Unleashing the Power of Reinforcement Learning in 3D Object Environments

Abstract. This paper explores innovative approaches to overcome challenges associated with configuring and determining the shortest path in a 3D mesh. By integrating Expected SARSA with Q-learning and SARSA, we conducted a comprehensive comparison with Q-learning and Deep Q-learning. While conventional methods like the Dijkstra algorithm and the A* algorithm are frequently employed for this purpose, our emphasis is on a detailed analysis of reinforcement learning methods, specifically the off-policy Q-learning and the on-policy SARSA. The study involves an exhaustive examination of results obtained from both traditional algorithms and reinforcement learning approaches, providing valuable insights into their relative efficacy. The findings contribute to a broader understanding of pathfinding in 3D environments, with implications for diverse applications spanning robotics to fluid simulation.
Keywords: Shortest path, 3D mesh, Dijkstra algorithm, A* algorithm, Reinforcement learning.

1 Introduction

In recent years, the landscape of artificial intelligence and computer science has undergone a remarkable transformation, marked by a burgeoning interest in pivotal domains such as reinforcement learning, 3D objects, and the quest for the shortest path. These areas not only captivate the imagination of researchers but also hold immense promise for addressing intricate challenges and enhancing the capabilities of autonomous agents in both real and virtual environments.
This study embarks on a multifaceted journey with the overarching objective of delving deeper into the fundamental principles underpinning reinforcement learning, 3D object manipulation, and the search for the shortest path. Our aim is twofold: to advance the theoretical understanding of these domains and to uncover practical applications spanning diverse fields, including robotics, virtual reality, trajectory planning, and interactive simulations.
As part of our exploration, we scrutinize three distinct approaches to tackle the challenges posed by the convergence of reinforcement learning, 3D objects, and shortest path search:

Dijkstra Algorithm: An exhaustive search algorithm that meticulously explores all possible paths from the starting point to the destination. We implemented this algorithm within a 3D mesh, utilizing a graph structure to represent the mesh and compute the optimal path [2].

A* Algorithm: Building upon the foundation of Dijkstra, the A* algorithm enhances efficiency by incorporating a heuristic function. This function guides the search more efficiently towards a solution while ensuring optimality [3].

Reinforcement Learning: Leveraging the power of modern AI, we established a reinforcement learning environment to train an agent in learning optimal policies for pathfinding. Our implementation encompasses fundamental algorithms such as Q-learning, SARSA, and Expected SARSA, alongside cutting-edge Deep Q-learning.

This comprehensive examination not only contributes to the theoretical understanding of these domains but also sheds light on practical methodologies and techniques employed in the realm of artificial intelligence. By unraveling the intricacies of reinforcement learning, 3D object manipulation, and optimal path-finding, we aspire to pave the way for advancements that transcend theoretical boundaries and find application in the real-world challenges of today and tomorrow.

2 Literature review

Significant strides have been made in the field of 3D pathfinding; however, a critical examination of existing literature reveals persistent gaps, particularly the pressing need for algorithms adept at effectively managing the inherent complexity of 3D meshes. The diverse structures of meshes, coupled with the prevalence of intricate obstacles, demand a more sophisticated approach to ensure optimal outcomes.

A noteworthy gap in the literature lies in the dearth of comprehensive studies comparing the performance of classical algorithms and reinforcement learning methods within the specific context of 3D meshes. While individual investigations have explored these approaches independently, a holistic comparative analysis highlighting their respective strengths and weaknesses is presently limited. Such a comparison is crucial to inform the selection of appropriate approaches tailored to the specific requirements of diverse application scenarios.

The identification of these gaps provides promising avenues for future research. The development of a hybrid approach that integrates the strengths of classical algorithms like Dijkstra and A* with the adaptive capabilities of reinforcement learning emerges as an innovative path to overcome challenges inherent in 3D pathfinding in complex meshes. Such a hybrid approach could not only ensure efficiency in structured environments but also dynamically adapt to unforeseen situations.

Furthermore, particular attention should be given to designing robust evaluation mechanisms that enable a fair comparison of algorithm performances. Establishing standardized benchmarks for 3D pathfinding problems will facilitate comparisons among existing and emerging approaches.

In conclusion, this literature review lays the groundwork for our own research, which aims to address these gaps by proposing an innovative approach to pathfinding in 3D meshes. Integrating insights from existing literature, our study aspires to make a significant contribution to the advancement of practical and effective solutions for navigation in complex three-dimensional environments.

3. Methodology

In graph theory, the comparison between Dijkstra's and A* algorithms focuses on their execution time efficiency. Dijkstra's ensures optimality but may be computationally intensive, while A* introduces a heuristic for potentially faster pathfinding. This article aims to assess and compare their execution times, shedding light on the superior choice for various graph-related problem-solving scenarios.

Dijkstra's algorithm, a fundamental tool in graph theory, determines the shortest path between nodes within a given mesh or graph [5]. It systematically initializes distances, employing a priority queue to efficiently explore and update minimum distances from a specified source node to all others. Through iterative steps of vertex extraction, marking as visited, and recalculating distances for neighboring vertices, Dijkstra's algorithm progressively refines distance values, resulting in arrays of distances and predecessor vertices. An associated function, Shortest Path, utilizes this computed data to derive the optimal path between specified source and destination nodes.
The adaptation of the Dijkstra algorithm to a 3D environment relies on the addition of the vertical dimension (z) in the distance (1) calculations and the representation of nodes as three-dimensional coordinates:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \qquad (1)$$

The A* algorithm, a versatile graph search algorithm, is employed to identify the shortest path composed of nodes and edges [3].
In our context, the graph is defined by considering each face as a node, with the

edges on the mesh representing the connections between faces. This representation encapsulates the shortest path length, or 'price,' through a node 'n' in the A* algorithm, expressed as

$$F(n) = G(n) + H(n) \tag{2}$$

Here, G represents the length of the calculated shortest path from the starting node to 'n,' while H is an estimation of the shortest distance from 'n' to the target. The overall cost, F, is the sum of G and H.

