Group 7, Project 4 Report

Michael Castleman, William Kwok, Jason Wong

Email: {mlc67, wck9, jaw79}@columbia.edu

Contents

1 Introduction **1**

2 Follow The Yellow-Brick Road **1**
2.1 Finding The Points 2
2.2 Connecting The Dots 4

3 Help Me, I'm Melting! **6**
3.1 Refining the Polygon 6

4 The Man Behind the Curtain **7**

1 Introduction

We present "Dorothy and Toto," a player for the *Avoid the Obstacles* game. This player was quite successful: the only player which consistently did better than we did used large parts of our codebase!

Our player operates in two steps: first, it attempts to find a path to follow, and then it attempts to find the largest shape which can be moved along that path. We do not currently consider rotations.

The player, as its name suggests, can be explained through metaphors to L. Frank Baum's well-known *Wizard of Oz*[1]; quotes from that work are interspersed throughout this text. At least one other player used Oz-like tactics (even if not consciously): `Group6PlayerHaHa` uses the tactic of clicking its heels and saying, "There's no place like home!" However, our player has not only a brain but also a heart: it does not cheat.

2 Follow The Yellow-Brick Road

> "The road to the City of Emeralds is paved with yellow brick," said the witch, "so you cannot miss it. When you get to Oz do not be afraid of him, but tell your story and ask him to help you. Good-bye my dear."[1]

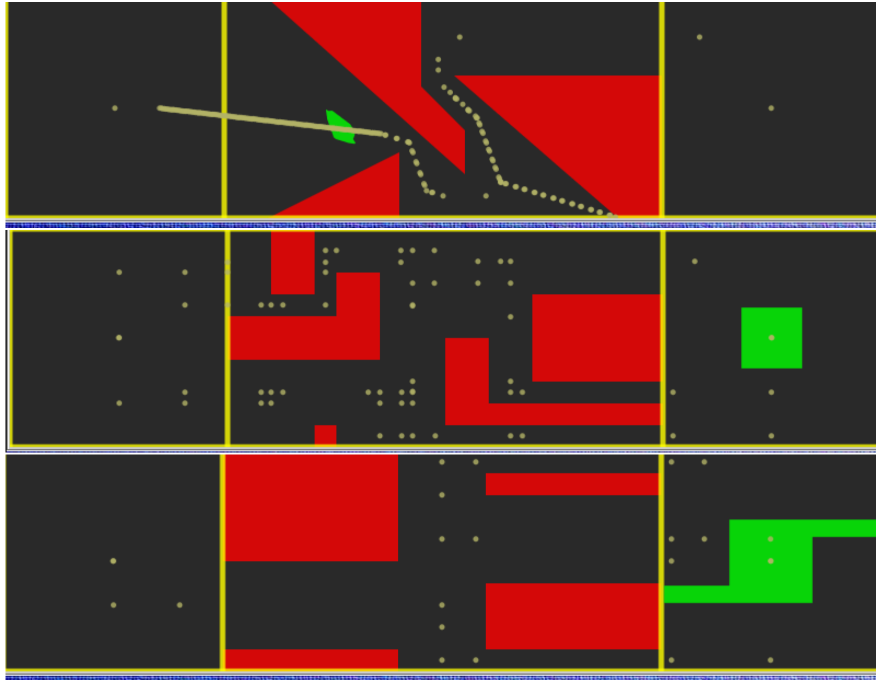

Figure 1: Some obstacle courses, and the associated set of points.

Just as Baum's characters followed a road of yellow brick towards their goal, so too does our player follow a path of yellow lines towards its. This path is generated by first finding a set of critical points on the board which we might like to visit and then selecting a path among these points which we believe will allow us to get a good score.

2.1 Finding The Points

We broke down the problem into several sections, with the ultimate goal of moving the largest polygon through the obstacle. From some analyzing of the problem, we came to realize that we needed to create a graph model from which we can get path(s) for our polygon to move through. In order to create this graph model, we needed nodes or points at certain spots.

