AI: Tactical and Strategic

Tactical Waypoints

CS 415: Game Development

Professor Eric Shaffer

Tactical and Strategic AI

Previously discuss pathfinding and decision making

- A*
- Decision trees

Limitations

- Intended for use by a single character (no intelligent coordination between characters)
- Lack ability predict strategic or “big picture” outcomes
- Poor utilization of environment to tactical advantage

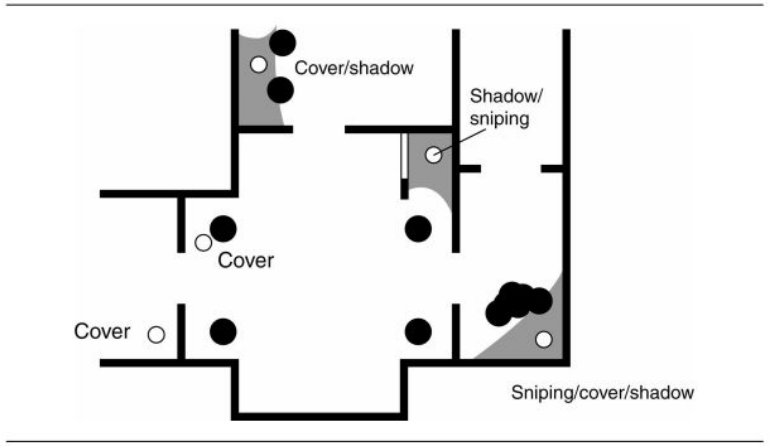
Tactical and strategic AI try to overcome these limitations

- Not needed in some game genres e.g. platformers or simple shooters

Tactical Waypoints

- Waypoint is a position in a game level
  - Used in pathfinding at intermediate nodes along a route

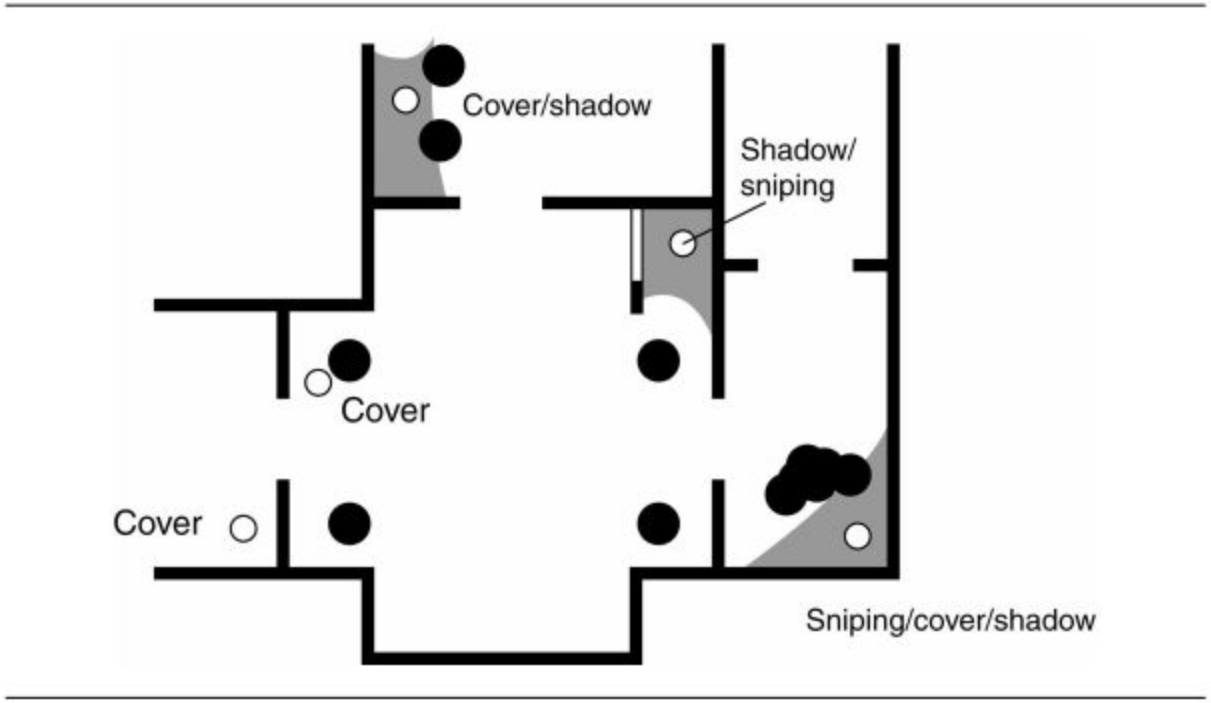
- Can use waypoints for other kind of decision making
  - Can add data indicating special tactical features
  - Sometimes called “rally points”
  - Examples…
    - Safe spot designated for troops to retreat to…
    - Cover points or sniper positions
    - Heavily shadowed areas



