What Makes Games Tick: Demystifying AI for Games

WinHacks 2026

Steven Rice

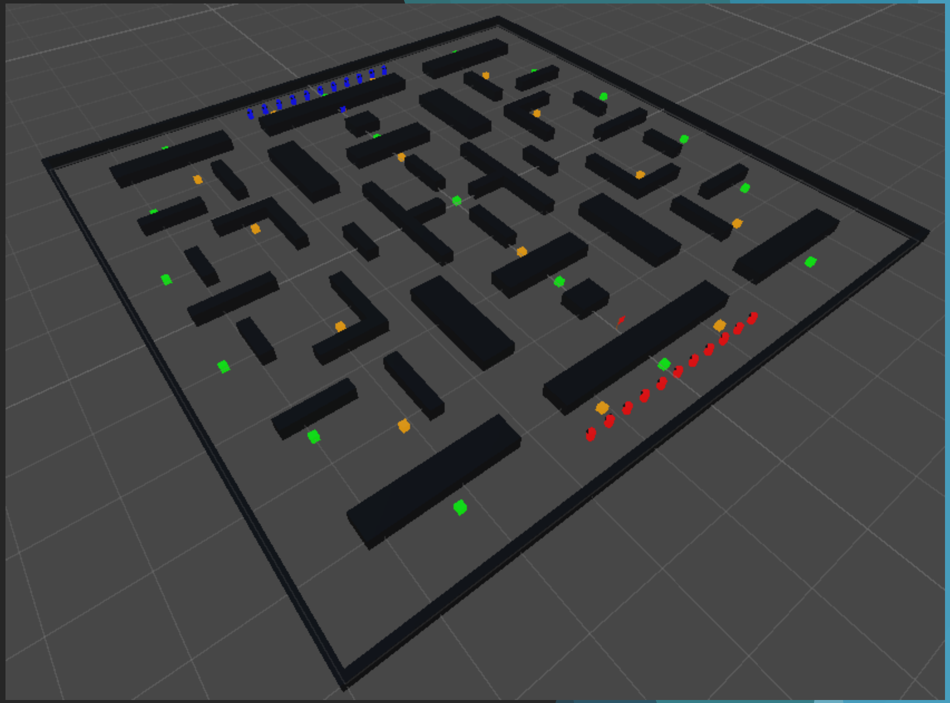
About Me

- PhD in Computer Science
- MSc in Computer Science – 2025
  - Vector Scholarship in AI
- BSc in Computer Science – 2024
- BCom in Business Administration – 2023
- Minor in Mathematics
- ✨Professional Student✨

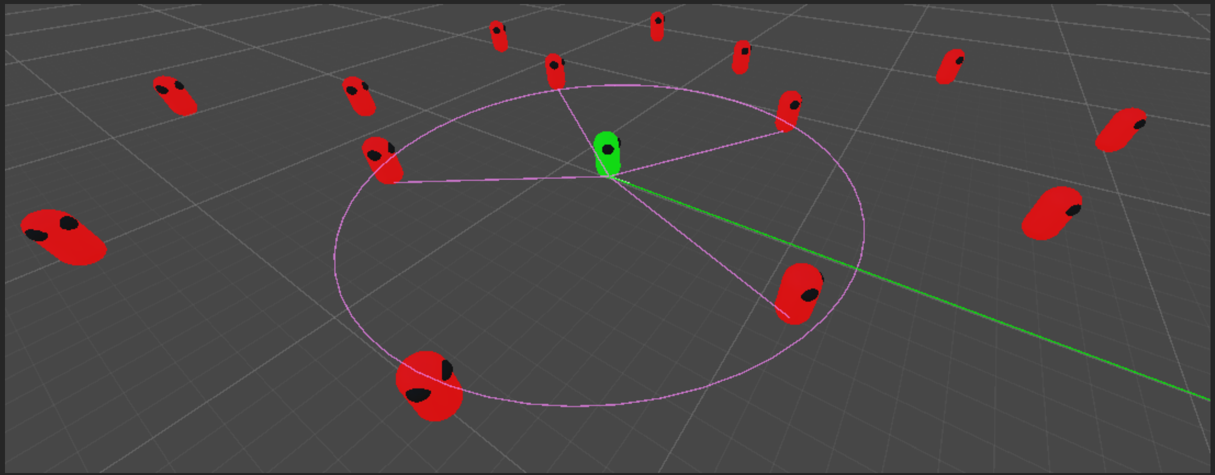
VECTOR INSTITUTE | INSTITUT VECTEUR

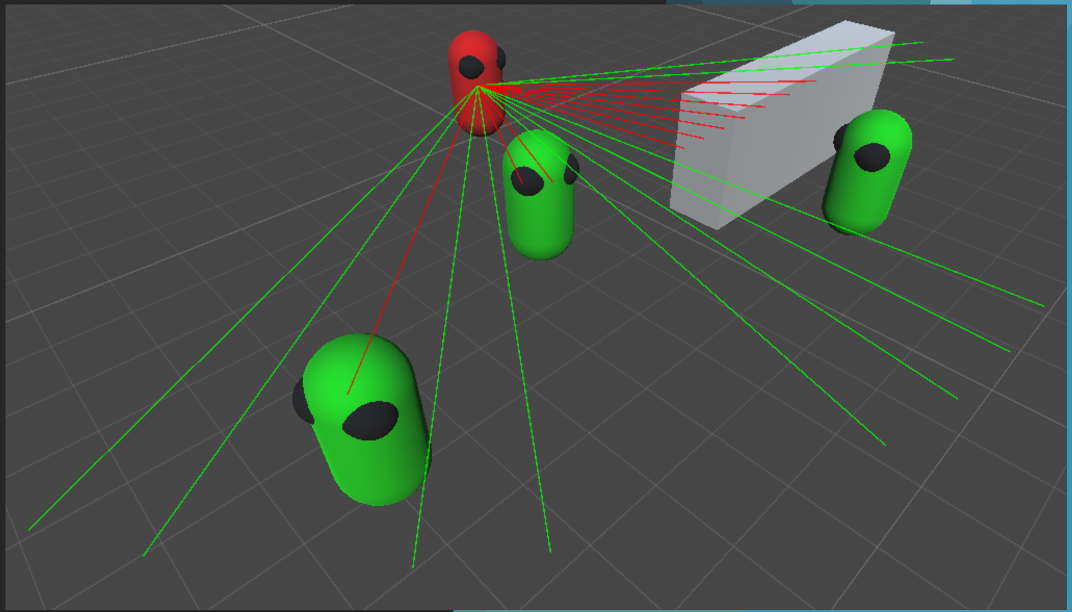
About Me

UNIVERSITY OF
DETROIT
MERCY

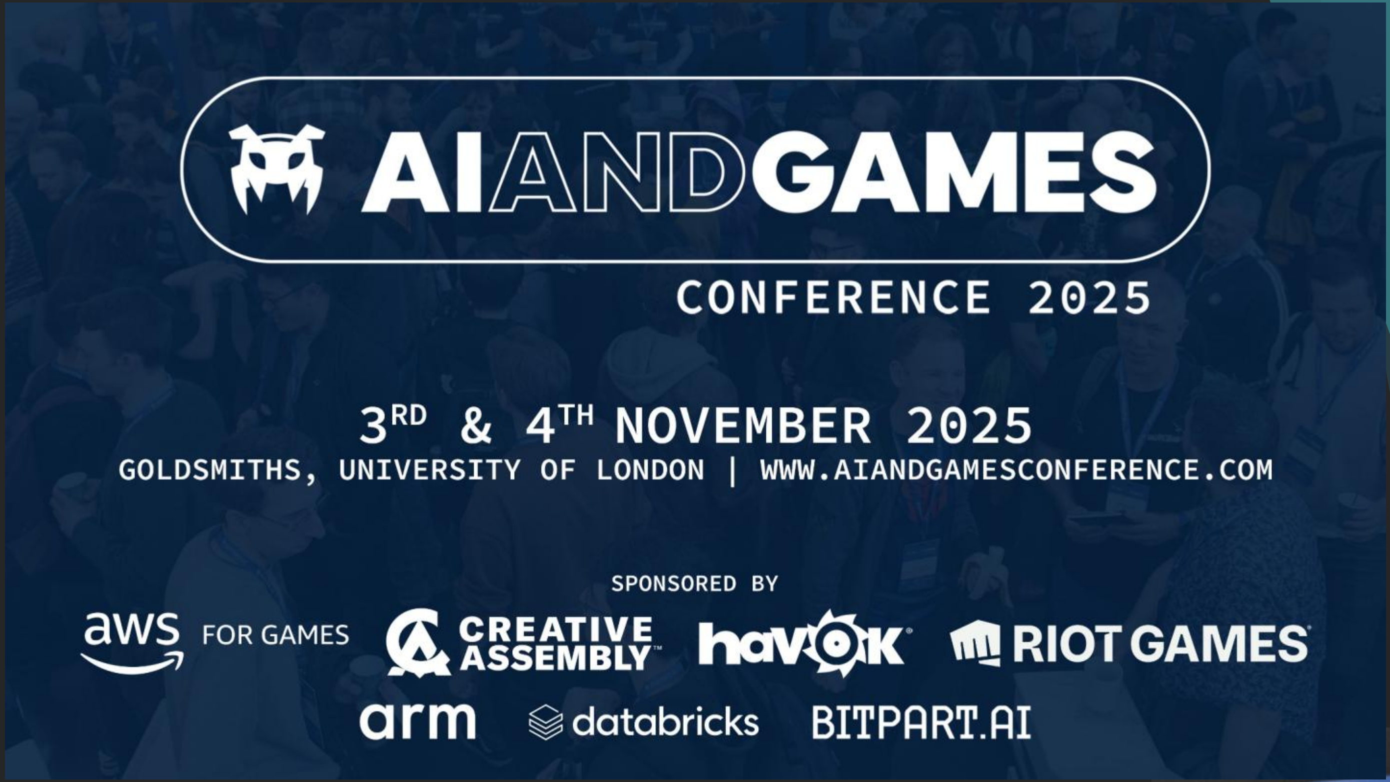


- Game design always been my passion
  - Never the time or attention span to release anything 😞
    - Yet 😳
  - Designed and taught first game development course at UDM
  - Teaching AI for Games here
    - COMP-4770





About Me


AIANDGAMES

CONFERENCE 2025

3RD & 4TH NOVEMBER 2025

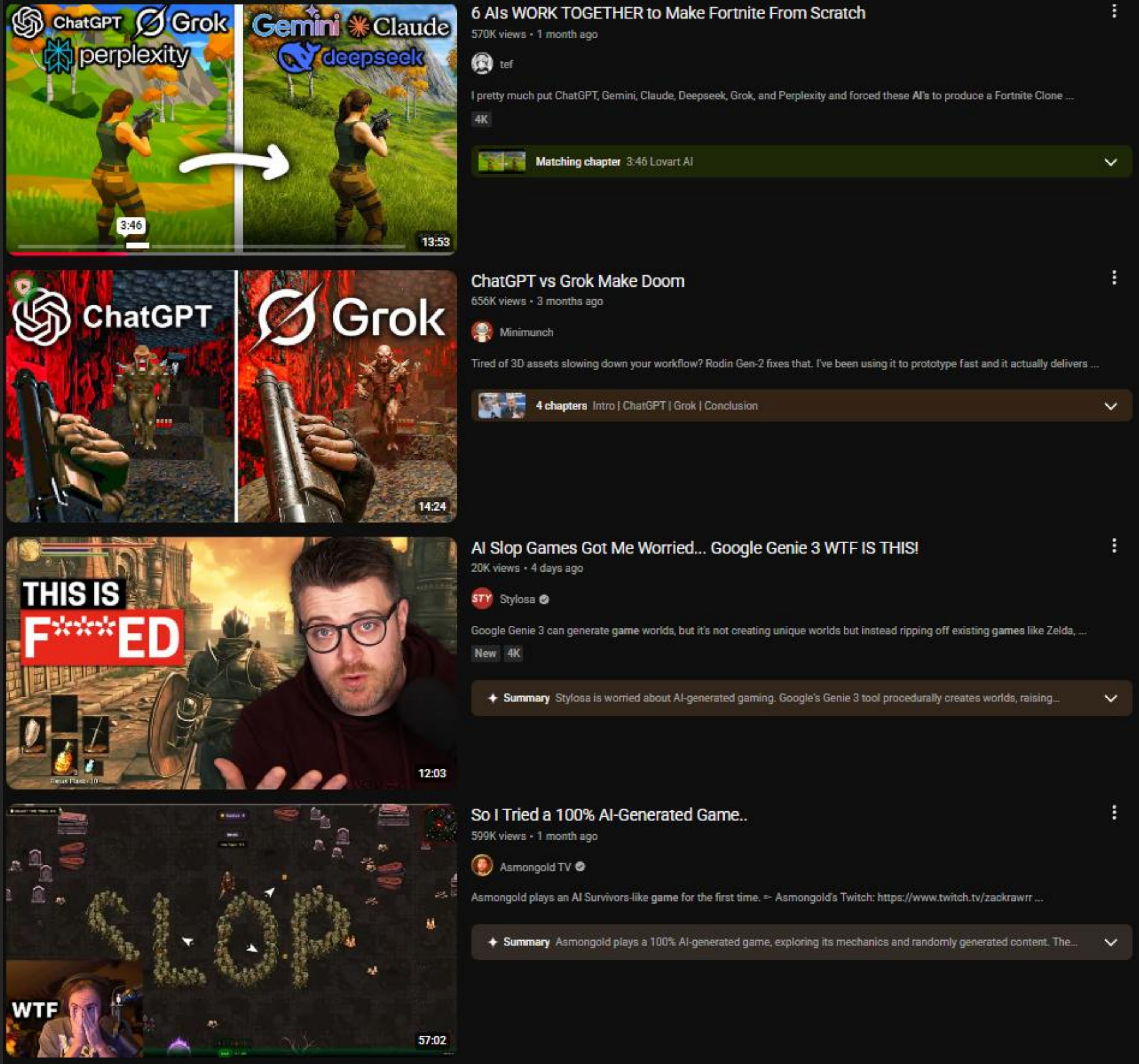
GOLDSMITHS, UNIVERSITY OF LONDON | WWW.AIANDGAMESCONFERENCE.COM

SPONSORED BY

aws FOR GAMES CREATIVE ASSEMBLY havok RIOT GAMES arm databricks BITPART.AI

Overview

- If you were to look up AI for games, you'll find plenty on common ways they are designed
  - Well, today you might find more of AI just making games…



6 AIs WORK TOGETHER to Make Fortnite From Scratch
570K views • 1 month ago
tef
I pretty much put ChatGPT, Gemini, Claude, Deepseek, Grok, and Perplexity and forced these AI's to produce a Fortnite Clone ...
4K
Matching chapter 3:46 Lovart AI

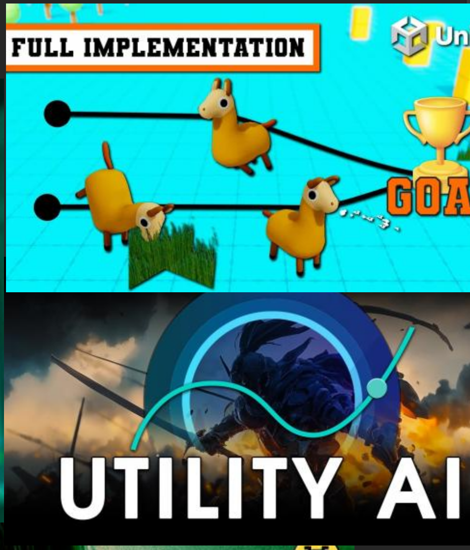
ChatGPT vs Grok Make Doom
656K views • 3 months ago
Minimunch
Tired of 3D assets slowing down your workflow? Rodin Gen-2 fixes that. I've been using it to prototype fast and it actually delivers ...
4 chapters Intro | ChatGPT | Grok | Conclusion

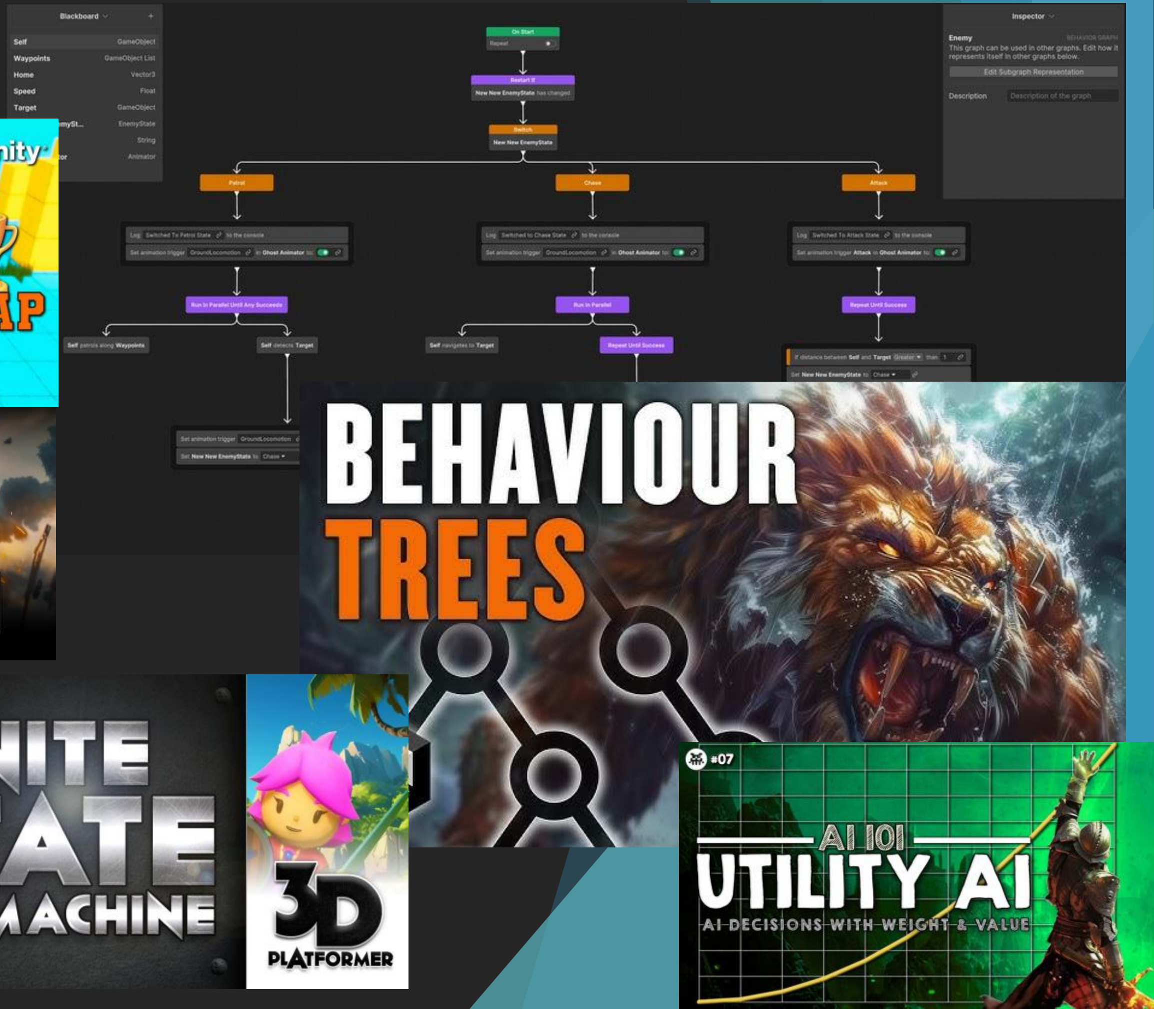
AI Slop Games Got Me Worried... Google Genie 3 WTF IS THIS!
20K views • 4 days ago
Stylosa
Google Genie 3 can generate game worlds, but it's not creating unique worlds but instead ripping off existing games like Zelda, ...
New 4K
✦ Summary Stylosa is worried about AI-generated gaming. Google's Genie 3 tool procedurally creates worlds, raising...

So I Tried a 100% AI-Generated Game..
599K views • 1 month ago
Asmongold TV
Asmongold plays an AI Survivors-like game for the first time. ➾ Asmongold's Twitch: https://www.twitch.tv/zackrawrr ...
✦ Summary Asmongold plays a 100% AI-generated game, exploring its mechanics and randomly generated content. The...

Overview

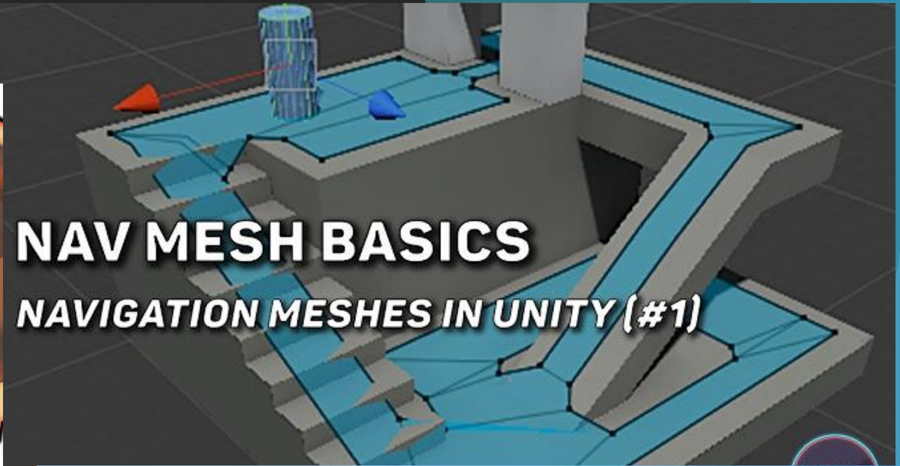
- Common AI development methods are easy to find
  - Handle decision making

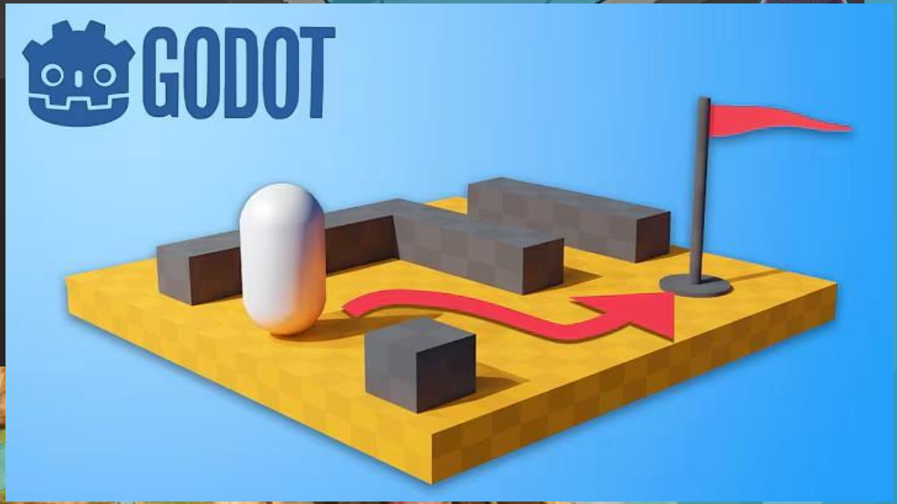


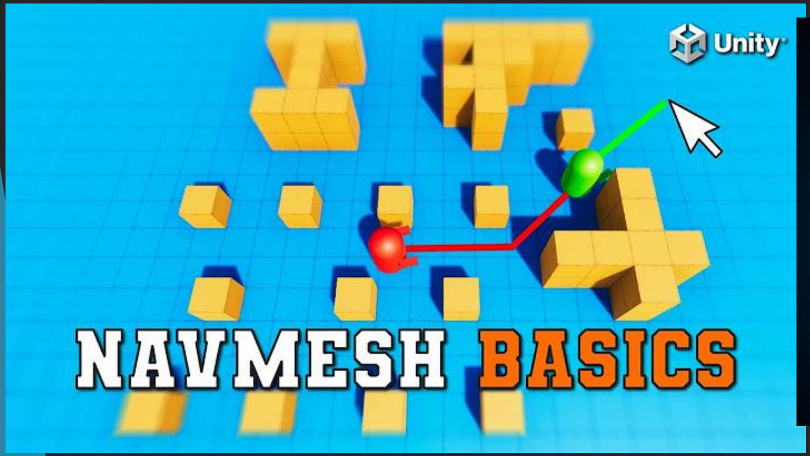


Overview

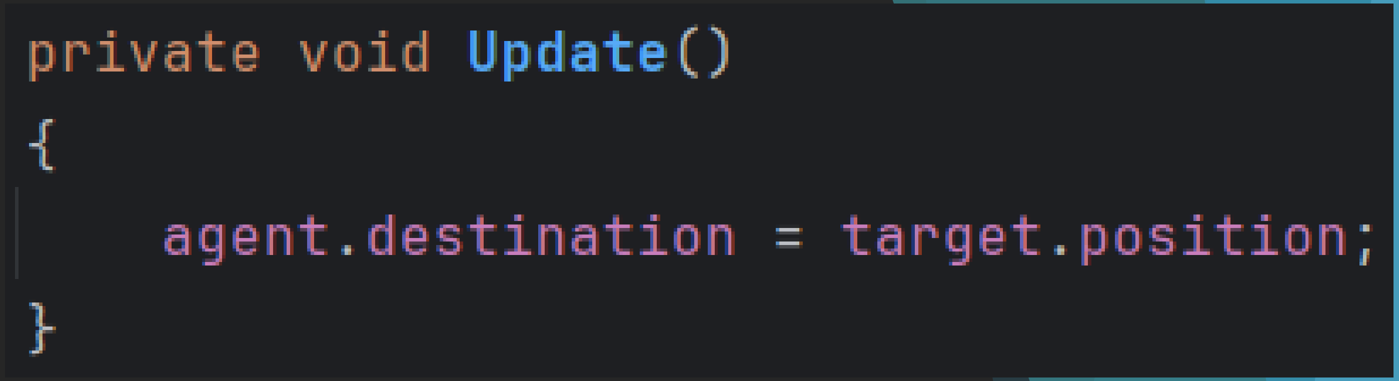
- Find a lot on movement
  - But never explaining how
  - Just using tools provided by the engine







Movement in Games



```
private void Update()
{
    agent.destination = target.position;
}
```

- Often treated as a black box
- Majority of people who play or even make games likely will never know how they move
  - Nothing wrong with this
    - These tools exist to make development easier
  - What happens if some functionality is missing?
  - How can you figure out why a character is moving a certain way?
- Let’s not be part of that majority

Movement in Games

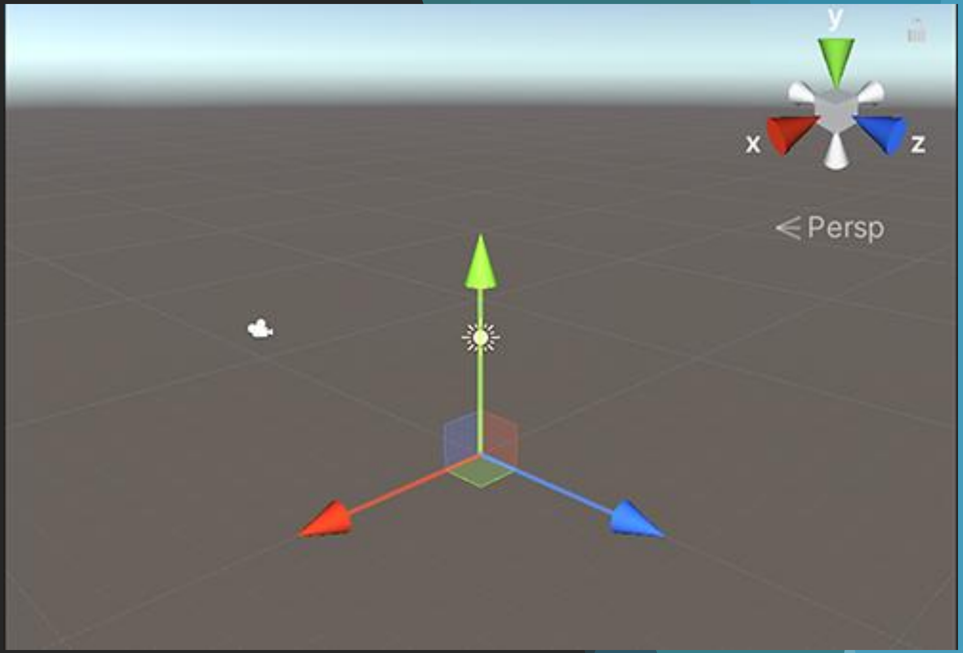
1. Basic movements
   - Move to a position
2. Compound movements
   - Follow a path
3. Navigation

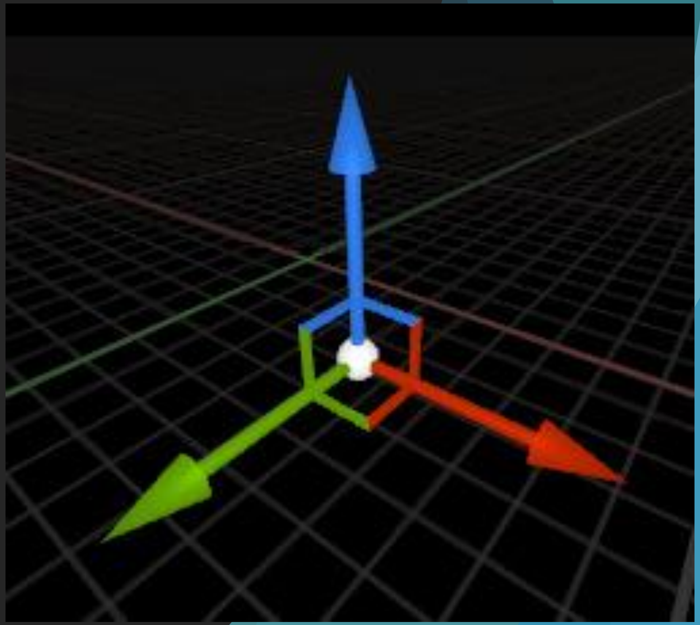
Step 0: Math 🤮

- Groundwork for our first basic movement
- What represents positions of objects in games?
  - Vectors
- If we want to move from one position to another position, what are we doing?
  - Current position vector → ??? → Target position vector

Vectors for positioning

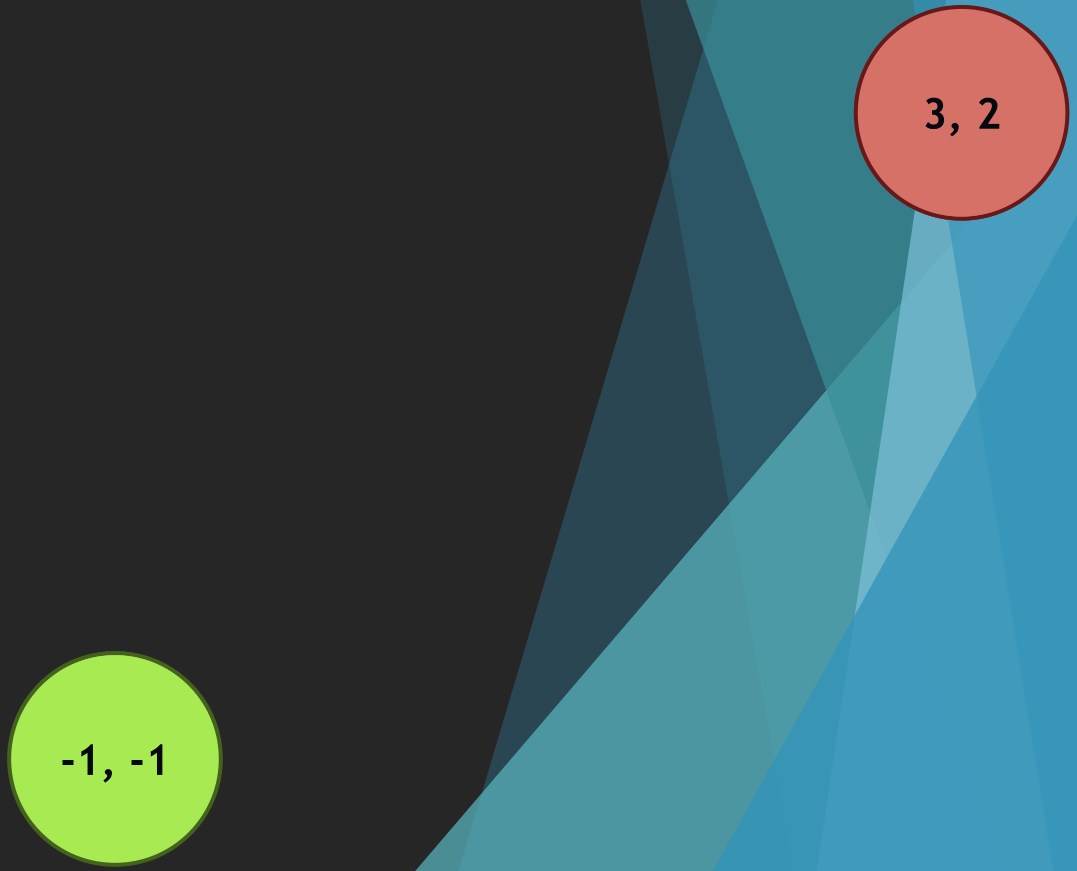
- Slightly different representations over every game engine
  - What axes means what
  - What “hand” they are: Index finger in “forward” direction
- Unity: Y-up, Z-forward, left-handed engine
- Unreal: Z-up, X-forward, left-handed engine
- Focus on 2D movements for this class
  - “Human-like” movement
    - Looking at the world from the “top-down” view
  - Unity: Z (forward) and X (right)
  - Unreal: X (forward) and Y (right)





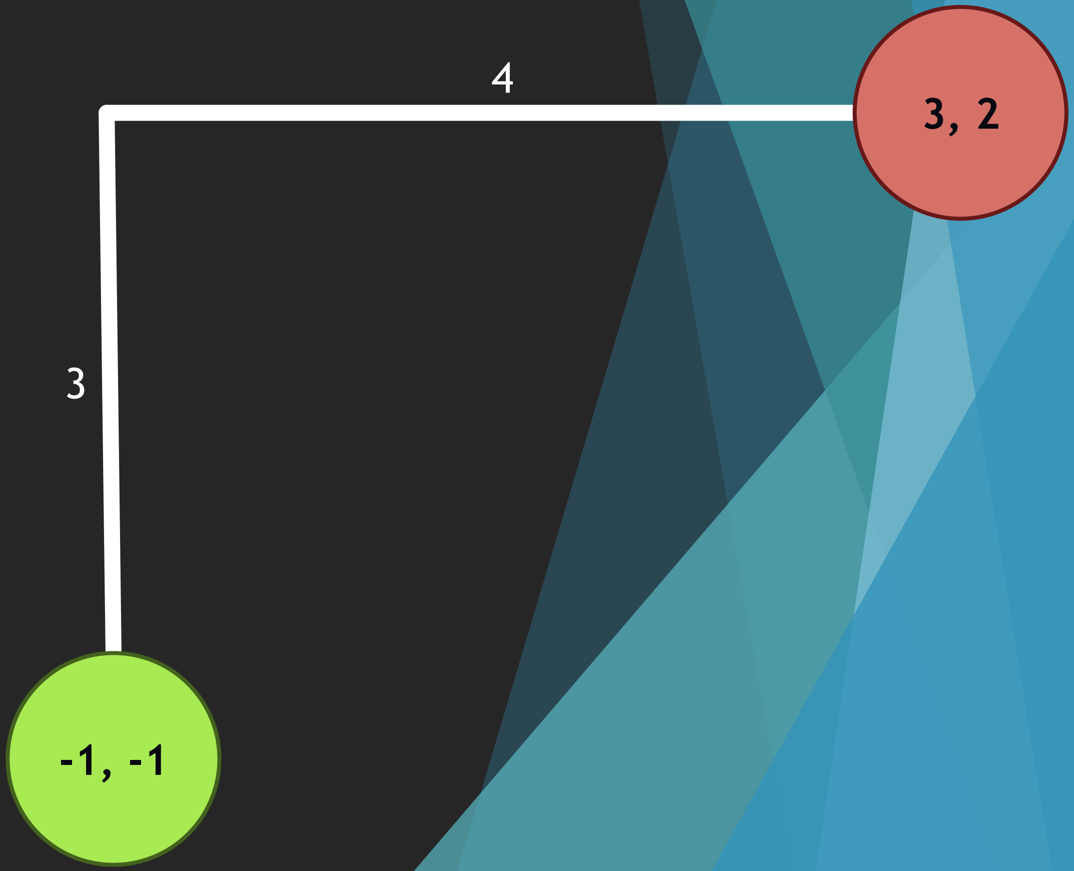
Basic Movements

- Let’s say our agent is at the position (-1, -1), and wants to move to the target at (3, 2)
- How can we perform this move?



