

***Module: ECS7002P Artificial Intelligence in Games***

***Assignment 3: Frozen Lake***

Hussain Alkhorsan [170365952]

Cuong Phan [210255513]

Jayesh Betala [210712854]

1. The implementation for the frozen lake environment was organized into a modular structure. The base environment model was placed in a separate module and another model for the frozen lake environment was created which inherits from the base model class. Furthermore, model-based and model-free algorithms were also put into separate modules and a testing module was implemented to test the environment as well as debug potential bugs and incorrect implementations

The main module is configured to require a flag argument which determines the kind of lake the agent will play in. Depending on which flag is passed, the program passes the correct lake array. This allows the program to be more flexible and dynamic allowing both environments to be evaluated within a single module.

A proposed improvement is implemented within Sarsa and q-learning control implementations. The improvement relates to assigning the correct states to the step function when the AI steps on either a hole or the goal state. This was implemented to promote the improvement of the policy upon reaching goal by increasing the reward the goal state receives while keeping the rewards of going to holes at 0. Upon implementation, Sarsa and Q-Learning resulted in higher values across the board, by a factor of 10. Removing the implementation returns the expected value with both implementations returning optimal policies.

**Directory structure**

*Frozen Lake*

* *frozen\_lake.py -- implementation of frozen lake*
* *main.py -- main function (takes flag arguments)*
* *model.py -- base of the model*
* *model\_based.py -- model based algorithms*
* *model\_free.py -- model free algorithms*
* *p.npy -- numpy array of correct probability matrix*
* *play.py -- interactive testing*
* *test.py -- check implementation, debug*
* *timer.py – time test for model-based algorithms*

2. For the big frozen lake, policy iteration returned an optimal policy after 6 iterations at a speed of approximately 0.36 seconds. Value iteration managed to return an optimal policy after 19 iterations at a speed of approximately 0.22 seconds. This suggests the value iteration algorithm was slightly faster than policy iteration however it did require more iterations to return an optimal policy.

3. For the purpose of this analysis, the policy move of death states and goal state are not relevant. In this case, an optimal policy is given when all non-states reach the goal state. Sarsa control achieved an optimal policy after approximately 900 states while Q-Learning achieved an optimal policy after around 700 states. However, these evaluations vary wildly between runs with the common point being the lack of a policy for the corner state 3 with rewards given.

Evaluation of Sarsa control (w/ proposed improvements) returned an optimal policy after approximately 300 episodes. While Q-learning control (w/ proposed improvements) returned an optimal policy for all states after approximately 600 episodes. On average, Sarsa performs better than Q-learning, with the proposed improvements decreasing the episodes required for an optimal policy by 250 episodes on average.

4. In linear action-value function approximation, each element of the parameter vector θ is interpreted by the fixed points of associated Bellman operators. These Bellman equations are very important for dynamic programming and reinforcement learning, as they allow computing the value function. The estimate belongs to a hypothesis space H = {Qˆ θ|θ ∈ R p} which specifies the architecture of the approximation.

The tabular model-free reinforcement learning algorithms that we implemented are a special case of the non-tabular model-free reinforcement learning algorithms mainly because of how the value of Q is calculated. In tabular model-free reinforcement learning algorithms, the Q is calculated as a whole 16 (states) \* 4 (moves) matrix whereas the non-tabular model-free reinforcement learning algorithms calculate the best probability for each move.

5. After numerous hyperparameter changes, it was not possible to return an optimal policy for the big frozen lake for every state for Sarsa. This was regardless of maximum Epsilon and Theta to promote exploration, with an exceedingly large number of episodes. It was able to return a policy that defines a safe path towards the goal. However, the model was too conservative and was not able to explore much of the bottom left of the large lake as opposed to the top right of the lake.

This is because Sarsa is an on-policy algorithm which will follow the policy to compute the next state. This is not the case with Q learning which was able to return an optimal policy with much less states of around 20,000 states. This is because QL takes the maximum reward of the new state and ignores the current policy, enabling a more volatile but with more exploration.