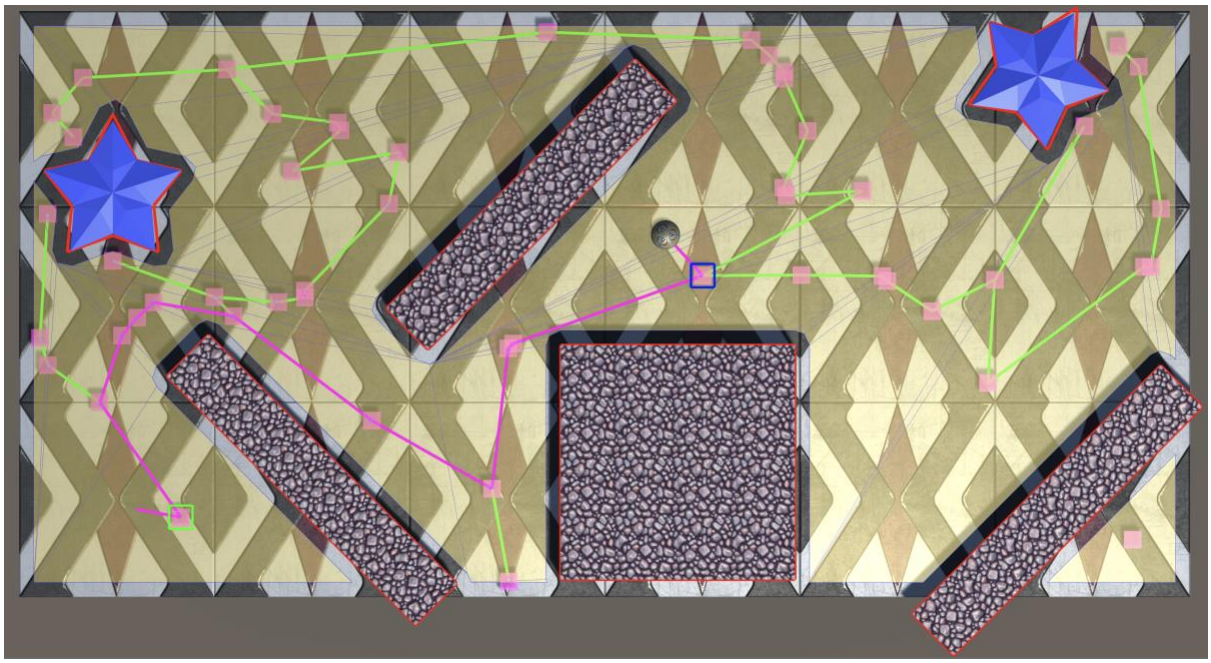
Homework 4: NavMesh Generation

In this homework you will be implementing an algorithm for creating NavMeshes.

A **navigation mesh** (or **navmesh**) is a method of creating a discretized space representation of a navigable area. A navigation mesh is a set of convex polygons overlaid on an environment such that the area within each polygon is guaranteed to be obstacle-free. The variable size and dimensions of navmesh polygons provide efficient representation of space as compared to the uniform structure of a grid lattice. It also captures more usable information about the environment than a path network alone. The convexity of the polygons is important because an agent within the area of a polygon can move to any other point within without crossing a boundary (e.g. colliding).



When two navmesh convex polygons are adjacent to each other (i.e., they share an edge), that edge can be thought of as a "portal"—an invisible door—from one safe navigation region to another. Connecting adjacent convex polygons into a network of safe paths results in a path graph through which an agent can travel between connected navigable locations.

There are various ways to convert a navmesh into a graph with path locations (e.g. a path network). A popular approach, and one that reduces the number of nodes, is to use the centroid of each navmesh polygon.

In this assignment, you will write the code to generate a navmesh for an arbitrary environment as well as an accompanying path network from the navmesh.

There are three main challenges of the assignment:

1. Form adjacent triangles that cover all the navigable space
2. Merge triangles into larger, more efficient polygons that are also convex

3. Create a path network from the navmesh

What you need to know

Please consult **homework 1** for background on the Game Engine. In addition to the information about the game engine provided there, the following elements will be used.

CreateNavMesh

This file contains a Create() method that you must implement. Additionally, be sure to change the student name string. Considerable scaffolding and comments are provided, which will aid in development of your solution.

Create(): Creates a navmesh and pathNetwork (associated with navmesh)
canvasOrigin: bottom left corner of navigable region in world coordinates
canvasWidth: width of navigable region in world dimensions
canvasHeight: height of navigable region in world dimensions
obstacles: a list of Polygons that are obstacles in the scene
agentRadius: the radius of the agent
offsetObst: these are the polygons that are expanded/offset. They are passed out for visualization purposes. Provided code already addresses this.
origTriangles: out param of the triangles that are used for navmesh generation. These triangles are passed out for visualization.
navmeshPolygons: out param of the convex polygons of the navmesh (list). These polys are passed out for visualization
pathNodes: out param: a list of graph nodes, centered on each navmeshPolygon
pathEdges: out param: graph adjacency list for each graph node. corresponding index of *pathNodes* to match node with its edge list. All nodes must have an edge list (no null list) entries in each edge list are indices into pathNodes

The scaffolding and inline comments of CreateNavMesh.Create() gives specific guidance and recommendations on other classes and methods to use (including methods at the top of the file).

Instructions

Download the project from Github and open in Unity. Open the Navmesh scene and the CreateNavMesh.cs file. Follow the comments in CreateNavMesh.Create() to build a working navmesh generator. **Don’t forget to set the student name to your name.**

Grading

We will grade your solution based generally on three criteria:

- **Coverage:** The navmesh should cover the entire navigable area of the map, no more and no less.

- **Reachability:** The path network should include pathNodes at the centroid of each navmesh polygon and pathEdges should span across portal edges of adjacent navmesh polygons. Valid edges must go in both directions and there must not be duplicates of the same edge. You do not need to test for obstacle distance from edges like in HW2. Instead only use the navmesh polygons to determine node connectivity.

- **Mesh optimization:** Your solution should effectively merge navmesh triangles (or more sided polygons) together resulting in a reduction of pathNetwork complexity

---

Submission

To submit your solution, upload your modified CreateNavMesh.cs (did you remember to change the student name string to your name?). All work should be done within this file. Helper methods within the same file are fine.

You should not modify any other files in the game engine.
DO NOT upload the entire game engine.