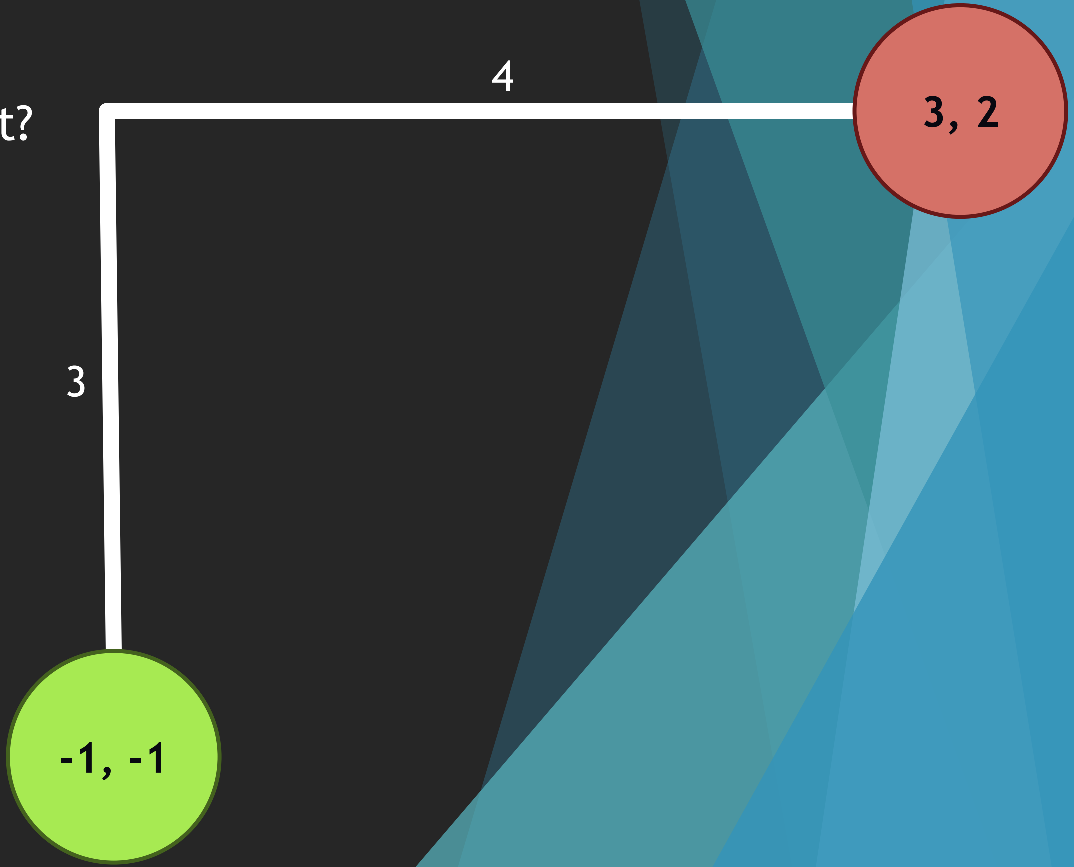
Basic Movements

- Take the difference between the two positions and that is your movement!
  - Known as the direction
  - $target - position = \ (3,2) - (-1,-1) = (4,3)$
- Apply the movement
  - $(-1,-1) + (4,3) = (3,2)$



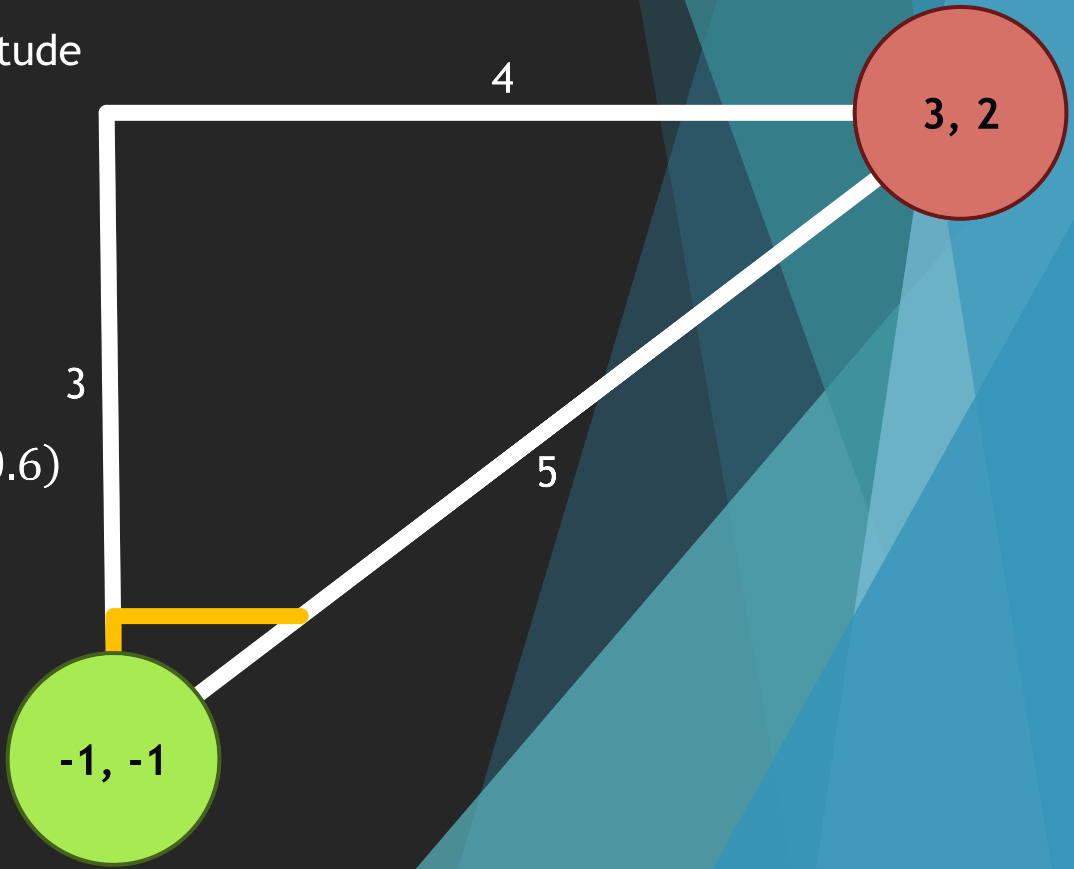
Basic Movements

- In a game we want to get there smoothly over time
  - Let’s say this movement is able to move at 2 units per second
    - Let’s calculate this movement allowed for one second
- How can we only move a partial amount of this movement?



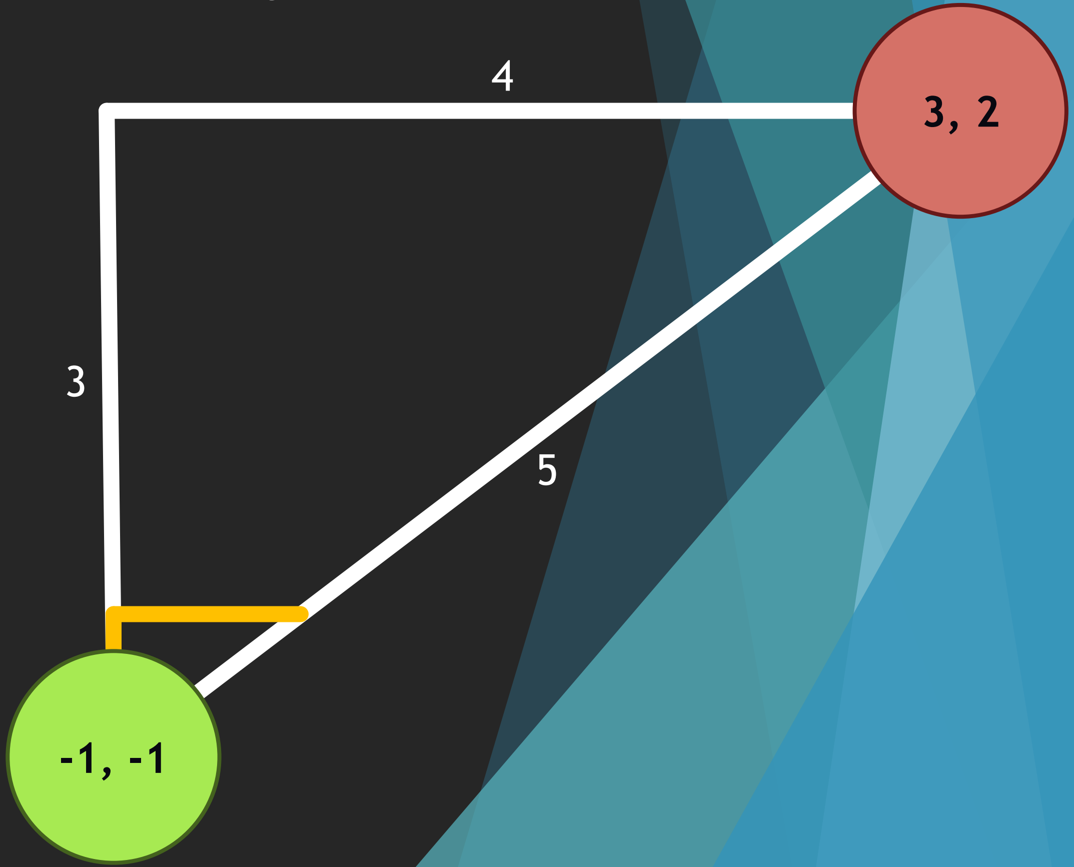
Basic Movements

- We can normalize the vector!
  - A normalized vector has a length of one
  - To get to this, divide each component by the vector's magnitude
- What is the magnitude?
  - Make a triangle and get the hypotenuse!
  - Then, take the square root
- $magnitude = \sqrt{x^2 + z^2} = \sqrt{4^2 + 3^2} = \sqrt{16 + 9} = \sqrt{25} = 5$
- $normalized = direction \div magnitude = (4{,}3) \div 5 = (0.8, 0.6)$



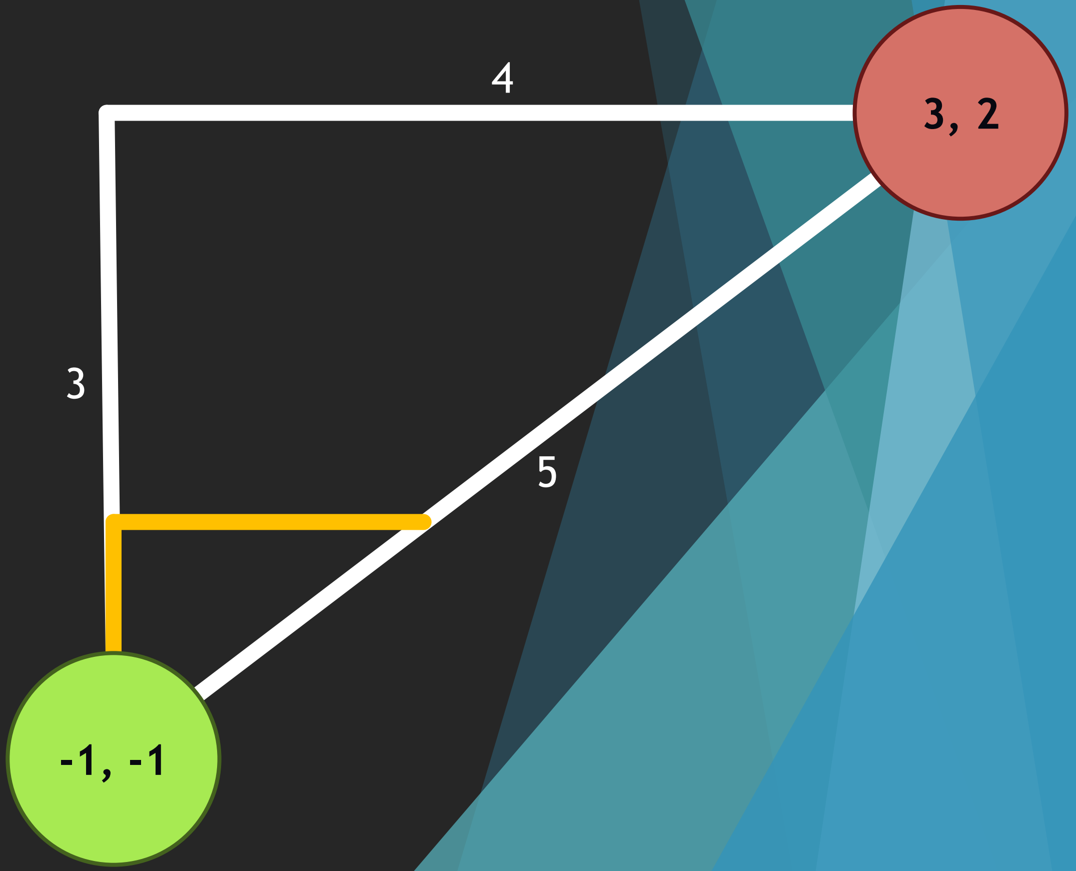
Basic Movements

- We can draw our normalized vector of (0.8, 0.6)
- Now, we know if we wanted to move exactly one unit towards the target, we can move by 0.6 in one direction and 0.8 in the other!
  - $(-1, -1) + (0.8, 0.6) = (-0.2, -0.4)$
- But we want to move at 2 units per second?



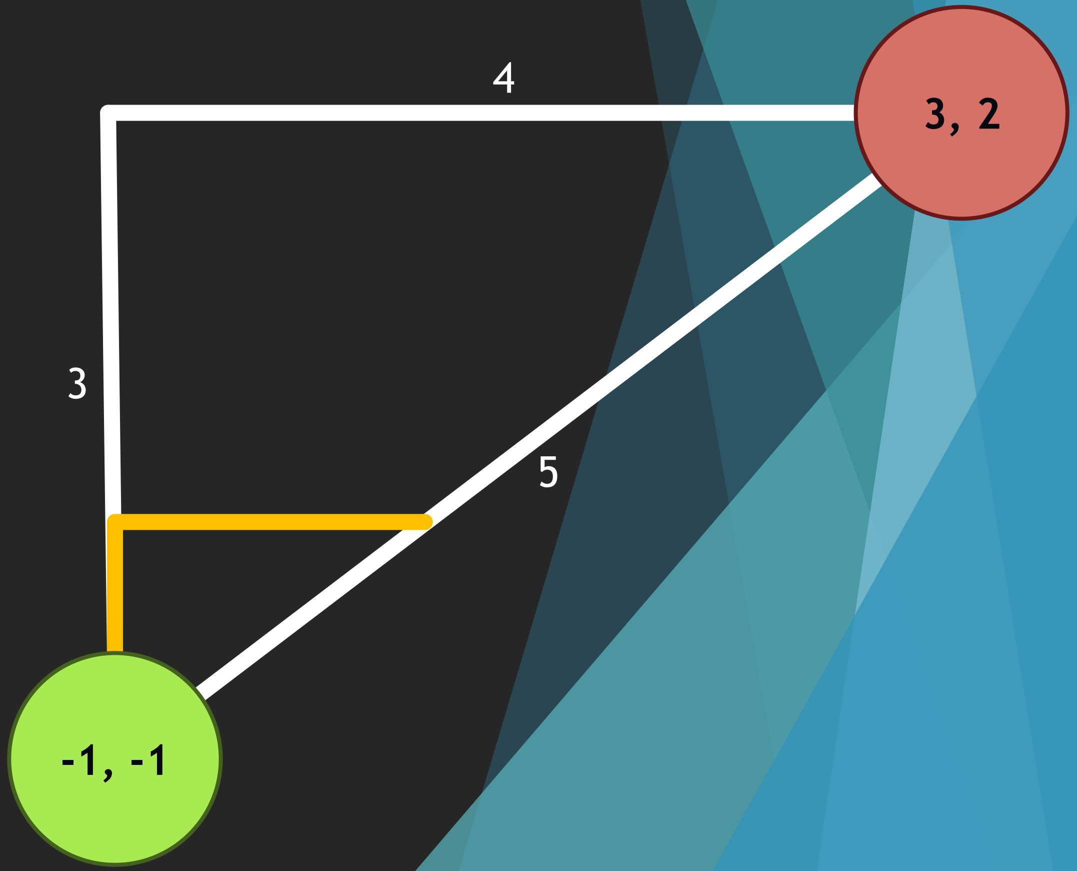
Basic Movements

- Multiply by the speed!
  - $normalized \times speed = (0.8, 0.6) \times 2 = (1.6, 1.2)$
- Hence, we move by (1.2, 1.6) and we are done!
  - $(-1, -1) + (1.6, 1.2) = (0.6, 0.2)$
- This makes the movement for the entire second
  - Games update at 60+ FPS
  - How can we move over this time?



Basic Movements

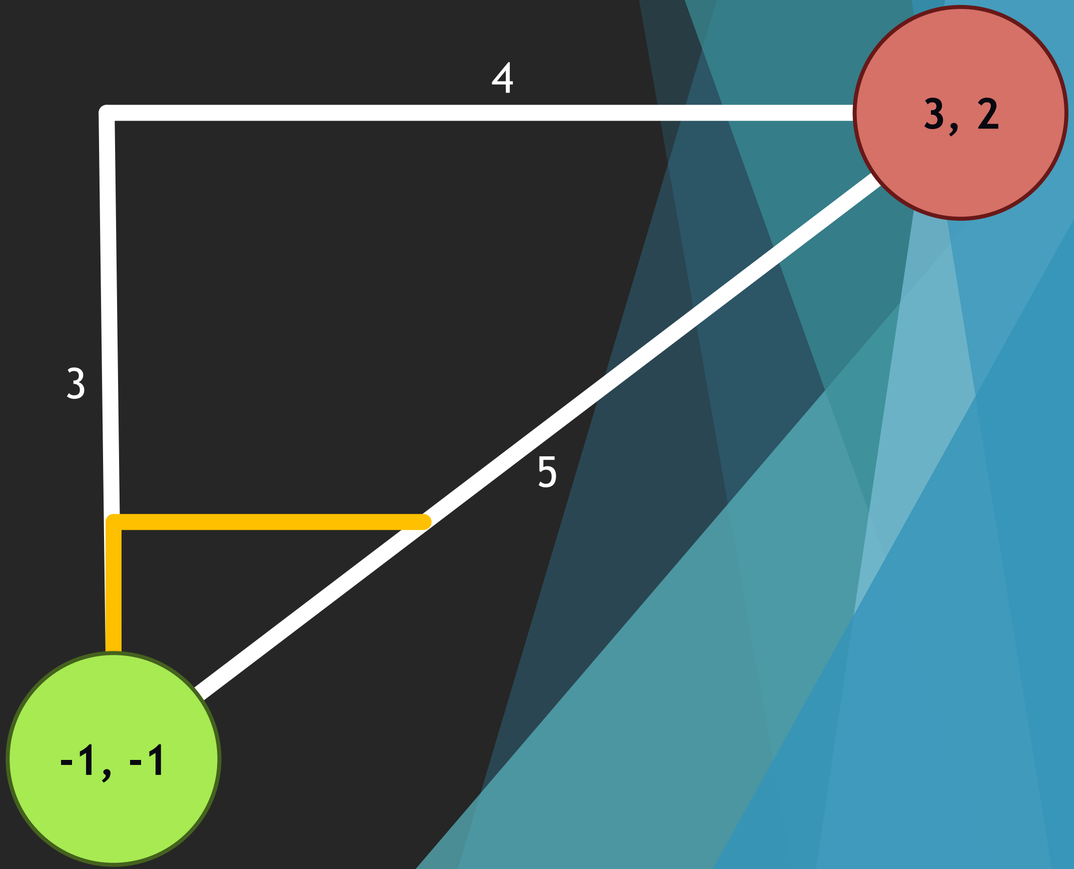
- 60 FPS = Divide by 60 every frame!
  - Problems with this?
- Framerate is variable!
- Is there a better solution?
  - Multiply by the time elapsed since the last frame!
  - All engines have an easy way to access this!
    - Unity: Time.deltaTime
    - Unreal: GetWorld()->GetDeltaSeconds()



Basic Movements

- Recap everything up to this point

1. Subtract our position from the target
2. Get the normalized vector
3. Multiply by speed



Basic Movements

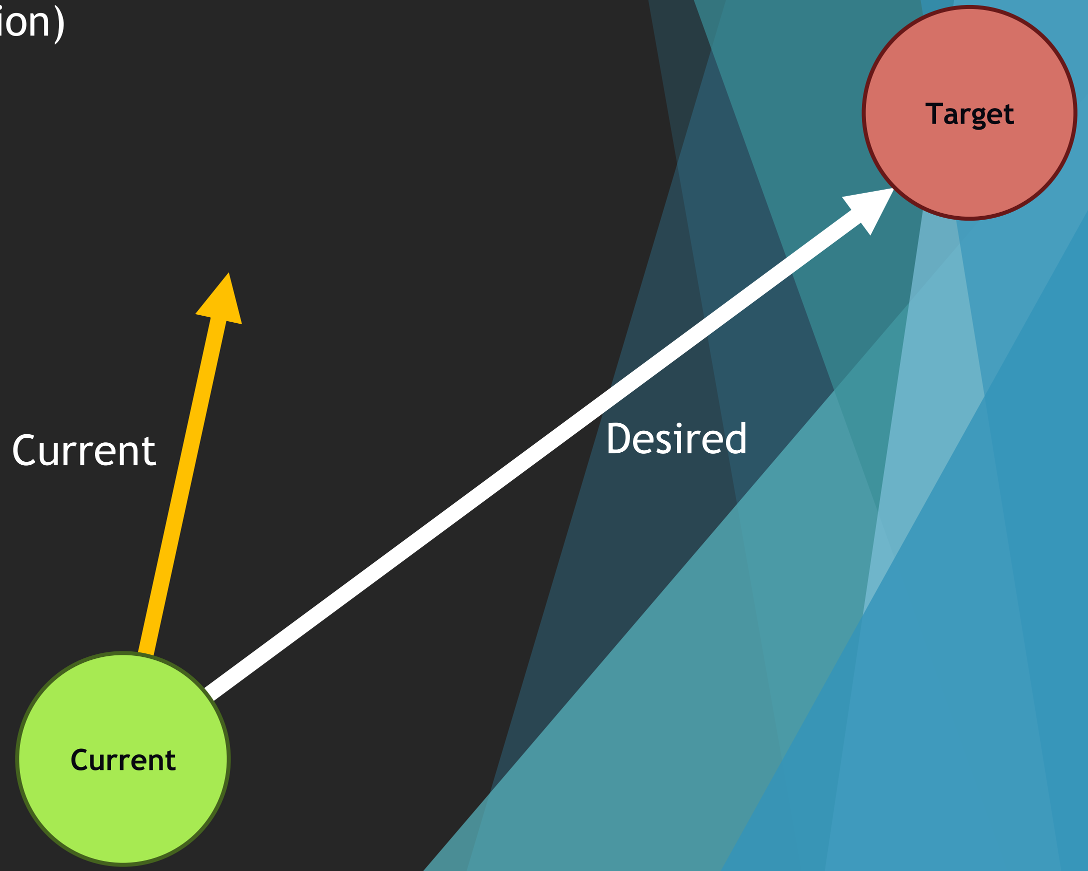
- (target – position).normalized * speed
  - Usually apply delta time smoothing at the end
    - If multiple functions
- Now imagine we wanted to move a character controller towards a target transform
- This would result in instant change in direction towards the target by the desired amount which is great, but what if we wanted to add momentum to our characters
  - Gradual change over time
    - Would need to “slow down” first if already moving in the other direction
  - How could we add that?

Basic Movements

- Simply subtract our current velocity!
- **(target - position).normalized * speed - velocity**
  - Like Time.deltaTime, this is often done at the end outside of the seek behaviour itself depending on your implementation
- This is what is known as the “Seek” steering behaviour!

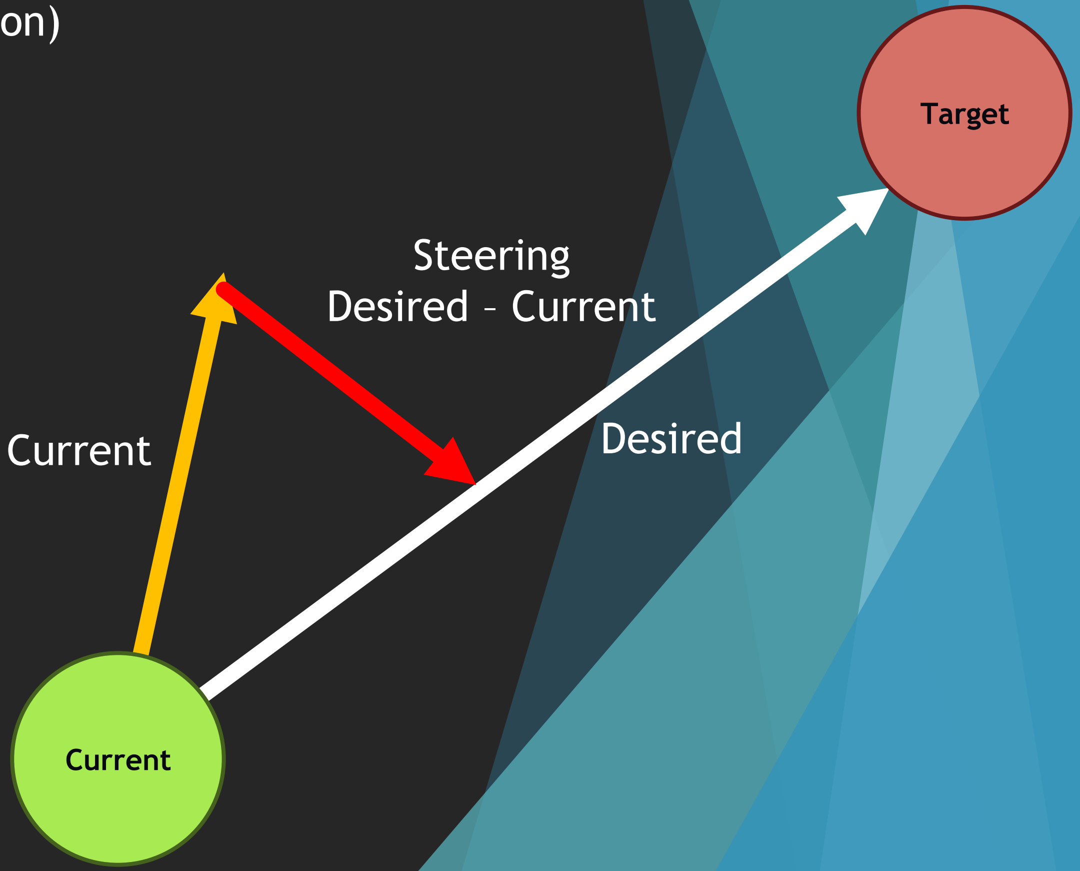
Steering Behaviours

- "Steer" towards where we want to go
  - Not an instant movement (unless we have instant acceleration)



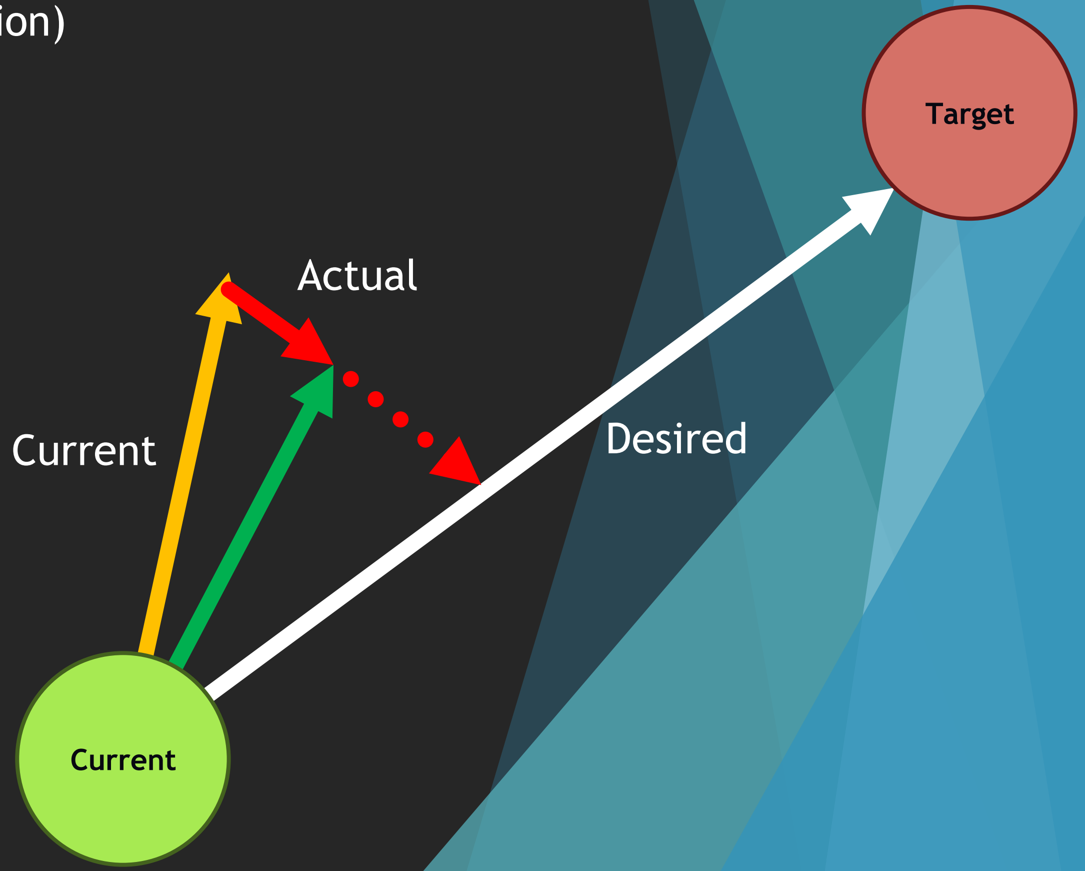
Steering Behaviours

- “Steer” towards where we want to go
  - Not an instant movement (unless we have instant acceleration)



Steering Behaviours

- “Steer” towards where we want to go
  - Not an instant movement (unless we have instant acceleration)



Steering Behaviours

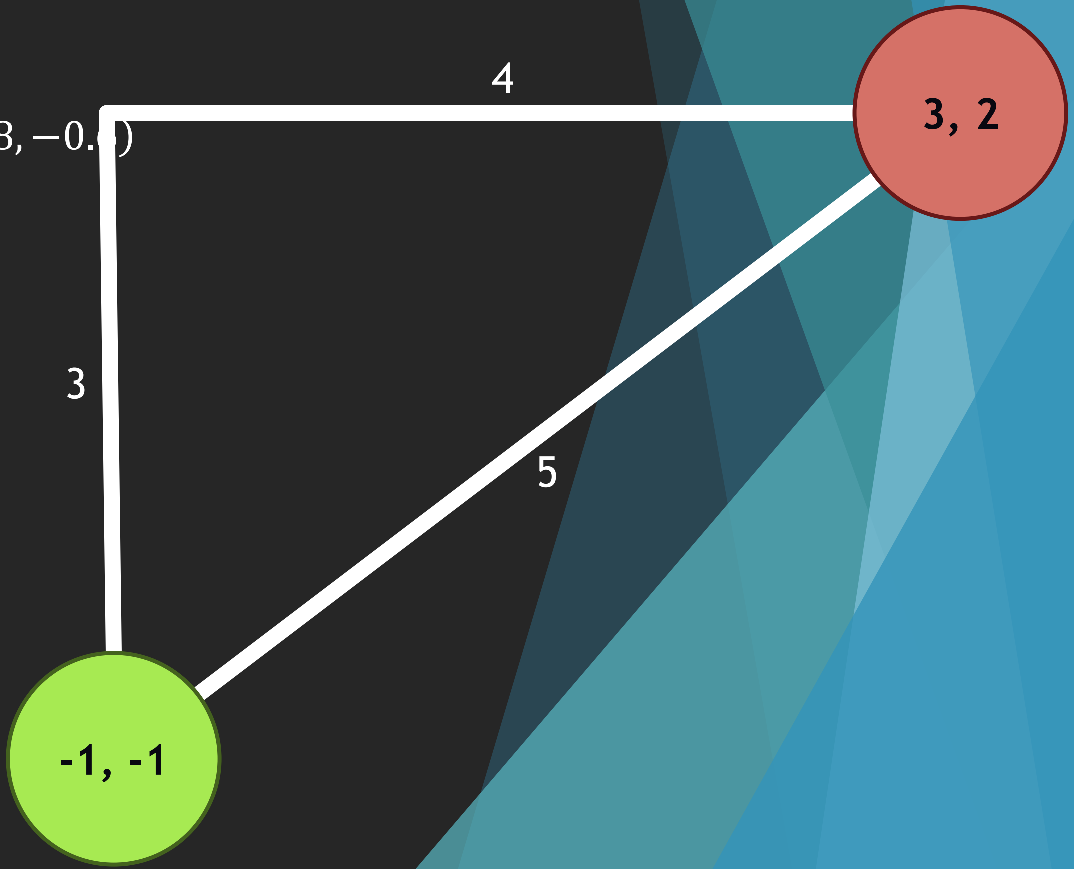
- Seek is the most basic of all movements in games
- How can we do other behaviours?
  - Let’s start simple: What would be the opposite of seeking to a target?
    - Running away from the target!

