(<https://ec.europa.eu/jrc/communities/sites/jrccties/files/andersjonsson-2018.pdf>)

Evaluation Criteria

* Theoretical Analysis
  + Performance Bounds – How far from optimal is an AI algorithm?
  + Time complexity – How fast is it?
  + Memory Complexity – How much memory is consumed?

Empirical Performance measures

* Measure winning?
* Scoring points?

Practices in reinforcement learning

* Run X trials, report average of 3 best runs

<https://ai.stackexchange.com/questions/5570/game-ai-evaluation-function-and-making-progress-towards-winning>

Case 1: Minimax / Alpha-Beta / other similar "exhaustive" searches

When using Minimax / Alpha-Beta / other search algorithms based on those, the easiest solution to the problem you describe is to use iterative deepening. As soon as you prove a win for yourself at a certain depth level d using iterative deepening, you can simply stop the search, don't search if there are any other wins to be proven at depth d + 1, just play along the line you've just proven to be a winning line. This way, you will always go for the win in the lowest number of moves.

## Case 2: Monte-Carlo Tree Search / other searches with randomness

Monte-Carlo Tree Search is a well-known search algorithm that incorporates an element of randomness in its search. With these kinds of algorithms, the problem you describe tends not to be a real issue. Due to the randomness in the search, wins that can be achieved in a small number of moves tend to be evaluated better than longer-distance wins in practice. In long-distance wins, there is a greater chance that the randomness in the search process causes an incorrect move to be played somewhere along the long-distance win, which reduces the evaluation of such a line of play.

## Case 3: (Reinforcement) Learning approaches

These approaches tend to involve some element of randomness due to the need for **exploration** in learning, which leads to similar reasoning as described for MCTS above. Also, in Reinforcement Learning, we typically use a discount factor gamma < 1.0 (e.g. gamma = 0.99) which causes distant rewards to be viewed as less important than close rewards, **even if we don't do such discounting for the final evaluation of the performance of an algorithm**. See, for example, a lot of the work on Atari games (DeepMind's DQN, etc.). Algorithms are evaluated according to their undiscounted scores, but learning still uses a bit of discounting because, in practice, this is found to be beneficial for learning.

<https://ai.stackexchange.com/questions/13886/how-to-evaluate-an-rl-algorithm-when-used-in-a-game?rq=1>

When you want to compare Reinforcement Learning algorithms, you might want to compare the average rewards they generate and how fast and close they get to the optimal policy. Often Reinforcement Learning algorithms are compared by using the rewards (either direct, maximum or average in time/iteration).