Advanced Real-time Pathhfind in Dynamic Environment in Supernauts

Harri Hatinen
Lead Programmer, Grand Cru

GDC 'Cn

GAME DEVELOPERS CONFERENCE™ CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
SHANGHAI, CHINA · OCTOBER 19–21, 2014

The Problem (1/2)

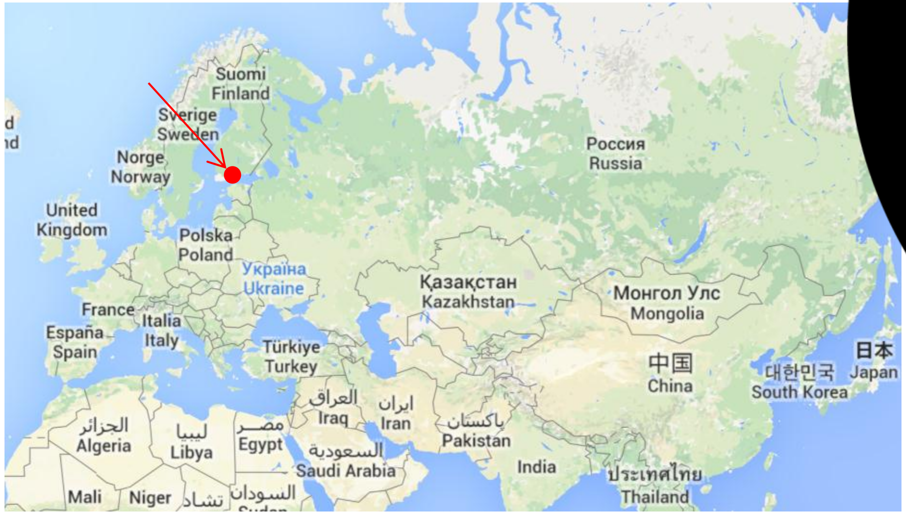
- A* is an optimal pathfind algorithm
- Ways to improve performance:
  - Better heuristics
  - **Smaller data structure**

The Problem (2/2)

- Navigation Mesh is a standard in modern games
- However there isn’t a good standard for **user generated content** and **changing environments**.

Grand Cru

- 20 employees
- Helsinki, Finland



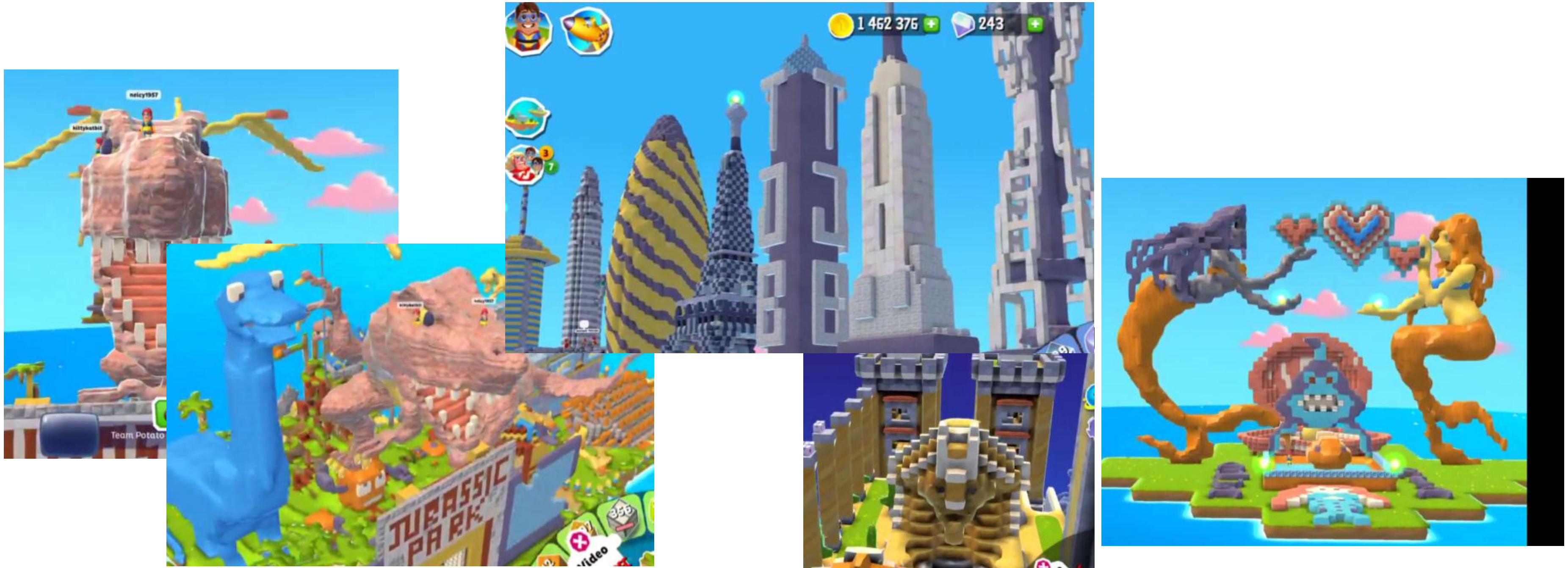
Est. 2011
Grand Cru
Kallio, Helsinki

Harri Hatinen

- Co-founder
- Lead Programmer
- harri.hatinen@grandcrugames.com

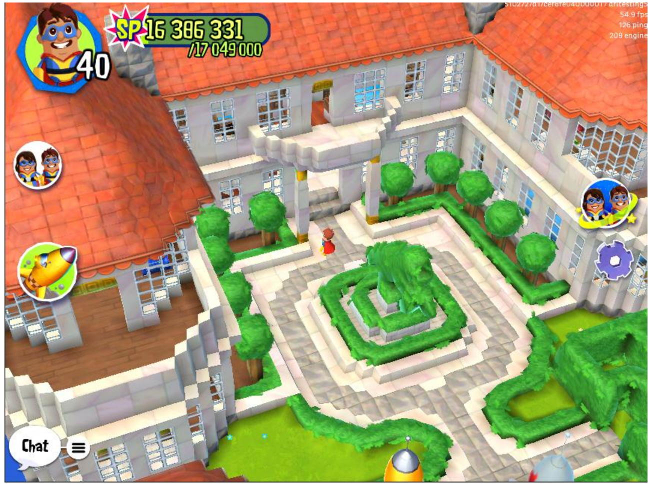
Supernauts

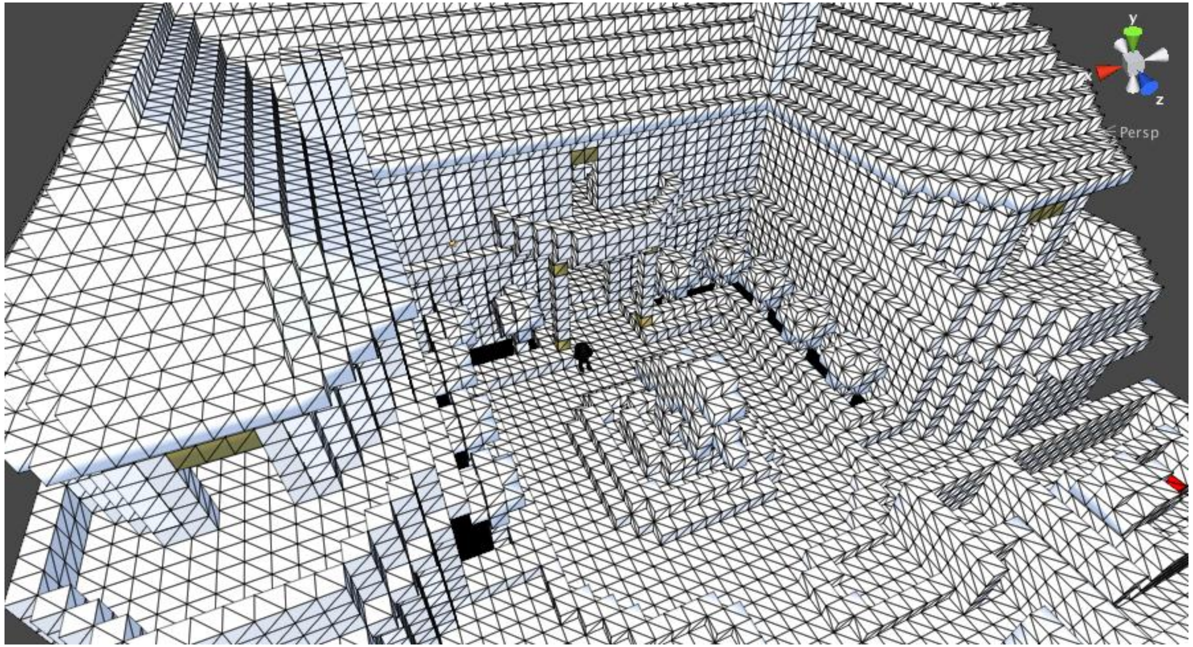
- Everything is User Generated



Supernauts

- Everything is built from cubes



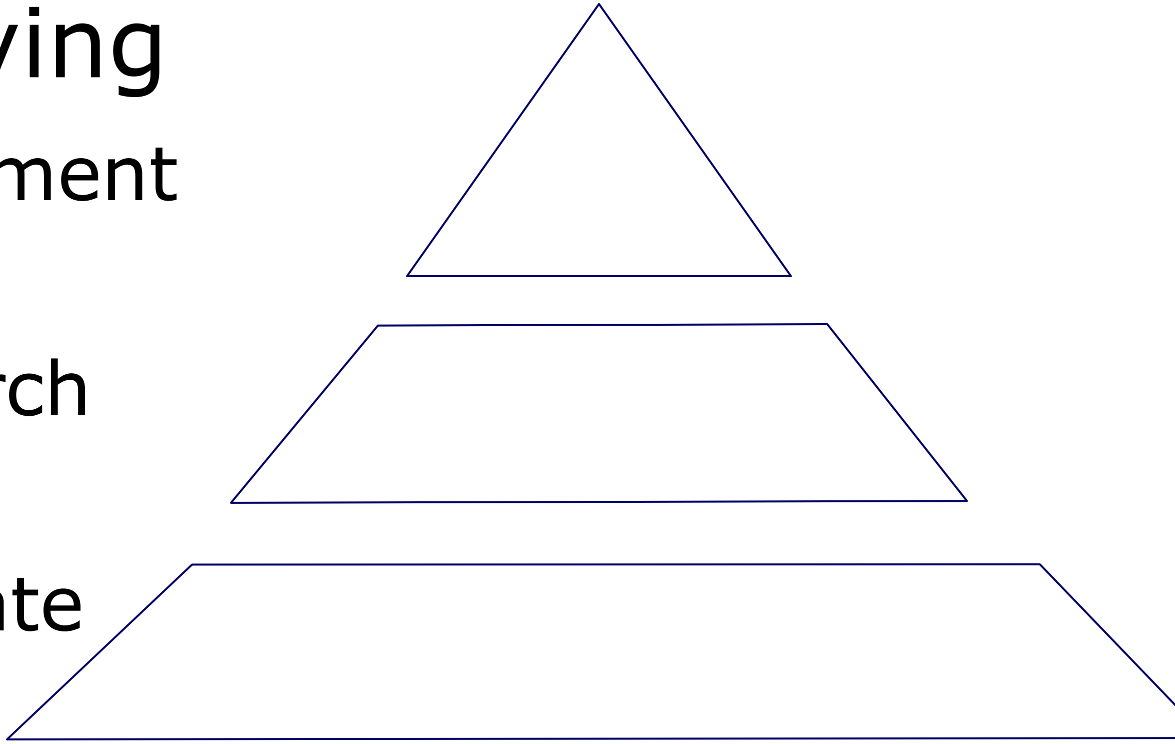


Supernauts

- Pathfind is used for:
  - movement controls
  - AI
- [Demo video]

