
# COMP 476

# Advanced Game Development

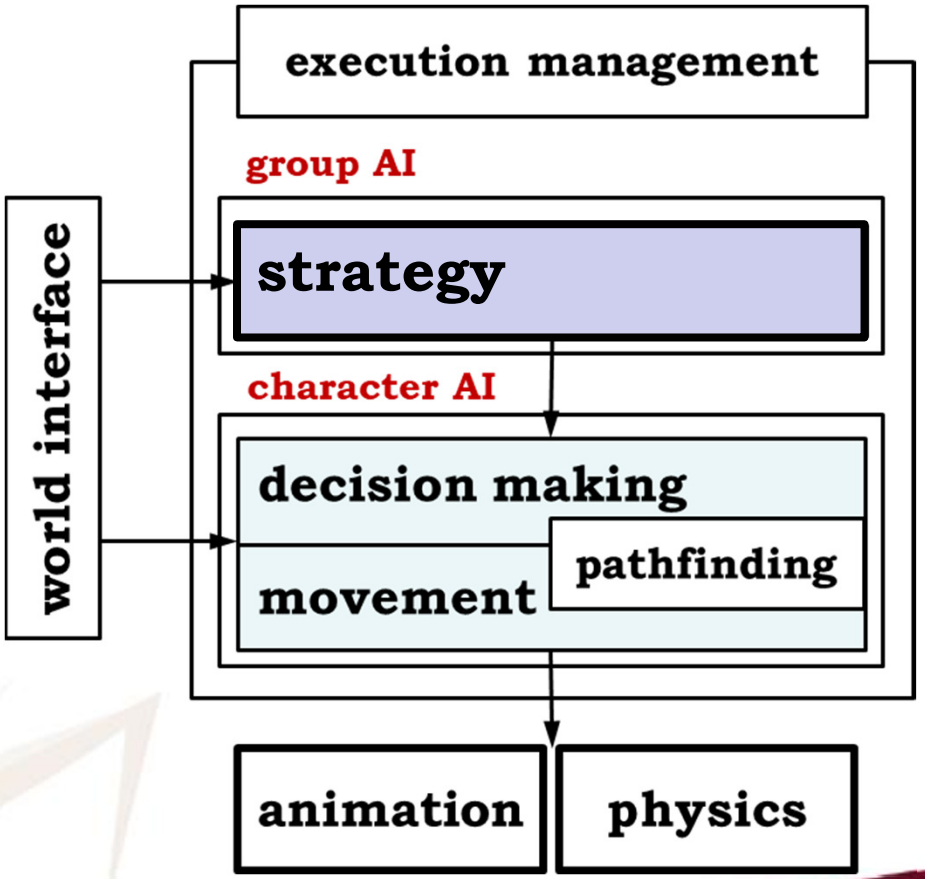
## Session 5
## Tactical AI

**Tactical Pathfinding (Reading: AI for G, Millington § 6.1-6.4)**

# Lecture Overview

- Waypoint Tactics
- Tactical Analysis
- Tactical Pathfinding
- Coordinated Action

# Tactical & Strategic AI in Millington's Model



# Waypoint Tactics

- **A waypoint – A *single position in the game level* (known as "nodes", "representative points" *used for pathfinding*)**
- **To use waypoints tactically ➔ *need to add more data to the nodes* (not just location info)**
- **We'll look at some examples of use of waypoints to represent positions in the level with *certain tactical features***
- **Normally the level designer has some say in this**
- **Can also *deduce first the tactical information* and *then the position* automatically.**

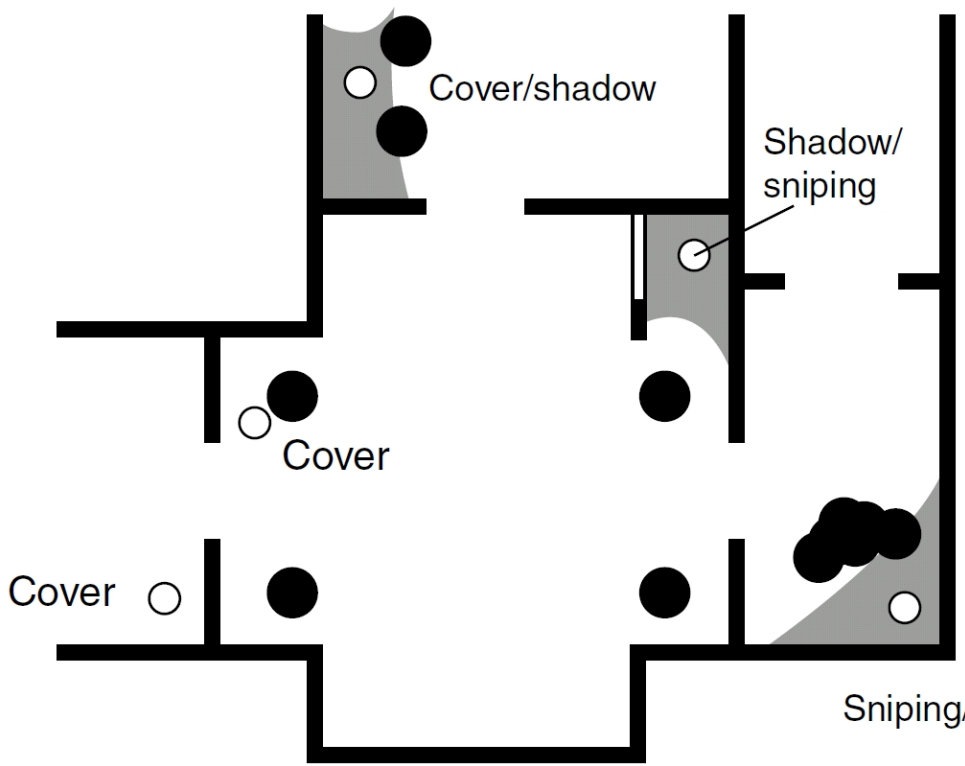
# Tactical Locations

- **Waypoints used for tactical purposes are sometimes called "rally points"**
- **E.g.,**
  - **Early use: to mark *a fixed safe location* for character *to retreat if losing fight* (defensive)**
  - **More commonly: *cover points when engaging enemy* (offensive)**
  - **To mark a pre-determined hiding spot that *can ambush or snipe incoming enemy* (offensive)**
  - **To *move secretly in shadow areas without being detected* (stealth)**
  - **Many more!**

# Tactical Points

## not the best pathfinding points

- Although common to combine two sets of waypoints (one for tactical, one for pathfinding), not efficient nor flexible



