**TUGAS UAS**

**ADVANCE MACHINE LEARNING**

**S2-TE-36-R1**

(Pengganti UAS Frozen Lake)

Dirancang untuk memenuhi tugas mata kuliah kelompok Advance Machine Learning

di Program Studi S2 Teknik Elektro

Disusun Oleh:

Fajri Nurfauzan - 2101221040

Carolina Dwi Sundari - 2101221063



**FAKULTAS TEKNIK ELEKTRO**

**UNIVERSITAS TELKOM**

**BANDUNG**

**2023**

**Reinforcement Learning on FrozenLake and CartPole Environments**

**Author**: Fajri Nurfauzan  
**NIM**: 2101221040  
**Date**: June 13, 2023

[*View on GitHub*](https://github.com/fajrinurf/Reinforcment-Learning-Exercise/blob/main/Frozen-Lake-Coba.ipynb) |  [*Open in Colab*](https://colab.research.google.com/drive/1YzLWOCgxDmrqkCqwFs7yKSYNM87wf_bF?usp=sharing)

**Reinforcement Learning: Pengenalan**

Di dalam Tutorial ini, akan menjelaskan penggunaan Reinforcement Learning dasar yang akan digunakan, dan juga untuk memenuhi tugas kuliah Advance Machine Learning. Referensi yang akan digunakan di dalam tutorial ini akan berbasis dari buku dan juga paper. untuk kasus yang akan dijelaskan disini adalah penggunaan Reinforcement Learning yang akan menerapkan metode Q-learning dan SARSA untuk [*Frozen Lake*](https://gym.openai.com/envs/FrozenLake-v0/) klasik Puzzle.

Reinforcement Learning bisa beroperasi dengan melakukan indentifikasi pola yang akan digunakan secara optimal, di dalam konteks dari masalah masalah yang diberikan, sehingga agen pada reinforcement learning dapat membuat keputusan terbaik untuk langkah berikutnya.

**import** numpy **as** np *# Digunakan untuk operasi matematika dan manipulasi array.***import** matplotlib.pyplot **as** plt *# Digunakan untuk visualisasi data.***import** random *# Digunakan untuk menghasilkan bilangan acak.***import** gym *# Digunakan untuk mengakses environment OpenAI Gym.***from** tqdm.notebook **import** tqdm *# Digunakan untuk membuat progress bar.***import** os *# Digunakan untuk mengakses file dan direktori.***from** moviepy.editor **import** ImageSequenceClip *# Digunakan untuk membuat video dari gambar.***from** IPython.display **import** Image *# Digunakan untuk menampilkan gambar.***from** collections **import** defaultdict *# Digunakan untuk membuat dictionary dengan nilai default.*

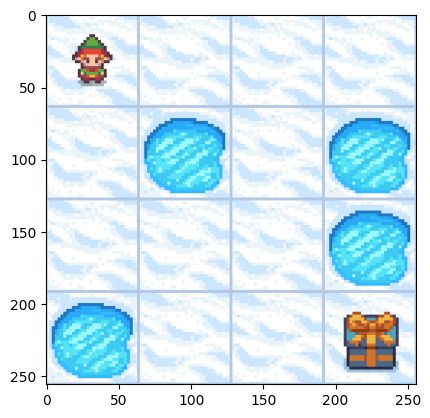
**Frozen Lake**

Frozen Lake environment adalah dunia kisi yang tidak pasti di mana seseorang memulai dari keadaan awal (kotak paling atas di sebelah kiri) untuk menuju ke keadaan akhir (kotak paling bawah di sebelah kanan). Environment yang tidak pasti di mana seseorang memulai dari keadaan yang berjalan di danau beku dengan ketebalan es yang bervariasi. Karena itu seseorang dapat jatuh ke air di kotak tertentu. Selain itu, esnya lebih licin di beberapa tempat, jadi mengambil langkah dapat membawa Anda lebih jauh dari yang diharapkan dan jika angin kencang.

Percobaan disini akan menggunakan SARSA, Q-Learning dan juga Monte Carlo

*# Membuat lingkungan FrozenLake dengan mode render "rgb\_array" dan kondisi licin (is\_slippery=True)*env = gym.make("FrozenLake-v1",render\_mode="rgb\_array",is\_slippery=True) *# Membuat lingkungan FrozenLake  
  
# Mereset lingkungan*env.reset() *# Menampilkan lingkungan*plt.imshow(env.render())

<matplotlib.image.AxesImage at 0x15cfe7aa0b0>



**Pengenalan Lingkungan Frozen Lake**

*# Mendapatkan jumlah state dalam lingkungan*numStates = env.observation\_space.nprint("there are ", numStates, "states\n") *# Menampilkan jumlah state*

there are 16 states

*# Mendapatkan jumlah aksi yang mungkin dalam lingkungan*numActions = env.action\_space.nprint("There Are ", numActions, "actions\n") *# Menampilkan jumlah aksi*print("0: LEFT \n1: DOWN \n2: RIGHT \n3: UP) \n") *# Menampilkan aksi yang mungkin*

There Are 4 actions0: LEFT 1: DOWN 2: RIGHT 3: UP)

*# Mendapatkan rentang reward dalam lingkungan*rewards=env.reward\_rangeprint("rewards:",rewards) *# Menampilkan rentang reward*print("Reach goal(G): +1 \nReach hole(H): 0\nReach frozen(F): 0\n") *# Menampilkan reward yang mungkin*

rewards: (0, 1)Reach goal(G): +1 Reach hole(H): 0Reach frozen(F): 0

*# Menampilkan matriks transisi*print("{action: P[s’], s’, r, done }") print(env.env.P[0]) *# Menampilkan matriks transisi untuk state 0*

