

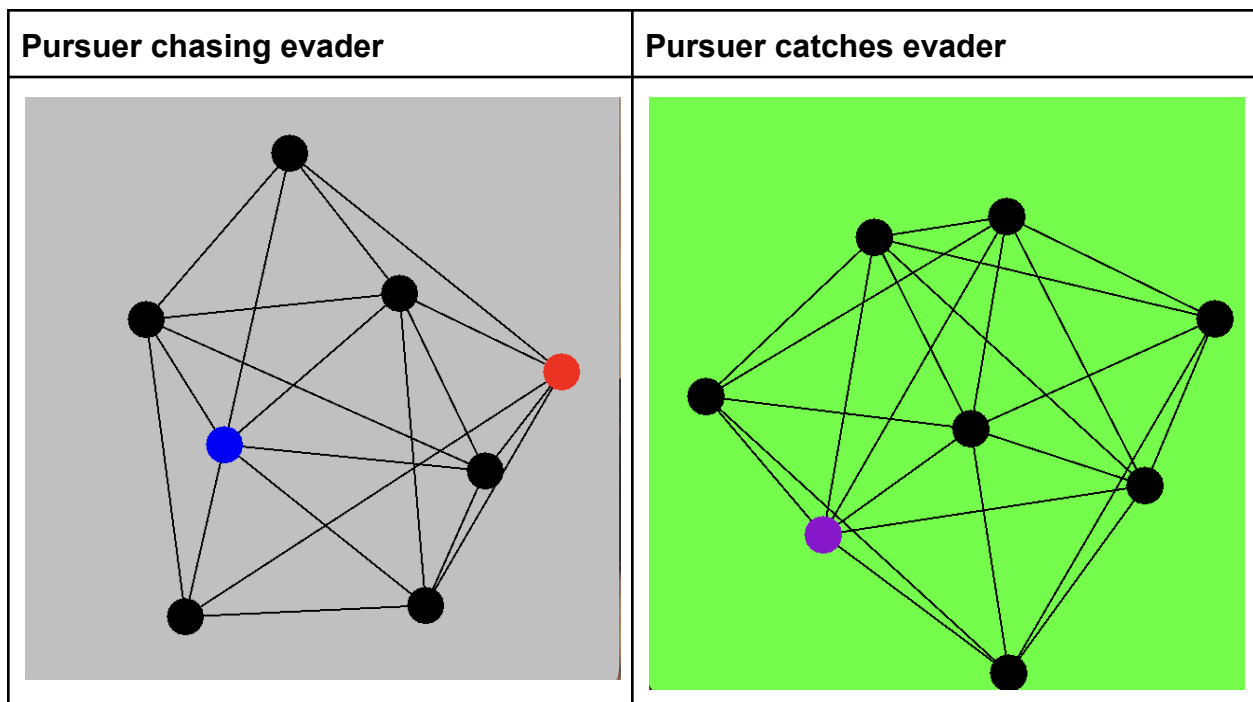
Sam Polyakov
Project 8
5/9/2023
CS231 B

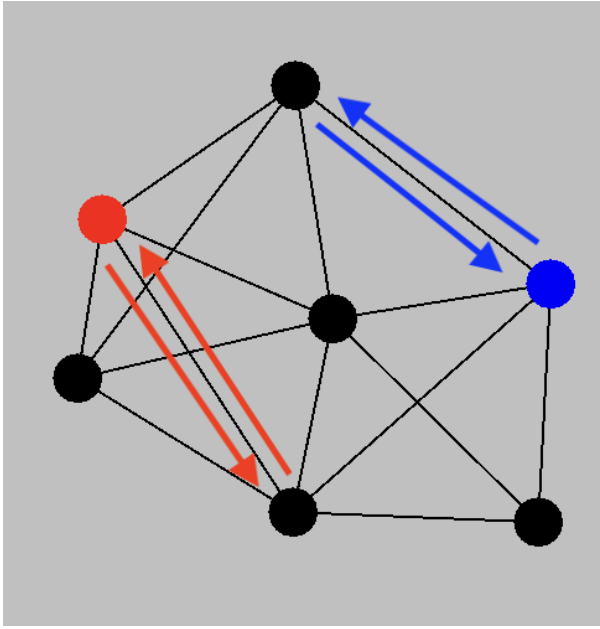
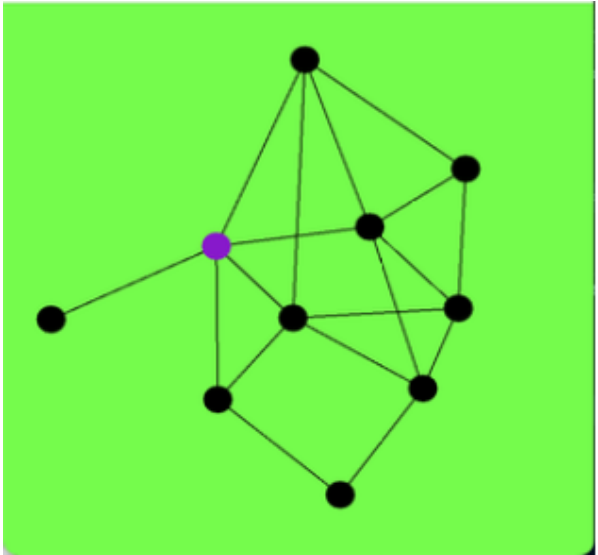
Pursuit Evasion on Graphs

Abstract:

This week, we created a java program that created a grid with a 'pursuer' and an 'evader'. We used java classes and objects, array lists, arrays, methods, loops, abstract classes, inheritance, linked lists, iterators, stacks, BST Maps, Heaps, graphs, and priority queues to build all of the classes. In the end, my program drew a grid with a given number of vertices and edges, and simulated a game of 'tag' between the pursuer and evader. For my extension, I made a smarter algorithm for selecting a starting vertex for both the pursuer and evader.