The A* algorithm initiates its exploration from the source node, managing a candidate set that includes nodes awaiting selection [1]. At the outset, the source node is part of the candidate set. In each iteration, a node with the lowest F value is selected from the open set and transferred to the visited set. Following this, the neighboring nodes of the selected node are added to the candidate set. This iterative process continues until the destination node is chosen from the candidate set.
An illustrative representation of our algorithm is presented in Fig. 3 [1].

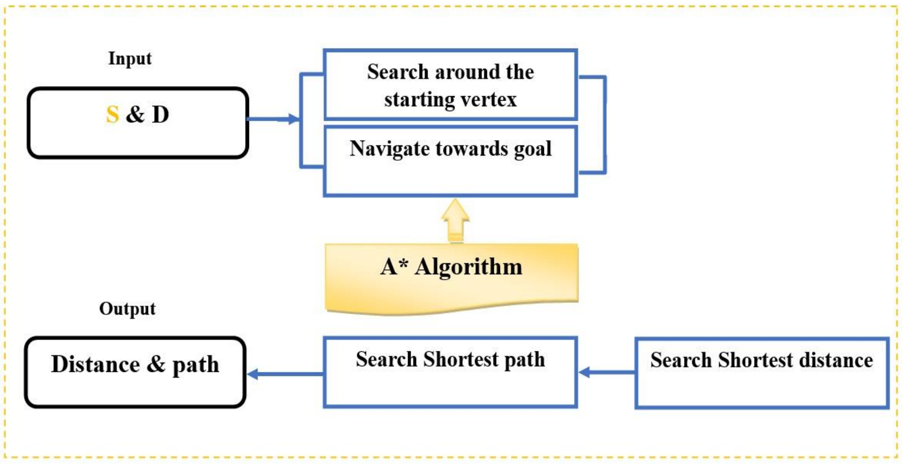


Fig. 3. Research composition diagram.

The difference between Dijkstra and A*
Fig. 4 shows a simplified representation of the A * algorithm compared with the Dijkstra algorithm.

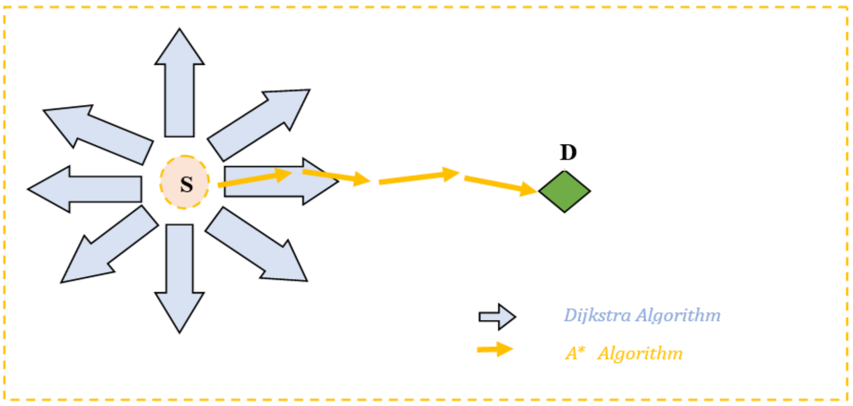


Fig. 4. Comparison of A* and Dijkstra Algorithms

The Dijkstra algorithm proceeds all the way from the starting vertex when searching for the shortest distance, but A * is a more efficient algorithm because it looks toward the target vertex like the yellow arrows in Fig. 4. [1]

4. Reinforcement Learning

Reinforcement learning is a machine learning technique in which an agent learns to make decisions by interacting with an environment and receiving rewards or penalties based on its actions [6]. This approach enables the agent to acquire decision-making skills through experience, as it takes actions and receives rewards or penalties that are inherently tied to its choices. Reinforcement learning stands out for its ability to solve complex problems through an iterative approach, wherein the agent gradually adjusts to optimize its performance throughout the learning process[7].
To enhance comprehension of the RL principle, Figure 6, titled 'Typical Reinforcement Learning Cycle', illustrates the key components and stages involved in a standard reinforcement learning scenario. This visual aid provides a graphical representation of the cyclical nature of the learning process, offering valuable insights into the sequential interactions between the agent and its environment [8].

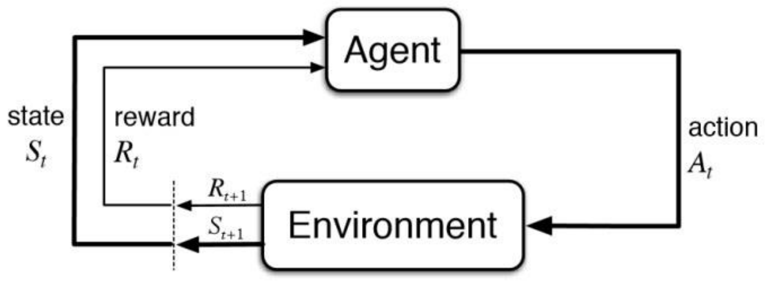


Fig. 6. Typical Reinforcement Learning cycle

Reinforcement learning necessitates the introduction of several fundamental concepts and metrics, including:

- Agent: A system or robot actively engaging and making decisions within the environment.
- Action (a): An individual action selected from a set of possible actions.
- State (s): A specific situation in which the agent currently exists.
- Policy (π): A strategic framework defining the behavioral strategy of the agent.
- Reward (r (s, a)): A positive or negative gain resulting from the execution of action 'a' in state 's.' The ultimate objective is to maximize the cumulative benefits of a given policy.
- Episode: Defined as a sequence of actions taken until the final state is reached or a predetermined duration elapses.
- Value function (V(s)): The value function of a state 's' represents the total expected rewards that an agent anticipates collecting from that state until the conclusion of the episode.
- Action-value function (Q (s, a)): The action-value function of action 'a' in state 's' signifies the total expected rewards derived from taking action 'a' in state 's' until the episode's conclusion [9].

2.1 Learning Algorithms

For a reinforcement learning model to succeed, the thoughtful selection of learning algorithms is paramount. These algorithms constitute the essence of the agent's adaptation process within its environment, determining how it makes decisions and adjusts its behavior based on received rewards and penalties.
The choice of these algorithms directly influences the model's ability to learn effectively from experience, optimize its performance, and iteratively solve complex problems. Hence, the success of a reinforcement learning model is closely tied to the relevance and efficiency of the underlying learning algorithms.