By looking at an obstacle, one can easily place nodes of the graph, and then draw it. Telling the computer to do this simple task turns out to be quite a challenge. To get points for the graph, we decided use a grid approach. Basically, we lay down a grid full of points where there isn't an obstacle. However, we don't want all those points, only certain points that are semi-centered between obstacles. To do this we used a merge algorithm to combine (merge) points near each other. Each call to the merge function does one "level" of merging. We keep merging until there are no more "group" of points. We also keep an

incremental counter at each point, which tells us not only that it's a point, but a distance representation of how far the point is from an obstacle. This later turns out to be useful in choosing the path.

The merging ideally would create points throughout the obstacle course, equidistant from the nearest obstacles. It would however create a line of points that would only complicate matters for out path generating algorithm, so we did more "cleaning" and got rid of more unnecessary points. The result of the merging and cleaning was ideal in generating a path on any obstacle for the largest circle to get through. However, even when we changed the circle to the largest square/rectangle that could fit through out path, the path may not necessarily be the ideal path for the largest polygon to get through, even without rotations. The path we generated "cut the corners" on certain obstacles. To alleviate this problem, we added strategic points that would help the path go around corners of obstacles.

Using a grid of points may not seem like such a good way to get nodes for the graph. However, given a nice gigahertz computer and a CPU time of one minute, we were able to set the grid size of 500×2000 points. This turns out to be accurate enough for our needs within the tournament, with an accuracy threshold of approximately 0.002.

Some examples of our point-finding process are shown in Figure 1. Here is a further, more explicit, example of the merging process. A 0 represents an obstacle; numbers greater than zero are merged points, and underscores represent the ignored, empty space.

```
createGrid():
_ _ _ _ _ _ _ 0 0 0 0
_ 1 _ 1 _ 1 _ 0 0 0 0
_ _ _ _ _ _ _ 0 0 0 0
_ 1 _ 1 _ 1 _ 0 0 0 0
_ _ _ _ _ _ _ _ _ _ _
_ 1 _ 1 _ 1 _ 1 _ 1 _
_ _ _ _ _ _ _ _ _ _ _
_ 1 _ 1 _ 1 _ 1 _ 1 _
_ _ _ _ _ _ _ _ _ _ _
_ 1 _ 1 _ 1 _ 1 _ 1 _
_ _ _ _ _ _ _ _ _ _ _
_ 1 _ 1 _ 0 0 0 0 0 0
_ _ _ _ _ 0 0 0 0 0 0

merge() once:
_ _ _ _ _ _ _ 0 0 0 0
_ _ _ _ _ _ _ 0 0 0 0
_ _ 2 _ 2 _ _ 0 0 0 0
_ _ _ _ _ _ _ 0 0 0 0
_ _ 2 _ 2 _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _
```

```
_ _ 2 _ 2 _ 2 _ 2 _ _
_ _ _ _ _ _ _ _ _ _ _
_ _ 2 _ 2 _ 2 _ 2 _ _
_ _ _ _ _ _ _ _ _ _ _
_ _ 2 _ _ _ _ _ _ _ _
_ _ _ _ _ 0 0 0 0 0 0
_ _ _ _ _ 0 0 0 0 0 0

merge() twice:
_ _ _ _ _ _ _ 0 0 0 0
_ _ _ _ _ _ _ 0 0 0 0
_ _ _ _ _ _ _ 0 0 0 0
_ _ _ 3 _ _ _ 0 0 0 0
_ _ _ _ _ _ _ _ _ _ _
_ _ _ 3 _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _
_ _ _ 3 _ 3 _ 3 _ _ _
_ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _
_ _ 2 _ _ _ _ _ _ _ _
_ _ _ _ _ 0 0 0 0 0 0
_ _ _ _ _ 0 0 0 0 0 0
```

2.2 Connecting The Dots

Once we have identified the critical points as described above, it then remains to connect them into some sort of path which we can follow.

