Markov Decision Processes
Assignment 4
CS 7641
N. Ryan Karel (nkarel3)

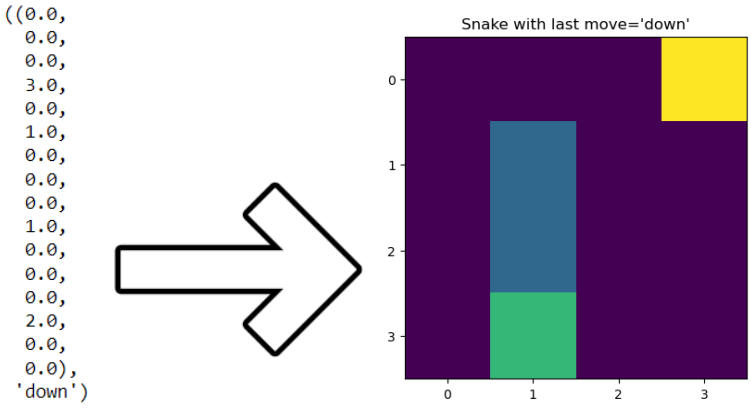
Introduction

The problems

The two problems that I focus on to explore solutions to Markov Decision Processes (MDPs) are single-player games. The first is an adaptation of Blackjack, or "twenty-one", a popular card game where the player tries to accumulate cards such that the point value is as close to without exceeding 21. The second is a form of the electronic Snake game (Wilson, 2003) where an increasingly long "snake" seeks out food on a 2-D grid without running into barriers or itself. Note that this Snake game is *not* a grid-world variant, or at least is not intended as such, as it has a substantially greater state space than a typical grid-world problem.

A state in Blackjack is represented fairly simply as a tuple of 3 quantities: the total point value of the player, the total observed point value for the dealer, and whether the player is still able to "hit" and receive more cards, or if they are in a "standing" position with no further actions available except to let the dealer systematically score the hands. Note that unlike the true game, aces always count as 11 points. This simplification reduces the state space and subsequent complexity of the problem dramatically. Another simplification is the exclusion of "splitting" and "doubling-down" actions for the player, who in this variant can only hit (receive another card) or hold (enter a standing position).

A state in the Snake game is represented by a 2-tuple showing the current state of the board as well as the action most recently taken by the player. A board is 2-dimensional grid with numbers 0, 1, 2, of 3 that show the body of the snake (represented by ones), the head of the snake (two), and the food that the head needs to come in contact with to grow the snake (three). See the example below for clarification, where the slight blue shows the snake body, green shows the snake head, and yellow shows the sought-after food. In terms of actions, the Snake player can only move in three directions at any one time. If the player has gone down most recently, they can go left, right, or keep going down, but they can't go up because they've just come from that direction and would immediately run into a portion of the tail of the snake.



The particular games I have written have a decent number of states between them, with Blackjack coming in at 1,271 states and Snake with 5,776 states. However, Blackjack has an effectively much smaller state space given that most states in its space have only a single action available, and in any case the policy can be represented by the recommended actions on about 100 interesting states, with the remaining portion not providing any interesting information.

Note that both of these are stochastic MDPs. For Blackjack, if the player decides to "hit", there is uncertainty around which card they will get. Note that like in the standard game, there is a 4/13 probability that the player will get a card worth 10 points. In the Snake game, the stochasticity comes from the fact that it is uncertain where the food will reappear after the snake comes in contact with it. Clearly, the Snake game has less uncertainty around it than the game of Blackjack. There is a policy that will always allow the player to win an instance of the Snake game, whereas Blackjack can be played optimally but not in a manner that will guarantee a win.

