Path Planning

(SENG 463 - Game Programming)

Dr.Çağatay ÜNDEĞER
Research and Innovation Director
SimBT Inc.

e-mail :
cagatay.undeger@simbt.com.tr
cagatay@undeger.com

Outline

- Path planning

Path Planning

- Path planning can be described as finding a sequence of moves from an initial state to a goal state.
- Path-planning algorithms:
  - Off-line,
  - On-line (Real-Time),
  - Hybrid.

Off-Line and On-Line Algorithms

- Off-line algorithms;
  - Find the whole solution in advance,
  - Suffer from execution time.
- On the other hand, on-line algorithms;
  - Require planning and execution phases to be coupled.
  - Not designed to be optimal,
  - Usually find poor solutions.

Incremental Heuristic Search

- Hybrid solution:
  - Incremental heuristic search.
- Optimal and more efficient than off-line path planning algorithms.
- Still slow for some real-time applications.
- Considered as efficient off-line algorithms.

Changing Goals

- Common to assume that goal state is static.
- When relaxed for covering changing goals (moving target search),
  - The problem becomes very complicated.

Dealing with Changing Goals

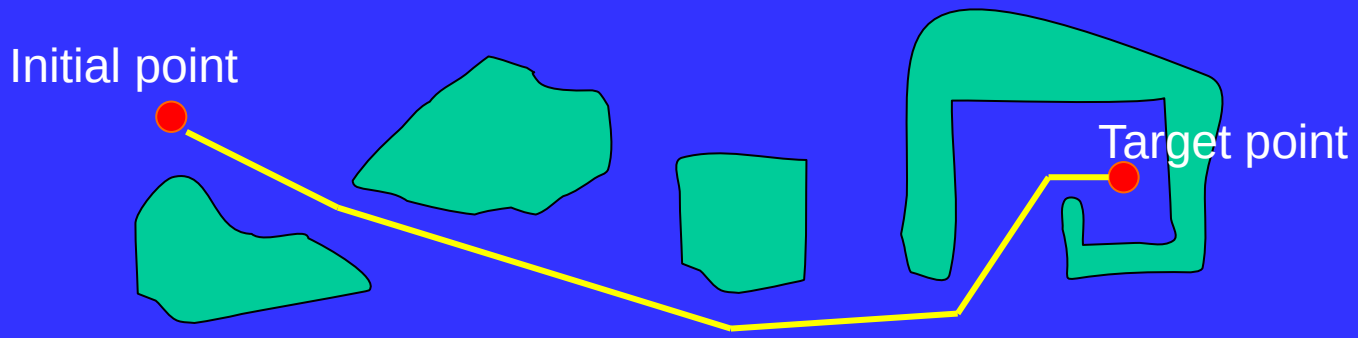
- Off-line and incremental path planning:
  - Require re-planning towards the changing goal from scratch in each step.
- On-line search:
  - Usually designed for partially observable environments, but not for changing goals,
  - Store search information collected during the exploration.
  - There are few number of algorithms that can handle changing goals/moving targets.

Multi-Agent Path Planing

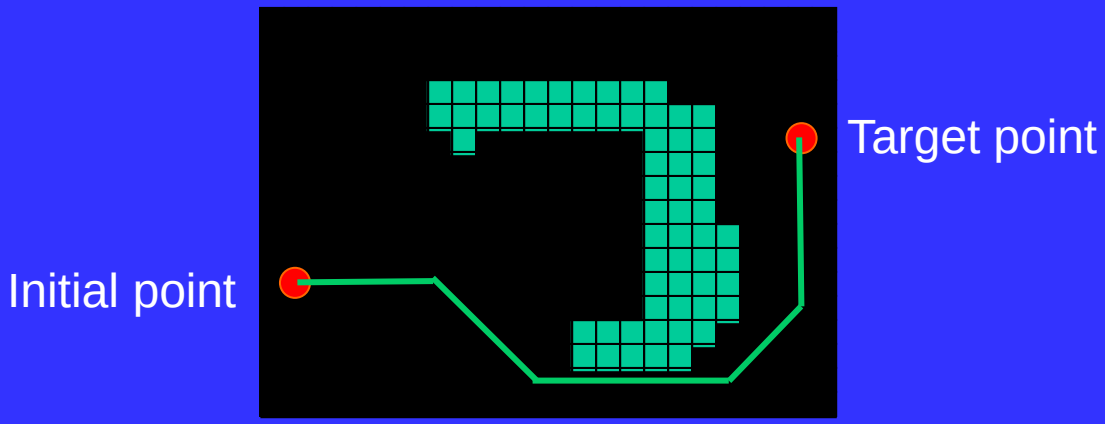
- By increasing the number of agents (predators) involved in the environment,
  - Problem can be extended to a search against a static or moving prey with multiple coordinated agents.
- A recent research area called multi-agent pursuit,
- Not an easy task, not much study done so far.