Problem Solving

- Step 1: Experiment
- Step 2: Research
- Step 3: Innovate

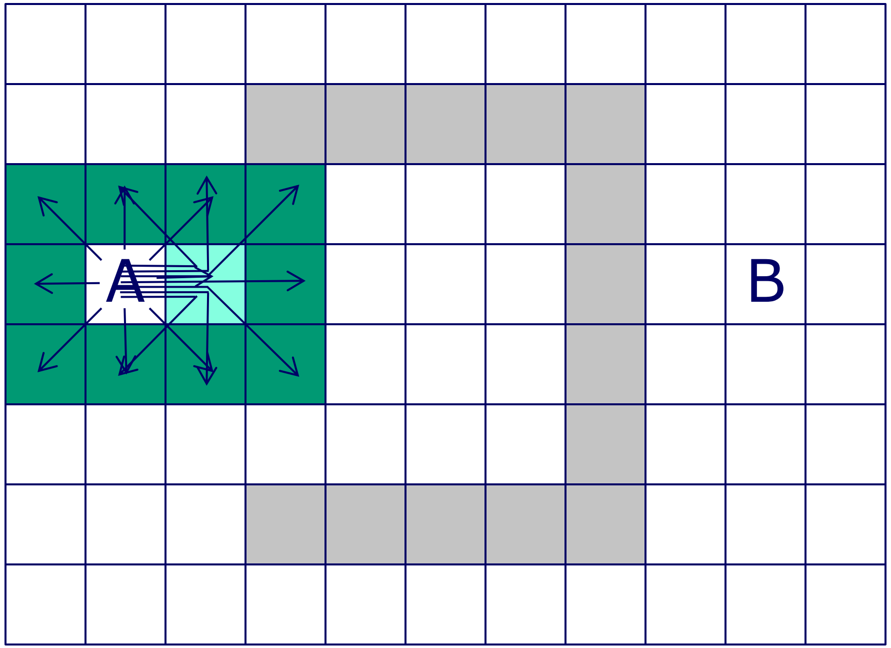


Step 1: Experiment

- Do we even need a costly and advanced path-find algorithm?
- **Don’t waste effort** on wrong features
- Make simple and fast implementation

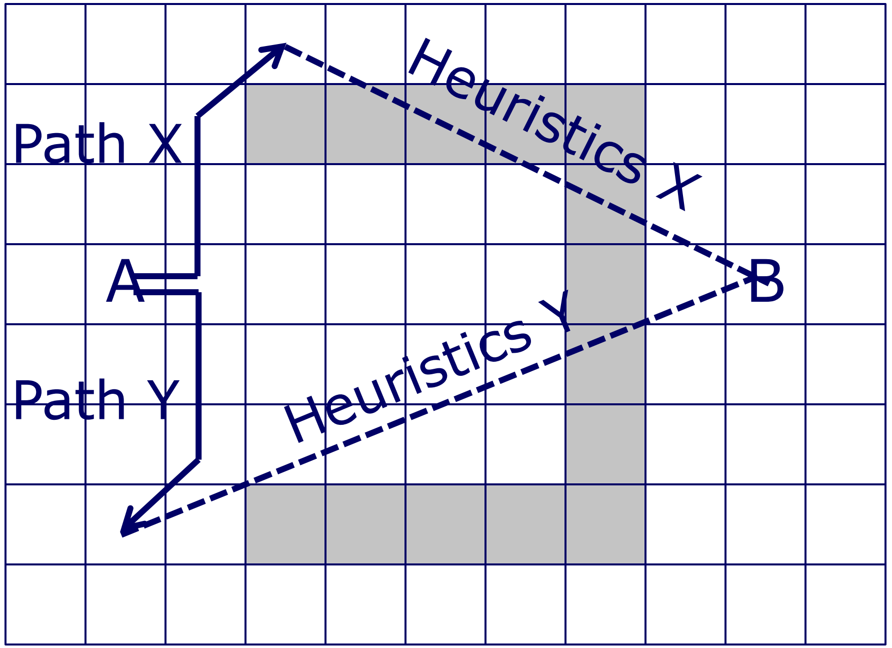
  -> Figure out requirements and limitations quickly

A* in a linear Grid



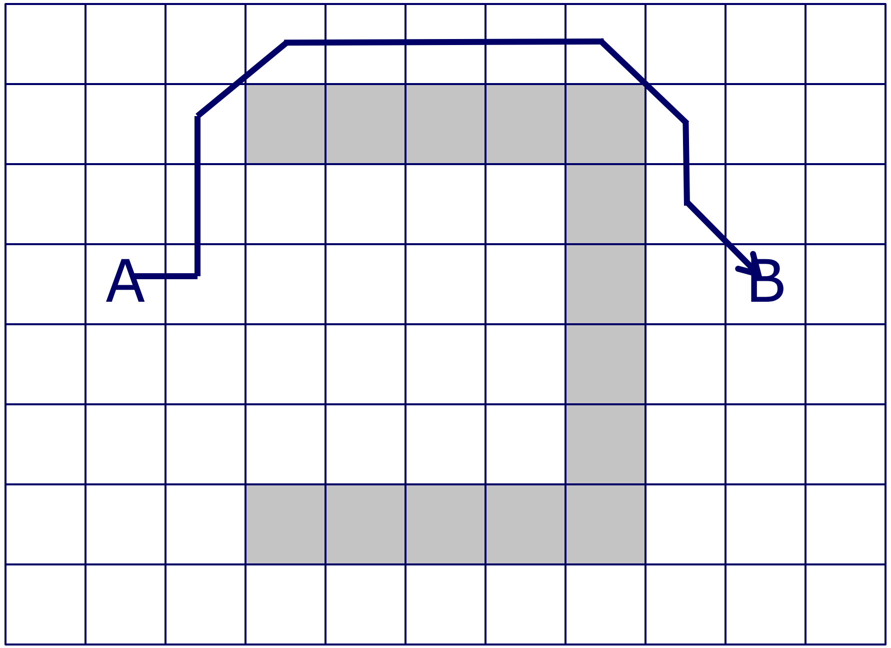
- From current node, expand the path in neighbouring nodes.
- Choose node with **most promising** path
- Repeat

A* in a linear Grid



- How can we know the most promising path?
- Length of Path X = Length of Path Y
  - Which one is better?
- By heuristics
- Score = Path + Heuristics
  - Smaller score = better

A* in a linear Grid



The best path could look something like this.

Experiment: Result

- 1-2 hours to implement
- Proved the need for a pathfind algorithm.
- However, it quickly became apparent that it is way too slow.

Step 2: Research

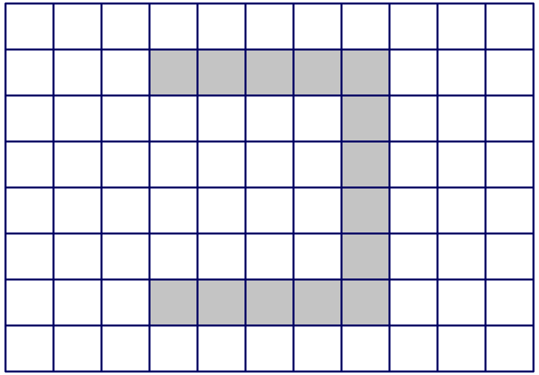
- Now we know that
  - A) We need a path-find algorithm
  - B) Simple implementation is not enough

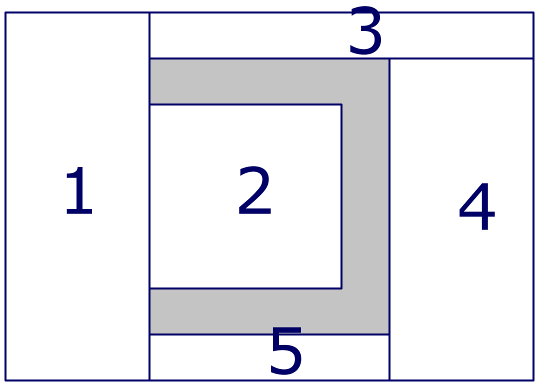
-> Find out all the existing knowledge on the subject

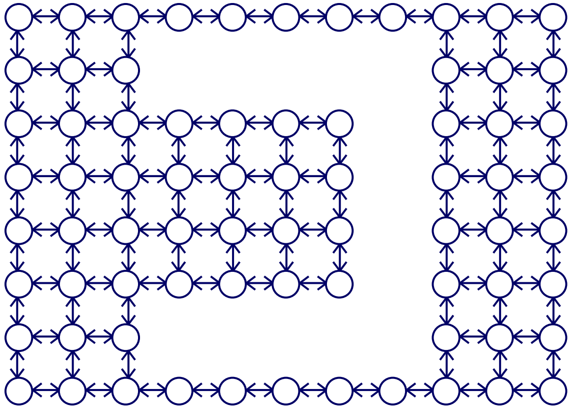
Research

- A* is the most optimal spatial search algorithm
- Optimization focuses on improving **data structures**
- NavMesh is de facto standard in modern video games

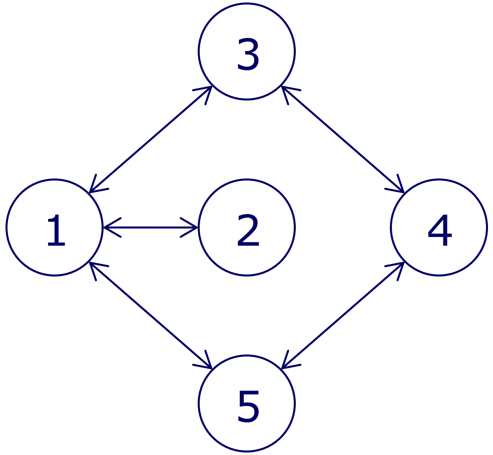
What is NavMesh?