In terms of the reward structures, the Blackjack player receives no reward until a terminal stage is reached, whereby the player will learn whether it is has won the game, and receive a reward of value 1, or lost the game, at which point it will receive a "reward" of -1. In the game of Snake, to encourage a speedy play style, the Snake player receives a small negative reward each time it is not in a winning state, which it immediately enters once the Snake is of length 5. The disincentive for existing too long is very small in comparison to the losing terminal states, but still encourages speediness. Note that the disincentive is reduced in magnitude as the snake grows in length. I expect the gamma variable to play a bigger role in determining the appropriate policy for the Blackjack game than the Snake game, given the lack of "reward-guidance" until a terminal state is reached for that game.

Contrasting the problems

These two problems vary most greatly along two dimensions: uncertainty and state space size. Blackjack has a much smaller effective state space size, but the randomness of the result of individual actions is substantial. On the other hand, the Snake game has so many potential states, but most transition probabilities are 1 as a movement from one state to another is nearly always guaranteed by the model. The only uncertainty comes from when the food moves from one location to another after being "eaten" by the snake. The size of the Snake game also would imply that each game is played with far more actions overall than that of Blackjack.

I expect Q-learning to struggle with the Snake game in particular, given the size of the state space alone. It will take a lot of exploring to cover the whole space adequately.

The algorithms

We employ policy iteration, value iteration (Tokuç, 2022), and Q-learning (Mitchell, 1997) to develop a strategies of playing these games optimally. For policy and value iteration, we tune two hyperparameters, gamma and epsilon, which represent respectively the discount applied to future rewards and the level of stability to observe in the value (or Q) function before considering the algorithm converged. We define convergence in terms of the maximum change at any point in the value (or Q) function at a given iteration. If the most the function changes at any point is less than epsilon, than the algorithm terminates and we consider it converged. There are also fail-safes to stop the algorithm in case no solution has been found after 20 minutes of running.

For Q-learning, there are several levers we have at our disposal to influence how the algorithm tackles the problems. The rate of decay of alphas, initialization of Q-values for each state and action pairing, action selection strategy, and gamma and epsilon values are all hyperparameters we may modify to view subsequent impacts on performance.

Algorithm Performance

As can be expected, different algorithms and their various parameterizations took different amounts of time and sometimes converged to different policies. Let us look at each in turn to see which parameterizations performed best.

Overall

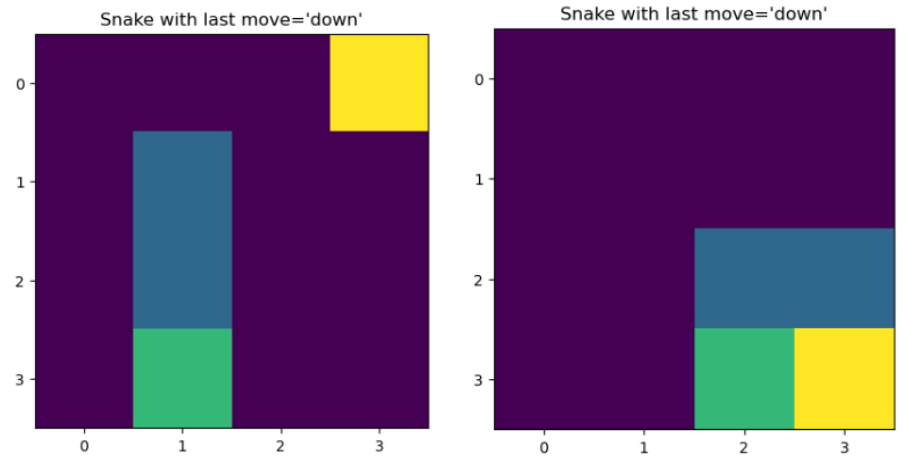
The majority of the algorithms converged to policies that passed a series of tests I wrote checking which actions the policies recommended. Of the ones that passed the accuracy test, the next most important quantity to me was how long the algorithms took to converge. As was expected, the algorithms operating on the Blackjack game generally took much less time than the corresponding algorithms for the Snake game.