Steering Behaviours

- Simply reverse the subtraction at the start of seek!
- Seek: (**target - position**).normalized * speed – velocity
- Flee: (**position - target**).normalized * speed – velocity

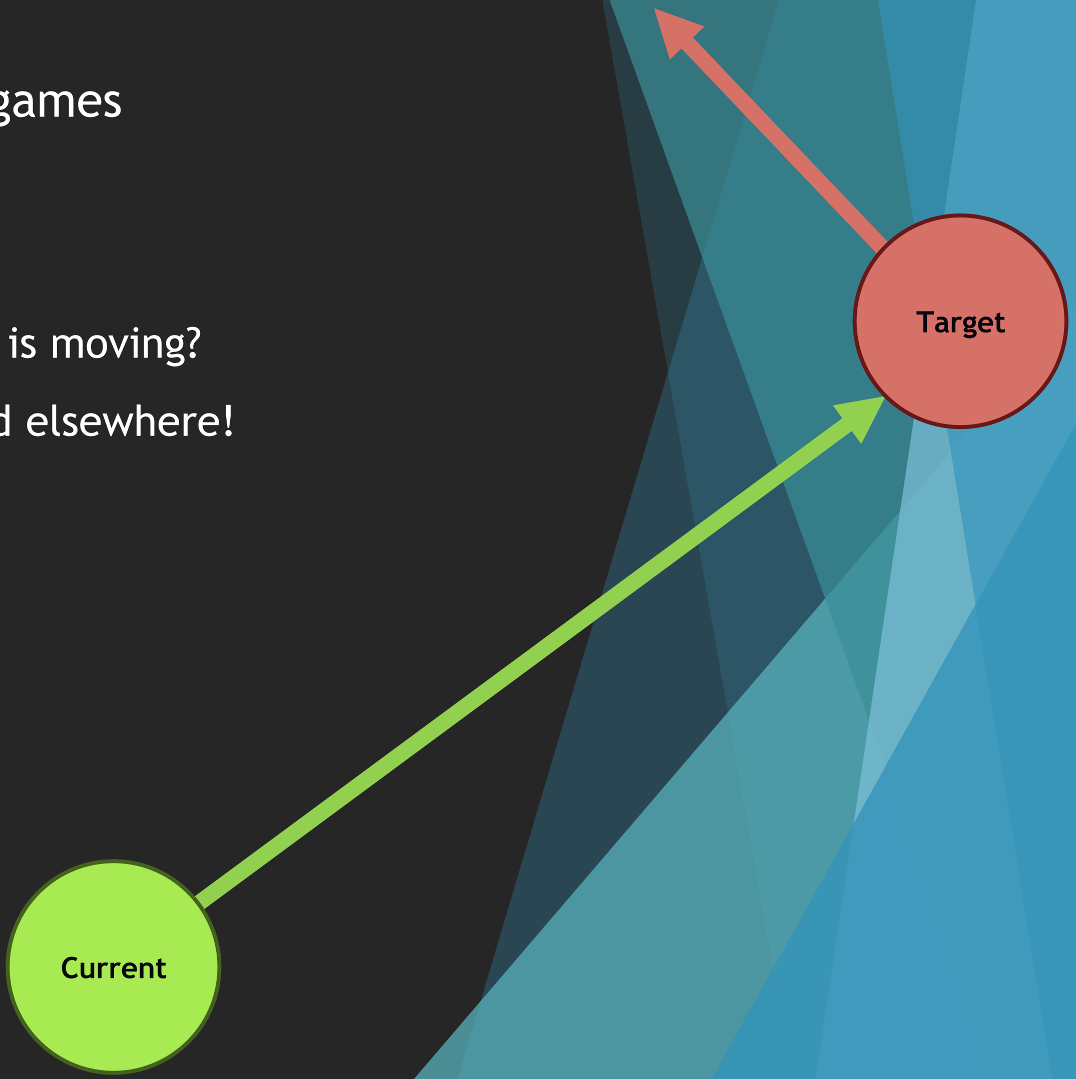
Steering Behaviours

- Flee: (position - target).normalized * speed – velocity
  - $direction = position - target = (-1,-1) - (3,2) = (-4,-3)$
  - $magnitude = \sqrt{x^2 + z^2} = \sqrt{-4^2 + -3^2} = \sqrt{16+9} = \sqrt{25} = 5$
  - $normalized = direction \div magnitude = (-4,-3) \div 5 = (-0.8,-0.6)$
  - $normalized * speed = (-0.8,-0.6) \times 2 = (-1.6,-1.2)$
  - Subtract the existing velocity of the agent



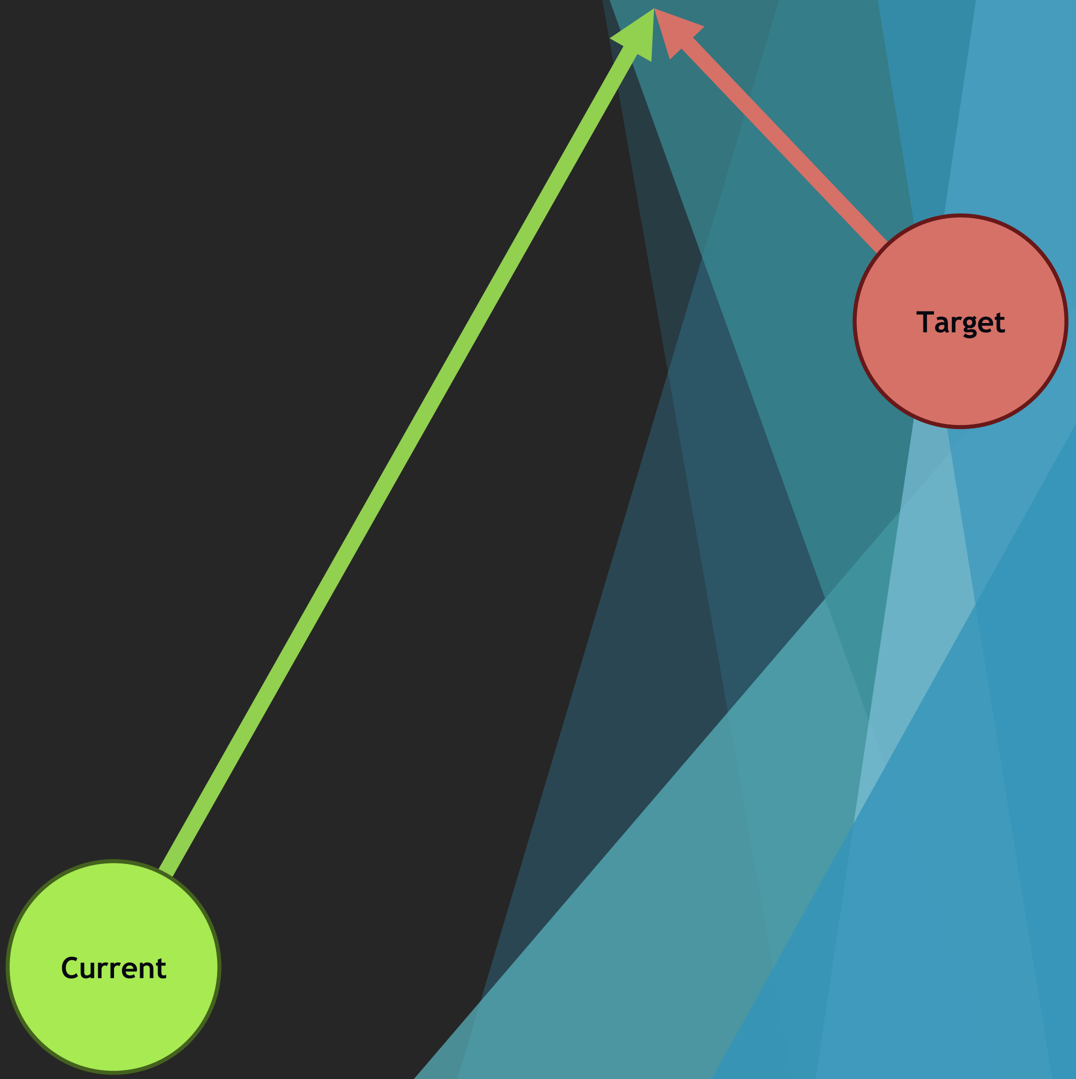
Steering Behaviours

- We now have the two most basic movements needed for games
  - What could be a limitation of these movements?
- Not “smart”!
  - What if we are trying to track down a target but that target is moving?
  - By the time we get to our “seeked” target, they have moved elsewhere!
    - What should we do?



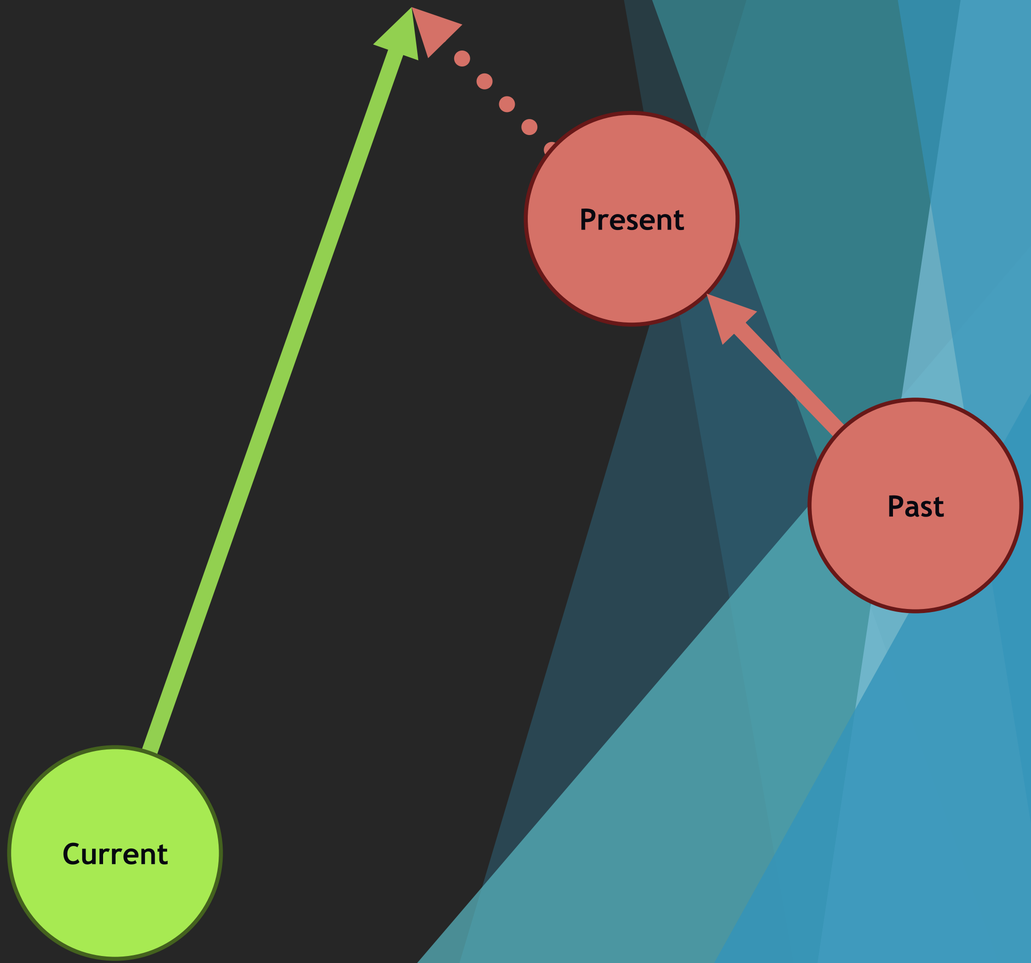
Steering Behaviours

- Intercept where the target is going!
- How can we know this?



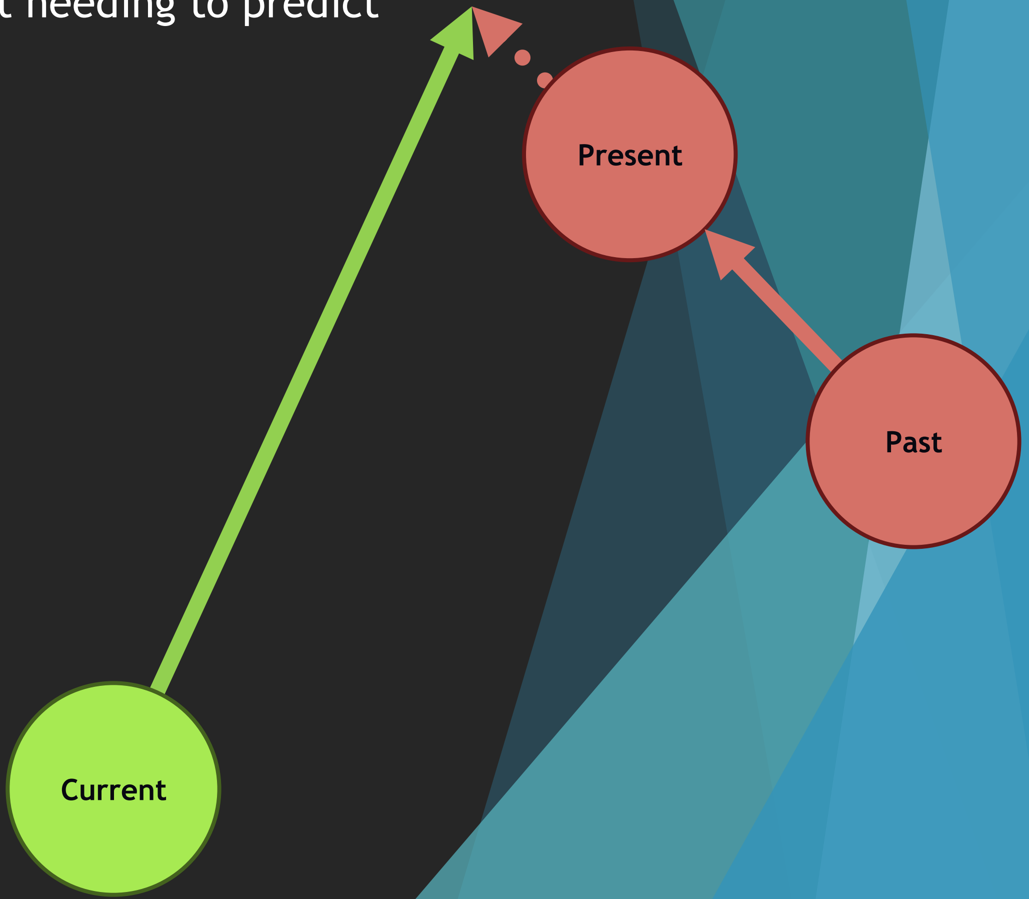
Steering Behaviours

- Use the past observed velocity!
  - $Present = (2, 2)$
  - $Previous = (4, 0)$
  - $Change = Present - Previous = (2, 2) - (4, 0) = (-2, 2)$
  - $Predicted = Present + Change = (2, 2) + (-2, 2) = (0, 4)$
- Now how do we move towards that predicted position?
  - Simply use seek on the predicted future position!*
    - *We need to account for our own speed first



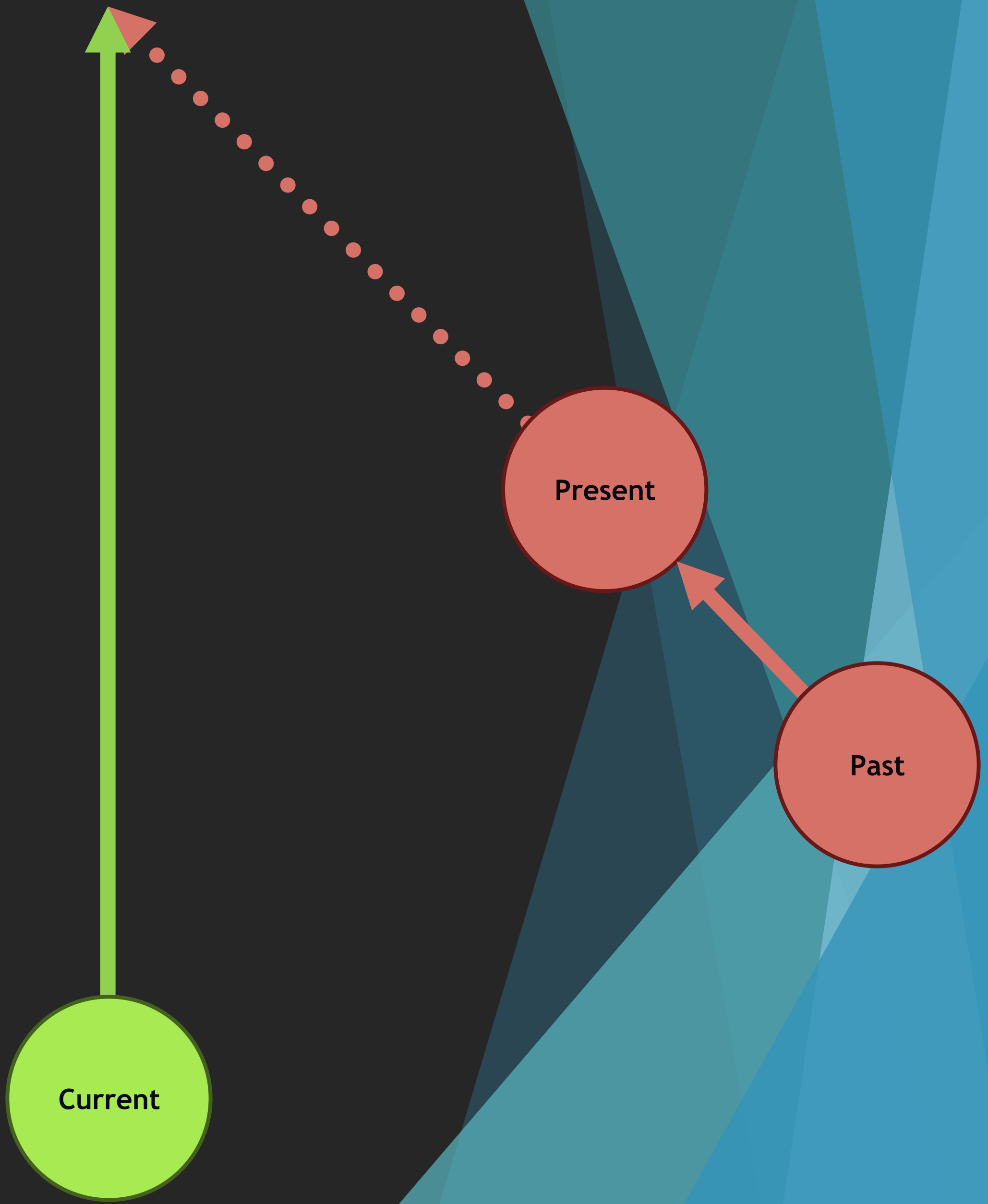
Steering Behaviours

- If we are fast enough, we could “catch up” super quick
  - If not immediately reach them where they currently are, not needing to predict



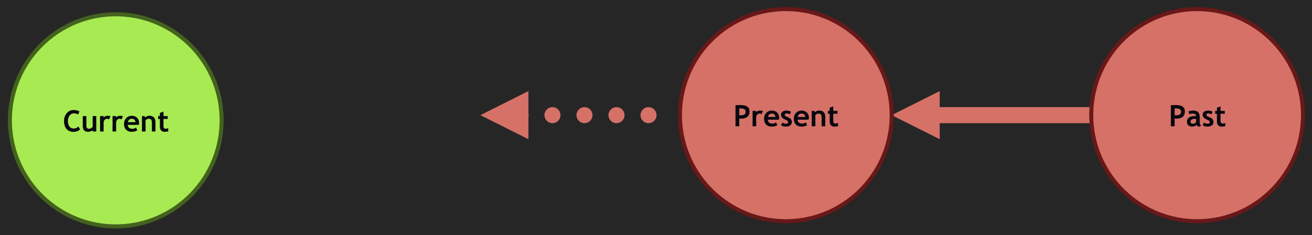
Steering Behaviours

- If we are slower, we need a longer intercept
- How can we calculate this?
  - Multiply the predicted target by the distance over speed!
- $future = present + predicted \ * (distance \div speed)$
- Then, seek to this position!
- Is there a situation this might have problems?



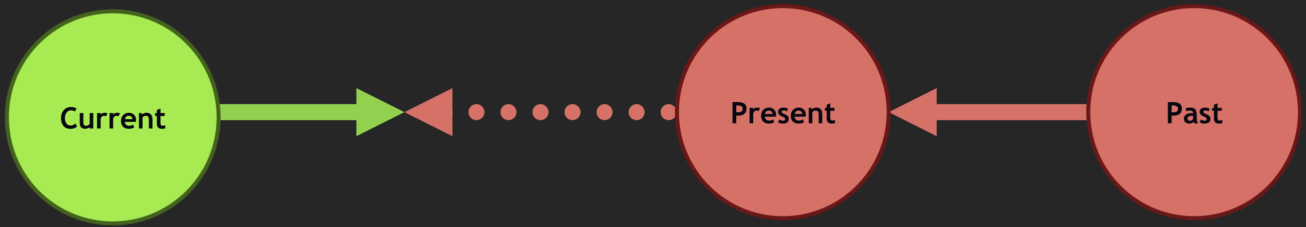
Steering Behaviours

- How would this work?



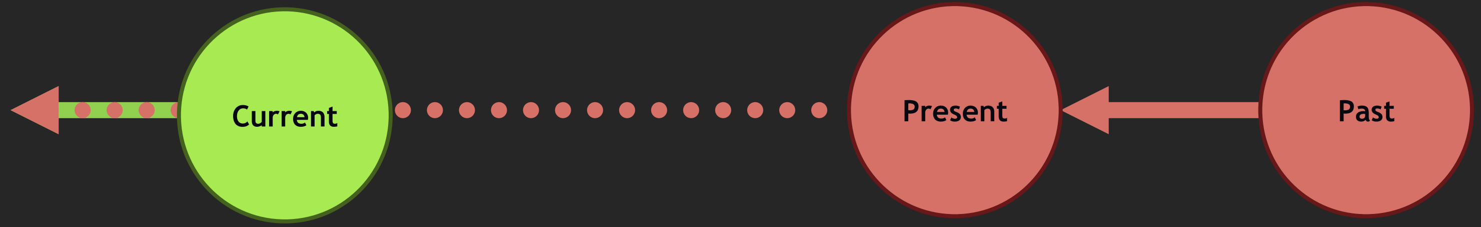
Steering Behaviours

- How would this work?
- If we are faster, then this works fine
  - Faster speed = smaller lookahead distance



Steering Behaviours

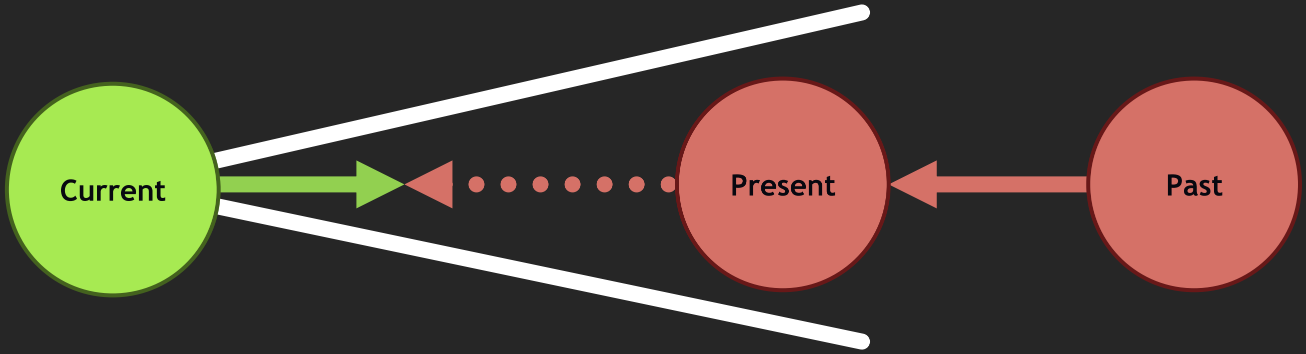
- How would this work?
- If we are faster, then this works fine
  - Faster speed = smaller lookahead distance
- If we are slower, the scaled distance ends up being larger
  - Could end up behind us!
  - We now are moving away from the target to “pursue” it!
  - How can we fix this?



Steering Behaviours

- Two approaches

1. Calculate if moving towards each other, and simply seek instead
    - Give a certain angle for the seek fallback behaviour
    - Use dot products for quick math
2. Sum both speeds
    - $future = present + predicted * (distance \div (speed + targetSpeed))$
    - More mathematically expensive
    - May now under predict when chasing, but often “feels” better



Steering Behaviours

- How can we make a predictive version of flee?
  - Literally the exact same logic!

1. Do the same calculations as pursue
2. Pass that predicted future target into flee!

Steering Behaviours

- Pursue and evade have shown the first example of “compound” movements
  - Being built upon the “atomic” movements of seek and flee
- Let’s continue to build up these more “advanced” movements
- Next up: following a path
  - How can we do this?

Steering Behaviours

- Simply seek to one point at a time in your list of points to follow!
- When one is reached, remove it and seek to the next!
  - How we get this list of points itself will be covered in the future
    - Path following = Easy
    - Path finding and planning = Hard*
- What is a potential issue with our current approaching algorithms?
  - What if we overshoot a target?

Steering Behaviours

- Two main options:

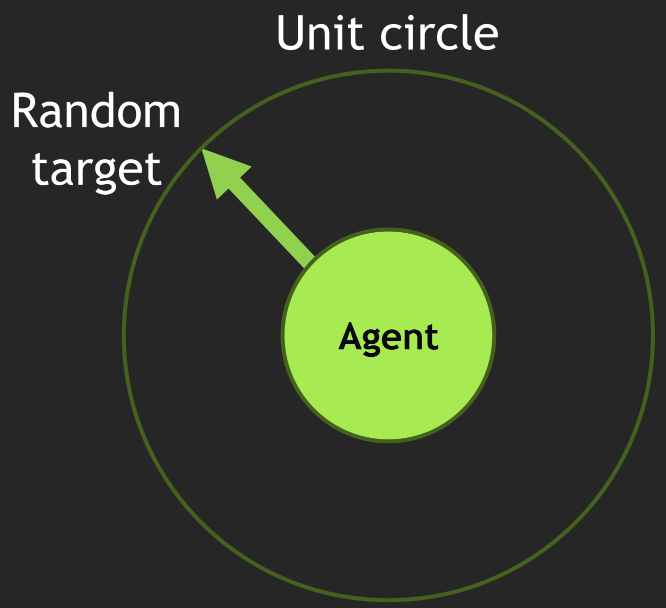
1. The “Arrive” behaviour
    - Slows down or completely stops based on a radius to the target
2. The “close enough” method
    - If we are within a radius to a target position, count this as reaching it
    - Even if you have an arrive behaviour, often a good idea to include this

Steering Behaviours

- Everything so far has assumed something:
    - We have a specific target in mind!
        - Either want to go to it or avoid it
- What if we don’t have a target?
    - We likely don’t want to just sit still!
        - Takes away from the life-likeness of the game
- Randomly choose to a point to seek to on the map
    - Could be behind walls however – may need to path find and then follow
    - This is however on the right track using seek!
        - What could we do with seek that is simpler than a random space on the map?

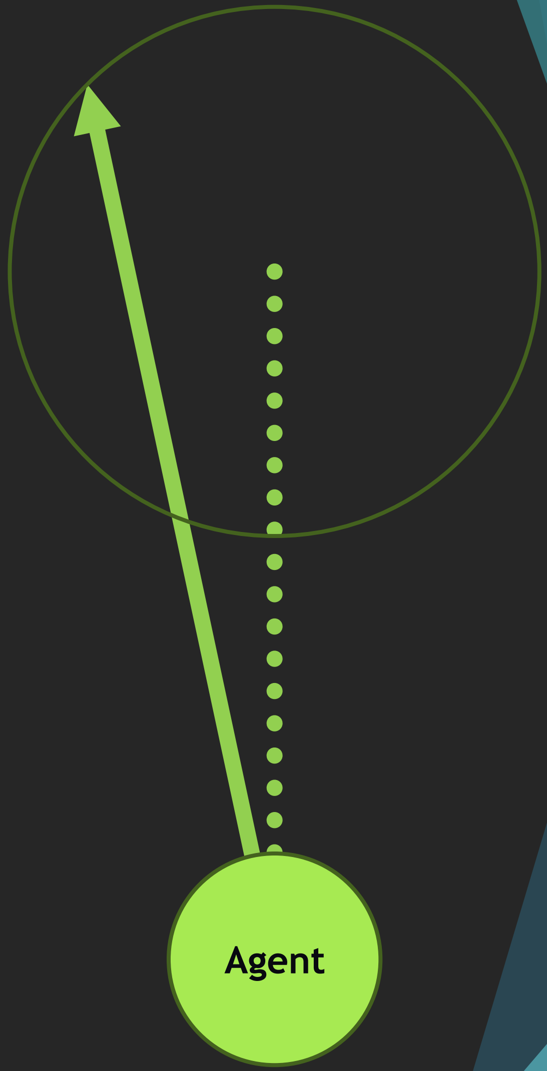
Steering Behaviours

- Move in a random direction of our agent!
- How can we get a direction?
  1. Project a unit circle around the agent
     - Radius of 1 (technically radius is irrelevant here)
  2. Seek to a random point on the circle
- What is an issue with this?
  - This will be completely random!
  - On average, the agent will stay in the same place!
  - What can we do instead?



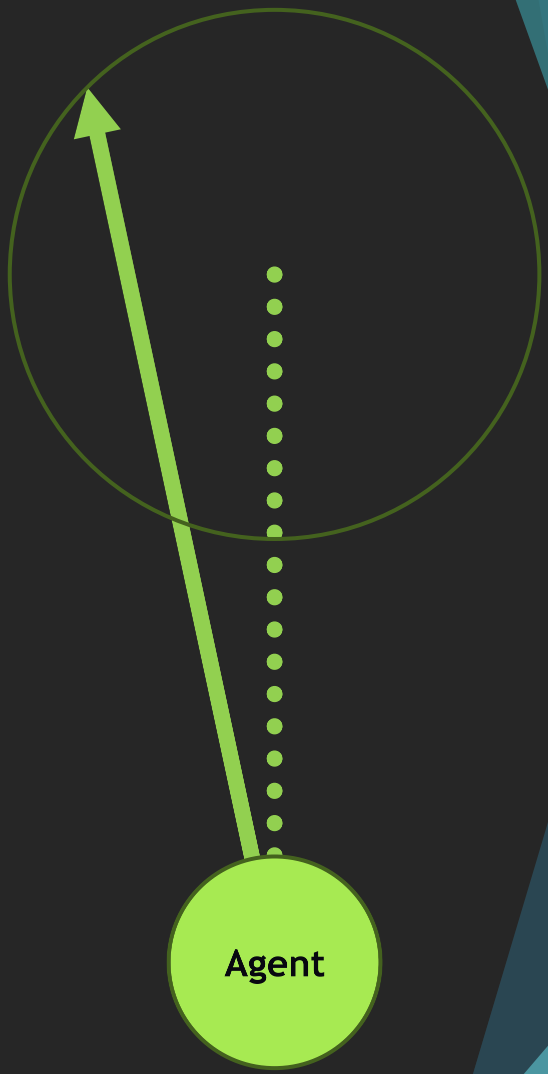
Steering Behaviours

- Project the circle ahead of the agent!
- Will allow for some random drift but in a general direction
- Can now adjust the distance and radius of the circle
  - Adjust to achieve your desired “wander” behaviour



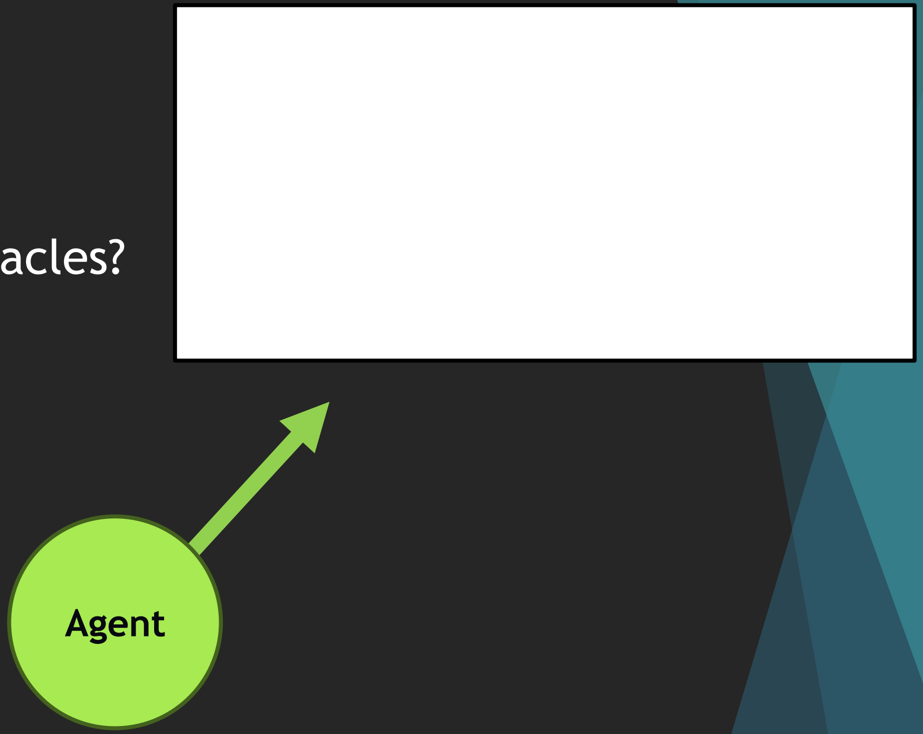
Steering Behaviours

- What is a problem our agents could still face with this?
  - Nothing wrong with the algorithm itself
    - Something external to the wander algorithm



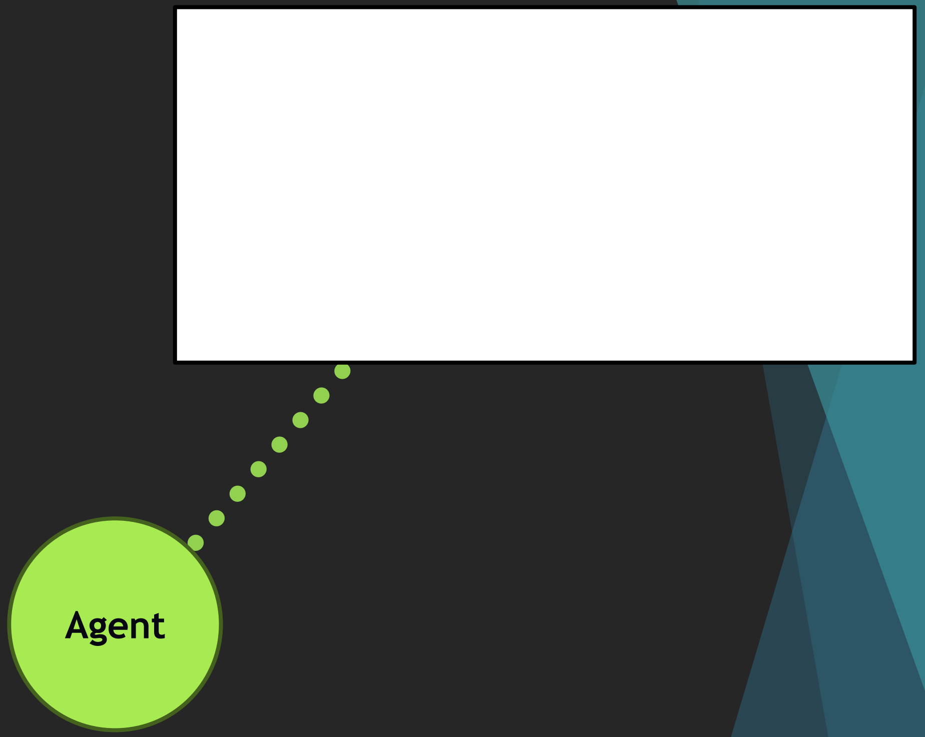
Steering Behaviours

- How does this (or any other movement) handle obstacles?
- How can we avoid obstacles?



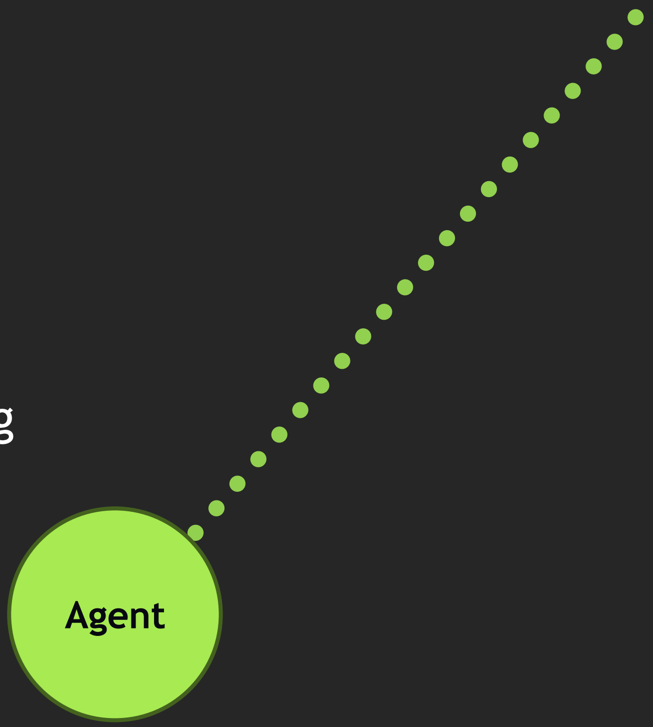
Steering Behaviours

- Cast a ray ahead of us to detect the obstacles!
- We do this for a set distance ahead of us



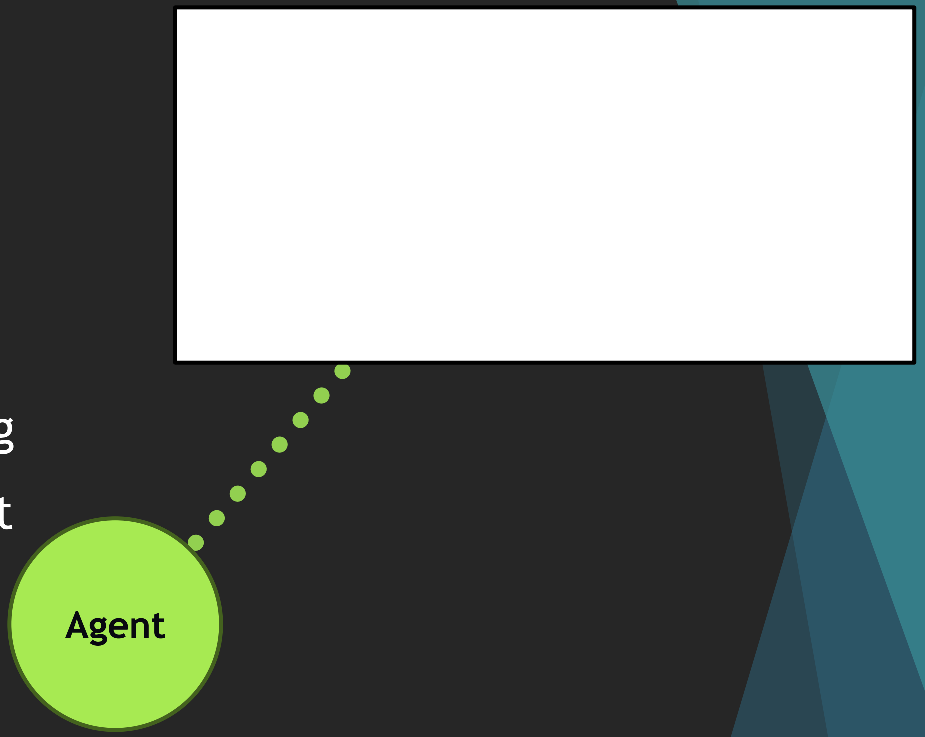
Steering Behaviours

- Cast a ray ahead of us to detect the obstacles!
- We do this for a set distance ahead of us
- If there is no wall, we hit nothing and we do nothing



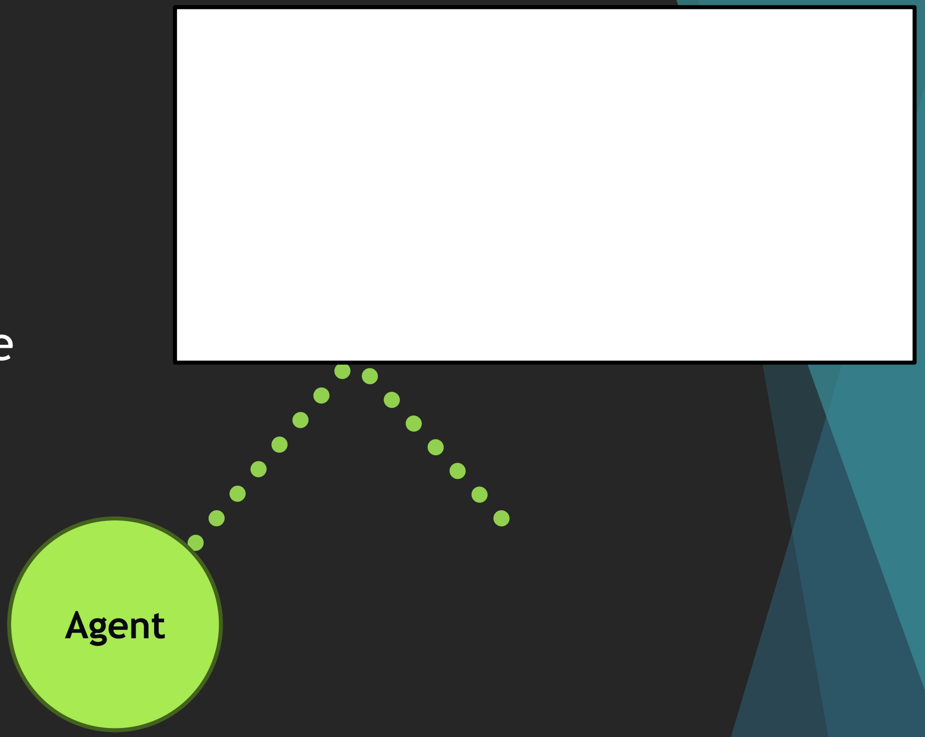
Steering Behaviours

- Cast a ray ahead of us to detect the obstacles!
- We do this for a set distance ahead of us
- If there is no wall, we hit nothing and we do nothing
- Otherwise, we need to change our course to avoid it
  - How should this be done?