Environment Representations

- Polygonal environments (e.g. Doom)

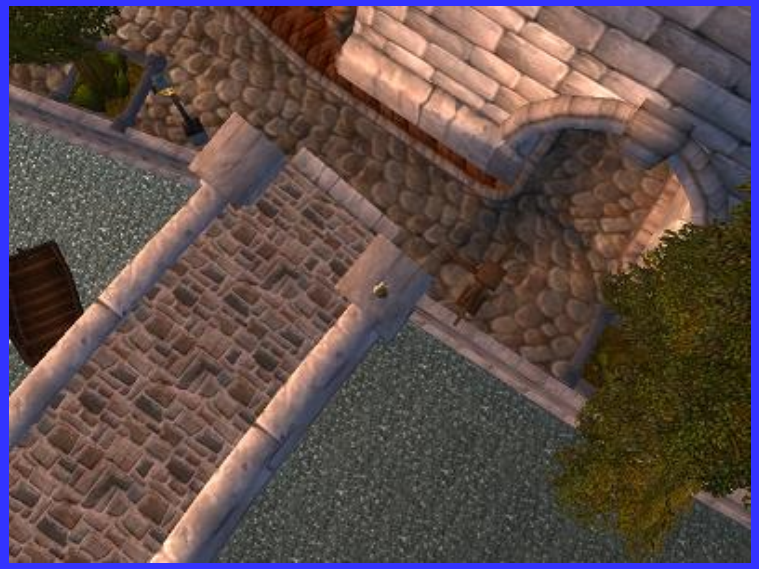


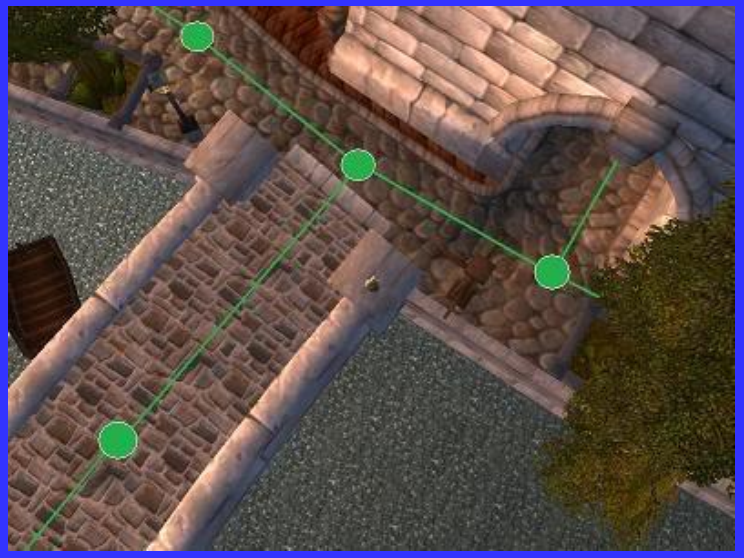
- Grid based environments (e.g. War Craft)