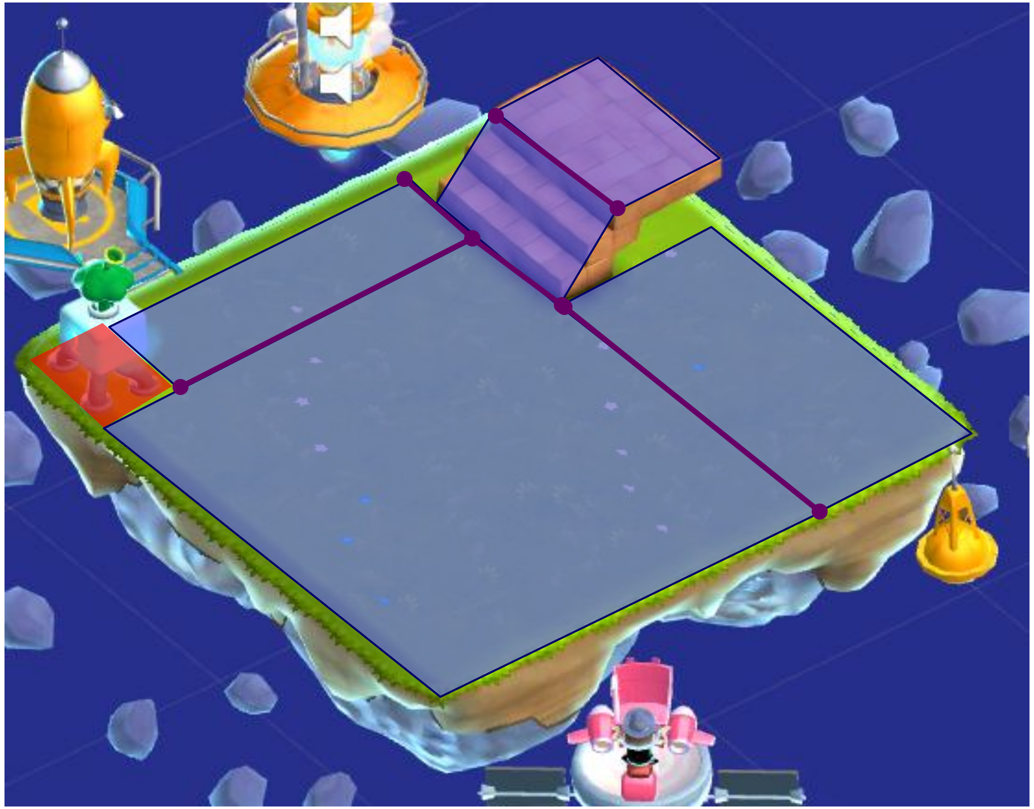


74 Nodes
114 Edges

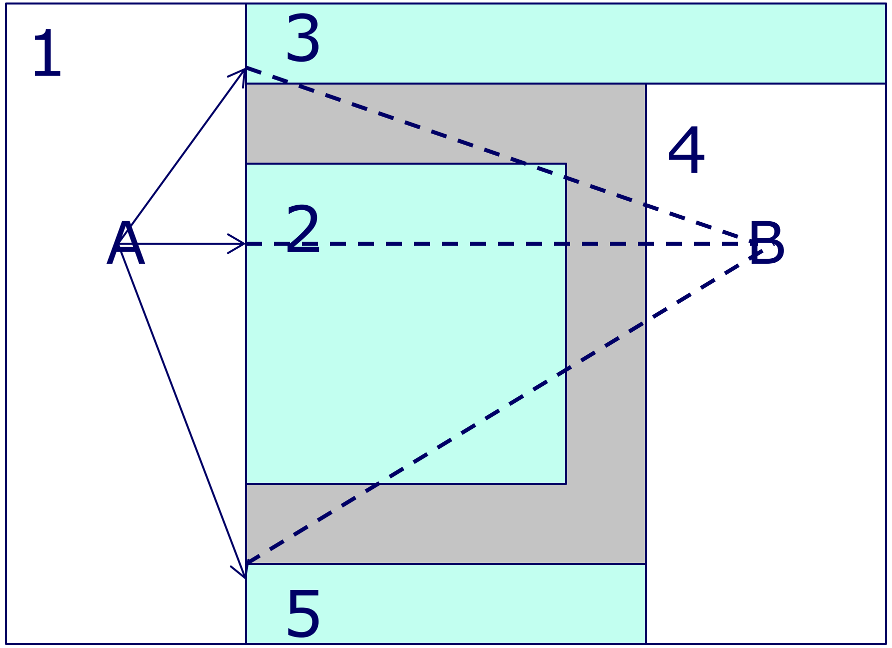


5 Nodes
5 Edges

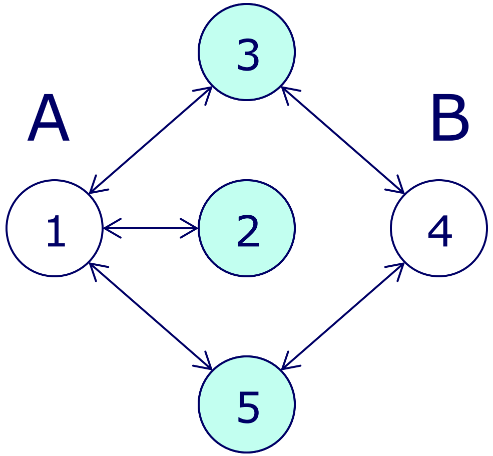
NavMesh in Supernauts



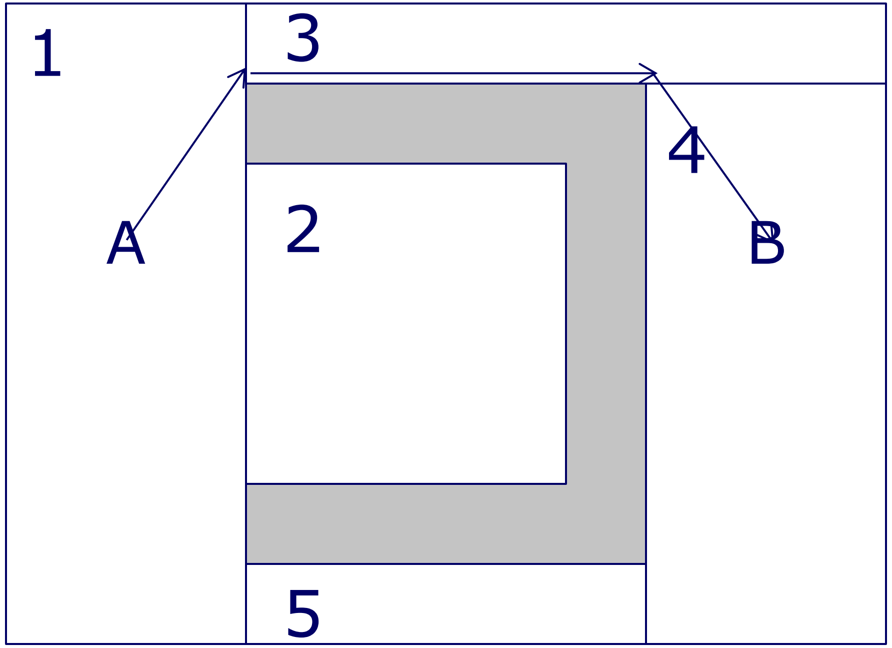
A* on NavMesh



- Similar than in grid
- Just iterate to **neighbour nodes** and use the **heuristics**

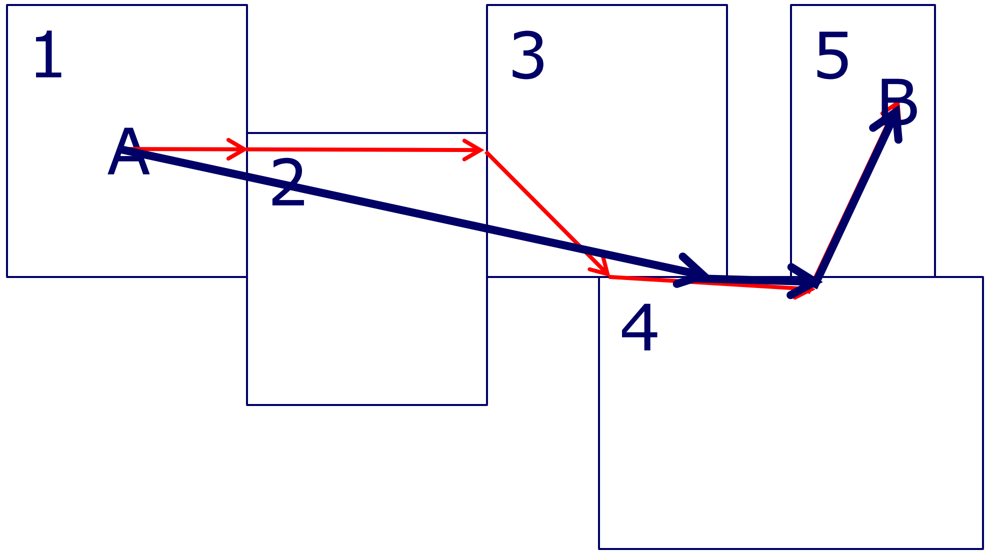


A* on NavMesh



$$\begin{tikzcd}
\overset{A}{\textcircled{1}} \arrow[r] & \textcircled{3} \arrow[r] & \overset{B}{\textcircled{4}}
\end{tikzcd}$$

Straight paths in NavMesh

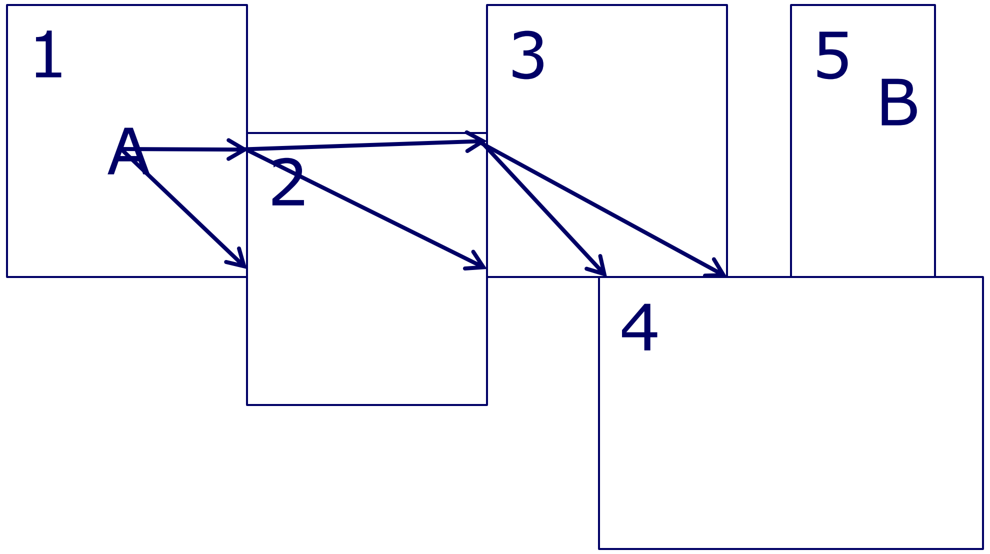


Shortest Path
is clearly:

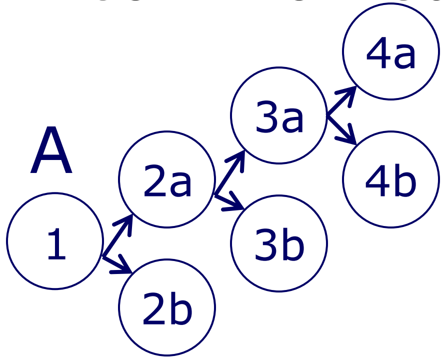
$$\begin{tikzcd} \overset{A}{\boxed{1}} \arrow[r] & \boxed{2} \arrow[r] & \boxed{3} \arrow[r] & \boxed{4} \arrow[r] & \overset{B}{\boxed{5}} \end{tikzcd}$$

But how to form
the path?