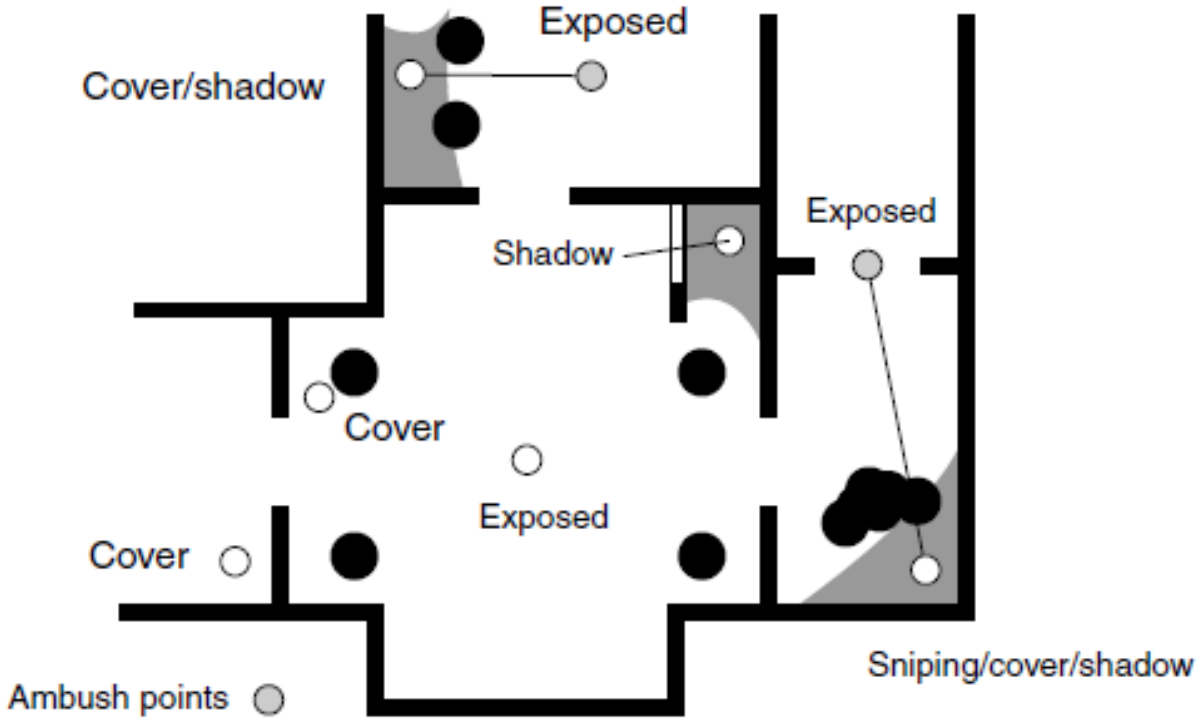

- E.g. Cover and sniping waypoint nodes are not always useful for pathfinding! May result in unrealistic movements within a level

# Primitive & Compound Tactics

- Most games have a set of pre-defined tactical qualities (*e.g.* sniping, shadow, cover, *etc.*)
- Shadow and cover are primitive defined tactics
- A point can have both defensive and offensive tactical features
- Combinations of primitive tactics result in locations with compound tactical qualities
  - *e.g.* Sniper Locations – Points that have a combination of both cover points and high-visibility points
- For an ambush, we could look for exposed locations with good hiding places nearby

# Primitive & Compound Tactics

- **For this e.g., how is an ambush point constructed from primitive tactical locations?**

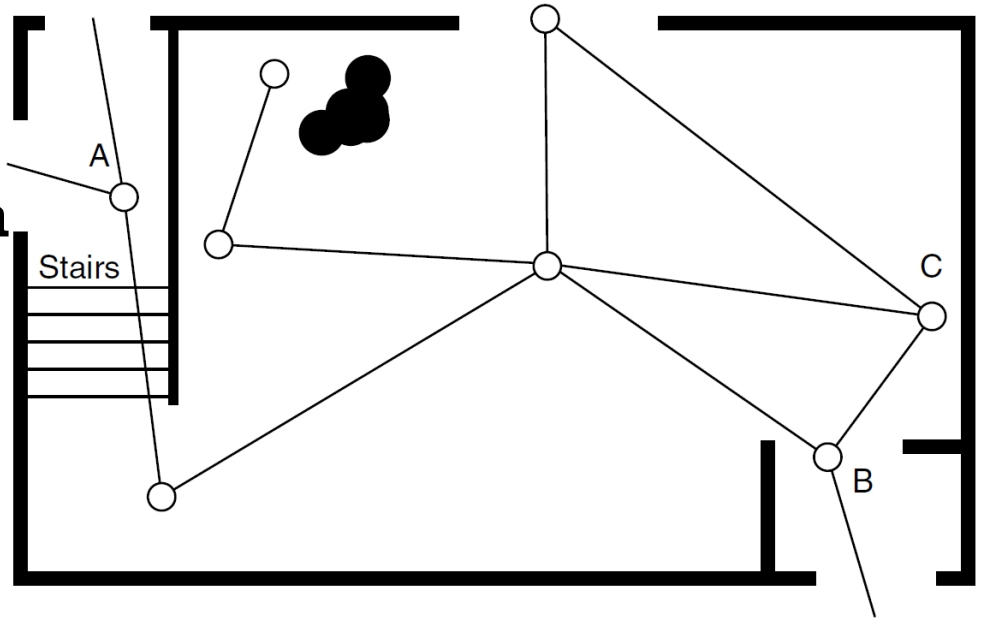


# Primitive & Compound Tactics

- We can take advantage of these compound tactics by storing only the primitive qualities.
- In the previous example, we stored three tactical qualities: cover, shadow, and exposure. From these we could calculate the best places to lay or avoid an ambush.
- Advantage: we can support a huge number of different tactics; use less memory
- Disadvantage: we lose in speed
  - But the character may have several frames to make tactical decisions.

# More Compound Tactics
# 1. Waypoint Graphs

- **Waypoints can be connected to form waypoint graphs (similar to pathfinding graphs) when the waypoints defined are not isolated/separated**
- **Topological Analysis: Where is the best spot for a hit-and-run move?**
- **What are some problems using waypoint graphs?**



# 2. Continuous Compound Tactics

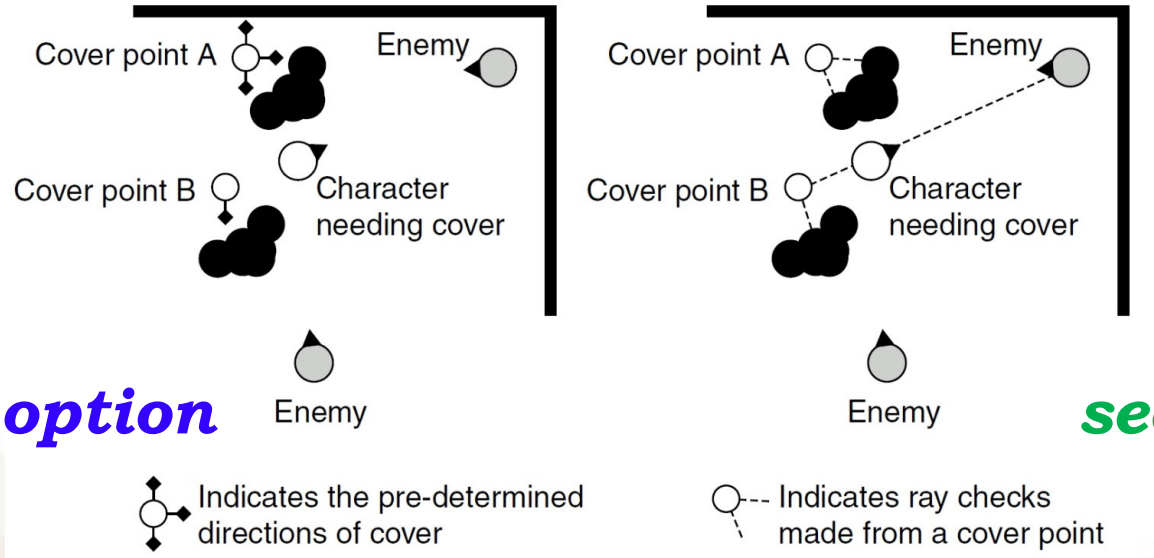
- Marking locations with numerical values (able to use fuzzy logic and probabilities) instead of Boolean values
- E.g. A waypoint will have a value for cover feature (0.9) and visibility feature (0.7)
- In choosing between a few cover points to go to, choose one that has better/higher value
- Using fuzzy logic rules can allow us to combine these values, E.g.
  - Sniper (value) = cover (value) & visibility (value)
  - Sniper = MIN(0.7, 0.9) = 0.7

