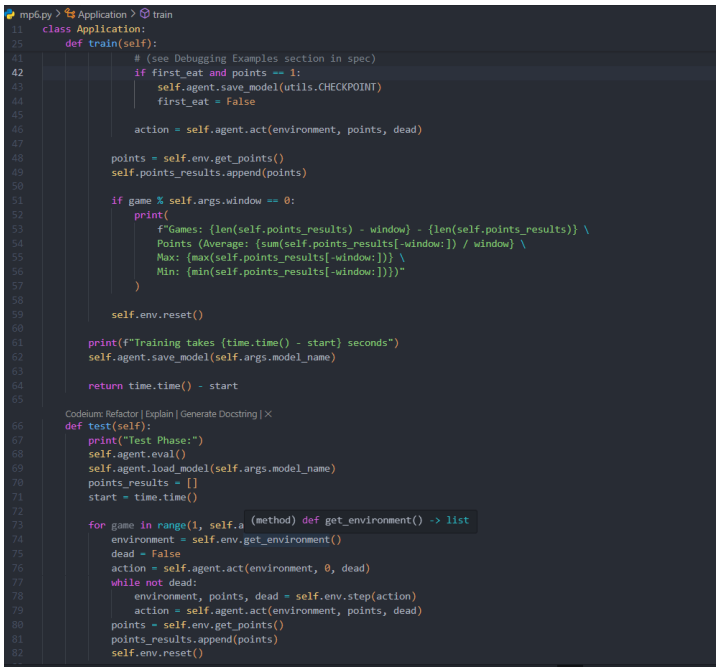


```
mp6.py > Application > execute
import pygame
from pygame.locals import *
import argparse

from agent import Agent
from snake import SnakeEnv
import utils
import time


Codeium: Refactor | Explain
class Application:
    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, args):
        self.args = args
        self.env = SnakeEnv(args.snake_head_x, args.snake_head_y, args.food_x, args.food_y)
        self.agent = Agent(self.env.get_actions(), args.Ne, args.C, args.gamma)

    Codeium: Refactor | Explain | Generate Docstring | ×
    def execute(self):

        if not self.args.human:
            if self.args.train_eps != 0:
                self.train()
            self.test()
        self.show_games()

    Codeium: Refactor | Explain | Generate Docstring | ×
    def train(self):
        print("Train Phase:")
        self.agent.train()
        window = self.args.window
        self.points_results = []
        first_eat = True
        start = time.time()

        for game in range(1, self.args.train_eps + 1):
            environment = self.env.get_environment()
            dead = False
            action = self.agent.act(environment, 0, dead)
            while not dead:
                environment, points, dead = self.env.step(action)

                # For debug convenience, you can check if your Q-table mathches ours for given setting of parameters
                # (see Debugging Examples section in spec)
                if first_eat and points == 1:
```

Penjelasan :

Kode ini mendefinisikan kelas Application untuk menjalankan game Snake dengan agen berbasis reinforcement learning. Kelas ini menggunakan pustaka pygame untuk antarmuka, serta argparse untuk menerima parameter dari pengguna. Metode __init__ menginisialisasi objek SnakeEnv (lingkungan permainan) dan Agent (agen yang akan dilatih atau diuji). Metode execute menjalankan proses pelatihan (train) atau pengujian (test) jika mode manusia (self.args.human) tidak diaktifkan, lalu menampilkan hasil permainan dengan show_games(). Metode train melatih agen melalui sejumlah episode yang ditentukan (self.args.train_eps), di mana setiap episode agen bertindak dalam

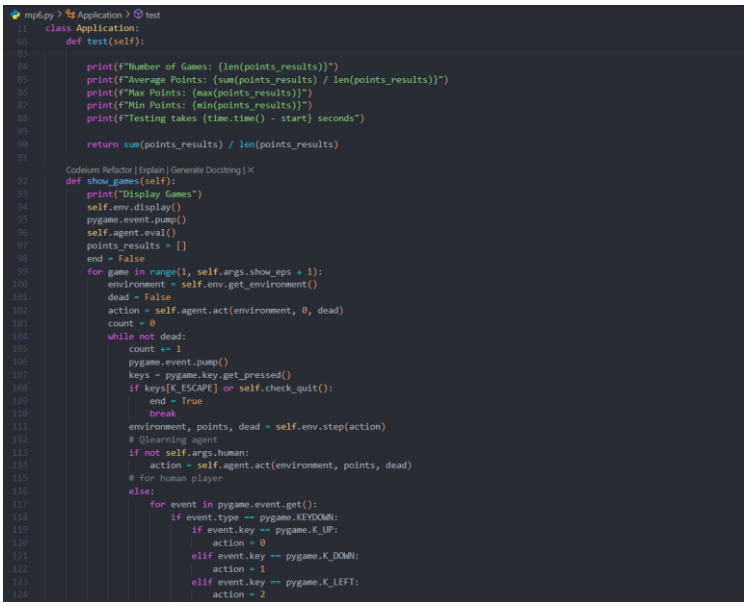
lingkungan untuk memaksimalkan reward dengan belajar dari tindakan sebelumnya. Kode ini mengatur alur pelatihan agen untuk belajar bermain Snake secara optimal, menyesuaikan kebijakan Q-learning berdasarkan pengalaman dari setiap episode pelatihan.