2.2 Q-learning.

Q-learning is a popular reinforcement learning algorithm used to solve Markov Decision Processes (MDP). It is a model-free approach, which means it does not require prior knowledge or understanding of the environment's dynamics[10].

Before learning begins, the Q function is arbitrarily initialized. Then, with each action choice, the agent observes the reward and the new state (which depends on the previous state and the current action). The core of the algorithm is an update to the value function.[11] The definition of the value function is updated at each step as follows:

$$Q [s, a] = (1-\alpha) Q [s, a] + \alpha(r+ \gamma \max Q [s',a']). \tag{4}$$

Here, s' represents the new state, s is the previous state, a is the chosen action, r is the received reward, α is the learning rate (ranging from 0 to 1), and γ is the discount factor. In essence, the Q-value, denoted as Q[s,a], undergoes an update through a weighted average. This update balances the previous Q-value (weighted by 1−α) with the expected gain, incorporating both the immediate reward (r) and the maximum Q-value of the new state-action pair (s', a') weighted by α [11].

Here is the pseudo-code for Q-learning [12]:

```
Initialize Q-table with zeros for all state-action
pairs
For each episode:
   Initialize the starting state
   Repeat until the episode is finished:
     Choose an action using an exploration
     strategy (e.g., epsilon-greedy)
     Take the chosen action, observe the reward, and
     the   new state Update the Q-value for the current
     state-action pair using the Q-learning equation:
     Q(s,a)=Q(s,a)+alpha*[reward+
     gamma*max(Q(new_state, all_actions))-Q(s, a)]
     Move to the new state
   End Repeat
End For
```

2.3 SARSA

The SARSA (State-Action-Reward-State-Action) algorithm, designed for sequential decision-making in agent-environment interactions, operates as an On-Policy algorithm, utilizing the policy being learned for value updates [13]. The key distinction between SARSA and Q-learning lies in their approaches to updating utility values (Q-values) during the learning process.

The SARSA update formula is as follows [7]:

$$Q\,[s, a] = (1 - \alpha)\, Q\,[s, a] + \alpha\, (r + \gamma\, Q\,[s', a']). \qquad (5)$$

Here, s ′ denotes the new state, s is the previous state, a is the chosen action, r is the agent's received reward, α is the learning rate, and γ is the discount factor.

In contrast, Q-learning updates Q-values using the current state-action pair (S, A) and the best possible action in the next state (S').This method estimates the maximum utility value of the next action, irrespective of the agent’s current policy [12].

The Q-learning update formula is as follows:

$$Q\,[\,s, a] = (1-\alpha)\, Q\,[\,s, a] + \alpha(\,r+ \gamma \max Q\,[s',a']). \qquad (6)$$

This distinction highlights the varied strategies employed by SARSA and Q-learning in updating their Q-values during the reinforcement learning process.

SARSA and Q-Learning techniques in Reinforcement Learning are algorithms that leverage Temporal Difference (TD) Update to enhance the agent’s behavior. Expected SARSA is an alternative for refining the agent’s policy, closely resembling SARSA and Q-Learning but differing in the action value function it employs.

2.4 Expected SARSA

The Expected SARSA (State-Action-Reward-State-Action) algorithm is a valuable reinforcement learning tool employed in decision-making within uncertain environments, functioning as an on-policy control method that updates its policy while adhering to it [14].

In Expected SARSA, the agent utilizes approximations of Q-values (expected rewards) for each action in a given state. These approximations guide the agent in determining its subsequent action in the next state. Here, the Q-value represents the anticipated cumulative reward the agent expects by taking a specific action in a specific state, then following its policy thereafter [15].

The primary distinction between SARSA and Expected SARSA arises in their Q-value estimation approaches. SARSA employs the Q-learning update rule, selecting the maximum Q-value of the next state-action pair. In contrast, Expected SARSA estimates the Q-value by calculating a weighted average of Q-values for all possible actions in the next state. These weights are determined by the probabilities associated with each action's selection in the next state, based on the current policy.
The Expected SARSA algorithm can be outlined through the following steps[15]:

```
1.Initialization:
   Set the initial Q-value estimates for each state-
action pair to a predetermined starting value.
2.Iteration:
   Repeat the following steps until convergence or a
predefined maximum number of iterations:
    a. Observe the current state.
```

b. Choose an action based on the current policy, utilizing the estimated Q-values for that specific state.

c. Observe the reward and the subsequent state.

d. Update the Q-value estimates for the current state-action pair using the Expected SARSA update rule.

e. Adjust the policy for the current state, considering the estimated Q-values.

3.Expected SARSA Update Rule:

Utilize the following formula for the Q-value update:

$Q(s, a) = (1-\alpha)Q(s,a) + \alpha\,[r + \gamma \sum \pi(a'|s')Q(s', a')]$.

where:

Q (s, a) represents the Q-value estimate for state s and action a, α is the learning rate, determining the impact of new information, r signifies the reward obtained for taking action a in state s and transitioning to the next state s', γ is the discount factor, influencing the importance of future rewards.

π(a'|s') is the probability of selecting action a' in state s', as per the current policy.

Q (s', a') is the estimated Q-value for the next state-action pair.

Expected SARSA proves to be a valuable algorithm for reinforcement learning, especially in scenarios where decision-making is contingent upon uncertain and dynamic environments. Its capability to estimate the expected reward for each action in the next state, while considering the prevailing policy, positions it as a valuable tool for tasks requiring real-time decision-making.

While SARSA is recognized as an on-policy technique and Q-learning as an off-policy technique, Expected SARSA exhibits a unique flexibility. Expected SARSA can be employed effectively in either an on-policy or off-policy manner, setting it apart as a more versatile option when compared to both SARSA and Q-learning [16].

Now, let's delve into a comparison of the action-value functions of these three algorithms to identify the distinctions inherent in Expected SARSA.