Waypoint Graphs

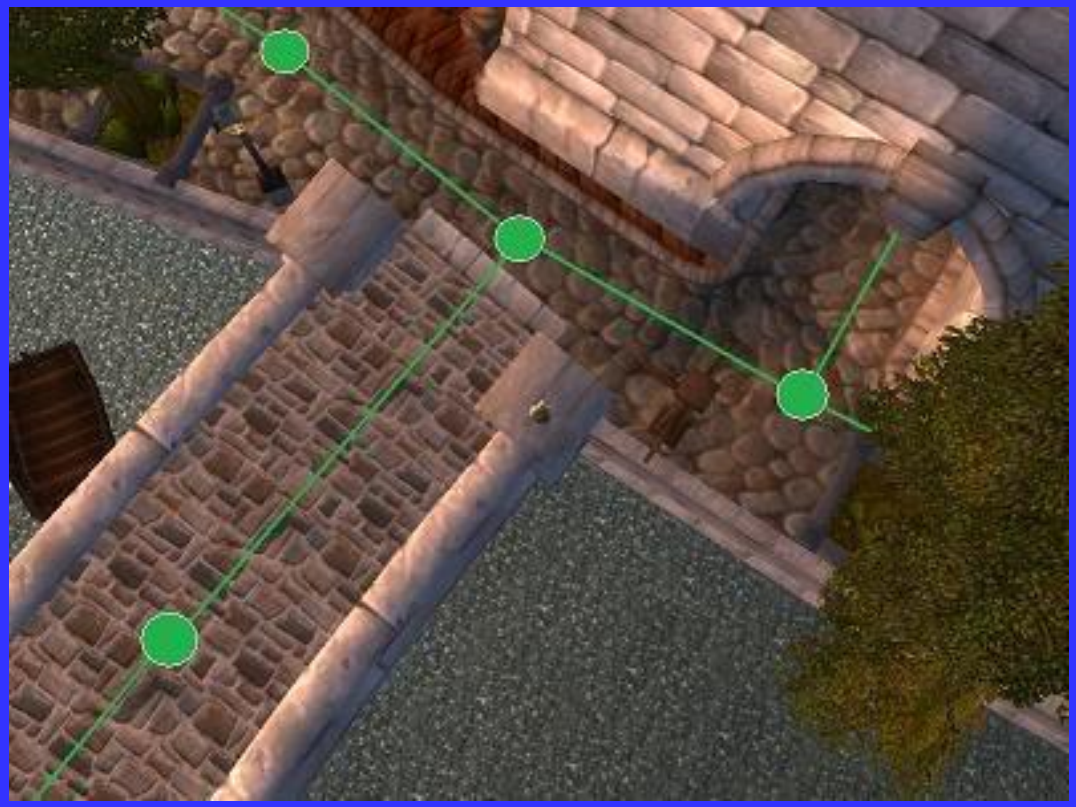
- To simplify work:
  - A pre-determined graph of waypoints showing the possible paths a game entity can walk through.
- A typical waypoint graph looks like below:





Advantages of Waypoint Graphs

- Using this pre-determined static graph,
  - Path planning is done easily and fast.



A*

- A* is an off-line path planning algorithm.
- It is common to use A* if environment is not very large.
- A* is optimal and efficient.
- The problem inputs are:
  - A graph of waypoints,
  - Initial point,
  - Target point.

Drawbacks of Waypoint Graphs

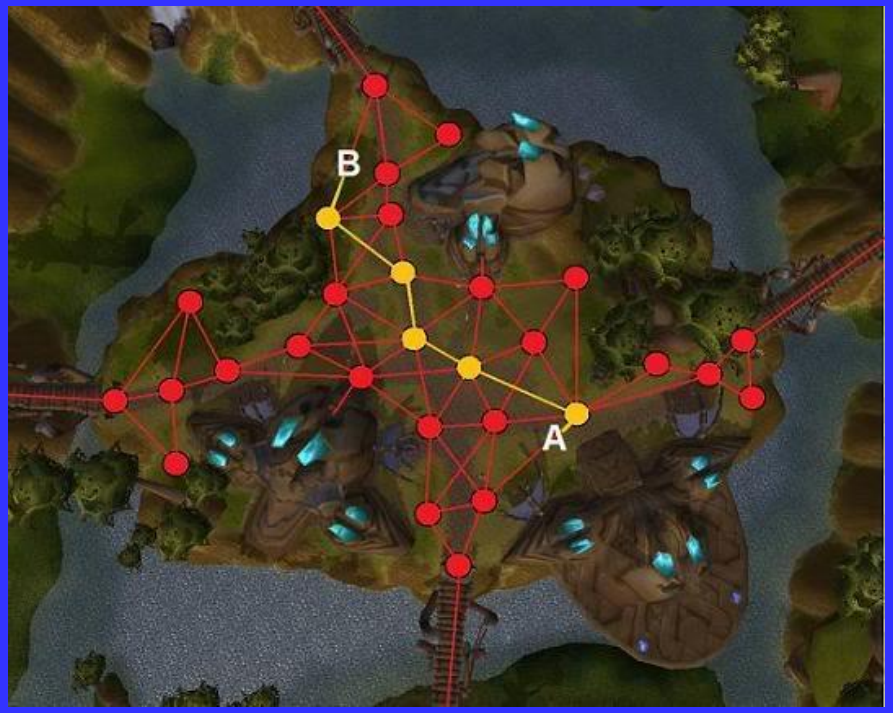
- Large, open areas usually require tons of waypoints throughout the game world to achieve adequate movement.
- With a waypoint-based system, we need to place a lot of waypoints to get full coverage.