Penjelasan :

Kode ini memperluas metode train dan test dalam kelas Application untuk melatih dan menguji agen yang bermain game Snake. Metode train melatih agen melalui beberapa episode, mencatat poin yang diperoleh dalam setiap episode. Ketika agen pertama kali memakan makanan (saat points mencapai 1), modelnya disimpan sebagai checkpoint. Selama pelatihan, tindakan agen diambil berdasarkan kondisi lingkungan, dan hasilnya disimpan dalam points_results. Setiap beberapa episode, statistik seperti rata-rata, maksimum, dan minimum poin dihitung dan ditampilkan untuk memantau progres pelatihan. Setelah setiap episode, lingkungan direset. Setelah pelatihan selesai, model akhir agen disimpan. Metode test menjalankan agen dalam mode evaluasi, memuat model yang dilatih, dan menjalankan beberapa episode tanpa pelatihan lebih lanjut. Agen bertindak dalam lingkungan dan skor setiap episode dicatat, memungkinkan evaluasi performa model yang telah dilatih. Kedua metode ini bertujuan untuk mengembangkan dan menguji kemampuan agen

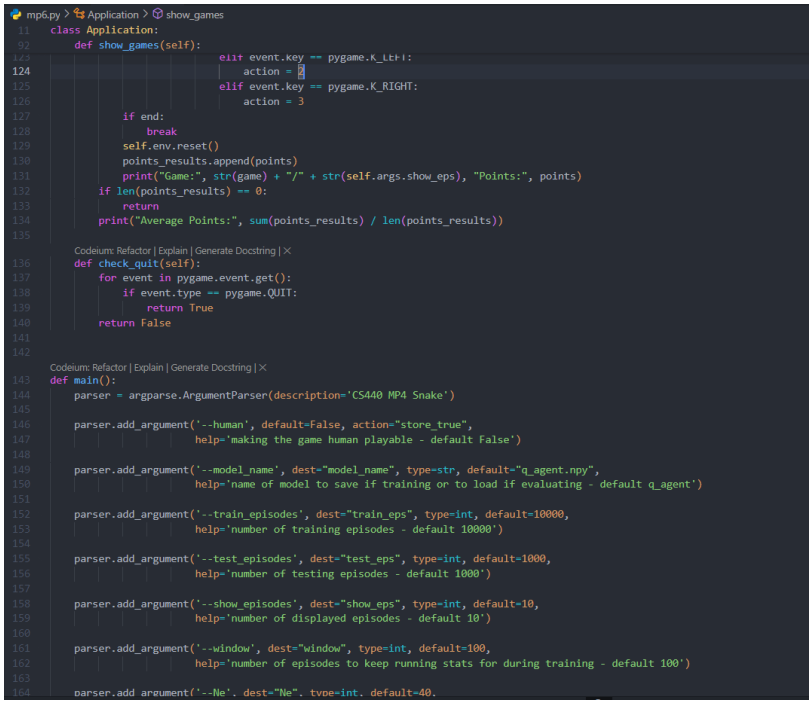
dalam bermain Snake secara efisien berdasarkan pengalaman yang diperoleh selama pelatihan.



```
mp6.py > Application > test
class Application:
    def test(self):

        print(f"Number of Games: {len(points_results)}")
        print(f"Average Points: {sum(points_results) / len(points_results)}")
        print(f"Max Points: {max(points_results)}")
        print(f"Min Points: {min(points_results)}")
        print(f"Testing takes {time.time() - start} seconds")

        return sum(points_results) / len(points_results)

    Codeium: Refactor | Explain | Generate Docstring | ×
    def show_games(self):
        print("Display Games")
        self.env.display()
        pygame.event.pump()
        self.agent.eval()
        points_results = []
        end = False
        for game in range(1, self.args.show_eps + 1):
            environment = self.env.get_environment()
            dead = False
            action = self.agent.act(environment, 0, dead)
            count = 0
            while not dead:
                count += 1
                pygame.event.pump()
                keys = pygame.key.get_pressed()
                if keys[K_ESCAPE] or self.check_quit():
                    end = True
                    break
                environment, points, dead = self.env.step(action)
                # Qlearning agent
                if not self.args.human:
                    action = self.agent.act(environment, points, dead)
                # for human player
                else:
                    for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN:
                            if event.key == pygame.K_UP:
                                action = 0
                            elif event.key == pygame.K_DOWN:
                                action = 1
                            elif event.key == pygame.K_LEFT:
                                action = 2
```