	Accuracy	Total Time (s)
Blackjack-VI-gamma=0.25; epsilon=0.25	✓	0.33
Blackjack-VI-gamma=0.99; epsilon=0.25	✓	0.66
Blackjack-VI-gamma=0.25; epsilon=0.001	✓	0.66
Blackjack-PI-gamma=0.25; epsilon=0.25	✓	0.77
Blackjack-VI-gamma=0.99; epsilon=0.001	✓	1.09
Blackjack-PI-gamma=0.99; epsilon=0.25	✓	1.31
Blackjack-PI-gamma=0.25; epsilon=0.001	✓	1.44
Blackjack-PI-gamma=0.99; epsilon=0.001	✓	2.03
Snake-VI-gamma=0.25; epsilon=0.25	✓	13.86
Snake-VI-gamma=0.25; epsilon=0.001	✓	29.35
Snake-VI-gamma=0.99; epsilon=0.25	✓	56.34
Snake-VI-gamma=0.99; epsilon=0.001	✓	90.84
Snake-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=introduce-randomness; epsilon=0.1	✓	186.02
Snake-PI-gamma=0.25; epsilon=0.25	✓	219.51
Snake-PI-gamma=0.25; epsilon=0.001	✓	256.52
Snake-PI-gamma=0.99; epsilon=0.25	✓	284.64
Snake-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=q-optimal; epsilon=0.05	✓	517.87
Snake-PI-gamma=0.99; epsilon=0.001	✓	852.17
Snake-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=introduce-randomness; epsilon=0.01	✓	1200.38
Snake-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=introduce-randomness; epsilon=0.05	✓	1200.72
Blackjack-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=introduce-randomness; epsilon=0.01	✗	928.17
Blackjack-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=introduce-randomness; epsilon=0.1	✗	956.09
Blackjack-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=introduce-randomness; epsilon=0.05	✗	980.76
Blackjack-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=q-optimal; epsilon=0.05	✗	1040.89
Blackjack-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=q-optimal; epsilon=0.1	✗	1093.80
Blackjack-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=q-optimal; epsilon=0.01	✗	1098.28
Snake-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=q-optimal; epsilon=0.01	✗	1200.56
Snake-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=q-optimal; epsilon=0.1	✗	1205.02

Likely due to the uncertainty in the Blackjack problem and size of the Snake problem, the Q-learning algorithm struggled on both games, though it struggled more on the Blackjack problem. Accuracy was determined by a series of problems ranging in difficulty from easy to hard for each algorithm. When you look at which problems Q-learning struggled on, you can see that it generally was able to answer the easier ones correctly but struggled on the more difficult Blackjack problem, at times even struggling on a couple easy Snake states.

	Problem0	Problem1	Problem2	Problem3	Problem4
Snake-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=introduce-randomness; epsilon=0.01	✓	✓	✓	✓	✓
Snake-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=introduce-randomness; epsilon=0.05	✓	✓	✓	✓	✓
Snake-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=q-optimal; epsilon=0.01	✓	✗	✓	✓	✓
Snake-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=q-optimal; epsilon=0.1	✗	✗	✓	✓	✓
Snake-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=introduce-randomness; epsilon=0.1	✓	✓	✓	✓	✓
Snake-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=q-optimal; epsilon=0.05	✓	✓	✓	✓	✓
Blackjack-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=introduce-randomness; epsilon=0.01	✓	✓	✓	✓	✗
Blackjack-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=introduce-randomness; epsilon=0.05	✓	✓	✓	✓	✗
Blackjack-Q-learning-decay_pattern=mitchell; initialization=zeros; exploration=q-optimal; epsilon=0.01	✓	✓	✓	✓	✗
Blackjack-Q-learning-decay_pattern=mitchell; initialization=first_reward; exploration=q-optimal; epsilon=0.1	✓	✓	✓	✓	✗
Blackjack-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=introduce-randomness; epsilon=0.1	✗	✓	✓	✓	✗
Blackjack-Q-learning-decay_pattern=iteration_based; initialization=first_reward; exploration=q-optimal; epsilon=0.05	✓	✓	✓	✓	✗

Consider the examples below.