# Context Sensitivity

- **Problem: The tactical properties of a location are almost always sensitive to the actions of the character or the current state of the game.**
  - **E.g., hiding behind a protruding rock is of no use if the enemy is behind you.**
- **There are two options for implementing *context sensitivity*.**
- **In the first option, we could store multiple values for each node. A cover waypoint, for example, might have four different directions. We call these four directions the states of the waypoint.**
- **We could use any number of different states.**

# Context Sensitivity

- In the second option, we use only one state per waypoint, as before.
- We add an extra step to check (e.g., against the game state) if it is appropriate.
  - E.g., at a cover point we might check for line of sight with our enemies.




*first option*

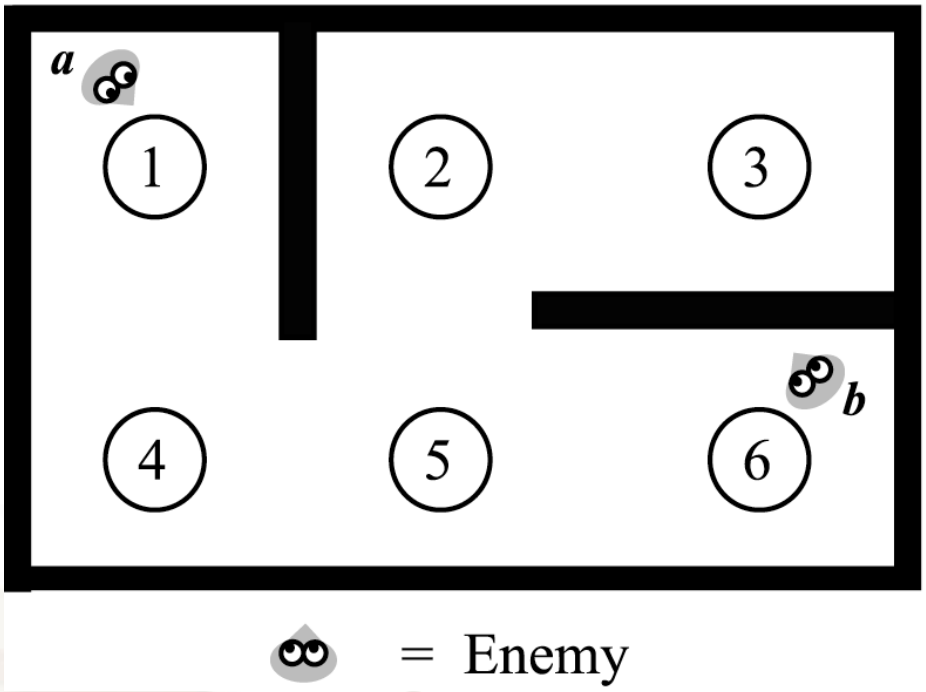
*second option*

*Tactical Analysis of Pathfinding Graphs*

# *Safe Attack Positions*

# Tactical Analysis

- Level designers place pathfinding graphs in the environment for navigation
- The graph contains node connectivity information for a level




- These nodes can also be evaluated for their visibility
- Information can be used to make tactical decisions:
  - to use some of these nodes as waypoints

# Node Analysis

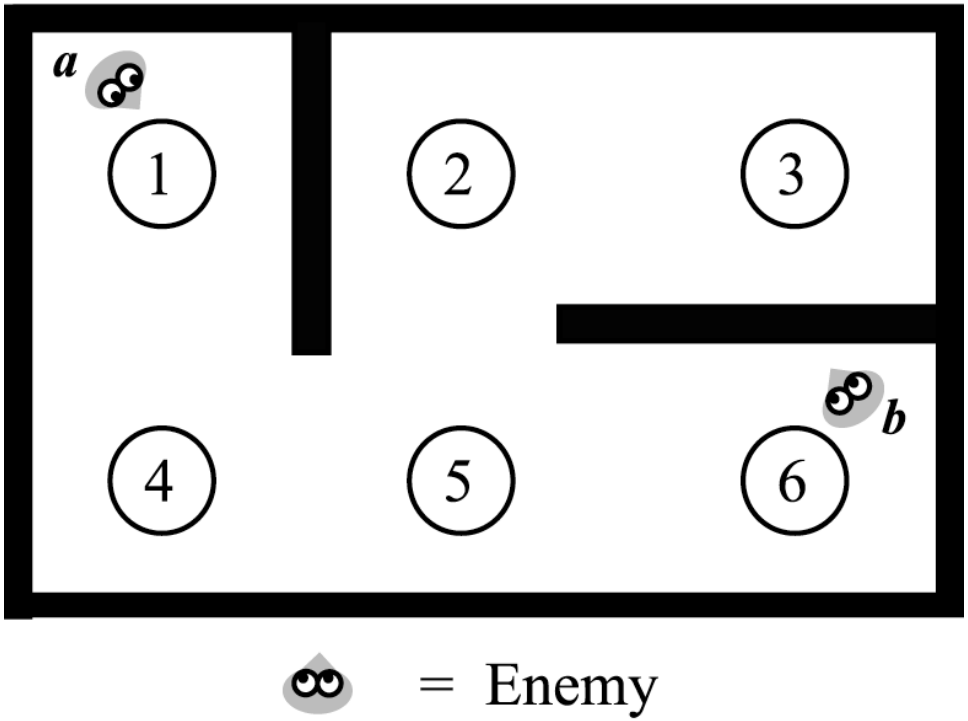
- **Constraints:**
  - **Limited CPU time**
  - **Decisions must be made quickly** **(as few CPU cycles as possible)**
  - **Data must be stored efficiently**

- **Store visibility data in a "bit-string" class**

$V_a$ = **visibility from node "a"**

# Node Analysis

| | | Node 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| (a) | $V_1$ = | 1 | 0 | 0 | 1 | 0 | 0 |
| | $V_2$ = | 0 | 1 | 1 | 0 | 1 | 0 |
| | $V_3$ = | 0 | 1 | 1 | 0 | 0 | 0 |
| | $V_4$ = | 1 | 0 | 0 | 1 | 1 | 1 |
| | $V_5$ = | 0 | 1 | 0 | 1 | 1 | 1 |
| (b) | $V_6$ = | 0 | 0 | 0 | 1 | 1 | 1 |



# Node Analysis

- **Danger Nodes**
  - **Determined by "OR"ing the visibility of *k* enemies' nearest nodes**

$$V = \bigcup_{j=0}^{k} V_j$$

- **Safe Nodes**
  - **Is the inverse of $V$:**

$$\bar{V}$$

# Node Analysis

DANGER NODES:

$V = V_a \cup V_b = 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1$

Nodes 1, 4, 5 and 6 are dangerous

SAFE NODES:

$\overline{V} = 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0$

Nodes 2 and 3 are safe

| | | Node 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| (a) | $V_1 =$ | 1 | 0 | 0 | 1 | 0 | 0 |
| | $V_2 =$ | 0 | 1 | 1 | 0 | 1 | 0 |
| | $V_3 =$ | 0 | 1 | 1 | 0 | 0 | 0 |
| | $V_4 =$ | 1 | 0 | 0 | 1 | 1 | 1 |
| | $V_5 =$ | 0 | 1 | 0 | 1 | 1 | 1 |
| (b) | $V_6 =$ | 0 | 0 | 0 | 1 | 1 | 1 |

# Node Analysis

DANGER NODES:

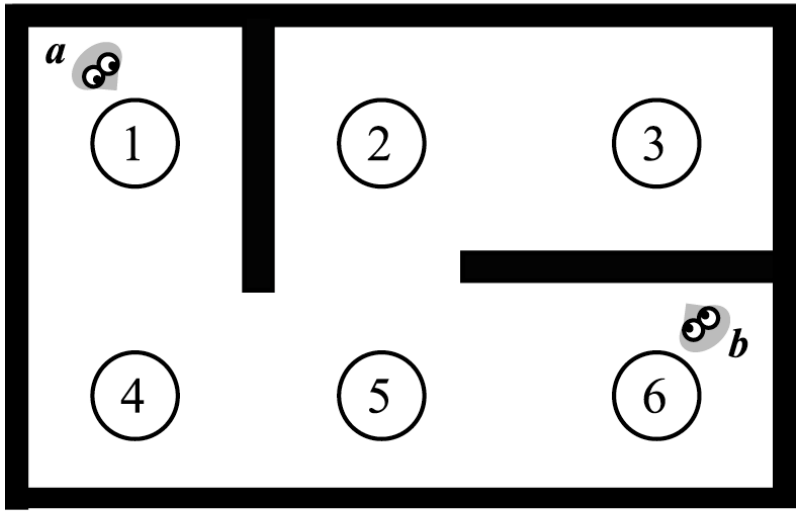
$V = V_a \cup V_b = 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1$

Nodes 1, 4, 5 and 6 are dangerous

SAFE NODES:

$\overline{V} = 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0$

Nodes 2 and 3 are safe

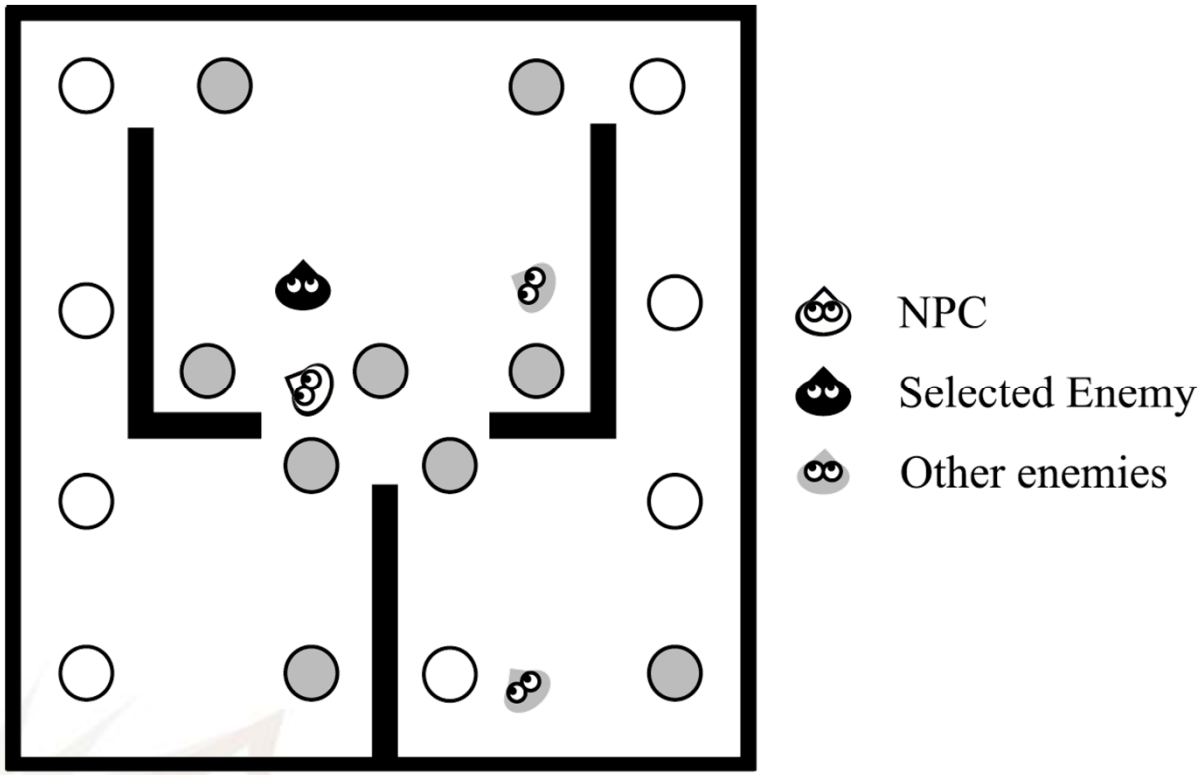



= Enemy

# 1. Finding a Safe Attack Position Analysis

- **While attacking a selected enemy, an NPC *shouldn't expose itself to other enemies***
- **A good attack position will:**
  - **Provide *line-of-site (LOS) to the selected enemy***
  - **Provide *cover from all other enemies***
- **Call selected enemy "a"**
- **To find such locations, first find all nodes, $V_a$, which have LOS to the selected enemy**

