CMPUT 355 Quiz 5 Marking Rubric

Grading Rubric

```
Problem 1
    0.4 points for each correct answer
        (Note: marks were deducted for answering incorrect moves, if
        more than 5 moves were circled.)

Problem 2
    a) 3 points max
        1/3 point for each correct number of simulations
        2 points for a reasonable justification

    b) 3 points max
        1 point for correct average number of winning moves
        2 points for reasonable justification

    Note: the justification sections were marked together,
    so if your justification for part a) helps justify your answer
    for part b) (and vice versa), you will get marks for it unless
    it contradicts your earlier justification.

Problem 3
    See below

Problem 4
    2 points for correct expansions
        Points will be deducted if extra expansions are given

    6 points for correct wins/sims for all nodes simulated (excluding the one
        that is given at the start)
        Points will be deducted if extra incorrect wins/sims are given
```

Quiz 5d

1. Winning moves for black: a2, a3, b2, c1, c2
2. (a) `uct = mean_res, ~16,000 simulations`
   `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims))), ~13,000 simulations`
   `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims)), ~11,000 simulations`

   For justification, something like the following:

   i. `uct = mean\_res` completes the most simulations because it is the fastest to compute, has the fewest/cheapest instructions, etc.

   ii. `uct = mean\_res+(self.c*(self.root\_node.sims/(child.sims+self.root\_node.sims)))` is in the middle of the other two because it contains more operations than `uct = mean\_res`, but the operations themselves (addition, multiplication, division, etc.) are faster than the log and sqrt operations in `uct = mean\_res+(self.c*sqrt(log(self.root\_node.sims)/child.sims))`.

   iii. `uct = mean\_res+(self.c*sqrt(log(self.root\_node.sims)/child.sims))` completes the fewest simulations because log and sqrt are slow to compute relative to other operations (addition, multiplication, division, etc.).

   (b) `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))`, between 19-20 winning moves

   `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims)))`, between 13-17 winning moves

   `uct = mean_res`, between 10-15 winning moves

   For justification, something like the following:

   i. `uct = mean_res` does not account for uncertainty in the estimates at all/treats estimates as 100% accurate/no exploration/all exploitation, so it spends simulations on only a few moves, and often misses winning moves.

   ii. `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims)))` uses the ratio of simulations from the root node to simulations from the root node plus simulations from the child node combined. However, this heuristic does not accurately approximate uncertainty, and while it explores better than `uct = mean_res` it still explores too poorly to find winning moves at the same rate as the more expensive way.

   iii. `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))` is the most accurate way of computing the upper confidence bound that accounts for uncertainty, so it is able to select good moves more often and bad moves less often as it becomes more certain about its estimates, effectively balancing exploitation of the current best move and exploration of other moves, allowing it to reliably find winning moves.

3. (a) (1 mark) White.

   (b) (1 mark) First player (it gives an initial move)

(c) (4 marks) White moves at c1, creating a VC from left to right. (1 mark) There are two possibilities (semi-connections). 1) White moves at b1, then either a1 or a2. (1 mark) 2) White moves at a3,nthen if possible, b2 to win. (1 mark) If not, it plays at b3, and then either c2 or c3. (1 mark)

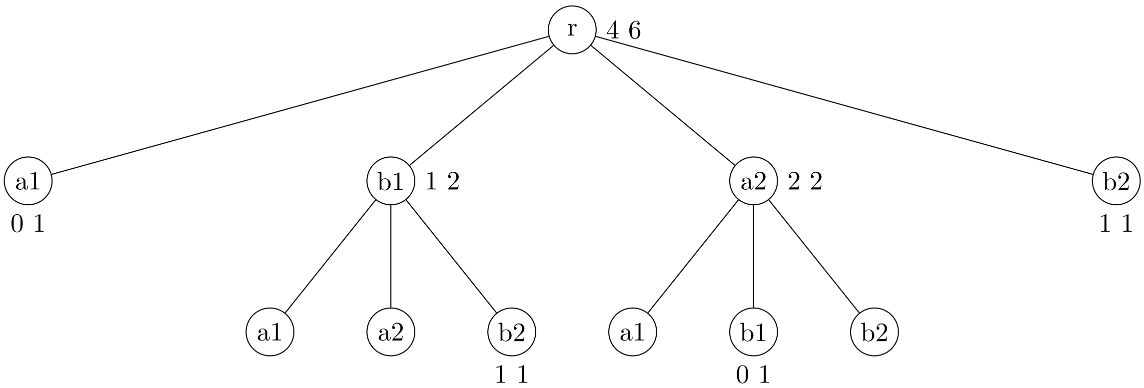
(d) (2 marks). Partial credit is awarded for work that demonstrates understanding.

```
5d)
                                                root
                                                 |
                                                 c1
  /                     /    /    /     /               \                      \    \
 b2                    b3   c1    c2   c3               b1                      a2   a1
  |                    |    |     |     |                 |                       |   |
  b1                   b1    b1    b1    b1                a3                     a3   a3
 /   \                 /\    /\    /\    /\               /  \                    /\   /  \
a1 (others (1))       a1 .  a1 .  a1 .  a1 .            b2  (others (2))        b2  .  b2  .
 |     |              |  |  |  |  |  |  |  |             |      |               |   |  |   |
a2     a1            a2 a1 a2 a1 a2 a1 a2 a1            b3      b2             b3  b2  b3   b2
                                                       /  \                    /\      /\
                                                     c2  (others (3))        c2 .    c2 .
                                                      |       |              |  |    |  |
                                                     c3      c2              c3 c2   c3 c2

others (1) = 5 possible moves
others (2) = 5 possible moves
others (3) = 4 possible moves

(1 + 1 + 1 + 1 + (2 * 5)) * 5       +     (1 + 1 + 1 + (2 * 5) + 1 + 1 + 1 + (2 *4)) * 3        +  1 + 1 = 146 nodes
```