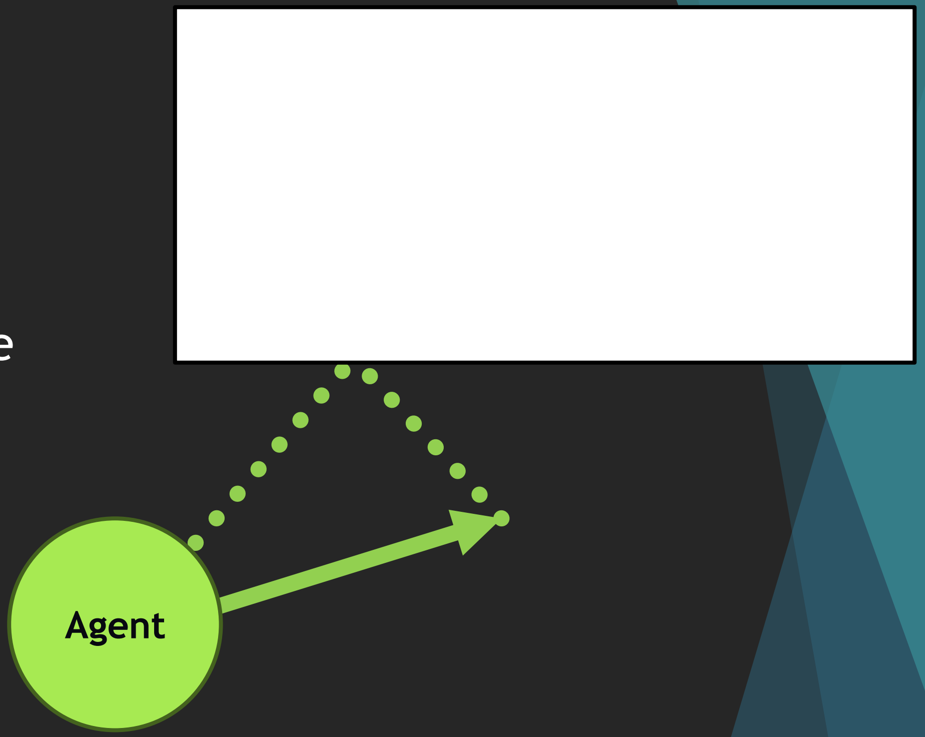
Steering Behaviours

- Take the normal of the hit
- Follow it until we reach a minimum desired distance from the wall



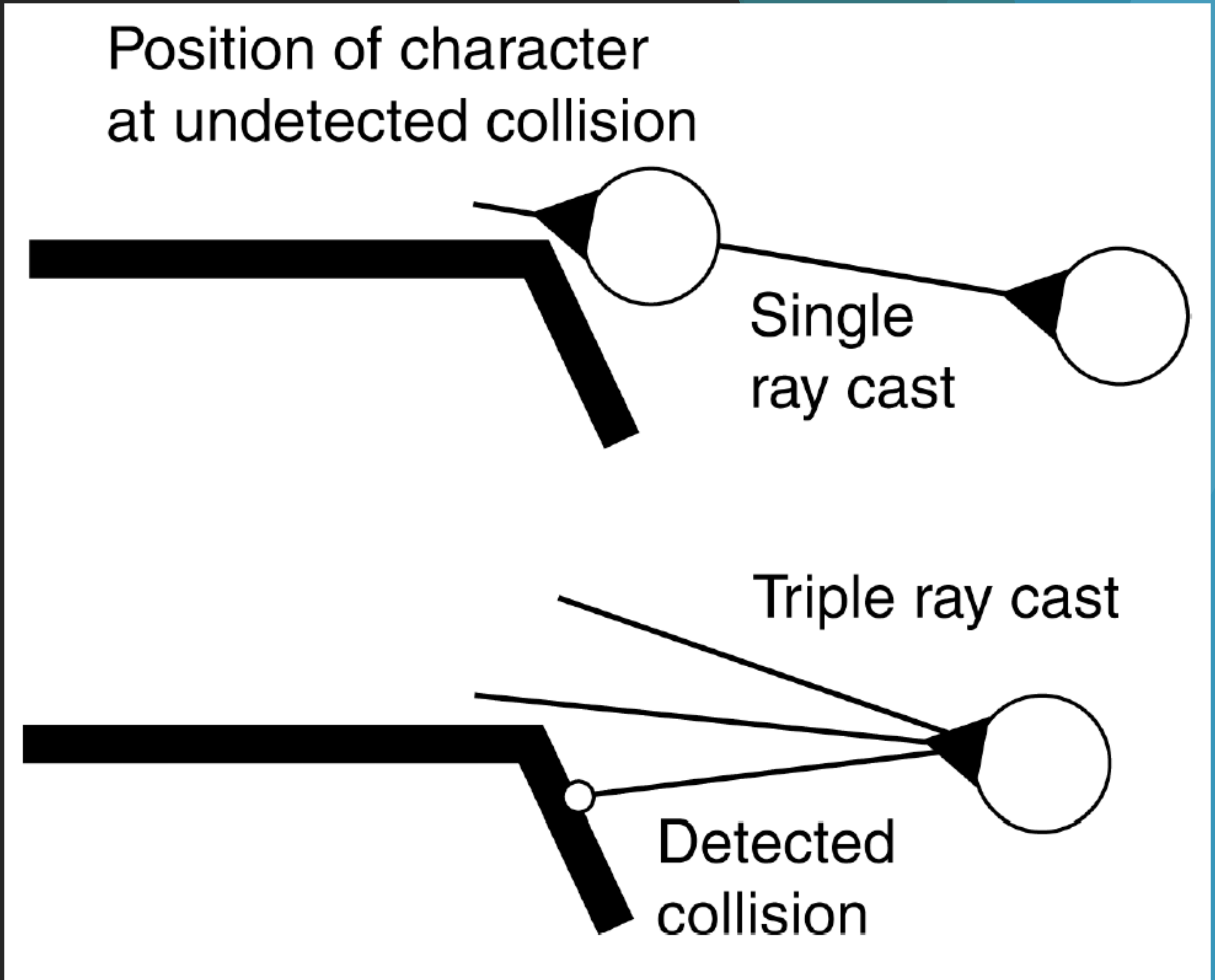
Steering Behaviours

- Take the normal of the hit
- Follow it until we reach a minimum desired distance from the wall
- Seek towards that spot instead



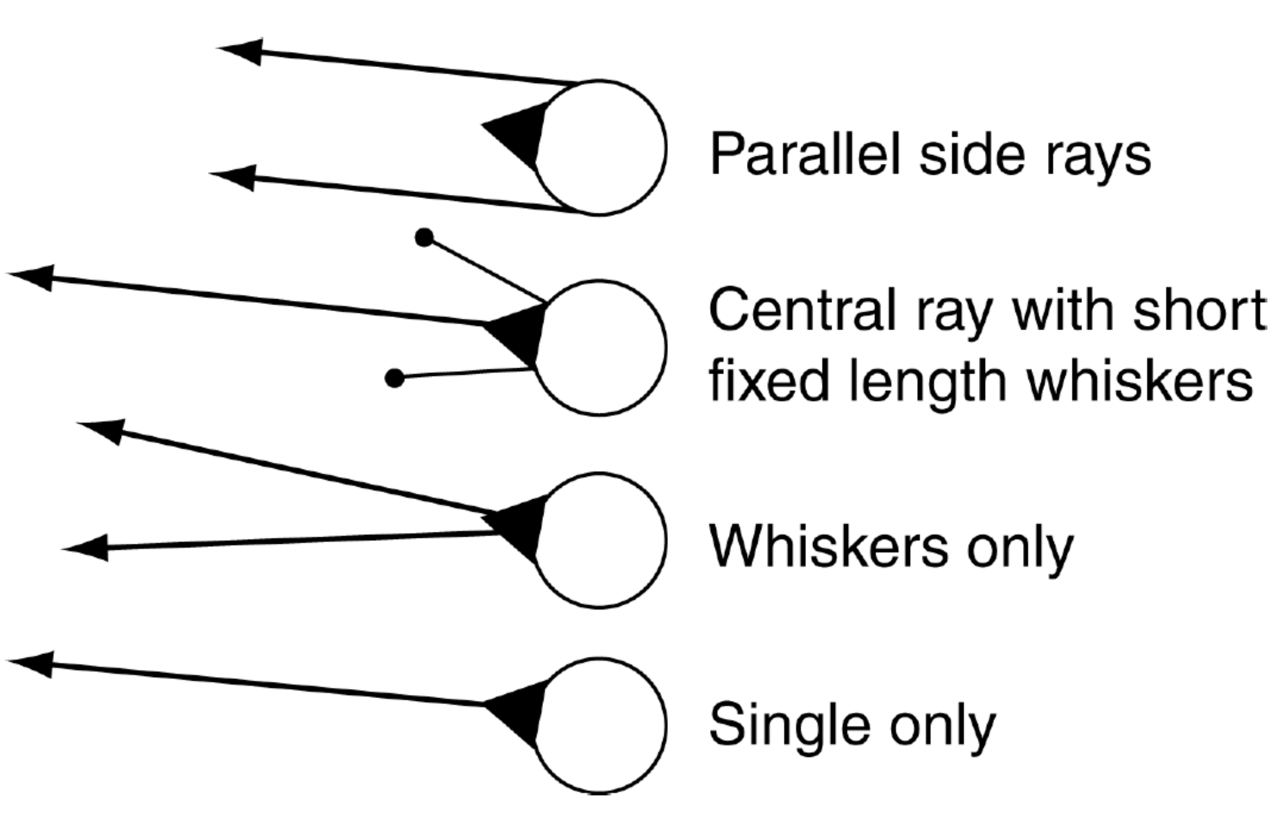
Steering Behaviours

- Obstacle avoidance is not without its limitations
- A single ray may not hit an obstacle!
  - Multiple rays can account for this
  - Side rays or “whiskers”



Steering Behaviours

- Multiple ray types
- No one best option for all situations
- What ray to get the normal of if multiple hit?
  - Usually the shortest ray

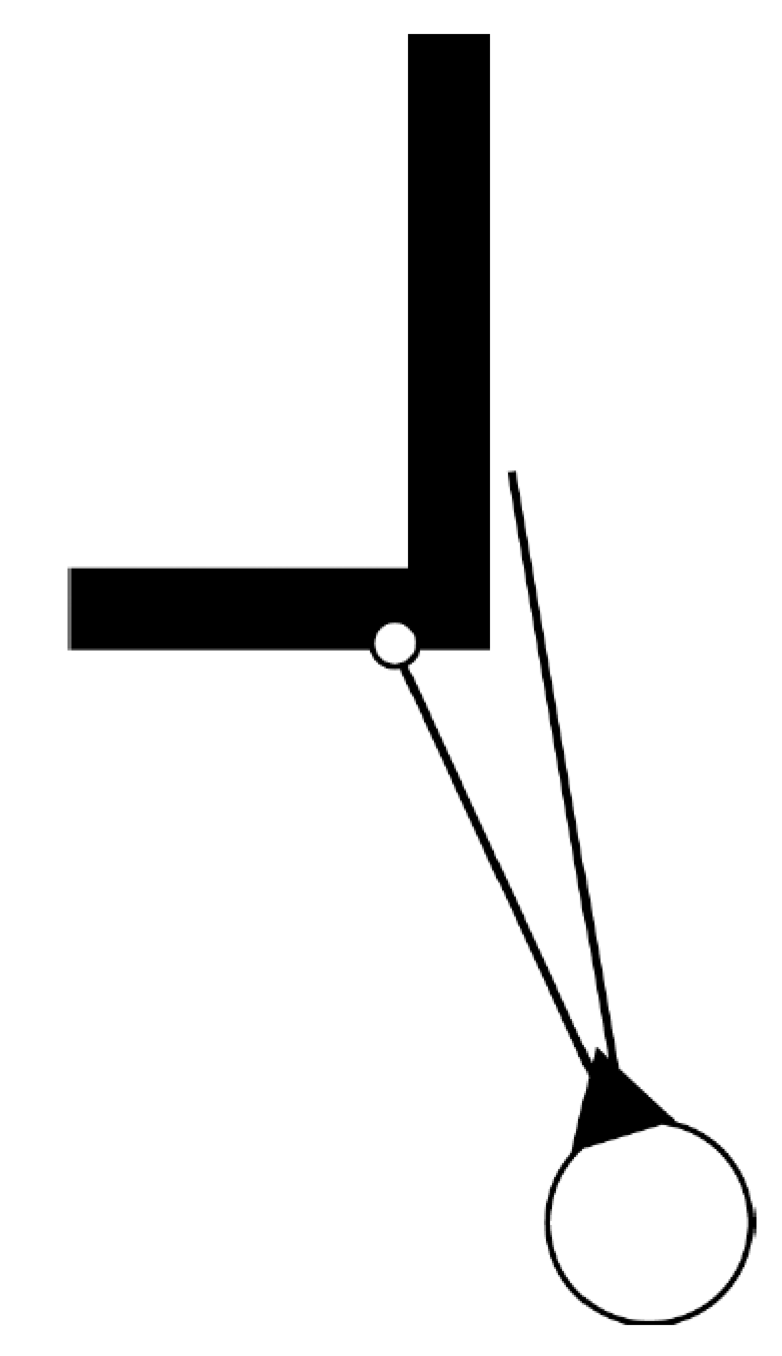


Steering Behaviours

- The corner trap!

1. Left ray hit → Normal will steer us left
2. Right ray hits → Now the normal will steer us right!
3. Left ray hits again → Normal will steer use left!!

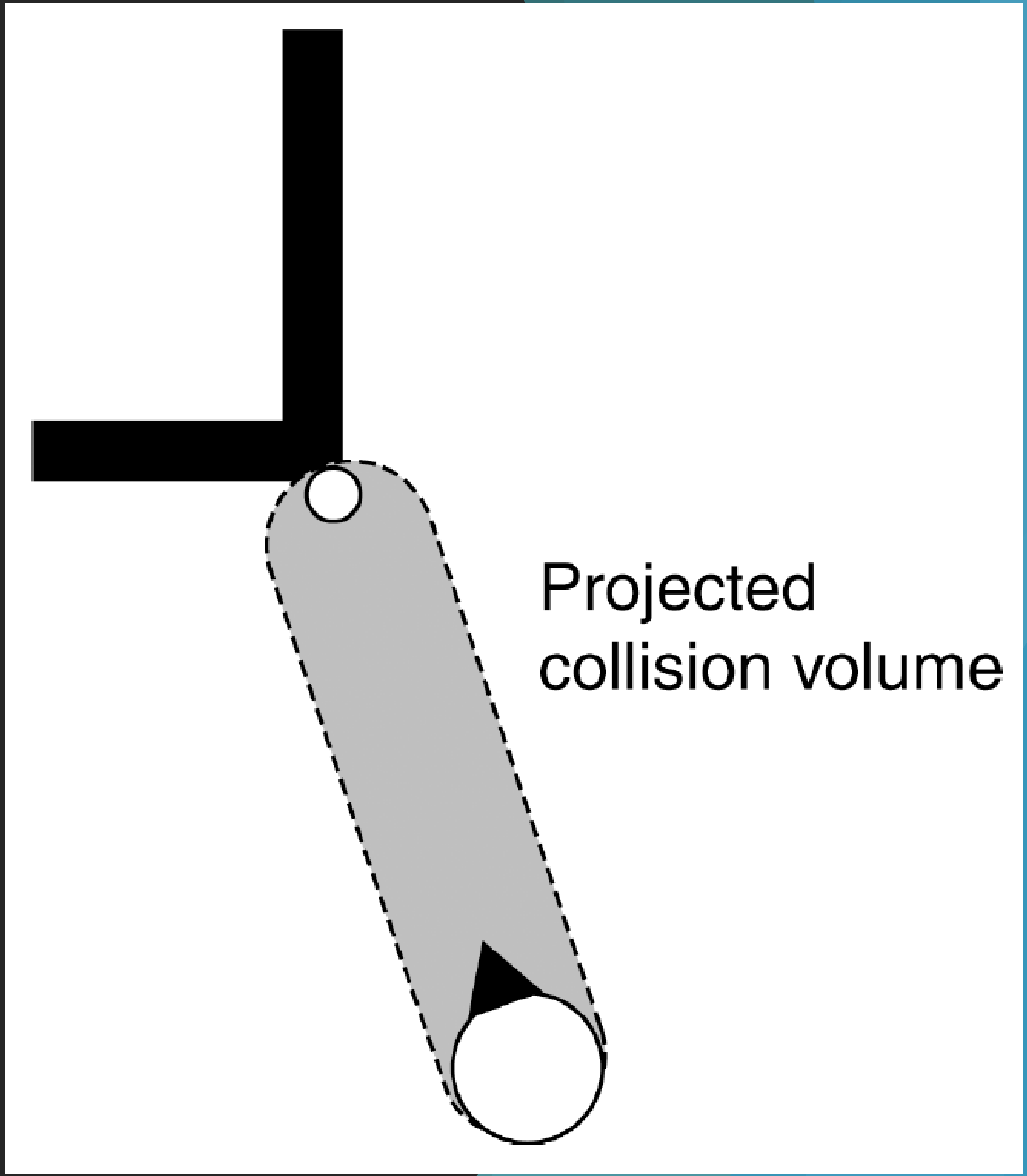
- We are trapped running into the corner!
  - How can we avoid this?



Steering Behaviours

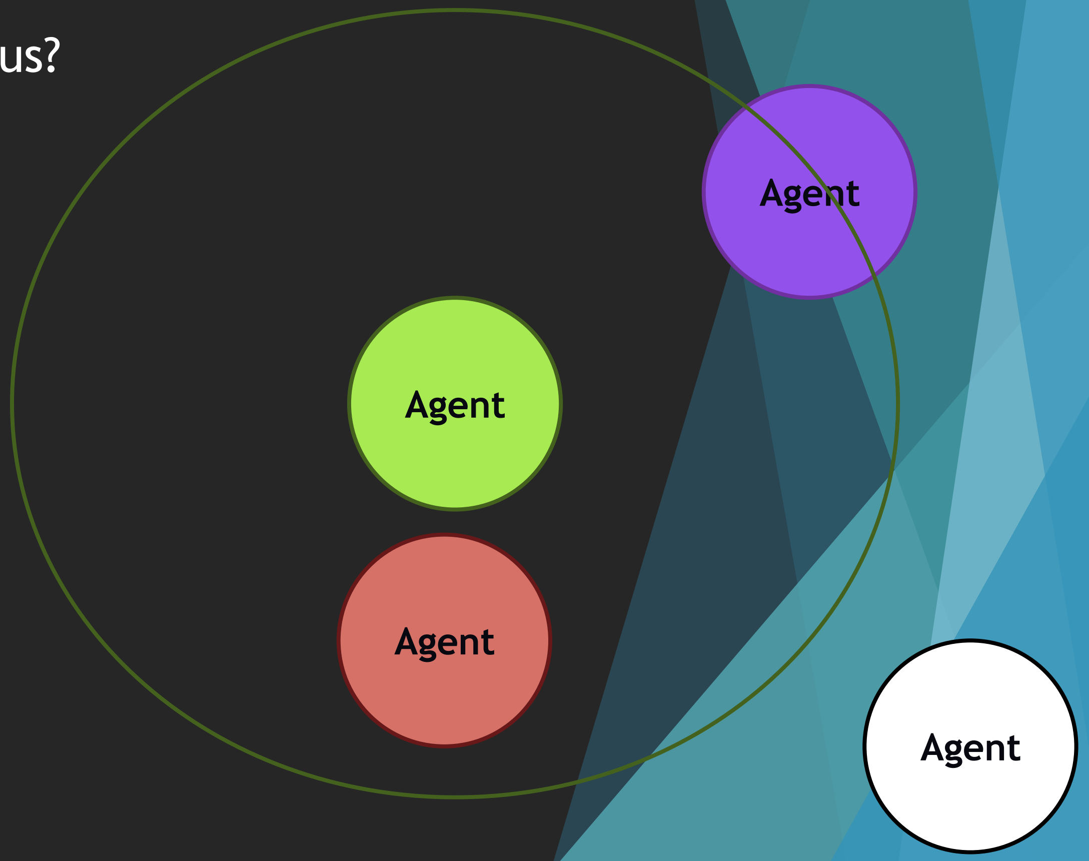
- No bullet-proof way

1. A wide fan or angle
2. Volume-based methods
3. Custom logic
   - Potentially trap rapidly oscillating avoidances
   - Then, perform a drastic trajectory change to escape



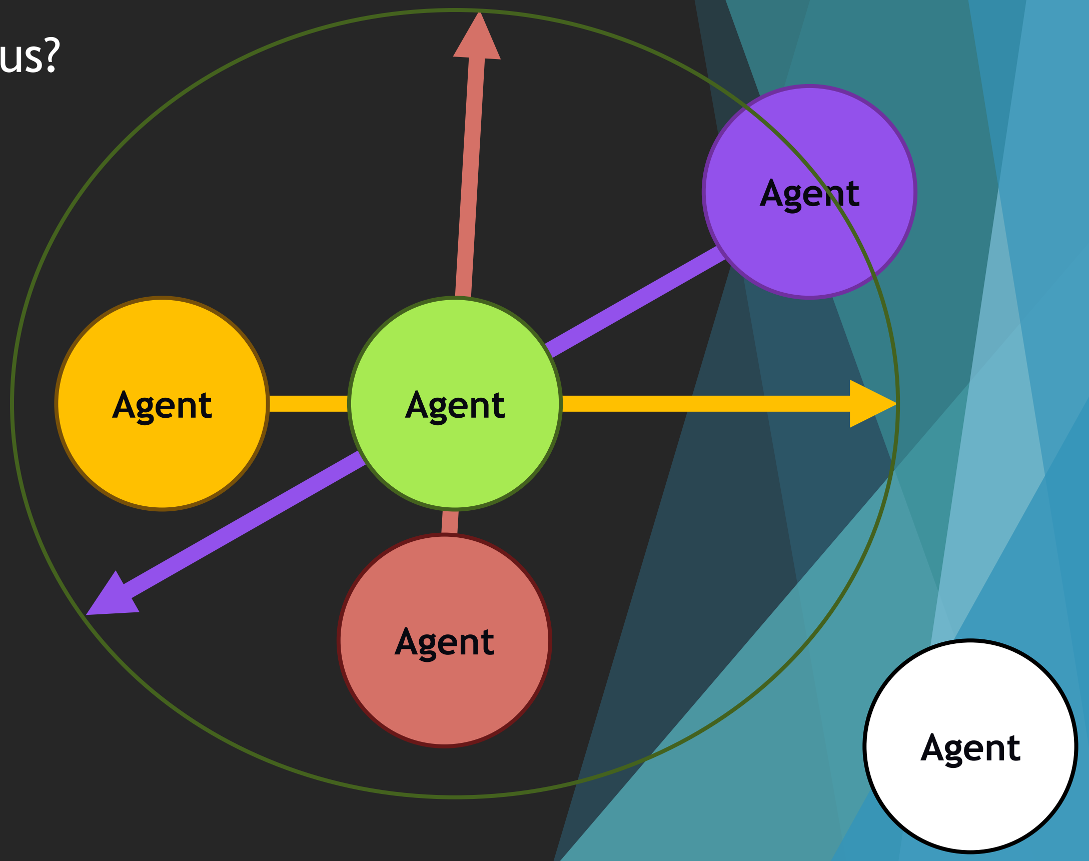
Steering Behaviours

- This was great for obstacles
- But how can we avoid other agents in a radius around us?
  - Helps avoid walking into each other
  - Spread out more?



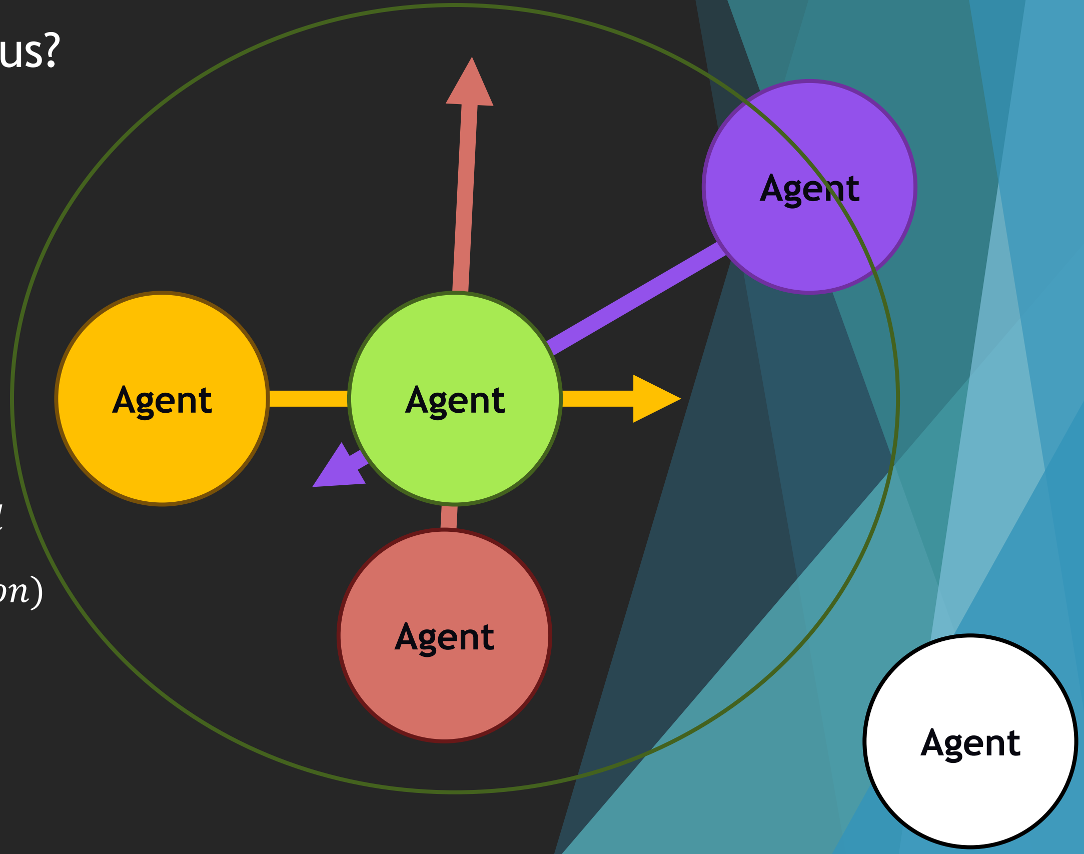
Steering Behaviours

- This was great for obstacles
- But how can we avoid other agents in a radius around us?
  - Helps avoid walking into each other
  - Spread out more?
- Try to move in the opposite direction of each agent
  - But by how much?



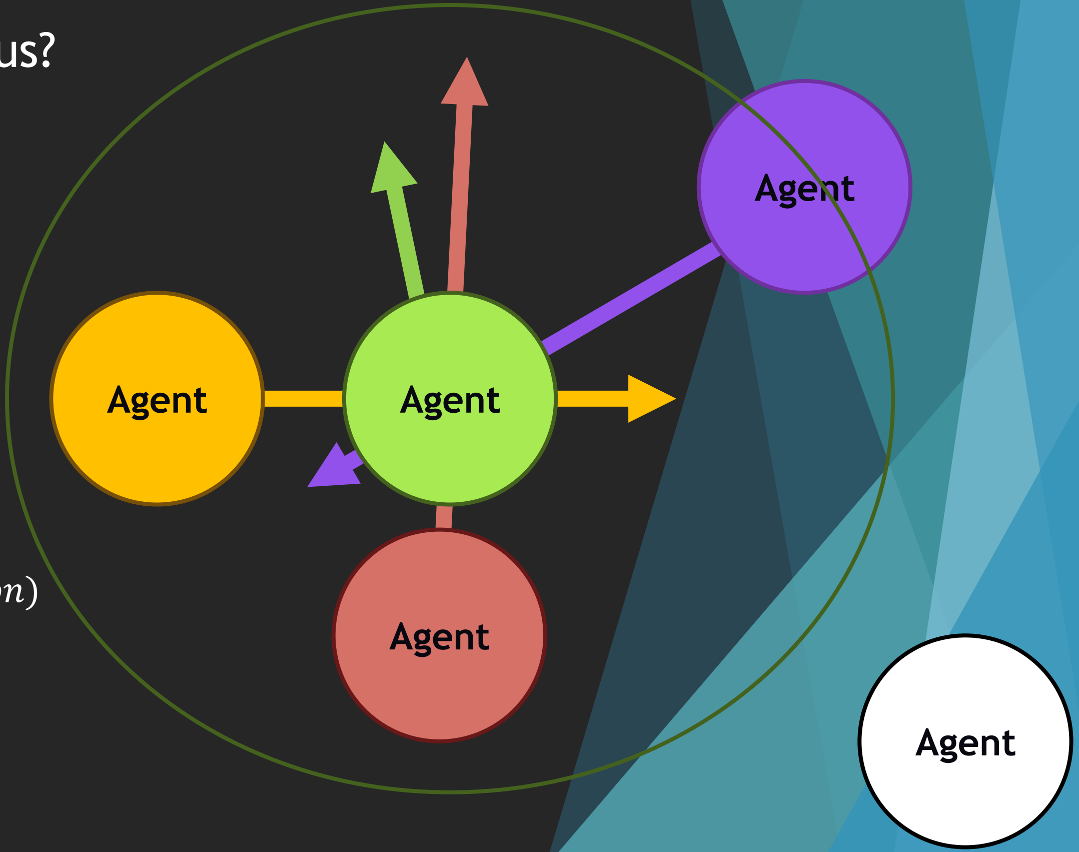
Steering Behaviours

- This was great for obstacles
- But how can we avoid other agents in a radius around us?
  - Helps avoid walking into each other
  - Spread out more?
- Try to move in the opposite direction of each agent
  - But by how much?
    - Inversely proportional to their distance!
    - Linear = $acceleration * (threshold - distance) / threshold$
    - Inverse Square = $min(k / (distance * distance), Acceleration)$



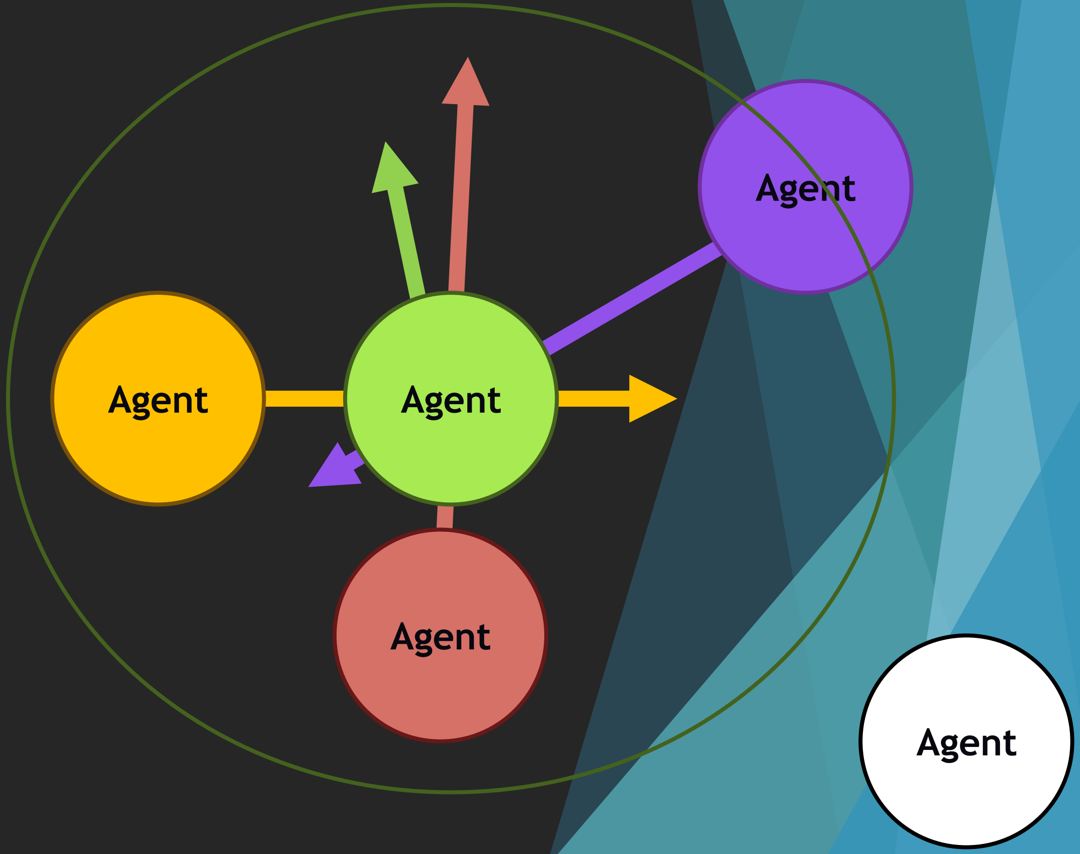
Steering Behaviours

- This was great for obstacles
- But how can we avoid other agents in a radius around us?
  - Helps avoid walking into each other
  - Spread out more?
- Try to move in the opposite direction of each agent
  - But by how much?
    - Inversely proportional to their distance!
    - Linear = $acceleration * (threshold - distance) / threshold$
    - Inverse Square = $min(k / (distance * distance), Acceleration)$
- Seek towards the average of the positions



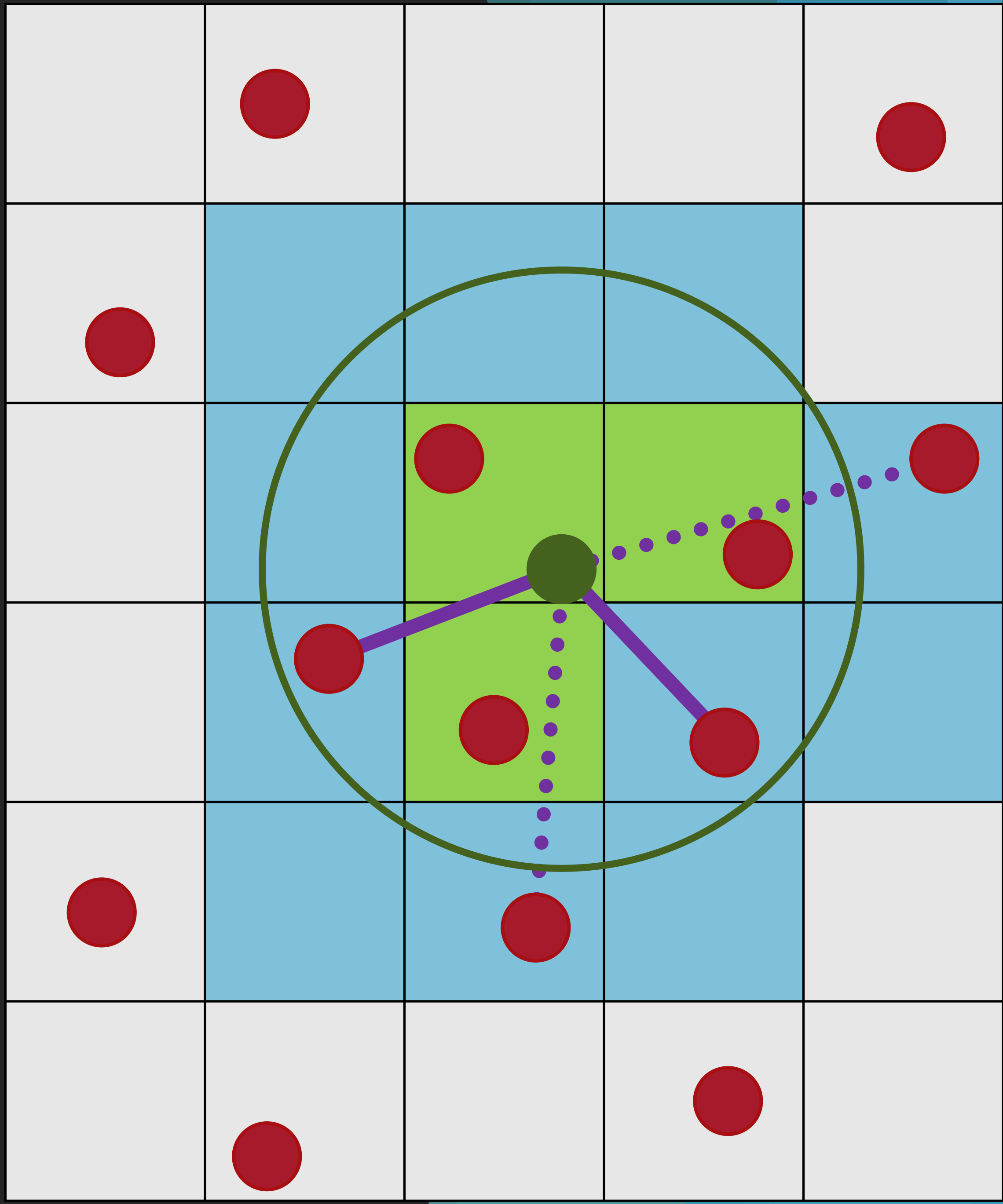
Steering Behaviours

- How efficient is this?
  - All prior methods only care about the current agent
    - O(1)
  - Need to compare against all other agents in the world!
    - O(n) even if no agents within the radius!
    - Running on all agents = O(n^2)
- What can we do to improve this?



