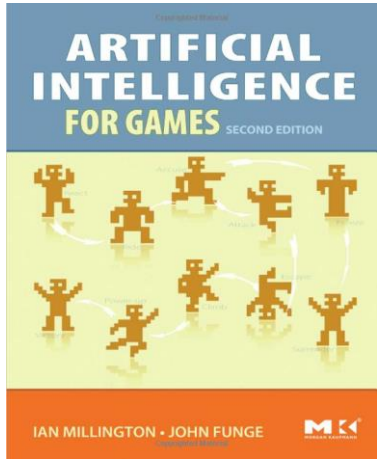
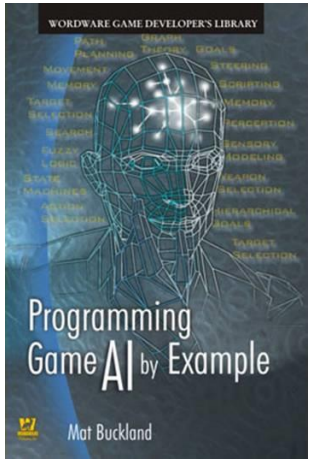


Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

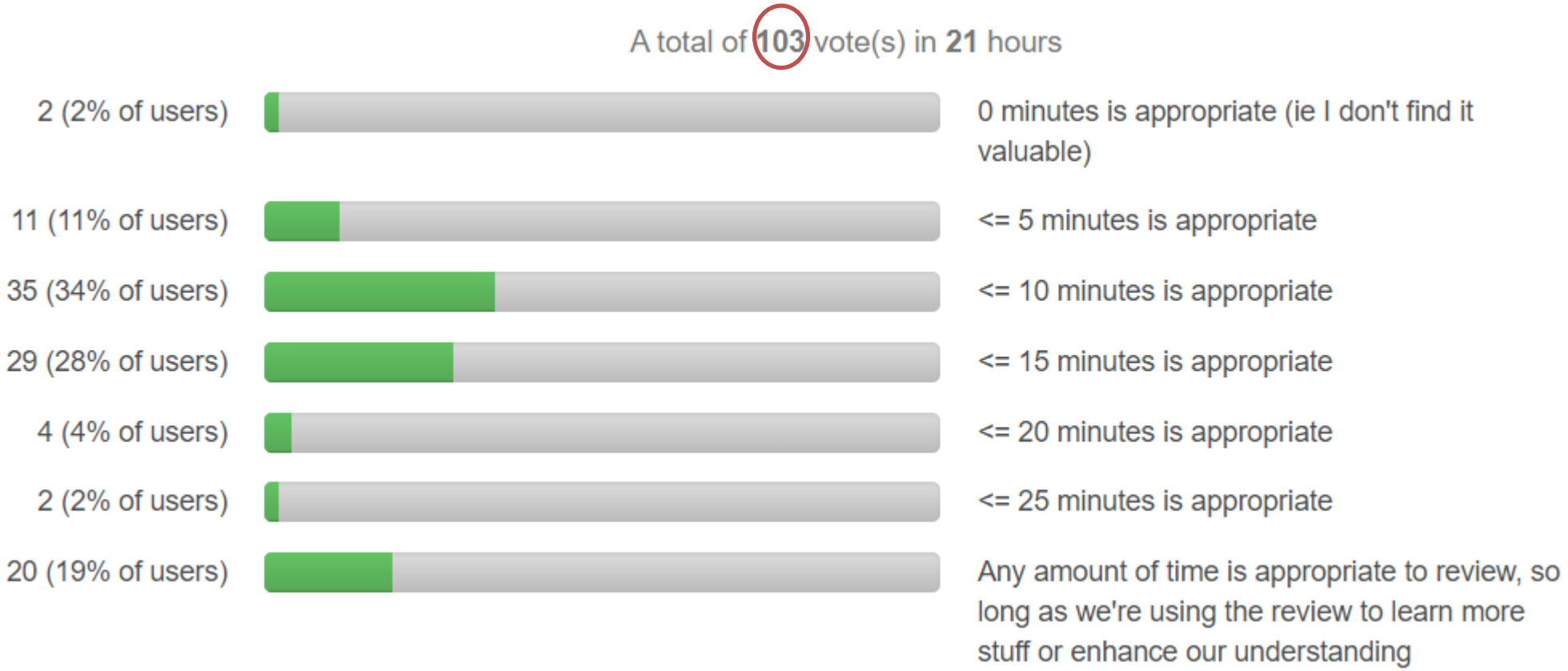


"I may not have gone where I intended to go, but I think I have ended up where I needed to be." –Douglas Adams

"All you need is the plan, the road map, and the courage to press on to your destination." –Earl Nightingale

Announcements

- HW2 (path network) due Sunday night, January 28 @ 11:55pm
- HW2 is more challenging than HW1. Start early!
  - You must write the code to generate the path network, as a set of edges between path nodes. An edge between path nodes exists when (a) there is no obstacle between the two path nodes, and (b) there is sufficient space on either side of the edge so that an agent can follow the line without colliding with any obstacles.
  - We will test path network using a random-walk navigator that moves the agent to the nearest path node and then follows a randomly generated path---sequence of adjacent path nodes.