Results/Exploration:



<p>Pursuer could have caught the evader, but instead indefinitely stays at least one vertex away</p>	<p>Evader could have evaded the pursuer indefinitely, but instead gets captured</p>
	

The above image on the left shows a graph on which the pursuer could have caught the evader, but instead indefinitely stayed at least one vertex away. In this graph, the pursuer and evader are both infinitely ‘bouncing’ between the two vertices shown. If the pursuer simply went to the “middle” vertex at any point, the evader would have no possible way to escape and would thus be caught. It makes sense that this is happening because every time the evader moves, the pursuer simply tries to move to minimize the distance between them. As the pursuer has no additional ‘strategy’ it cannot see that moving toward the middle vertex would trap the evader.

The image on the right shows a graph on which the evader could have evaded the pursuer indefinitely, but instead got captured. In this graph, the evader moves towards the left-most ‘separated’ vertex and gets trapped because of this. It makes sense that this is happening because the evader is simply trying to move as far away as possible from the pursuer, which will often be the ‘separated’ vertex on the left. As the evader has no additional ‘strategy’ it cannot see that moving to this separated vertex will essentially trap it.

Extension:

This week, for my extension I made a smarter algorithm for selecting a starting vertex for both the pursuer and evader. By default, each player was supposed to go to a random vertex and start the game from there.

I changed this so that the pursuer, without any other information, would go to the vertex with the shortest average edge length. This was advantageous because it allowed the pursuer to move away from the “worst” vertex as quickly as possible while also being more likely to start far away from the evader, making the game last longer.

For the evader, I made it so that the default starting vertex without information about the pursuers location was the vertex with the longest average edge length. This was advantageous because it allowed the evader to move far away from the pursuer as quickly as possible, resulting in a longer game.

If the evader knew the pursuers starting position, I made it so that the evader chose the farthest possible vertex from the pursuer. I did this by using Dijkstra's algorithm to find the farthest vertex. I did this as the evader would, obviously, want to start as far as possible from the pursuer.

If the pursuer knew the evader's starting position, I made it so that the pursuer simply chose the vertex that the evader was currently on. This was obviously advantageous for the pursuer as it would always win immediately.

Pursuer No Information Starting Pos.

```
@Override
public Vertex chooseStart() {
    // returns a Vertex for the player to start at and updates the current Vertex to that location.
    // Chooses the vertex with the minimum average edge length
    double minAverage = Double.POSITIVE_INFINITY;
    Vertex tempVertex = new Vertex();
    for(Vertex vertex : graph.getVertices()){
        ArrayList<Edge> edges = vertex.edges;
        double totalDistance = 0.0;
        for(Edge edge : edges){
            totalDistance += edge.distance();
        }
        double averageDistance = totalDistance/edges.size();
        if(averageDistance < minAverage){
            minAverage = averageDistance;
            tempVertex = vertex;
        }
    }
    setCurrentVertex(tempVertex);
    return tempVertex;
}
```

Evader No Information Starting Pos.

```
@Override
public Vertex chooseStart() {
    // returns a Vertex for the player to start at and updates the current Vertex to that location.
    // Chooses the vertex with the maximum average edge length
    double maxAverage = Double.NEGATIVE_INFINITY;
    Vertex tempVertex = new Vertex();
    for(Vertex vertex : graph.getVertices()){
        ArrayList<Edge> edges = vertex.edges;
        double totalDistance = 0.0;
        for(Edge edge : edges){
            totalDistance += edge.distance();
        }
        double averageDistance = totalDistance/edges.size();
        if(averageDistance > maxAverage){
            maxAverage = averageDistance;
            tempVertex = vertex;
        }
    }
    setCurrentVertex(tempVertex);
    return tempVertex;
}
```

Pursuer With Information Starting Pos.	Evader With Information Starting Pos.
<pre>@Override public Vertex chooseStart(Vertex other) { // chooses the closest vertex to the other player to start at return other; }</pre>	<pre>@Override public Vertex chooseStart(Vertex other) { //chooses the farthest vertex from the player to start at HashMap<Vertex, Double> distance = graph.distanceFrom(other); double furthestDst = 0.0; Vertex furthestVtx = new Vertex(); for (MapSet.KeyValuePair<Vertex, Double> vertex : distance.entrySet()){ if(vertex.getValue() > furthestDst){ furthestDst = vertex.getValue(); furthestVtx = vertex.getKey(); } } setCurrentVertex(furthestVtx); return furthestVtx; }</pre>

References/Acknowledgements:

This week, I worked with Dave Boku and used ChatGPT for some help debugging