We would expect the policy to recommend action='right' We would expect the policy to recommend action='left'

That last one is the hard problem: there is an immediate benefit to seeking out the food to the right of the head (green square), but it must be foregone so that the snake is not forced into the wall or itself immediately after the intermediate benefit. Note that the last element of the snake's tail is not moved once it encounters a food item, hence the conclusion above.

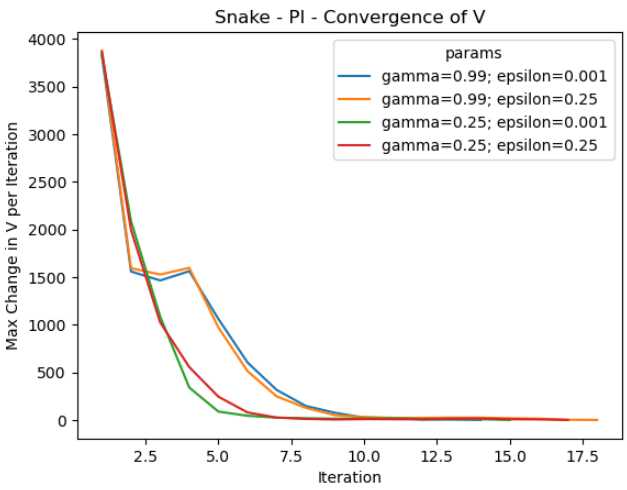
For Blackjack, the tests consisted of easy test cases like "hit when you have 7 and the dealer only shows 6" and a harder one like "hit when you have 11 but so does the dealer", the thought being that the dealer is too likely to not bust when it deals itself cards.

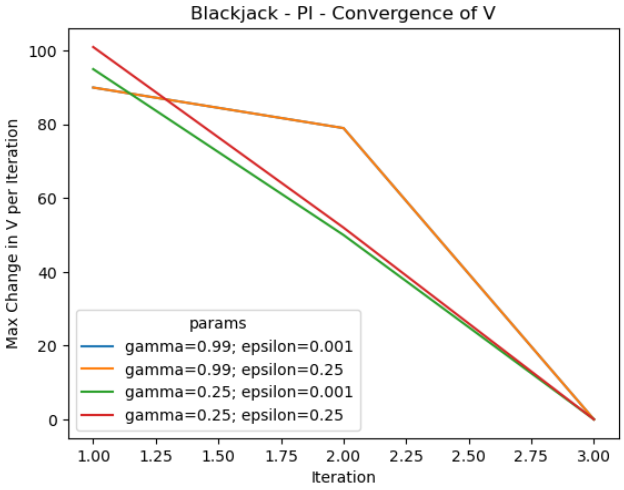
In terms of the average time and iterations required for each algorithm, Value Iteration generally outperformed the others on the time front, but iterations required was more balanced between PI and VI. Both PI and VI required far fewer iterations than Q-learning, likely due to the fact that they had access to the transition model directly and could easily perform arg-maxes and expectations over the possible transitions' values directly. Q-learning, on the other hand, had to explore to learn about the rewards and transition likelihoods, which slowed it down dramatically. It is also interesting to see that for the Snake problem and its larger state space, policy iteration took a decent bit longer to converge than value iteration. Perhaps because the number of transitions is a lot higher per episode there was more information to backpropagate for each iteration than seen in the Blackjack scenario, the policy evaluation solution step took much longer for the Snake problem.

Problem	Algorithm	Mean Time (s)	Mean required iterations
Blackjack	PI	1.39	3.00
Blackjack	Q-learning	1016.33	10000.00
Blackjack	VI	0.68	6.25
Snake	PI	403.21	16.00
Snake	Q-learning	918.43	306.33
Snake	VI	47.60	9.25

From the table, we can see that Q-learning on the Blackjack problem always used all 10,000 available iterations but did not truly converge.