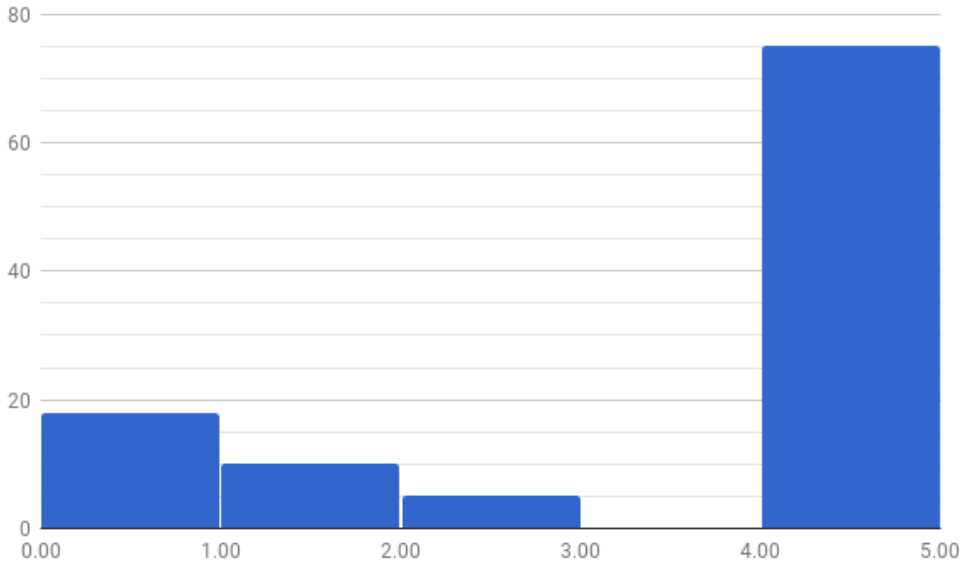

A total of 103 vote(s) in 21 hours
2 (2% of users)
0 minutes is appropriate (ie I don't find it valuable)
11 (11% of users)
<= 5 minutes is appropriate
35 (34% of users)
<= 10 minutes is appropriate
29 (28% of users)
<= 15 minutes is appropriate
4 (4% of users)
<= 20 minutes is appropriate
2 (2% of users)
<= 25 minutes is appropriate
20 (19% of users)
Any amount of time is appropriate to review, so long as we're using the review to learn more stuff or enhance our understanding

Verification of Participation (Deadline: Fri. Jan. 26, 2018 at 4:00 p.m.): Verification of Participation is a process whereby instructional faculty report to the Registrar's Office and the Office of Scholarships & Financial Aid whether they have students enrolled in their classes who are not engaged with the course. This verification by faculty is a Federal Title IV requirement (visit https://registrar.gatech.edu/faculty-and-staff/verification-of-participation for more information about the requirement and what constitutes participation). This includes all credit-bearing course hours – undergraduate and graduate; research hours included – that an instructor is responsible for. The window is open Jan. 22-26. Please go to the following site to verify your student's participation. http://verifyparticipation.gatech.edu/.

HW1: Avg 3.8/5

```
# Creates a grid as a 2D array of True/False values (True = traversable). Also returns the dimensions of the grid as a (columns, rows) list.
def myCreateGrid(world, cellsize):
        grid = None
        dimensions = (0, 0)
        ### YOUR CODE GOES BELOW HERE ###


        ### YOUR CODE GOES ABOVE HERE ###
        return grid, dimensions
```



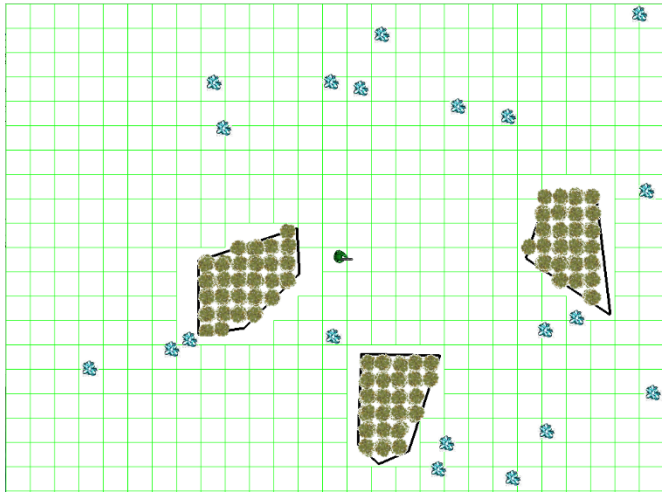
- http://game-ai.gatech.edu/sites/default/files/documents/documentation/object-hierarchy.png
- http://game-ai.gatech.edu/sites/default/files/documents/documentation/gaige-object-documentation.pdf