Steering Behaviours

- Cache agents into chunks of the map
  - Divide the world by a set size
  - Whenever an agent moves, update its position
- We now know agents in close chunks are within distance
  - Manually check the distance of those in bordering cells
  - Don’t check those beyond that



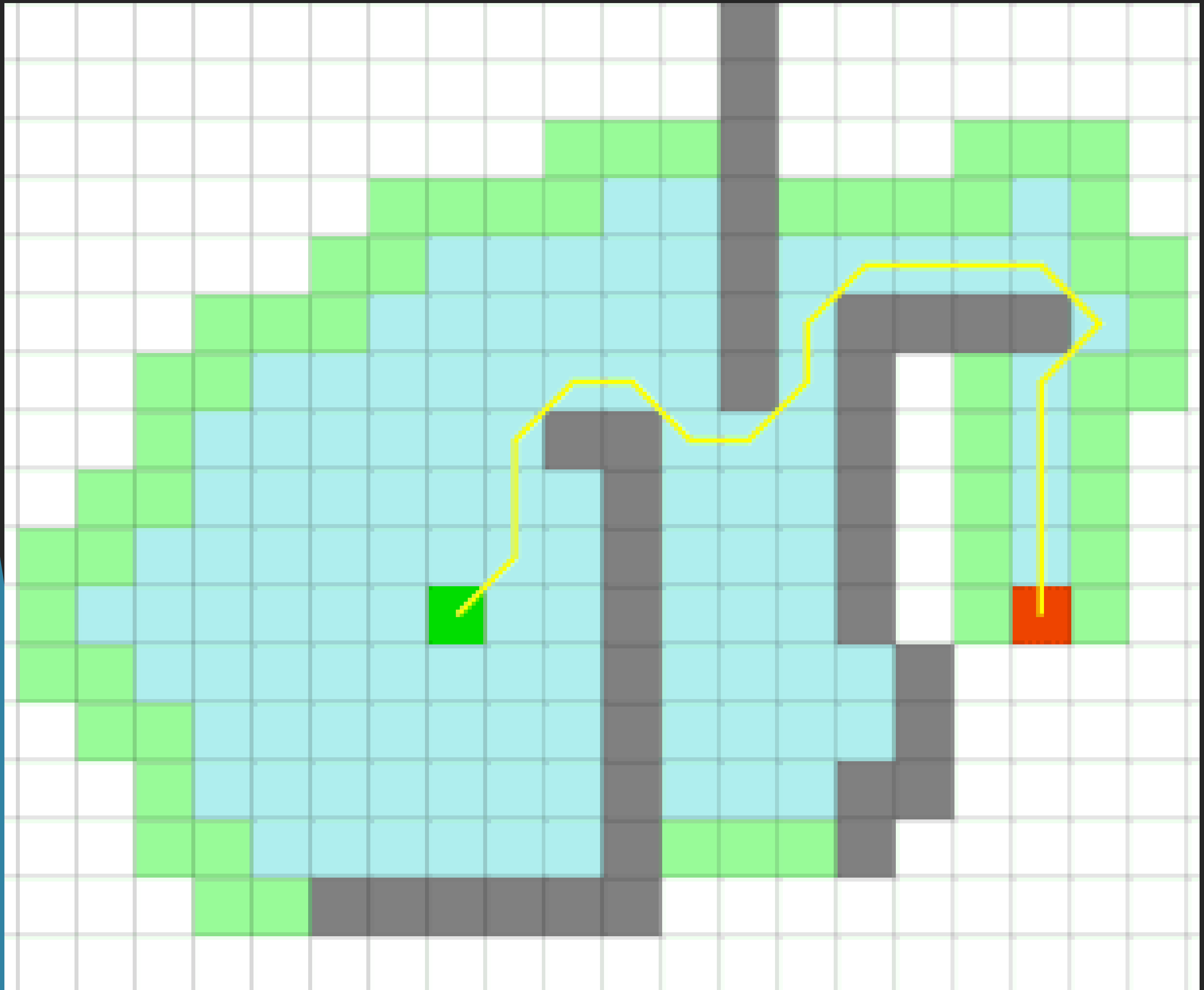
Multiple Steering Behaviours

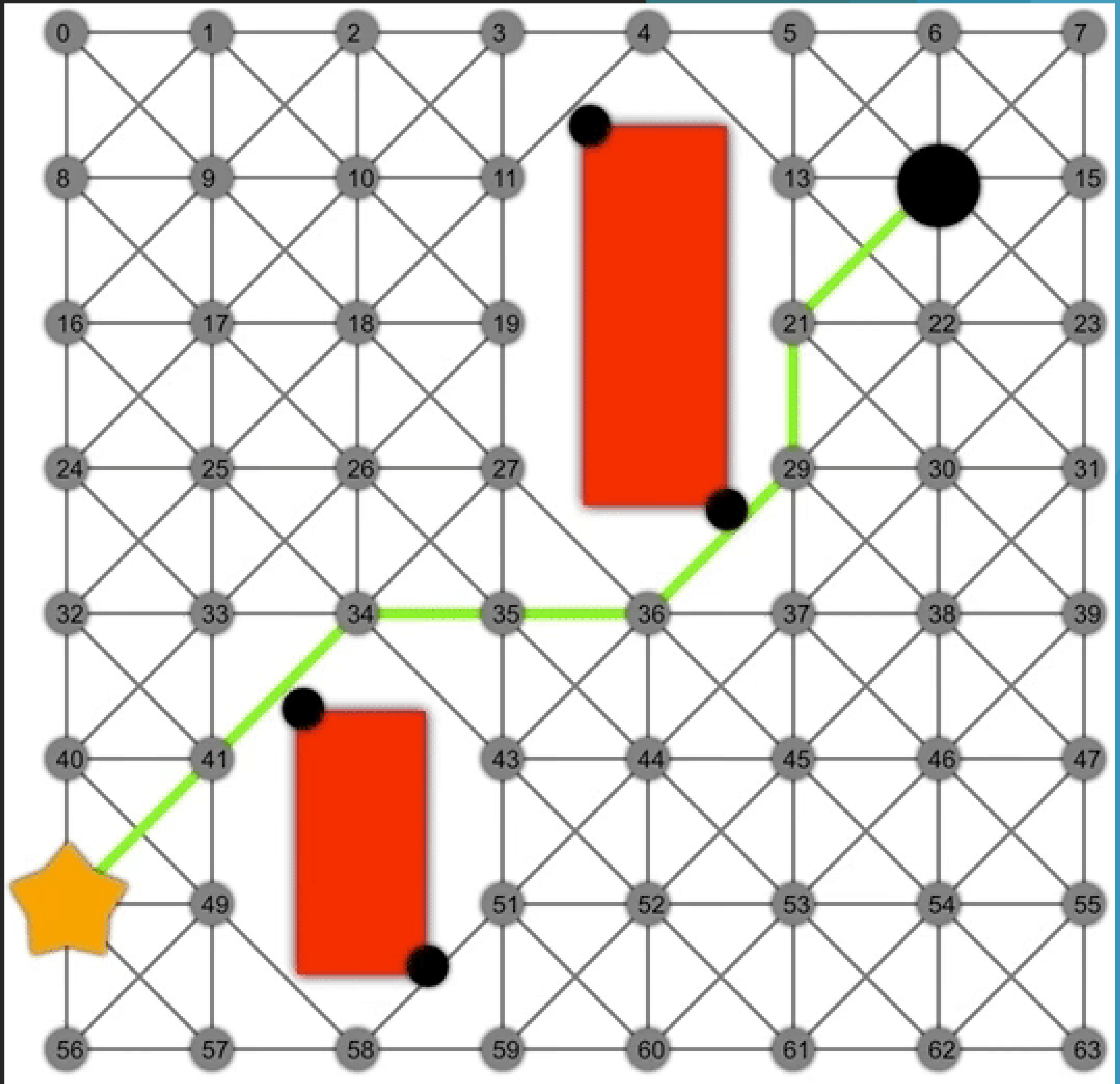
- Most common method is to assign a weight to each
  - Take the weighted average
- Some methods only run the top X highest weight or priority
- Potentially intelligently limit certain options
  - Can have separation and collision avoidance together
  - No point in having a seek, pursue, flee, or evade with each other

Path Finding

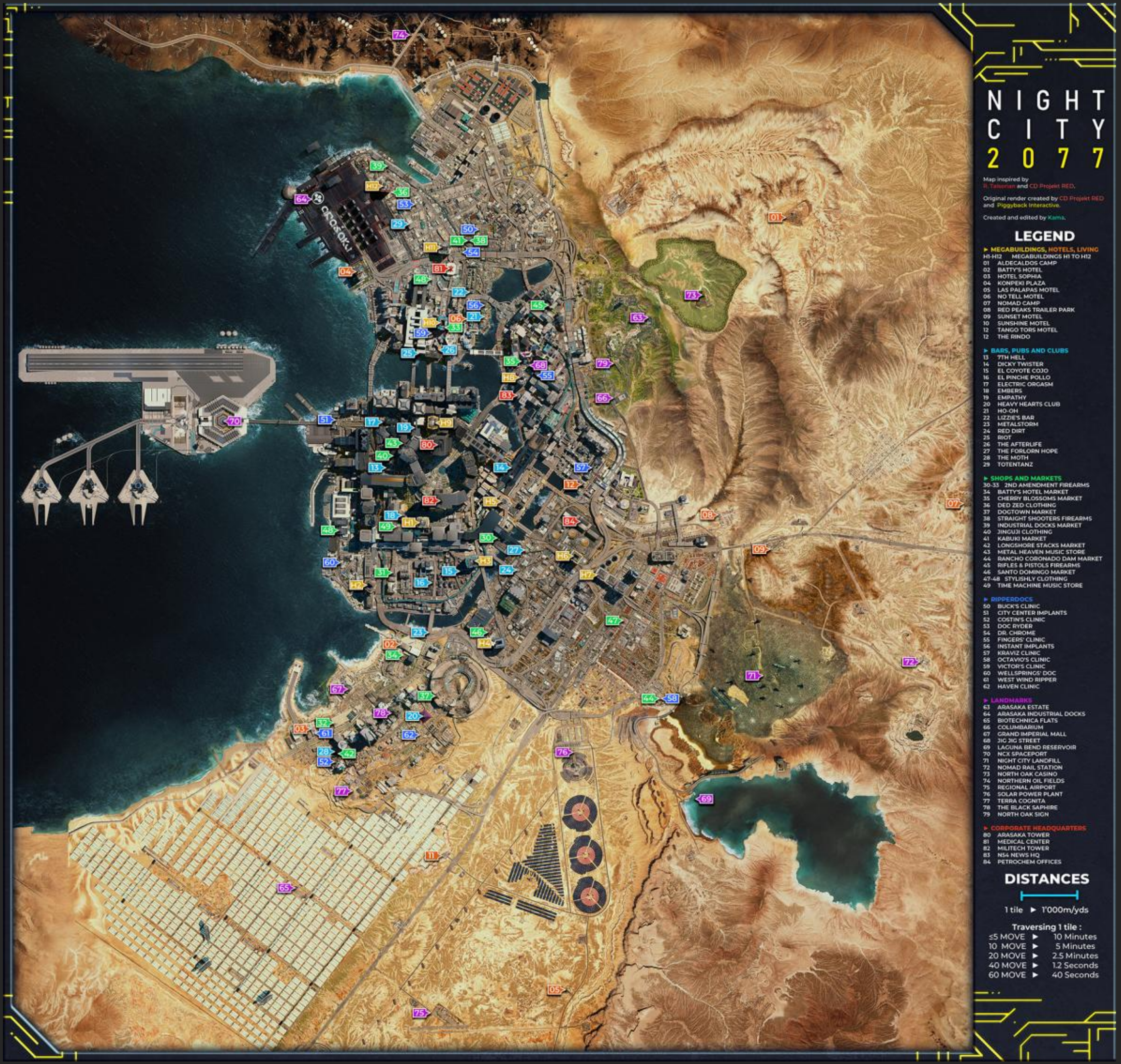
- Obstacle avoidance can work okay for simple environments, but can’t effectively get us around a complex environment
- Following a path is super easy
  - Just repeated seek calls
- How can we find a path to follow?
  - What do we need from our pathfinding algorithm?

A* Algorithm




0
1
2
3
4
5
6
7
8
9
10
11
13
15
16
17
18
19
21
22
23
24
25
26
27
29
30
31
32
33
34
35
36
37
38
39
40
41
43
44
45
46
47
49
51
52
53
54
55
56
57
58
59
60
61
62
63

Where do we even start?



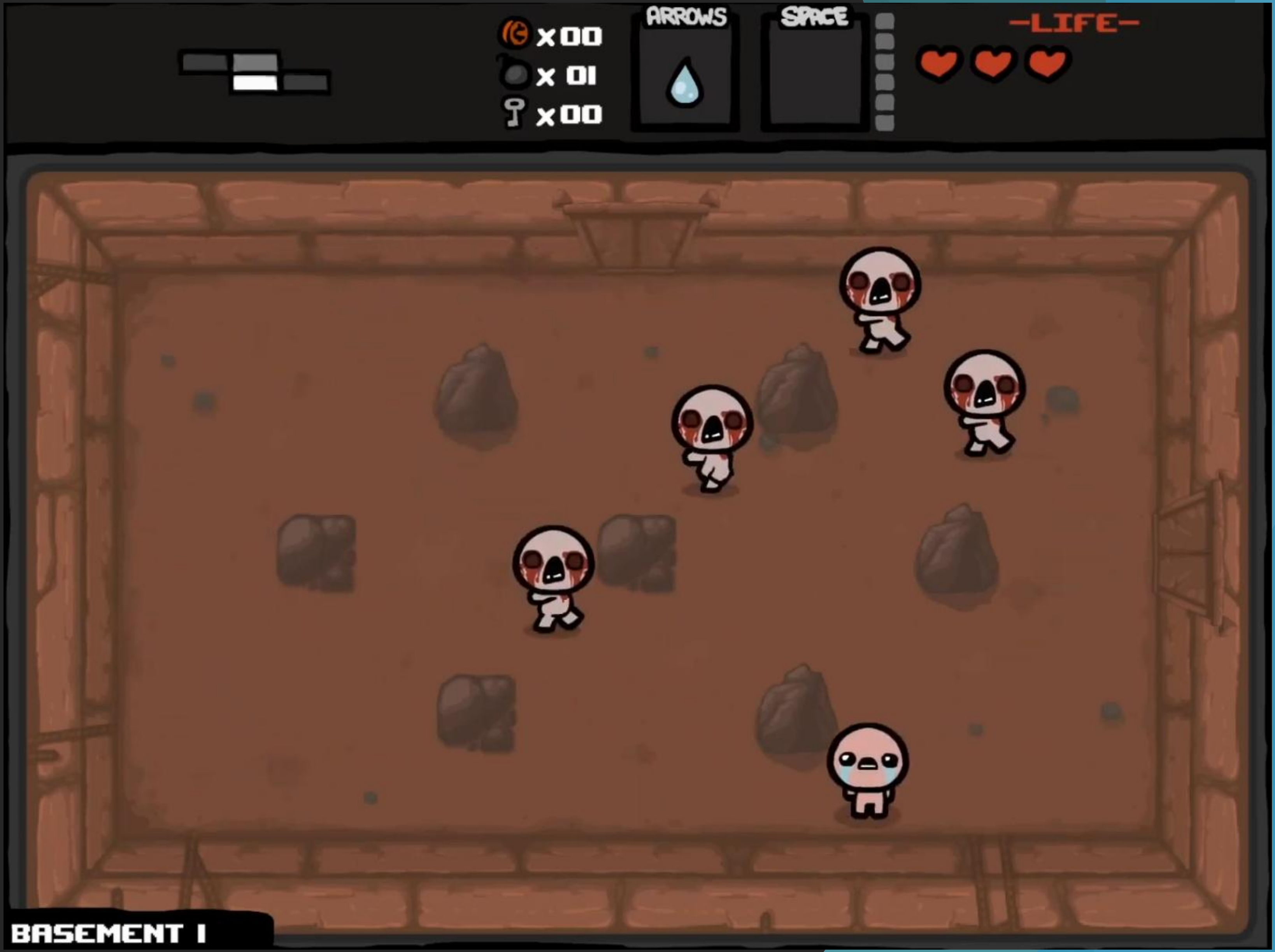
Encoding our Worlds for Path Finding

- How can we utilize A* in our virtual worlds?

1. Your level is already a grid 😊
2. Your level is not already a grid 😔
    - Place nodes in the world

Grid Levels

1. Encode in your grid structure
2. Choose a heuristic function
3. Run A* to the destination
4. Follow that path
5. Profit*
    - *I am not actually guaranteeing the financial success of your game

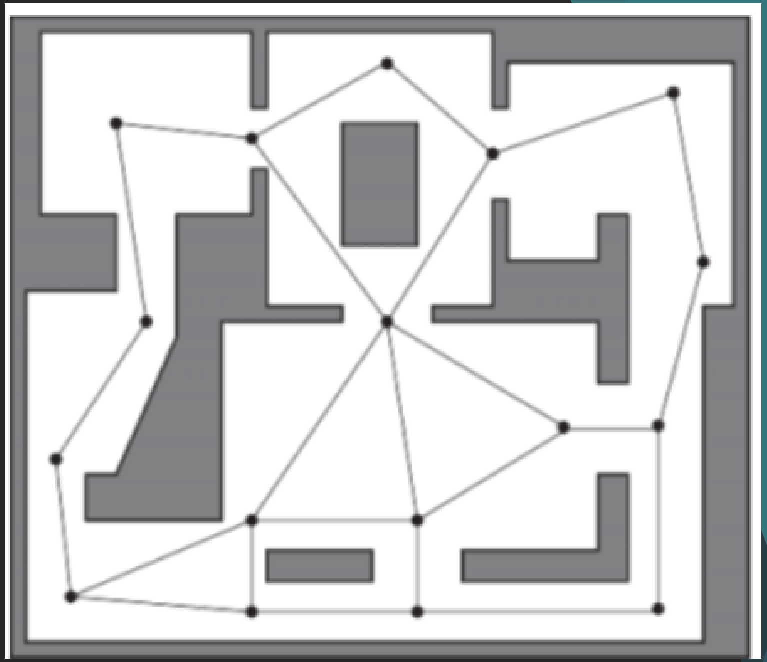


Non-Grid Levels

- How can we place nodes?

1. Manually
2. Automatically

Manual Node Placement

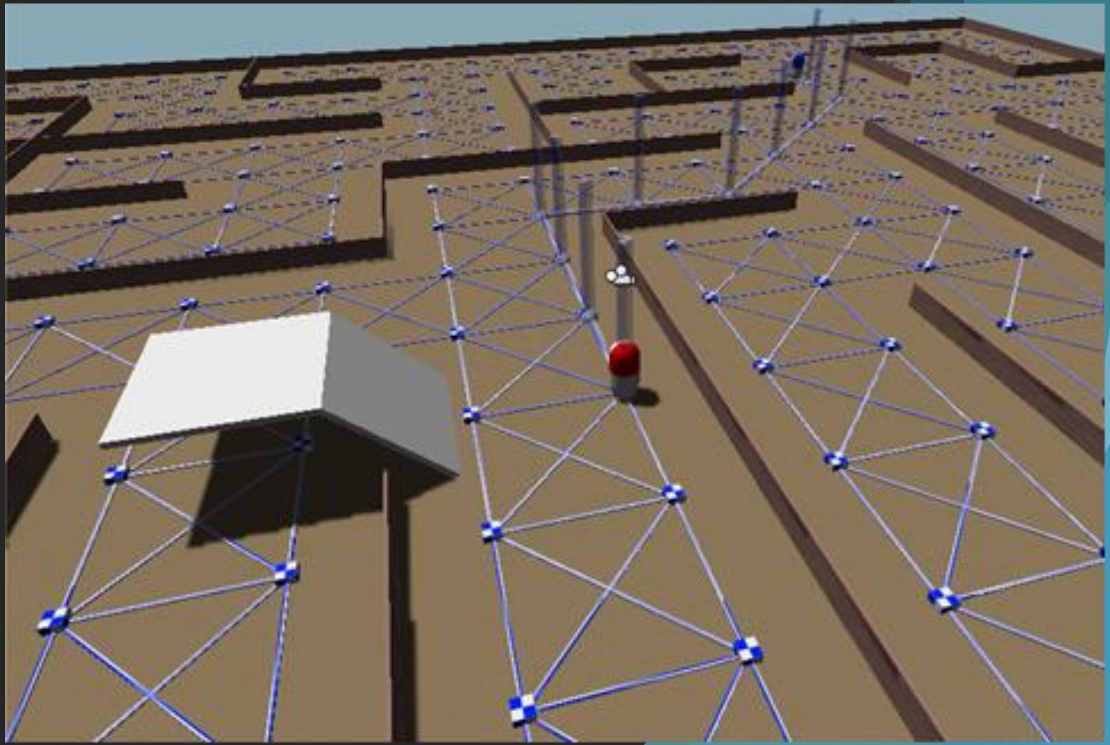


Pros

- Can manage total graph size

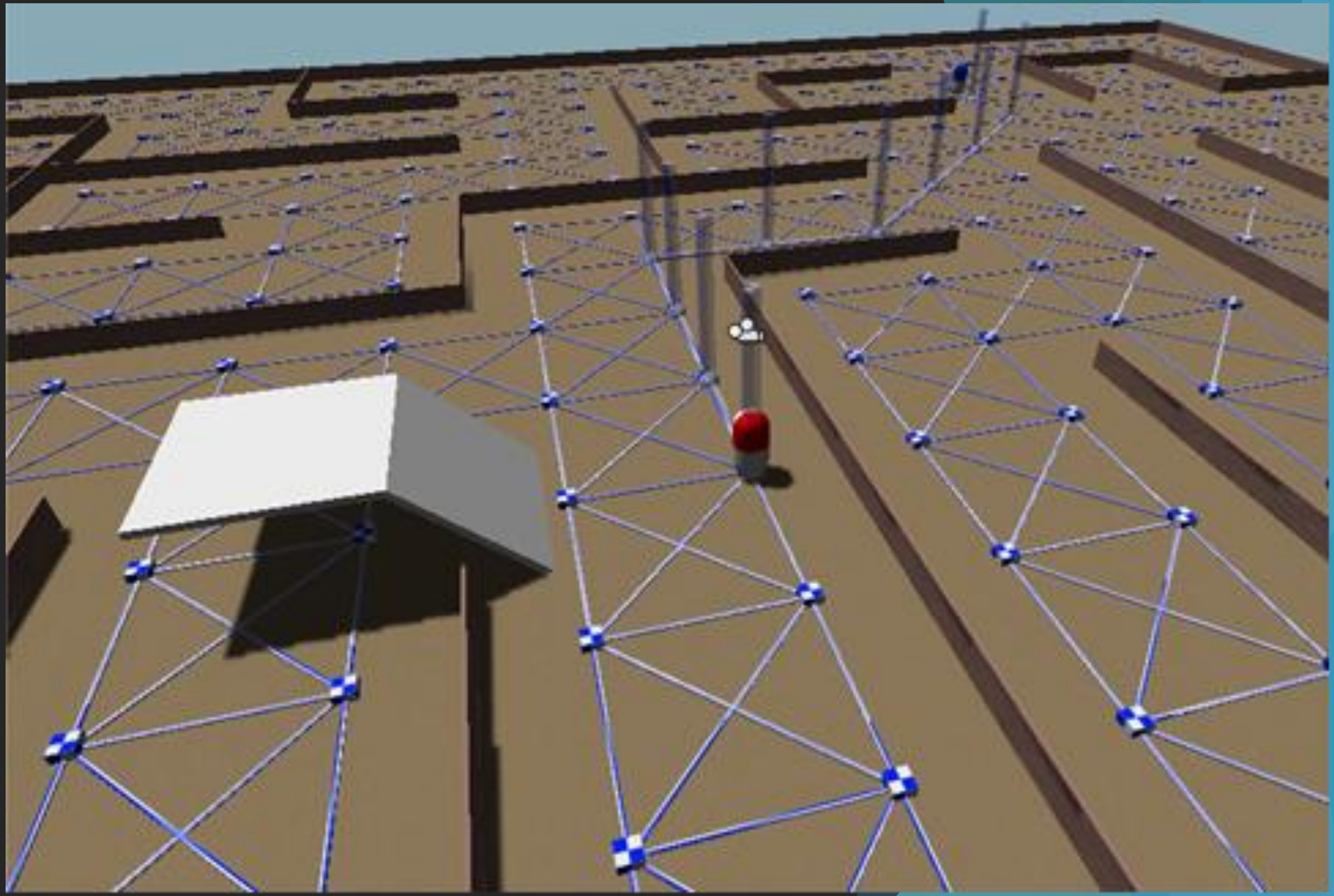
Cons

- Need to place literally every node
  - And connect them!
  - Every level, and every change



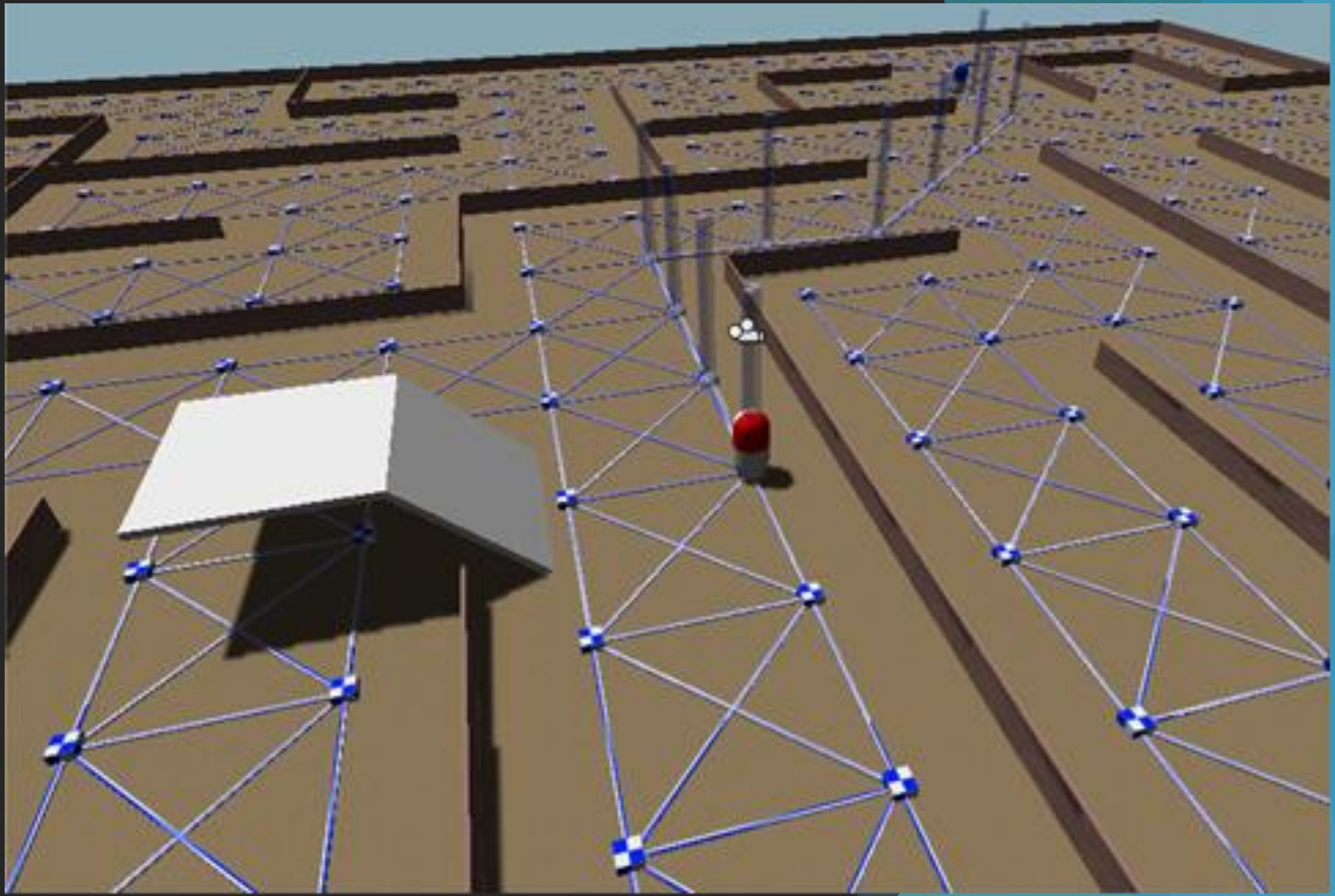
Automatic Node Placement

- How could we place in a simple environment like this?
  - Place and connect?
- Placing – Raycast from the sky
  - Ground? Place node!
  - Obstacle? Don’t place node
- Connecting – Line of sight
  - See other node? Connect!
- Problems with this?



Automatic Node Placement

- Implementation won’t work for every game
  - Roofs or indoor areas
- Assume we can place the nodes
  - So many nodes!
  - Pathfinding will take a long time!



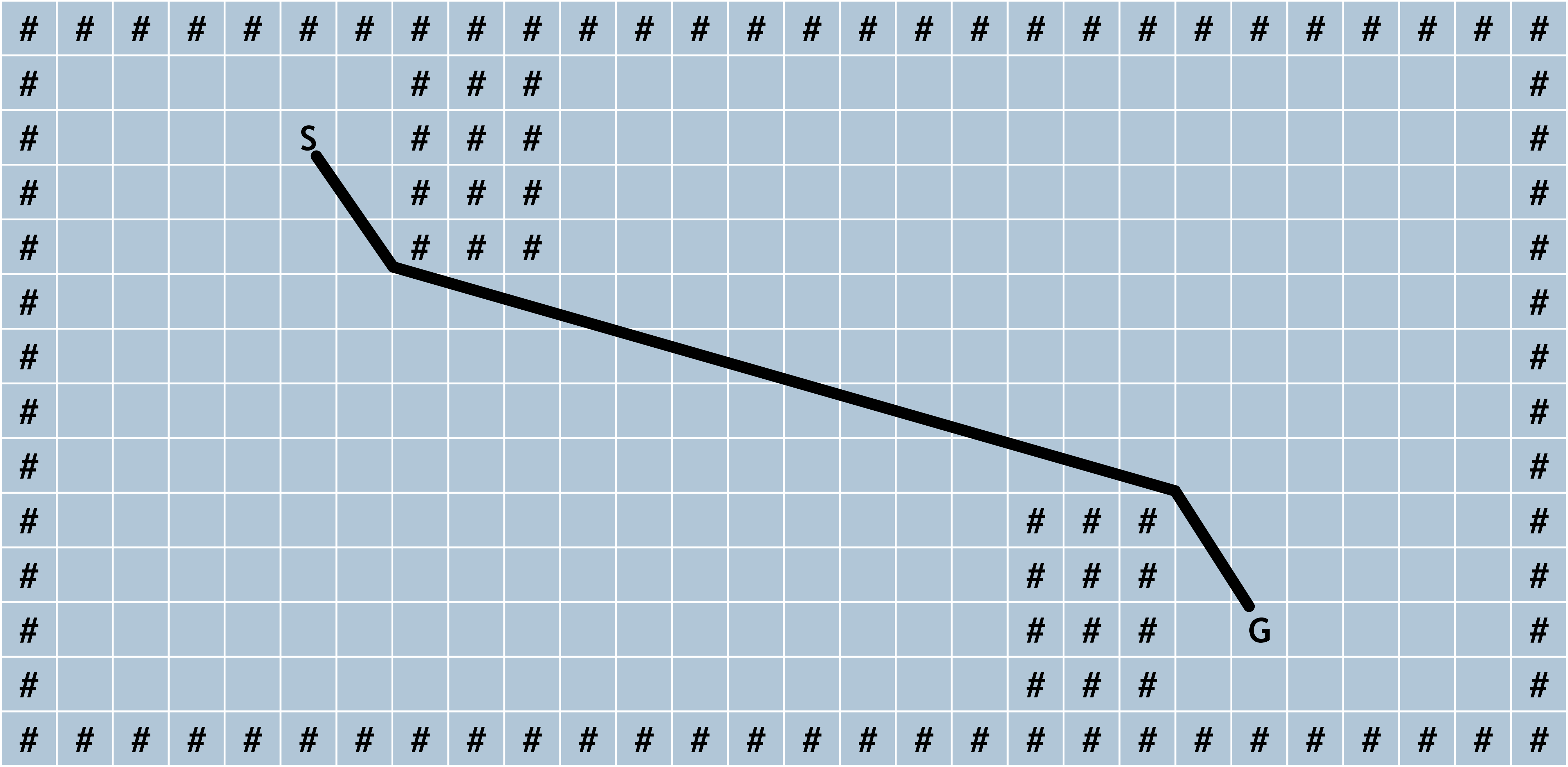
Where do we need nodes?

#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Where do we need nodes?

