REPORT REINFORCEMENT LEARNING
TIC TAC TOE

Disusun Oleh:

Kelompok 6

1. **Mira Juwita Ali (G1A021056)**
2. **Syakira Az Zahra (G1A021057)**
3. **Triana Kesumaningrum (G1A021068)**

Dosen Mata Kuliah:

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

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

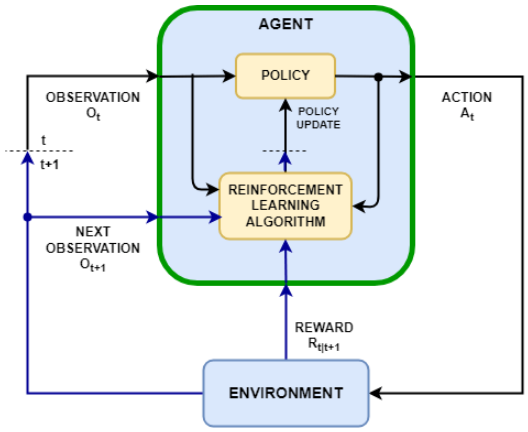
UNIVERSITAS BENGKULU

2024

PENDAHULUAN

Tic-tac-toe (Bahasa Inggris Amerika), *noughts and crosses* (Bahasa Inggris Persemakmuran), atau Xs and Os (Bahasa Inggris Kanada atau Irlandia) adalah permainan kertas dan pensil untuk dua pemain yang bergiliran menandai ruang dalam kotak tiga-kali-tiga dengan X atau O. Pemain yang berhasil menempatkan tiga tanda mereka dalam baris horizontal, vertikal, atau diagonal adalah pemenangnya. Ini adalah permainan yang diselesaikan, dengan undian yang dipaksakan dengan asumsi permainan terbaik dari kedua pemain. Tic-tac-toe dimainkan pada kotak berukuran tiga kali tiga oleh dua pemain, yang secara bergantian menempatkan tanda X dan O di salah satu dari sembilan ruang di kotak tersebut.

Agen Q-learning dan SARSA diimplementasikan di agent.py. Masing-masing dari dua agen pembelajaran mewarisi dari kelas pembelajar induk. Perbedaan utama antara keduanya adalah fungsi pembaruan nilai Q mereka. Agen Guru diimplementasikan di teacher.py. Guru mengetahui kebijakan optimal untuk setiap status yang disajikan. Namun, agen ini hanya mengambil pilihan optimal dengan probabilitas yang ditetapkan. Di game.py, kelas permainan utama ditemukan. Kelas Permainan menyimpan status setiap instansi permainan tertentu, dan berisi sebagian besar fungsionalitas permainan utama. Loop permainan utama dapat ditemukan di fungsi kelas playGame().



Dalam *reinforcement learning,* agen berinteraksi dengan lingkungan yang tidak pasti, melakukan tindakan, dan terus dilatih untuk mempelajari cara berinteraksi dengan benar dengan dunia yang dinamis. Agen terdiri dari jaringan kebijakan dan algoritma *reinforcement learning* seperti pembelajaran Q learning atau SARSA. Mari kita gunakan permainan tic-tac-toe sebagai contoh. *Environment* atau lingkungan direpresentasikan oleh permainan tic-tac-toe. Agen (x) mengamati status yang

direpresentasikan oleh konfigurasi papan saat ini (posisi × dan ∘). Contoh status dapat terlihat seperti berikut:

$$\begin{matrix} \circ & & \circ \\ & \times & \\ \circ & \times & \times \end{matrix}$$

Berdasarkan gambar, agen melakukan suatu tindakan. Tindakan ini menyebabkan lingkungan bertransisi ke status baru. Tindakan yang tersedia adalah serangkaian gerakan yang diizinkan. Setelah tindakan, lingkungan memberikan hadiah. Hadiah tersebut adalah nilai skalar, di mana nilai yang lebih tinggi lebih baik. Tindakan agen didasarkan pada suatu kebijakan. Kebijakan adalah fungsi yang memetakan status (observasi lingkungan saat ini) ke distribusi probabilitas tindakan yang akan diambil dan dapat dimodelkan oleh jaringan saraf yang parameterθ-nya dipelajari.