{action: P[s’], s’, r, done }{0: [(0.3333333333333333, 0, 0.0, False), (0.3333333333333333, 0, 0.0, False), (0.3333333333333333, 4, 0.0, False)], 1: [(0.3333333333333333, 0, 0.0, False), (0.3333333333333333, 4, 0.0, False), (0.3333333333333333, 1, 0.0, False)], 2: [(0.3333333333333333, 4, 0.0, False), (0.3333333333333333, 1, 0.0, False), (0.3333333333333333, 0, 0.0, False)], 3: [(0.3333333333333333, 1, 0.0, False), (0.3333333333333333, 0, 0.0, False), (0.3333333333333333, 0, 0.0, False)]}

*# Fungsi untuk encode dan decode state. Dalam kasus ini, tidak melakukan apa-apa karena state sudah dalam bentuk yang bisa diproses.***def** encode(state): **return** state**def** decode(state): **return** state

**Random Strategy**

*# Strategi aksi yang diambil secara acak***def** strategie\_aleatoire(): **return**(random.choice([0,1,2,3]))

*# Fungsi untuk mengevaluasi strategi acak***def** evaluer\_aleatoire(nbr=1000): *# nbr: jumlah episode yang akan dievaluasi* nbr=0 *# Inisialisasi jumlah episode* obs\_i, \_ =env.reset() *# Mereset lingkungan* done=False *# Inisialisasi kondisi episode* frame=[] *# Inisialisasi frame*  **while** **not** done: *# Looping sampai episode selesai* action=random.choice([0,1,2,3]) *# Mengambil aksi secara acak* obs\_j, reward, done,\_,\_=env.step(action) *# Melakukan aksi dan mendapatkan hasilnya* frame.append(env.render()) *# Menambahkan frame* nbr=nbr+1 *# Menambahkan jumlah episode* print("Episode ended after {} iterations".format(nbr)) *# Menampilkan jumlah episode* print("Average Achieved Rewards:",reward) *# Menampilkan rata-rata reward* clip=ImageSequenceClip(frame, fps=10).resize(1.0) *# Membuat video dari frame* clip.write\_gif("FrozenLake\_Rate\_Randomly.gif", fps=10) *# Menyimpan video*

*# Menjalankan dan mengevaluasi strategi acak*evaluer\_aleatoire() *# Menampilkan hasil dalam bentuk GIF*Image(filename="FrozenLake\_Rate\_Randomly.gif")

Episode ended after 9 iterationsAverage Achieved Rewards: 0.0MoviePy - Building file FrozenLake\_Rate\_Randomly.gif with imageio.

A screenshot of a video game

Description automatically generated

**1) -greedy**

*# Fungsi untuk melakukan pemilihan aksi berdasarkan distribusi probabilitas***def** draw(q\_s): *# q\_s: distribusi probabilitas* i=np.random.choice(len(q\_s),1, p=q\_s)[0] *# Mengambil aksi berdasarkan distribusi probabilitas* **return** i *# Mengembalikan aksi*

*# Fungsi untuk melakukan pemilihan aksi berdasarkan strategi epsilon-greedy***def** epsGreedy(epsilon, q\_s): *# epsilon: probabilitas eksplorasi, q\_s: distribusi probabilitas* x = random.uniform(0,1) *# Mengambil bilangan acak antara 0 dan 1* **if** x<epsilon: *# Jika bilangan acak lebih kecil dari epsilon* **return**(draw(np.ones(len(q\_s))\*(1/len(q\_s)))) *# Mengambil aksi secara acak* **else**: argmax=q\_s==max(q\_s) *# Mengambil aksi dengan nilai q\_s terbesar* **return**(draw(np.array([1/sum(argmax) **if** argmax[k] **else** 0 **for** k **in** range(len(q\_s))]))) *# Mengambil aksi berdasarkan distribusi probabilitas*

**2) Test A Policy**

*# Fungsi untuk melakukan evaluasi policy***def** policy\_evaluation(q,gamma,nbr=1000,nombre\_iteration=False,gif=False): *# q: policy, gamma: discount factor, nbr: jumlah iterasi, nombre\_iteration: menampilkan waktu, gif: menampilkan lingkungan* s=0 *# inisialisasi return* **if** nombre\_iteration : k=0 *# inisialisasi waktu* **if** gif :frame=[] *# inisialisasi frame* **for** i **in** range(nbr): *# melakukan iterasi sebanyak nbr* obs\_i, \_ =env.reset() *# menginisialisasi lingkungan dan mengekstrak state awal* **if** (i==0) **and** gif :frame.append(env.render()) *# menampilkan lingkungan* done=False *# inisialisasi kondisi berhenti* G=0 *# inisialisasi return* h=0 *# inisialisasi waktu* **while** **not** done: *# melakukan iterasi sampai kondisi berhenti* obs\_j, reward, done,\_,\_=env.step(epsGreedy(0, q[encode(obs\_i)])) *# melakukan aksi berdasarkan strategi epsilon-greedy* **if** (i==0) **and** gif :frame.append(env.render()) *# menampilkan lingkungan* obs\_i=obs\_j *# mengupdate state* G=G+reward\*pow(gamma,h) *# mengupdate return* h=h+1 *# mengupdate waktu* **if** nombre\_iteration : k=k+h *# mengupdate waktu* s=s+G *# mengupdate return* s=s/nbr *# menghitung rata-rata return* **if** nombre\_iteration **and** gif : *# menampilkan lingkungan* k=k//nbr *# menghitung rata-rata waktu* **return** [s,k,frame] *# mengembalikan return, waktu, dan frame* **elif** nombre\_iteration: *# mengembalikan return dan waktu* k=k//nbr *# menghitung rata-rata waktu* **return**[s,k] *# mengembalikan return dan waktu* **else**: *# mengembalikan return* **return** s *# mengembalikan return*