- SARSA:

$$Q[s, a] = (1 - \alpha)\, Q[s, a] + \alpha\, (r + \gamma\, Q[s', a']). \quad (7)$$

- Q-Learning:

$$Q[s, a] = (1-\alpha)\, Q[s, a] + \alpha(r+ \gamma \max Q[s',a']). \quad (8)$$

- Expected SARSA:

$$Q[s, a] = (1-\alpha)\, Q[s, a] + \alpha(r+ \gamma \textstyle\sum_a \pi(a/s')Q[s',a']). \quad (9)$$

It is important to highlight that Expected SARSA calculates a weighted sum of all possible next actions, taking into account the probability associated with each action. In situations where the Expected Return aligns with a greedy approach, simplifying the equation resembles Q-Learning. Conversely, when the Expected Return demonstrates on-policy characteristics, Expected SARSA computes the expected return for all actions, in contrast to the random action selection approach employed by SARSA.

2.5 Deep Q-learning

The integration of Deep Q-learning in the context of finding the shortest path in a 3D mesh environment represents a significant advancement. Leveraging deep neural networks, Deep Q-learning distinguishes itself from traditional RL algorithms such as standard Q-learning, SARSA, and Expected SARSA, providing a more efficient solution for handling the complex intricacies of a three-dimensional mesh. The capabilities of neural networks to capture non-linear relationships and generalize from diverse data are particularly beneficial in this context, enabling a more precise representation of the Q functions necessary for determining the optimal path. This approach proves to be better suited for addressing the specific challenge of finding the shortest path in a 3D mesh environment, opening new avenues for trajectory optimization in complex three-dimensional spaces [17].
Deep Q-Learning is an approach that extends classical Q-Learning by employing a deep neural network to represent the state-action value function (Q) [18].

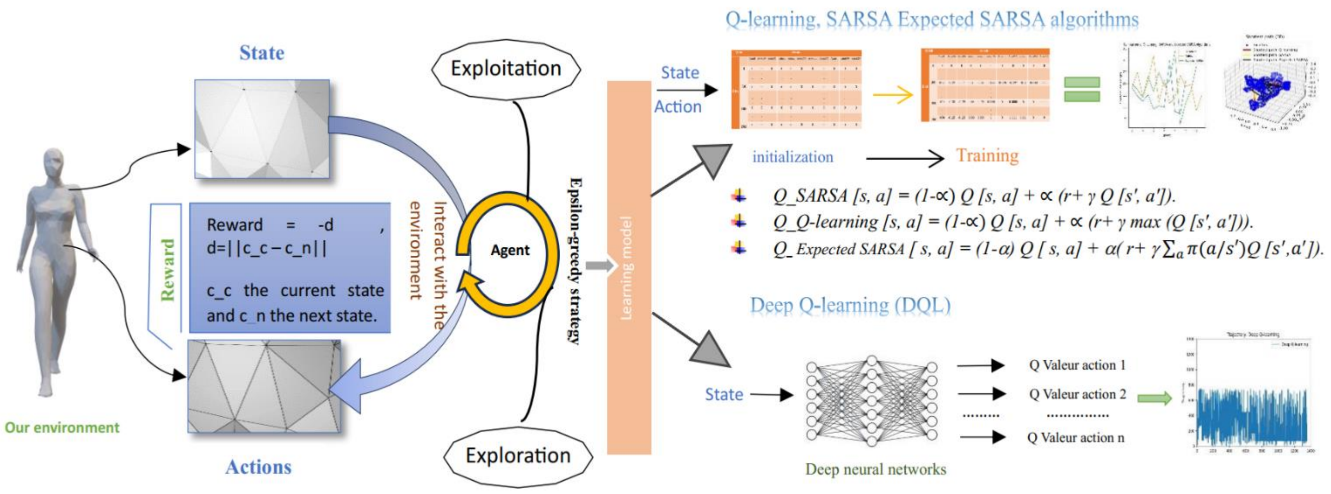

State
Exploitation
Reward = -d , d=||c_c – c_n||
c_c the current state and c_n the next state.
Reward
Interact with the environment
Agent
Epsilon-greedy strategy
Learning model
Our environment
Actions
Exploration
Q-learning, SARSA Expected SARSA algorithms
State
Action
initialization
Training
Q_SARSA [s, a] = (1-∝) Q [s, a] + ∝ (r+ γ Q [s', a']).
Q_Q-learning [s, a] = (1-∝) Q [s, a] + ∝ (r+ γ max (Q [s', a'])).
Q_ Expected SARSA [s, a] = (1-α) Q [s, a] + α(r+ γ∑a π(a/s')Q [s',a']).
Deep Q-learning (DQL)
State
Q Valeur action 1
Q Valeur action 2
Q Valeur action n
Deep neural networks

Fig. 12 The principle of reinforcement learning on our Mesh

Figure 12 illustrates the fundamental principles of reinforcement learning as applied to our custom Mesh environment. The visual representation encapsulates the core concepts and dynamics governing the reinforcement learning process within our experimental framework. Subsequent to this visual depiction, the forthcoming article delves into a comprehensive analysis of the experimental results, shedding light on the intricate interplay between reinforcement learning algorithms and the unique characteristics of our Mesh environment. Through detailed discussions and empirical evidence, the article aims to contribute valuable insights to the field of reinforcement learning in dynamic, custom environments.

5. Results and Discussion

In the context of our study aimed at finding the shortest distance and path between two faces on a polygon mesh, we employed a mesh model with 1500 faces, as depicted in Fig. 1. The polygon mesh serves as the most fundamental representation for describing objects in computer graphics. In 3D games, characters, and various 3D objects are portrayed through diverse polygon meshes. A polygon mesh is comprised of numerous vertices, where an edge connects two vertices, and a face is formed by three or more edges. Figure 1 illustrates our mesh model with 1500 faces, which was utilized in our analysis to determine the shortest distances and paths between specific faces.

Fig. 1. Mesh Model with 1500 Faces.

Figure 1 depicts a polygon mesh with 1500 faces and 752 vertices. A higher count of vertices and faces enables a more accurate representation of the original shape. However, it's important to note that as the number of vertices and faces increases, operations or calculations on the polygon mesh become more computationally expensive.

