PuzzleSolver

The following defines a puzzle:

* An initial state (Position initialPos)
* A set of moves at each state (vector<Moves> getMoves(Position pos))
* Rules for determining if a state is a primitive (bool isPrimitive(Position pos))

Note: Win by reaching a primitive state. Losing positions where no moves are available are not considered primitive. A puzzle is still solvable (winnable) if and only if there exists a path from the current position to a primitive state.

High level approach:

1. Starting at the initial position, run BFS to determine all primitive states and construct the backward graph.
2. Starting at each primitive state, run BFS on the backward graph, updating minimum remoteness of each node.

Implementation of backward graph:

* Unordered map from states to a vector of parent states.

BFS algorithm for finding primitive states and constructing backward graph:

1. Initialize empty queue Q.
2. Initialize empty *closed* set C.
3. Add initial state to Q.
4. While Q is not empty,
   1. S = Q.dequeue().
   2. If S is in C, continue loop.
   3. Add S to C, effectively closes S.
   4. If S is a primitive state,
      1. Add S to the set of primitive states.
   5. Else,
      1. For each child of S,
         1. Add child state to Q
         2. Add S to the vector of parents of child in backward graph
5. Garbage collect Q and C.

Note:

1. Shortest forward path can be easily obtained from remoteness. We can just loop through all forward moves and pick the one that leads to a decrease in remoteness.