ILLINOIS

Tactical vs Pathfinding

Generally makes sense to separate these two kinds of waypoints



These aren’t important spots for pathfinding

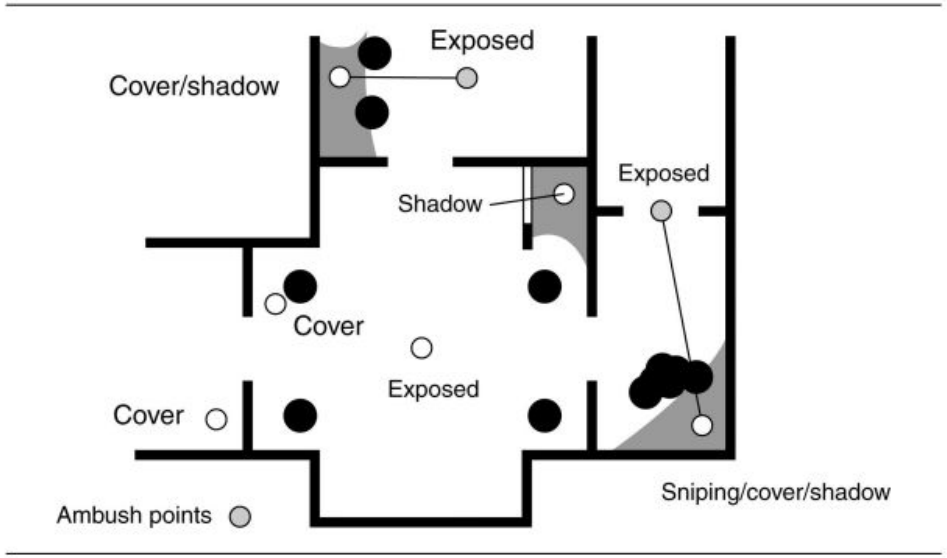
ILLINOIS

Creating a Tactical System

Define a set of primitive tactical properties….for example

- Cover
- Shadow
- Exposed

Level designer will manually label these points



Can define compound tactical properties

- Sniper = cover + nearby exposed
- Hiding = cover+ shadow

Compound tactical waypoints can be automatically generated

- Often “discovered” by AI on the fly during gameplay…reduces memory use
- Eases burden on level designer

ILLINOIS

Open Problem: Topological Analysis

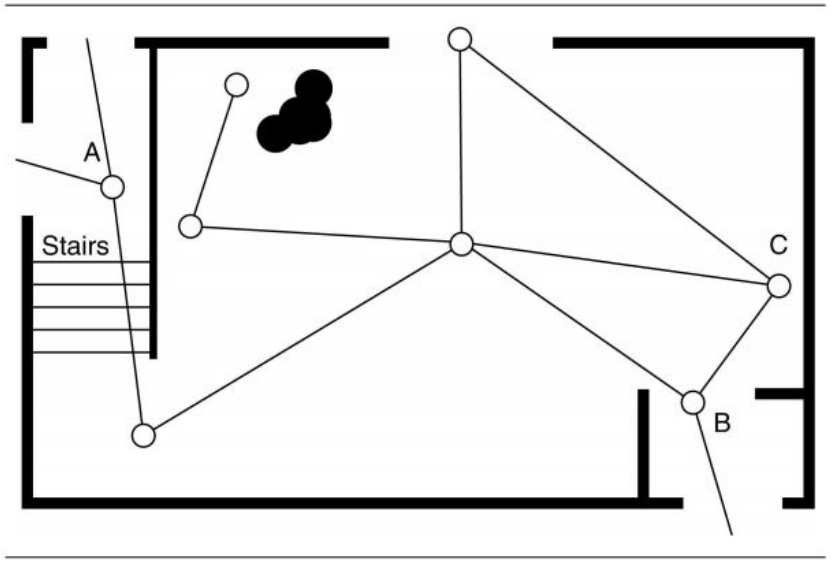
We would like to support sophisticated AI behavior