**3) Visualization**

*# Fungsi untuk menampilkan tabel Q untuk FrozenLake 4x4 dan 8x8***def** plotQ(q\_table, map\_size): *# q\_table: tabel Q, map\_size: ukuran lingkungan* *# Membuat lingkungan* **if** (map\_size==4): *# Jika ukuran lingkungan 4x4* MAP = [  "SFFF", "FHFH", "FFFF", "HFFG" ] **else**: MAP=[ "SFFFFFFF", "FFFFFFFF", "FFFHFFFF", "FFFFFHFF", "FFFHFFFF", "FHHFFFHF", "FHFFHFHF", "FFFHFFFG" ] best\_value = np.max(q\_table, axis = 1).reshape((map\_size,map\_size)) *# Mengambil nilai Q terbesar* best\_policy = np.argmax(q\_table, axis = 1).reshape((map\_size,map\_size)) *# Mengambil aksi terbaik*  fig, ax = plt.subplots() *# Membuat figure* im = ax.imshow(best\_value) *# Membuat visualisasi tabel Q* **for** i **in** range(best\_value.shape[0]): *# Menampilkan panah* **for** j **in** range(best\_value.shape[1]): *# Menampilkan panah* **if** MAP[i][j] **in** 'GH': *# Jika state adalah goal atau hole* arrow = MAP[i][j] *# Menampilkan goal atau hole* **elif** best\_policy[i, j] == 0: *# Jika aksi terbaik adalah 0* arrow = '<' *# Menampilkan panah ke kiri* **elif** best\_policy[i, j] == 1: *# Jika aksi terbaik adalah 1* arrow = 'v' *# Menampilkan panah ke bawah* **elif** best\_policy[i, j] == 2: *# Jika aksi terbaik adalah 2* arrow = '>' *# Menampilkan panah ke kanan* **elif** best\_policy[i, j] == 3: *# Jika aksi terbaik adalah 3* arrow = '^' *# Menampilkan panah ke atas* **if** MAP[i][j] **in** 'S': *# Jika state adalah start* arrow = 'S ' + arrow *# Menampilkan start dan panah* text = ax.text(j, i, arrow, ha = "center", va = "center", color = "black") *# Menampilkan panah*  cbar = ax.figure.colorbar(im, ax = ax) *# Menampilkan colorbar*  fig.tight\_layout() *# Menampilkan figure* plt.show() *# Menampilkan figure* *# (Kode ini untuk menampilkan tabel Q dalam bentuk visual)*

*# Fungsi untuk menampilkan hasil evaluasi dan tabel Q***def** plot(q,evaluation,taille=4,plot\_Q=True,gamma=1): *# q: policy, evaluation: hasil evaluasi, taille: ukuran lingkungan, plot\_Q: menampilkan tabel Q, gamma: discount factor* *#plot evaluation* **if** type(evaluation[0])==np.ndarray: *# Jika hasil evaluasi adalah array* fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,7)) *# Membuat figure* ax1.plot([i \* 1000 **for** i **in** range(len(evaluation[:,0]))],evaluation[:,0],"r") *# Menampilkan hasil evaluasi* ax1.title.set\_text("average gain") *# Menampilkan judul* ax2.plot([i \* 1000 **for** i **in** range(len(evaluation[:,1]))],evaluation[:,1],"b") *# Menampilkan hasil evaluasi* ax2.title.set\_text("number of time steps") *# Menampilkan judul* **else**: plt.figure(figsize=(20,7)) *# Membuat figure* plt.plot([i \* 1000 **for** i **in** range(len(evaluation))], evaluation,"r") *# Menampilkan hasil evaluasi* plt.xlabel('episode') *# Menampilkan label sumbu x* plt.ylabel('average gain') *# Menampilkan label sumbu y* plt.show() *# Menampilkan figure* *#plotQ only for FrozenLake* **if** plot\_Q: plotQ(q,taille) *# Menampilkan tabel Q* s,k,frame=policy\_evaluation(q,gamma,nbr=1000,nombre\_iteration=True,gif=True) *# Menampilkan hasil evaluasi* print("Episode finished after {} average iterations".format(k)) *# Menampilkan hasil evaluasi* print("Average obtained rewards:",s) *# Menampilkan hasil evaluasi* **return**(frame) *# (Kode ini untuk menampilkan grafik evaluasi dan tabel Q)*

**4) Algorithm**

*A) SARSA*

*# Fungsi untuk melakukan algoritma SARSA***def** sarsa(gamma,nbr,Nombre\_etat,alpha=0.02,eps=0.2,nbr\_eval=1000,nbr\_iteration=False,eps\_variable=False,freq\_eval=1000): *# gamma: discount factor, nbr: jumlah episode, Nombre\_etat: jumlah state, alpha: learning rate, eps: nilai epsilon, nbr\_eval: jumlah evaluasi, nbr\_iteration: jumlah iterasi, eps\_variable: nilai epsilon berubah, freq\_eval: frekuensi evaluasi* *# Inisialisasi tabel Q* q = np.zeros((Nombre\_etat,env.action\_space.n)) *# Menginisialisasi tabel Q* evaluation=[] *# Menginisialisasi hasil evaluasi* epsilon=eps *# Menginisialisasi nilai epsilon*  **for** i **in** tqdm(range(nbr)): *# Melakukan iterasi sebanyak jumlah episode* *# Menginisialisasi state* obs\_i,\_ = env.reset() *# Menginisialisasi state* done=False *# Menginisialisasi kondisi episode selesai* *# Memilih aksi berdasarkan strategi epsilon-greedy* action=epsGreedy(epsilon,q[encode(obs\_i)])   *# Melakukan iterasi sampai episode selesai* **while** **not** done:  *# Melakukan aksi dan mendapatkan reward dan state baru* obs\_j, reward, done,\_,\_=env.step(action) *# Jika episode selesai, update tabel Q* **if** done: q[encode(obs\_i)][action]=q[encode(obs\_i)][action]+alpha\*(reward-q[encode(obs\_i)][action]) *# Update tabel Q* **if** i% freq\_eval==0: *# Evaluasi hasil setiap 1000 episode* evaluation.append(policy\_evaluation(q,gamma,nbr\_eval,nombre\_iteration=nbr\_iteration)) *# Evaluasi hasil* **if** eps\_variable: epsilon=epsilon\*0.99 *# Mengurangi nilai epsilon* **break** **else**: *# Jika episode belum selesai, update tabel Q dan pilih aksi baru* action\_=epsGreedy(epsilon,q[encode(obs\_j)]) *# Memilih aksi baru* q[encode(obs\_i)][action]=q[encode(obs\_i)][action]+alpha\*(reward+gamma\*q[encode(obs\_j)][action\_]-q[encode(obs\_i)][action]) *# Update tabel Q* obs\_i=obs\_j *# Mengupdate state* action=action\_ *# Mengupdate aksi* **return**(q,evaluation) *# Mengembalikan tabel Q dan hasil evaluasi*