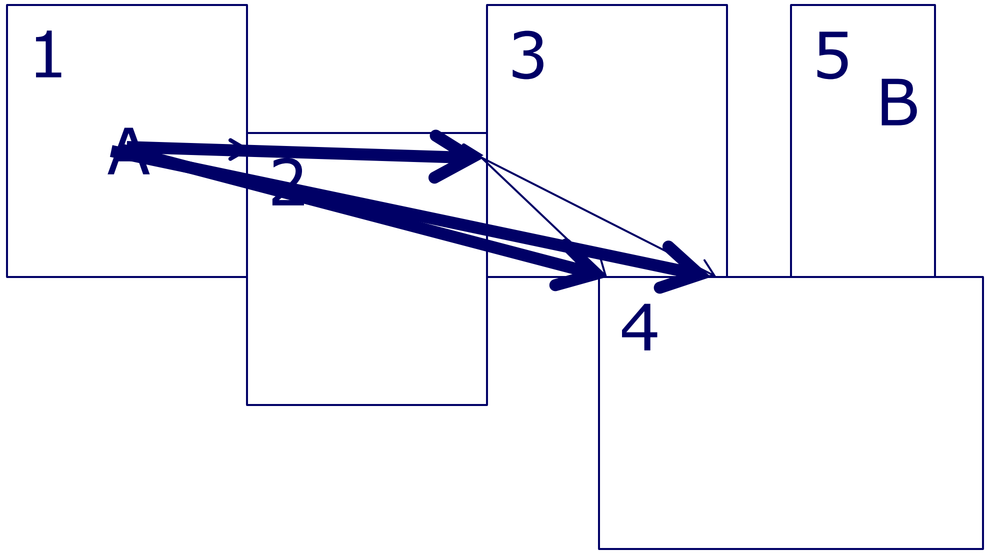
Straight paths in NavMesh



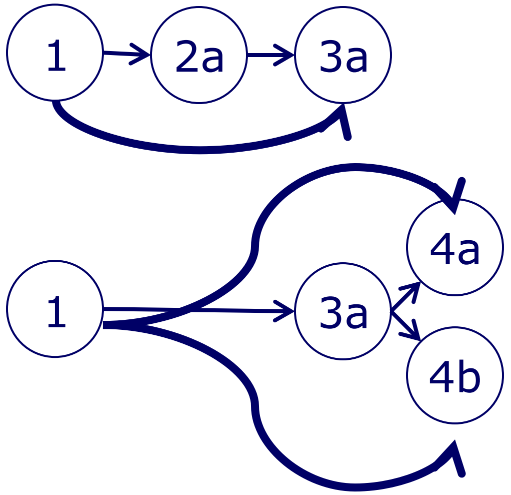
- Expand to neighbour nodes with two alternative routes
- Furthest ends of common edge



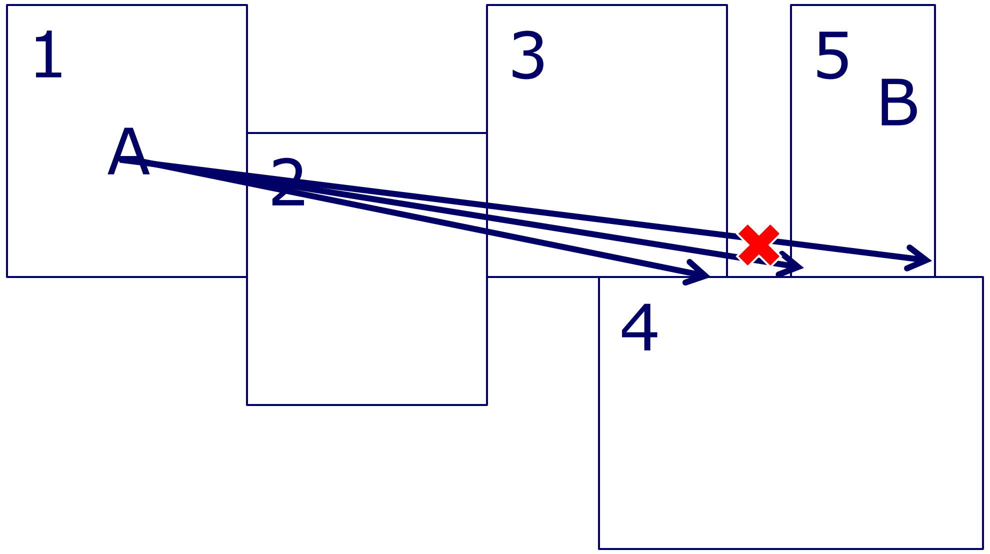
Straight paths in NavMesh

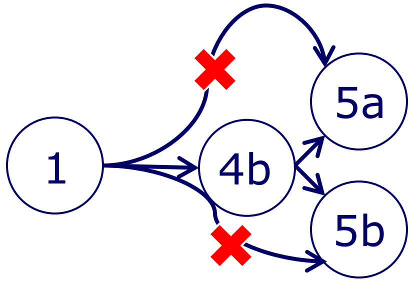


- On each iteration, trace path backwards as much as possible



Straight paths in NavMesh

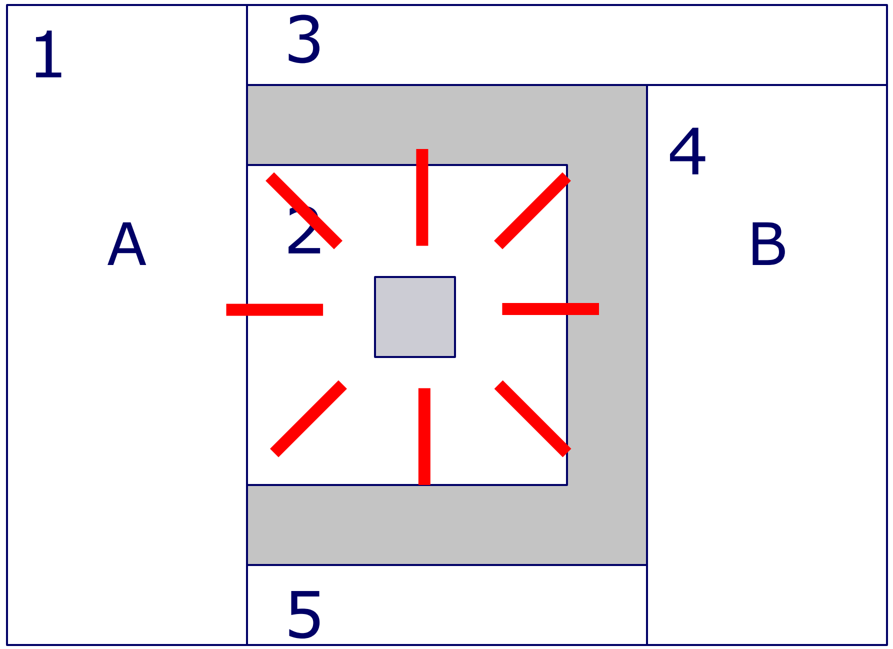


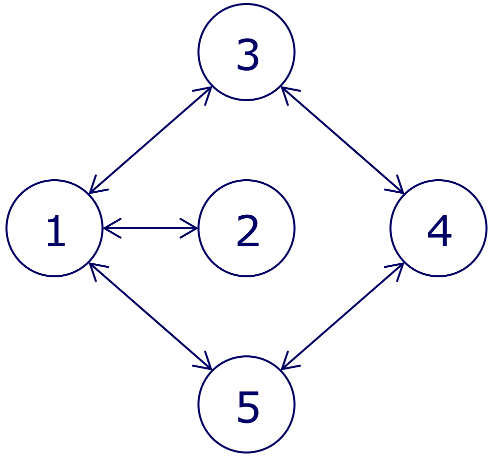


Generating NavMesh

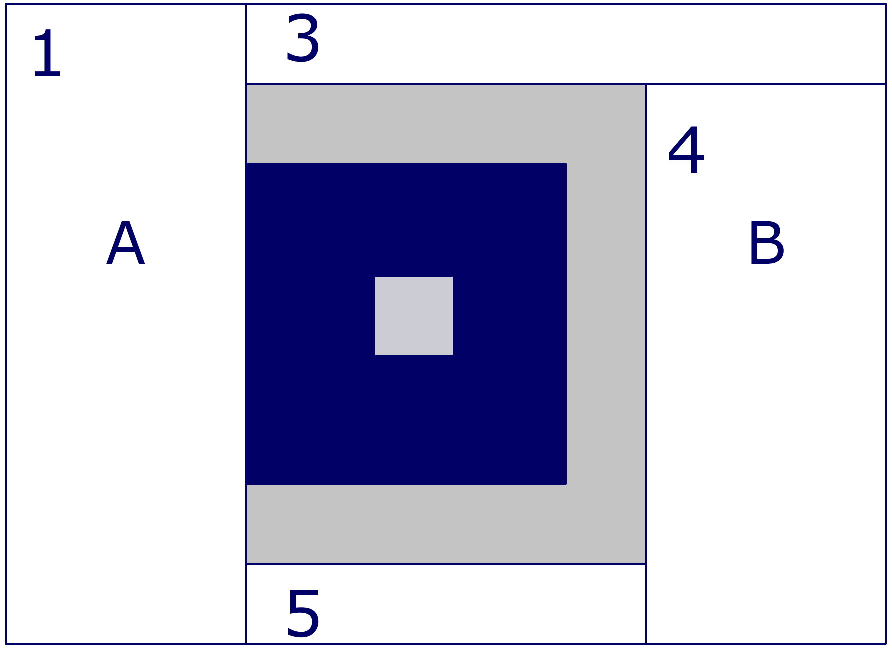
- Greedy Selection yields near-optimal result
- Iterate walkable and free positions and expand as much as possible
- Repeat until all positions filled

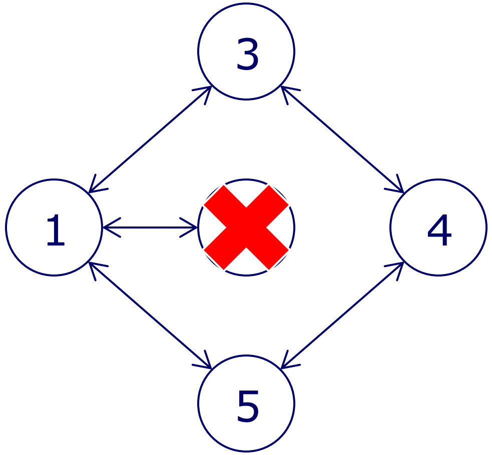
Updating NavMesh



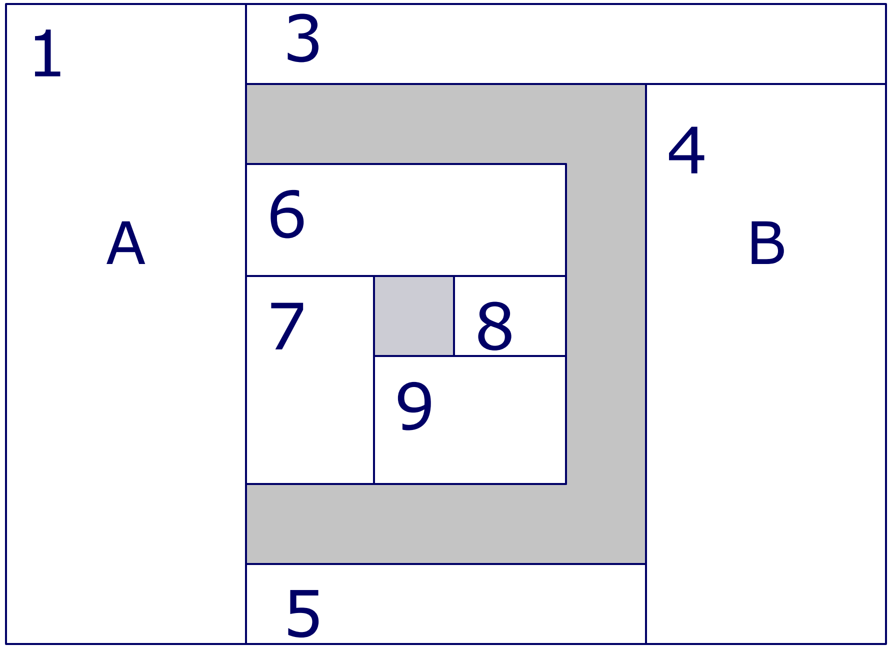


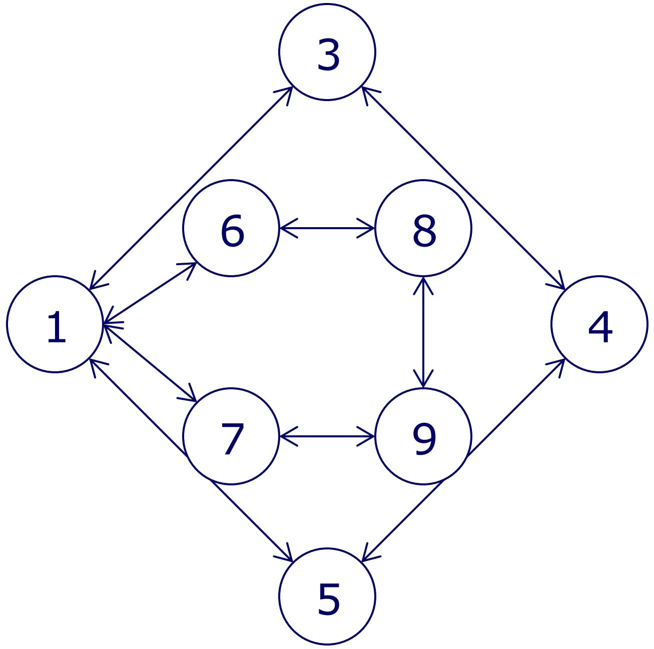
Updating NavMesh





Updating NavMesh





Research: Result

- We have a fast enough pathfind!
- But: Generating & updating is still slow!