$$\text{action} = \text{policy}(\text{state};\theta)$$

Selama pelatihan, agen berinteraksi dengan lingkungan dan metode pengoptimalan yang dipilih, menyesuaikan kebijakan agen untuk memaksimalkan ekspektasi hadiah di masa mendatang.

$$\begin{matrix} -1 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 1 & 1 \end{matrix} = \begin{matrix} \circ & & \circ \\ & \times & \\ \circ & \times & \times \end{matrix}$$

Berdasarkan kondisi tersebut, agen memilih satu dari sembilan aksi yang tersedia. Aksi-aksi ini ditentukan oleh kebijakan (*policy*) agen saat ini. Di sini, kebijakan tersebut dimodelkan sebagai jaringan saraf (*neural network*) dengan sembilan neuron keluaran. Aksi-aksi tersebut berbentuk bilangan bulat dan diambil dengan menerapkan fungsi argmax pada neuron-neuron keluaran jaringan tersebut. Meskipun ada sembilan aksi yang tersedia, tidak semua gerakan diperbolehkan. Jika aksi yang dipilih sah, lawan akan bergerak, dan agen mengamati kondisi baru permainan. Kondisi di mana permainan dimenangkan akan memberikan reward sebesar +1. Kalah dalam permainan atau menandai area yang sudah ditempati akan memberikan reward sebesar -1. Hukuman untuk gerakan yang salah mendorong agen untuk mempelajari hanya gerakan yang sah seiring waktu. Semua kondisi lainnya (termasuk hasil seri) akan memberikan reward sebesar 0.

PENJELASAN KODE

```python
import random

class Teacher:
    def __init__(self, ability_level=0.5):
        # Ability level now determines how smart the Teacher is
        self.ability_level = ability_level
        self.memory = []  # Adding memory for a learning element

    def make_move(self, board):
        # Decision-making logic based on teacher's ability level
        if random.random() > self.ability_level:
            return self.random_move(board)

        if self.win(board, key='X'):
            return self.find_winning_move(board, key='X')

        if self.block_win(board):
            return self.block_win(board)

        if self.fork(board):
            return self.fork(board)

        if self.block_fork(board):
            return self.block_fork(board)

        return self.random_move(board)

    def win(self, board, key):
        # Check if the teacher can win in the next move
        return self.find_winning_move(board, key)

    def find_winning_move(self, board, key):
        # Check rows, columns, and diagonals for winning move
        for i in range(4):
            # Check rows
            if board[i].count(key) == 3 and board[i].count(' ') == 1:
                return board.index(i)
            # Check columns
            column = [board[j][i] for j in range(4)]
            if column.count(key) == 3 and column.count(' ') == 1:
                return column.index(' ') * 4 + i
        # Check diagonals
        if all(board[i][i] == key for i in range(4)):
            for i in range(4):
                if board[i][i] == ' ':
                    return i * 4 + i
        if all(board[i][3 - i] == key for i in range(4)):
            for i in range(4):
                if board[i][3 - i] == ' ':
                    return i * 4 + (3 - i)
        return None
```

```python
def block_win(self, board):
    # Block opponent's winning move
    return self.find_winning_move(board, key='O')

def fork(self, board):
    # Logic to create a fork opportunity
    for i in range(4):
        if board[i].count('X') == 2 and board[i].count(' ') == 2:
            return i  # Return the index of the row to create a fork
    return None

def block_fork(self, board):
    # Logic to block opponent's fork opportunities
    for i in range(4):
        if board[i].count('O') == 2 and board[i].count(' ') == 2:
            return i  # Return the index of the row to block a fork
    return None

def random_move(self, board):
    # Teacher makes a random move if no better move is available
    available_moves = [i for i, cell in enumerate(board) if cell == ' ']
    return random.choice(available_moves) if available_moves else None

def learn(self, outcome):
    # Learning from the outcome (win/loss/draw)
    self.memory.append(outcome)
    if len(self.memory) > 100:
        self.memory.pop(0)  # Forget old outcomes if memory exceeds size
    # Adjust the ability or strategy based on previous outcomes
```

```python
# Example of how to use the Teacher class in a game loop
def print_board(board):
    for row in board:
        print(' | '.join(row))
        print('-' * 15)

def play_game():
    board = [[' ' for _ in range(4)] for _ in range(4)]
    teacher = Teacher(ability_level=0.7)

    for turn in range(16):
        print_board(board)
        if turn % 2 == 0:  # Teacher's turn
            move = teacher.make_move(board)
            if move is not None:
                board[move // 4][move % 4] = 'X'
        else:  # Player's turn (you can add player input logic here)
            move = int(input("Enter your move (0-15): "))
            if board[move // 4][move % 4] == ' ':
                board[move // 4][move % 4] = 'O'
            else:
                print("Masukkan tidak valid. Harap coba lagi.")
                continue

        # Check for win or draw condition
        # (Implement your win/draw checking logic here)

    print_board(board)
    print("Game Over!")

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

Gambar 1. teacher.py

Kode di atas mendefinisikan kelas Teacher, yang berfungsi sebagai pemain dalam permainan Tic-Tac-Toe. Kelas ini mengatur tingkat kecerdasan dalam membuat keputusan dengan parameter ability_level. Semakin tinggi nilai parameter tersebut, semakin sering Teacher membuat tindakan terbaik. Dalam metode make_move, teacher memilih langkah berdasarkan beberapa skenario: mencoba untuk menang, mencegah kemenangan lawan, menciptakan peluang ganda (fork) untuk menang, atau mencegah fork lawan. Jika tidak ada strategi terbaik, langkah tersebut diambil secara acak. Metode find_winning_move digunakan untuk mengidentifikasi langkah yang memungkinkan kemenangan dengan melihat baris, kolom, dan diagonal. Selain itu, Teacher memiliki metode learn untuk menyimpan hasil permainan dalam ingatan dan belajar dari hasilnya. Pada permainan, teacher (diwakili oleh simbol "X") dan pemain (diwakili oleh simbol "O") berganti-ganti mengambil langkah.

