Name: Solution Guide **CCID:** _____

1. **[4 points]** In the example code `mcts/mcts1.py`:

   (a) What is the difference between `get_best_move` and `best_uct`?
   The `best_uct` method uses UCT (i.e., winrate + exploration bonus) to select a child within the search tree to traverse; it is called repeatedly until a leaf in the search tree is reached. The `get_best_move` method is used after search to select a single move from the root node to actually play; it returns the child of the root with the most visits.

   (b) Consider the following code from the `best_uct` method:

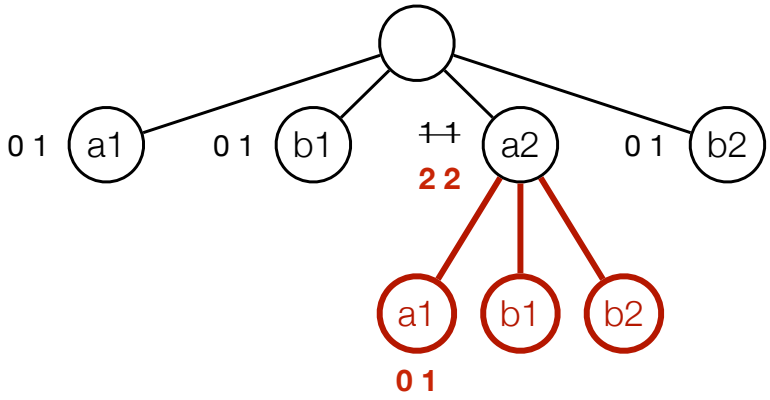
   ```
   # calculate UCT, update if best
   mean_res = child.results / child.sims
   uct = ___(1)___ + (self.c * sqrt(log(self.root_node.sims) / child.sims))
   if best_uct is None or uct > best_uct:
       best_uct, best_child = uct, child
   ```

   What should missing expression (1) be?
   `mean_res`

   (c) `self.c` gets initialized to 0.3. What would be the effect of setting it to something much larger (e.g., 50)?
   Making c larger makes the exploration bonus term larger. This means that exploration will be weighted more heavily than exploitation; relatively less-visited nodes will be chosen more often even if their winrates are low than they would have been with a smaller value of c.

   (d) If you run `mcts/main.py` multiple times and ask for a winning move for Black (using the command `g x`), it will sometimes generate different moves in different runs. Why?
   MCTS chooses nodes based on their estimated quality. However, their quality is estimated based on random rollouts, which are noisy; on different runs, any given random rollout may give a different result, which can cause nodes that appear high-quality in some runs to appear low-quality in others.

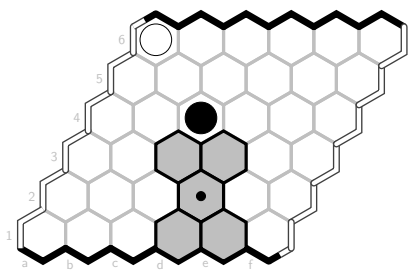
2. **[3 points]** Below is the Monte Carlo search tree after the first 4 simulations of running the `g x` command of `mcts/main.py` with a 2×2 board. Show the tree after simulation 5 (output below); explain each change. The indices map to positions as 5=a1, 6=b1, 9=a2, 10=b2. Additions in red.




```
trv_xpnd bu * .4 .4 1.4 .4 9
  xpnd_nd * 9 > 5
  xpnd_nd * 9 > 6
  xpnd_nd * 9 > 10
sim 5. * 9 5 roll 6 parent loss
```

After simulation 4, MCTS traverses the search tree from the root to node 9 (i.e., a2). Since a2 has a previous simulation, it expands the node, adding its three children (a1, b1, b2) to the search tree, and then performs a simulation/rollout starting from the new child 5 (i.e., a1, chosen arbitrarily from the children). The simulation results in a loss for a1's parent, so we record "0 wins/1 visit" on a1; since it was a loss for a1's parent, it was a win for the parent's parent, and so we add 1 to the win count and 1 to the visit count for the parent a2.

3. **[5 points]** Consider the following virtual connection in 6×6 Hex:



(a) Does this depict a *full connection* or a *semi-connection* between the Black stone and the bottom border? Explain your reasoning.

This is a *semi-connection*: If it is currently Black's turn, they can play on d2, which is strongly connected to c4 by a bridge pattern (c3 and d3) and also to the bottom border by another bridge pattern (d1 and e1). However, if it is White's turn currently, they can block this strategy by playing on d2.

(b) Write this connection in logical form as an AND/OR strategy.

$d2 \wedge (c3 \vee d3) \wedge (d1 \vee e1)$

(c) To follow this strategy, where must Black play after `1.B[c4] 2.W[a6] 3.B[d2] 4.W[e1]`? d1