The first step of this process is to construct the Relative Neighborhood Graph (RNG) of our points. This graph was first described by Toussaint[4], in an attempt to model the way which humans might naturally connect points. It draws an edge between two points p_i and p_j if, $\forall p_k \neq p_i, p_j,\ d(p_i, p_j) \leq \max[d(p_i, p_k), d(p_j, p_k)]$, where d represents the distance function between two points. It is perhaps interesting that the RNG of a set of a points is always a superset of the minimal spanning tree and a subset of the Delaunay triangulation.

There is an $\mathcal{O}(n \log n)$ algorithm[3] for finding the RNG of a set of points in the Euclidean plane. This running time can be proven to be a lower bound for the Euclidean case. However, we do not use this algorithm for two reasons: firstly, because it is difficult to implement, and secondly, because our points do not obey a pure Euclidean distance metric: sometimes, two points which seem very close may be separated by an obstacle, in which case no line should be drawn between them and so we set their distance to infinity. Instead, we use the naïve $\mathcal{O}(n^3)$ algorithm described in Toussaint's paper. The performance of this algorithm seems to be acceptable for our purposes.

After computing the RNG, we draw it on the screen. So, we have for example, the graphs shown in Figures 2 and 3. Note the yellow brick road shown in Figure

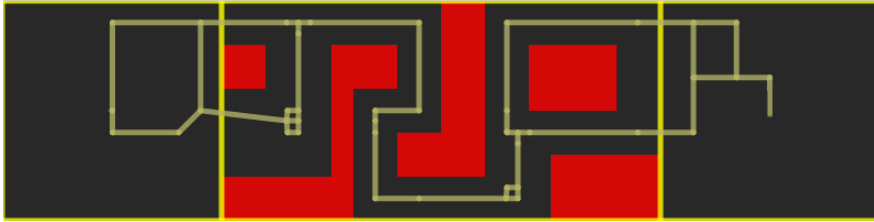

Figure 2: The relative neighborhood graph of `Group6Maze.game`.

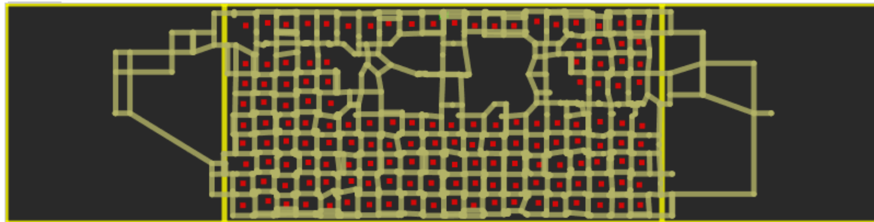

Figure 3: The relative neighborhood graph of `tourn3.game`.

3!

The RNG, for us, represents the set of all allowable edges. It contains an edge between two points only if they are close enough that we want to be able to go directly from one to the other. It then remains for us to find the **best** path among all these edges.

There is an obvious method for generating a valid path in our graph: simply perform a depth-first search from a vertex in the starting region to a vertex in the destination region. This is obviously not, in general, going to give us the best path. However, the merge algorithm described above tells us more than just the location of each point: we know the number of steps taken in the merge. The greater the number of such steps, the more empty space surrounding the point, thus allowing us to pass a larger obstacle through the point.

We can exploit this knowledge to generate an "emptiness" value for each edge by taking the smaller of the two merge numbers for the points which the edge connects. This value gives an estimate for the size of the largest shape which can be moved along the edge in question. To find the best path, then, is to find the path with the largest minimum emptiness value.

To achieve this, we loop, removing the worst (smallest-emptiness) edge as long as the graph remains connected. (More precisely, we need only ensure that we have a path from the start node to the end node. If some other component of the graph becomes disconnected, we do not have to be concerned.) When we can remove no more edges while maintaining a valid path, we simply perform a depth-first search, giving us the path to follow—our yellow-brick road.