# 1. Finding a Safe Attack Position Analysis

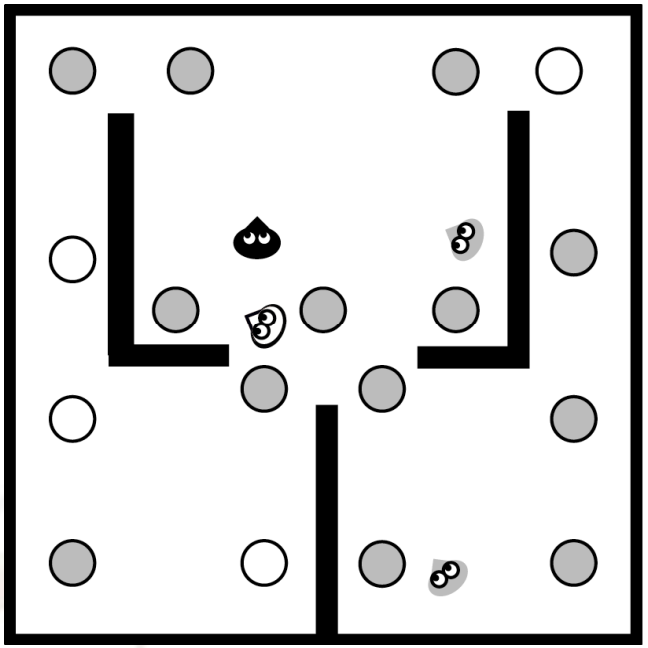


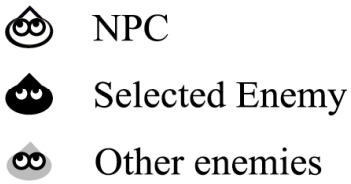

**Nodes visible to selected enemy, $V_a$**

# 1. Finding a Safe Attack Position Analysis

- **Next determine the set of nodes that are *visible to all other enemies***

$$V_{\bar{a}} = \bigcup_{j=0}^{k} V_j, j \neq a$$

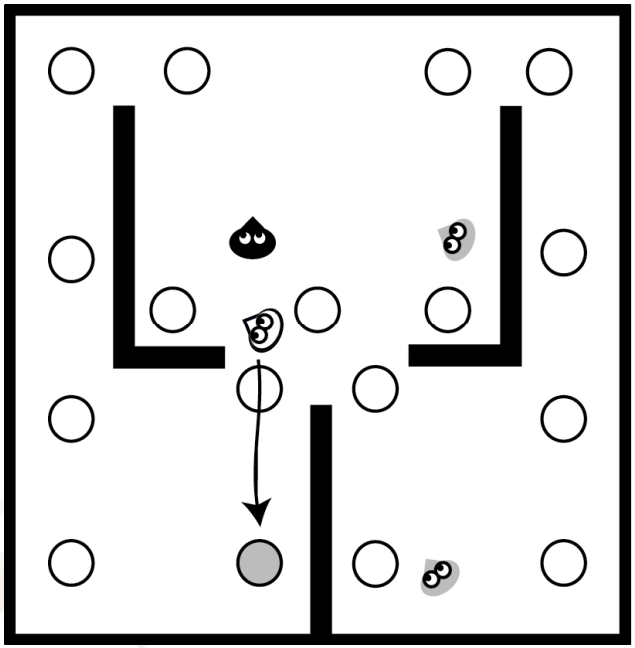





**Nodes visible to other enemies**

# 1. Finding a Safe Attack Position Analysis

- **The set of good attack positions is the *set of nodes with LOS to enemy* intersected with the *inverse of the set of nodes with LOS to all other enemies***

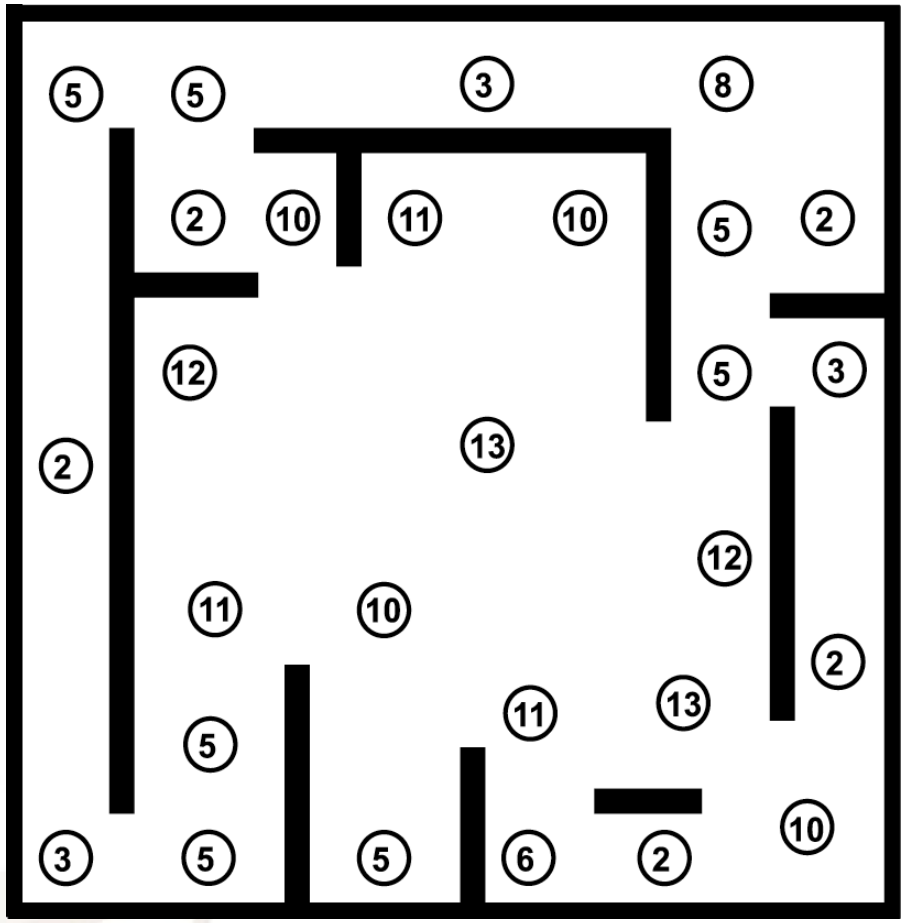



$$V_a' = V_a \cap \overline{V_{\bar{a}}}$$

# 2. Wait-In Attack Position

- **Unless cheating is employed, *NPCs don't have full knowledge of the world*.**
- ***May not know where all their enemies are located***
- **Instead, find a good location to wait in for attack**
  - **Not all positions are created equal**
- **To find a good set up position:**
  - **Establish the exposure of all nodes in a map**
  - **Process can be done off line, before game is even started**

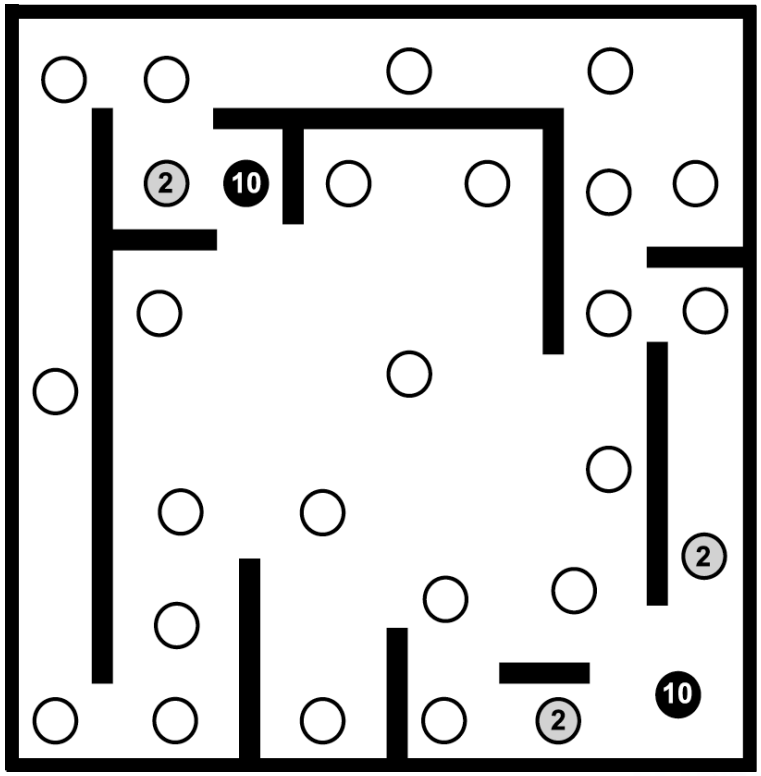
# Static Node Evaluation




**Each node evaluated for visibility (*i.e.*, number of other nodes visible to a node)**

# Static Node Evaluation

- **A good location to wait in for attack is one which:**
  - **Has high exposure (visibility)**
    - **Easy to locate enemies**
    - **Easy to establish LOS to attack and enemy**
  - **Has areas of low exposure nearby**
    - **Can hide easily**
    - **Can run for cover easily**