4.



Quiz 5e

1. Winning moves for white: a3, b1, b2, b3, c1

2. (a) `uct = mean_res, ~16,000 simulations`
   `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims))), ~13,000 simulations`
   `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims)), ~11,000 simulations`

   For justification, something like the following:

   i. `uct = mean\_res` completes the most simulations because it is the fastest to compute, has the fewest/cheapest instructions, etc.
   ii. `uct = mean\_res+(self.c*(self.root\_node.sims/(child.sims+self.root\_node.sims)))` is in the middle of the other two because it contains more operations than `uct = mean\_res`, but the operations themselves (addition, multiplication, division, etc.) are faster than the log and sqrt operations in `uct = mean\_res+(self.c*sqrt(log(self.root\_node.sims)/child.sims))`.
   iii. `uct = mean\_res+(self.c*sqrt(log(self.root\_node.sims)/child.sims))` completes the fewest simulations because log and sqrt are slow to compute relative to other operations (addition, multiplication, division, etc.).

   (b) `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))`, between 19-20 winning moves

   `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims)))`, between 13-17 winning moves

   `uct = mean_res`, between 10-15 winning moves

   For justification, something like the following:

   i. `uct = mean_res` does not account for uncertainty in the estimates at all/treats estimates as 100% accurate/no exploration/all exploitation, so it spends simulations on only a few moves, and often misses winning moves.
   ii. `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims)))` uses the ratio of simulations from the root node to simulations from the root node plus simulations from the child node combined. However, this heuristic does not accurately approximate uncertainty, and while it explores better than `uct = mean_res` it still explores too poorly to find winning moves at the same rate as the more expensive way.
   iii. `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))` is the most accurate way of computing the upper confidence bound that accounts for uncertainty, so it is able to select good moves more often and bad moves less often as it becomes more certain about its estimates, effectively balancing exploitation of the current best move and exploration of other moves, allowing it to reliably find winning moves.

3. (a) (1 mark) White.
   (b) (1 mark) First player (it gives an initial move)

(c) (4 marks) White moves at a3, creating a VC from left to right. (1 mark) There are two possibilities (semi-connections). 1) White moves at b3, then either c3 or c2. (1 mark) 2) White moves at c1, then if possible, b2 to win. (1 mark) If not, it plays at b1, and then either a2 or a1. (1 mark)

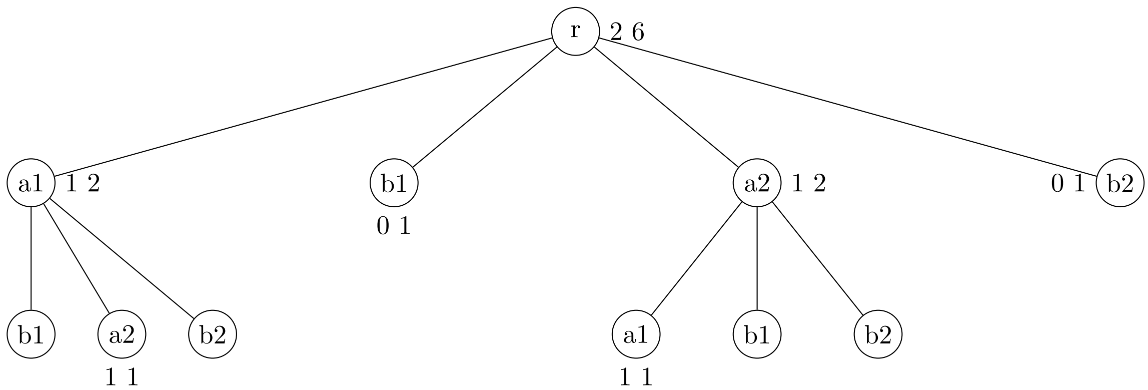
(d) (2 marks). Partial credit is awarded for work that demonstrates understanding.

```
                                                 root
                                                  |
                                                 a3
  /                     /    /    /     /               \                      \    \
 b1                    b2   a1    a2   a3               c2                      b3   c3
  |                     |    |     |     |                |                       |    |
 b3                    c2   c2     c2    c2               c1                      c1   c1
 /   \                 /\    /\    /\    /\               /   \                   /\   /   \
c2 (others (1))       b3 .  b3 .  b3 .  b3 .            b2  (others (2))        b2   . b2   .
 |     |              |  |  |  |  |  |  |  |              |      |              |   |  |    |
c3     c2            c3 c2 c3 c2 c3 c2 c3 c2           b1      b2            b1  b2  b1   b2
                                                      /   \                   /\      /\
                                                    a1  (others (3))         a1 .    a1 .
                                                    |       |                | |    |  |
                                                   a2      a1               a2 a1  a2 a1

others (1) = 5 possible moves
others (2) = 5 possible moves
others (3) = 4 possible moves

(1 + 1 + 1 + 1 + (2 * 5)) * 5        +     (1 + 1 + 1 + (2 * 5) + 1 + 1 + 1 + (2 *4)) * 3        +  1 + 1 = 146 nodes
```