This paper delves into the methodological intricacies of determining the shortest distance and path between two designated vertices, denoted as S and D, within a polygon mesh as depicted in Fig. 2. The study leverages advanced algorithms, specifically the Dijkstra and A*, in pursuit of this objective [1].

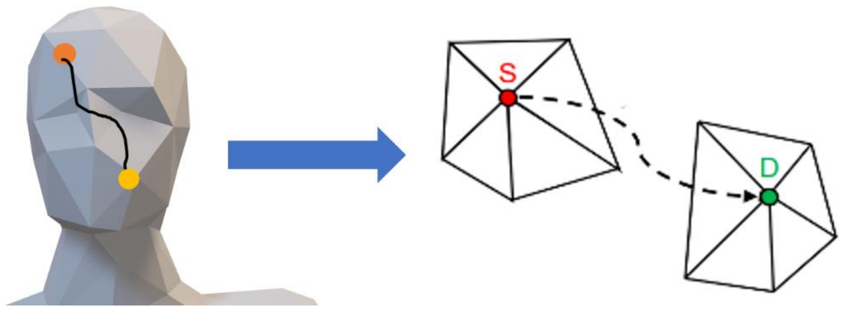


Fig. 2. Two points on polygon mesh.

Illustratively, in the context depicted in Fig. 2, two vertices are positioned on the surface of a polygon mesh body. The primary focus of our investigation is to ascertain the shortest path along the surface of the polygon mesh connecting these two designated points.

5.1 Results of the Dijkstra and A* algorithms

Upon implementing both algorithms within our environment (mesh), we achieved the following outcome.

Table 1. Comparison of distances and execution times between Dijkstra and A*.

The parameters	Dijkstra algorithm	A* algorithm
The number of faces	**25**	**25**
The distance traveled	**1.761972**	**1.635209**
Execution time /Second	**4.69**	**3.91**
The number of different facets	**8**	

Throughout our examination, we noted a subtle divergence in the quantity of facets and a shift in their characteristics along our course. This variance is particularly discernible in relation to the distance covered and the duration of execution.

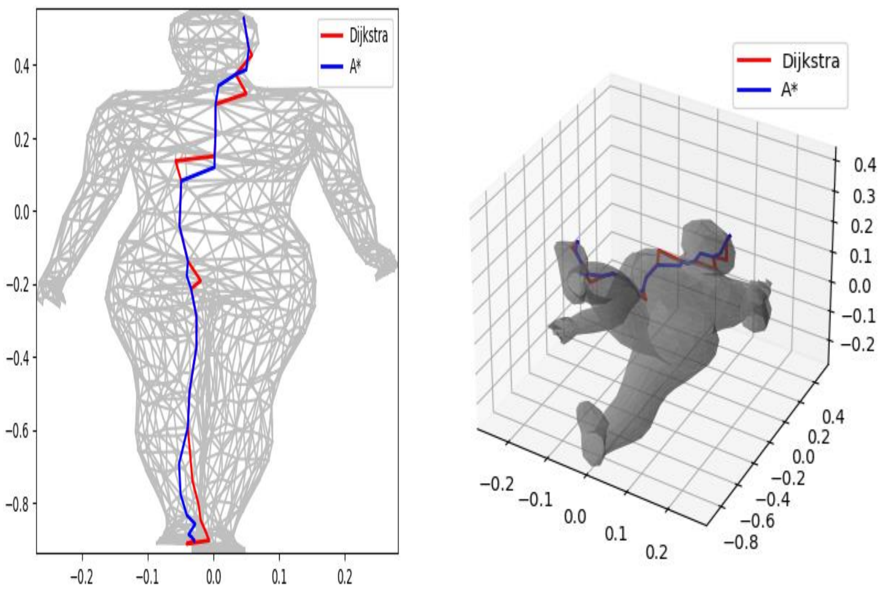


Fig. 5. The shortest path of the Dijkstra and A* algorithms.

In summary, the disparities between the Dijkstra and A* algorithms are distinctly observable in the graphical representation, primarily attributed to the employment of the heuristic function in the A* algorithm. This differentiation may lead to fluctuations in the paths obtained, influencing both distance and execution time

5.1 Reinforcement learning results and their algorithms

The foundation of reinforcement learning in our project encompasses the following elements:

- State space: This represents the current state of the agent within the environment, with the agent's position defined by the facets.
- Action space: This signifies the potential actions that the agent can undertake, involving movements towards accessible neighboring facets.

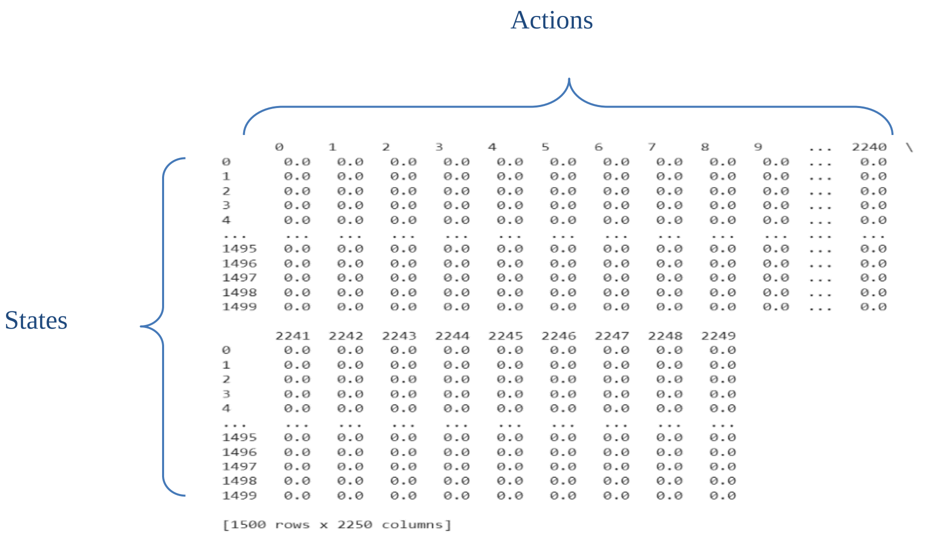
For effective learning, the agent must receive rewards or penalties based on its actions. In our specific case, rewards are determined by the distance between facets:

$$\text{Reward} = -d \quad , \quad d = \|c_c - c_n\|. \tag{3}$$

where c_c represents the current state and c_n denotes the next state.