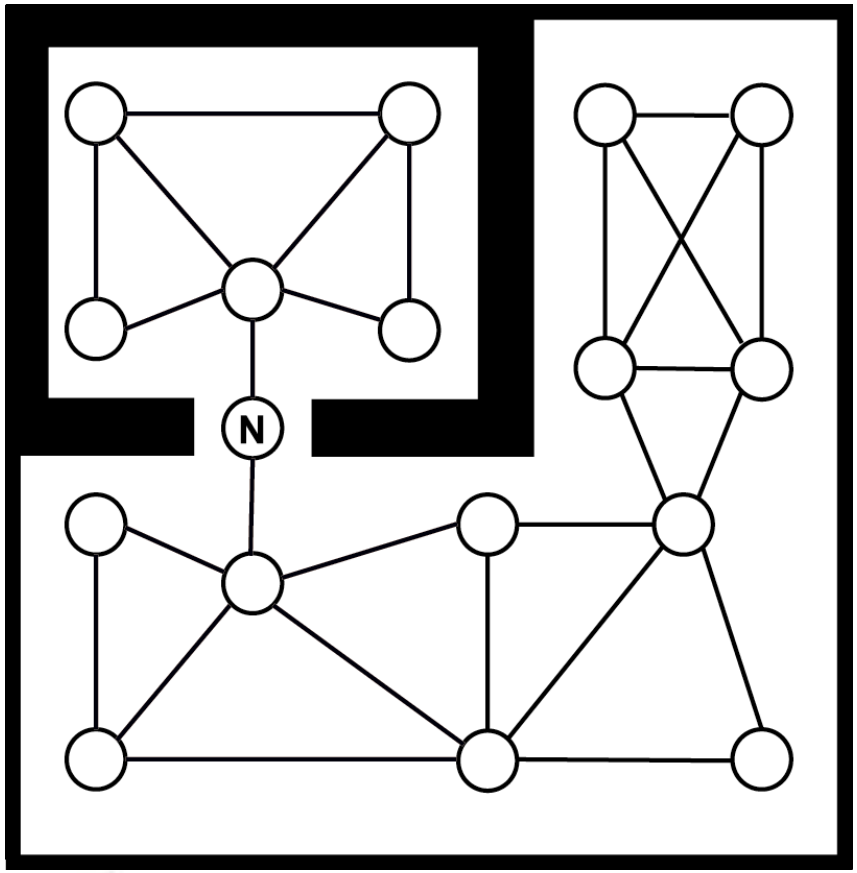
# 3. Pinch Points

- **Observation of human players reveals that experienced players anticipate the actions of their opponents [Laird 2000]**
  - **For example, if an enemy enters a room with only a single exit an experienced player will wait just outside the exit setting up an ambush**
- **Such "pinch points" can be pre-calculated by analyzing the (pathfinding) node graph**

# 3. Pinch Points

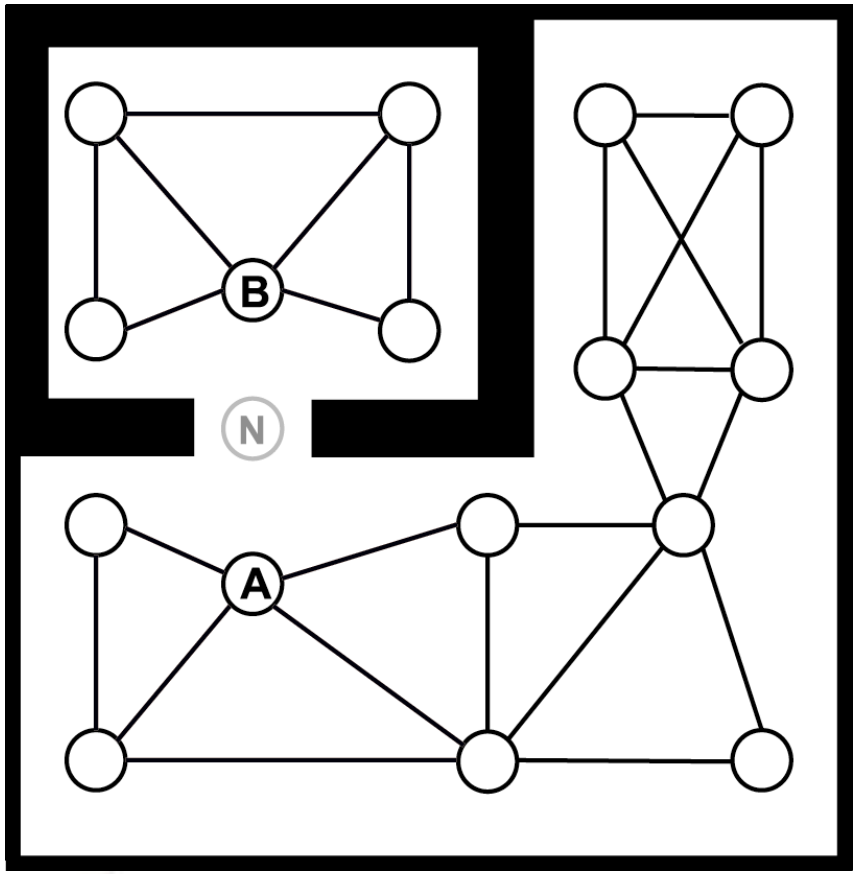
- To find pinch points:
  - For each node, N in the node graph with only two neighbors:
    - Temporarily eliminate node, N, from the graph, call its neighbors as A & B.
    - If both A & B are connected to large regions, N is not a pinch point, try another N.
    - Attempt to find a path between A & B (not through N)
    - If path exists, N is not a pinch point, try another N.
    - Call the node connected to the larger region, O (for outside).
    - Call the node connected to the smaller region, I (for inside).
- Let's do that again step-by-step:

# 3. Pinch Points



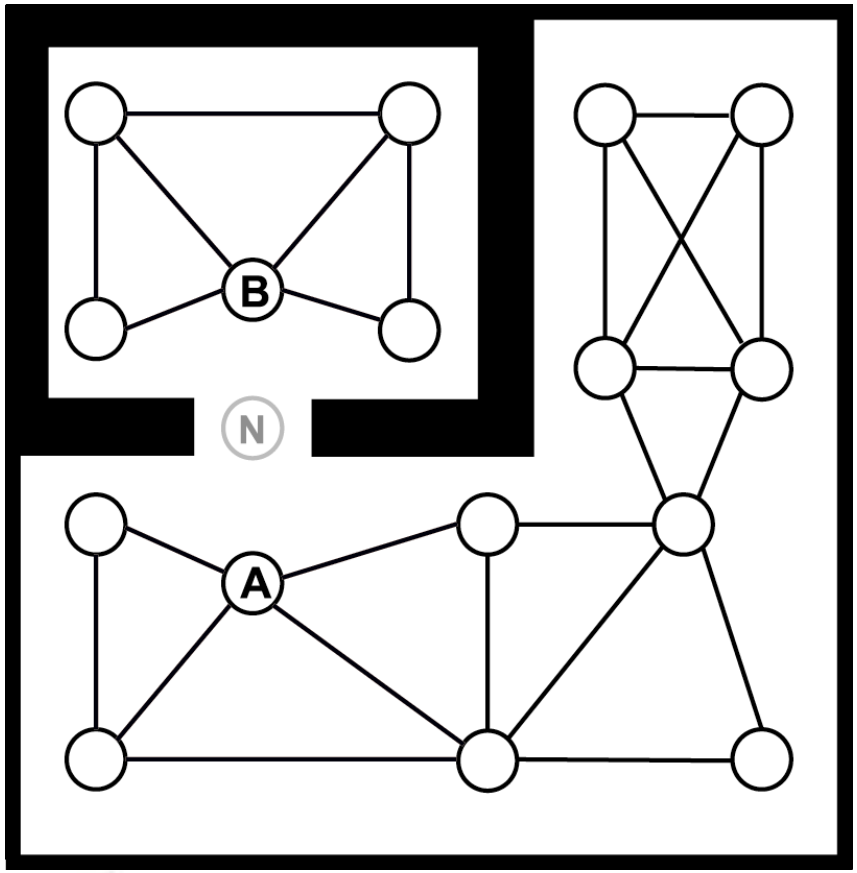

- For each node, N in the node graph with only two neighbors:

# 3. Pinch Points



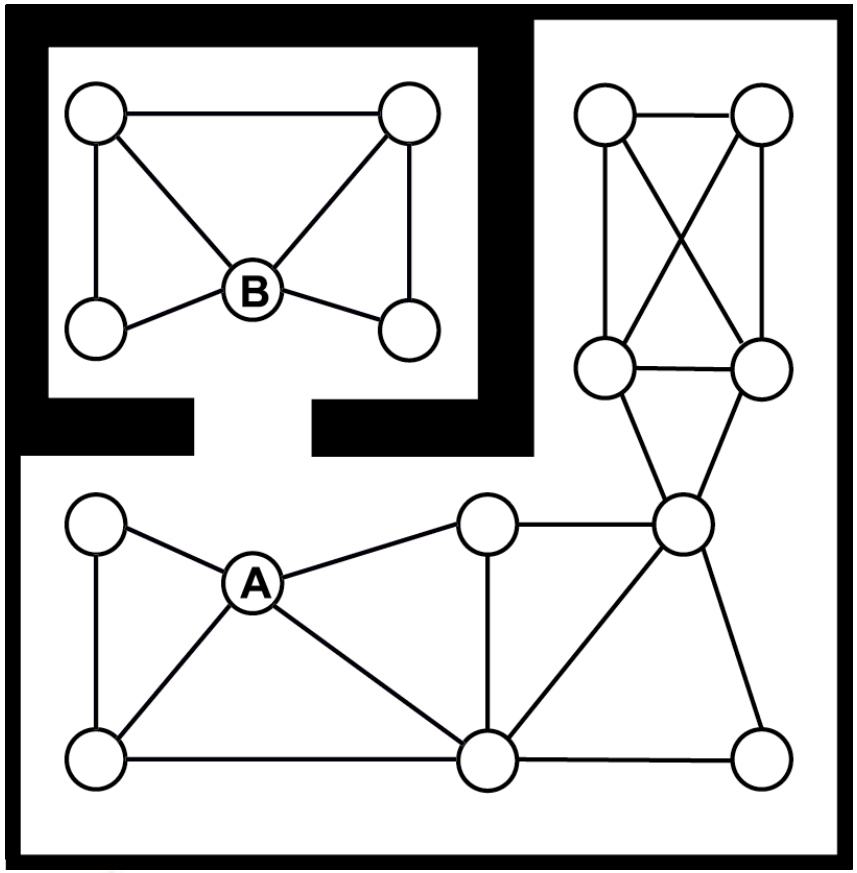

**- Temporarily eliminate node, N, from the graph, call its neighbors as A & B**

# 3. Pinch Points



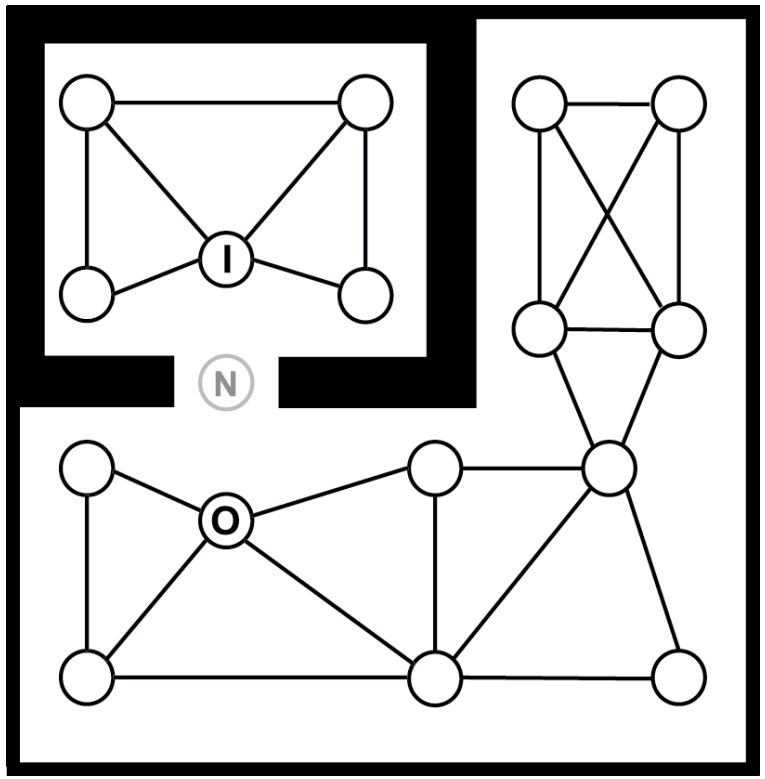

- If both A & B are connected to large regions, N is not a pinch point, try another N

# 3. Pinch Points




**- Attempt to find a path between A& B, if exists try another N.**

# 3. Pinch Points



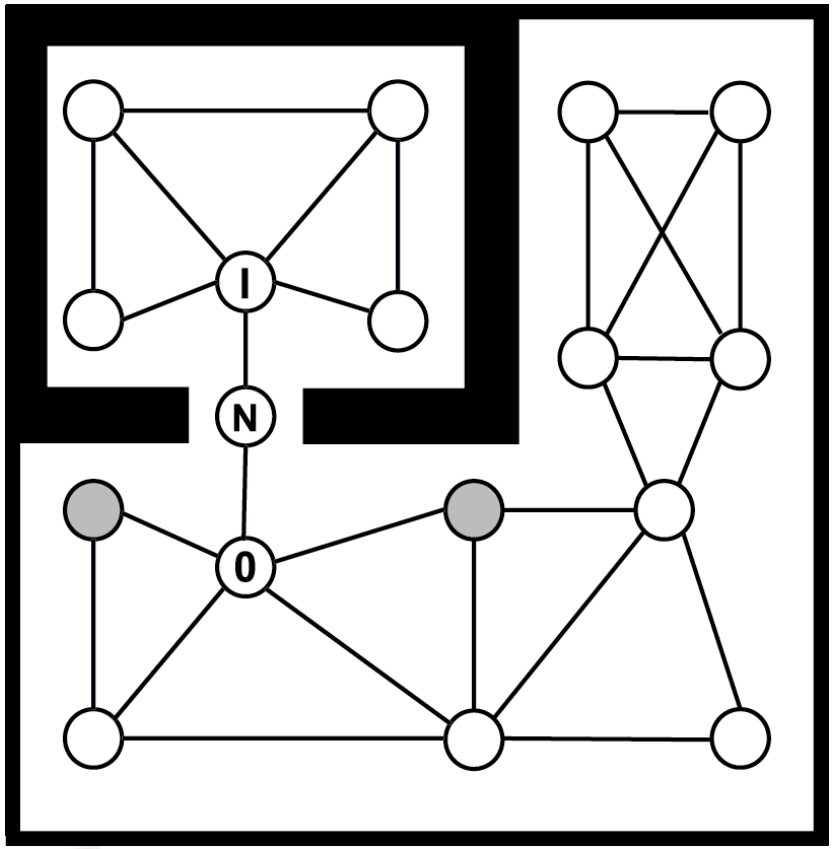

- Call the node connected to the larger region, O (for outside).
- Call the node connected to the smaller region, I (for inside).