3 Help Me, I'm Melting!

> "See what you have done!" [the Witch] screamed. "In a minute I shall melt away."
>
> "I'm very sorry, indeed," said Dorothy, who was truly frightened to see the Witch actually melting away like brown sugar before her very eyes. [1]

Once we have the path to follow, it remains to select the shape which we should take along our path. Here we take our cue from Baum's Wicked Witch of the West, and melt. Our melting, however, does not result in a pile of brown sugar but rather in a polygon which we can take along our path.

Our melting algorithm is simple in concept. First, we have to know the area in which we may travel, the black area shown on the screen. To find this, we simply start by considering the rectangle with corners $(0, 0)$ and $(4, 1)$, and then subtract the union of the red obstacles. Then, we take the unit square, start it at the first point on our path, and move it along our chosen path in small, discrete steps. At each step, we intersect our current notion of the resultant figure with the allowable area generated above. This yields the largest polygon which can move along our selected path.

In order to avoid implementing our own computational geometry library for this algorithm, we make heavy use of the `java.awt.geom` package, particularly the `Area` and `GeneralPath` classes.

3.1 Refining the Polygon

Except in unusual cases, the polygon which results from this algorithm will not be exactly correct. First, because we test the polygon only at discrete locations, there may often be small protrusions which would cause our polygon to fail when subjected to the more exact intersection-testing methods of the game engine. Secondly, the generated polygons sometimes had as many as 800 edges, violating the requirement that we have a maximum of 100. To combat both of these problems, we first simplify the polygon, and then shrink it.

A number of sophisticated algorithms (e.g., [2]) exist for polygon simplification. However, a precisely optimal algorithm is not required for our purposes, and so we use the following simple heuristic, which has the added benefit of running in $\mathcal{O}(n)$ time: starting at any point on the graph, we remove edges until we have an edge of length greater than some ε, at which point we keep the resulting edge and proceed around the polygon. This heuristic has the desired effect, as shown in Figure 4.

After this, we need to "shrink" our polygon, so it will not collide with any walls. A naïve shrinking algorithm might simply scale each point towards the center of the polygon. However, this algorithm does not work in the case of a non-convex polygon, as it might sometimes move edges onto obstacles rather than away from them. So, our algorithm instead considers each corner, determines the angle of the line which bisects that corner, and moves the corner

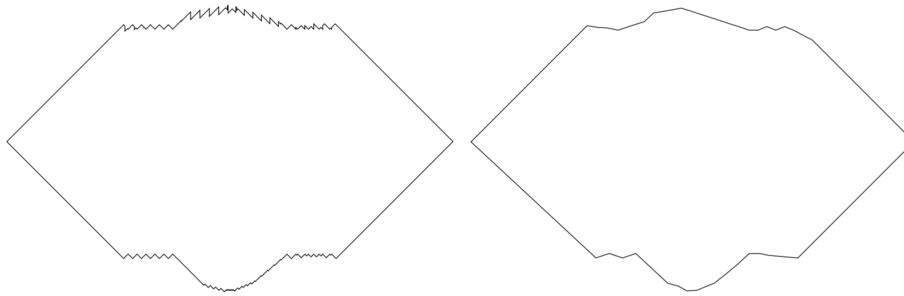

Figure 4: The generated polygon for `Group6CharlieBrown.game`, before and after simplification.

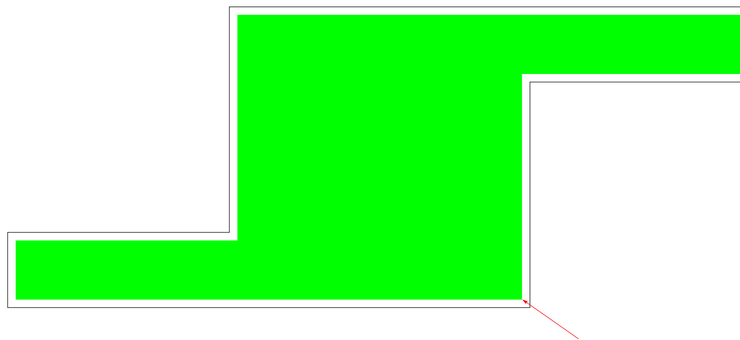

Figure 5: An example of shrinking the polygon used for `nottoohard.game`.

towards the interior of the polygon by the desired amount. An example of this is shown in Figure 5. The black outline here is the polygon before shrinking, while the green shape is after shrinking.