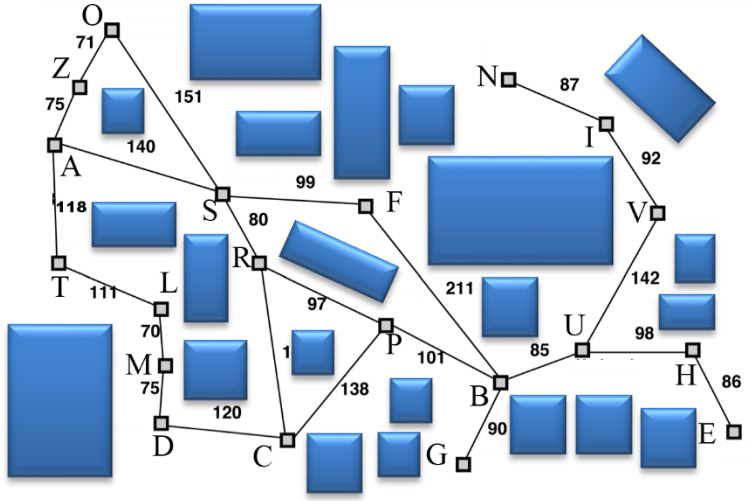
HW1: Grid Generation Hints

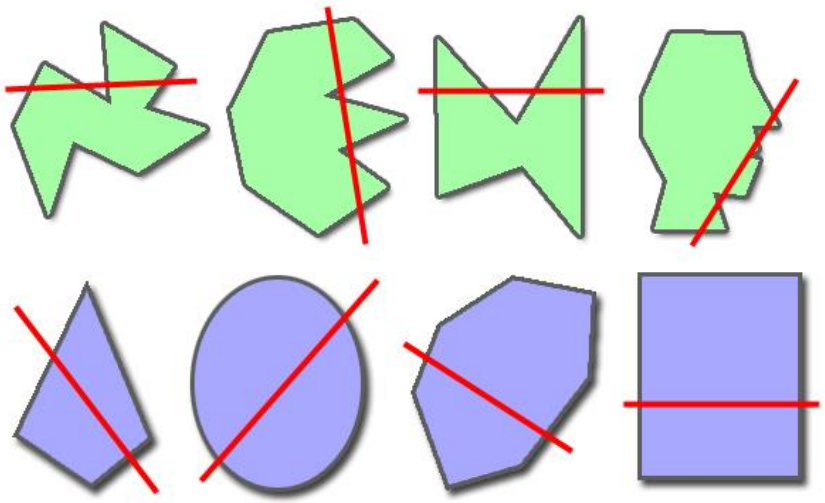
- Verify no world line goes through grid lines (rayTraceWorld)
- Verify no obstacle point within grid cell (pointInsidePolygonPoints, for each obst.)
- Please check the following sections
  - “Miscellaneous utility functions”
  - “Hints”