# 3. Pinch Points

- **Once a pinch point has been located a good ambush location is one which:**
  - **Has a line of sight to the node outside the pinch location "O"**
  - **Can't be seen from the pinch location "N"**
  - **$V_O$ : Nodes that have a line of sight to pinch location "O"**
  - **$\overline{V_N}$ : Can't be seen from the pinch location "N"**
  - **Good ambush locations is their intersection:**

$$V_P = V_O \cap \overline{V_N}$$

# 3. Pinch Points




*Others?*

# Using Tactical Locations

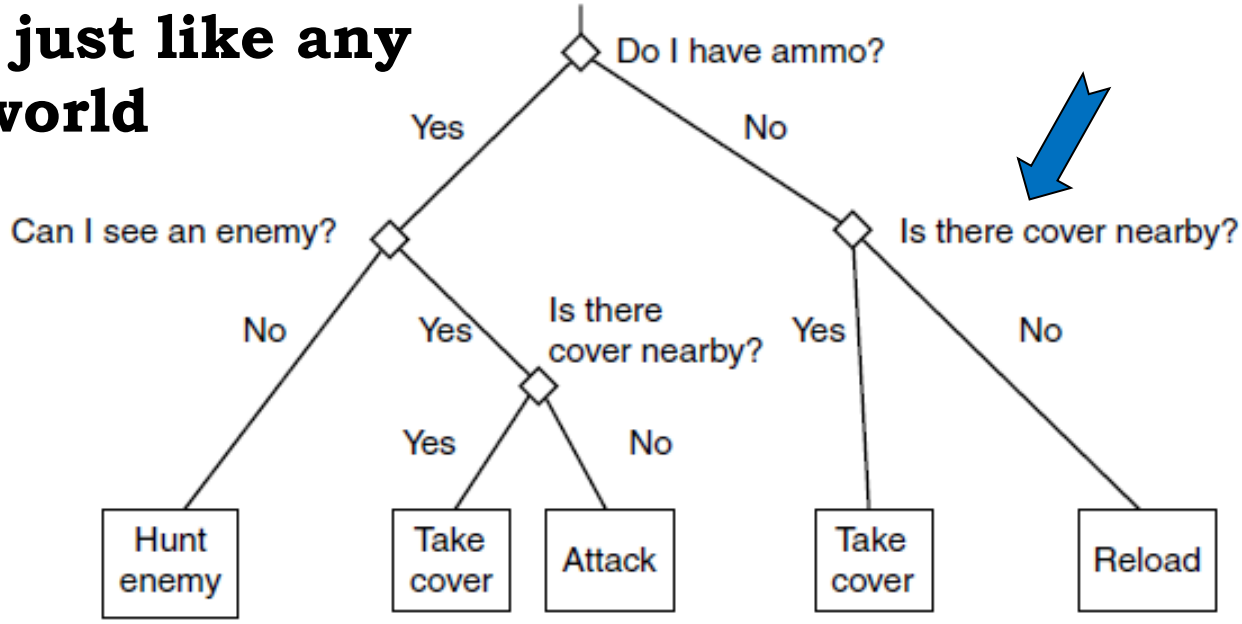
- How do we build a tactical mechanism within the character's decision making AI?
- Three approaches:
  1. Controlling tactical movement (simple method)
  2. Incorporate tactical information into decision-making
  3. Use tactical information during pathfinding to produce character motion that is always tactically aware
- For now, we'll limit our focus to tactical decision making for a single character

# 1. Tactical Movement

- Tactical waypoints are queried during game when the character AI needs to make a tactical move
- E.g., Character needs to reload bullets, it queries the tactical waypoints in the immediate area to look for “nearest suitable location” to stop and reload, before continuing
- This method: Action decision is carried out first, then apply tactical information to achieve its decision
- Some limitation in realism, and not able to use tactical information to influence decision-making due to limited use.

# 2. Tactical Information in Decision-Making

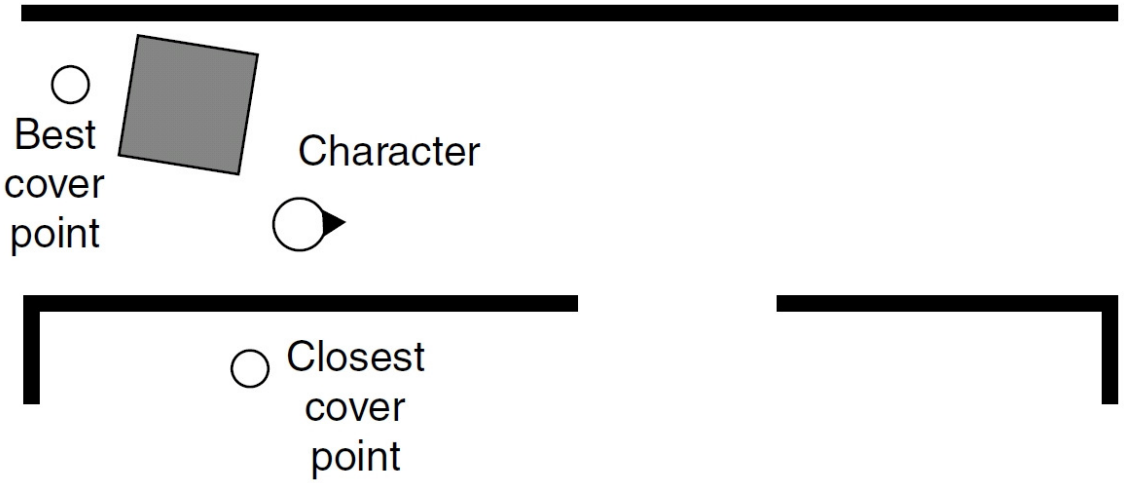
- Give the "decision-maker" access to tactical information, just like any other game world information
- DT example:




- SM: Trigger transitions only when certain waypoints are available and/or fulfill required numeric value (if used)

# Finding Nearby Waypoints

- If we use any of these approaches, we will need a fast method of generating nearby waypoints.
- Most game engines provide a mechanism to rapidly work out what objects are nearby using, e.g., quad-trees or binary space partition trees (BSPs).
- Distance isn't the only thing to take into account (e.g., add pathfinding to the decision).



# 3. Tactical Information during Pathfinding

- Relatively simple extension of basic pathfinding.
- Rather than finding shortest/quickest path, it takes into consideration the tactical situation of the game
- Simplest way is to manipulate graph connection costs
  - add "tactical cost" to locations that are difficult/dangerous
  - reduce "tactical cost" to locations that are easy/safe