*B) Q-Learning*

*# Fungsi untuk melakukan algoritma Q-Learning***def** Q\_learning(gamma,nbr,Nombre\_etat,alpha=0.02,eps=0.2,nbr\_eval=1000,nbr\_iteration=False,eps\_variable=False,freq\_eval=1000): *# gamma: discount factor, nbr: jumlah episode, Nombre\_etat: jumlah state, alpha: learning rate, eps: nilai epsilon, nbr\_eval: jumlah evaluasi, nbr\_iteration: jumlah iterasi, eps\_variable: nilai epsilon berubah, freq\_eval: frekuensi evaluasi* *# Inisialisasi tabel Q* q = np.zeros((Nombre\_etat,env.action\_space.n)) *# Menginisialisasi tabel Q* evaluation=[] *# Menginisialisasi hasil evaluasi* epsilon=eps *# Menginisialisasi nilai epsilon*  **for** i **in** tqdm(range(nbr)): *# Menginisialisasi state* obs\_i,\_ = env.reset() *# Menginisialisasi state* done=False *# Menginisialisasi kondisi episode selesai* *# Memilih aksi berdasarkan strategi epsilon-greedy* action=epsGreedy(epsilon,q[encode(obs\_i)])  *# Melakukan iterasi sampai episode selesai* **while** **not** done: *# Melakukan aksi dan mendapatkan reward dan state baru* obs\_j, reward, done,\_,\_=env.step(action) *# Jika episode selesai, update tabel Q* **if** done: q[encode(obs\_i)][action]=q[encode(obs\_i)][action]+alpha\*(reward-q[encode(obs\_i)][action]) *# Update tabel Q* **if** i% freq\_eval==0: *# Evaluasi hasil setiap 1000 episode* evaluation.append(policy\_evaluation(q,gamma,1000,nombre\_iteration=nbr\_iteration)) *# Evaluasi hasil* **if** eps\_variable: epsilon=epsilon\*0.99 *# Mengurangi nilai epsilon* **break** **else**: *# Jika episode belum selesai, update tabel Q dan pilih aksi baru* max\_action=epsGreedy(0,q[encode(obs\_j)]) *# Memilih aksi baru* q[encode(obs\_i)][action]=q[encode(obs\_i)][action]+alpha\*(reward+gamma\*q[encode(obs\_j)][max\_action]-q[encode(obs\_i)][action]) *# Update tabel Q* action\_=epsGreedy(epsilon,q[encode(obs\_j)]) *# Memilih aksi baru*  obs\_i=obs\_j *# Mengupdate state* action=action\_ *# Mengupdate aksi* **return**(q,evaluation) *# Mengembalikan tabel Q dan hasil evaluasi*

*C) Monte Carlo on Policy*

*# Fungsi untuk melakukan algoritma Monte Carlo On-Policy***def** onPolicyMC(gamma,nbr,Nombre\_etat,eps=0.2,nbr\_iteration=False,eps\_variable=False,freq\_eval=1000): *# gamma: discount factor, nbr: jumlah episode, Nombre\_etat: jumlah state, eps: nilai epsilon, nbr\_iteration: jumlah iterasi, eps\_variable: nilai epsilon berubah, freq\_eval: frekuensi evaluasi* *# Inisialisasi tabel Q* q = np.zeros((Nombre\_etat,env.action\_space.n)) *# Menginisialisasi tabel Q* count\_state\_action = defaultdict(**lambda**: np.zeros(env.action\_space.n)) *# Menghitung jumlah kunjungan ke setiap pasangan state-aksi* epsilon=eps *# Menginisialisasi nilai epsilon* evaluation=[] *# Menginisialisasi hasil evaluasi*  **for** i **in** tqdm(range(nbr)): *# Melakukan iterasi sebanyak jumlah episode* *# Simulasi episode* obs\_i, \_ =env.reset() *# Menginisialisasi state* episode=[] *# Menginisialisasi episode* done=False *# Menginisialisasi kondisi episode selesai* **while** **not** done: *# Memilih aksi berdasarkan strategi epsilon-greedy* action=epsGreedy(epsilon,q[encode(obs\_i)]) *# Melakukan aksi dan mendapatkan reward dan state baru* obs\_j, reward, done,\_,\_=env.step(action) *# Menyimpan hasil dalam episode* episode.append((encode(obs\_i),action,reward)) obs\_i=obs\_j *# Mengupdate state* **if** i%freq\_eval==0: *#* *# Evaluasi policy setelah setiap freq\_eval episode* evaluation.append(policy\_evaluation(q,gamma,1000,nombre\_iteration=nbr\_iteration)) *# Evaluasi hasil* **if** eps\_variable: epsilon=epsilon\*0.99 *# Mengurangi nilai epsilon* *# Menghitung return dan melakukan update tabel Q* G=0 *# Menginisialisasi return* **for** s,a,r **in** reversed(episode): *# Melakukan iterasi mundur* new\_s\_a\_count = count\_state\_action[s][a] + 1 *# Menghitung jumlah kunjungan ke pasangan state-aksi* G = r + gamma \* G *# Menghitung return* count\_state\_action[s][a] = new\_s\_a\_count *# Mengupdate jumlah kunjungan ke pasangan state-aksi* q[s][a] = q[s][a] + (G - q[s][a]) / new\_s\_a\_count *# Update tabel Q* **return** q,evaluation *# Mengembalikan tabel Q dan hasil evaluasi*

