COMP250: Artificial Intelligence

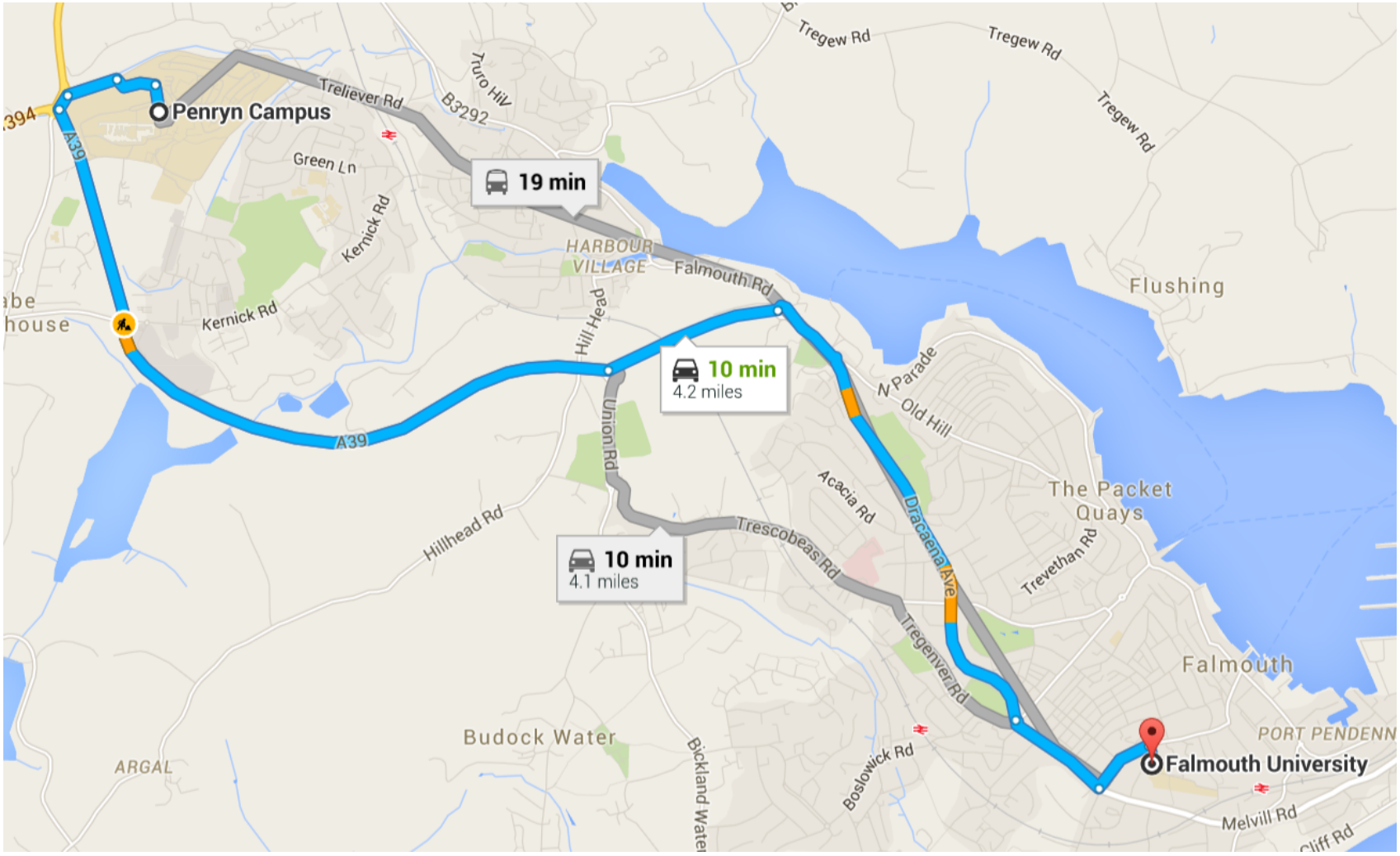
6: Navigation

Pathfinding

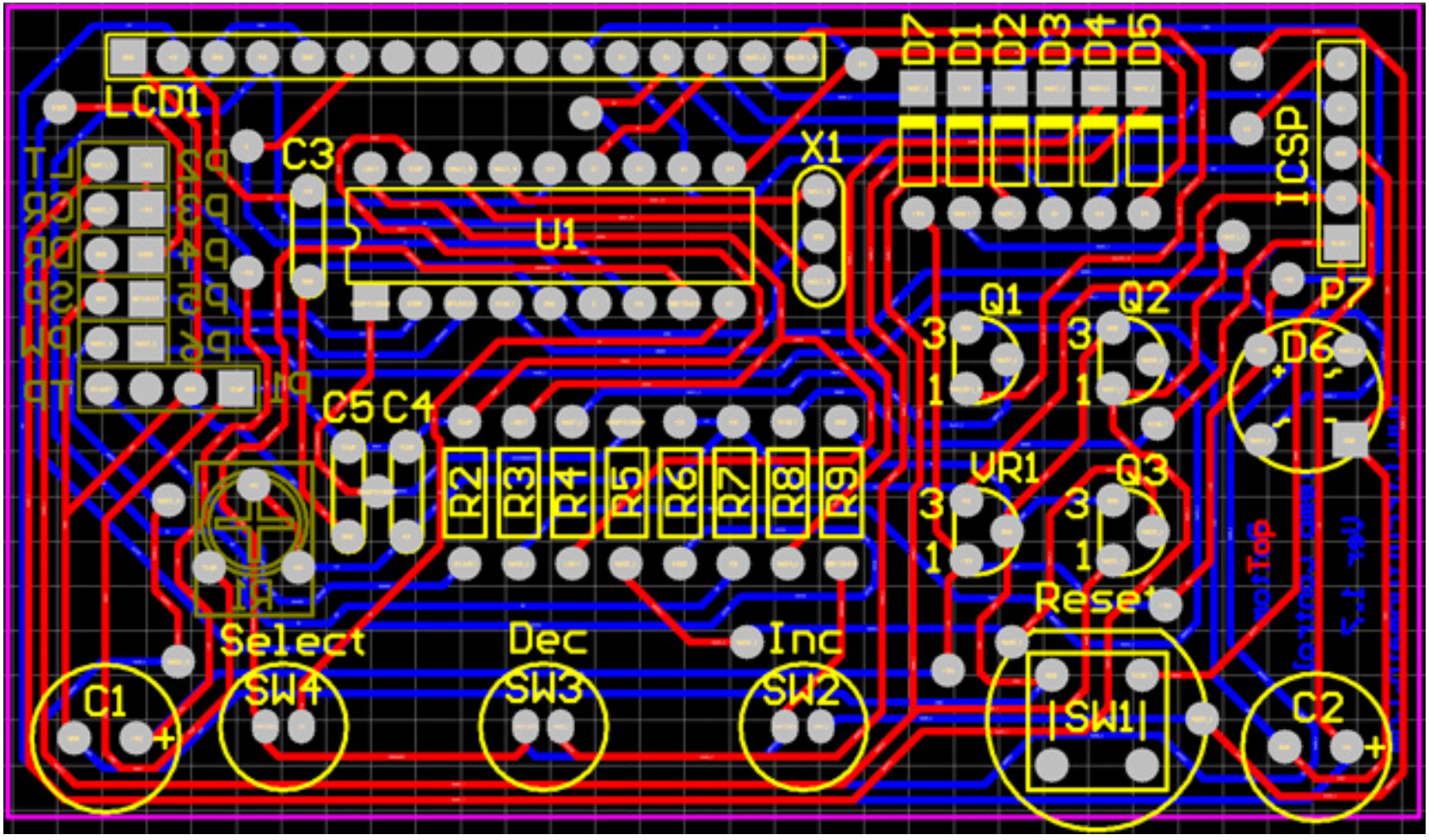
The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads
- E.g. a tile-based 2D game
  - Nodes = grid squares
  - Edges = connections between adjacent squares
- Given two nodes *A* and *B*, find the **shortest path** from *A* to *B*
  - "Shortest" in terms of edge lengths — could be distance, time, fuel cost, ...

Applications of pathfinding



Applications of pathfinding



Applications of pathfinding

Many applications in game AI

- Non-player character AI
- Mouse-based movement (e.g. strategy games)
- Maze navigation
- Puzzle solving

Pathfinding example

- `https://github.com/falmouth-games-academy/bsc-live-coding`
- Open `06_pathfinding` in PyCharm

Aside: data structures

- **Stack**: can **push** to the top and **pop** from the top
  - "Last in, first out"
- **Queue**: can **enqueue** to the back and **dequeue** to the front
  - "First in, first out"
- **Priority queue**: maintains its elements in **sorted** order
  - **Enqueue** automatically puts the element in the correct position according to its priority
  - **Dequeue** gives the highest priority element currently in the queue
  - Usually implemented as a **heap** or a **balanced tree**...
  - ... but implementations are available for all popular programming languages