Penjelasan :

Kode ini melanjutkan metode test dan show_games dalam kelas Application untuk menampilkan hasil pengujian dan menjalankan permainan Snake secara visual. Metode test mengevaluasi agen yang sudah dilatih dengan menjalankan beberapa episode dan mencatat skor setiap episode dalam points_results. Setelah pengujian selesai, kode mencetak statistik seperti jumlah total permainan, rata-rata, nilai maksimum, dan minimum dari poin yang diperoleh, serta waktu yang dibutuhkan untuk pengujian. Hasil rata-rata poin dikembalikan sebagai output dari fungsi. Metode show_games digunakan untuk menampilkan permainan secara visual, baik untuk agen otomatis maupun untuk pemain manusia jika self.args.human diaktifkan. Ini menginisialisasi lingkungan dan kemudian menjalankan loop permainan di mana agen mengambil tindakan berdasarkan status saat ini. Jika permainan diatur untuk mode manusia, input keyboard seperti panah (K_UP, K_DOWN, K_LEFT, K_RIGHT) digunakan untuk mengendalikan gerakan. Jika tidak, agen yang terlatih membuat keputusan secara otomatis. Metode ini memungkinkan pengguna untuk melihat bagaimana agen bermain di lingkungan permainan, baik dalam mode otomatis maupun manual.



Penjelasan ;

Kode ini melanjutkan implementasi metode show_games, serta menambahkan metode check_quit dan definisi fungsi main() untuk mengatur parameter program. Metode show_games memungkinkan pemain manusia untuk mengendalikan agen Snake menggunakan tombol panah (K_LEFT untuk kiri, K_RIGHT untuk kanan) atau membiarkan agen otomatis menggerakkan karakter berdasarkan kebijakan yang telah dipelajari. Saat permainan selesai atau tombol QUIT ditekan, lingkungan direset dan hasil poin dari setiap game disimpan dalam points_results dan ditampilkan di konsol. Metode check_quit menangani event keluar dari pygame, memastikan permainan dapat dihentikan saat pengguna menutup jendela. Fungsi main() mengatur parsing argumen dari command line menggunakan argparse, menyediakan opsi seperti --human untuk mengaktifkan kontrol manual, --model_name untuk menentukan nama file model Q-learning, --train_episodes untuk jumlah episode pelatihan, --test_episodes untuk jumlah episode pengujian, dan --show_episodes untuk menampilkan beberapa episode secara visual. Parameter ini memungkinkan pengguna untuk mengkustomisasi perilaku program dan menyesuaikan proses pelatihan dan pengujian agen. Kode ini mengatur cara interaksi pengguna dengan permainan dan mengelola bagaimana agen dilatih dan diuji.

```
    parser.add_argument('--Ne', dest="Ne", type=int, default=40,
                        help='the Ne parameter used in exploration function - default 40')

    parser.add_argument('--C', dest="C", type=int, default=40,
                        help='the C parameter used in learning rate - default 40')

    parser.add_argument('--gamma', dest="gamma", type=float, default=0.7,
                        help='the gamma paramter used in learning rate - default 0.7')

    parser.add_argument('--snake_head_x', dest="snake_head_x", type=int, default=5,
                        help='initialized x position of snake head  - default 5')

    parser.add_argument('--snake_head_y', dest="snake_head_y", type=int, default=5,
                        help='initialized y position of snake head  - default 5')

    parser.add_argument('--food_x', dest="food_x", type=int, default=2,
                        help='initialized x position of food  - default 2')

    parser.add_argument('--food_y', dest="food_y", type=int, default=2,
                        help='initialized y position of food  - default 2')

    args = parser.parse_args()
    app = Application(args)
    app.execute()


if __name__ == "__main__":
    main()
```

Penjelasan :