**FrozenLake 4x4**

*SARSA*

*# Melakukan algoritma SARSA dan menyimpan hasilnya dalam variabel q1 dan evaluation1*q1,evaluation1=sarsa(1,30000,numStates)

{"model\_id":"33976c5f99f142b3b5454a172c2cfd9d","version\_major":2,"version\_minor":0}

*# Membuat plot dari hasil evaluasi dan menyimpannya dalam variabel frame*frame=plot(q1,evaluation1,gamma=1)

A picture containing text, diagram, plot, line

Description automatically generated

A picture containing screenshot, colorfulness, text, rectangle

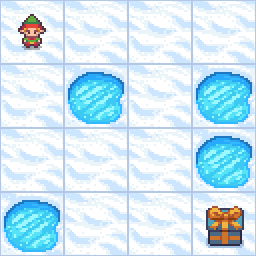
Description automatically generated

Episode finished after 37 average iterationsAverage obtained rewards: 0.646

*# Membuat GIF dari frame dan menyimpannya dengan nama "FrozenLake\_sarsa.gif"*clip=ImageSequenceClip(frame, fps=10).resize(1.0)clip.write\_gif("FrozenLake\_sarsa.gif", fps=10)

MoviePy - Building file FrozenLake\_sarsa.gif with imageio.

*# Menampilkan GIF yang telah dibuat*Image(filename="FrozenLake\_sarsa.gif")



*Q Learning*

*# Melakukan algoritma Q-Learning dan menyimpan hasilnya dalam variabel q2 dan evaluation2*q2,evaluation2=Q\_learning(1,30000,numStates)

{"model\_id":"9bec867f5be443e3a47ead2fb12ccefb","version\_major":2,"version\_minor":0}

*# Membuat plot dari hasil evaluasi dan menyimpannya dalam variabel frame*frame=plot(q2,evaluation2)

A red line graph on a white background

Description automatically generated with low confidence

A picture containing screenshot, colorfulness, rectangle, square

Description automatically generated

Episode finished after 48 average iterationsAverage obtained rewards: 0.84

*# Membuat GIF dari frame dan menyimpannya dengan nama "FrozenLake\_q\_learning.gif"*clip=ImageSequenceClip(frame, fps=10).resize(1.0)clip.write\_gif("FrozenLake\_q\_learning.gif", fps=10)

MoviePy - Building file FrozenLake\_q\_learning.gif with imageio.

*# Menampilkan GIF yang telah dibuat*Image(filename="FrozenLake\_q\_learning.gif")

A screenshot of a video game

Description automatically generated

*Monte Carlo on policy*

*# Melakukan algoritma Monte Carlo On-Policy dan menyimpan hasilnya dalam variabel q3 dan evaluation3*q3,evaluation3=onPolicyMC(1,30000,numStates)

{"model\_id":"881a069b5df547829a3bedf1e8199833","version\_major":2,"version\_minor":0}

*# Membuat plot dari hasil evaluasi dan menyimpannya dalam variabel frame*frame=plot(q3,evaluation3)

A red line on a white background

Description automatically generated with low confidence

A picture containing screenshot, colorfulness, text, rectangle

Description automatically generated

Episode finished after 43 average iterationsAverage obtained rewards: 0.774

*# Membuat GIF dari frame dan menyimpannya dengan nama "FrozenLake\_MC.gif"*clip=ImageSequenceClip(frame, fps=10).resize(1.0)clip.write\_gif("FrozenLake\_MC.gif", fps=10)

MoviePy - Building file FrozenLake\_MC.gif with imageio.

*# Menampilkan GIF yang telah dibuat*Image(filename="FrozenLake\_MC.gif")

A screenshot of a video game

Description automatically generated

**Frozen Lake 8X8**

*# Menutup lingkungan yang sebelumnya*env.close() *# Membuat lingkungan baru dengan FrozenLake 8x8*env = gym.make("FrozenLake8x8-v1",render\_mode="rgb\_array",is\_slippery=True)

*# Mendapatkan jumlah state*numStates = env.observation\_space.n

*SARSA*

*# Melakukan algoritma SARSA dan menyimpan hasilnya dalam variabel q1\_ dan evaluation1\_*q1\_,evaluation1\_=sarsa(1,30000,numStates)

{"model\_id":"5f503e488778482cb5842832ec1804e6","version\_major":2,"version\_minor":0}

*# Membuat plot dari hasil evaluasi dan menyimpannya dalam variabel frame*frame=plot(q1\_,evaluation1\_,8)

A picture containing diagram, line, text, plot

Description automatically generated

A picture containing screenshot, colorfulness, rectangle, square

Description automatically generated

Episode finished after 138 average iterationsAverage obtained rewards: 0.991

*# Membuat GIF dari frame dan menyimpannya dengan nama "FrozenLake\_sarsa8x8.gif"*clip=ImageSequenceClip(frame, fps=10).resize(1.0)clip.write\_gif("FrozenLake\_sarsa8x8.gif", fps=10)

MoviePy - Building file FrozenLake\_sarsa8x8.gif with imageio.

*# Menampilkan GIF yang telah dibuat*Image(filename="FrozenLake\_sarsa8x8.gif")



*Q Learning*

*# Melakukan algoritma Q-Learning dan menyimpan hasilnya dalam variabel q2\_ dan evaluation2\_*q2\_,evaluation2\_=Q\_learning(1,30000,numStates)

{"model\_id":"678396061bb04135ba603998282216c0","version\_major":2,"version\_minor":0}

*# Membuat plot dari hasil evaluasi dan menyimpannya dalam variabel frame*frame=plot(q2\_,evaluation2\_,8)

A red line on a white background

Description automatically generated with low confidence

A picture containing colorfulness, screenshot, square, pattern

Description automatically generated

Episode finished after 362 average iterationsAverage obtained rewards: 1.0

*# Membuat GIF dari frame dan menyimpannya dengan nama "FrozenLake\_q\_learning8x8.gif"*clip=ImageSequenceClip(frame, fps=10).resize(1.0)clip.write\_gif("FrozenLake\_q\_learning8x8.gif", fps=10)

MoviePy - Building file FrozenLake\_q\_learning8x8.gif with imageio.

