Objects

• Board
  – int* _tiles, int _len;

• PuzzleMove
  – Board* b, int g, int h, int tileMoved, PuzzleMove* parent (a.k.a. pred)

• PuzzleSolver
  – Board _b, deque<int> _solution, int _expansions;
Key Functions

- **Board::potentialMoves()**
  - Figures out which tiles can be moved and creates new boards representing the results as if you had made that move
  - Returns a map\<int = tile, Board* = pointer to new board w/ tile moved>\n
- **PuzzleSolver::run()**
  - Implements the A* algorithm
  - You will declare your heap (open list) here, your closed set, and any other data structures you need
  - You will results a class called PuzzleHeuristic as an argument. It will have a function that can score a Board for you (using either Manhattan distance, TilesOutOfPlace, etc.)

- **PuzzleHeuristic and derived classes**
  - PuzzleManhattanHeuristic::compute()
  - PuzzleOutOfPlaceHeuristic::compute()
  - PuzzleBFSHeuristic::compute()
  - Computes the score of a Board provided as an input argument
Run()

1. Remove minimum scored move from the heap
2. potentialMoves()
3. PMptr
4. PuzzleMoves made from potential Boards and then entered back into the openList if it isn't already in the closedList
Run()
Pointers and STL Containers

- Don't put a pointer to a locally allocated object into a data structure
  - Board b;
  - Map<int, Board*> nextMoves;
  - nextMoves[tile] = &b;
  - return nextMoves;
  - // the map is returned by has pointers to dead Boards

- STL data structures always store a COPY of what you add to them.
STL Containers of Pointers

- When an STL data structure dies it will free up the objects that remain inside of it (destructor would be executed for the object)
- **BUT** if you are storing pointers, it will delete the memory of the pointer, but not delete what it is pointing at
  - vector<PuzzleMove*> mylist;
  - PuzzleMove* p = new PuzzleMove(...)
  - mylist.push_back(p);
  - ...  
  - // mylist dies...does that PuzzleMove that was dynamically allocated get deleted...NO!!!!!!!!!!
  - You must iterate through remaining items in a data structures containing pointers and delete them manually if you want
  - for(vector<PuzzleMove*>::iterator it = mylist.begin(); ...)  
  - { delete *it; }
When to Delete / Ownership

• In this assignment dynamically allocated stuff (really pointers to them) will pass from one data structure to the next
• It is up to you not to lose a dynamically allocated object
  – Make sure someone has a pointer
• You must decide which object or data structure "owns" that dynamically allocated object so that when that owning object dies, you know you will also deallocate the dynamically allocated object
  – Who owns Boards? [i.e. will a PuzzleMove delete it when it dies via is destructor, will you somehow delete them manually?]
  – Who owns PuzzleMoves?
• What containers will store PuzzleMoves?
  – Make sure when you find a solution, any and all PuzzleMoves are deleted (but don't do it before you've used the info you need from the PuzzleMoves)