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- Inefficient — generally has to explore the **entire map**
- Finds a path, but probably not the **shortest**

Greedy search

- Always try to move **closer** to the goal
- Can be implemented with a **priority queue**
- Doesn't handle **dead ends** well
- Not guaranteed to find the **shortest** path

A* search

- Let $h(x)$ be an estimate of the distance from x to the goal
- Let $g(x)$ be the distance of the path found from the start to x
- Choose a node that minimises $g(x) + h(x)$
  - Contrast with greedy search, which just minimises $h(x)$

Properties of A* search

- A* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
  - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance
- The more accurate $h(x)$ is, the more efficient the search
  - E.g. $h(x) = 0$ is admissible, but not very helpful
- $h(x)$ is a **heuristic**
  - In AI, a heuristic is an estimate based on human intuition
  - Heuristics are often used to prioritise search, i.e. explore the most promising options first

Tweaking A*

- Can change how $g(x)$ is calculated
  - Increased movement cost for rough terrain, water, lava...
  - Penalty for changing direction
- Different $h(x)$ can lead to different paths (if there are multiple "shortest" paths)

String pulling

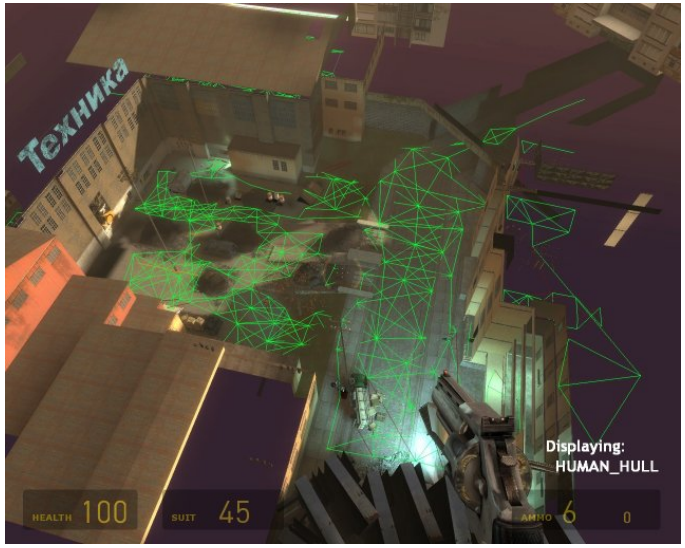
- Paths restricted to edges can look unnatural
- Intuition: visualise the path as a string, then pull both ends to make it taut
- Simple algorithm:
  - Found path is $p[0], p[1], \ldots, p[n]$
  - If the line from $p[i]$ to $p[i+2]$ is unobstructed, remove point $p[i+1]$
  - Repeat until there are no more points that can be removed

Navigation meshes

Pathfinding in videogames

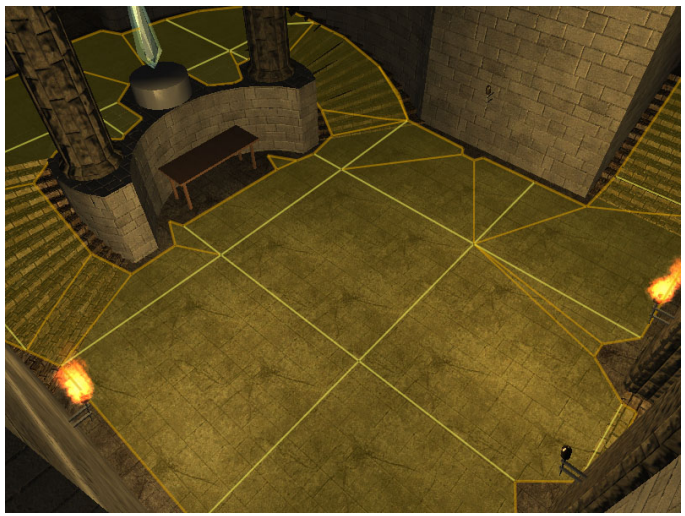
- A* works on any **graph**
- But what if the game world is not a graph? E.g. complex 3D environments