Q-learning algorithm:

Implementation of the Q-learning algorithm involved applying the provided pseudo-code to our mesh. The action-state value function in a table was initialized to 0 for each action-state pair, with each cell being subsequently updated during the execution of the Q-learning algorithm.

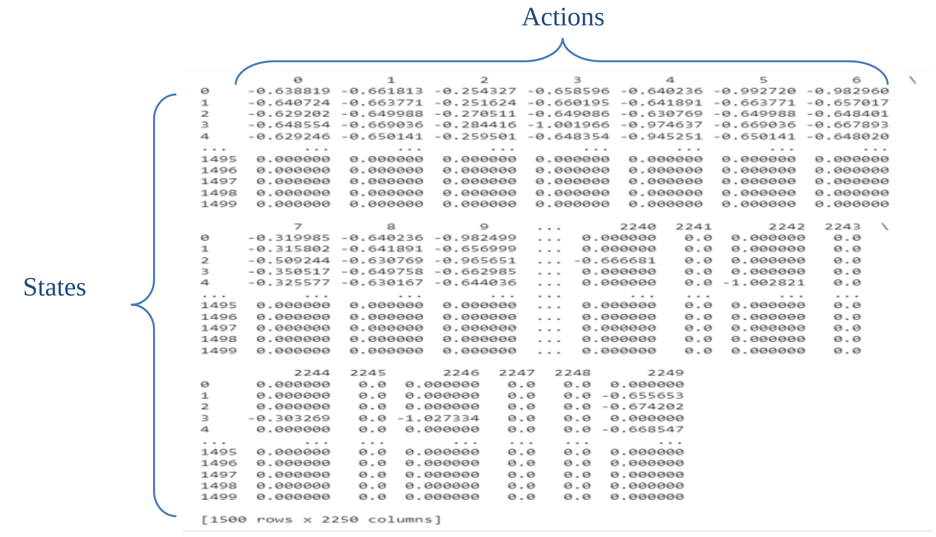

Actions
States

```
        0    1    2    3    4    5    6    7    8    9  ...  2240  \
0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
1     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
2     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
3     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
4     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...   ...
1495  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
1496  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
1497  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
1498  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0
1499  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0

      2241  2242  2243  2244  2245  2246  2247  2248  2249
0      0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1      0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
2      0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
3      0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
4      0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
...    ...   ...   ...   ...   ...   ...   ...   ...   ...
1495   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1496   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1497   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1498   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1499   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

[1500 rows x 2250 columns]
```

After the training phase, resulting in the following outcomes:


Actions
States

```
             0         1         2         3         4         5         6  \
0    -0.638819 -0.661813 -0.254327 -0.658596 -0.640236 -0.992720 -0.982960
1    -0.640724 -0.663771 -0.251624 -0.660195 -0.641891 -0.663771 -0.657017
2    -0.629202 -0.649988 -0.270511 -0.649086 -0.630769 -0.649988 -0.648401
3    -0.648554 -0.669036 -0.284416 -1.001966 -0.974637 -0.669036 -0.667893
4    -0.629246 -0.650141 -0.259501 -0.648354 -0.945251 -0.650141 -0.648020
...        ...       ...       ...       ...       ...       ...       ...
1495  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1496  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1497  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1498  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1499  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

             7         8         9  ...      2240  2241      2242  2243  \
0    -0.319985 -0.640236 -0.982499  ...  0.000000   0.0  0.000000   0.0
1    -0.315802 -0.641891 -0.656999  ...  0.000000   0.0  0.000000   0.0
2    -0.509244 -0.630769 -0.965651  ... -0.666681   0.0  0.000000   0.0
3    -0.350517 -0.649758 -0.662985  ...  0.000000   0.0  0.000000   0.0
4    -0.325577 -0.630167 -0.644036  ...  0.000000   0.0 -1.002821   0.0
...        ...       ...       ...  ...       ...   ...       ...   ...
1495  0.000000  0.000000  0.000000  ...  0.000000   0.0  0.000000   0.0
1496  0.000000  0.000000  0.000000  ...  0.000000   0.0  0.000000   0.0
1497  0.000000  0.000000  0.000000  ...  0.000000   0.0  0.000000   0.0
1498  0.000000  0.000000  0.000000  ...  0.000000   0.0  0.000000   0.0
1499  0.000000  0.000000  0.000000  ...  0.000000   0.0  0.000000   0.0

          2244  2245      2246  2247  2248      2249
0     0.000000   0.0  0.000000   0.0   0.0  0.000000
1     0.000000   0.0  0.000000   0.0   0.0 -0.655653
2     0.000000   0.0  0.000000   0.0   0.0 -0.674202
3    -0.303269   0.0 -1.027334   0.0   0.0  0.000000
4     0.000000   0.0  0.000000   0.0   0.0 -0.668547
...        ...   ...       ...   ...   ...       ...
1495  0.000000   0.0  0.000000   0.0   0.0  0.000000
1496  0.000000   0.0  0.000000   0.0   0.0  0.000000
1497  0.000000   0.0  0.000000   0.0   0.0  0.000000
1498  0.000000   0.0  0.000000   0.0   0.0  0.000000
1499  0.000000   0.0  0.000000   0.0   0.0  0.000000

[1500 rows x 2250 columns]
```

Fig. 7 A table of rewards for a pair (state, action).

Hence, the table includes reward values assigned to individual (state, action) pairs. Instances where values are 0.000000 represent zero rewards, and values set at 0.0 indicate no alteration associated with the corresponding state-action pair. By employing this table, we can ascertain the shortest path by leveraging the values it contains.

Comparison between Q-learning and SARSA in our mesh:

Similar to the Q-learning algorithm, we have also implemented the SARSA algorithm. The difference between these two algorithms lies in the update formula used. This distinction has an impact on the rewards obtained for each algorithm.