```python
import random

class Game:
    """ The game class. New instance created for each new game. """
    def __init__(self, agent, teacher=None):
        self.agent = agent
        self.teacher = teacher
        # Initialize the game board
        self.board = [['-', '-', '-', '-'],
                      ['-', '-', '-', '-'],
                      ['-', '-', '-', '-'],
                      ['-', '-', '-', '-']]

    def playerMove(self):
        """ Query player for a move and update the board accordingly. """
        if self.teacher is not None:
            action = self.teacher.makeMove(self.board)
            self.board[action[0]][action[1]] = 'X'
        else:
            printBoard(self.board)
            while True:
                move = input("Giliranmu! Silakan pilih baris dan kolom dari 0-3 "
                             "dalam format baris, kolom: ")
                print('\n')
                try:
                    row, col = map(int, move.split(','))
                except ValueError:
                    print("MASUKAN TIDAK VALID! Harap gunakan format yang benar.")
                    continue
                if row not in range(4) or col not in range(4) or self.board[row][col] != '-':
                    print("MASUKAN TIDAK VALID! Silahkan pilih kembali.")
                    continue
                self.board[row][col] = 'X'
                break

    def agentMove(self, action):
        """ Update board according to agent's move. """
        self.board[action[0]][action[1]] = 'O'

    def checkForWin(self, key):
        """ Check to see whether the player/agent with token 'key' has won. """
        # Check for player win on diagonals
        a = [self.board[i][i] for i in range(4)]
        b = [self.board[i][3 - i] for i in range(4)]
        if a.count(key) == 4 or b.count(key) == 4:
            return True
        # Check for player win on rows/columns
        for i in range(4):
            col = [self.board[j][i] for j in range(4)]
            row = self.board[i]
            if col.count(key) == 4 or row.count(key) == 4:
                return True
        return False
```