Policy Iteration

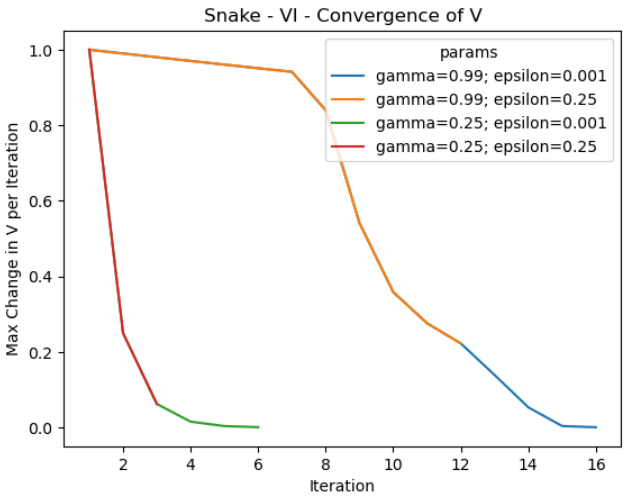


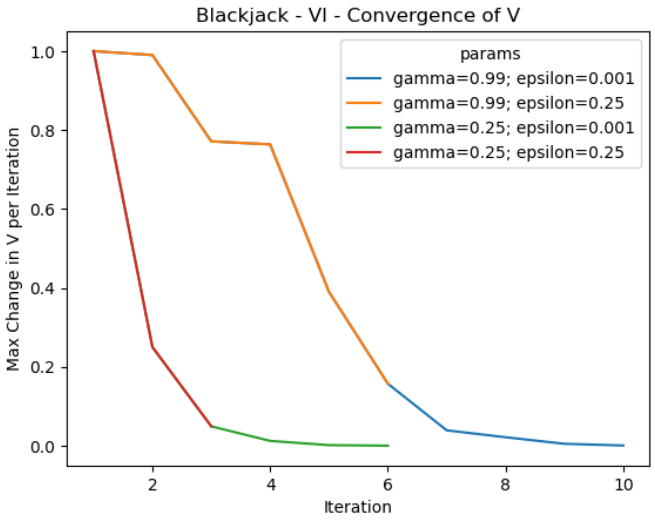


	Accuracy	Total Time (s)
Snake-PI-gamma=0.99; epsilon=0.001	✓	564.00
Snake-PI-gamma=0.99; epsilon=0.25	✓	212.92
Snake-PI-gamma=0.25; epsilon=0.001	✓	144.30
Snake-PI-gamma=0.25; epsilon=0.25	✓	139.80
Blackjack-PI-gamma=0.99; epsilon=0.001	✓	2.00
Blackjack-PI-gamma=0.99; epsilon=0.25	✓	1.16
Blackjack-PI-gamma=0.25; epsilon=0.001	✓	1.42
Blackjack-PI-gamma=0.25; epsilon=0.25	✓	0.78

The main difference we see for the Snake algorithm is that when gamma is lower, there was a steeper drop-off in the maximum change in value at any given state after the first couple iterations than for the higher gamma values. The smaller effective state space in Blackjack results in some fast convergence, both in terms of time and number of iterations, to the point where we don't get to observe a lot of interesting activity in the convergence plots.

Value Iteration





	Accuracy	Total Time (s)
Snake-VI-gamma=0.99; epsilon=0.001	✓	60.08
Snake-VI-gamma=0.99; epsilon=0.25	✓	45.02
Snake-VI-gamma=0.25; epsilon=0.001	✓	22.47
Snake-VI-gamma=0.25; epsilon=0.25	✓	11.24
Blackjack-VI-gamma=0.99; epsilon=0.001	✓	1.08
Blackjack-VI-gamma=0.99; epsilon=0.25	✓	0.65
Blackjack-VI-gamma=0.25; epsilon=0.001	✓	0.65
Blackjack-VI-gamma=0.25; epsilon=0.25	✓	0.30

When gamma was smaller for the Blackjack learner, the epsilon mattered less, probably because the higher discount actually shifted the observed strategy from the typical concept of optimal play in blackjack to a more conservative, bust-avoidance strategy. That was a interesting find, and it leads me to think that gamma should be less of a tuning parameter for the Blackjack application and more or a problem statement provision that should be fairly high to align it with my conception as to how Blackjack should be played.