Step 3: Innovate

- Now it's time for the most fun part: inventing something new!
- Let's break the problem apart:
  - A) Slow initialization
  - B) Slow update

A) Slow initialization

- Even simplest greedy selection iterates huge amount of blocks
  - Algorithm needs to find all surfaces and iterate them
    - Practically N^3 Complexity, huge amount of blocks
  - Because of user generated content and constantly changing environment, no precomputation is possible!

B) Slow update

- When changing environment on large node large area needs to be re-itarated.

Problems…

- So carefully analyzing we have manged to break the problem into smaller problems:
    - World has too many blocks to iterate over
    - Nodes are too large

…and their solutions

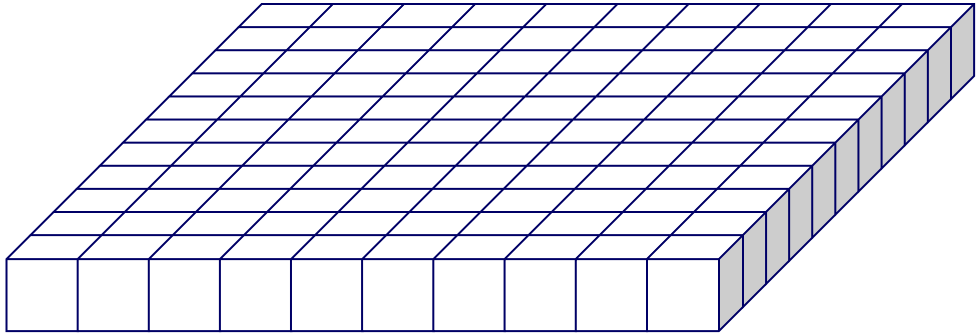
- Too many iterations
  - -> Iterate only blocks we need
- Too large nodes
  - -> Limit node size

Innovation!

- Too many iterations
  - -> Limit size of NavMesh!
  - -> Yields an upper limit for node size, no problematic updates!
- But small NavMesh can’t contain whole level
  - -> Let’s add as many small NavMeshes as needed!
  - -> Allows us to iterate areas that are only needed!

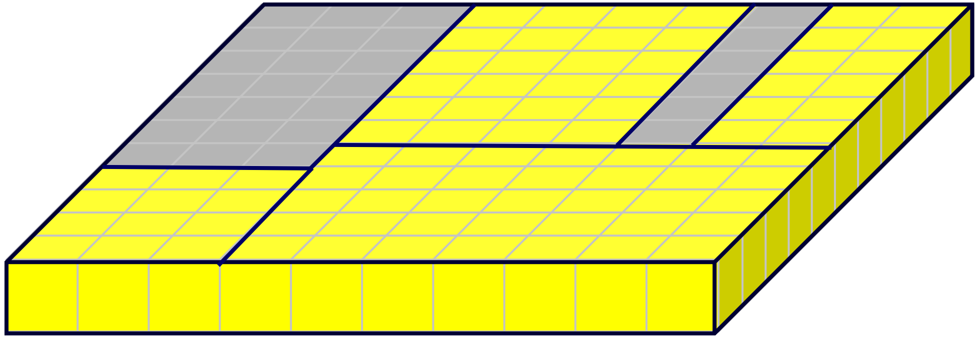
Supernauts NavMesh

- 10x1x10 mini navmesh



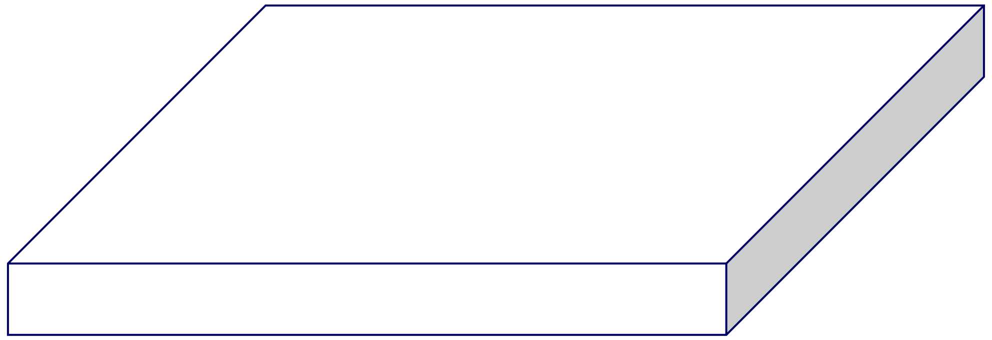
Supernauts NavMesh

- 10x1x10 mini navmesh



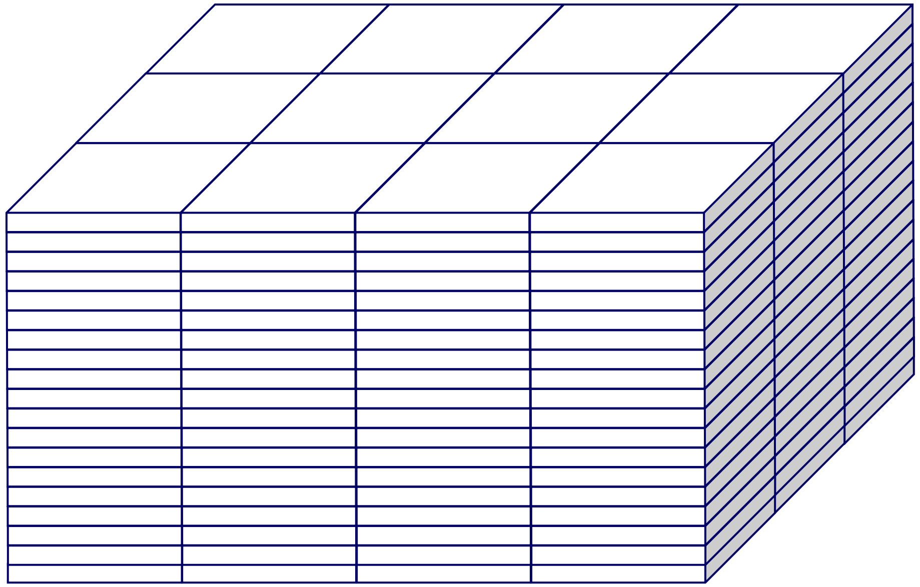
Supernauts NavMesh

- Whole world split evenly in mini NavMeshes

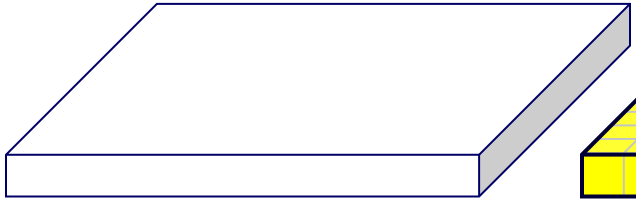


Supernauts NavMesh

- Whole world split evenly in mini NavMeshes

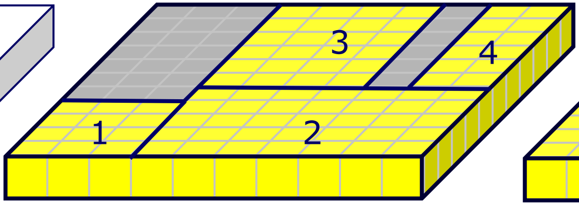


Mini NavMesh



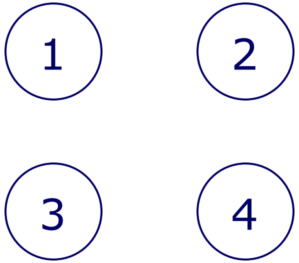
EMPTY state

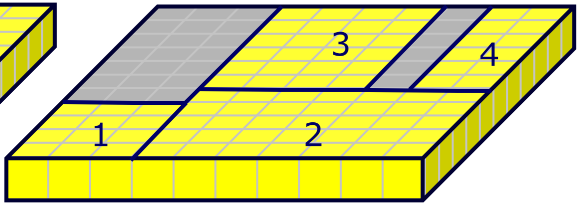
Can be just a NULL pointer.



