Algorithm 1 kthBestPick(Node *node*)

1: $numPicksRemaining \leftarrow \text{ceil}\left(\frac{node.\text{getNumUnownedTerritories()}}{node.\text{getNumPlayers()}}\right)$
2: $bestToWorstPicks \leftarrow$ sort(*node*.getUnownedTerritories(), terrEvalFunction(*node*)) {How to handle tie-breaks?}
3: **for** $k = numPicksRemaining - 1$ to 0 **do**
4: $terr \leftarrow bestToWorstPicks(k)$
5: $higherRankedTerrs \leftarrow bestToWorstPicks(0..k-1)$ {More tie-break issues here}
6: $activePlayer \leftarrow node$.getActivePlayer()
7: *node*.assignTerritory(*terr*, *activePlayer*)
8: $makeThisPick \leftarrow$ territoriesPickedBy(*higherRankedTerrs*, *activePlayer*, *node*)
9: *node*.unassignTerritory(*terr*)
10: **if** *makeThisPick* **then**
11: **return** *terr*
12: **end if**
13: **end for**

Algorithm 2 territoriesPickedBy(List<Territory> *territories*, Player *originalPlayer*, Node *node*)

1: $copyOfNode \leftarrow node$.clone()
2: **while** !*territories*.isEmpty() **do**
3: $pick \leftarrow$ kthBestPick(*copyOfNode*)
4: **if** *territories*.contains(*pick*) **then**
5: **if** *copyOfNode*.getActivePlayer() == *originalPlayer* **then**
6: *territories*.remove(*pick*)
7: **else**
8: **return** false
9: **end if**
10: **end if**
11: *copyOfNode*.assignTerritory(*pick*, *copyOfNode*.getActivePlayer())
12: **end while**
13: **return** true

The objective of kthBestPick is to pick the territory *terr* with the worst rank k possible so that we may eventually pick all $k-1$ territories ranked higher than *terr* with our later picks. If we assume that we and all of our opponents follow the same logic throughout the draft and use the same territory evaluation function, then Algorithm 1 returns this territory. The algorithm works as follows: First, we determine the number of picks we have remaining (line 1) and set k to this value (line 3), as we do not have enough picks to consider lower-ranked territories. Then, we check whether all of the $k-1$ higher-ranked territories are picked by us in the later rounds (line 7), given that we picked the territory ranked k (lines 4 and 6).

If they are all picked by us, then we return the territory ranked k (line 10); otherwise, we decrease k by one (line 3) and repeat the check.

The work of Algorithm 1 is mostly done on line 7, and is captured by Algorithm 2. Looking at this more closely, territoriesPickedBy(, ,) returns true if all of the passed in territories are eventually picked by *originalPlayer*, given the current state of the draft (which is contained in *node*) and that all players make decisions via kthBestPick. It operates by sequentially determining the next picks made by the players (line 3) and checking whether these picks are in the passed in list of territories (line 4). If a pick is not in the list, then we move on to the next pick (line 11 and back to line 3). However, if a pick is in the list of territories, we either remove it from the list (line 6) (as we terminate once all territories in the list have been picked by originalPlayer (lines 2 and 13)), or return false (line 8), as another player has chosen one of the passed in territories.

Note that within one external call to kthBestPick, it is likely that a lot of work is being redone in the territoriesPickedBy algorithm. At line 3, we compute the kthBestPick, which itself will call kthBestPick on the next state before returning the pick. Then, on the next iteration through the while loop, we call kthBestPick on the next state again. This indicates that we will probably want to store a hash table of the states which we have called kthBestPick on with their corresponding returned pick.

Finally, we need a territory evaluation function so that we can rank the picks at a given state. We still need to decide on a good method of acquiring such a function.