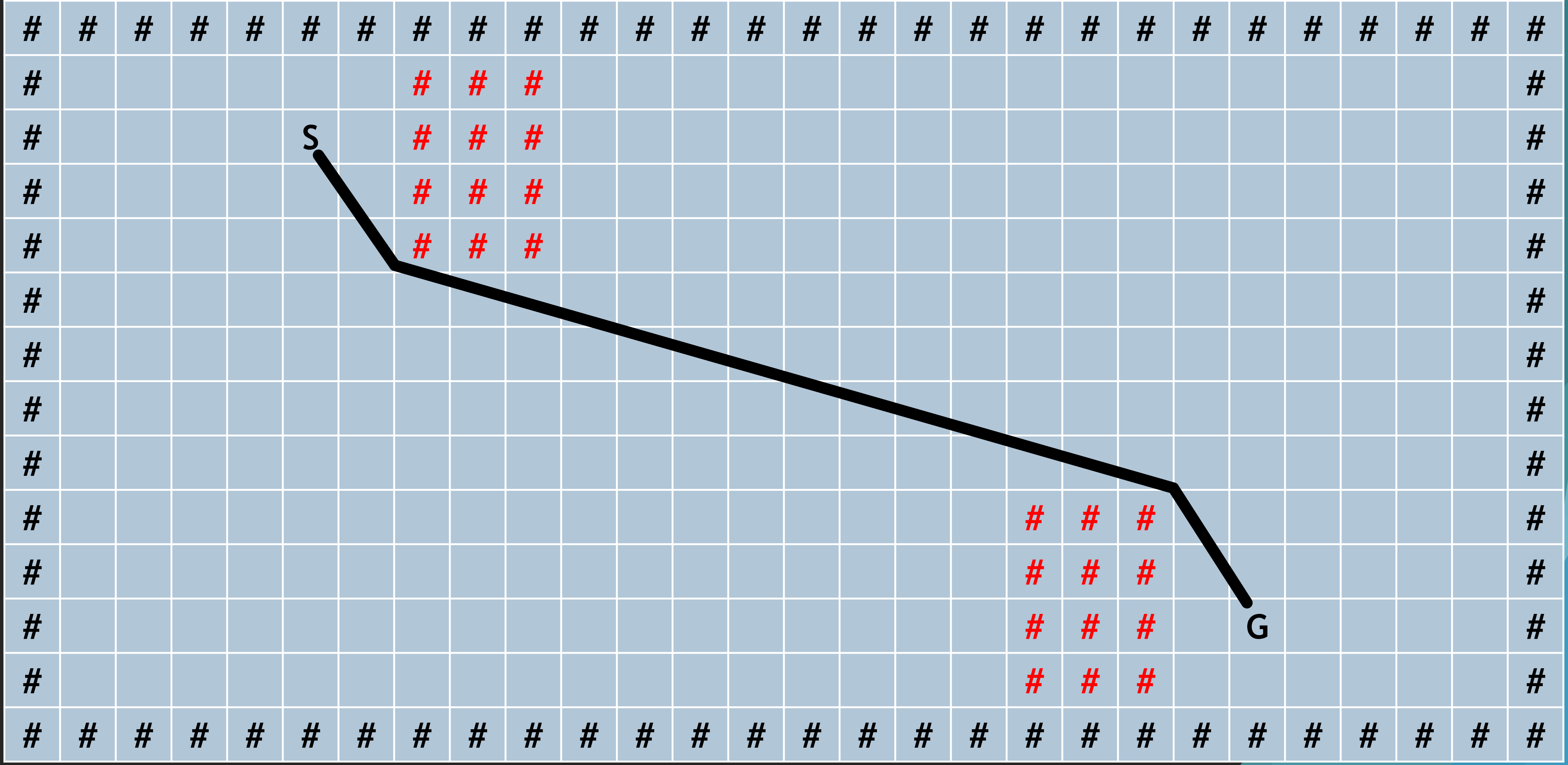
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#					S		#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#		G					#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Where do we need nodes?



#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#					S		#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#		G					#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Needed – Around Obstacles!



#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#					S		#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#		G					#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Identify Convex Corners

#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Corner Graph in Practice

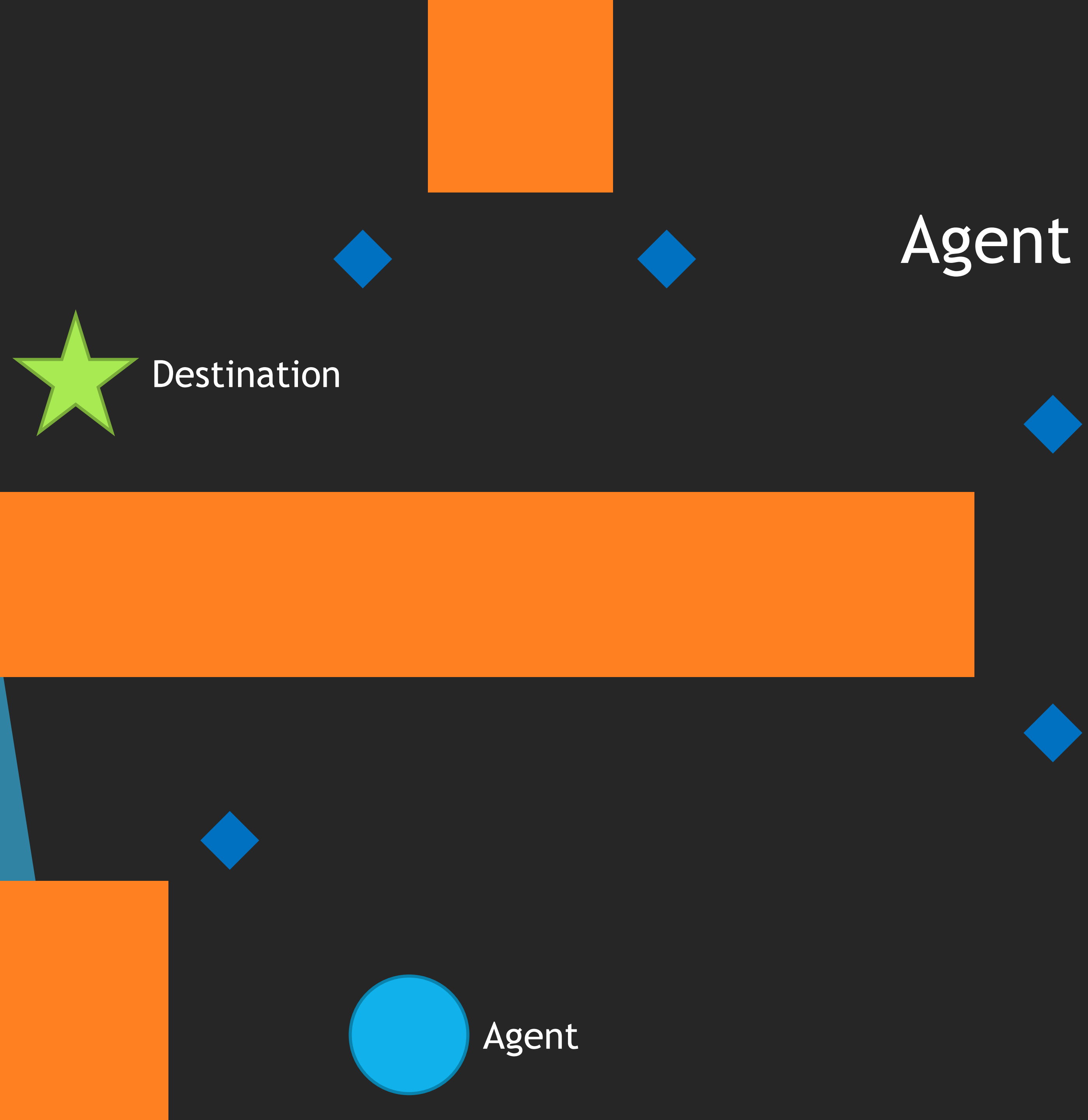
- Using this method, to find a path we:

1. Find the node nearest to our agent
2. Find the node nearest to our target
3. Find a path between the nodes

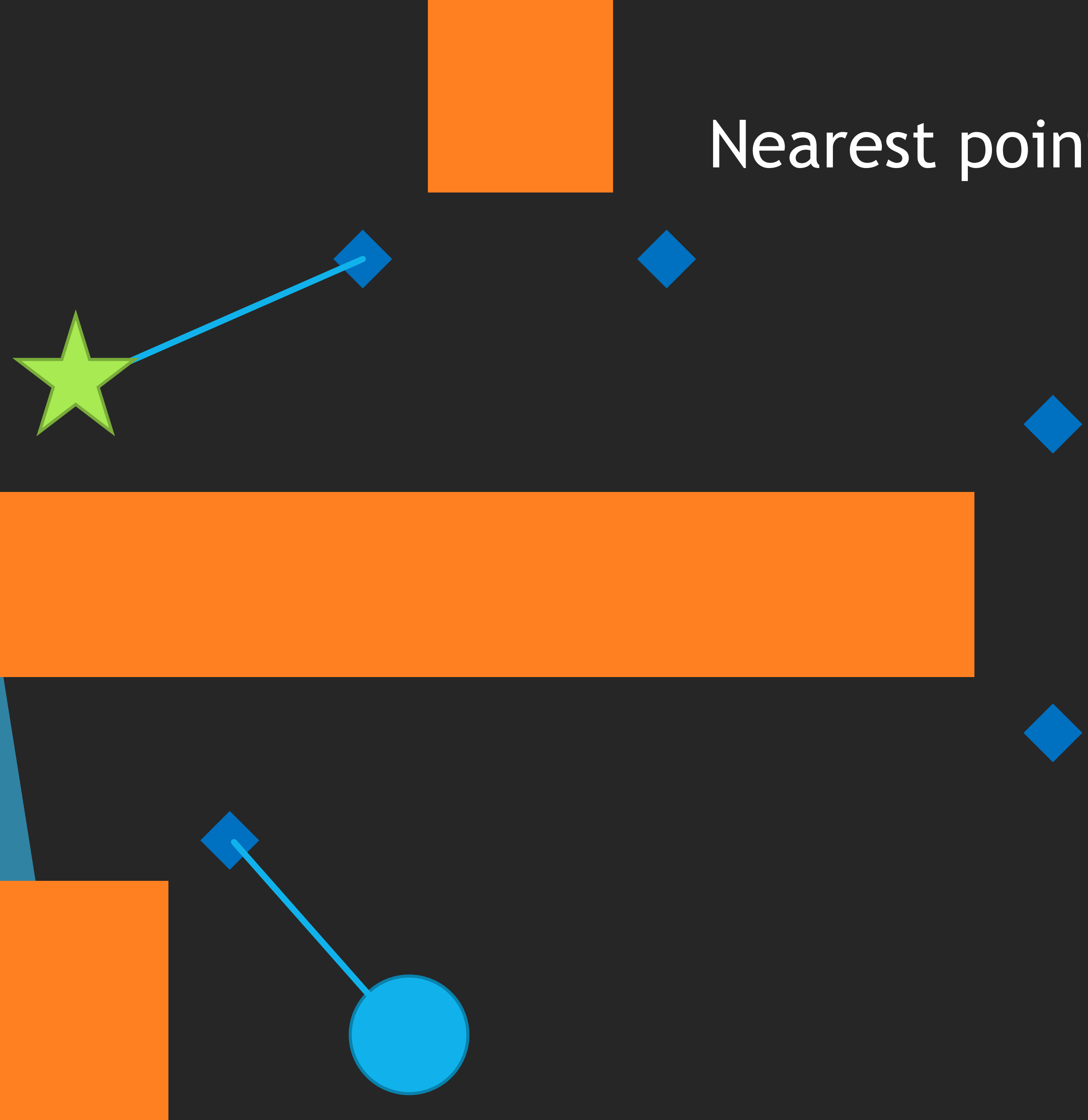
- Once the path is found, we:

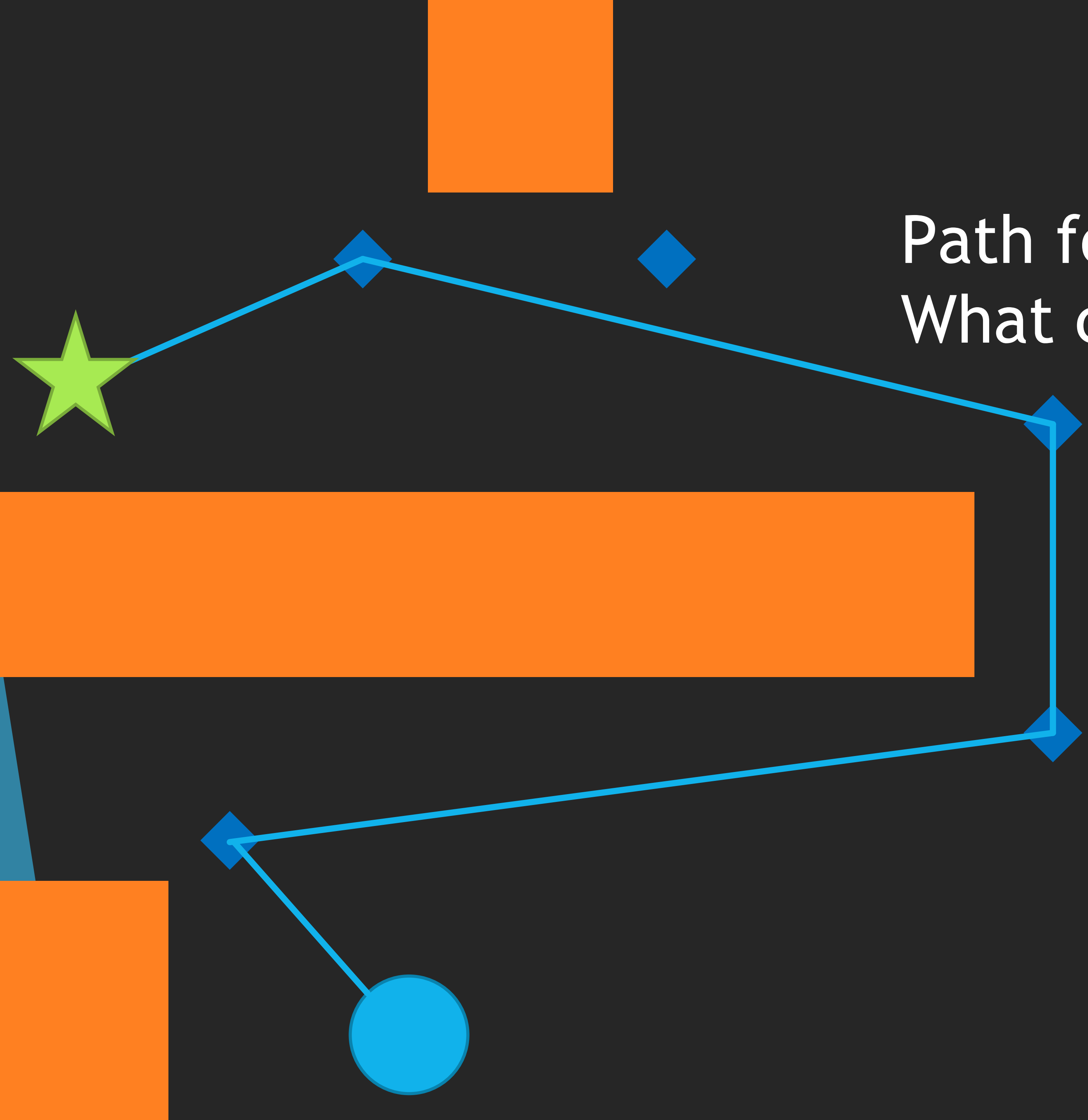
1. Move (seek) to the nearest starting node
2. Follow the found path
3. Move (seek) to the target

Agent sends request

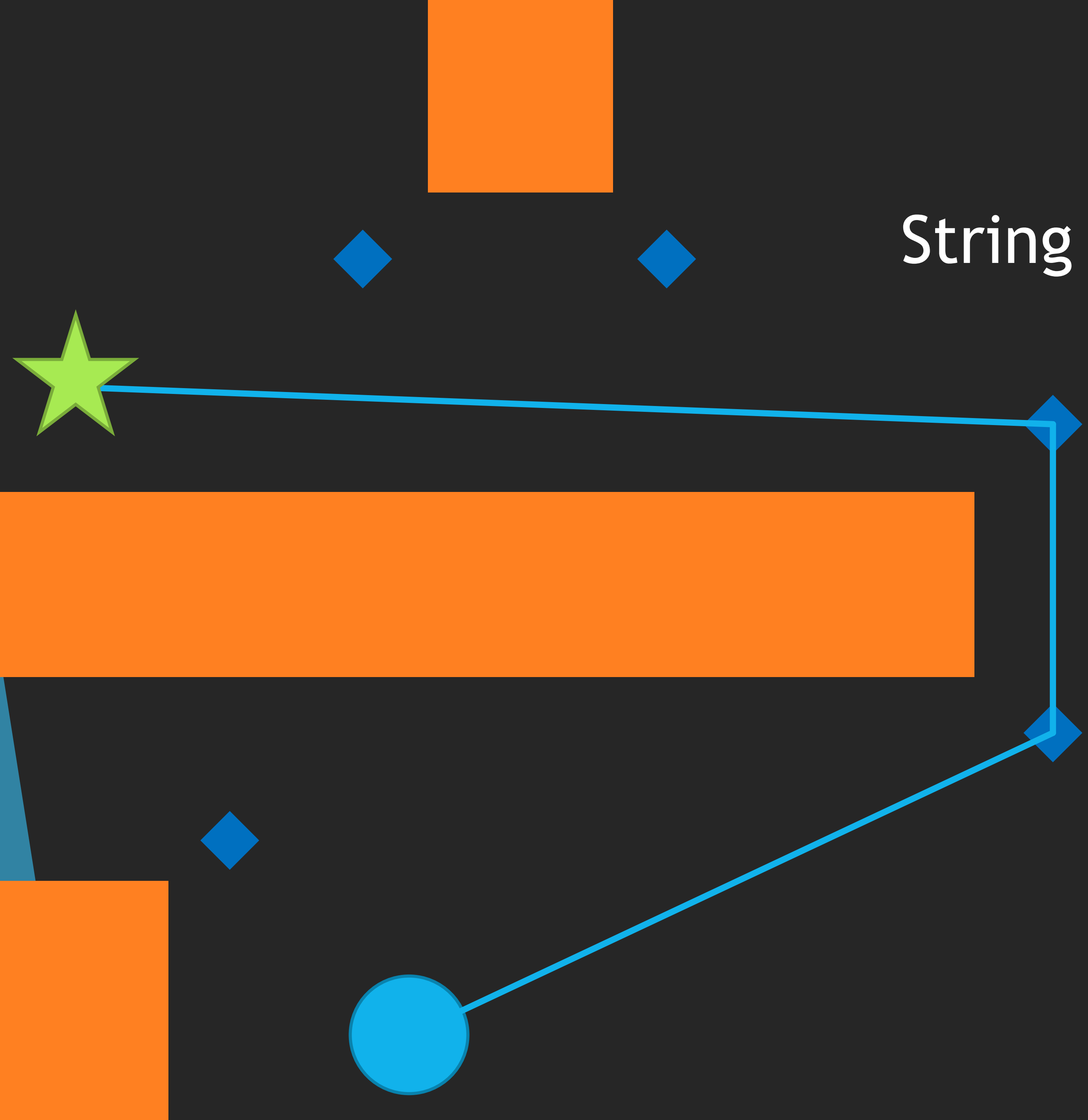


Nearest points determined





Path found
What can we improve?



String pulling!

Corner Graphs

- Nice in theory but how do we place these?
  1. Using the ray casting method – Do checks for the open spaces and keep corners
     - This was part of an old assignment for this class
  2. Extract mesh information from the world
- Why are these corners all we need?
  - What is so special about them?

How can we break up the level?

#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

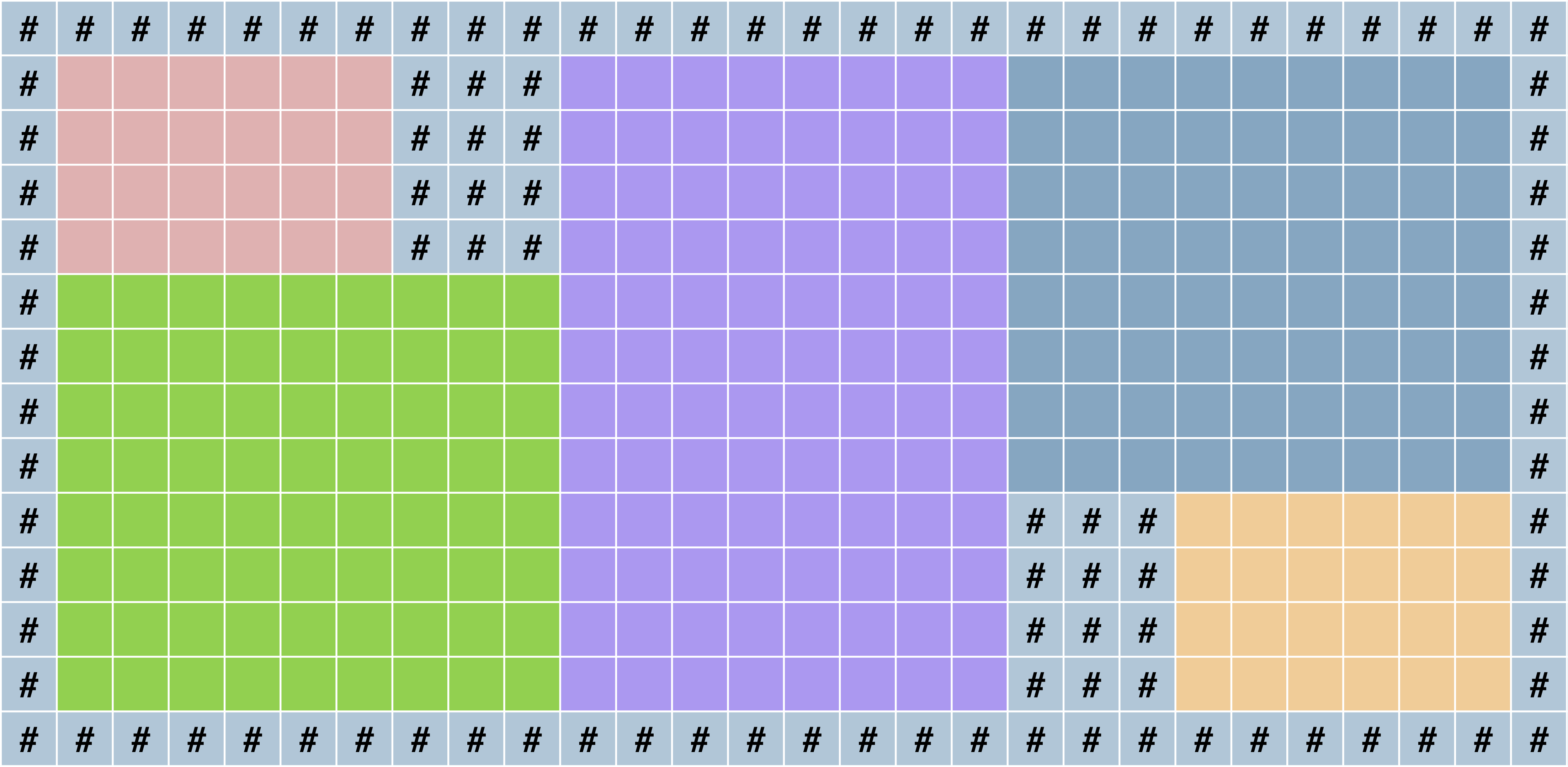
Potentially like this

#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Or maybe this

#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Or this



#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#							#	#	#																		#
#																											#
#																											#
#																											#
#																											#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#																		#	#	#							#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

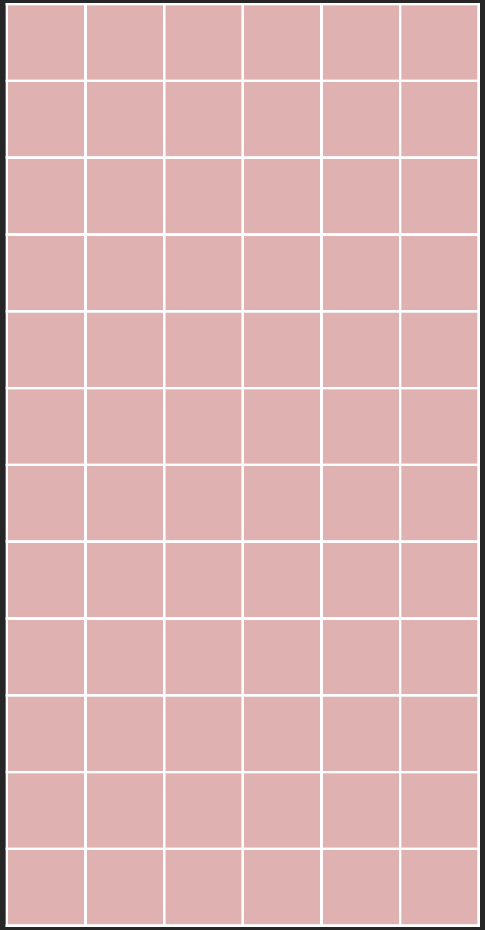
Corner Graphs

- Which layout is correct?
  - All of them!
- Why?

Corner Graphs

#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Single Part

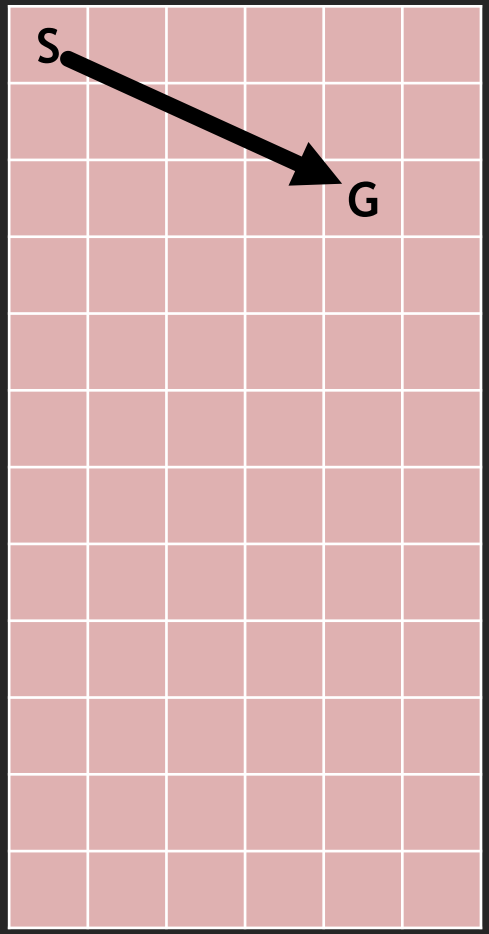


Single Part

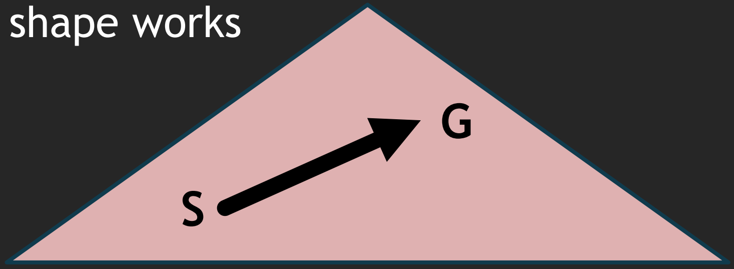
S					
				G	

- If I ask you to move from the start position to the goal, how can you do that?

Single Part



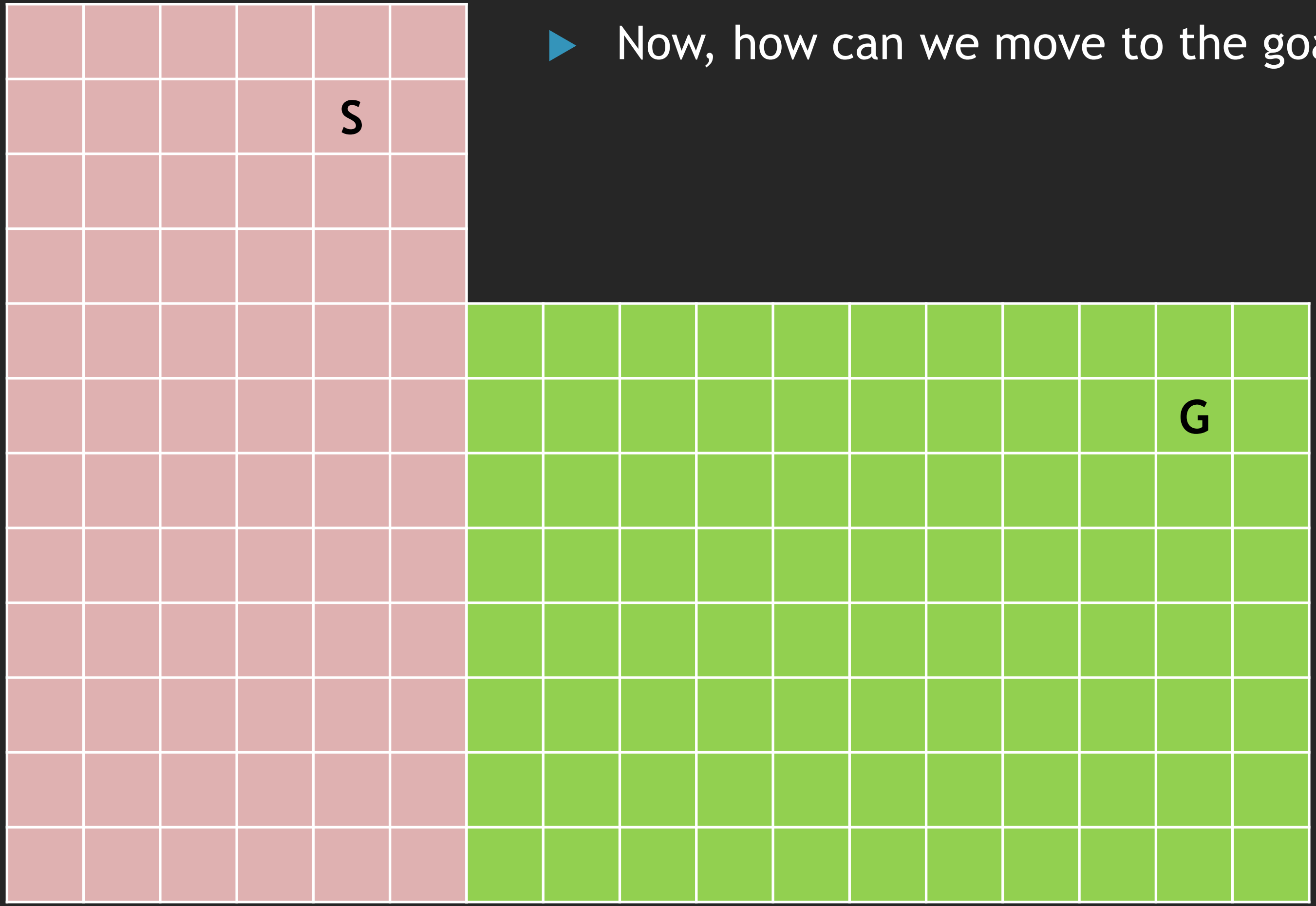
- If I ask you to move from the start position to the goal, how can you do that?
  - Just seek!
- If I ask you to move anywhere in this part, how can you do that?
  - Always just seek!
- Why does this work?
  - Rectangle → **Convex shape**
  - Any convex shape works



Corner Graphs

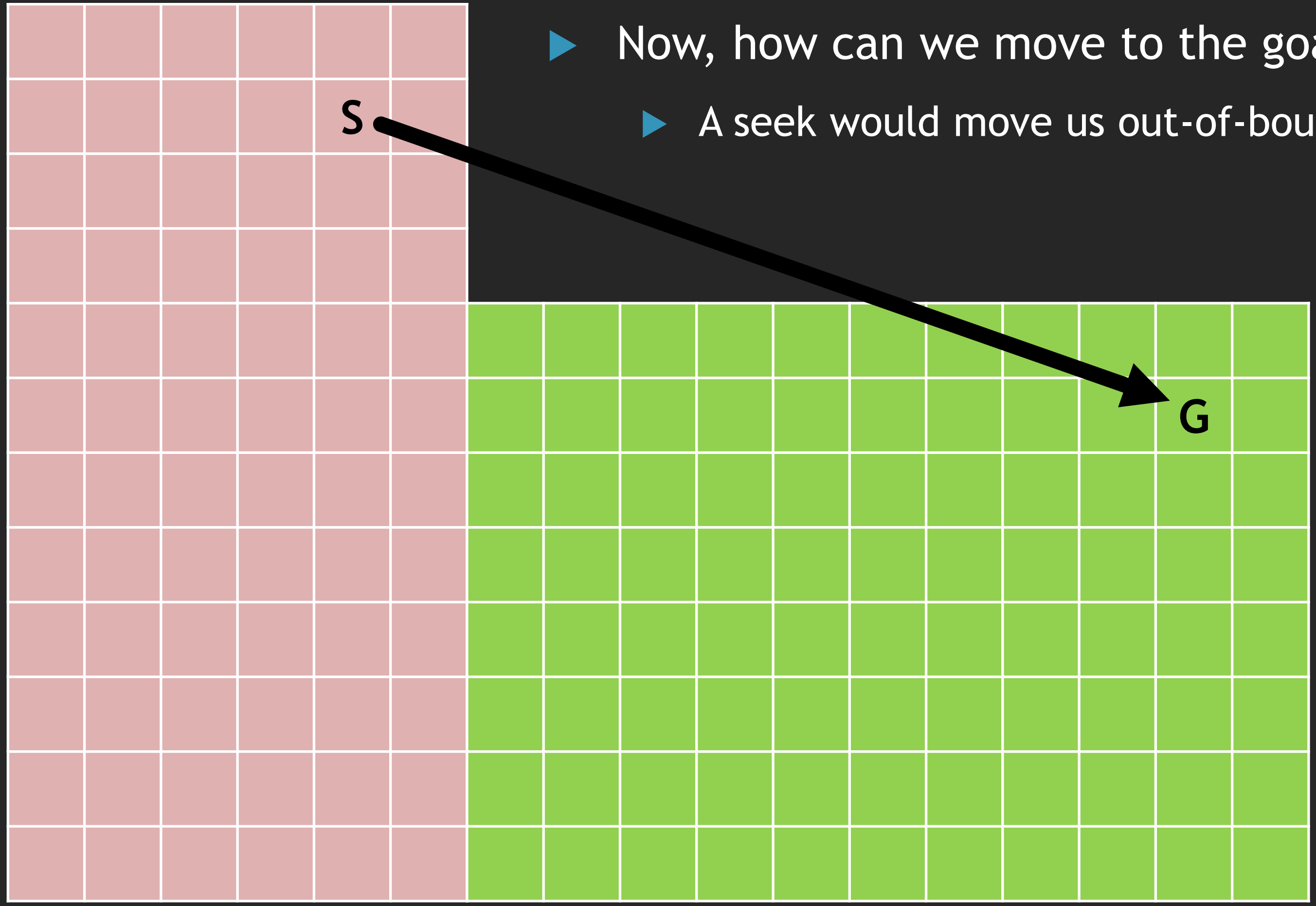
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	#	#	#	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	#	#	#	🟧	🟧	🟧	🟧	🟧	🟧	#
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Two Parts



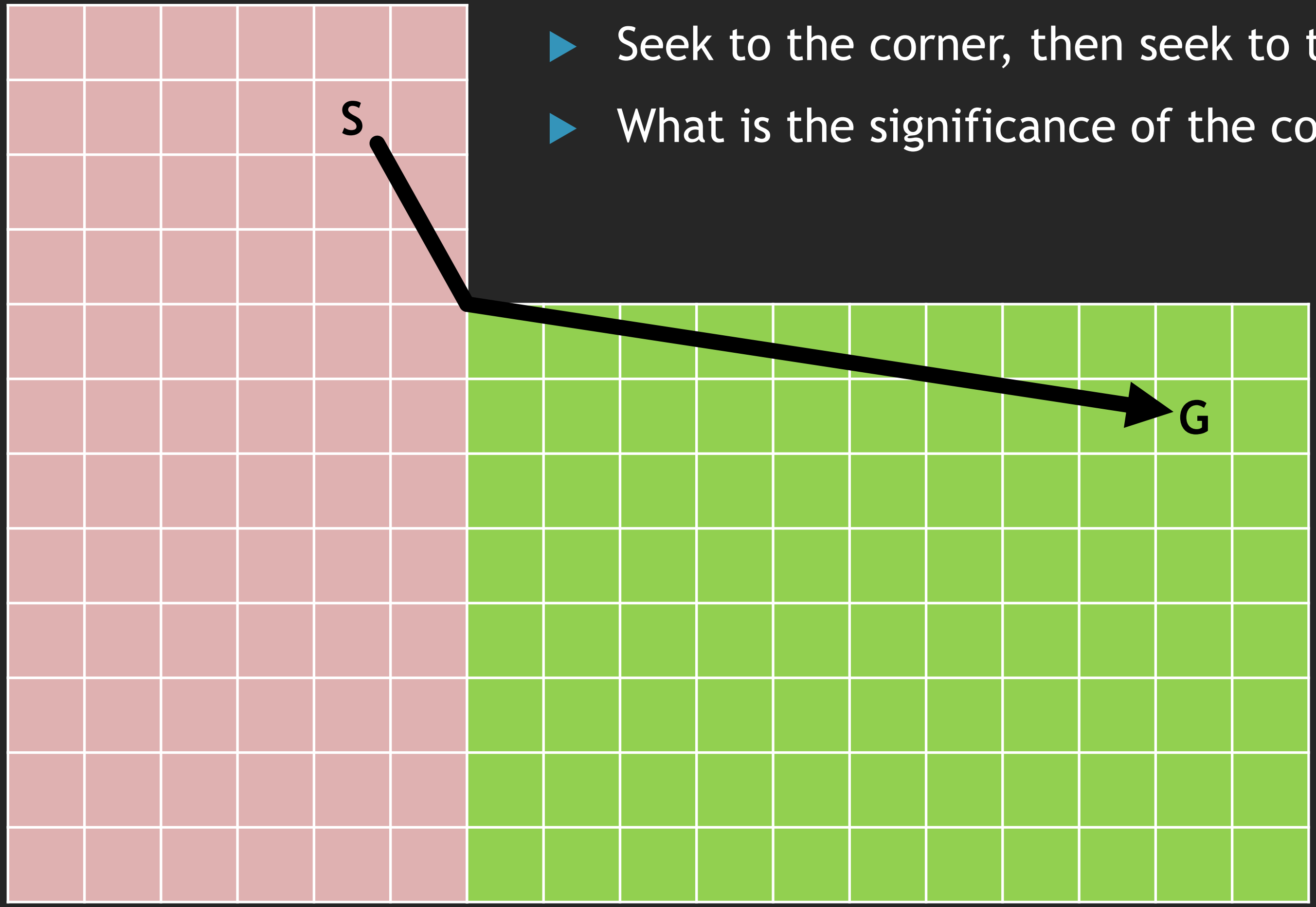
- Now, how can we move to the goal?