Q-Learning

As we alluded to earlier, there are several hyperparameters that we can tune to facilitate the Q-learner fit.

Decay

I tested two different alpha-decay strategies. One followed directly from the Mitchell textbook (Mitchell, 1997) and set alpha equal to $(1 + visit_n(s, a))^{-1}$ so that the weight decreases as the state and action pairings are visited more and more often. The other strategy involves slowly decreasing alpha as the iteration count increases, regardless of the number of times a state and action pair is visited. It will likely not surprise the reader to learn that the first approach was more successful then the latter.

Initializations

There were two approaches that were considered when initializing the Q function. The first set all values to zero, and the other set all values equal to the state's immediate reward value, which risks converging to a local solution biased towards short-term wins instead of the longer-term global optimum.

Exploration

Another hyperparameter involved looking at how the actions would be selected as the learner explored the state space. There were two approaches: "introduce-randomness" and "q-optimal". The latter is a bit of a misnomer in that it uses the iteration's *current* understanding of the Q function to select seemingly optimal actions, which gives more purpose to the exploration but reduces tangential movements that might give a more full picture of the state space. On the other hand, the "introduce-randomness" approach uses the Q-optimal action half the time, and for the remaining half, it randomly selects a valid action.

Results

problem	decay	initialization	exploration	Time (s)	iterations
Blackjack	iteration_based	first_reward	introduce-randomness	956.1	10000
Blackjack	iteration_based	first_reward	q-optimal	1040.9	10000
Blackjack	mitchell	first_reward	introduce-randomness	980.8	10000
Blackjack	mitchell	first_reward	q-optimal	1093.8	10000
Blackjack	mitchell	zeros	introduce-randomness	928.2	10000
Blackjack	mitchell	zeros	q-optimal	1098.3	10000
Snake	iteration_based	first_reward	introduce-randomness	186.0	100
Snake	iteration_based	first_reward	q-optimal	517.9	100
Snake	mitchell	first_reward	introduce-randomness	1200.7	631
Snake	mitchell	first_reward	q-optimal	1205.0	216
Snake	mitchell	zeros	introduce-randomness	1200.4	585
Snake	mitchell	zeros	q-optimal	1200.6	206

Much to my surprise, the only theme that we can find in these results (combined with the "accuracy" measures shown above) shows that one helpful factor in both timely convergence as well as accurate predictions is to use the "iteration-based" decay methodology instead of the procedure more aware of how often each state has been visited. This alternative approach simply reduces alpha by a constant factor (in this case 0.99) each iteration. One theory as to why this might be could lie in the fact that states are not visited very often given the time and iteration constraints, and thus alpha does not decrease slowly enough using the process described in the Mitchell textbook. The iteration-based approach decreases alpha regardless of how often the state is visited, which maybe when combined with the "first-reward" initialization parameter makes for a winning combination.

Of course, these conclusions are only drawn from the Snake problem. For the Blackjack problem, I think there was too much entropy in the transition model for the Q-learner to have success. Instead, the algorithms that could make use of the transition model directly vastly outperformed the Q-learning approach.

Policy Depictions

Let us examine the outputs of our learners, which as I alluded to above vary as a function of gamma.

Blackjack

In the description of the problem I mentioned that we are really only interested in about 100 states when it comes to this particular formulation of Blackjack. Specifically, when gamma is low (0.25), we see the learner output a conservative strategy that eschews "hitting" in order to avoid "busting". Here is a table representing the policy for the most interesting 100 states that the player might find themselves in when gamma is equal to 0.25. Notice that if the player has anything over 11, this particular policy would say to never hit. Anything under 11, always hit (there's no chance of busting), and when you have 11 exactly, your response should change based on the observed value of the dealer's hand. If the dealer has something higher than 6, hit on 11 because it's too likely that the dealer will not bust. This makes sense as a win is more discounted with the lower gamma.