- Real-life opponents are capable of compound tactics…e.g. hit and run attacks

Typically we are looking for a set of positions for a compound activity

Need to define rule set to guide search

Let's consider an example…

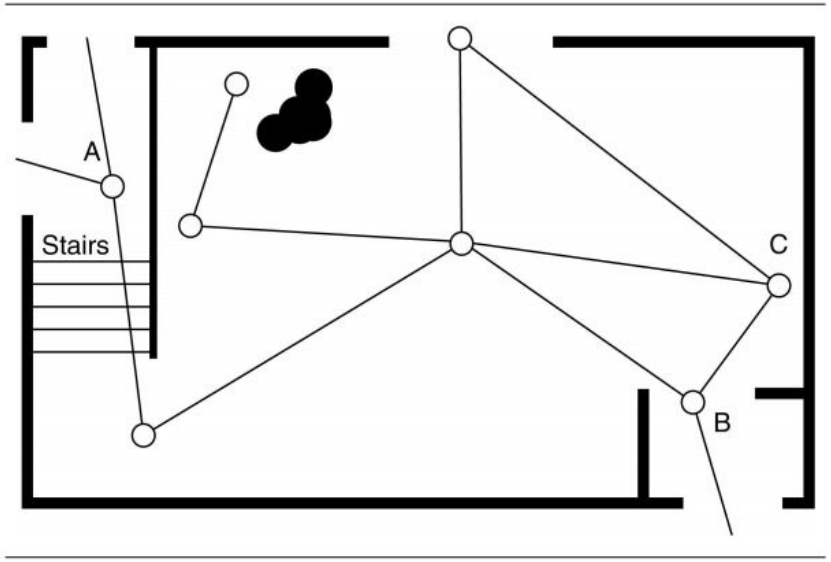


ILLINOIS

Open Problem: Topological Analysis

A hit-and-run attack needs:

- Connected set of waypoints
- Attack location with good visibility
- Attack location has multiple exits to retreat locations
- A retreat locations have cover



Searching for such locations in an automated way is topological analysis

Complicated to implement (like real AI…not game AI)

Currently done as a pre-process by level designers

ILLINOIS

Continuous Tactics

Can increase sophistication by moving away from Boolean states

Use numerical values for “cover” and “shadow”

Use fuzzy rules to make decisions

```
sniper = cover AND visibility
```

If we have a waypoint with cover = 0.9 and visibility = 0.7, we can use the fuzzy rule:

$$m_{(\text{A AND B})} = \min(m_{\text{A}}, m_{\text{B}})$$

where m_{A} and m_{B} are the degrees of membership of A and B. Adding in our data we get:

$$\begin{aligned} m_{\text{sniper}} &= \min(m_{\text{cover}}, m_{\text{visibility}}) \\ &= \min(0.9, 0.7) \\ &= 0.7 \end{aligned}$$