```python
def checkForDraw(self):
    """ Check to see whether the game has ended in a draw. """
    return all(elt != '-' for row in self.board for elt in row)

def checkForEnd(self, key):
    """ Checks if player/agent with token 'key' has ended the game. """
    if self.checkForWin(key):
        if self.teacher is None:
            printBoard(self.board)
            if key == 'X':
                print("Pemain menang!")
            else:
                print("Agent RL menang!")
        return 1
    elif self.checkForDraw():
        if self.teacher is None:
            printBoard(self.board)
            print("Seri!")
        return 0
    return -1
```

```python
def playGame(self, player_first):
    """ Begin the tic-tac-toe game loop. """
    # Initialize the agent's state and action
    if player_first:
        self.playerMove()
    prev_state = getStateKey(self.board)
    prev_action = self.agent.get_action(prev_state)

    # Iterate until game is over
    while True:
        self.agentMove(prev_action)
        check = self.checkForEnd('O')
        if check != -1:
            reward = check
            break
        self.playerMove()
        check = self.checkForEnd('X')
        if check != -1:
            reward = -1 * check
            break
        reward = 0
        new_state = getStateKey(self.board)

        # Determine new action (epsilon-greedy)
        new_action = self.agent.get_action(new_state)
        # Update Q-values
        self.agent.update(prev_state, new_state, prev_action, new_action, reward)
        # Reset "previous" values
        prev_state = new_state
        prev_action = new_action

    # Game over. Perform final update
    self.agent.update(prev_state, None, prev_action, None, reward)
```

```python
    def start(self):
        """ Determine who moves first, and subsequently, start the game. """
        if self.teacher is not None:
            if random.random() < 0.5:
                self.playGame(player_first=False)
            else:
                self.playGame(player_first=True)
        else:
            while True:
                response = input("Apakah kamu ingin mulai duluan? [y/n]: ")
                print('')
                if response.lower() in ['n', 'no']:
                    self.playGame(player_first=False)
                    break
                elif response.lower() in ['y', 'yes']:
                    self.playGame(player_first=True)
                    break
                else:
                    print("Masukan tidak valid. Masukkan 'y' atau 'n'.")

def printBoard(board):
    """ Prints the game board as text output to the terminal. """
    print('    0   1   2   3\n')
    for i, row in enumerate(board):
        print('%i   ' % i, end='')
        for elt in row:
            print('%s   ' % elt, end='')
        print('\n')

