> *Exercise 1.1: Self-Play* Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?

With self play learning, the value function would include a value for every game state rather than just X moves. Let us assume that the value represents the value for the X player. In that case the policy that the O player should follow would be to select the move that leads to the lowest value state transition rather than the highest for the X player. If we use temporal difference learning, sampling moves from this policy, then eventually it will converge to the value function for the minimax solution in which both sides exhibit optimal play. The exploration parameter in the training process must be large enough that the entire state space is explored, otherwise there will not be accurate value estimates for certain states. This policy would never lose to a suboptimal opponent, but there might be cases where it considers a number of candidate moves identical because they all should lead to a draw. The policy trained against an imperfect opponent might favor a particular move that should not be different under perfect play but has a chance of leading to a win only against that opponent. For example, in perfect play no starting move is preferable over another because any starting move results in a draw. But against an opponent that plays randomly, certain moves like in the corners lead to more game states with winning paths if the opponent makes mistakes.

> *Exercise 1.2: Symmetries* Many tic-tac-toe positions appear different but are really the same because of symmetries. How might we amend the learning process described above to take advantage of this? In what ways would this change improve the learning process? Now think again. Suppose the opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have the same value?

The value function table can be modified so that symmetrically equivalent positions are mapped to the same value rather than storing a separate value for each position. For evaluating candidate moves that result in symmetrically equivalent positions we might pick randomly if they all contain the highest value function estimate. If we play an opponent that does not treat equivalent positions the same, then we should not consider the symmetries for our value function because the behavior of the opponent will cause them to not be equivalent regarding our ability to win the game.

> *Exercise 1.3: Greedy Play* Suppose the reinforcement learning player was greedy, that is, it always played the move that brought it to the position that it rated the best. Might it learn to play better, or worse, than a nongreedy player? What problems might occur?

When we begin the learning process, the value function is an inaccurate estimation that is continually updated. The first move or set of moves that lead to a winning outcome will be followed by the greedy policy unless in future sampling they in fact lead to losing outcomes. In either case the greedy policy will converge to playing the first set of moves that on average are marginally better than the average outcome. This policy is not necessarily the optimal one and would not have accurate value function estimates for states that are never explored. A non-greedy player will explore some additional moves that may not have a higher value estimate at the moment. However after the value function is updated there is a greater chance of finding a move that has a higher value that what was previously considered optimal. The greedy player could also get stuck in a bad policy because it has inaccurate value estimates for the other states. Unless a player is allowed to explore the entire state space, it is not guaranteed to find the optimal policy.

> *Exercise 1.4: Learning from Exploration* Suppose learning updates occurred after all moves, including exploratory moves. If the step-size parameter is appropriately reduced over time (but not the tendency to explore), then the state values would converge to a different set of probabilities. What (conceptually) are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?

The two sets of probabilities represent two different value functions. The original method that does not update after exploritory moves represents the value estimates for the greedy optimal policy; that is the policy that only considers moves that maximize the value estimate of the state transition. If we also update from exploritory moves, then we are calculating the value estimate for policy that takes exploratory moves with some probability. Then the policy optimization is occuring under the constraint that sometimes random moves are made, so the algorithm will find the best policy under that restriction. If we call the probability of making random exploratory moves ϵ, then this type of optimal policy can be called the ϵ-greedy policy since it makes greedy moves $1 - \epsilon$ percent of the time. The formal definition of this type of policy appears in subsequent chapters. Since this policy is restricted, it will result in a worse final policy with fewer wins than the policy that ignores exploratory moves for learning updates.. Also, if we plan to eventually follow the greedy policy with respect to the value function when we deploy the agent, the value function will no longer accurately represent the policy being followed as it will have failed to converge to the optimal greedy policy

(converging instead to the optimal ϵ-greedy policy. If, however, we consider an agent that continues to make exploratory moves after being deployed, then updating after exploratory moves will lead to a more accurate value function for that agent's policy. It should lead to more wins than the original method because it accounts for a certain degree of random moves in the future.

> *Exercise 1.5: Other Improvements* Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the tic-tac-toe problem as posed?

Since the game is very simple, rather than estimating the value function and updating it with one forward step perhaps we could see what happens to the value function multiple steps into the future considering every possible response from our opponent. That way we could explore more of the action space even if the opponent does not actually play those moves. Even if we ignore all the value function estimation techniques, we can do this type of exhaustive search to just evaluate every starting position and subsequent positions to see which states lead to wins, losses, and draws. This approach would simply simulate all the game states and have a policy based on every possible outcome from each state. For a game this simple that approach is possible, but for games with many more states such an exhaustive search is intractable. Also having a different reward assigned to draws and losses would help an agent distinguish between the two outcomes. It may not increase the win rate, but it could avoid losses in cases where a draw is possible.