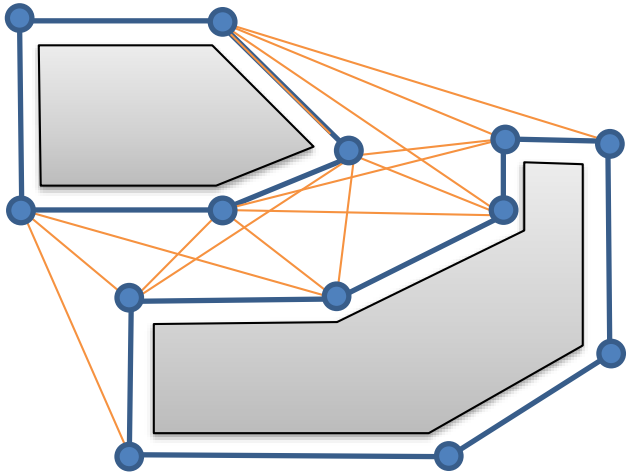
PREVIOUSLY ON…

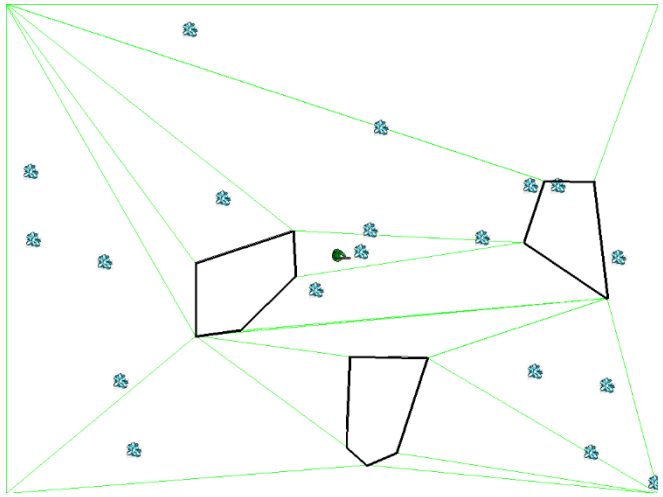
Modelling and Navigating the Game World

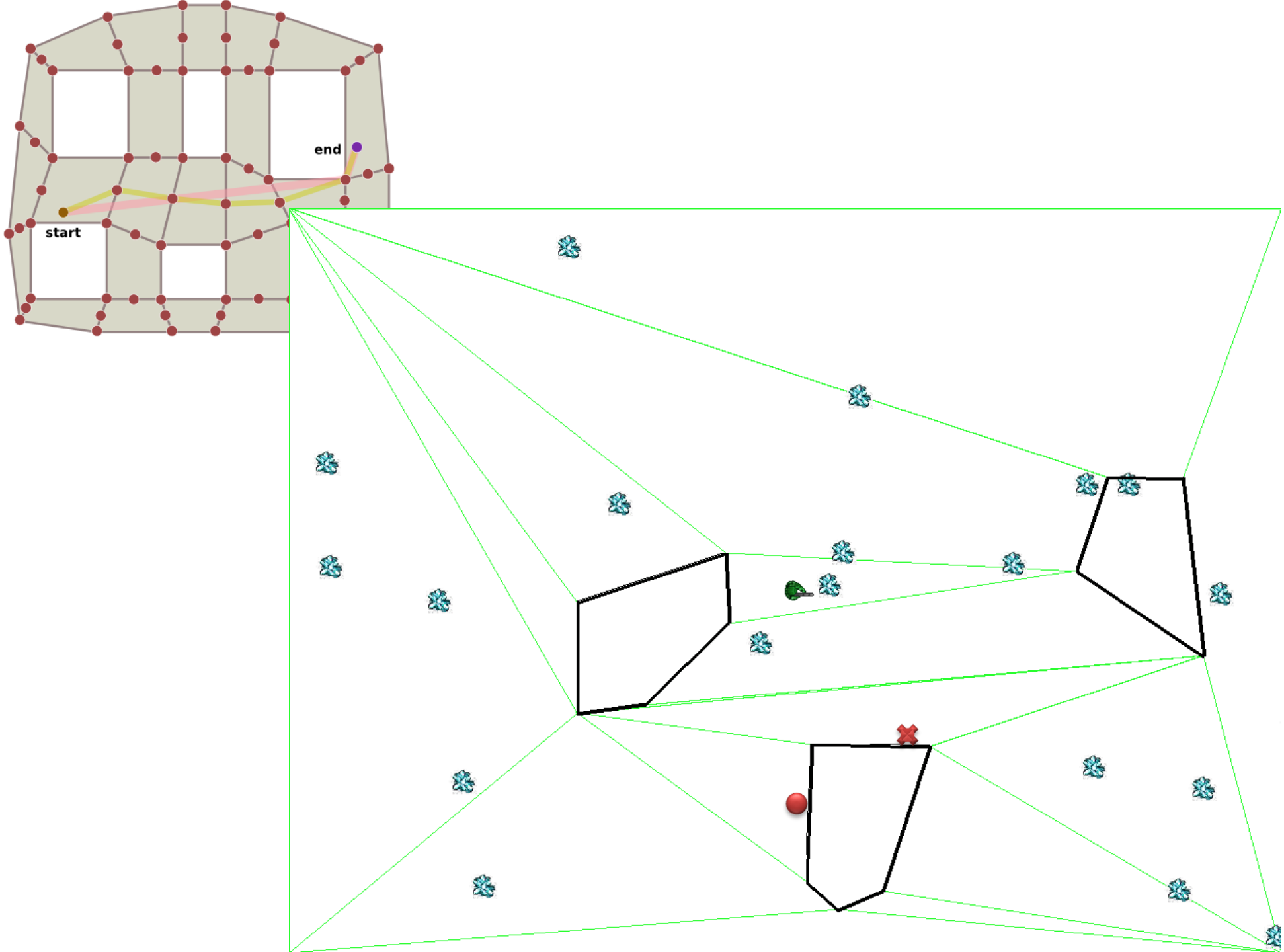


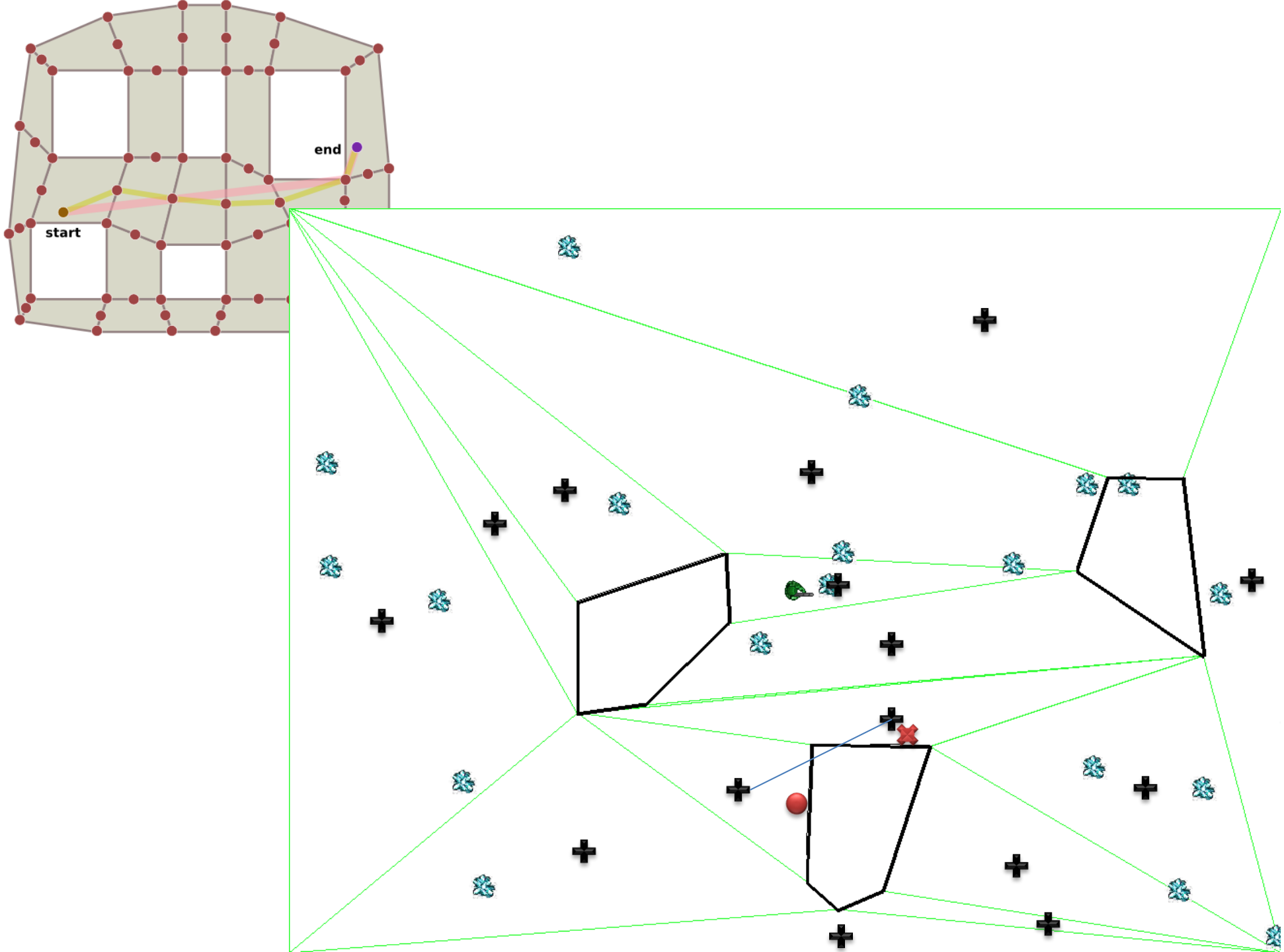


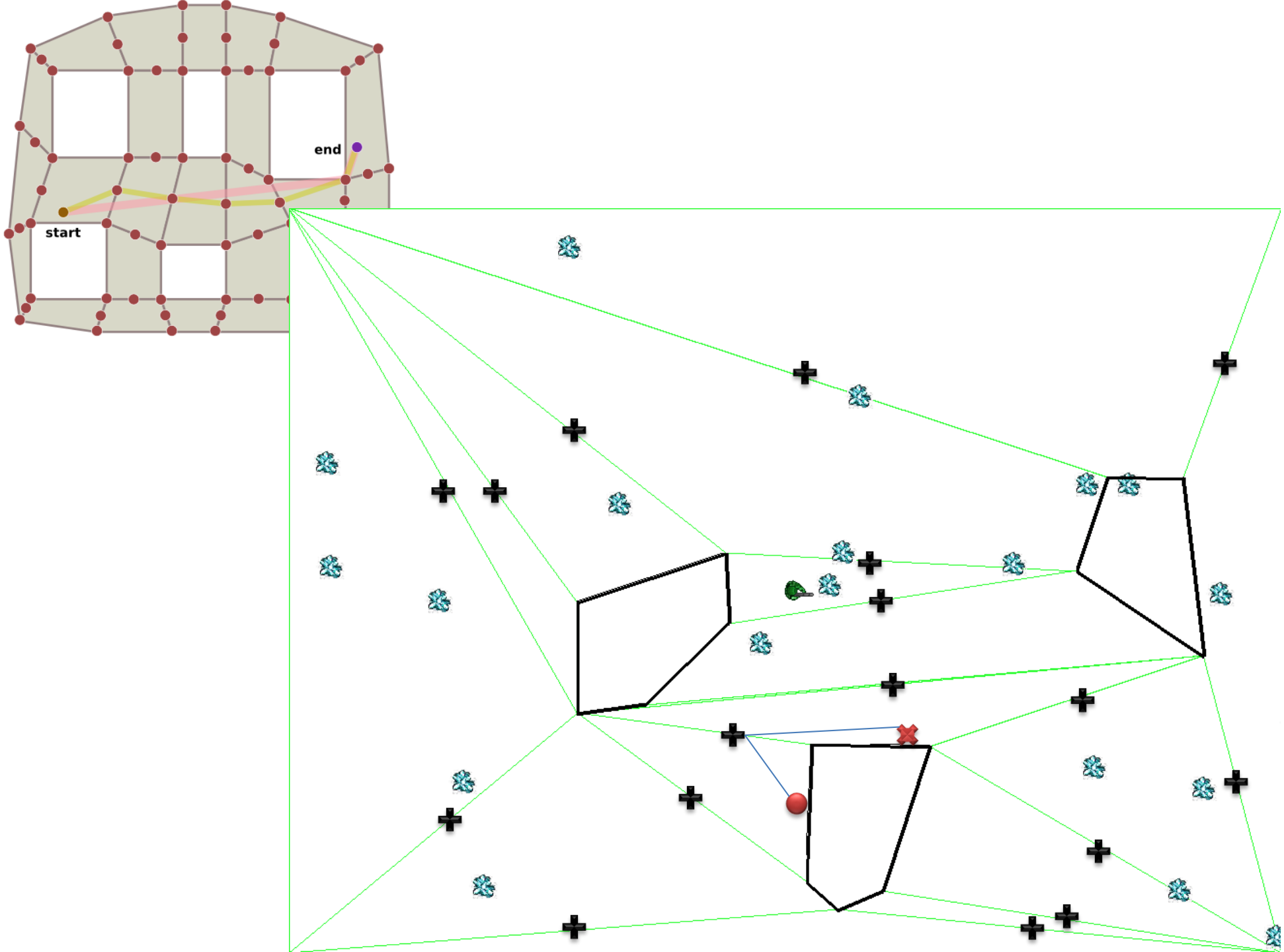


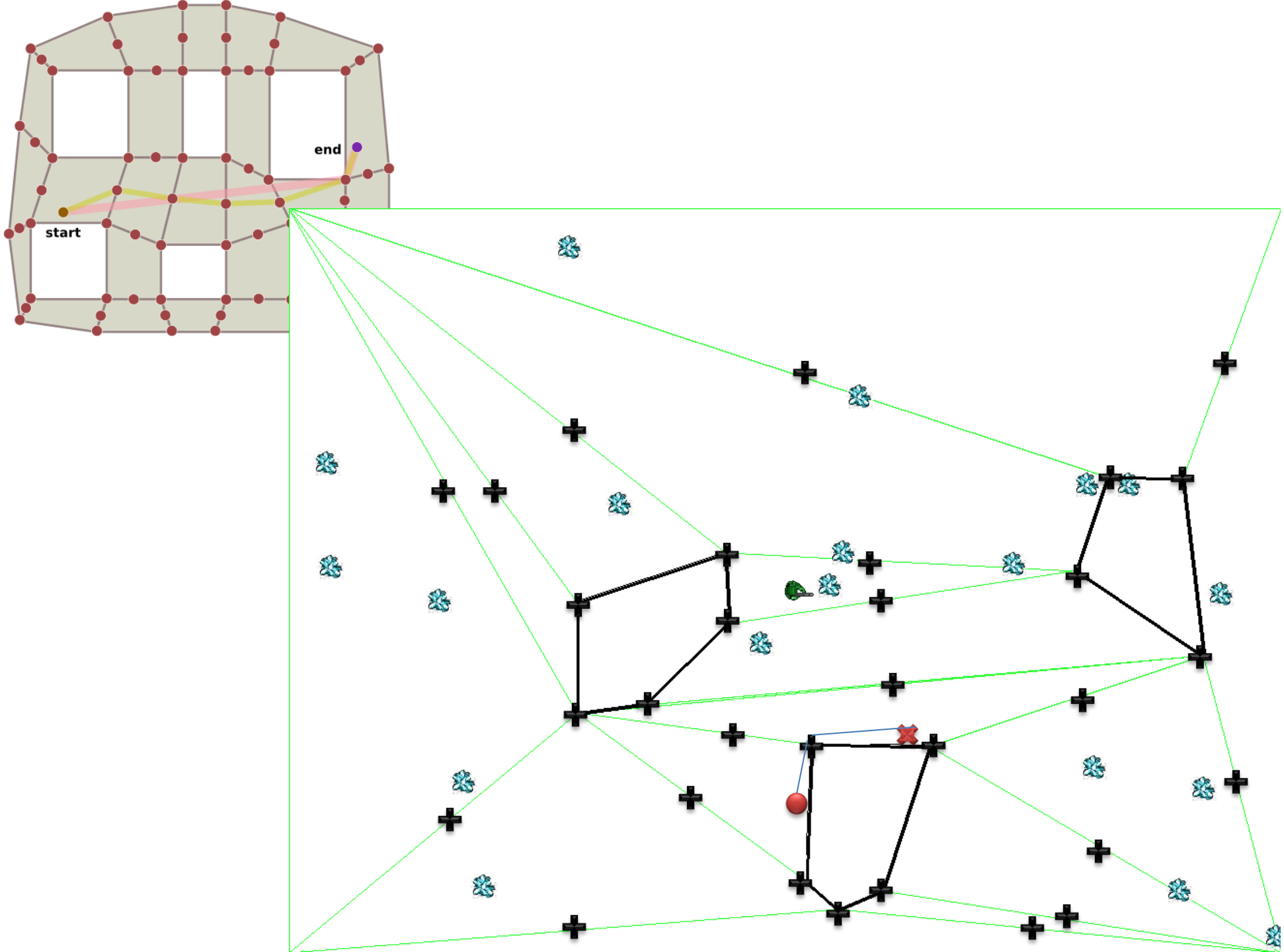


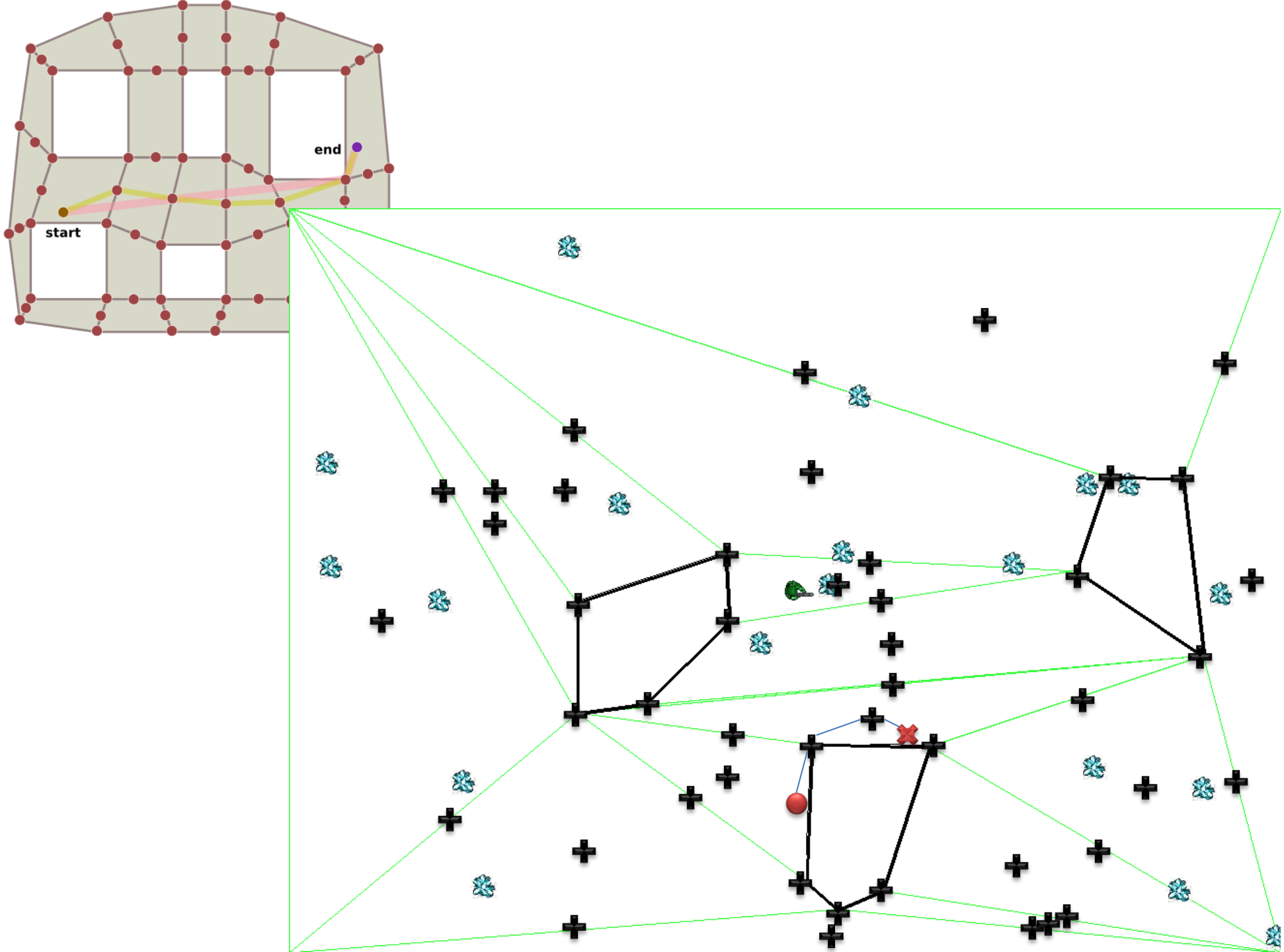



end
start


end
start


end
start


end
start

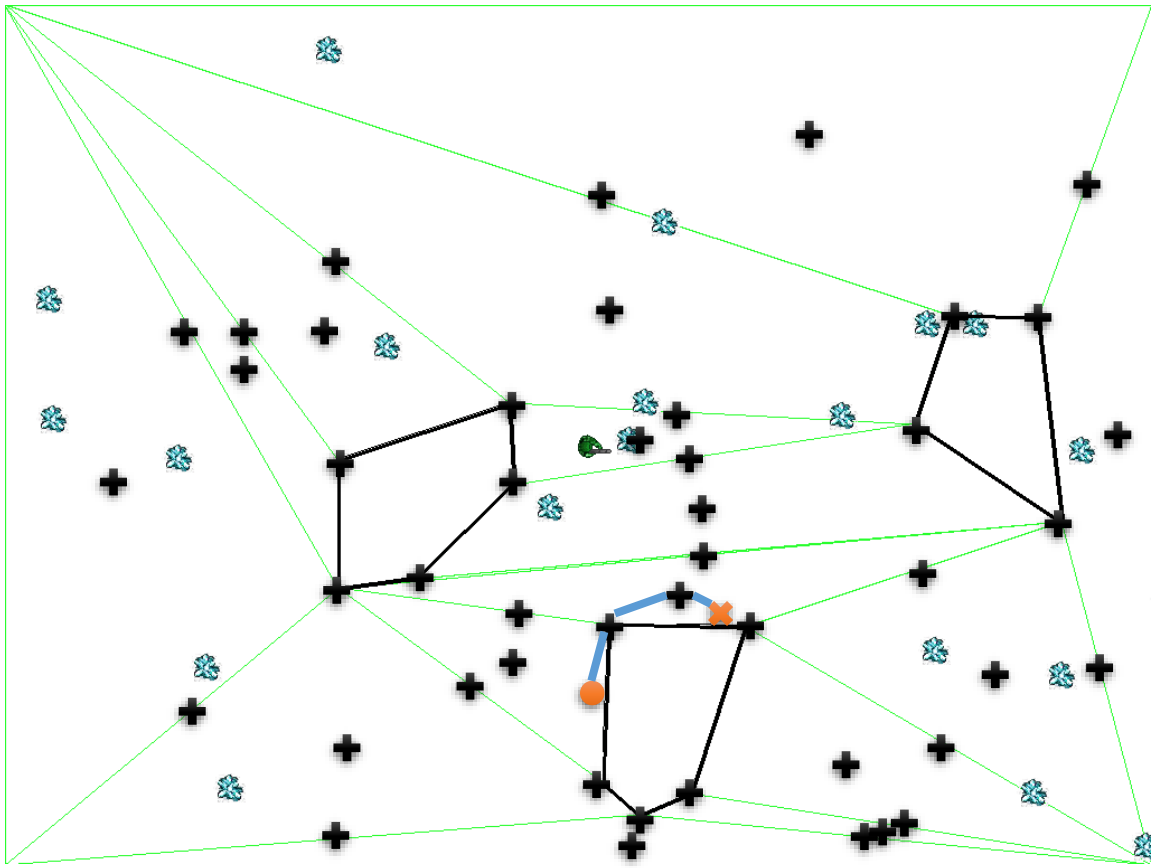

end
start

N-2: Grids, Path Networks

1. What's the intuition behind iterative deepening?
2. What are some pros/cons of grid navigation?
3. What are some benefits of path networks?
4. Cons of path networks?
5. What is the flood fill algorithm?
6. What is a simple approach to using path navigation nodes?
7. What is a navigation table?
8. How does the expanded geometry model work? Does it work with map gen features?
9. What are pros and cons of expanded geometry?

N-1: Nav Mesh

1. What are the major wins of a Nav Mesh?
2. How do we ensure convexity?