def getStateKey(board):
    """ Converts 2D list representing the board state into a string key for that state. """
    return ''.join(''.join(row) for row in board)
```

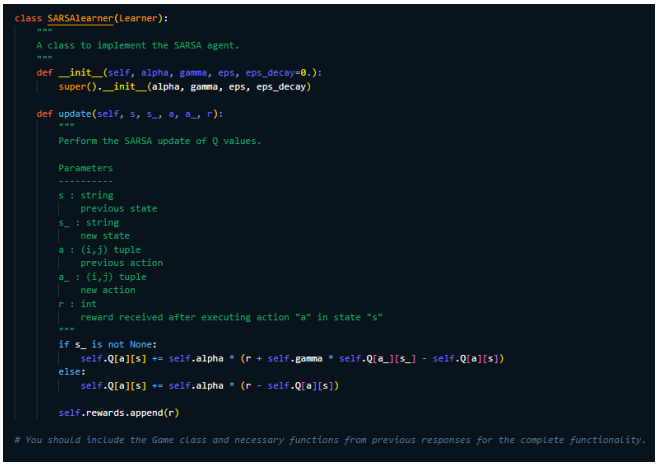
Gambar 2. game.py

Kelas game didefinisikan oleh kode di atas dan digunakan untuk mengelola permainan Tic-Tac-Toe 4x4 antara dua pemain: teacher (yang memiliki token "X") dan agen RL (yang memiliki token "O"). Sebuah instance baru dari kelas permainan dibuat setiap kali permainan baru dimulai. Dua parameter diberikan kepada game dalam konstruktor (__init__): agen (agen RL yang akan mengambil tindakan berdasarkan model pembelajaran) dan teacher (opsional, jika ada pelatih yang dapat memainkan permainan secara otomatis). Papan permainan dimulai

sebagai matriks 4 kali 4 dengan simbol "-" yang menunjukkan ruang kosong. Metode playerMove() digunakan untuk meminta pemain manusia memasukkan langkah mereka. Jika ada instruktur, instruktur akan membuat langkah berdasarkan logika metode makeMove, tetapi jika tidak, pemain akan memilih langkah dengan memasukkan koordinat baris dan kolom papan. Jika tidak, pemain akan memasukkan koordinat baris dan kolom papan untuk memilih langkah. Teknik ini juga memeriksa validitas input dan memperbarui papan permainan setelah langkah pemain. Metode agentMove() memperbarui papan permainan dengan langkah yang diambil oleh agen RL. Metode checkForWin() memeriksa apakah pemain atau agen RL telah memenangkan permainan dengan melihat apakah ada empat simbol berturut-turut yang sama pada baris, kolom, dan diagonal papan. Metode checkForDraw() juga memeriksa apakah semua sel di papan telah terisi, yang menandakan bahwa permainan telah berakhir dengan hasil seri. Metode checkForEnd() menentukan apakah permainan telah selesai, baik dengan kemenangan atau seri, dan menampilkan hasil. Metode playGame() adalah inti dari permainan, yang mengatur giliran antara pemain dan agen RL. Permainan dimulai dengan langkah pemain atau agen, tergantung siapa yang dipilih untuk memulai. Untuk memperbaiki strategi permainannya secara bertahap, agen RL mengambil tindakan berdasarkan keadaan papan dan memperbarui nilai Q. Permainan tidak berhenti sampai salah satu tim menang atau sampai seri. Terakhir, siapa yang akan memulai permainan diputuskan dengan metode start(). Jika tidak ada instruktur, pemain akan diminta untuk mulai. Selain itu, ada dua fungsi tambahan: printBoard() menampilkan papan permainan, dan getStateKey() mengubah keadaan papan menjadi string yang dapat digunakan oleh agen RL untuk menentukan langkahnya.

```python
from abc import ABC, abstractmethod
import os
import pickle
import collections
import numpy as np
import random


class Learner(ABC):
    """
    Parent class for Q-learning and SARSA agents.
    """
    def __init__(self, alpha, gamma, eps, eps_decay=0.):
        # Agent parameters
        self.alpha = alpha
        self.gamma = gamma
        self.eps = eps
        self.eps_decay = eps_decay
        # Possible actions correspond to the set of all x,y coordinate pairs
        self.actions = [(i, j) for i in range(4) for j in range(4)]
        # Initialize Q values to 0 for all state-action pairs.
        self.Q = {action: collections.defaultdict(int) for action in self.actions}
        # Keep a list of reward received at each episode
        self.rewards = []

    def get_action(self, s):
        """
        Select an action given the current game state.

        Parameters
        ----------
        s : string
            state
        """
        # Only consider the allowed actions (empty board spaces)
        possible_actions = [a for a in self.actions if s[a[0] * 4 + a[1]] == '-']
        if random.random() < self.eps:
            # Random choose.
            action = random.choice(possible_actions)
        else:
            # Greedy choose.
            values = np.array([self.Q[a][s] for a in possible_actions])
            ix_max = np.where(values == np.max(values))[0]
            action = possible_actions[np.random.choice(ix_max)]

        # update epsilon; geometric decay
        self.eps *= (1. - self.eps_decay)

        return action
```

```python
    def save(self, path):
        """ Pickle the agent object instance to save the agent's state. """
        if os.path.isfile(path):
            os.remove(path)
        with open(path, 'wb') as f:
            pickle.dump(self, f)

    @abstractmethod
    def update(self, s, s_, a, a_, r):
        pass


class Qlearner(Learner):
    """
    A class to implement the Q-learning agent.
    """
    def __init__(self, alpha, gamma, eps, eps_decay=0.):
        super().__init__(alpha, gamma, eps, eps_decay)

    def update(self, s, s_, a, a_, r):
        """
        Perform the Q-Learning update of Q values.

