

**Class Number:** CPSC 481-04

**Project Number and Name:** Project 1, A\* Search

**Team Name:** IDK\_Guys

**Team Members:** Tommy Huynh, Juheng Mo, Calvin Nguyen, Chi (Michael) Lam

## Project Algorithm Report

### Step Count:

|                        |  |
|------------------------|--|
| (p5 draw_bot 2 1 0 )   | <a href="#">cs-sketch-paint.js:192</a> |
| end draw_bot)          | <a href="#">cs-sketch-paint.js:201</a> |
| (p5 draw_bot 2 35 26 ) | <a href="#">cs-sketch-paint.js:192</a> |
| end draw_bot)          | <a href="#">cs-sketch-paint.js:201</a> |
| Step Count = 84        | <a href="#">cs-sketch-paint.js:313</a> |

The bot goes through several different paths creating several routes in the process. The step count does not count backtracking or other paths to create the true step count. The final step count is decided when the bot reaches the goal tile and then counts backwards and compares it to other finished paths to check for the shortest path. The final step counts for the best path being 84 steps.

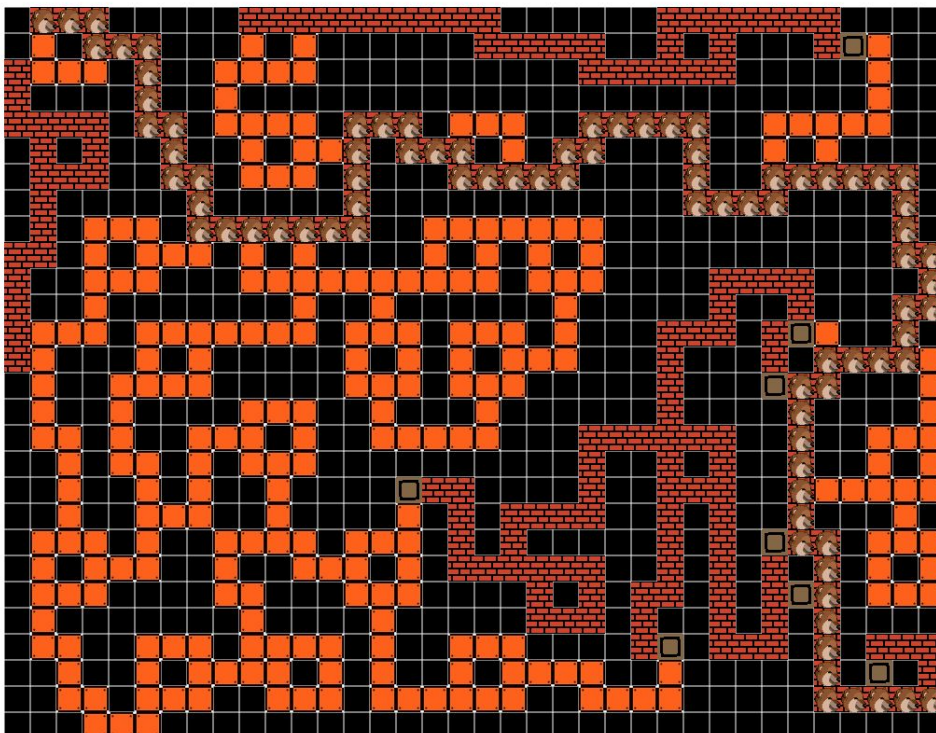
### Final Board Position:

#### CPSC 481 - Section 4

[Save Image](#)

Team IDK Guys

Press 'S' to Start/Pause



As shown above, this is the shortest path deemed by the AI with a step count of 84. The orange boxes represent the path the AI Bot has taken and evaluated, while the Brown boxes indicate the potential paths the bot will eventually evaluate. The pixel dog represents the bot's current best path eventually leading the end. There is an example of where a path could be taken by the bot, but it deemed it longer than the best path shown, leaving the alternate path as a brown box.

## Algorithm:

```
function draw() // P5 Frame Re-draw Fcn, Called for Every Frame.
{
  ++g_frame_cnt;
  if (!g_stop && (0 == g_frame_cnt % g_frame_mod))
  {
    //console.log( "p5 draw" );
    //move_bot_to_mouse( );
    //draw_update( );

    if (openSet.length > 0) {
      // keep going
      var lowestIndex = 0;
      for (var i = 0; i < openSet.length; i++) {
        if (openSet[i].f < openSet[lowestIndex].f) {
          lowestIndex = i;
        }
      }
      var current = openSet[lowestIndex];
      console.log(current)

      for (var i = 0; i < openSet.length; i++) {
        draw_bot(0, openSet[i].x, openSet[i].y);
      }

      for (var i = 0; i < closeSet.length; i++) {
        draw_bot(3, closeSet[i].x, closeSet[i].y);
      }

      if (current == end) { // end the loop when reach the end
        noLoop();
      }

      removeFromArray(openSet, current);
      closeSet.push(current);

      // calculate the f, g, and h for all the neighbors
      var neighbors = current.neighbors;
      for (var i = 0; i < neighbors.length; i++) {
        var neighbor = neighbors[i];

        if (!closeSet.includes(neighbor)){
          // Counting the amount of g/steps
          var tempSteps = current.g + 1;

          if (openSet.includes(neighbor)) {
            if (tempSteps < neighbor.g) {
              neighbor.g = tempSteps;
            }
          }
          else {
            neighbor.g = tempSteps;
            openSet.push(neighbor);
          }

          neighbor.h = heuristic(neighbor, end);
          neighbor.f = neighbor.g + neighbor.h;
          neighbor.previous = current;
        }
      }

      // draw the best path
      var stepCount = 0;
      console.log("DONE!");
      path = [];
      var temp = current;
      path.push(temp);
      while (temp.previous) {
        path.push(temp.previous);
        temp = temp.previous;
        stepCount++;
        draw_bot(2, temp.x, temp.y);
      }
      draw_bot(2, current.x, current.y);
      console.log("Step Count = " + stepCount);
    }
  }
}
```

For our algorithm we first set up a 2d array to allow a place for each tile to store information of its location, using f, g, and h values being its neighboring tile and the previous tile in relation to the current tile. Using the Draw() function, we locate the tile with the lowest f value in the open set and we set the bot to that tile setting it as the current tile. This is when the function starts working and we would then check all neighbouring tiles to see if they belong in the closed set, if not we then would check if they are in the open set. Should the tile pop up to be in the open set, we calculate its f, g, h value and set the previous tile to the current tile, linking up the 2 tiles, this is how the bot creates the paths seen in the program. We then draw the current best path at the end of each loop. Should none of the tiles be in the open set the previous steps are repeated until a new path is found and continues until the final tile is reached. Once the bot has reached the end tile, it calculates the shortest path amongst the other paths, ending the program.

**Code References:**

C. (2017, January 16). Coding Challenge 51.1: A\* Pathfinding Algorithm - Part 1. Retrieved October 16, 2020, from <https://www.youtube.com/watch?v=aKYlikFAV4k>