BUILT state

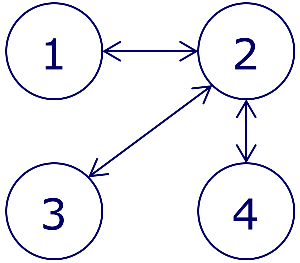
Nodes are built, but not connected



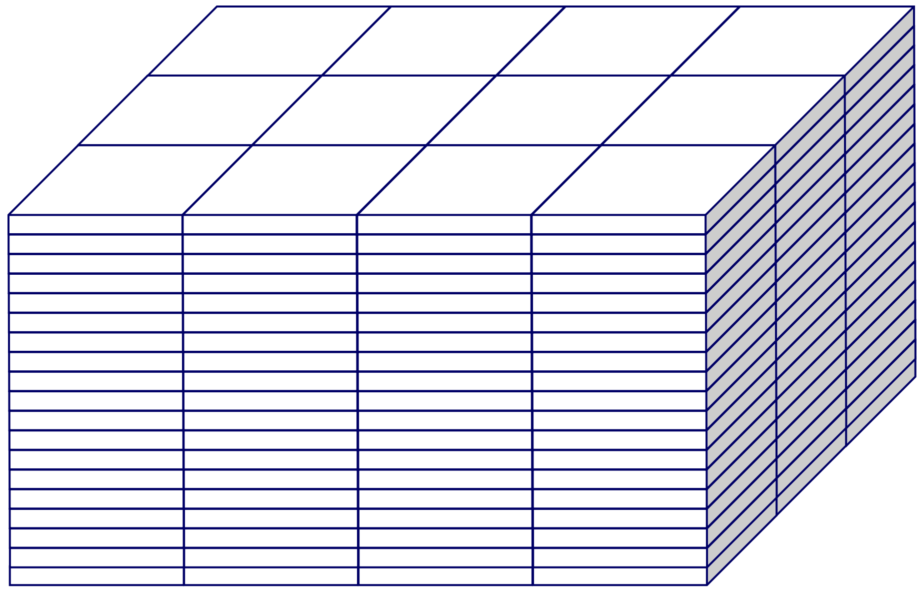


CONNECTED state

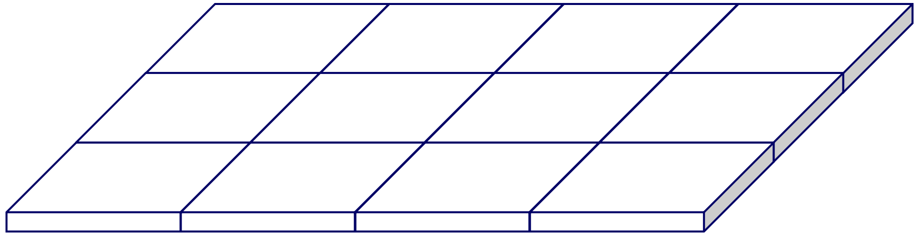
Nodes are built and connected



A* in Supernauts NavMesh

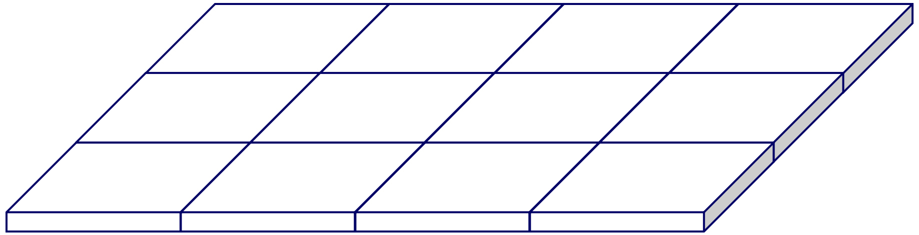


A* in Supernauts NavMesh

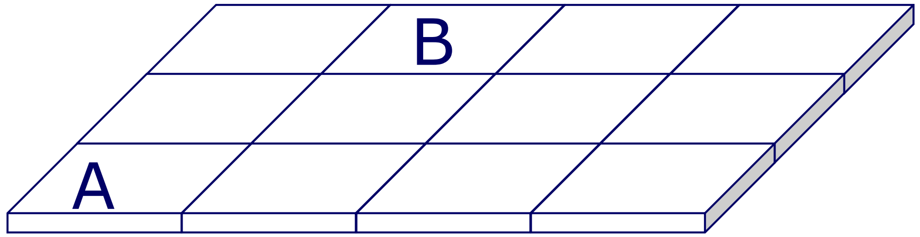


A* in Supernauts NavMesh

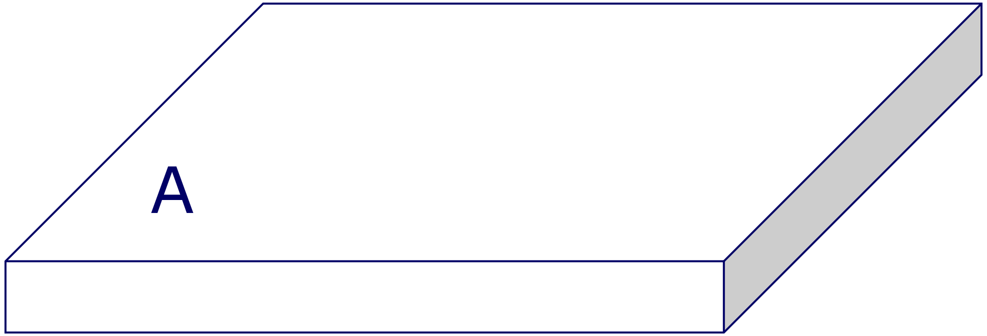
- Initially the whole world can be just array of NULL mini NavMeshes.



A* in Supernauts NavMesh

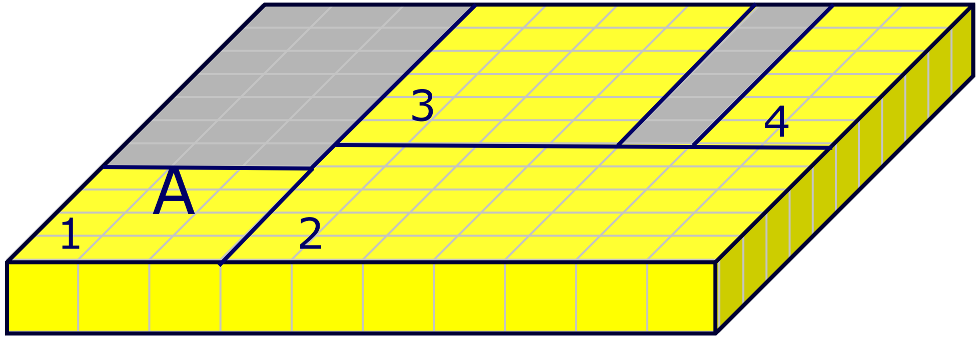


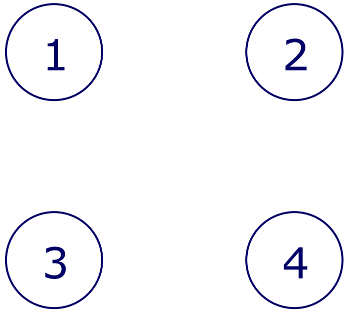
A* in Supernauts NavMesh



A* in Supernauts NavMesh

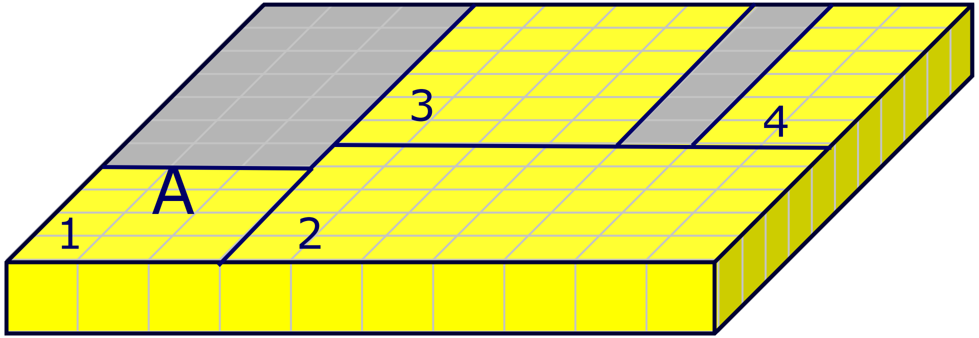
- Build nodes





A* in Supernauts NavMesh

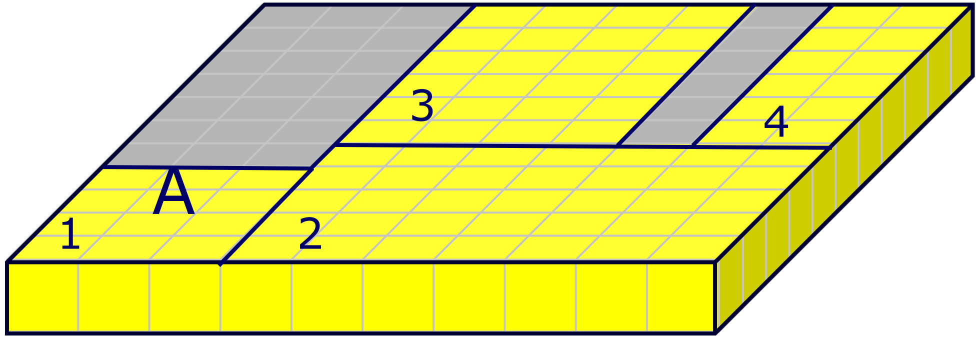
- …and connect the nodes
- But! That is not yet everything.
- We need to know connections to outside, otherwise we'll never get out!

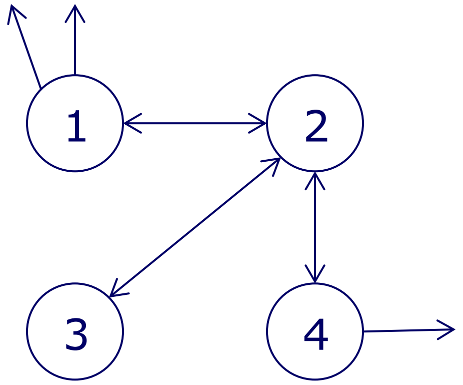


$$\begin{tikzcd}
1 \arrow[r, leftrightarrow] & 2 \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] \\
3 & 4
\end{tikzcd}$$

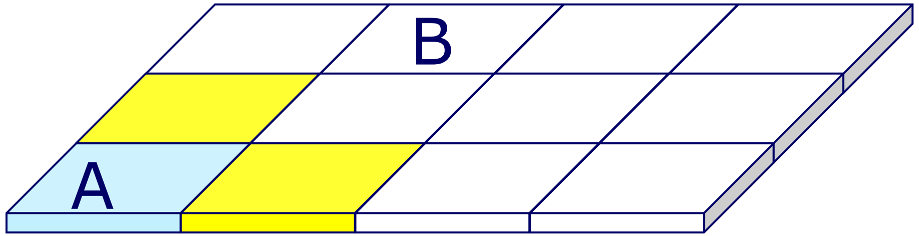
A* in Supernauts NavMesh

- We need to check all 4 x 3 neighbors because the character can climb 1 block and drop 1 block.
- Upgrade neighbouring nodes to BUILT state so we can connect nodes to outside.





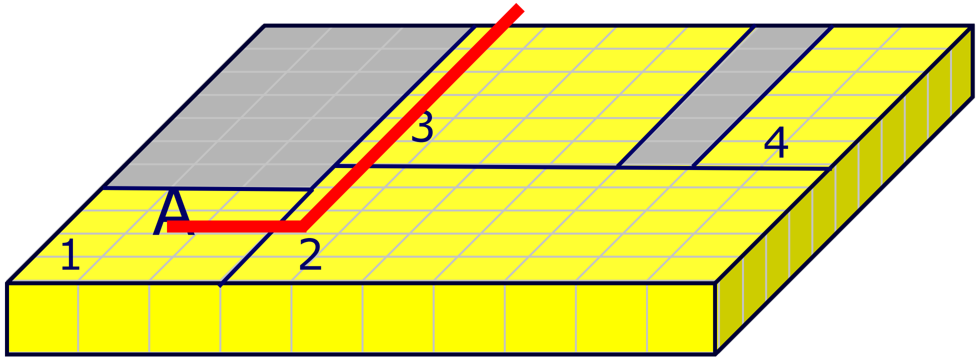
A* in Supernauts NavMesh

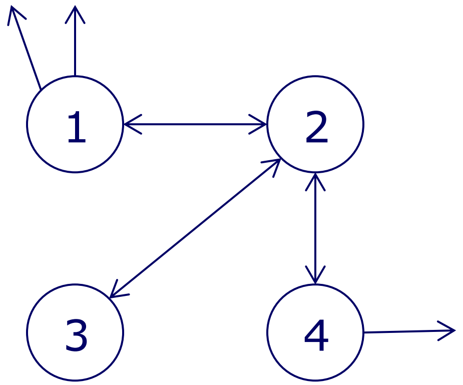


🟨 BUILT

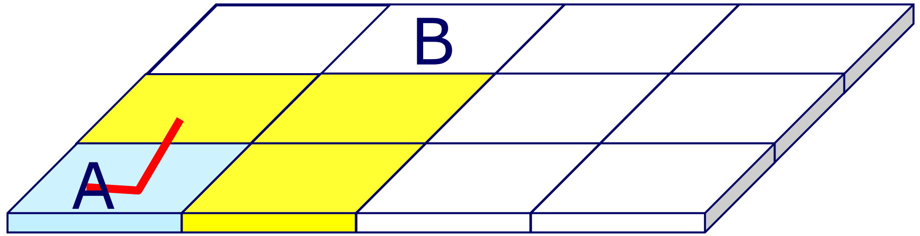
🟦 CONNECTED

A* in Supernauts NavMesh




$$\begin{tikzcd}
\boxed{1} \arrow[r, leftrightarrow] & \boxed{2} \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] & \\
\boxed{3} & \boxed{4} \arrow[r] &
\end{tikzcd}$$

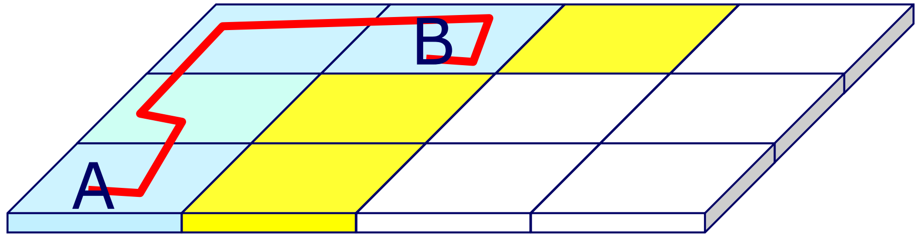
A* in Supernauts NavMesh



🟨 BUILT

🟦 CONNECTED

A* in Supernauts NavMesh



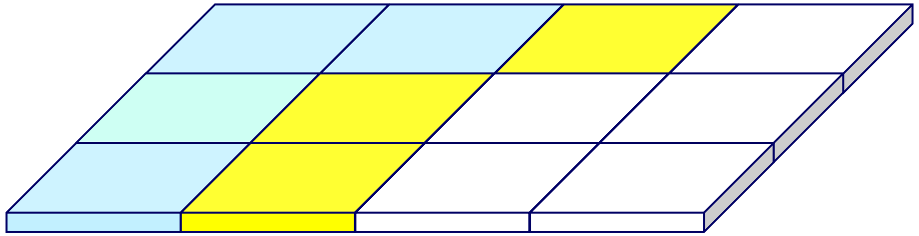
🟨 BUILT

🟦 CONNECTED

Updating Supernauts NavMesh

- What about updating?
  - Solution is surprisingly simple!