        Parameters
        ----------
        s : string
            previous state
        s_ : string
            new state
        a : (i,j) tuple
            previous action
        a_ : (i,j) tuple
            new action. NOT used by Q-learner!
        r : int
            reward received after executing action "a" in state "s"
        """
        if s_ is not None:
            possible_actions = [action for action in self.actions if s_[action[0] * 4 + action[1]] == '-']
            Q_options = [self.Q[action][s_] for action in possible_actions]
            self.Q[a][s] += self.alpha * (r + self.gamma * max(Q_options) - self.Q[a][s])
        else:
            self.Q[a][s] += self.alpha * (r - self.Q[a][s])

        self.rewards.append(r)
```



```
class SARSAlearner(Learner):
    """
    A class to implement the SARSA agent.
    """
    def __init__(self, alpha, gamma, eps, eps_decay=0.):
        super().__init__(alpha, gamma, eps, eps_decay)

    def update(self, s, s_, a, a_, r):
        """
        Perform the SARSA update of Q values.

        Parameters
        ----------
        s : string
            previous state
        s_ : string
            new state
        a : (i,j) tuple
            previous action
        a_ : (i,j) tuple
            new action
        r : int
            reward received after executing action "a" in state "s"
        """
        if s_ is not None:
            self.Q[a][s] += self.alpha * (r + self.gamma * self.Q[a_][s_] - self.Q[a][s])
        else:
            self.Q[a][s] += self.alpha * (r - self.Q[a][s])

        self.rewards.append(r)

# You should include the Game class and necessary functions from previous responses for the complete functionality.
```

Gambar 3. agent.py

Kode di atas mendefinisikan dua jenis agen reinforcement learning yaitu Q-Learning dan SARSA, yang digunakan dalam permainan seperti Tic-Tac-Toe. Keduanya berasal dari kelas abstrak Learner, yang menetapkan kerangka dasar untuk agen pembelajaran. Parameter pembelajaran seperti alpha (*learning rate*), gamma (discount factor), dan eps (epsilon untuk eksplorasi dalam epsilon-greedy). Selain itu, agen ini memiliki daftar aksi yang dapat dilakukan, yaitu koordinat pasangan (i, j) di papan 4x4, dan tabel Q, yang berisi nilai Q untuk setiap pasangan state-action. Tabel Q dimulai dengan nilai 0 untuk semua state-action pairs. Strategi epsilon-greedy menggunakan metode get_action() untuk memilih aksi. Agen kadang-kadang memilih aksi acak (eksplorasi) atau memilih aksi terbaik berdasarkan nilai Q saat ini. Nilai eps dikurangi secara eksponensial setelah setiap pilihan tindakan untuk meningkatkan eksploitasi akhir pembelajaran. Nilai Q diperbarui oleh Kelas Qlearner menggunakan algoritma Q-Learning. Metode update() memperbarui nilai Q untuk setiap pasangan state-action berdasarkan reward yang diterima dan nilai maksimal dari aksi di state berikutnya. Nilai Q hanya diperbarui setelah permainan selesai, yang berarti tidak ada state baru lagi. Kelas SARSA learner menggunakan algoritma SARSA. Yang membedakan metode update() SARSA dari metode sebelumnya adalah bahwa pembaruan nilai Q menggunakan aksi aktual yang diambil di state berikutnya daripada nilai maksimal. Dibandingkan dengan Q-Learning, yang lebih "off-policy", SARSA lebih "on-policy". Kedua agen memiliki kemampuan untuk menyimpan keadaan pembelajaran mereka ke dalam file melalui metode save(), yang menggunakan modul pickle untuk serialisasi. Metode ini memungkinkan agen untuk melanjutkan pembelajaran dari keadaan yang telah disimpan. Selain itu, kode ini menyimpan reward yang diterima di setiap episode ke dalam daftar rewards. Daftar ini dapat digunakan untuk melacak kinerja agen secara berkala.