cmput396 fall 2018 **asn1 solutions** **15 marks**

2. For a position in a game and a player, call a strategy *winning* if it wins against all possible opponent strategies. A move is *winning* if it is the first move in a winning strategy.

   Hex has no draws, so for every position, either the player or her opponent has a winning strategy, so either there is a winning move (the player-to-move has a winning strategy) or there is no winning move (the player-to-move has no winning strategy); in the former case, we say the position has a *first-player* winning strategy, in the latter case, a *second-player* winning strategy.

   After 1.W[b1] Black can reply 2.B[c2] and can win with either b2 or c2, White can block at most one of these.

3. 1.W[b2] wins. Now White can join b2 to the left at one of {b1,a3} and to the right at one of {c1,c2}. Black can block at most one of each of these threats.

4. see `https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov`

   In chess the branching factor — and so the search space — is much smaller than in go. More importantly, in chess there is an easily-computed heuristic scoring function (1 point for a pawn, 3 for a knight or bishop, etc.) for arbitrary positions. No such function is known for go.

5. Silver and Huang contributed equally to the paper, so they were apparently the main architects of the AlphaGo design. From other sources we know that Huang was the lead programmer, and that he was the AlphaGo operator during the AlphaGo contests against Fan Hui, Lee Sedol and Ke Jie. Hassabis is one of 3 co-founders, and current CEO, of DeepMind.

6. unlike strong humans, who often play to maximize expected territory, AG plays to maximize the winning probability. so it's not an error if, by giving a way a few points, it slightly increases its estimated win probability.

   the algorithm was modified between matches

   training of the neural nets that were used by the program during matches continued non-stop between the tournaments

7. an allowed overtime interval. a byoyomi is used up when a player, during a move, uses more than that amount of time.

   e.g. assume you have 3 1-minute byoyomi, and you take 45 seconds to move. then after the move you still have 3 byoyomi left.

   on your next move, you take 125 seconds. that has used up 2 byoyomi, so you still have 1 left.

   on your next move, you take 59 seconds. you still have 1 left.

on your next move, you take 61 seconds. that uses up your last byoyomi, so you lose.

AG's time usage was extremely consistent on each move.

humans tend to play quickly on moves that match common variations (joseki) and will often linger on moves that warrant more search.

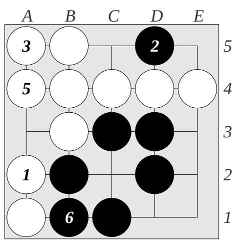
Lee Sedol never before played anyone (who had a chance of beating him) who used this time management policy.

8. avoiding fights: pivoting away from opponent moves to take space elsewhere

9. (i) W 7 stones 3 points, B 8 stones 4 points, B wins by 2

   (ii) After 1.W[a2] the obvious reply is 2.B[b1] which captures white stones. But then 3.W[d5] 4.B[a2] 5.W[a5] final score W 10 B 15, B by 5.

   But 2.B[d5] is better. Black can prevent White from making 2 eyes inside its group. White will die unless she can make an eye outside: she tries with 3.W[a5], capturing black stones. But now 4.B[a3] 5.W[a2] and the board looks like this



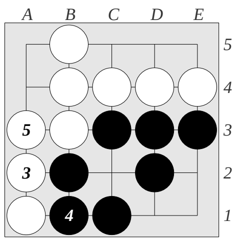
   Black can kill all white stones, so Black wins by 25.

   The computer program `fuego` agrees that 2.B[d5] is the best move for black. I set the komi to 20.5, forcing Black to try to capture the white group order to win. After a 100 second search, it picks 2.B[d5], with estimated winrate 1.00.

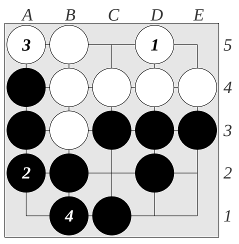
   After 2.B[d5] `fuego` as White finds 3.W[a5] has a winrate of 0.001 with main line of play A5 A3 E1 C5 D1 E2 E5 B1 A2 A1 E1 D1 D5 A2 A4 C5 B4 B5 A4 E5 B3 A5 D5 E4 C4 D4 C5 A5 B5 A5 B4 A4 PASS B5 B3 C4 B4 B3 C5 D5 PASS.

   As you can see, even on the small 5×5 board, Go problems can be hard to solve. Here, we have not proved anything, this is just what the search has found.

   (iii) After 1.W[pass], 2.B[a5] we have 3.W[a2]. Now Black preserves its territory so 4.B[b1] and 5.W[a3] and final score W 14 B 11, so White wins by 4. Fuego agrees with this analysis.



(iv) After 1.W[d5], Black connects 2.B[a2] and each player has a safe group. White gains a point with 3.W[a5], Black captures with 4.B[b1], final score B 15 W 10, Black wins by 5. Fuego agrees with this analysis.



10. there are 361 empty cells, after each move one is filled

    stones can be captured, and the empty cells then refilled

11. 4^{10} 4^{100} since the tree has at most this many leaves after 100 moves, and after evaluating all leaves, you can deduce the best move (using an algorithm we will see later called minimax).

12. There are two wander functions in rmaze.py, rwander and wander2, but each shuffles the list of neighours before traversing the list. So, if you comment out the command `shuffle(nbr_offsets)`, the algorithm is now deterministic. (Except that python makes no guarantees about the order in which elements are traversed in a for loop. So, if you want to be sure that the python code will implement deterministically, you should also force the for loops to iterate in a particular order, e.g. change `for shift in nbr_offsets` to `for shift in sorted(nbr_offsets)`.

    LIFO

    (ii) not change. heres why. In this answer I am assuming that the implementation is deterministic. Assume that you run the algorithm with a FIFO list. Let n be the number of maze cells. Let $\alpha_0\alpha_1 \ldots\ \alpha_{n-1}$ be the n cells of the maze in the order they are discovered, where α_0 is the start location. Now, if the target is in location α_j, the algorithm takes j iterations. Each location is equally likely, so the average number of iterations can be found by summing each location-execution once and dividing by n, so $(\sum_0^{n1} j = (n(n1)2)/n = (n1)/2$. This average value is the same for any algorithm which discovers one new cell with each iteration. The fewest number of iterations will be the number of cells on a shortest path from the start to the target. The maximum number of iterations will be the number of cells in a longest path from the goal to the target.