*# Menampilkan GIF yang telah dibuat*Image(filename="FrozenLake\_q\_learning8x8.gif")



*Monte Carlo on Policy*

*# Melakukan algoritma Monte Carlo On-Policy dan menyimpan hasilnya dalam variabel q3\_ dan evaluation3\_*q3\_,evaluation3\_=onPolicyMC(1,30000,numStates)

{"model\_id":"469a0b6f86114a0d94adaaf008dca14c","version\_major":2,"version\_minor":0}

*# Membuat plot dari hasil evaluasi dan menyimpannya dalam variabel frame*frame=plot(q3\_,evaluation3\_,8)

A red line graph on a white background

Description automatically generated with low confidence

A picture containing screenshot, colorfulness, rectangle, square

Description automatically generated

Episode finished after 88 average iterationsAverage obtained rewards: 0.702

*# Membuat GIF dari frame dan menyimpannya dengan nama "FrozenLake\_MC.gif"*clip=ImageSequenceClip(frame, fps=10).resize(1.0)clip.write\_gif("FrozenLake\_MC.gif", fps=10)

MoviePy - Building file FrozenLake\_MC.gif with imageio.

*# Menampilkan GIF yang telah dibuat*Image(filename="FrozenLake\_MC.gif")



**Comparasion**

**Comparasion of Frozen Lake 4x4**

*# Membuat plot untuk membandingkan hasil evaluasi dari ketiga algoritma pada lingkungan 4x4*plt.figure(figsize=(15,5))plt.plot([i \* 1000 **for** i **in** range(len(evaluation1))],evaluation1) *# Membuat plot dari hasil evaluasi algoritma SARSA*plt.plot([i \* 1000 **for** i **in** range(len(evaluation2))],evaluation2) *# Membuat plot dari hasil evaluasi algoritma Q-Learning*plt.plot([i \* 1000 **for** i **in** range(len(evaluation3))],evaluation3) *# Membuat plot dari hasil evaluasi algoritma Monte Carlo On-Policy*plt.legend(["Sarsa","Q\_learning","MC\_on\_policy"])

<matplotlib.legend.Legend at 0x15c8a5ad360>

A picture containing text, plot, diagram, line

Description automatically generated

**Comparasion of Frozen Lake 8x8**

*# Membuat plot untuk membandingkan hasil evaluasi dari ketiga algoritma pada lingkungan 8x8*plt.figure(figsize=(15,5))plt.plot([i \* 1000 **for** i **in** range(len(evaluation1\_))],evaluation1\_) *# Membuat plot dari hasil evaluasi algoritma SARSA*plt.plot([i \* 1000 **for** i **in** range(len(evaluation2\_))],evaluation2\_) *# Membuat plot dari hasil evaluasi algoritma Q-Learning*plt.plot([i \* 1000 **for** i **in** range(len(evaluation3\_))],evaluation3\_) *# Membuat plot dari hasil evaluasi algoritma Monte Carlo On-Policy*plt.legend(["Sarsa","Q\_learning","MC\_on\_policy"])

<matplotlib.legend.Legend at 0x15c8a4f0520>

A picture containing text, map, diagram, line

Description automatically generated

**RESULTS FROZEN LAKE**

Didalam kode ini menggunakan tiga metode pembelajaran penguatan - SARSA, Q-Learning, dan Monte Carlo On-Policy - untuk menyelesaikan lingkungan FrozenLake dari OpenAI Gym.

Lingkungan ini adalah grid 4x4 atau 8x8 di mana agen harus mencapai tujuan tanpa jatuh ke dalam lubang. Setiap metode mencoba mempelajari kebijakan yang optimal untuk mencapai tujuan ini.

Hasilnya ditampilkan dalam bentuk grafik yang menunjukkan rata-rata penghargaan yang diperoleh selama pelatihan, dan juga dalam bentuk animasi yang menunjukkan bagaimana agen bergerak di lingkungan berdasarkan kebijakan yang dipelajari.

Dan hasilnya Reinforcement Learning berhasil mempelajari kebijakan yang cukup baik untuk menavigasi lingkungan dengan sukses, dan mendapatkan hasol seperti yang ada di gambar comparasion, yang dimana Q\_learning cukup bagus dibandingkan sarsa dan Monte Carlo, yang dimana dalam episode yang rendah avarage gainnya langsung meningkat

**Cart-pole in tabular**

env.close() *# Menutup lingkungan yang sebelumnya*env = gym.make("CartPole-v1",render\_mode= 'rgb\_array') *# Membuat lingkungan baru dengan CartPole*

env.reset() *# Menginisialisasi lingkungan*plt.imshow(env.render()) *# Menampilkan lingkungan*

<matplotlib.image.AxesImage at 0x15cff6bd510>

A picture containing text, screenshot, line, diagram

Description automatically generated

print("environment with ", env.action\_space.n, " actions") *# Menampilkan jumlah aksi yang dapat dilakukan*print("the space of states is coded with a class", env.observation\_space,  "which represents a continuous space") *# Menampilkan ruang state yang dapat ditempuh*print("the state is a vector of 4 values", env.observation\_space.sample()) *# Menampilkan contoh state*print("the lower bounds of the intervals are: ", env.observation\_space.low) *# Menampilkan batas bawah dari interval*print("the upper bounds of the intervals are: ",env.observation\_space.high) *# Menampilkan batas atas dari interval*env.reset() *# Menginisialisasi lingkungan*plt.imshow(env.render()) *# Menampilkan lingkungan*

environment with 2 actionsthe space of states is coded with a class Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32) which represents a continuous spacethe state is a vector of 4 values [ 3.5130734e+00 -3.2125574e+38 -1.5111980e-02 1.0101089e+38]the lower bounds of the intervals are: [-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]the upper bounds of the intervals are: [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]