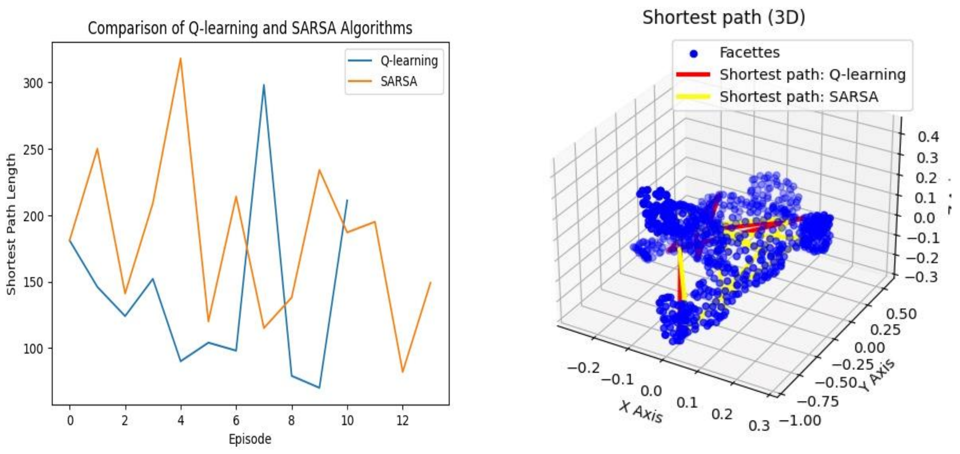


Fig. 9 Comparison between Q-learning and SARSA in our mesh.

In examining the graphical representations of Sarsa and Q-learning, it is evident that Sarsa demonstrates a more extensive interaction with the environment, engaging with a greater number of facets. In contrast, Q-learning exhibits a more restrained interaction, involving fewer facets compared to Sarsa. This observation suggests that Q-learning is primarily focused on exploiting acquired knowledge, while Sarsa is inclined towards exploring new actions within the context of its current policy.
This distinction underscores the contrasting learning strategies employed by the two algorithms, with Q-learning emphasizing exploitation and Sarsa prioritizing exploration in the reinforcement learning process.

Algorithmic Comparison in Custom Mesh Environment: A Practical Analysis:

Considering the theoretical foundations and mathematical formulations, let's conduct a comparative analysis of all three algorithms through a practical experiment. In this experiment, we will utilize our own mesh as the designated environment.

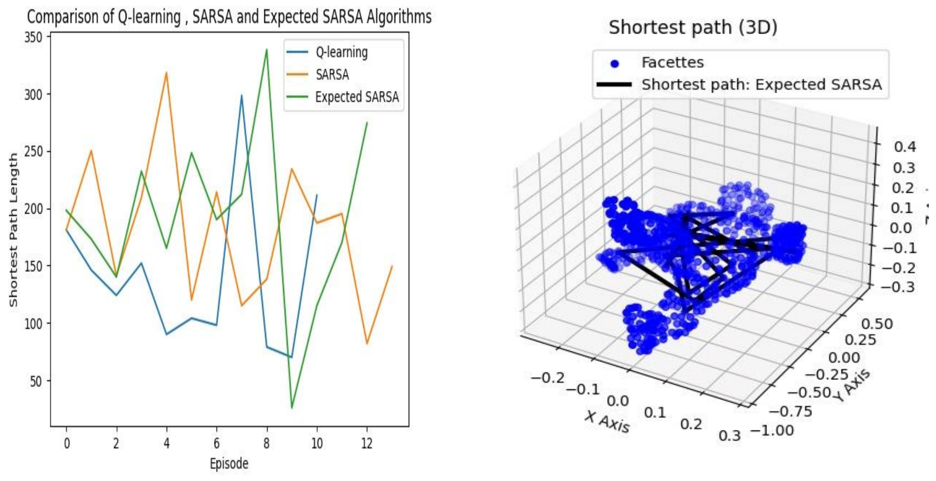


Fig.10 The impact of Expected SARSA on our mesh and its comparison with Q-learning and SARSA

The impact of the Expected SARSA algorithm warrants attention, despite yielding results somewhat closer to Q-learning and deviating slightly from the outcomes achieved by SARSA. This suggests that Expected SARSA strikes a balance between exploration and exploitation strategies, blending aspects of both Q-learning's emphasis on exploiting acquired knowledge and SARSA's inclination towards exploring new actions within the current policy framework. The nuanced performance of Expected SARSA highlights its adaptability and effectiveness in navigating the trade-off between exploration and exploitation, presenting a compelling alternative in the spectrum of reinforcement learning algorithms.

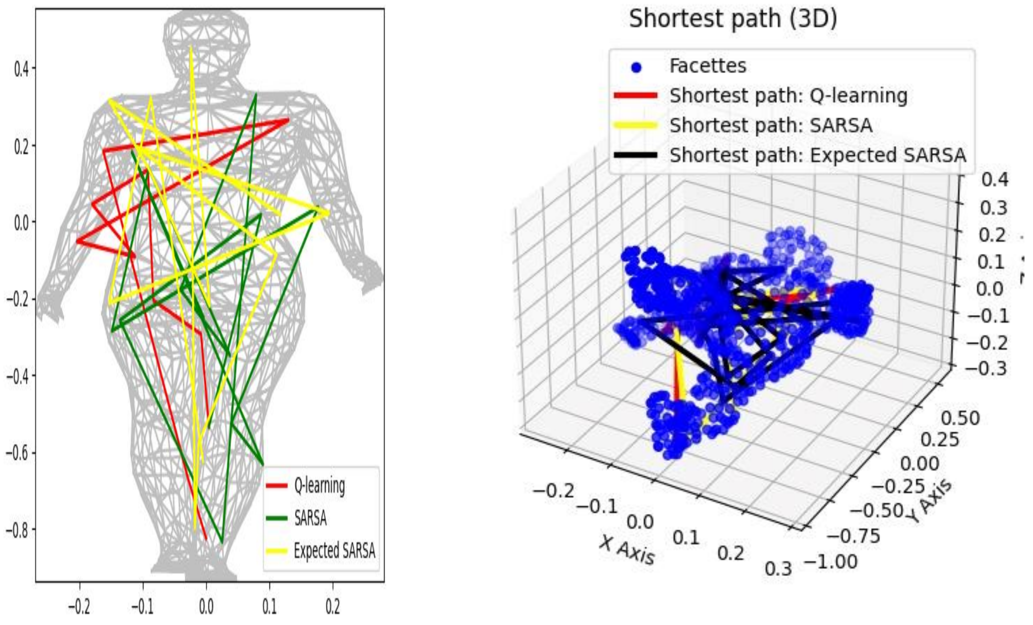


Fig. 11 Analyzing Shortest Paths: A Comparative Study of Expected SARSA, Q-learning, and SARSA on Our Mesh.