Waypoint navigation



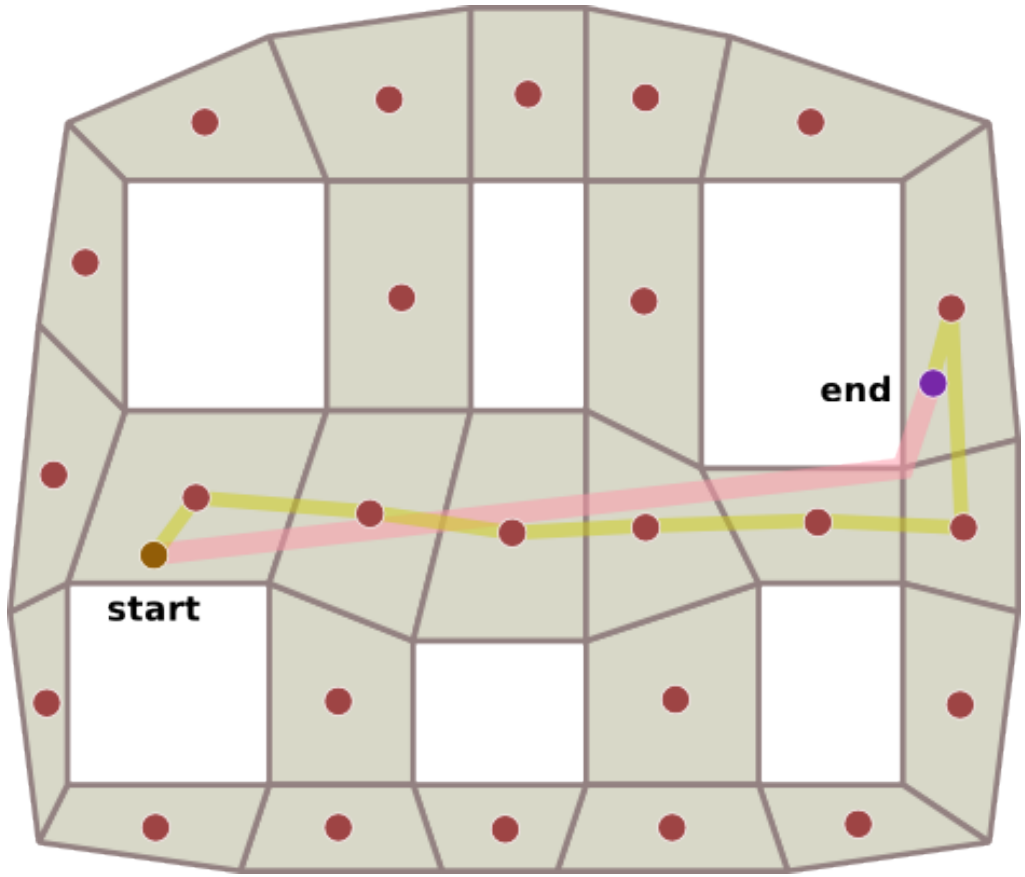
- Manually place graph nodes in the world
- Place them at key points, e.g. in doorways, around obstacles
- Works, but...
  - More work for level designers
  - Requires lots of testing and tweaking to get natural-looking results
  - No good for dynamic environments