Drawbacks of Waypoint Graphs

- Path planning will be inefficient in very large graphs. Possible solutions:
  - Methods to minimize graphs may be used
  - Non-optimal A * versions may be used.
  - Random search algorithms (e.g. random trees, genetic algorithms, probabilistic roadmaps) may be used.
  - Increamental heuristic search algorithms (e.g. D*, D*-Lite) may be used.
  - Real-time search algorithms (e.g. RTA*, MTES, MAPS) may be used.

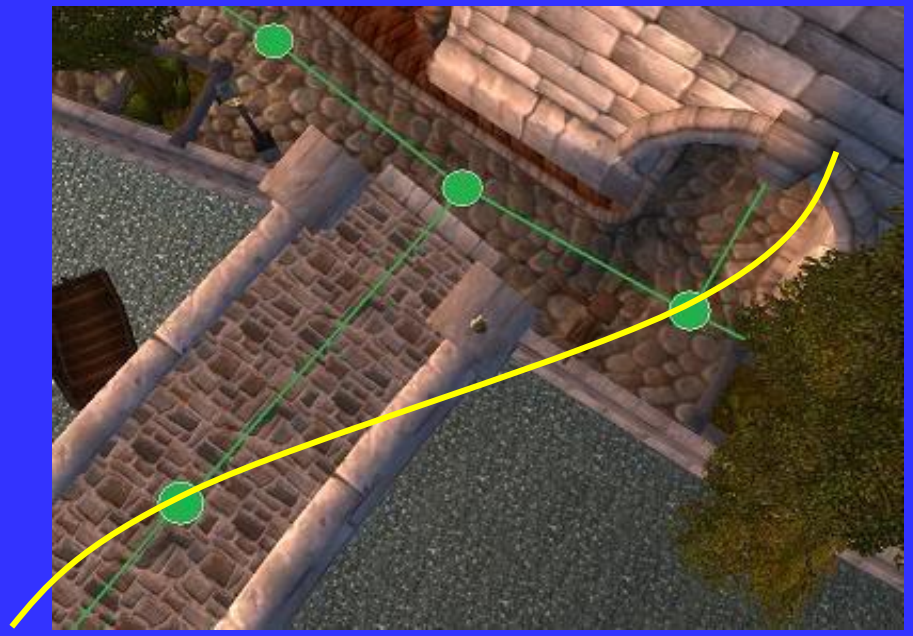
Drawbacks of Waypoint Graphs

- Waypoint graphs force your characters to stick to the graphs you create.
- This means that your characters are effectively on rails, doing zig zags,
  - And will almost never take the *optimal* path.



Drawbacks of Waypoint Graphs

- After we created the path, we usually adjust to make it smoother.
- Don’t have any information about anything outside the network.
  - Makes it impossible/hard to do path-smoothing and adjustment safely and reliably.



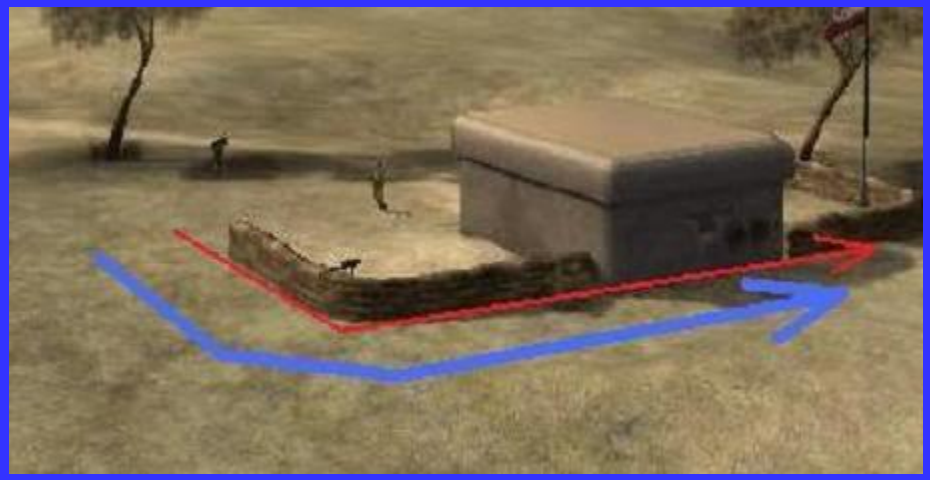
Drawbacks of Waypoint Graphs

- If there is a dynamic entity/obstacle in front of us, we need to avoid hitting it.
- But since waypoint graphs don't have any information about anything outside the graph.
  - That makes it impossible for path finding systems to modify waypoint paths to deal with dynamic obstacles robustly.