Two Parts



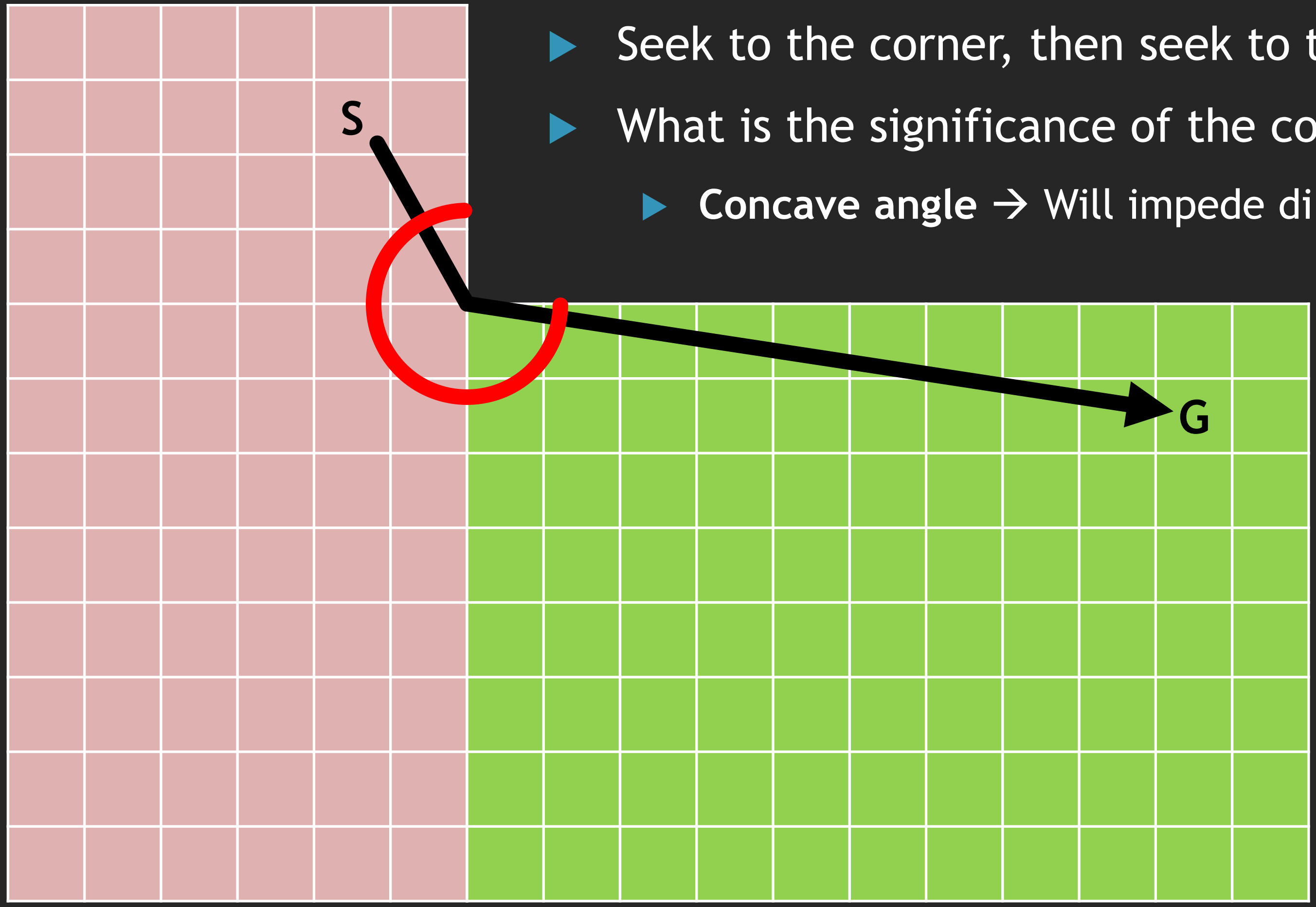
- Now, how can we move to the goal?
  - A seek would move us out-of-bounds

Two Parts



- Seek to the corner, then seek to the goal!
- What is the significance of the corner?

Two Parts

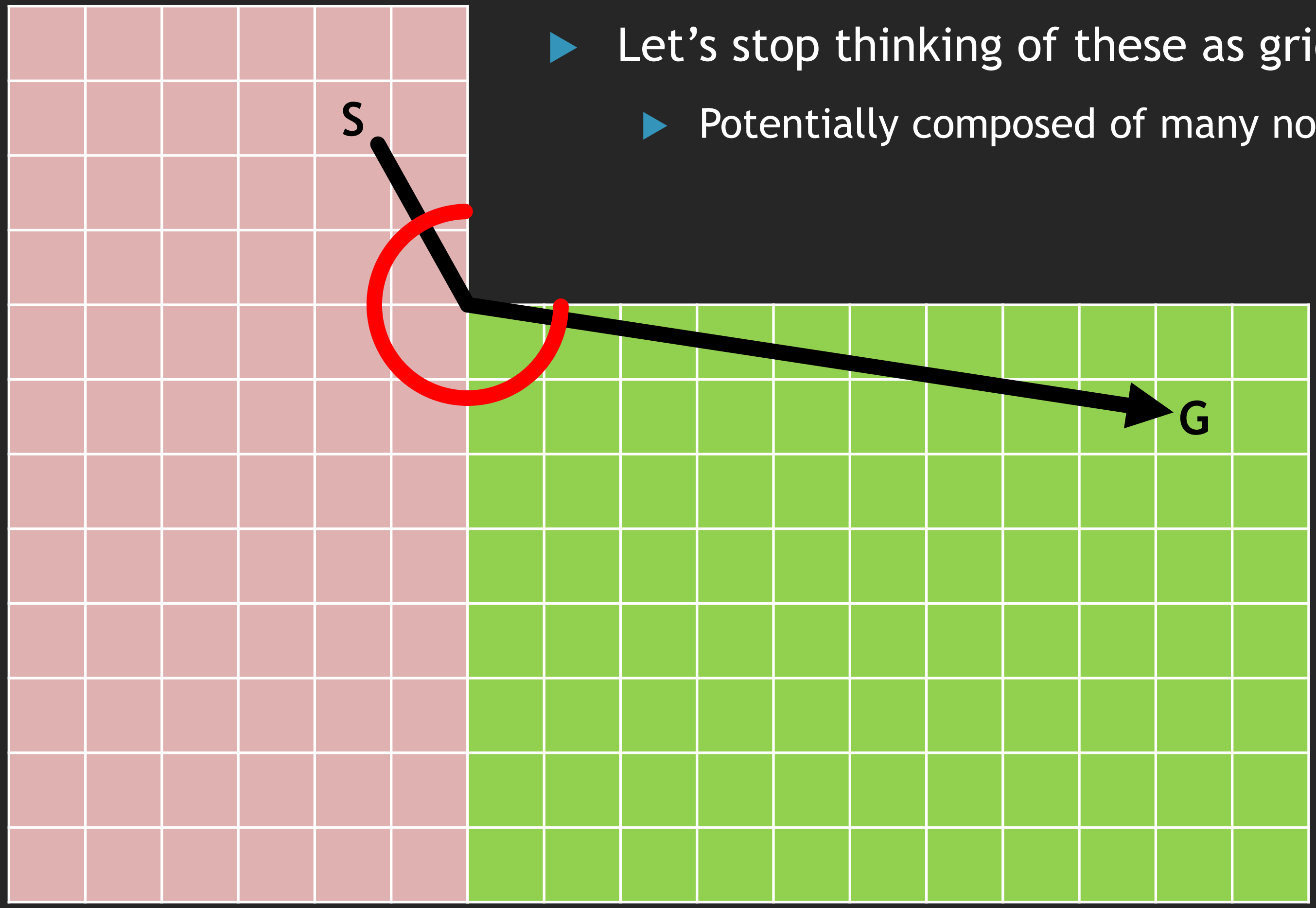


- Seek to the corner, then seek to the goal!
- What is the significance of the corner?
  - **Concave angle** → Will impede direct seeks

Takeaways

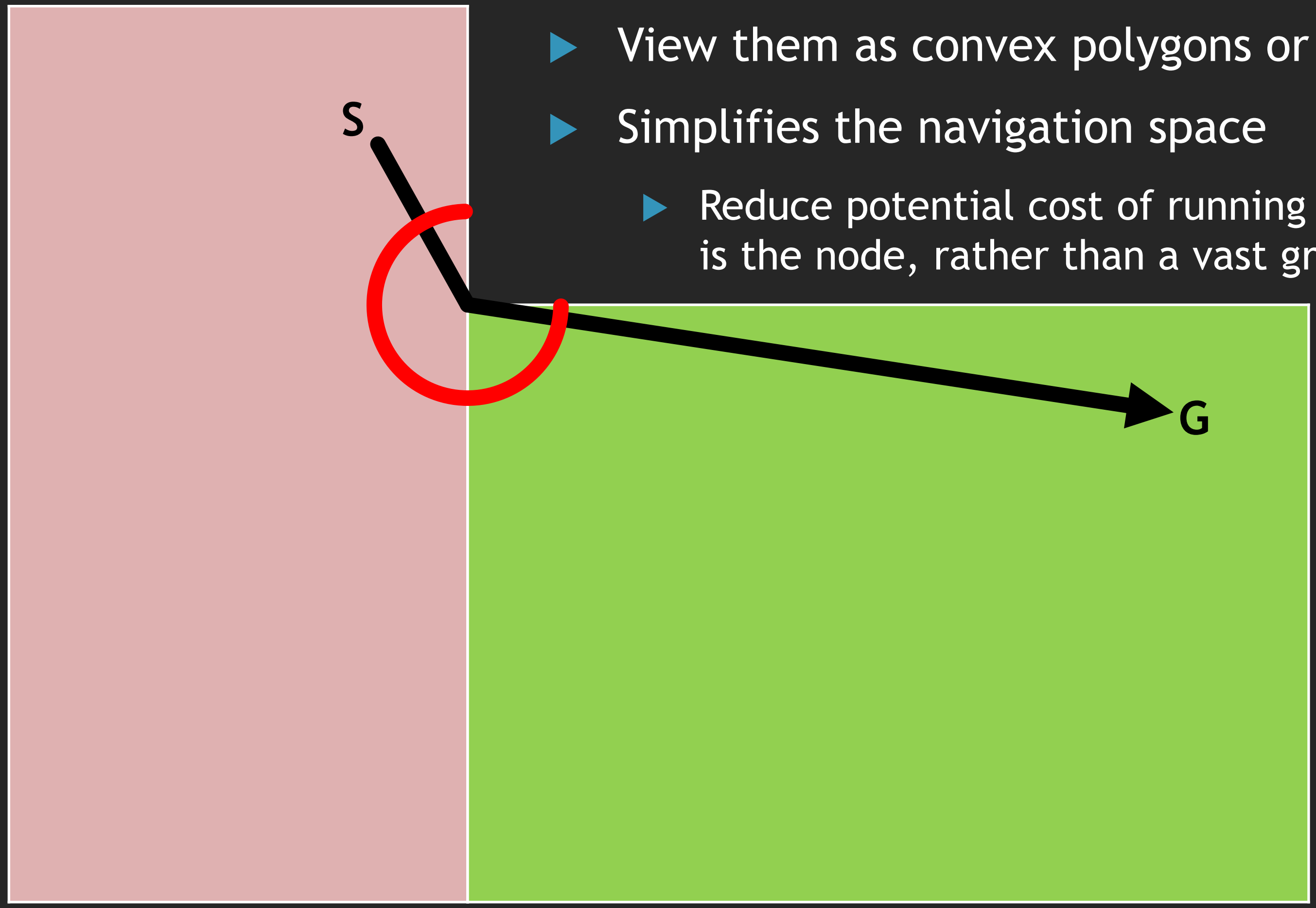
- Concave corners impede direct movement → Require navigation
- What are we navigating between?
  - The convex polygons!

Polygon Navigation



- Let’s stop thinking of these as grids
  - Potentially composed of many nodes

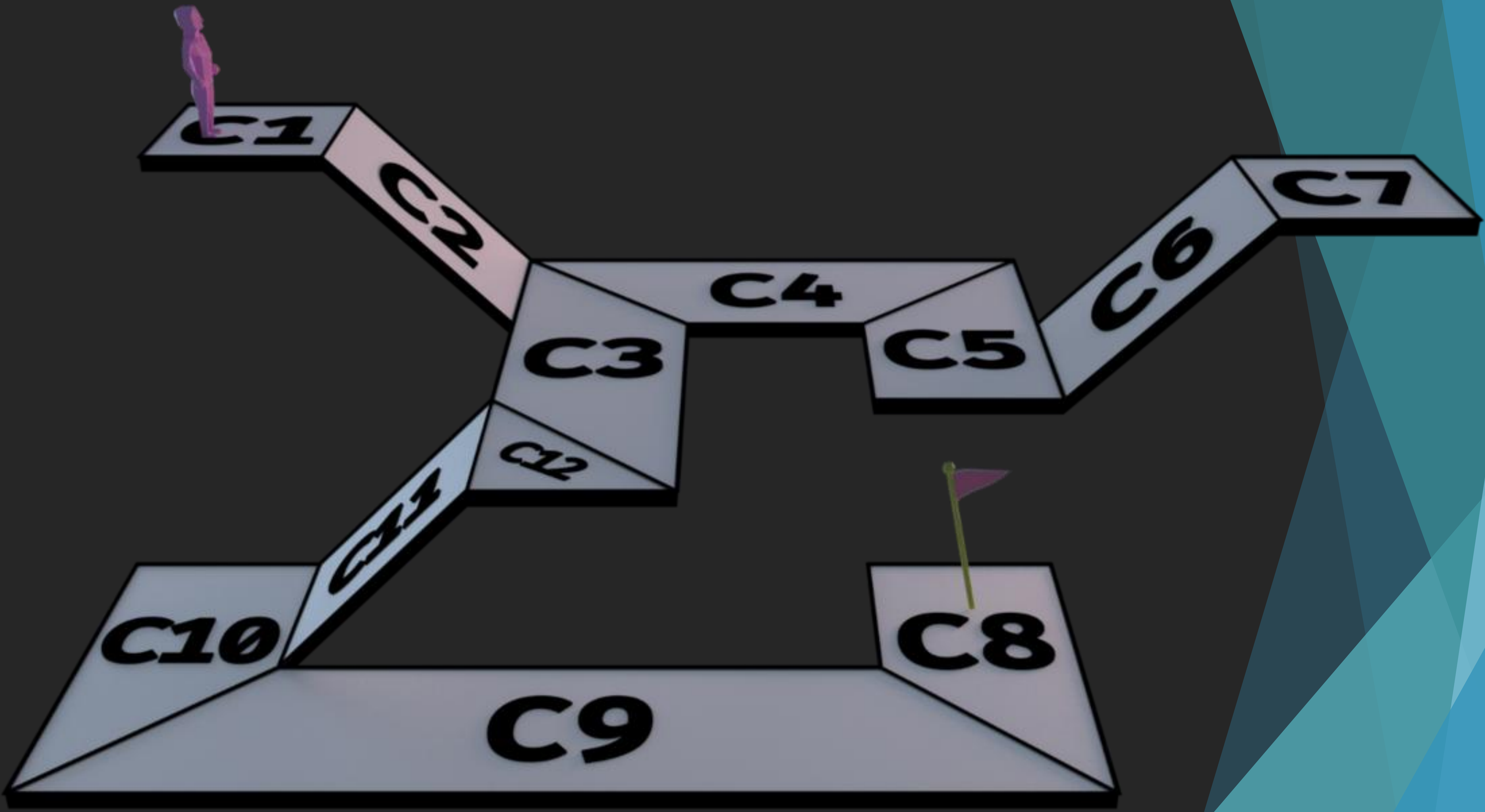
Polygon Navigation

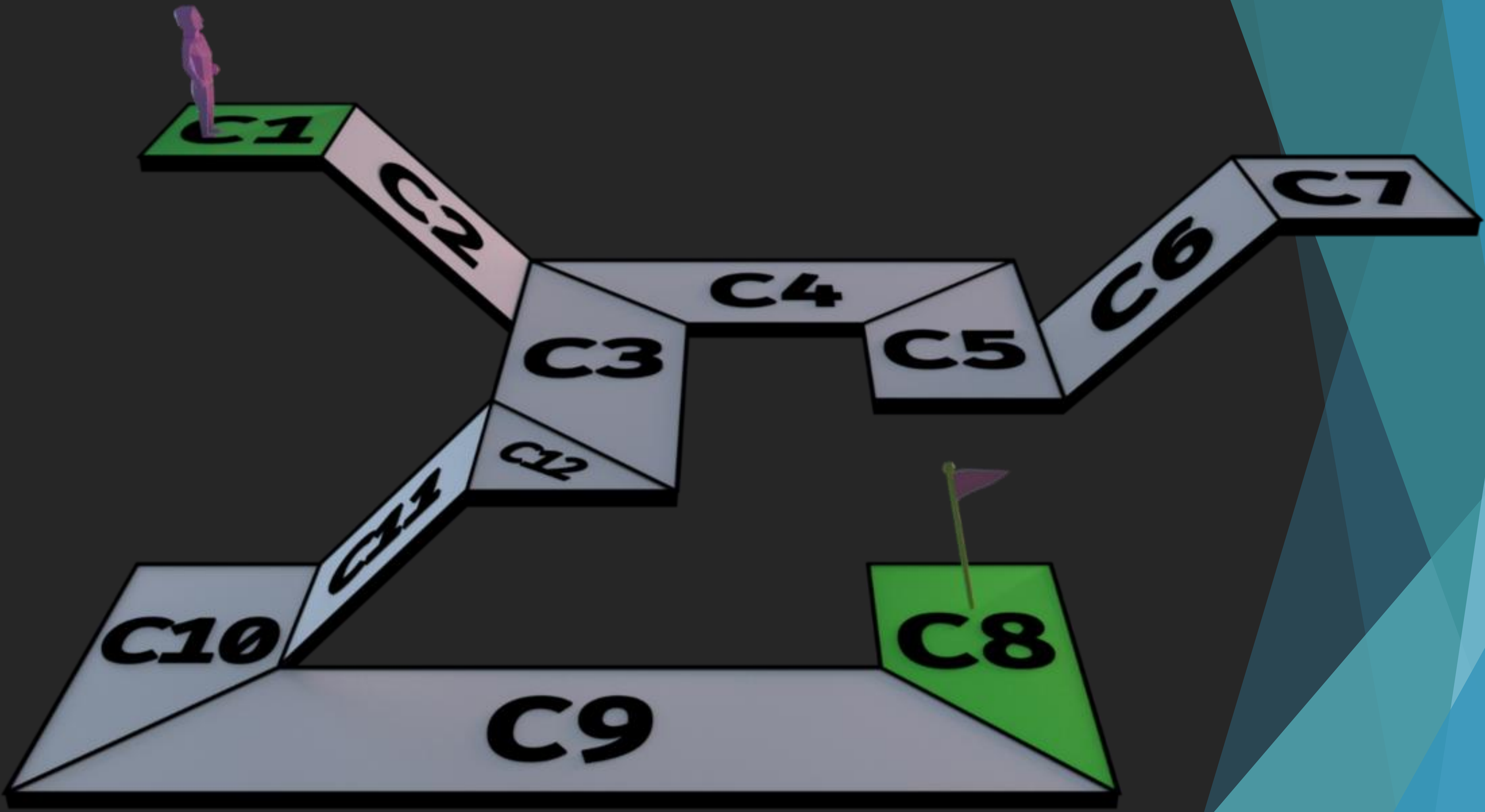


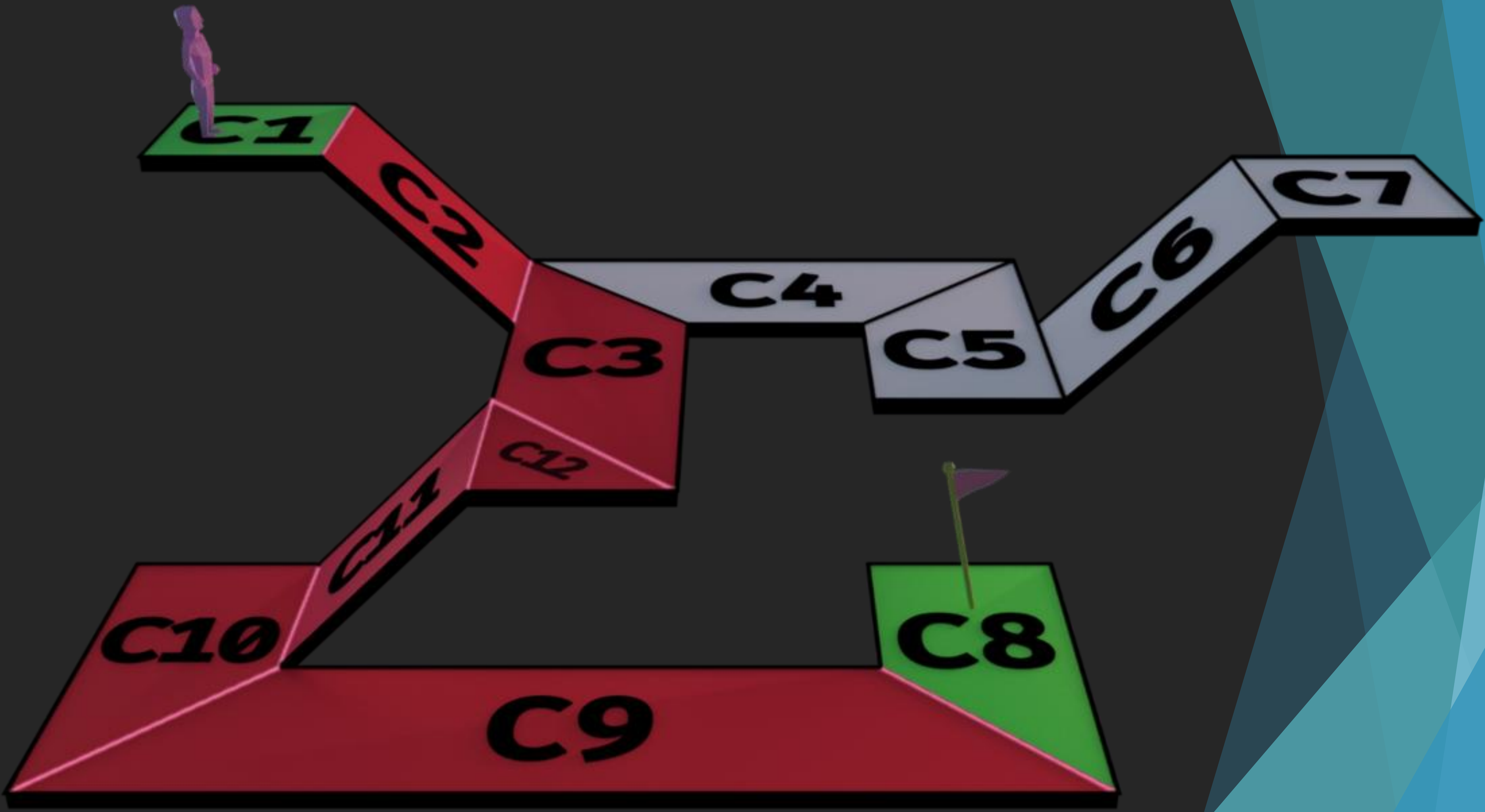
- View them as convex polygons or meshes!
- Simplifies the navigation space
    - Reduce potential cost of running A* as each polygon is the node, rather than a vast grid

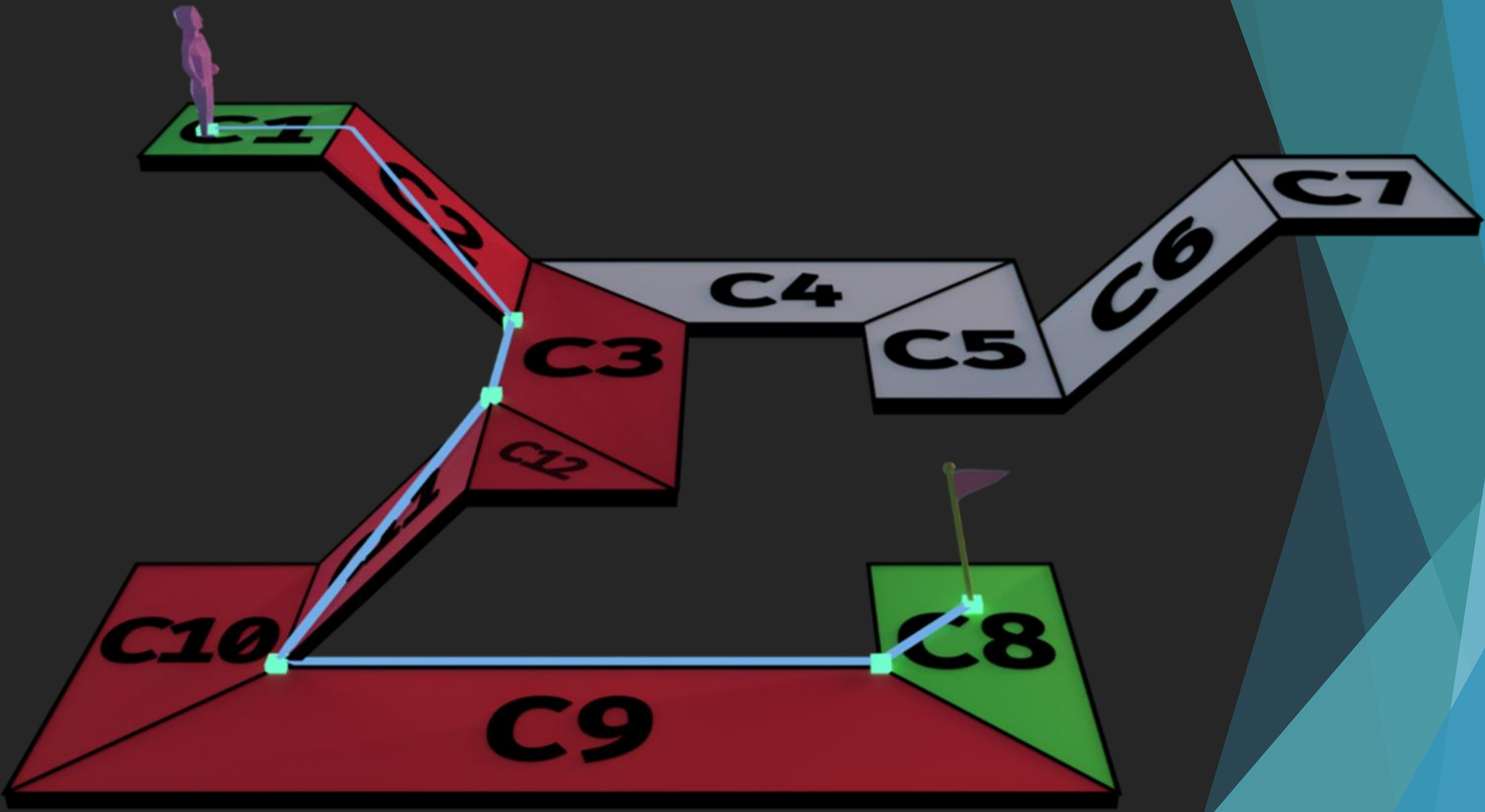
Navigation Meshes

- The most used method for navigation in modern games
- Reads physics (or visual) geometry and breaks it into convex meshes
    - Navigation calculated between meshes, simple seek movements within each mesh
- Agents have a radius?
    - “Step” the corners in, intuitively similar to corner-graphs
- Walls or obstacles?
    - Rule them out based on angle or explicitly defined obstacles


C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12


C1
C2
C3
C4
C5
C6
C7
C12
C11
C10
C9
C8


C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12

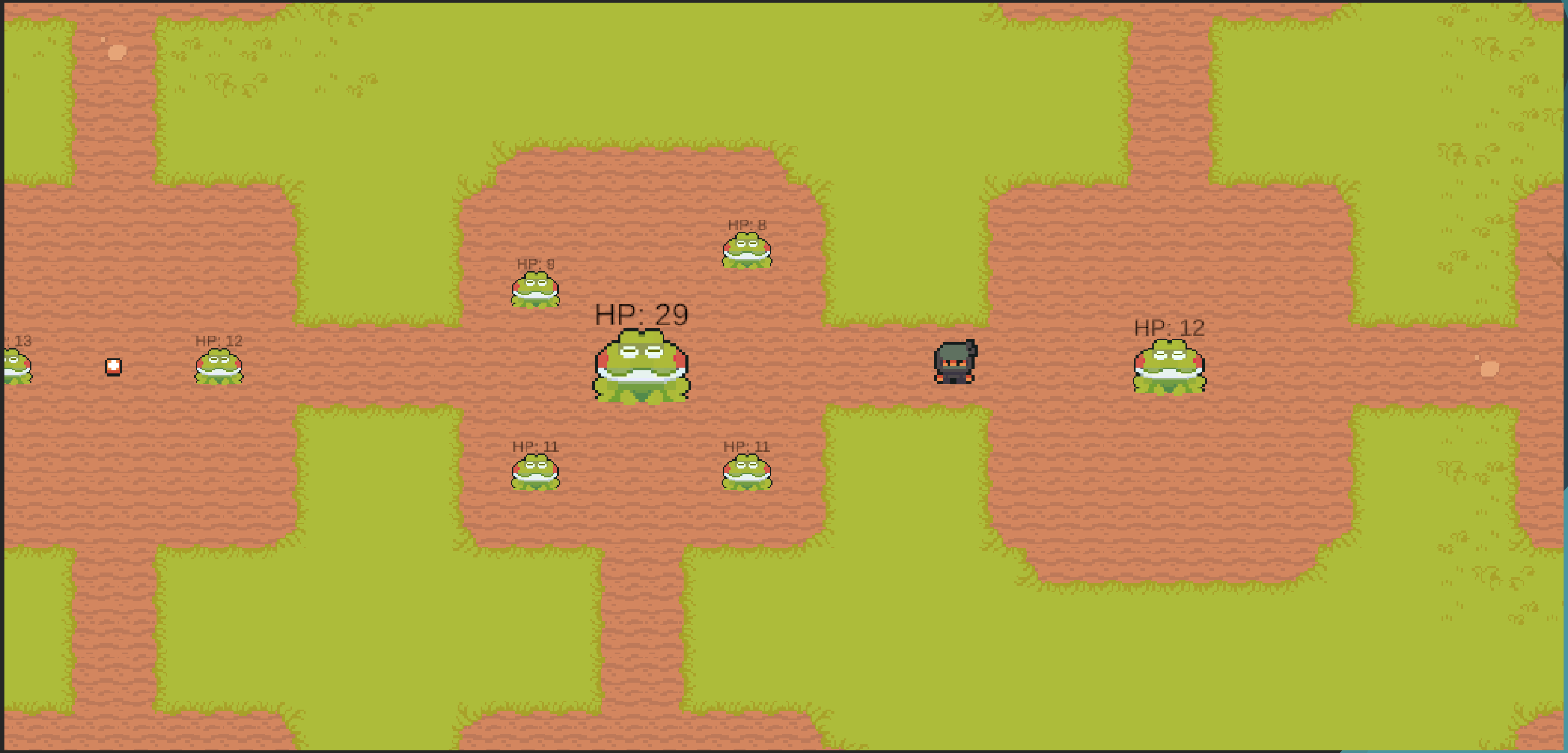

C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12

Navigation Meshes

- Should navigation meshes always be used today?
  - Game dependent!
- Sometimes, there is a clear way to place nodes simpler than using meshes

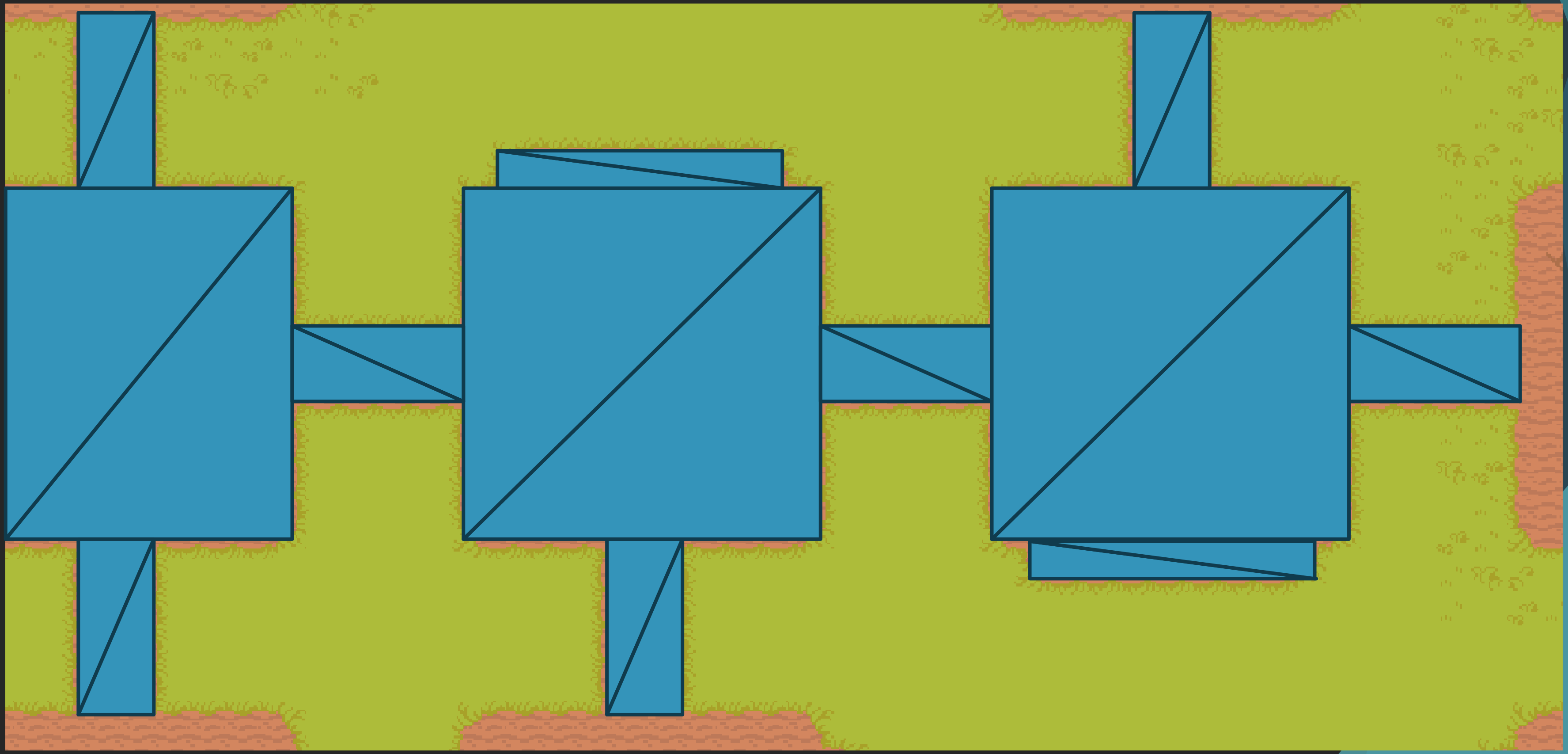
Navigation Meshes

- How might navigation meshes handle this level?