Navigation meshes



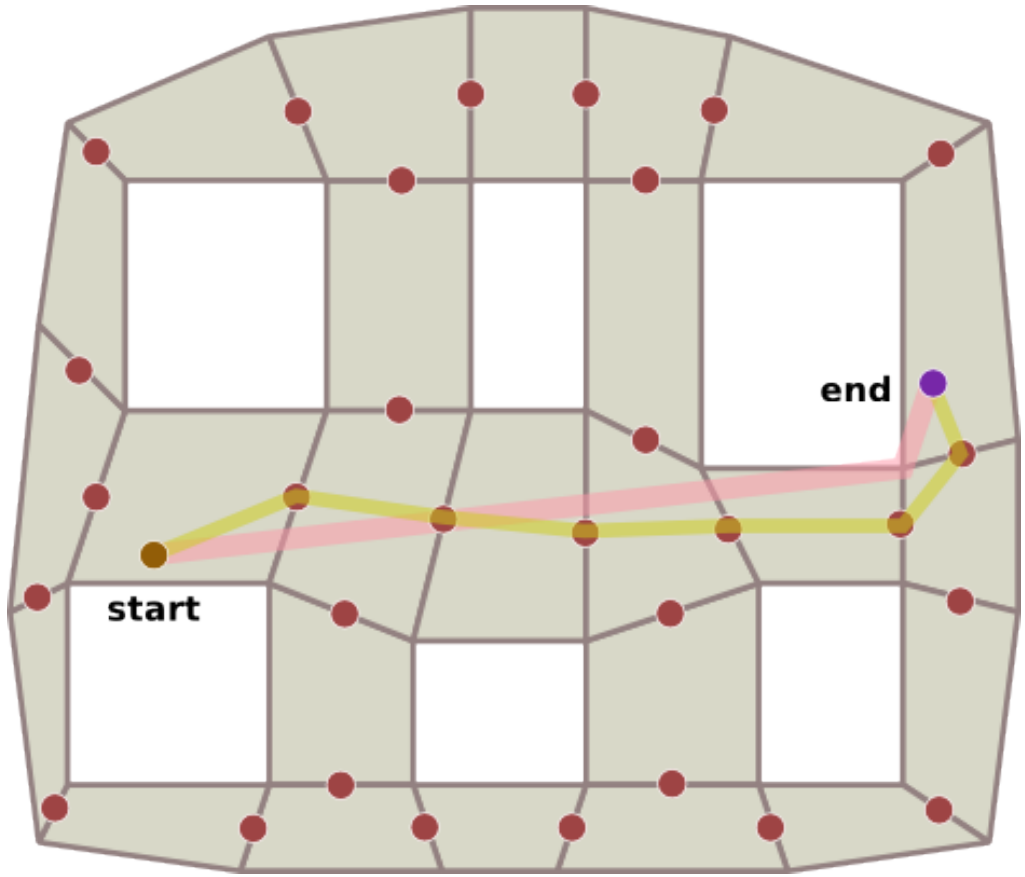
- Automatically generate navigation graph from level geometry
- Basic idea:
  - Filter level geometry to those polygons which are **passable** (i.e. floors, not walls/ceilings/obstacles)
  - Generate graph from polygons