<matplotlib.image.AxesImage at 0x15cff67a830>

A picture containing screenshot, text, line, diagram

Description automatically generated

nbr=0 *# Inisialisasi variabel untuk menghitung jumlah iterasi*done=False *# Inisialisasi variabel untuk menandakan apakah episode telah selesai***while** **not** done: *# Melakukan iterasi selama episode belum selesai* obs, reward, done,\_,\_ = env.step(np.random.randint(2)) *# Melakukan aksi secara acak* nbr+=1 *# Menambahkan jumlah iterasi*print("Episode ended after {} iterations".format(nbr)) *# Menampilkan jumlah iterasi yang telah dilakukan*env.close() *# Menutup lingkungan yang sebelumnya*

Episode ended after 15 iterations

state = env.reset() *# Menginisialisasi lingkungan*state *# Menampilkan state awal*

(array([-0.01349759, 0.04250308, 0.04161211, -0.0134709 ], dtype=float32), {})

*discretisation state*

*# Fungsi untuk mendiskritkan rentang nilai***def** discritiser\_range(xmin, xmax, num\_bins): *# Membuat array dengan titik-titik yang merata dalam interval yang diberikan* **return** np.linspace(xmin, xmax, num\_bins + 1)[1:-1]

*# Fungsi untuk mendiskritkan nilai***def** discritizer\_valeur(value, bins): *# Mengembalikan indeks bin yang berisi nilai yang diberikan* **return** np.digitize(x=value, bins=bins)

*# Fungsi untuk mengubah keadaan yang berkelanjutan menjadi keadaan yang diskrit***def** encode(state): L=[] **for** i, feature **in** enumerate(state): *# Mendiskritkan setiap fitur dan menambahkannya ke daftar* L.append(discritizer\_valeur(feature, state\_bins[i]) \* ((max\_bins + 1) \*\* i)) *# Mengembalikan jumlah dari semua fitur yang telah didiskritkan* **return**(sum(L))

*# Menentukan jumlah nilai diskrit yang bisa diambil oleh setiap fitur*Nval = 10  *# Mendiskritkan setiap fitur dalam ruang keadaan*state\_bins = [ discritiser\_range(-1, 1, Nval), *# posisi kereta* discritiser\_range(-2, 2, Nval), *# kecepatan kereta* discritiser\_range(-0.2095, 0.295, Nval), *# sudut tiang* discritiser\_range(-2, 2, Nval) *# kecepatan ujung tiang*] *# Menghitung jumlah maksimum bin di antara semua fitur*max\_bins = max(len(bin) **for** bin **in** state\_bins) *# Menghitung jumlah total keadaan diskrit*numStates= (max\_bins + 1) \*\* len(state\_bins) *# Menampilkan jumlah total keadaan diskrit*print("The number of states here will be ", numStates)

The number of states here will be 10000

*SARSA*

*# Mendefinisikan parameter untuk algoritma SARSA*gamma=0.9 nbr=15000Nombre\_etat=numStatesalpha=0.02eps=0.2nbr\_eval=1000nbr\_iteration=Trueeps\_variable=Falsefreq\_eval=1000

*# Melakukan pelatihan dengan algoritma SARSA dan menyimpan hasil evaluasi*q\_1\_,evaluation\_1\_= sarsa(gamma,nbr,Nombre\_etat,alpha,eps,nbr\_eval,nbr\_iteration,eps\_variable,freq\_eval)

{"model\_id":"57a77dcbaa57457c86faf4f2cca3cd3b","version\_major":2,"version\_minor":0}

*# Mengubah bentuk hasil evaluasi menjadi array 2D*evaluation\_1\_=np.array(evaluation\_1\_).reshape(-1,2)

*# Membuat plot dari hasil evaluasi dan menyimpannya sebagai GIF*frame=plot(q\_1\_,evaluation\_1\_,plot\_Q=False,gamma=0.9) *# Membuat plot dari hasil evaluasi*clip=ImageSequenceClip(frame, fps=10).resize(1.0) *# Mengubah plot menjadi GIF*clip.write\_gif("Cart-pole\_sarsa.gif", fps=10) *# Menyimpan GIF*Image(filename="Cart-pole\_sarsa.gif") *# Menampilkan GIF*

A picture containing line, plot, diagram, text

Description automatically generated

Episode finished after 134 average iterationsAverage obtained rewards: 9.994126954395771MoviePy - Building file Cart-pole\_sarsa.gif with imageio.

A picture containing tool

Description automatically generated

*Q Learning*

*# Mendefinisikan parameter untuk algoritma Q-Learning*gamma=0.9nbr=15000Nombre\_etat=numStatesalpha=0.02eps=0.2nbr\_eval=1000nbr\_iteration=Trueeps\_variable=Falsefreq\_eval=1000

*# Melakukan pelatihan dengan algoritma Q-Learning dan menyimpan hasil evaluasi*q\_2\_,evaluation\_2\_=Q\_learning(gamma,nbr,Nombre\_etat,alpha,eps,nbr\_eval,nbr\_iteration,eps\_variable)

{"model\_id":"e2c0d91ec2ca4baa9431c2028b332003","version\_major":2,"version\_minor":0}

*# Mengubah bentuk hasil evaluasi menjadi array 2D*evaluation\_2\_=np.array(evaluation\_2\_).reshape(-1,2)

*# Membuat plot dari hasil evaluasi dan menyimpannya sebagai GIF*frame=plot(q\_2\_,evaluation\_2\_,plot\_Q=False,gamma=0.9) *# Membuat plot dari hasil evaluasi*clip=ImageSequenceClip(frame, fps=10).resize(1.0) *# Mengubah plot menjadi GIF*clip.write\_gif("Cart-pole\_Q\_learning.gif", fps=10) *# Menyimpan GIF*Image(filename="Cart-pole\_Q\_learning.gif") *# Menampilkan GIF*

A picture containing text, line, plot, diagram

Description automatically generated

Episode finished after 158 average iterationsAverage obtained rewards: 9.999562167539588MoviePy - Building file Cart-pole\_Q\_learning.gif with imageio.

