EECS 367 Lab: Assignment 1 Code Overview
Administrative

- Next **Wednesday, September 18**
  - Quiz 1 during regular lecture time and location
  - Assignment #1: Path Planning
    - Due 11:59pm, Wednesday, September 18
  - Send Professor Jenkins your Git account/repo (through slack)
Lab Takeaways

1) Stencil Overview
2) Walkthrough Heap Insert
3) Validate Implementation
4) Search Canvas Introduction
5) Data Structure Considerations

How to start assignment 1
Assignment 1

<table>
<thead>
<tr>
<th>Points</th>
<th>Sections</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>All</td>
<td>Heap implementation</td>
</tr>
<tr>
<td>8</td>
<td>All</td>
<td>A-star search</td>
</tr>
<tr>
<td>2</td>
<td>Grad</td>
<td>BFS</td>
</tr>
<tr>
<td>2</td>
<td>Grad</td>
<td>DFS</td>
</tr>
<tr>
<td>2</td>
<td>Grad</td>
<td>Greedy best-first</td>
</tr>
</tbody>
</table>
KinEval Stencil

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>js</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>kineval</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>project_pathplan</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>project_pendulum</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>robots</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>tutorial_heapsort</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>tutorial_ja</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>worlds</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>LICENSE</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>README.md</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
<tr>
<td>home.html</td>
<td>initial commit</td>
<td>7 days ago</td>
</tr>
</tbody>
</table>

Bulk of code needed to complete Assignment 1

Starter code to help debug heap
heapsort.html

A script element contains JavaScript code for the browser to execute. Script code could be in another file, specified in the "src" attribute, as in the case below for your heap code.

This external source file should produce the object "minheaper" that has two function methods for inserting and extracting heap elements.

<!-- you will want to uncomment this tag-->
<script src="heap.js"></script>

Specify JavaScript source file to make accessible for HTML
Represent heap as JS array
Repeatedly call heap insert method
Print state of heap to screen
Heap Insert

Heap operations: Insert

1) add new element to end of tree

2) swap with parent

3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

```c
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    // TODO: Find index for new_element
    // TODO: Find index of new_element's parent
    // TODO: Add new_element to the heap array
    // TODO: Initialize heap condition check
    // TODO: As long as heap condition not satisfied
    //     TODO: Swap new_element with parent
    //     TODO: Update index for new_element
    //     TODO: Update index for new_element's parent
    //     TODO: Update heap condition check
}
```
Heap Insert

Heap operations: Insert

1) add new element to end of tree

2) swap with parent

3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

// define insert function for min binary heap

```javascript
function minheap_insert(heap, new_element) {
  var elntIdx = heap.length;
  // TODO: Find index of new_element's parent
  // TODO: Add new_element to the heap array
  // TODO: Initialize heap condition check
  // TODO: As long as heap condition not satisfied
  //   TODO: Swap new_element with parent
  //   TODO: Update index for new_element
  //   TODO: Update index for new_element's parent
  // TODO: Update heap condition check
}
```
Heap Insert

Heap operations: Insert

1) add new element to end of tree

2) swap with parent

3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

// define insert function for min binary heap
function minheap_insert(heap, new_element) {
  var elntIdx = heap.length;
  var prntIdx = Math.floor((elntIdx - 1) / 2);

  // TODO: Add new_element to the heap array
  // TODO: Initialize heap condition check

  // TODO: As long as heap condition not satisfied
  //      TODO: Swap new_element with parent
  //          TODO: Update index for new_element
  //          TODO: Update index for new_element's parent

  // TODO: Update heap condition check
}
Heap Insert

Heap operations: Insert

1) add new element to end of tree

2) swap with parent

3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

```javascript
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var printIdx = Math.floor((elntIdx - 1) / 2);
    heap.push(new_element);

    // TODO: Initialize heap condition check
    // TODO: As long as heap condition not satisfied
    //   TODO: Swap new_element with parent
    //   TODO: Update index for new_element's parent
    //   TODO: Update heap condition check
}
```
Heap Insert

Heap operations: Insert

1) add new element to end of tree
   \[ \begin{array}{c}
   11 \\
   5 \\
   8 \\
   \end{array} \]

2) swap with parent
   \[ \begin{array}{c}
   11 \\
   15 \\
   8 \\
   \end{array} \]

3) until heaped, do (2)
   \[ \begin{array}{c}
   15 \\
   8 \\
   \end{array} \]

For priority queue, previously non-queued locations will be inserted with f_score priority

---

Note: example is for max heap...

```javascript
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor((elntIdx - 1) / 2);
    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    // new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    // TODO: As long as heap condition not satisfied
    // TODO: Swap new_element with parent
    // TODO: Update index for new_element
    // TODO: Update index for new_element's parent
    // TODO: Update heap condition check
}
```
code is for min heap
Heap Insert

Heap operations: Insert

1) add new element to end of tree

2) swap with parent

3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

---

```javascript
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor((elntIdx - 1) / 2);

    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    // new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
        // TODO: Swap new_element with parent
        //
        // TODO: Update index for new_element
        //
        // TODO: Update index for new_element's parent
        //
        // TODO: Update heap condition check
        }
    }
```
Heap Insert