There is, however, one caveat. Occasionally, some very sharp angles will cause our polygon-shrinking algorithm actually to move one of the points **outside** of the original polygon! This occurred with `tilted.game`, shown in Figure 6. The original polygon is the black polygon, which is then reduced to the green outline. The solution, then, is just to take the largest contiguous area whenever we do not have a simple polygon. This is shown as the red area, which clearly lies wholly within the black outline.

4 The Man Behind the Curtain

> **Cowardly Lion**: Alright, I'll go in there for Dorothy. Wicked Witch or no Wicked Witch, guards or no guards, I'll tear them apart. I may not come out alive, but I'm going in there. There's only one thing I want you fellows to do.
>
> **Tin Woodsman, Scarecrow**: What's that?

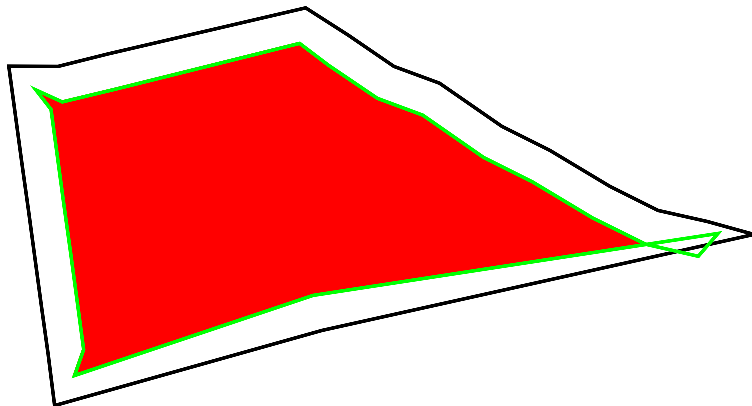

Figure 6: A caveat to our shrinking algorithm, demonstrated on `tilted.game`.

Lion: Talk me out of it.[5]

Examining the results of the tournament, we find that our player successfully navigated 19 out of the 20 obstacle courses. The only one on which we fail is due to a bug in which we throw an exception in certain unusual circumstances. In addition to navigating all these boards, we do so with large shapes. In no case was our shape smaller than 62% of the smallest shape found; our closest competitor for this statistic (`Group4Player1`) achieves only 15%.

Calculating the average score over all 20 boards, Dorothy and Toto place first with a score of 0.774292. The closest competitor was the "Chip-Off Player"—which used a lot of our path generating code.

See the attached spreadsheet for more detailed results.

References

[1] Baum, L. Frank. *The Wonderful Wizard of Oz*. 1900. `ftp://sailor.gutenberg.org/pub/gutenberg/etext93/wizoz10.txt`.

[2] Eu, David, and Godfried T. Toussaint. "On Approximating Polygonal Curves in Two and Three Dimensions." *CVGIP: Graphical Models and Image Processing* 56 (1994): 231-246.

[3] Supowit, Kenneth J. "The Relative Neighborhood Graph, with an Application to Minimum Spanning Trees." *Journal of the ACM* 30 (1983): 428-448.

[4] Toussaint, Godfried T. "The Relative Neighbourhood Graph of a Finite Planar Set." *Pattern Recognition* 12 (1980): 261-268. `http://cgm.cs.mcgill.ca/~godfried/publications/relng.ps.gz`

[5] *The Wizard of Oz*. Dir. Victor Fleming. Perf. Judy Garland et.al. MGM, 1939.