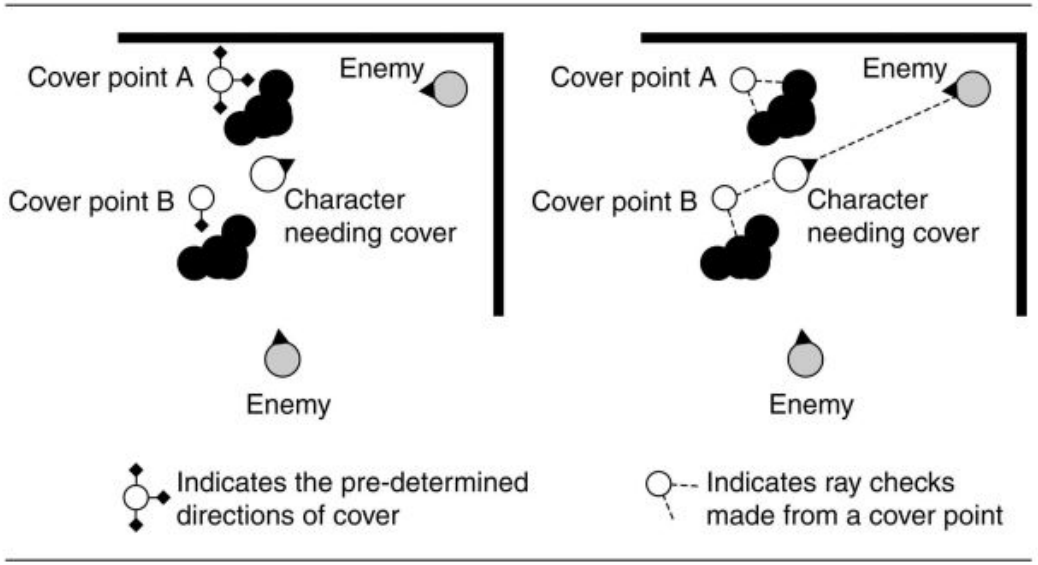
ILLINOIS

Context Sensitivity

Location attributes are sensitive to game context

- Hiding behind a protruding rock is of no use if the enemy is behind you.
  The aim is to put the rock between the character and the incoming fire.

Can pre-compute context values or compute during game play



ILLINOIS

Context Sensitivity

Only games aimed at hard-core players that really benefit from good tactical play, such as squad-based shooters, need the runtime checking approach.

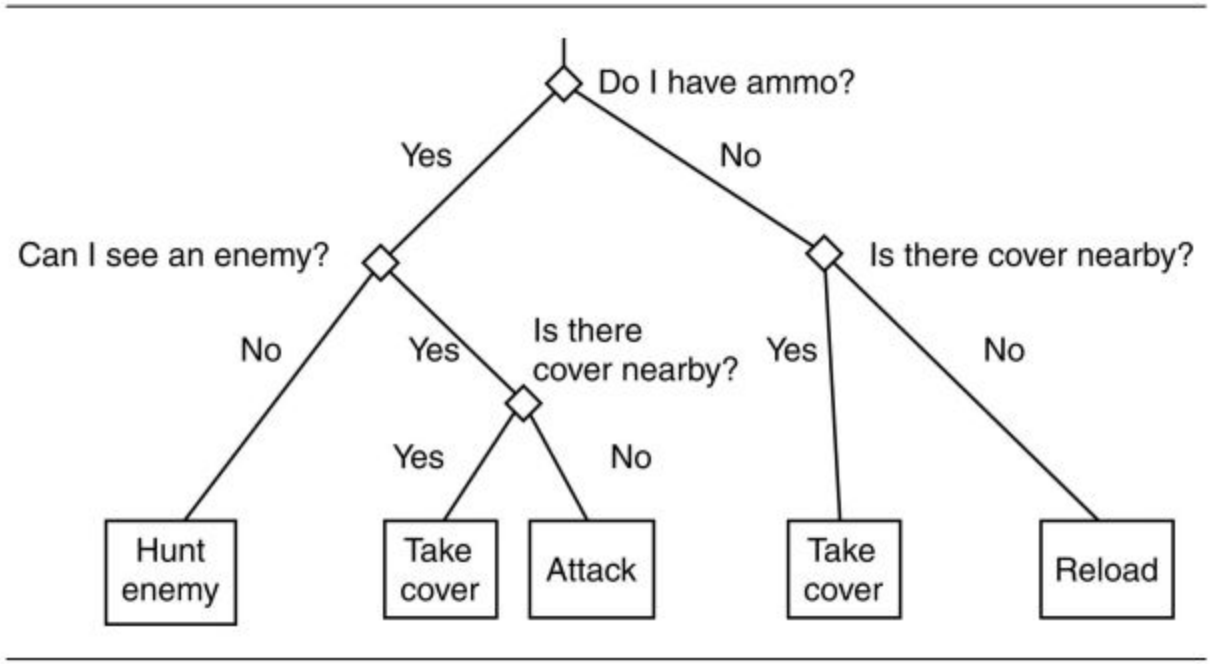
For games where tactics aren’t the focus, small number of states sufficient.