Need dropping down tons of waypoints until your waypoint graph is so large, which makes path finding so slow.

Drawbacks of Waypoint Graphs

- Waypoint graphs tend to work poorly for entities with different heights, widths, turn rates, etc.
- AI systems need to control lots of different kinds of units such as tanks, planes, ships, motorcycles, etc.
- Usually impossible to use a single, unified waypoint graph for all the different types of units in our game.



Problems with Path Finding

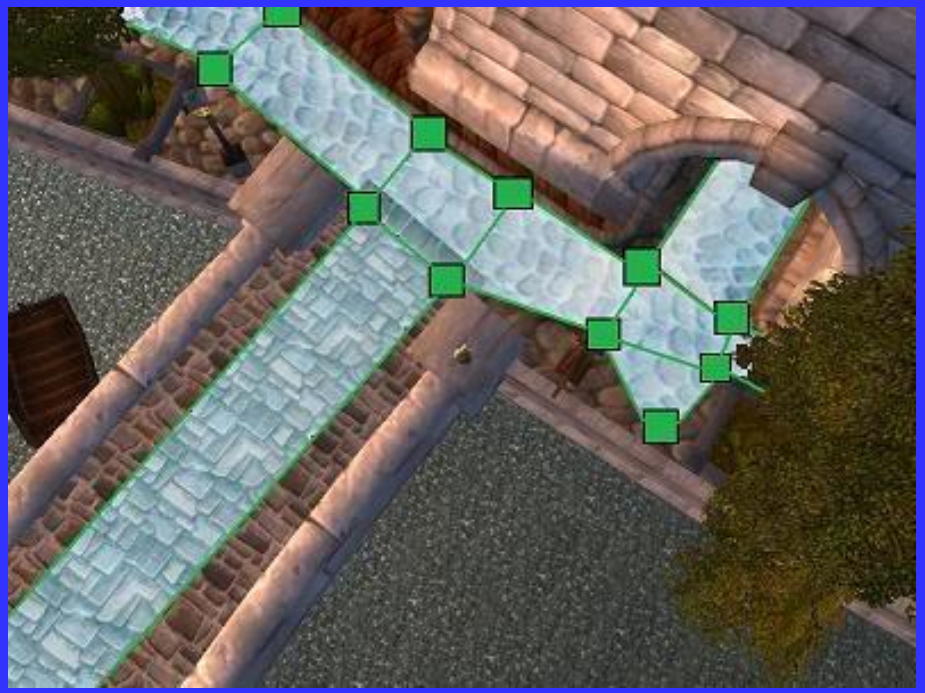
- Because of these reasons, in games, we often see path finding problems such as:
  - Going through silly and non-optimal paths,
  - Hitting obstacles and being stuck somewhere because of obstacles.

Problems with Path Finding

- Most of these problematic games use *waypoint graphs* for path finding.
- That's why waypoint graphs are not mostly prefered in today's modern games.
- The new widely used approach today is to use navigation meshes.

Navigation Meshes

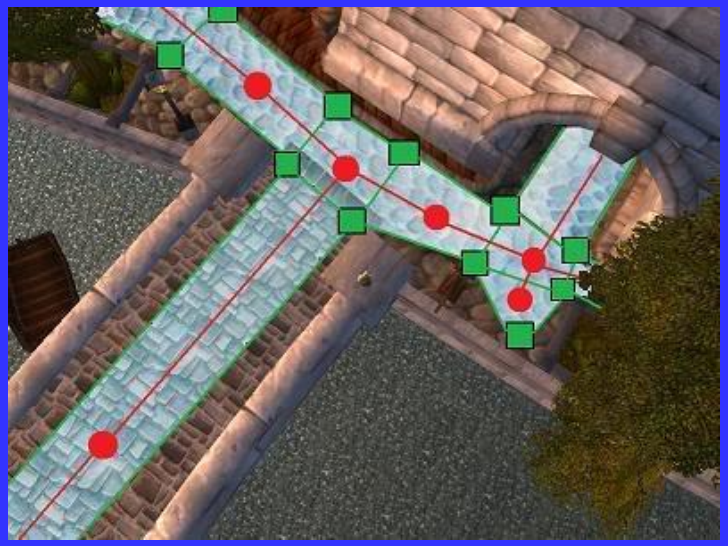
- Convex polygons are used to describe where AI characters can travel in the environment.
- Navigation meshed are illustrated below:



Navigation Meshes

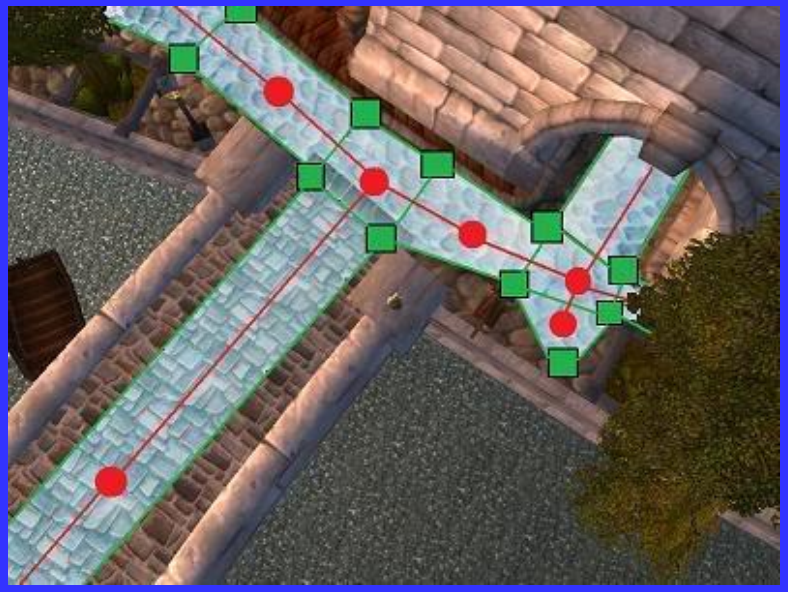
- It is a graph, just like a waypoint graph.
- Core path finding routines are very similar.
- A navigation mesh has a polygon associated with each graph node.

A navigation mesh can actually be significantly faster since We usually cover the same area with fewer nodes



Navigation Meshes

- Just run A* on nodes of neighbor polygons to find a path.
- Then generate actual path passing through shared edges of connected polygons.



Advantages of Navigation Meshes

- A navigation mesh may be significantly faster than waypoint graphs,
- Since we usually cover the same area with fewer nodes.

Advantages of Navigation Meshes

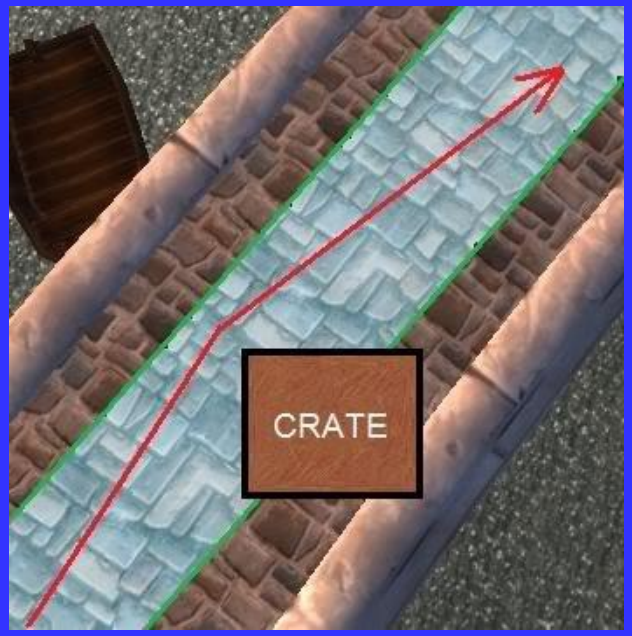
- With a navigation mesh,
  - We can describe the navigable area with few number of convex polygons

Advantages of Navigation Meshes

- Since we know where a character can safely walk,
  - We can smoot out the path in any way we like
    - As long as the smoothing keeps the path on the navigation mesh.