4.



Quiz 5f

1. Winning moves for white: b1, c1, b2, a3, b3
2. (a) `uct = mean_res, ~16,000 simulations`
   `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims))),`
   `~13,000 simulations`
   `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims)), ~11,000 simulations`

   For justification, something like the following:

   i. `uct = mean\_res` completes the most simulations because it is the fastest to compute, has the fewest/cheapest instructions, etc.

   ii. `uct = mean\_res+(self.c*(self.root\_node.sims/(child.sims+self.root\_node.sims)))` is in the middle of the other two because it contains more operations than `uct = mean\_res`, but the operations themselves (addition, multiplication, division, etc.) are faster than the log and sqrt operations in
   `uct = mean\_res+(self.c*sqrt(log(self.root\_node.sims)/child.sims))`.

   iii. `uct = mean\_res+(self.c*sqrt(log(self.root\_node.sims)/child.sims))` completes the fewest simulations because log and sqrt are slow to compute relative to other operations (addition, multiplication, division, etc.).

   (b) `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))`, between 19-20 winning moves

   `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims)))`, between 13-17 winning moves

   `uct = mean_res`, between 10-15 winning moves

   For justification, something like the following:

   i. `uct = mean_res` does not account for uncertainty in the estimates at all/treats estimates as 100% accurate/no exploration/all exploitation, so it spends simulations on only a few moves, and often misses winning moves.

   ii. `uct = mean_res+(self.c*(self.root_node.sims/(child.sims+self.root_node.sims)))` uses the ratio of simulations from the root node to simulations from the root node plus simulations from the child node combined. However, this heuristic does not accurately approximate uncertainty, and while it explores better than `uct = mean_res` it still explores too poorly to find winning moves at the same rate as the more expensive way.

   iii. `uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))` is the most accurate way of computing the upper confidence bound that accounts for uncertainty, so it is able to select good moves more often and bad moves less often as it becomes more certain about its estimates, effectively balancing exploitation of the current best move and exploration of other moves, allowing it to reliably find winning moves.

3. (a) (1 mark) Black.

   (b) (1 mark) First player (it gives an initial move)

(c) Black moves at c1, creating a VC from top to down. (1 mark) There are two possibilities (semi- connections). 1) Black moves at c2, then either b3 or c3. (1mark) 2) Black moves at a3, then if possible, b2 to win. (1 mark) If not, it plays at a2, and then either a1 or b1. (1 mark)

(d) (2 marks). Partial credit is awarded for work that demonstrates understanding.

```
                                                root
                                                 |
                                                 c1
  /                      /    /    /     /               \                      \    \
 b1                    b2   a1    a2    a3               c2                      b3   c3
  |                    |    |     |      |                |                       |   |
 c2                    c2   c2      c2    c2               a3                      a3   a3
 /   \                 /\    /\    /\    /\                /  \                    /\   /  \
b3 (others (1))       b3 .  b3 .  b3 .  b3 .             b2  (others (2))        b2  .  b2  .
 |     |              |   | |  |  |  |  |  |              |      |               |   |  |   |
c3     b3            c3 b3 c3 b3 c3 b3 c3 b3             a2      b2             a2  b2  a2    b2
                                                         /  \                    /\      /\
                                                       a1  (others (3))         a1 .    a1 .
                                                        |       |               |  |    |  |
                                                        b1      a1              b1 a1  b1 a1

others (1) = 5 possible moves
others (2) = 5 possible moves
others (3) = 4 possible moves

(1 + 1 + 1 + 1 + (2 * 5)) * 5       +     (1 + 1 + 1 + (2 * 5) + 1 + 1 + 1 + (2 *4)) * 3          +  1 + 1 = 146 nodes
```

4.