Player Value	11	12	13	14	15	16	17	18	19	20
Dealer Value										
2	hold	hold	hold	hold	hold	hold	hold	hold	hold	hold
3	hold	hold	hold	hold	hold	hold	hold	hold	hold	hold
4	hold	hold	hold	hold	hold	hold	hold	hold	hold	hold
5	hold	hold	hold	hold	hold	hold	hold	hold	hold	hold
6	hold	hold	hold	hold	hold	hold	hold	hold	hold	hold
7	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
8	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
9	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
10	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
11	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold

The recommendation changes noticeably when gamma is set to 0.99, and outputs a more reasonable set of recommendations.

Player Value	11	12	13	14	15	16	17	18	19	20
Dealer Value										
2	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
3	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
4	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
5	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
6	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold
7	hit	hit	hit	hit	hold	hold	hold	hold	hold	hold
8	hit	hit	hit	hold	hold	hold	hold	hold	hold	hold
9	hit	hit	hit	hold	hold	hold	hold	hold	hold	hold
10	hit	hit	hold	hold	hold	hold	hold	hold	hold	hold
11	hit	hold	hold	hold	hold	hold	hold	hold	hold	hold

When there are little-to-no discounts applied to winning the hand, the strategy becomes a little more bold as it evaluates the benefit of pushing the hand higher relative to the risk of busting. With this particular parameterization, we

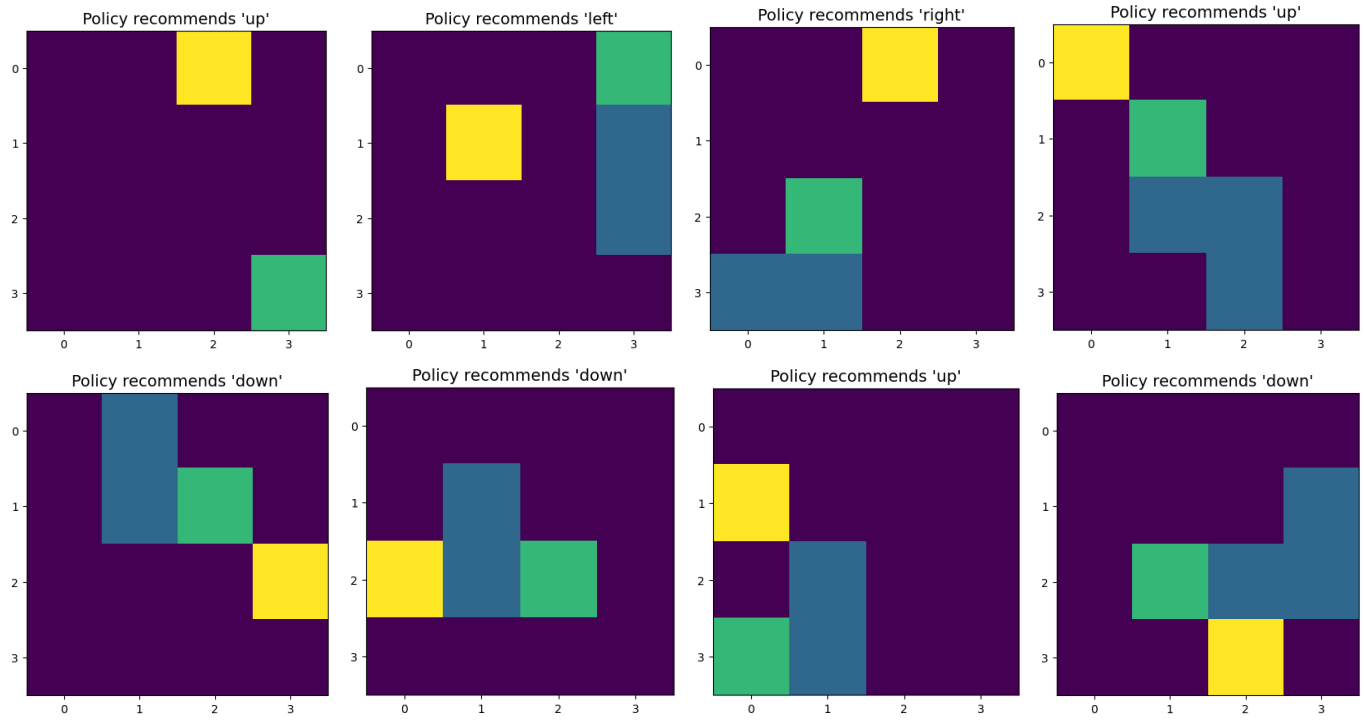
can also view the value function estimates of each state. The riskiest state that the policy still recommends a bet on is a 14 and 7 for the player and dealer respectively.