Investigating DQL Response During Model Training:

In the context of our study, during the model training phase with one episode, we observed the response of DQL within our environment:

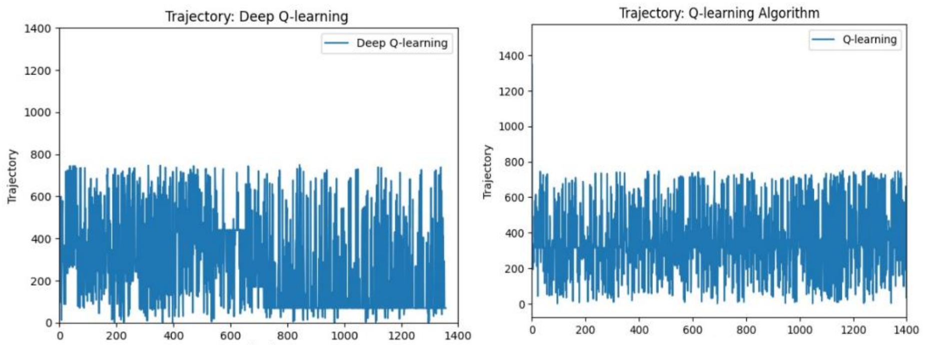


Fig. 12 The impact of Deep Q-learning on our mesh and its comparison with Q-learning algorithm

In our study, it is notable that DQL exhibits a more pronounced adaptation to the environment even with just a single episode, whereas the Q-learning results reflect the agent's interaction with the environment over 100 episodes. However, it is imperative to acknowledge a limitation concerning hardware constraints, as the DQL process is halted in this phase.

Conclusion

Navigating through 3D mesh environments to find the shortest path poses a formidable challenge, and our exploration into reinforcement learning (RL) methods has yielded noteworthy insights. The effectiveness of RL methodologies, becomes evident as our agent adeptly learns and refines its decision-making policies through iterative interactions with the environment.

In contrast to traditional shortest path algorithms like Dijkstra's or A*, which rely on a predefined understanding of the environment, RL methods excel in scenarios where the environment is not known in advance. This adaptability arises from the trial-and-error principle embedded in RL, enabling the agent to dynamically explore and learn optimal paths without prior knowledge. This flexibility is particularly advantageous in real-world applications where environments may be dynamic or unpredictable.

In conclusion, our study underscores the potential of RL, in revolutionizing problem-solving and highlights the need for a nuanced understanding of practical constraints in the pursuit of effective real-world implementation.

6 References

1. Xu, M.H., Liu, Y.Q., Huang, Q.L., Zhang, Y.X., Luan, G.F.: An improved Dijkstra's shortest path algorithm for sparse network. Applied Mathematics and Computation. 185, 247–254 (2007). https://doi.org/10.1016/j.amc.2006.06.094.
2. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In: Lerner, J., Wagner, D., and Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. pp. 117–139. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02094-0_7.
3. Anderson, T.R., Slotkin, T.A.: Maturation of the adrenal medulla--IV. Effects of morphine. Biochem Pharmacol. 24, 1469–1474 (1975). https://doi.org/10.1016/0006-2952(75)90020-9.
4. Choi, J., Zhang, B., Oh, K.: THE SHORTEST PATH FROM SHORTEST DISTANCE ON A POLYGON MESH. . Vol. (2005).
5. Frana, P.L., Misa, T.J.: An interview with Edsger W. Dijkstra. Commun. ACM. 53, 41–47 (2010). https://doi.org/10.1145/1787234.1787249.
6. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. jair. 4, 237–285 (1996). https://doi.org/10.1613/jair.301.
7. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource Management with Deep Reinforcement Learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks. pp. 50–56. ACM, Atlanta GA USA (2016). https://doi.org/10.1145/3005745.3005750.
8. Hu, B., Li, J., Yang, J., Bai, H., Li, S., Sun, Y., Yang, X.: Reinforcement Learning Approach to Design Practical Adaptive Control for a Small-Scale Intelligent Vehicle. Symmetry. 11, 1139 (2019). https://doi.org/10.3390/sym11091139.
9. Lee, D., Seo, H., Jung, M.W.: Neural Basis of Reinforcement Learning and Decision Making. Annu. Rev. Neurosci. 35, 287–308 (2012). https://doi.org/10.1146/annurev-neuro-062111-150512.

10. Bouzy, B.: Apprentissage par renforcement (3).

11. Poole, D.L., Mackworth, A.K.: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511794797.
12. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. (2013). https://doi.org/10.48550/ARXIV.1312.5602.
13. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. The MIT Press, Cambridge, Massachusetts (2018).
14. Van Seijen, H., Van Hasselt, H., Whiteson, S., Wiering, M.: A theoretical and empirical analysis of Expected Sarsa. In: 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. pp. 177–184. IEEE, Nashville, TN, USA (2009). https://doi.org/10.1109/ADPRL.2009.4927542.

15. Yang, L., Zheng, G., Zhang, Y., Zheng, Q., Li, P., Pan, G.: On Convergence of Gradient Expected Sarsa(λ). AAAI. 35, 10621–10629 (2021). https://doi.org/10.1609/aaai.v35i12.17270.
16. Ganger, M., Duryea, E., Hu, W.: Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning. JDAIP. 04, 159–176 (2016). https://doi.org/10.4236/jdaip.2016.44014.
17. symmetry-11-01139.pdf.
18. Zhou, S.K., Le, H.N., Luu, K., Nguyen, H.V., Ayache, N.: Deep reinforcement learning in medical imaging: A literature review, http://arxiv.org/abs/2103.05115, (2021).