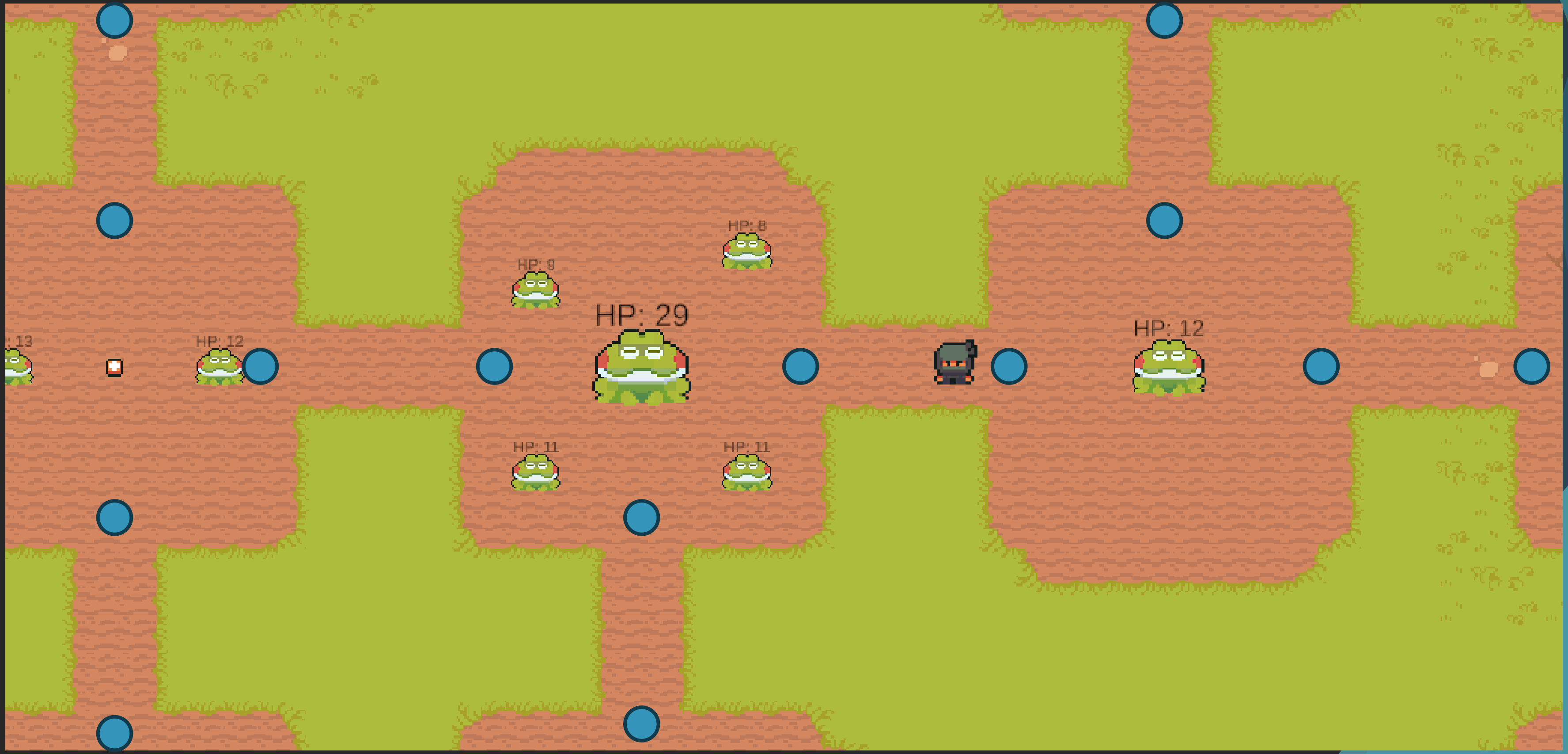
Navigation Meshes

- How might navigation meshes handle this level?



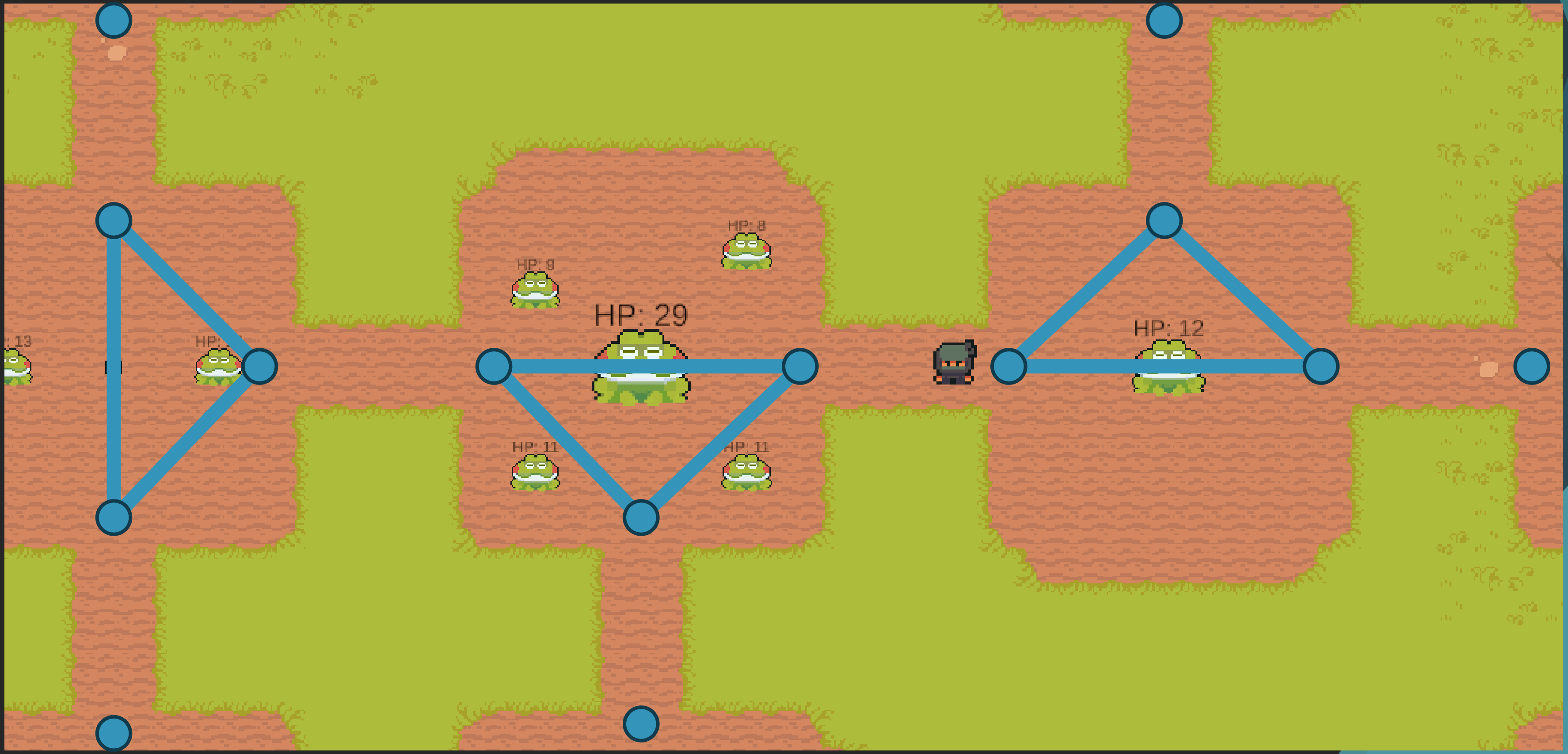
Navigation Meshes

- We could just add nodes at each entrance



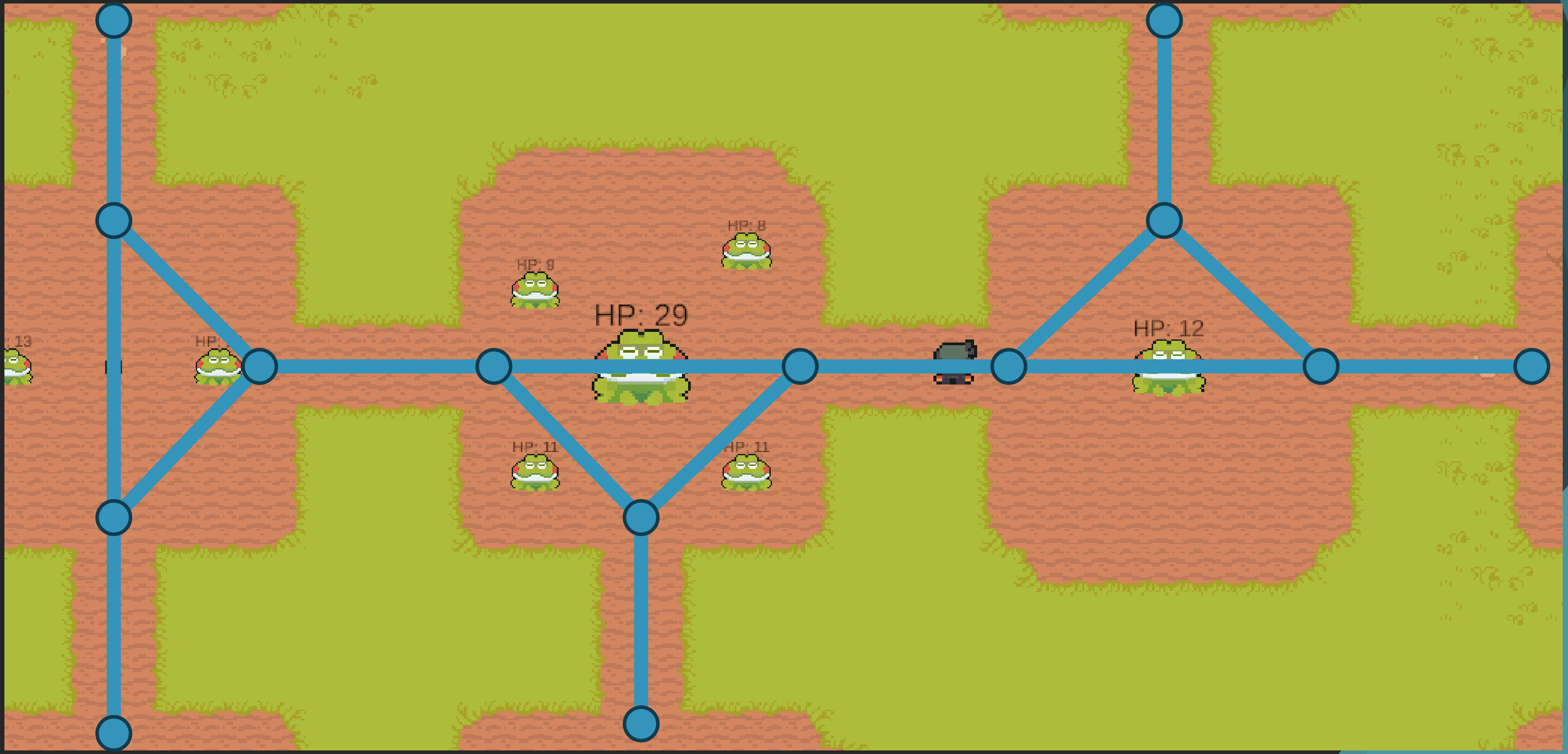
Navigation Meshes

- Attach nodes within the rooms to each other



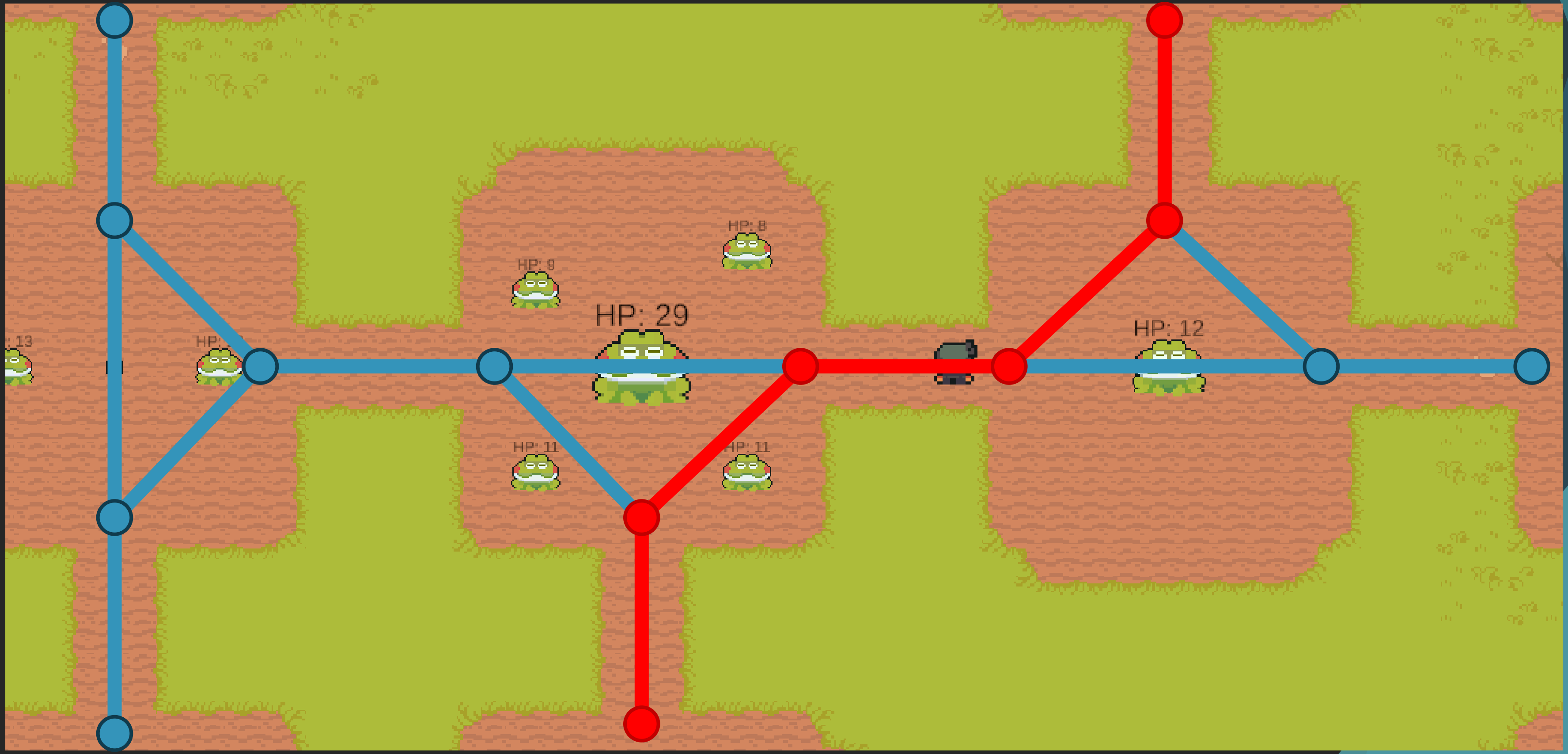
Navigation Meshes

- Attach adjoining doors together



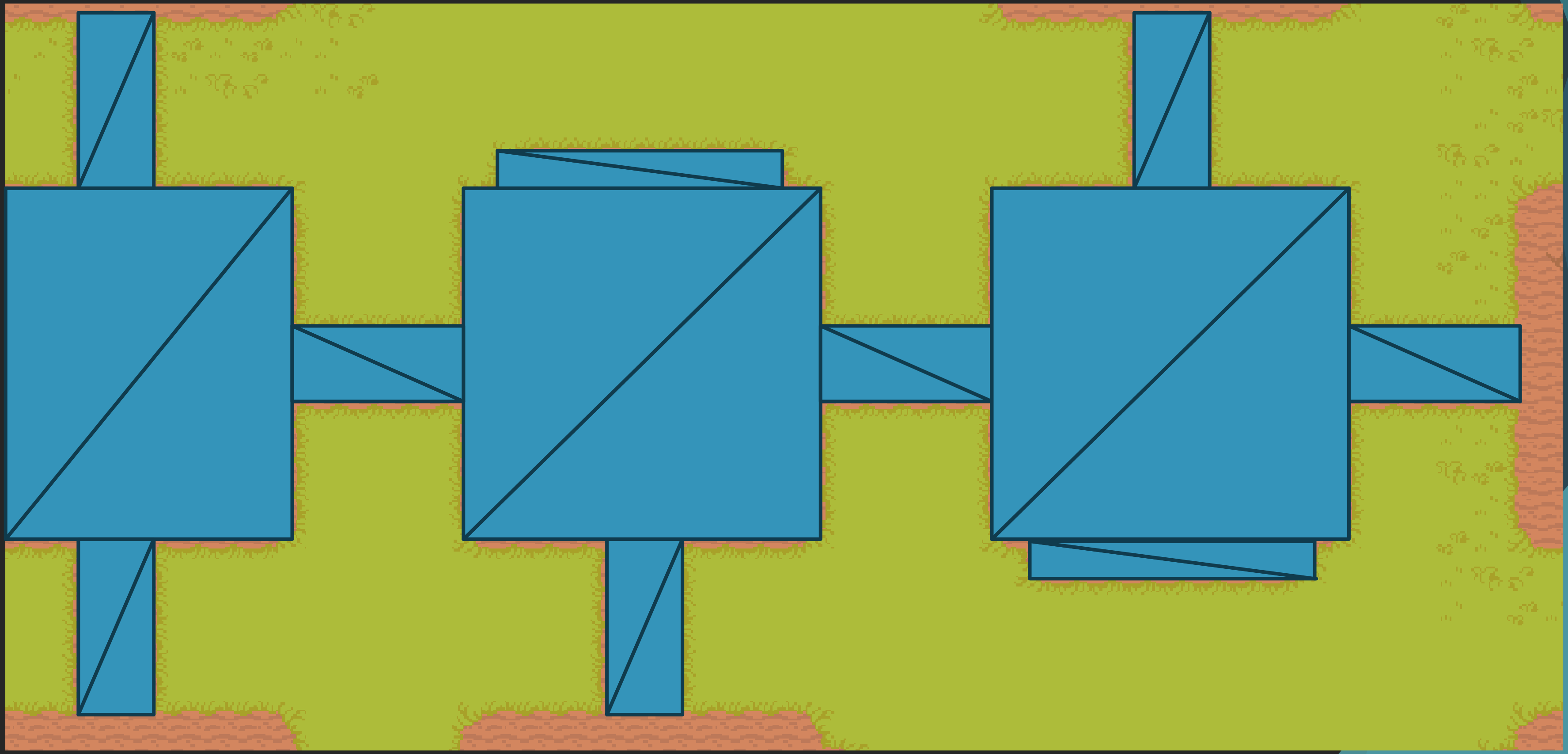
Navigation Meshes

- Now we have sufficient pathfinding with a simpler graph for A*



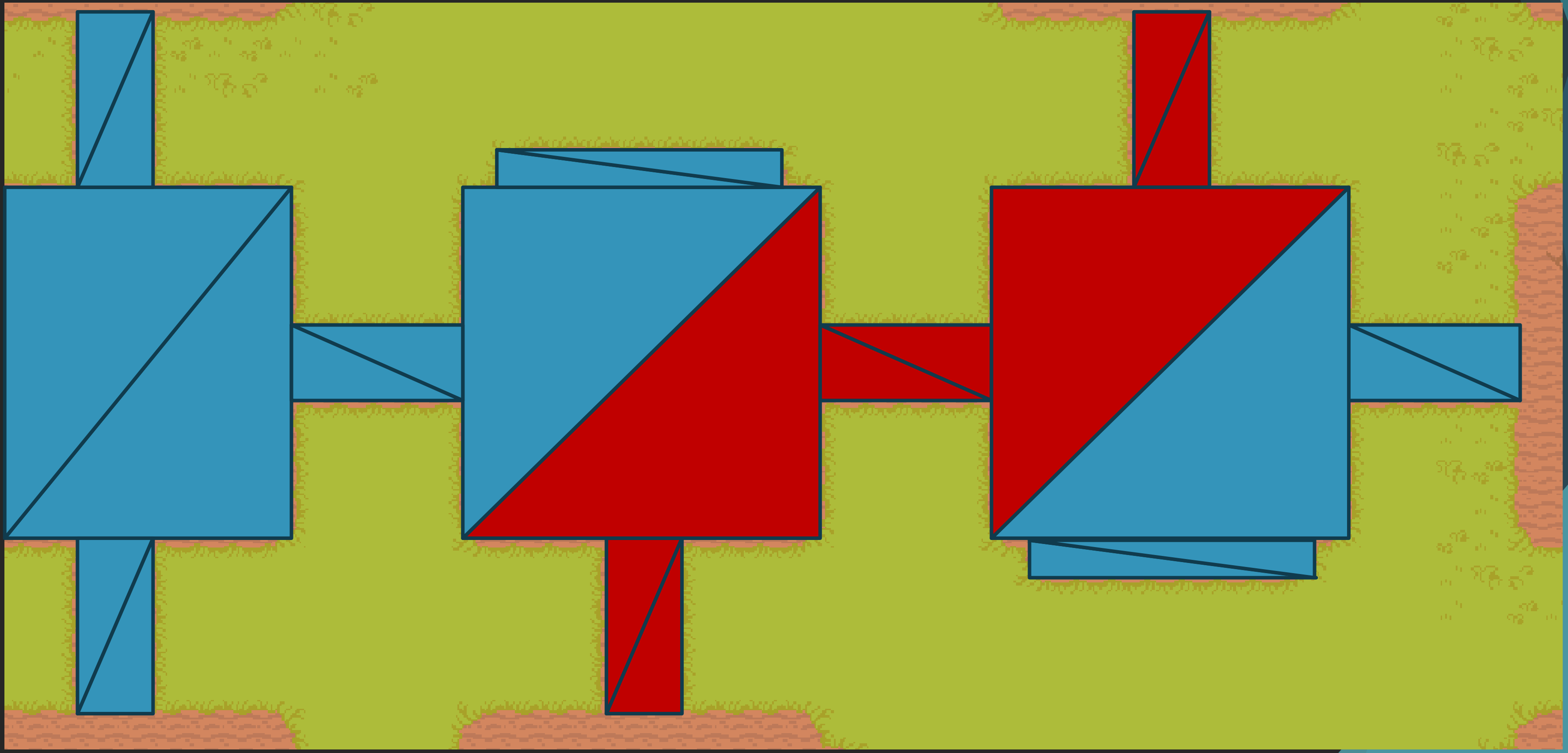
Navigation Meshes

- The same navigation with navigation meshes



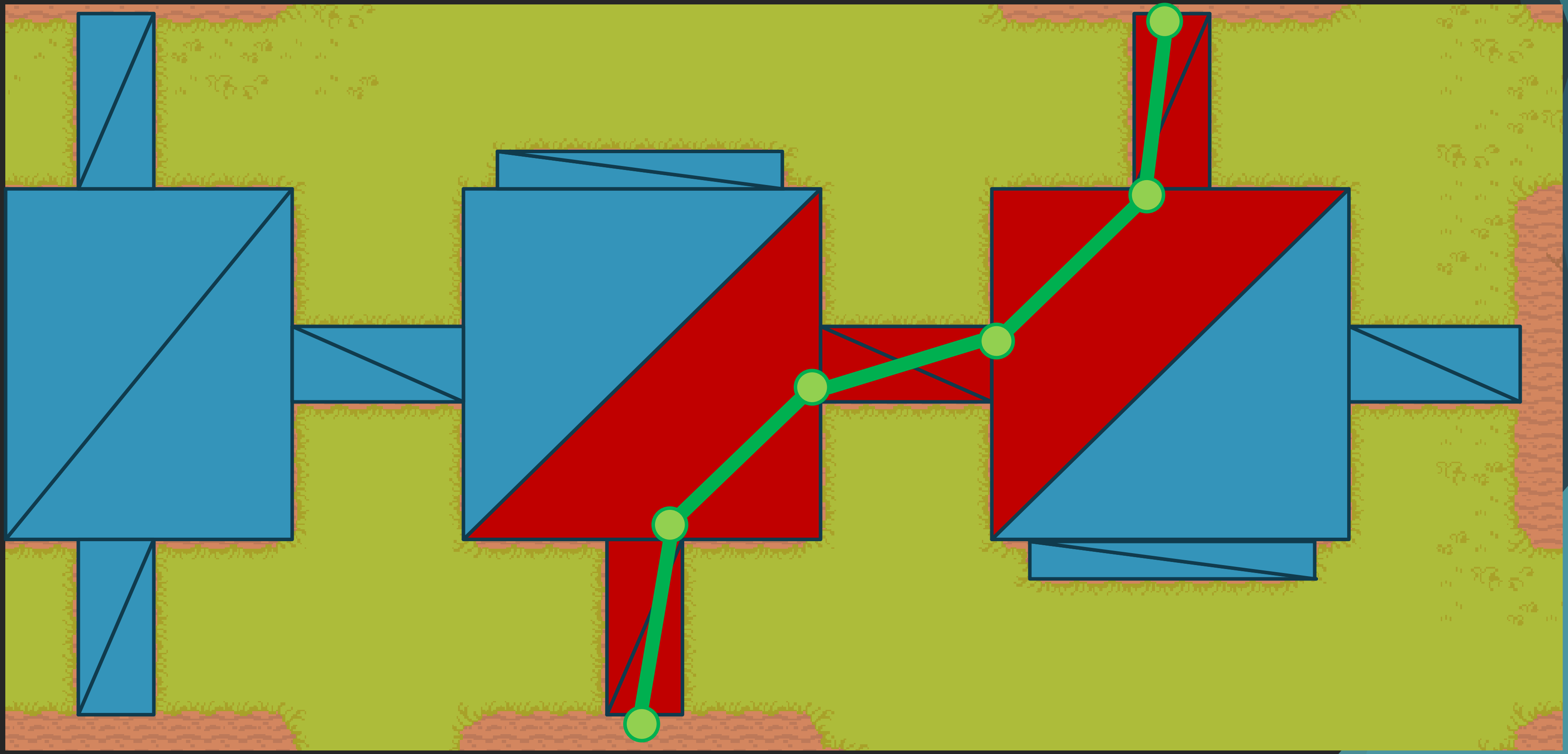
Navigation Meshes

- First find the path between meshes



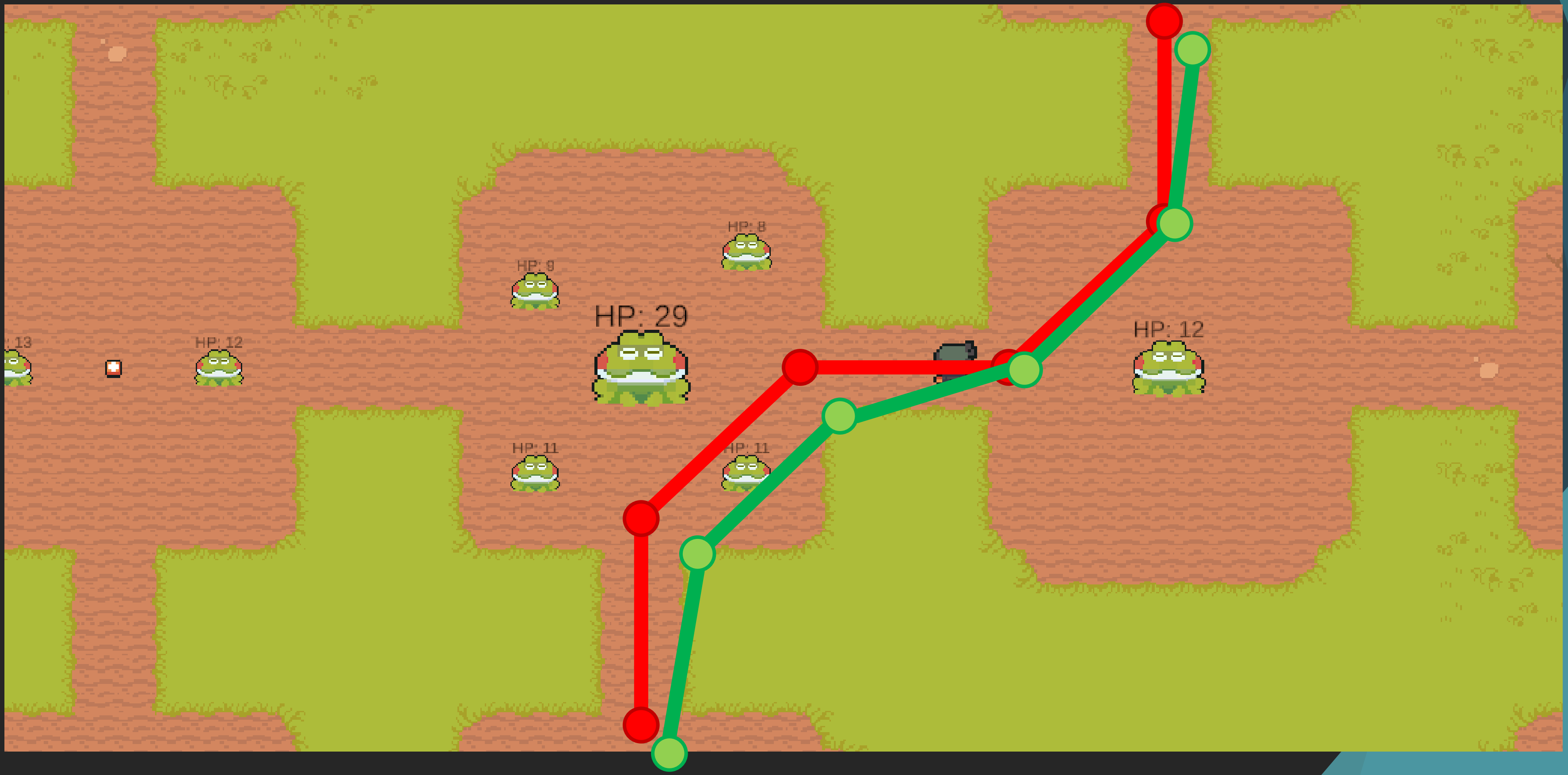
Navigation Meshes

- Then “string pull” along the corners



Navigation Meshes

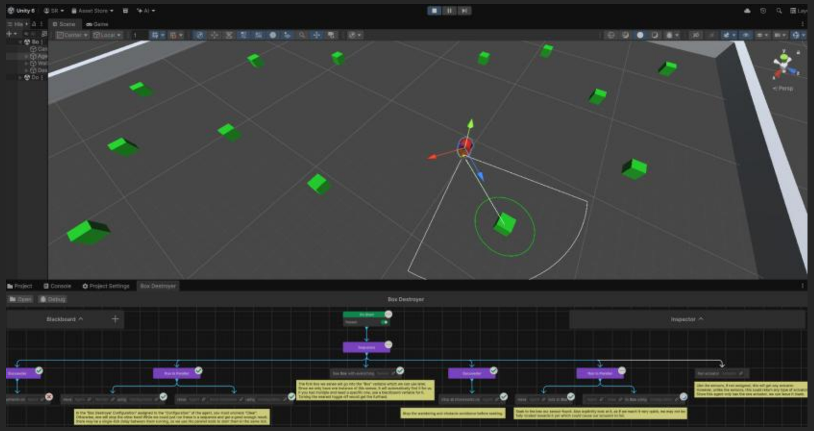
- Which path is better?

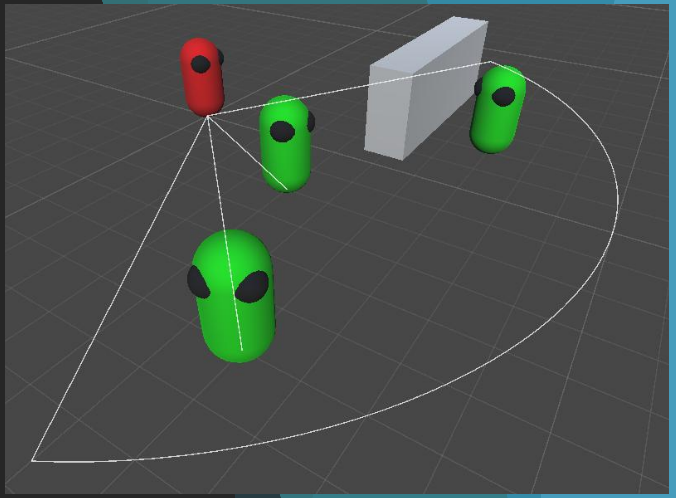


Navigation Meshes

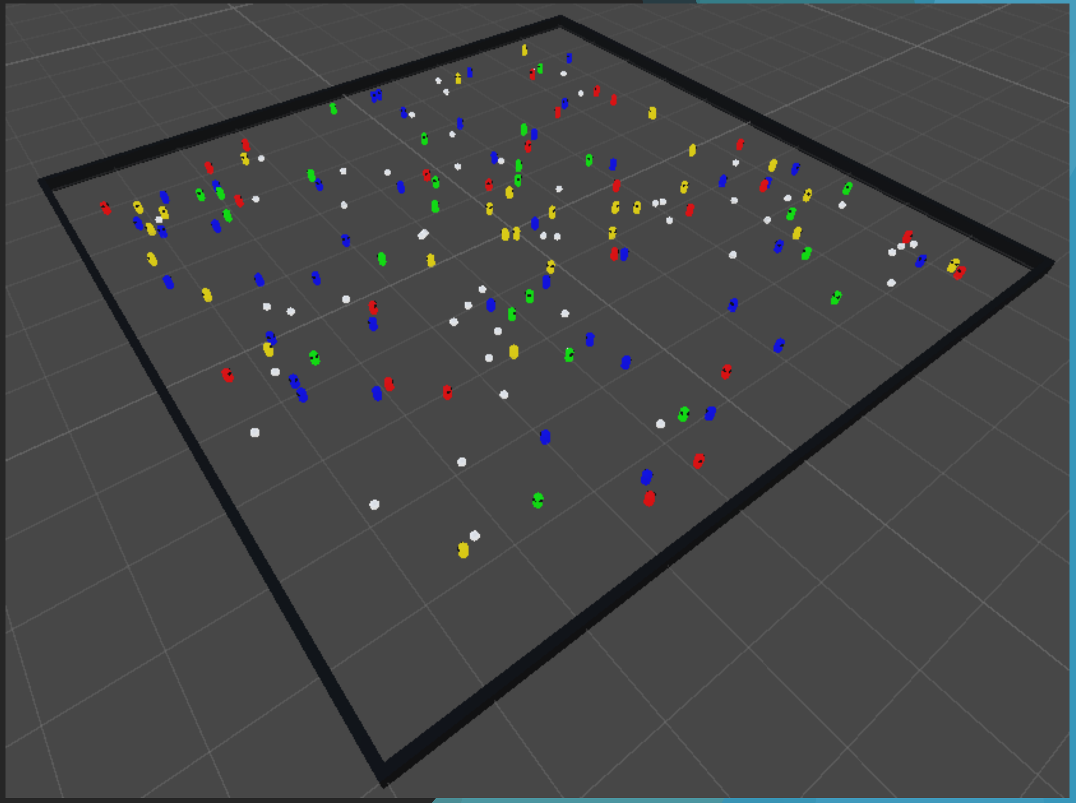
- Either is likely “good enough”
- Navigation meshes
  - “Better” in terms of a shorter path
  - No need to implement your own pathfinding logic
- Custom node placement
  - “Better” as more “human-like”?
    - Not hugging the walls → A common “issue” with navigation meshes!
      1. Create a wider agent radius to avoid corners
      2. Place invisible “obstacles” along the walls to force the agent to walk in the center

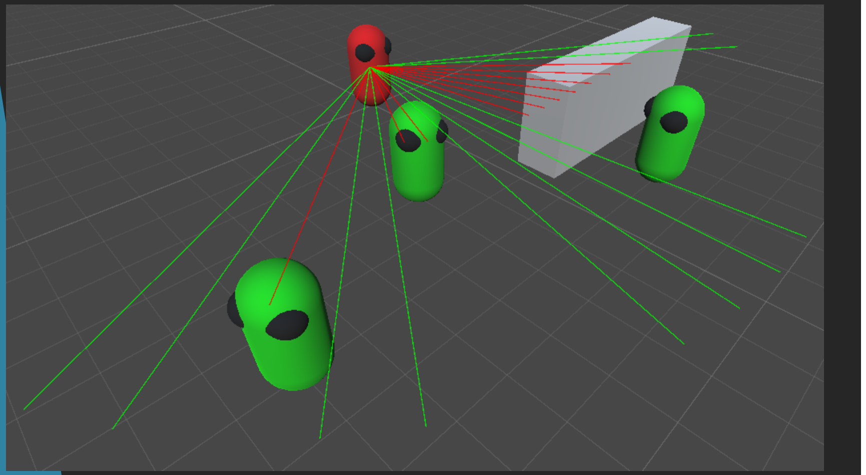
Takeaways





- Whether you want to make games or just enjoy them, hope you've learned something about what goes on inside of them
- University of Windsor
  - COMP-3770 and COMP-4770
- Kaiju Agents: agents.kaijusolutions.ca





Thank You for Listening!

StevenRice.ca

Contact@StevenRice.ca

StevenRice99

StevenRice99