

# ICS-171 Homework Milestone 1

## — Part 1 — Uninformed Path Planning —

### Scenario

You are employed by a company that makes interactive video games. The game's 2D world has a large number of fixed polygonal obstacles of various shapes and sizes on the screen. You are assigned to write the controller for one of the moving figures on the screen. Whenever a gold coin appears on the screen, your figure is to walk to the coin as rapidly as possible, avoiding all obstacles. Input is the screen locations of your figure, the gold coin, and the locations and shapes of all obstacles. Output is to be the path your figure will follow to the coin.

Interestingly, this is also the same basic problem as robot arm navigation through a crowded workspace. In the robot arm navigation problem, the task is to go from one arm position (the initial position) to some other specified arm position (the goal position) without colliding with any objects in the workspace. In general, this problem is representative of a whole class of related planning problems.

Note: "as rapidly as possible" normally means "optimal," but DFS and BFS are not optimal searches. The DFS/BFS assignment is mainly to set the stage for A\*, which will be optimal. The first assignment is intended provide (1) a point of contrast by which you can measure the improvement that a heuristic search yields (in terms of \*both\* speed and solution quality), (2) a near-term working system (in a rapid prototyping sense) which will be easier to extend to A\*, and finally (3) experience in working with the general search algorithm.

For Part 1 it is sufficient to find \*ANY\* path to the goal. You will see that this does not really satisfy the intent of the video game, because sometimes it will find really "dumb" paths to the goal — it may wander the screen a long time — and it may require an inordinate amount of time to find even "dumb" paths, especially in complex worlds with maze-like obstacles. (Your test cases should include worlds like this, of course.) Part 2 will fix these problems.

### Methodology

Due to space and time constraints, it does not make sense to precompute all possible paths from all possible positions of your figure to all possible positions of the gold coin. Your controller is expected to run in large virtual worlds where such exhaustive precomputation would be wasteful.

Consequently, you decide to implement your controller as a search procedure. When the gold coin appears, you will conduct a search through the space of possible paths. The result of the search will be a path to the coin.

## Input

In describing the data, observe a convention that the  $x$  axis points from left to right across the screen, with larger  $x$  values to the right of smaller ones. The  $y$  axis points from bottom to top up the screen, with larger  $y$  values above smaller ones. The  $z$  axis completes a right-hand coordinate system, and points out of the screen toward the user. The origin is in the lower-left corner of the screen. Consequently, (1)  $x$  coordinates are always non-negative and increase to the right; (2)  $y$  coordinates are always non-negative and increase to the top; (3)  $z$  coordinates in the plane of the screen are always zero. Coordinates are measured in integer pixels.

The polygonal obstacles are simple, possibly non-convex, and non-intersecting. “Simple” means that the boundary of each polygon never crosses itself. “Possibly non-convex” means that parts of the polygon may be concave, that is, go inward. “Non-intersecting” means that the boundary of each polygon never touches or crosses the boundary of any other polygon, i.e., their pointwise intersection is empty.

Input is:

- A pair of integers giving the  $(x, y)$  of your figure;
- A pair of integers giving the  $(x, y)$  of the gold coin;
- An arbitrary number of simple, possibly non-convex, but non-intersecting polygonal obstacles.
  - Each polygon is described by giving a list of the  $(x, y)$  of its vertices.
  - The vertices are given in order around the boundary of the polygon.
  - The interior of the polygon is always on your left-hand side as you go through the vertices in this order.
  - The list of vertices is terminated by -1, which means to connect the last vertex to the first.

## Output

Output is to be a list of  $(x, y)$  points giving the path your figure will follow. The first point must be your figure’s current location. The last point must be the location of the gold coin. Terminate the list with -1. Going from one point to the next must not collide with any obstacle.

## State Space

A state in this space is a position of your figure. An operator is to move to another position. The initial state is your figure’s starting position. The goal state is the position

of the gold coin. A path through state space is a succession of operators that move your figure from position to position.

## Problem Constraints

You can simplify the problem considerably by exploiting natural constraints. Except for the first and last move (i.e., from your figure's initial position, and to the gold coin), you only need to go from (just outside) one corner of an obstacle to (just outside) another corner (possibly of the same obstacle). From any corner of an obstacle, the possible corners to go to are exactly those that are visible according to line-of-sight. The visible corners (plus the gold coin if it is visible) are the children that result from expanding the node corresponding to the corner your figure is at currently.

## Bonuses.

- (5 pts) Implement a better (= tighter) heuristic than straight-line Euclidean distance.
- (10 pts) Code Dijkstra's single-source shortest path algorithm to find all shortest paths from the start point. Make a table that compares it to Depth First and Breadth First. Can you find a maze on which Dijkstra's algorithm is faster? on which Depth First or Breadth First is faster? Can you characterize the types of mazes on which each is fastest?
- (50 pts) Eight puzzle (see Bonus-1).

# ICS-171 Homework Milestone 1

## — Part 2 — Heuristic Path Planning —

### Scenario

Your marketing people are disturbed. People LOVE THE CONCEPT behind your new “CoinFinder” (TM) software agent. However, sometimes the paths it finds are bizarre, and sometimes it takes too long to find one. They are worried that you will lose market share based on this problem. They ask: Can’t you find higher quality solutions faster?

You reassure them: Of course you can. You have already begun the design and analysis behind the next generation of the software, which will use “Heuristic Path Planning” technology. The next version will find better paths in less time.

They immediately begin to plan the marketing campaign for the new “Smart CoinFinder” (TM) software. They plan to demonstrate that the basic concept remains the same, but the quality and speed have improved. They ask if you can generate data showing the new system out-performing the old.

You tell them that this will be no problem. You have generated several difficult test cases already that are problematic for DFS/BFS, and you anticipate that “Heuristic Path Planning” will work much better on these. Also, based on your experience with BFS/DFS, you are confident that you can construct new test cases that will show clear improvement.

### Admissible heuristic.

You must choose an admissible heuristic for some of your searches. For purposes of the homework, you may choose any admissible heuristic you like. The Euclidean straight-line distance is a natural and obvious candidate.

However, during your design and analysis, you note that other heuristics may be better. For example, if the straight-line path to the goal intersects a polygon, obviously your figure must go around it; in this case, the two-line path to the closest extrema of the polygon and then to the goal is also admissible, and provides a better estimate.

In general, tighter lower bounds result in a lower branching factor and therefore slower exponential build-up, but are more complicated and slower to compute. This results in an engineering trade-off that must be made during the design and analysis phase.

### Assignment.

You must code three search strategies:

- Uniform Cost (queue sorted by  $g(n)$  )
- Greedy Best First (queue sorted by  $h'(n)$  )
- A\* (queue sorted by  $f'(n) = g(n) + h'(n)$  )

Compare and contrast their performance with each other and with DFS and BFS, on mazes of different types. Make a table that shows problem size vs. search speed for the different searches, across a diverse and representative selection of mazes.

Can you find a set of mazes such that a different search algorithm is fastest on each maze? Can you characterize the types of mazes on which each is fastest?

## Bonuses.

- (5 pts) Implement a better (= tighter) heuristic than straight-line Euclidean distance.
- (10 pts) Code Dijkstra's single-source shortest path algorithm to find all shortest paths from the start point. Make a table that compares it to Best First and A\*. Can you find a maze on which Dijkstra's algorithm is faster? on which Best First or A\* is faster? Can you characterize the types of mazes on which each is fastest?
- (50 pts) Eight puzzle (see Bonus-1).