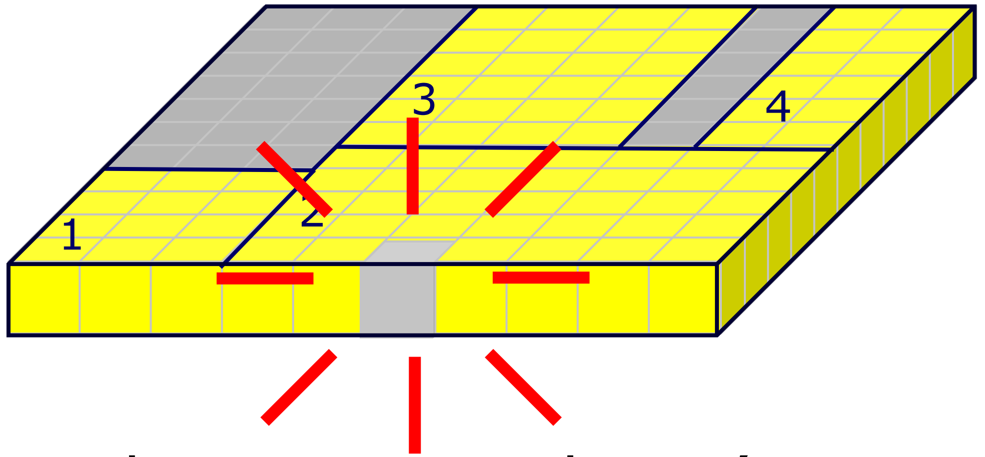
Updating Supernauts NavMesh

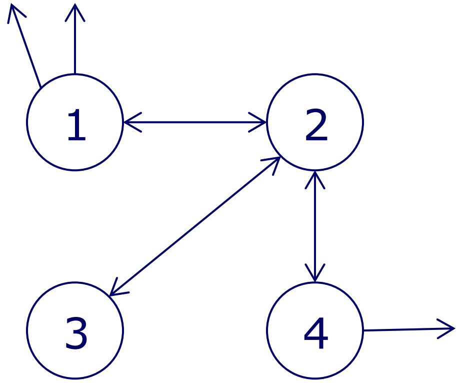


🟨 BUILT

🟦 CONNECTED

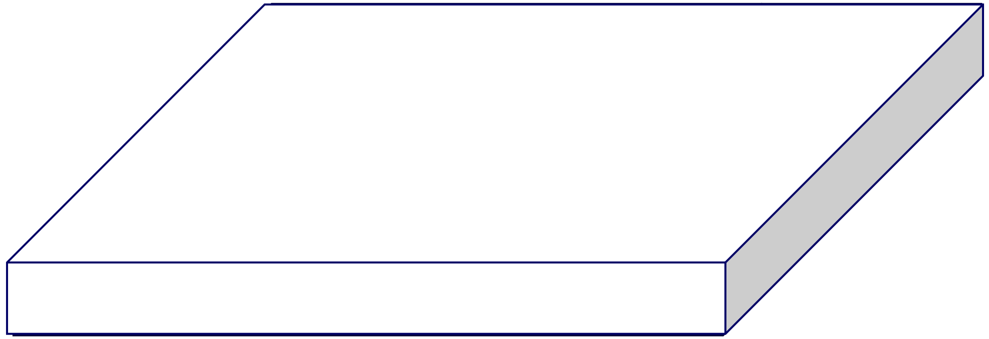
Updating Supernauts NavMesh





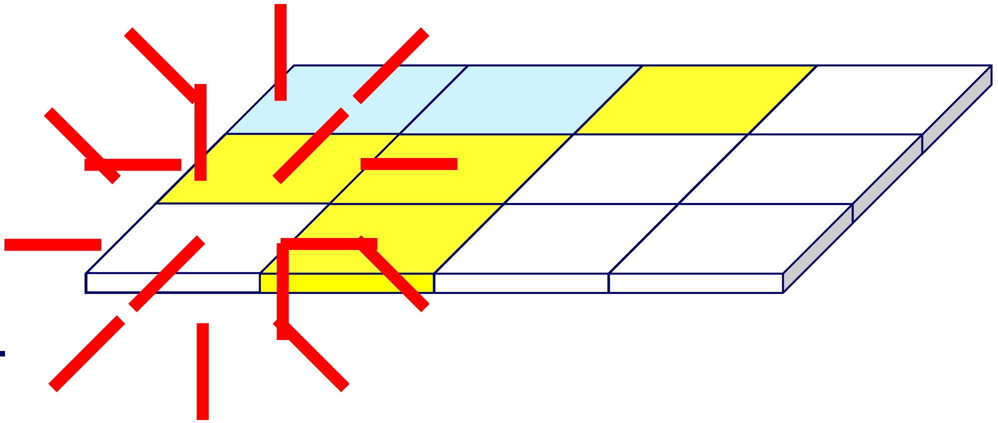
- The structure doesn't represent reality anymore
- Because mini NavMesh is tiny…

Updating Supernauts NavMesh



- The structure doesn't represent reality anymore
- Because mini NavMesh is tiny…
- We can just dispose it!

Updating Supernauts NavMesh

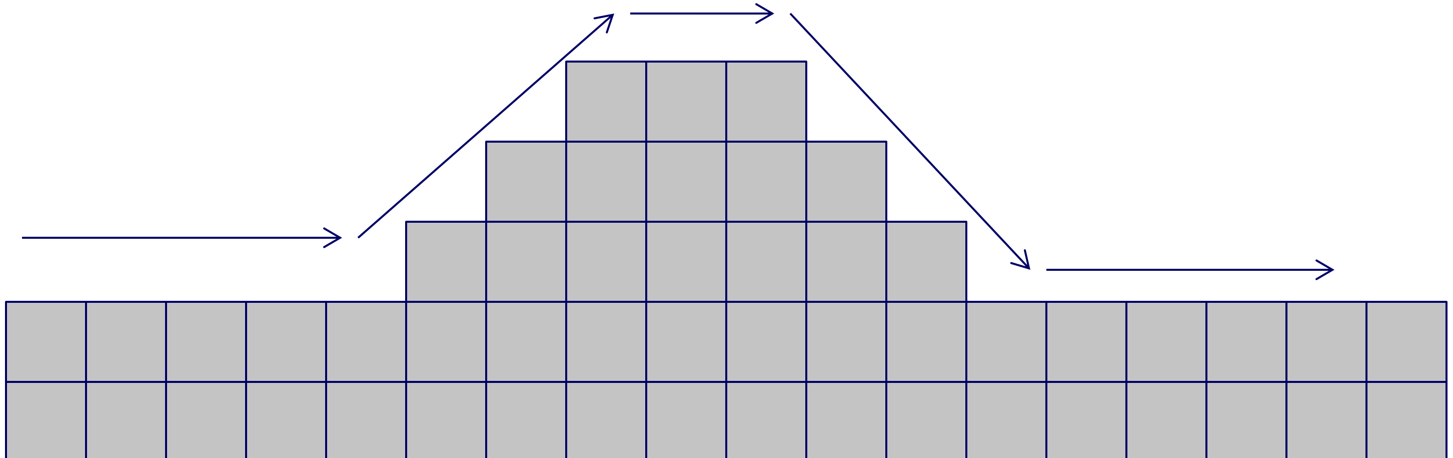


🟨 BUILT

🟦 CONNECTED

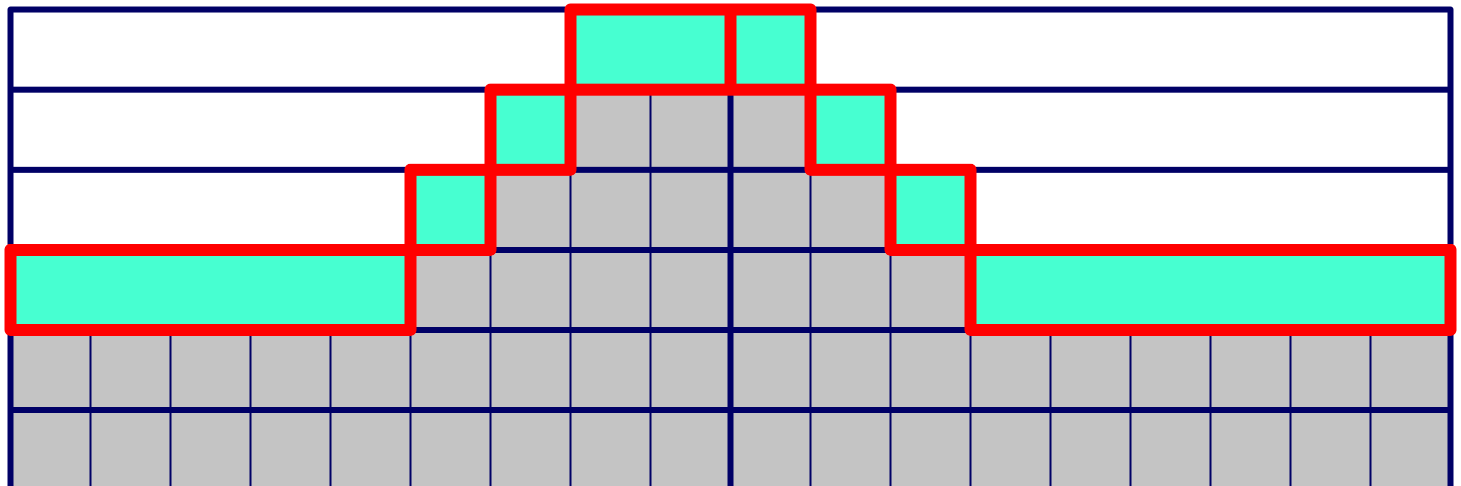
Improving Supernauts NavMesh

- In Supernauts characters can walk stairs up and down.
- This can be used to reduce number of nodes