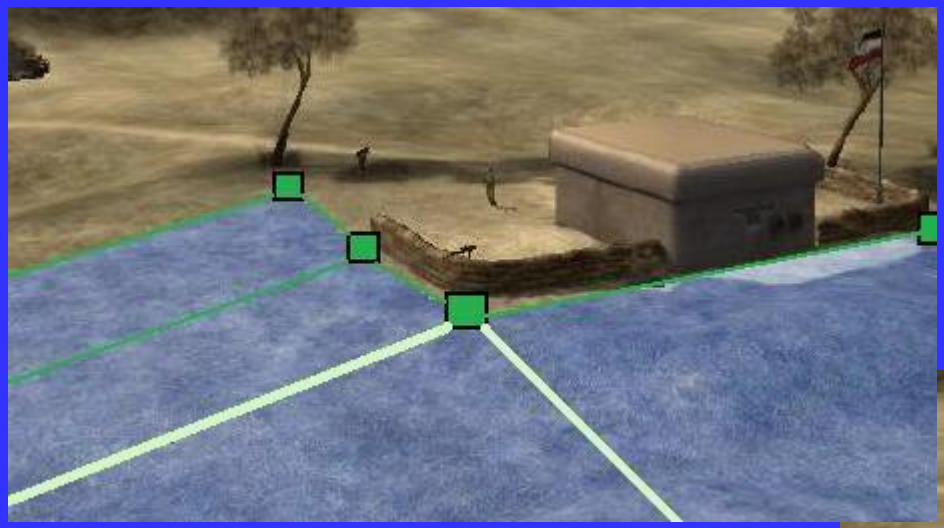
Advantages of Navigation Meshes

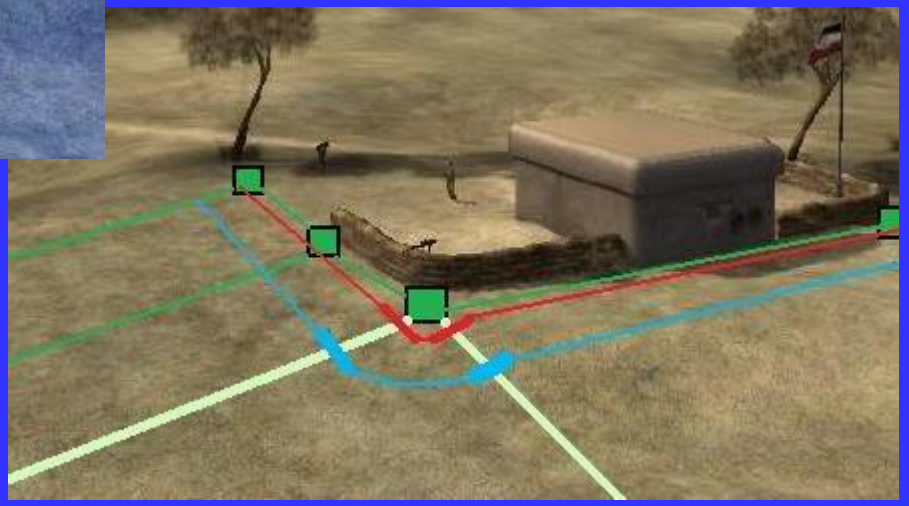
- It's a lot easier to avoid dynamic entities/obstacles,
- We know what areas we're allowed to walk in.
- We do a bit of ray-casting against the obstacle and adjust our path around it.



Advantages of Navigation Meshes

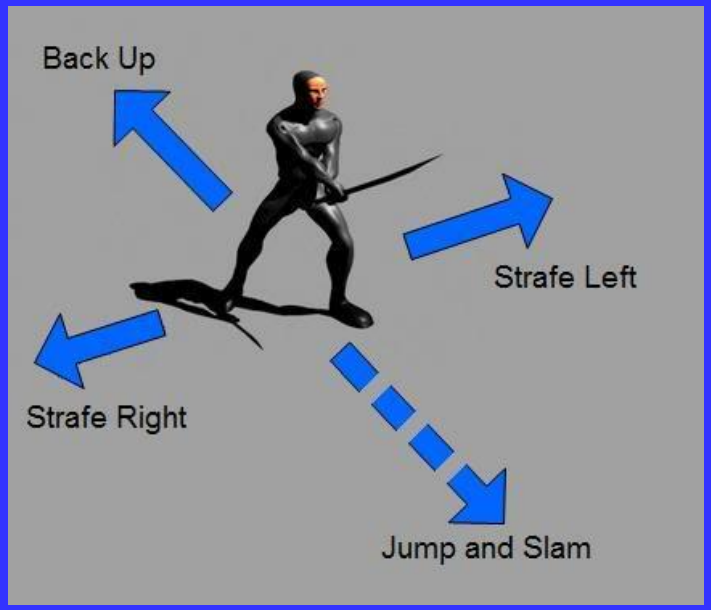
- Easier to handle different kinds of entities with one navigation mesh.