Meshes to graphs



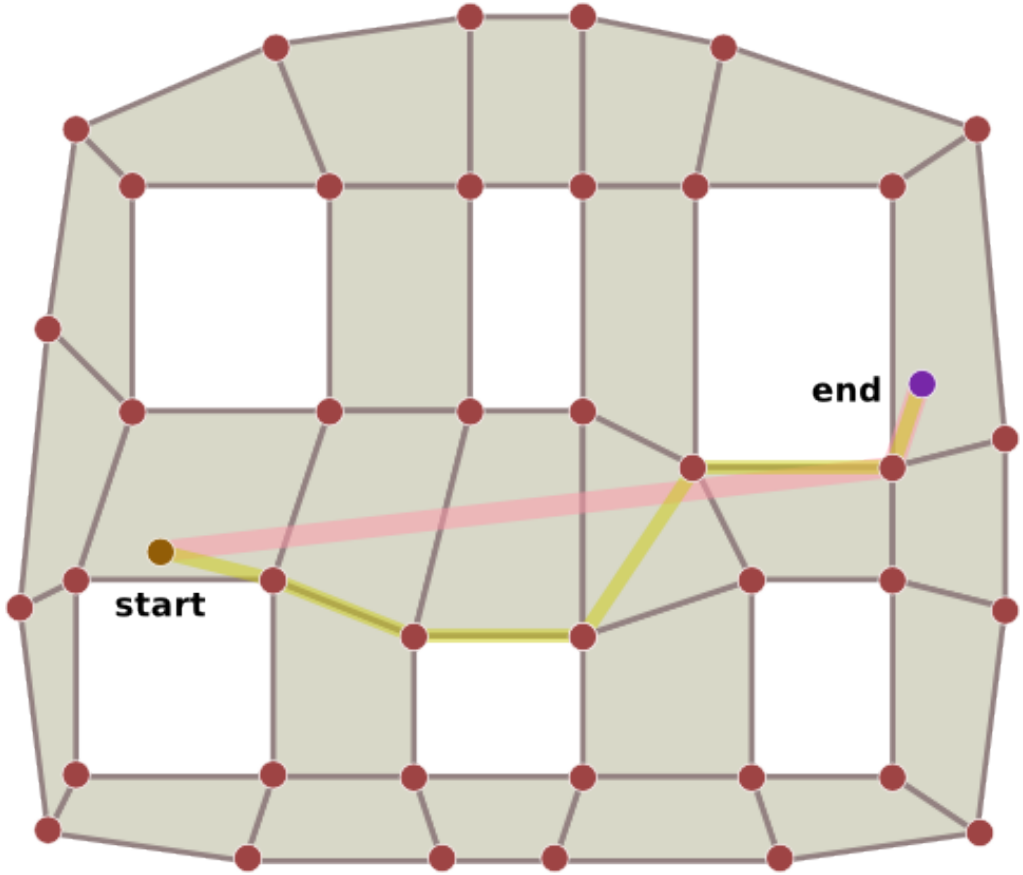
Centres of polygons

Meshes to graphs



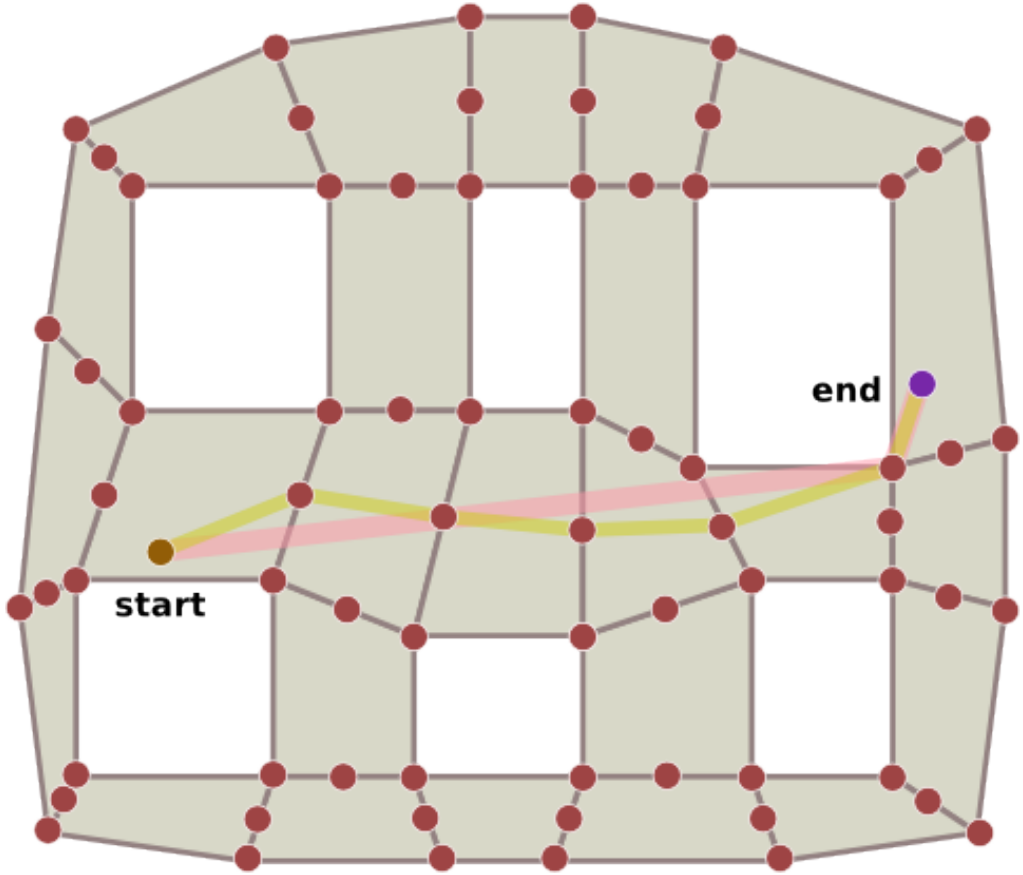
Centres of edges

Meshes to graphs



Vertices of polygons

Meshes to graphs



Hybrid approach: edges and vertices

Following the path

- **Funnelling**: like string pulling but for navigation meshes
  - `http://digestingduck.blogspot.co.uk/2010/03/simple-stupid-funnel-algorithm.html`
  - `http://jceipek.com/Olin-Coding-Tutorials/pathing.html`
- **Steering**: don't have your AI agent follow the path exactly, but instead try to stay close to it
- **Dynamic environments**: may need to re-run pathfinder if environment changes (e.g. movable obstacles, destructible terrain)

The travelling salesman problem

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node *S*, find the **shortest possible path** that visits every node **exactly once** and returns to *S*
- Many real-world applications
  - Transport and logistics
  - Manufacturing
  - Playing Pac-Man
  - Pub crawls
    (`http://www.math.uwaterloo.ca/tsp/pubs/`)

Solving TSP

- TSP is **NP-complete**
  - If $P \neq NP$, then there is **no** polynomial-time algorithm for solving it
- Entire research field devoted to finding efficient **search algorithms** and **heuristics**