A picture containing tool

Description automatically generated

*Monte Carlo on Policy*

*# Mendefinisikan parameter untuk algoritma Monte Carlo On-Policy*gamma=0.9nbr=15000Nombre\_etat=numStateseps=0.2nbr\_eval=1000nbr\_iteration=Trueeps\_variable=Falsefreq\_eval=1000

*# Melakukan pelatihan dengan algoritma Monte Carlo On-Policy dan menyimpan hasil evaluasi*q\_3\_,evaluation\_3\_=onPolicyMC(gamma,nbr,Nombre\_etat,eps,nbr\_iteration,eps\_variable,freq\_eval)

{"model\_id":"bc6c5f7a09f44efdb98fcbed50ec33c4","version\_major":2,"version\_minor":0}

*# Mengubah bentuk hasil evaluasi menjadi array 2D*evaluation\_3\_=np.array(evaluation\_3\_).reshape(-1,2)

*# Membuat plot dari hasil evaluasi dan menyimpannya sebagai GIF*frame=plot(q\_3\_,evaluation\_3\_,plot\_Q=False,gamma=0.9) *# Membuat plot dari hasil evaluasi*clip=ImageSequenceClip(frame, fps=10).resize(1.0) *# Mengubah plot menjadi GIF*clip.write\_gif("Cart-pole\_MC.gif", fps=10) *# Menyimpan GIF*Image(filename="Cart-pole\_MC.gif") *# Menampilkan GIF*

A picture containing line, plot, diagram, text

Description automatically generated

Episode finished after 243 average iterationsAverage obtained rewards: 9.999999980253289MoviePy - Building file Cart-pole\_MC.gif with imageio.

A picture containing candle

Description automatically generated

**Comparasion**

*# Membuat figure dengan ukuran 15x5*plt.figure(figsize=(15,5)) *# Membuat plot untuk rata-rata gain dari setiap algoritma*plt.plot([i \* 1000 **for** i **in** range(len(evaluation\_1\_))],evaluation\_1\_[:,0]) *# Membuat plot untuk algoritma SARSA*plt.plot([i \* 1000 **for** i **in** range(len(evaluation\_2\_))],evaluation\_2\_[:,0]) *# Membuat plot untuk algoritma Q-Learning*plt.plot([i \* 1000 **for** i **in** range(len(evaluation\_3\_))],evaluation\_3\_[:,0]) *# Membuat plot untuk algoritma Monte Carlo On-Policy  
  
# Menambahkan legenda untuk plot*plt.legend(["Sarsa","Q\_learning","MC\_on\_policy"]) *# Menambahkan judul untuk plot*plt.title("average gain")

Text(0.5, 1.0, 'average gain')

A picture containing text, screenshot, plot, line

Description automatically generated

*# Membuat figure baru dengan ukuran 15x5*plt.figure(figsize=(15,5)) *# Membuat plot untuk rata-rata jumlah langkah dari setiap algoritma*plt.plot([i \* 1000 **for** i **in** range(len(evaluation\_1\_))],evaluation\_1\_[:,1]) *# Membuat plot untuk algoritma SARSA*plt.plot([i \* 1000 **for** i **in** range(len(evaluation\_2\_))],evaluation\_2\_[:,1]) *# Membuat plot untuk algoritma Q-Learning*plt.plot([i \* 1000 **for** i **in** range(len(evaluation\_3\_))],evaluation\_3\_[:,1]) *# Membuat plot untuk algoritma Monte Carlo On-Policy  
  
# Menambahkan legenda untuk plot*plt.legend(["Sarsa","Q\_learning","MC\_on\_policy"]) *# Menambahkan judul untuk plot*plt.title("average number of steps")

Text(0.5, 1.0, 'average number of steps')

A picture containing line, plot, diagram, text

Description automatically generated

**RESULTS CART POLE**

Sama seperti diatas penerapan kode ini menggunakan metode SARSA, Q-Learning, dan Monte Carlo On-Policy, tetapi kali ini untuk menyelesaikan lingkungan CartPole dari OpenAI Gym.

Dalam lingkungan ini, agen harus menjaga tiang tetap tegak dengan menggerakkan kereta ke kiri atau ke kanan. Karena ruang keadaan CartPole adalah kontinu, kode ini juga mencakup proses diskritisasi untuk mengubah ruang keadaan kontinu menjadi diskrit.

Hasilnya ditampilkan dalam bentuk grafik yang menunjukkan rata-rata penghargaan yang diperoleh selama pelatihan, dan juga dalam bentuk animasi yang menunjukkan bagaimana agen bergerak di lingkungan berdasarkan kebijakan yang dipelajari.

Dan hasilnya Reinforcement Learning berhasil mempelajari kebijakan yang cukup baik untuk menavigasi lingkungan dengan sukses, dan mendapatkan hasol seperti yang ada di gambar comparasion, yang dimana Monte Calro cukup bagus untuk rata rata stepsnya dibandingkan sarsa dan q laerning, yang dimana dalam episode yang rendah avarage gainnya langsung meningkat dan stabil

**RESULTS**

Dalam kedua kasus, metode Reinforcment Learning berhasil mempelajari kebijakan yang cukup baik untuk menavigasi lingkungan dengan sukses. Namun, efektivitas metode tertentu dapat bervariasi tergantung pada lingkungan dan parameter yang digunakan dalam pelatihan. dalam hal ini mungkin bisa melakukan berbagai percobaan lagi untuk mendapatkan hasil yang lebih baik lagi.

**REFERENCES:**

[1] [*https://ieeexplore.ieee.org/document/9558862/*](https://ieeexplore.ieee.org/document/9558862/)

[2] [*https://link.springer.com/book/10.1007/978-1-4842-5127-0*](https://link.springer.com/book/10.1007/978-1-4842-5127-0)