Advantages of Navigation Meshes

- For user controlled characters,
  - Navigation meshes will increase the efficiency of collision tests to find infeasible locations to move.
  - Ray-casting can be expensive for game worlds.



Drawbacks of Navigation Meshes

- More difficult to implement path finding algorithms on navigation meshes.

Real-Time Path Planning

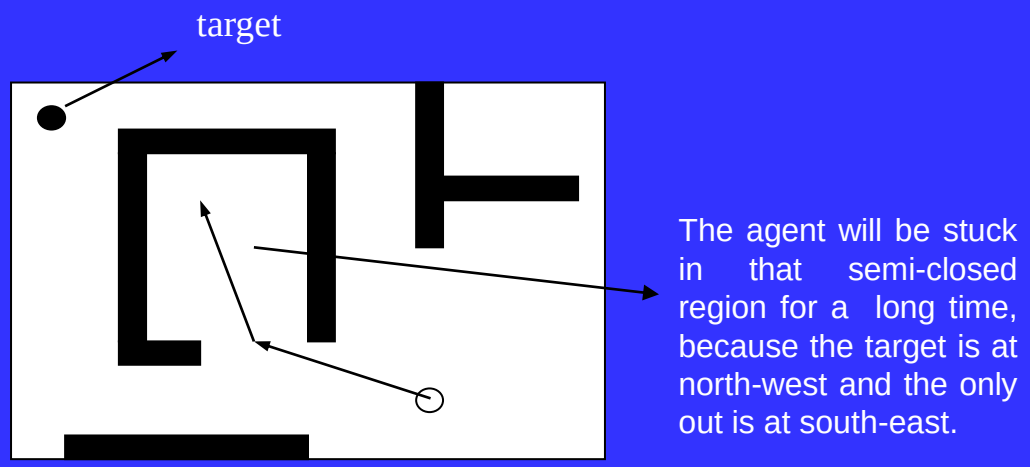
- Path planning will be inefficient in very large graphs.
- If methods to minimize your graphs are not enough in your problem domain,
  - You may use real-time path planning algorithms;
    - Which are fast,
    - But provide low quality solutions.
  - Real-time search algorithms (e.g. RTA*, MTS, MTES, MAPS, etc.) may be used.

Real-Time A* (RTA*)

- Learning Real-Time A* (LRTA*) and
- Real-Time A* (RTA*)
  - Proposed by Korf,
  - Former generic heuristic search algorithms ***for fixed goals***.
  - Build and update a table containing heuristic estimates.
- LRTA* is able to learn optimal table values in single or multiple runs.
- RTA* performs better than LRTA* in the first run,
- Lack of learning optimal table values.

Heuristic Depression

- The original RTA* only considers immediate successors to determine the next move.
- Stuck in semi-closed regions for a long time.
- ***A heuristic depression*** is a local maximum, whose heuristic values have to be filled up before the agent can escape from it.



RTA* with *n*-Look-Ahead Depth

- RTA* can easily be extended to have any arbitrary ***look-ahead depth***.
- Instead of examining immediate neigbors of the current state, *n* level neighbors are used.
- Reduces the number of moves to reach the goal significantly,
- Exponential in the look-ahead depth.
- Large *n* is not preferred in practice.

Moving Target Search (MTS)

- RTA* and many variations of these algorithms are all limited to work on fixed goals,
- Ishida and Korf proposed Moving Target Search (MTS).
  - Built on LRTA*
  - Capable of pursuing a moving target.
- MTS is a poor algorithm.
- When the target moves, the learning process has to start all over again.
- A performance bottleneck in heuristic depressions.

MTS-c & MTS-d

- Since original MTS is a poor algorithm, two MTS extensions:
  - *Commitment to Goal* (MTS-c) and
  - *Deliberation* (MTS-d) are proposed by Ishida.
- To use the learned table values more effectively,
  - MTS-c ignores some of the target's moves,
  - MTS-d performs an off-line search to update the heuristic values.