Video Game AI, Spring 2013
HW #3
Due April 23 at 12 midnight

This assignment is to be done individually. While you can use the textbook or web resources to understand the A* algorithm, all code should be your own.

In this assignment you will write the core loop of the A* search algorithm. All of the data structures and other functions required for A* have already been provided. On the course web page is a sample project which contains movement code. You must implement the function DoSingleSearchStep inside AStarCoreSearch.h (In the "Code To Edit" group/folder). This function returns `true` when the search is complete (the goal is found, or no path is found). Otherwise it should return `false`. Overall, it should perform the following tasks:

1. Check if the open list is empty.
    a. If so, the search is complete.
2. Expand the next state from OPEN.
    a. If this is the goal, the optimal path should be returned (by reference).
3. Get the neighbors of the next state.
4. Update each neighbor appropriately, depending on whether it is on the closed or open list (or neither).

For reference, the sample solution is about 40 lines long.

On the next page are the pre-defined functions which can be used to implement the core A* logic.

```
// returns the number of items on the open list
void GetNumOpenItems();

// returns the best state on open and moves it to the closed list
state GetNextState();

// returns true if theState is the goal
bool IsGoal(const state &theState);

// puts the path from theState back to the start state in thePath
void ExtractPathToStart(state &theState, std::vector<state> &thePath);

// puts the neighbors of theState in the vector neighbors
void GetNeighbors(const state &theState, std::vector<state> &neighbors);

// finds whether theState is on kOpenList, kClosedList, or kNotFound
dataLocation FindStateLocation(const state &theState);

// returns the stored g-cost of theState
double GetGCost(const state &theState);

// sets the stored g-cost of theState to cost and
// re-sorts the open list accordingly
void SetGCost(const state &theState, double cost);

// returns the cost of the edge between state1 and state2
double GetEdgeCost(const state &state1, const state &state2);

// updates the parent of theState in the open list
void SetParent(uint64_t &theState, const state &parent);

// adds theState to the open list with the given g-cost, h-cost and parent
void AddToOpen(const state &theState, double gCost, double hCost,
               const state &parent);

// returns the heuristic between theState and the goal
double GetHCost(const state &theState);
```