3. Would you calculate an optimal nav-mesh?
4. Do we still need waypoints? If so, where to place them?



On Convexity

- by definition: convex polygons have all internal angles of less than 180 degrees
  - all diagonals are contained within the polygon
  - a line drawn through a convex polygon in any direction will intersect at exactly two points
- utils.py has isConvex(points): returns true if all the angles made up by a list of points are convex. Points is a list (point1, point2, ..., pointN) and a point is a tuple (x, y). The list must contain at least three points.

Graphs, Search, & Path Planning
Concluded: Models of world for path planning

2018-01-23;

See also: Buckland Ch 5 & 8,

Millington & Funge Ch 4.4

Path finding models

1. **Tile-based graph – “grid navigation”**
   - Simplest topography
   - assume static obstacles
   - imaginary latice of cells superimposed over an environment such that an agent can be in one cell at a time.
   - Moving in a grid is relatively straightforward: from any cell, an agent can traverse to any of its four (or eight) neighboring cells
2. Path Networks / Points of Visibility NavGraph
3. Expanded Geometry
4. NavMesh

Path finding models

1. Tile-based graph – “grid navigation”
2. **Path Networks / Points of Visibility NavGraph**
   - does not require the agent to be at one of the path nodes at all times. The agent can be at any point in the terrain.
   - When the agent needs to move to a different location and an obstacle is in the way, the agent can move to the nearest path node accessible by straight-line movement and then find a path through the edges of the path network to another path node near to the desired destination.
3. Expanded Geometry
4. NavMesh

Path finding models

1. Tile-based graph – “grid navigation”
2. Path Networks / Points of Visibility NavGraph
3. **Expanded Geometry**
   - Discretization of space can be smaller
   - 2 tier nav: Continuous, non-grid movement in local area
   - Can work with auto map generation
   - Can plan nicely with “steering behaviors”
4. NavMesh

Path finding models

1. Tile-based graph – “grid navigation”
2. Path Networks / Points of Visibility NavGraph
3. Expanded Geometry
4. **NavMesh**
   - Win: compact rep, fast search, auto create
   - Each node (list of edges) is a convex polygon
   - Convex = Any point w/in polygon is unobstructed from any other
   - Can be generated from the polygons used to define a map

Generating the Mesh: Greedy/Simple Approach

```
For point a in world points:
    For point b in world points:
        For point c in world points:
            if (it is a valid triangle) and !exists:
                add triangle to mesh
```

Iterate through triangles to merge to quads

Iterate through quads to merge to 5-sided shapes...

Question 2:
Memory

Rank these four space representations according to the memory they would use for the same simple scene (empty space and obstacles):

1. Grid
2. Path network (designed)
3. Path network (flood fill)
4. Nav Mesh + Path network

Why?

Is this good for all games?