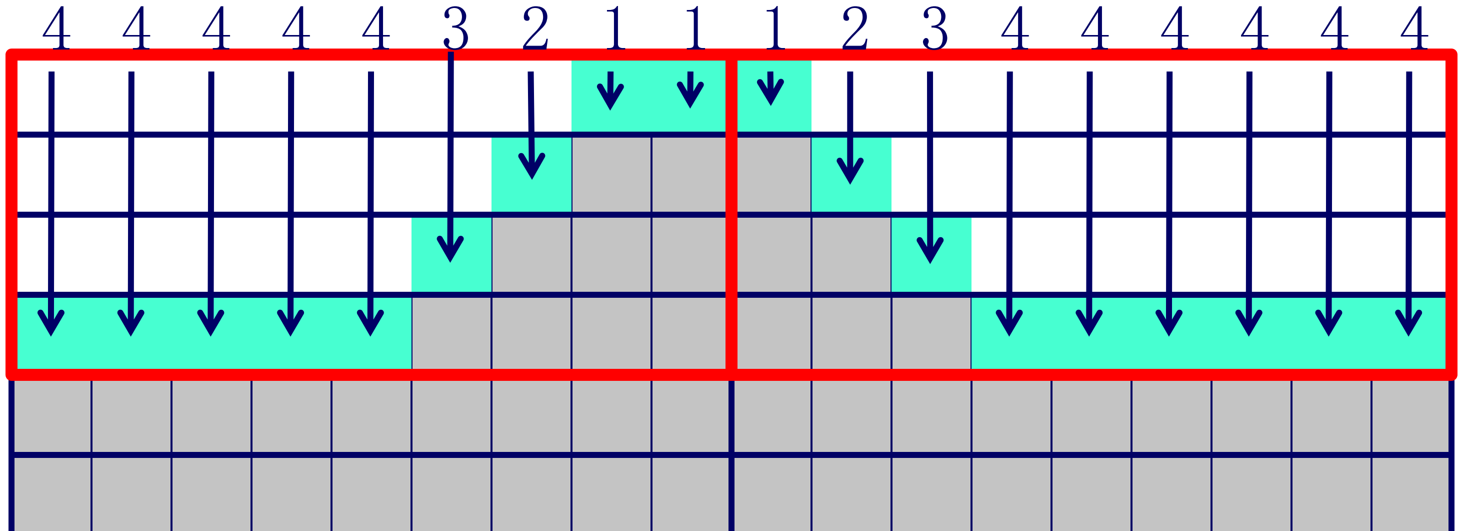
Improving Supernauts NavMesh

- -> 8 nodes
- The character **can move freely** along the nodes.
- Can we merge them in any way?



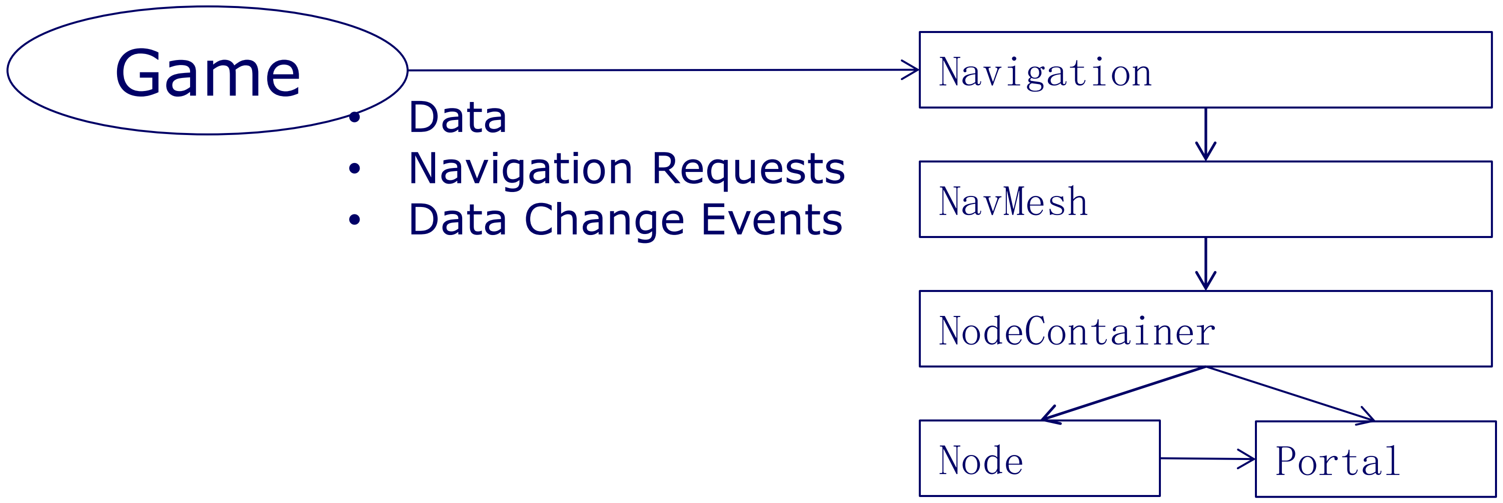
Improving Supernauts NavMesh

- Multiple mini NavMeshes share the same node
- A **height field** for each node to help distuingish overlapping nodes.



Video demonstrating NavMesh in practice

Example Implementation

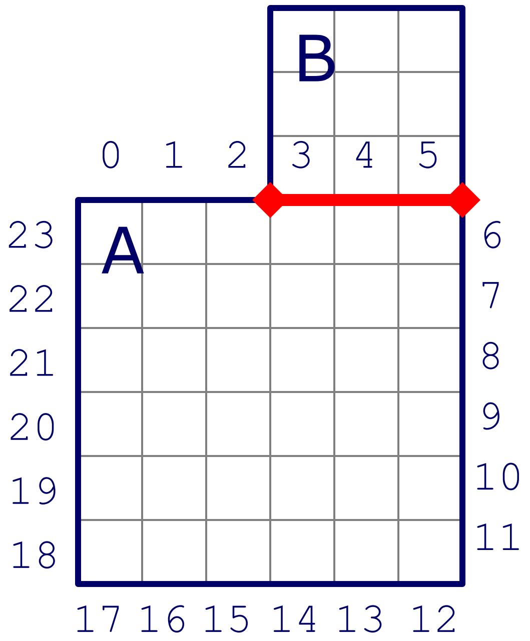


- Data
- Navigation Requests
- Data Change Events

```
class Node {
    int id;
    IntBounds bounds;
    int [,] heightField;
    List <Portal> connections;
};

class Portal {
    int positionA, positionB;
    int nodeA, nodeB;
};
```

Node & Portal



```
new Portal {
    positionA = 3,
    positionB = 6,
    nodeA = A,
    nodeB = B
};
```

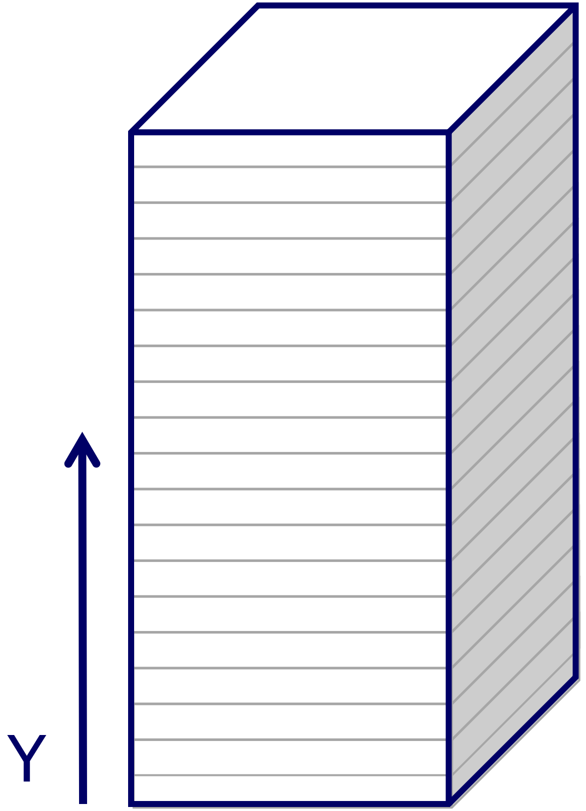
```
class NavMesh {
    private NodeContainer [,] nodeLookUp;

    public Node GetAndCreateNode(Vector3 point);
    public Node GetNode(int id);
};

class NodeContainer {
    private int [,,] positionToNode;
    private State [] sliceStates;
    private HashSet<int> [] nodeSetsPerSlice;
    private Dictionary<int, Node> nodesIndexed;
    // ...
};
```

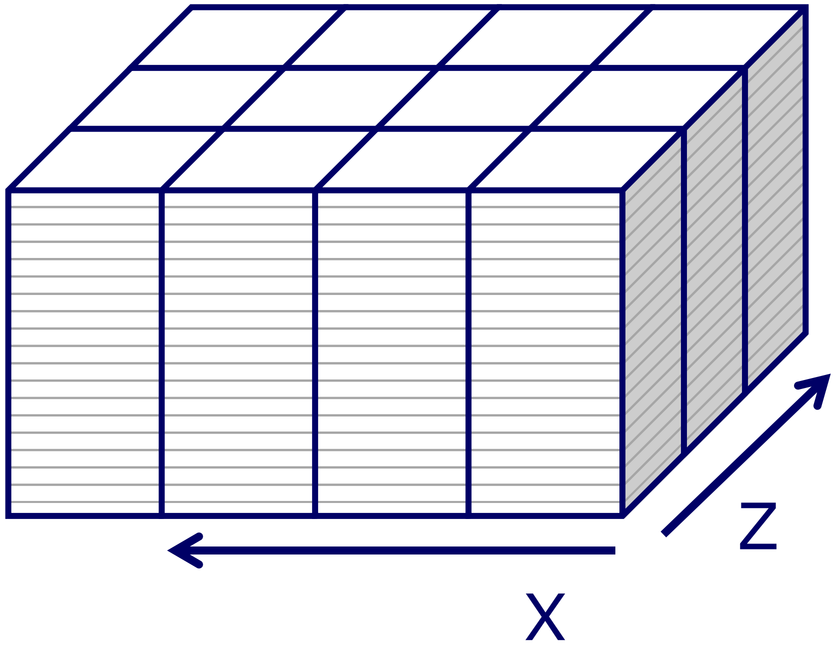
NodeContainer

```
class NodeContainer {
    private int [,,] positionToNode;
    private State [] sliceStates;
    private HashSet<int> [] nodeSetsPerSlice;
    private Dictionary<int, Node> nodesIndexed;
    // ...
};
```



NavMesh

```
NodeContainer [,] nodeLookUp;
```



```
class Navigation {
    private NavMesh navMesh;

    public Path FindPath(Vector3 start, Vector3 target);
    public Path FindPath(Vector3 start, Bounds area);

    public bool PathExists(Vector3 start, Vector3 target);

    public Vector3 GetRandomPosition(Bounds area);
    public Vector3 GetRandomReachablePosition(Vector3 point);

    public void Invalidate(Bounds area);
};
```

```
public Path FindPath(Vector3 start, Vector3 target);
public Path FindPath(Vector3 start, Bounds area)
```

- Destination doesn’t need to be a point, it can be an area.
- Simply end the algorithm when it enters the bounds.

```
public bool PathExists(Vector3 start, Vector3 target);
public bool PathExists(Vector3 start, Vector3 target);
```

- Simply querying if path exists is much faster than finding the path, because you don’t have to form the path itself.

```
public Vector3 GetRandomPosition(Bounds area);
public Vector3 GetRandomReachablePosition(Vector3 point);
```

- You can also get **evenly distributed** random positions easily.
- Simply weight the random selection by node surface area.

Conclusion

- Each game has unique requirements.
- Complicated problem can become simple when
  - Divided in small pieces
  - Iteratively developed better instead of doing the most advanced solution at first
- Avoid doing more than your game actually needs!

Thank You!

- Q&A
- harri.hatinen@grandcrugames.com