Heap operations: Insert

1) add new element to end of tree

2) swap with parent

3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

```javascript
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
  var elntIdx = heap.length;
  var prntIdx = Math.floor((elntIdx - 1) / 2);

  heap.push(new_element);

  // Heap condition is true if new element added as root, or if
  // new element is less than or equal to its parent element
  var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

  while (!heaped) {
    // Swap element and parent
    var tmp = heap[prntIdx];
    heap[prntIdx] = heap[elntIdx];
    heap[elntIdx] = tmp;

    // TODO: Update index for new_element
    // TODO: Update index for new_element's parent
    // TODO: Update heap condition check
  }
}
```
Heap operations: Insert

1) add new element to end of tree
2) swap with parent
3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor((elntIdx - 1) / 2);
    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    // new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
        // Swap element and parent
        var tmp = heap[prntIdx];
        heap[prntIdx] = heap[elntIdx];
        heap[elntIdx] = tmp;

        // Update element and parent index
        elntIdx = prntIdx;
        prntIdx = Math.floor((elntIdx - 1) / 2);

        // TODO: Update heap condition check
    }
}
Heap Insert

Heap operations: Insert

1) add new element to end of tree
2) swap with parent
3) until heaped, do (2)

For priority queue, previously non-queued locations will be inserted with f_score priority

```javascript
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor((elntIdx - 1) / 2);
    heap.push(new_element);

    // Heap condition is true if new element added as root, or if new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
        // Swap element and parent
        var tmp = heap[prntIdx];
        heap[prntIdx] = heap[elntIdx];
        heap[elntIdx] = tmp;

        // Update element and parent index
        elntIdx = prntIdx;
        prntIdx = Math.floor((elntIdx - 1) / 2);

        // Re-evaluate heap condition
        heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);
    }
}
```
heapsort.html
Lab Takeaways

1) Stencil Overview
2) Walkthrough Heap Insert
3) Validate Implementation
4) Search Canvas Introduction
5) Data Structure Considerations

How to start assignment 1
**initSearch() function:**
Instantiates global variables, commence algorithms

```javascript
function initSearch() {
  // Instantiate search algorithm to use for planning
  search_alg = "A-star";
  // Instantiate the world for the planner
  planning_scene = "maze";
  // Instantiate start location in the world
  q_start_config = q_init;
  // Instantiate goal location in the world
  q_goal_config = q_goal;

  // q_init and q_goal will be specified in URL
}
```
search_canvas.html
search_canvas.html

G[ i ][ j ]: search graph cell is a JavaScript object
search_canvas.html

$G$: search graph

$G[i][j]$: search graph cell

.x and .y specifies cell center in world coordinates

cell has .distance along path through .parent

eps: cell height and width

$[G[i][j].x, G[i][j].y]$
test configuration at visited cell center for collision:

testCollision([G[i][j].x, G[i][j].y])
search_canvas.html

search starts from cell that contains configuration $q_{\text{init}}$

$q_{\text{init}}= [0, 0]$ (world coordinates)

search goal in cell that contains configuration $q_{\text{goal}}$

$q_{\text{goal}}= [4, 4]$ (world coordinates)
Important to identify discrete start indices within graph from continuous world position.

```javascript
function initSearchGraph() {
    // KE: visit queue not created for certain values of eps
    visit_queue = [];  
    // initialize search graph as 2D array over configuration space 
    // of 2D locations with specified spatial resolution 
    G = [];  
    for (iind=0; xpos<7; iind++, xpos+=eps) { 
        G[iind] = [];  
        for (jind=0; ypos<7; jind++, ypos+=eps) { 
            G[iind][jind] = { 
                i:iind, j:jind, // mapping to graph array 
                x:xpos, y:ypos, // mapping to map coordinates 
                parent:null, // pointer to parent in graph along motion path 
                distance:10000, // distance to start via path through parent 
                visited:false, // flag for whether the node has been visited 
                priority:0, // visit priority based on gscore 
                queued:false // flag for whether the node has been queued for visiting 
            };
        };
    };
    // STENCIL: determine whether this graph node should be the start point for the search 
    visit_queue = [];
}
```
The `animate()` function is responsible for calling your iterate functions. Animation is accomplished by iterative `drawUpdate()` → `computeUpdate()`.
Including excessive loops may cause browser to become unresponsive

Ensure your implementations are isolated to single search steps

```javascript
function iterateGraphSearch() {
    // STENCIL: implement a single iteration of a graph search algorithm
    // for A-star (or DFS, BFS, Greedy Best-First)
    // An async timing mechanism is used instead of a for loop to avoid
    // blocking and non-responsiveness in the browser.
    // Return "failed" if the search fails on this iteration.
    // Return "succeeded" if the search succeeds on this iteration.
    // Return "iterating" otherwise.
    // Provided support functions:
    // testCollision - returns whether a given configuration is in collision
drawHighlightedPathGraph - draws a path back to the start location
    draw_2D_configuration - draws a square at a given location

    search_iterate = false;  // stop search
}
```