Player Value	11	12	13	14	15	16	17	18	19	20
Dealer Value										
2	0.22	-0.17	-0.17	-0.17	-0.17	-0.17	-0.17	0.07	0.31	0.54
3	0.24	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	0.09	0.33	0.55
4	0.26	-0.10	-0.10	-0.10	-0.10	-0.10	-0.10	0.12	0.35	0.56
5	0.28	-0.05	-0.05	-0.05	-0.05	-0.05	-0.05	0.15	0.36	0.57
6	0.29	-0.06	-0.06	-0.06	-0.06	-0.06	-0.06	0.20	0.40	0.60
7	0.18	-0.32	-0.36	-0.41	-0.42	-0.42	-0.42	0.28	0.54	0.69
8	0.12	-0.37	-0.41	-0.45	-0.45	-0.45	-0.45	-0.23	0.47	0.72
9	0.06	-0.43	-0.47	-0.49	-0.49	-0.49	-0.49	-0.28	-0.05	0.63
10	-0.01	-0.49	-0.53	-0.53	-0.53	-0.53	-0.53	-0.33	-0.12	0.10
11	0.03	-0.46	-0.46	-0.46	-0.46	-0.46	-0.46	-0.27	-0.07	0.13

Here we can evaluate the relative strength of each of the states. Aligning with intuition, we find that an 11 is a strong hand if the dealer has something less than 8 face-up, but becomes something of a coin toss when the dealer has 8 or higher.

Snake

Given the sheer number of possible states, it is difficult to entirely visualize the policy recommended for playing Snake. We can output random states and view the policy's recommendation on them, though. All of the recommendations shown below are reasonable to me. The last one looked a little tricky,

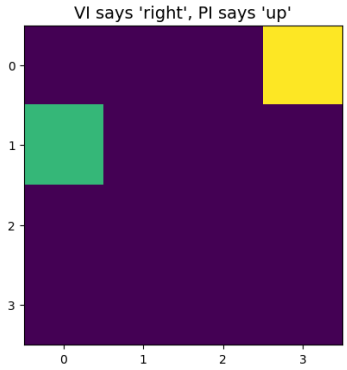


Policy Differences

For Blackjack, (as long as gamma is equal to 0.99), both policy iteration and value iteration converge to the same exact function over all the states. For Snake, there were 3 states that received a different recommendation from policy iteration vs. value iteration. These are shown below.

		VI	PI
(0.0, 0.0, 0.0, 3.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)	**up**	right	up
	right	right	up
(0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)	**up**	right	up
	right	right	up
(0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0)	**up**	right	up
	right	right	up

Let us examine one in detail. As you can see below, there is no substantive difference between the outcomes of the actions in terms of the ability of the snake head to reach the first food item. So although there technically are some differences between recommendations, in effect the two algorithms converge to the same policy.



References

Mitchell, T. (1997). *Machine Learning.* McGraw Hill.

Tokuç, A. A. (2022). *Value Iteration vs. Policy Iteration in Reinforcement Learning*. Retrieved from Baeldung: https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration

Wilson, R. D. (2003). High score!: the illustrated history of electronic games . McGraw-Hill Professional.