- Not necessarily
- Find the right solution for your problem

Game design can cover Game AI

- Cheating / hiding the problem
  - Most AIs don't live long enough to let you spot the flaws in their pathfinding (LOS stop, shoot)
  - Many 1P FPS, AIs don't move very much, shoot from relatively fixed position.
  - FPS games with AI sidekicks kill the enemy AIs so quickly they don't have time to move very far.
  - AI agents can “give up” and return to a safe default
    - https://www.youtube.com/watch?v=gXjUzHhNjIA

FPS implications

- What if we force characters to use melee weapons (e.g. Covenant soldiers in Halo, the Icarus stealth assassins in F.E.A.R., or the Metroids in a Metroid Prime game)? Which technique did they use?

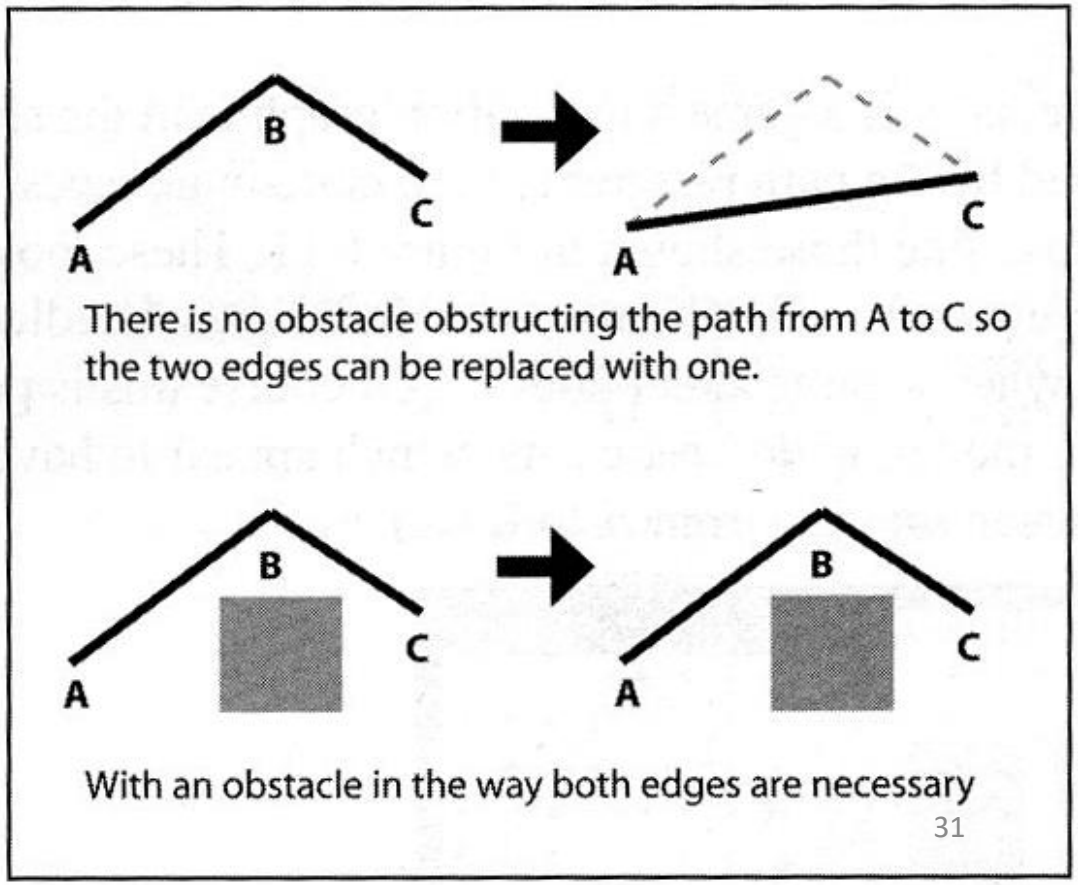
How do you handle walking under bridges?

Having discussed the model of the world, lets talk about searching it and solving various problems that arise with search, and how we fix those problems

PREVIEW: SEARCHING THE MODELS

Path Smoothing via “String pulling”

- Zig-zagging from point to point looks unnatural
- Post-search smoothing can elicit better paths



Long search times / Weird final paths

- Precomputing paths
  - Faster than computation on the fly
  - Especially with large maps or lots of agents
- Add extra heuristic to mark certain grid cells as more “costly” to step through.
  - Cells near obstacles
  - Cells that an agent can get “caught” on
  - Cells that an agent can get “trapped” in
- Path smoothing can help with weird final paths

Precomputing Paths

- Faster than computation on the fly esp. large maps and many agents
- Use Dijkstra's or Floyd-warshall algorithm to create lookup tables
- Lookup cost tables
- What is the main problem with pre-computed paths?

Hierarchical Path Planning

- Used to reduce CPU overhead of graph search
- Plan with coarse-grained and fine-grained maps
- Example: Planning a trip to NYC based on states, then individual roads

Sticky Situations

- Dynamic environments can ruin plans
- What do we do when an agent has been pushed back through a doorway that it has already “visited”?
- What do we do in “fog of war” situations?
- What if we have a moving target?

Heuristic Search

- A*
  - Use when we can't precompute
    - Dynamic environments
    - Memory issues
  - Optimal when heuristic is admissible (and assuming no changes)
  - Replanning can be slow on really big maps
- Hierarchical A* is the ~same, but faster
  - Within 1% of A* but up to 10x faster
- Real-time A*
  - Stumbling in the dark, 1 step lookahead
  - Replan every step, but fast!
  - Realistic? For a blind agent that knows nothing
  - Optimal when completely blind
- Real-time A* with lookahead
  - Good for fog-of-war
  - Replan every step, with fast bounded lookahead to edge of known space
  - Optimality depends on lookahead

Heuristic Search

- D* Lite
  - Assume everything is open/clear
  - Replan when necessary
  - **Worst case: Runs like Real-Time A***
  - Best case: Never replan
  - Optimal including changes