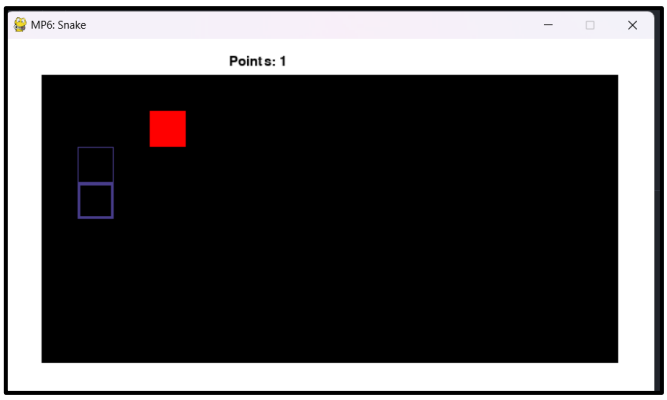
Kode ini melanjutkan pengaturan argumen argparse dalam fungsi main() untuk mengkonfigurasi parameter yang memengaruhi proses pembelajaran agen dalam game Snake. Beberapa parameter yang diatur meliputi -Ne, -C, dan --gamma, yang digunakan dalam fungsi eksplorasi dan pembelajaran (default masing-masing 40 dan 0.7). Parameter ini mengatur bagaimana agen mengeksplorasi lingkungannya dan menyesuaikan nilai Q berdasarkan pengalaman. Argumen lainnya mengatur posisi awal kepala ular (--snake_head_x, --snake_head_y) dan makanan (--food_x, --food_y), yang masing-masing diatur ke nilai default (5,5) dan (2,2). Pengguna dapat mengubah posisi awal ini untuk memulai game dalam kondisi yang berbeda. Setelah argumen diproses dengan parser.parse_args(), sebuah instance dari kelas Application dibuat dengan argumen yang diberikan, dan kemudian permainan dijalankan menggunakan app.execute(). Fungsi ini memastikan bahwa pengguna memiliki kontrol penuh terhadap parameter permainan melalui command line, memungkinkan penyesuaian yang fleksibel terhadap proses pelatihan agen.

Output :

```
Min Points: 2
Testing takes 0.5553672313690186 seconds
Testing takes 0.5553672313690186 seconds
Display Games
Game: 1/10 Points: 6
Game: 2/10 Points: 5
Game: 3/10 Points: 9
Game: 4/10 Points: 7
Game: 5/10 Points: 6
Game: 6/10 Points: 10
Game: 5/10 Points: 6
Game: 6/10 Points: 10
Game: 6/10 Points: 10
Game: 7/10 Points: 8
Game: 8/10 Points: 11
Game: 9/10 Points: 11
Game: 10/10 Points: 6
Average Points: 7.9
PS D:\RL\UTS\(TD) Temporal Difference-snakeGame>
```

Penjelasan:

Output ini menunjukkan hasil pengujian dari agen yang bermain game Snake. Setelah sesi pengujian, sistem menampilkan waktu yang dibutuhkan untuk menyelesaikan pengujian (Testing takes 0.5553672313690186 seconds) dan hasil dari beberapa game yang dijalankan selama pengujian. Setiap game ditampilkan dengan jumlah poin yang diperoleh oleh agen dalam format Game: X/10 Points: Y, di mana X adalah nomor game dan Y adalah poin yang diperoleh dalam game tersebut. Di sini, agen menyelesaikan 10 game, dengan skor bervariasi dari 5 hingga 11 poin per game. Sistem juga menampilkan rata-rata poin yang diperoleh agen dari 10 game tersebut (Average Points: 7.9). Rata-rata ini memberikan gambaran mengenai performa agen dalam mengumpulkan poin selama beberapa game. Hasil ini menunjukkan bagaimana agen beradaptasi dengan lingkungan dan mengambil keputusan untuk mendapatkan skor yang optimal, dan waktu eksekusi yang cepat menunjukkan efisiensi dalam menjalankan simulasi.



Penjelasan:

Output ini menampilkan jendela permainan Snake yang dijalankan menggunakan pygame, dengan judul "MP6: Snake". Di dalam jendela tersebut, terlihat area permainan dengan latar belakang hitam. Pada area permainan, terdapat objek berbentuk kotak merah yang berfungsi sebagai makanan yang harus dikumpulkan oleh ular (agen). Ular diwakili oleh kotak-kotak berwarna biru tua yang menampilkan tubuhnya yang bertambah panjang setelah memakan makanan. Di bagian atas layar, terdapat teks "Points: 1", yang menunjukkan bahwa ular (agen) telah berhasil memakan satu makanan,

menghasilkan satu poin dalam prosesnya. Tujuan dari permainan ini adalah agar ular bergerak di layar, menghindari tabrakan dengan dinding atau tubuhnya sendiri, dan terus mengumpulkan makanan untuk meningkatkan skor. Agen yang bermain ini kemungkinan telah dilatih untuk mengarahkan ular menuju makanan secara otomatis berdasarkan kebijakan yang dipelajari menggunakan Q-learning atau algoritma reinforcement learning lainnya.