It is possible to combine both approaches in the same game, with the multiple states providing a filtering mechanism that reduce the number of different cover waypoints requiring line-of-sight checks.

Using Tactical Locations: Simple Tactical Movement

How do we use tactical waypoints in decision making?

Use it in decision tree with other info to decide on movement…



ILLINOIS

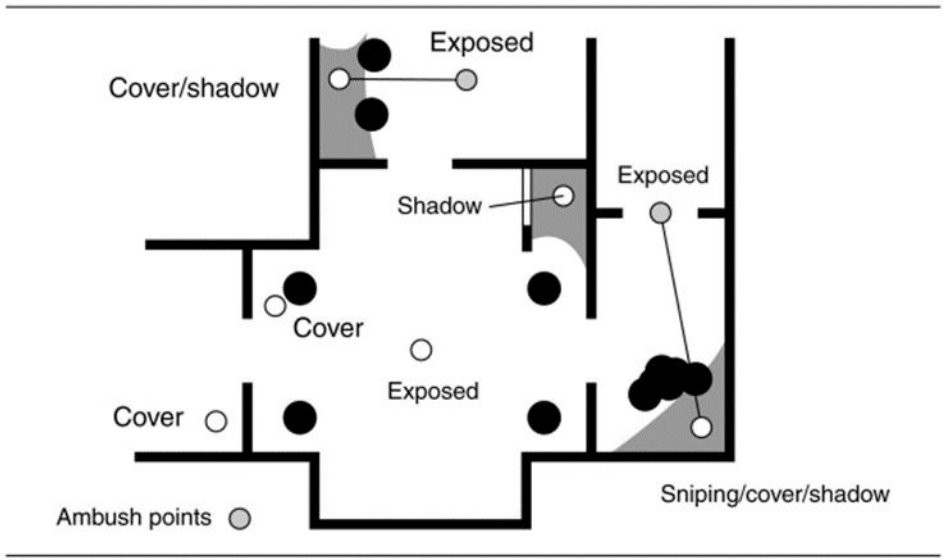
Generating Tactical Properties of Waypoints

Games have a spatial data structure for querying what objects are nearby

- Quad-tree or oct-tree, BSP, kd-tree

Can use this to automatically generate tactical properties

- Cover
- Visibility
- Shadow



ILLINOIS

Cover

Measure how many incoming attacks might succeed

- Ray-cast from scene locations
- To person sized volume at cover location
- Use a semi-random pattern of test locations
- Keep track of a ratio of hits to rays

If you are computing during gameplay

optimize to only care about directions attacks might come from…

```
function getCoverQuality(location: Vector,
                         iterations: int,
                         characterSize: float) -> float:
    # Set up the initial angle.
    theta = 0

    # We start with no hits.
    hits = 0

    # Not all rays are valid.
    valid = 0
    for i in 0..iterations:
        # Create the from location.
        from = location
        from.x += RADIUS * cos(theta) + randomBinomial() * RAND_RADIUS
        from.y += random() * 2 * RAND_RADIUS
        from.z += RADIUS * sin(theta) + randomBinomial() * RAND_RADIUS

        # Check for a valid from location.
        if not inSameRoom(from, location):
            continue
        else:
            valid++

        # Create the to location.
        to = location
        to.x += randomBinomial() * characterSize.x
        to.y += random() * characterSize.y
        to.z += randomBinomial() * characterSize.z

        # Do the check.
        if doesRayCollide(from, to):
            hits++

        # Update the angle.
        theta += ANGLE

    return float(hits) / float(valid)
```

Visibility

Visibility points are calculated in a similar way to cover points…uses many line of sight tests.

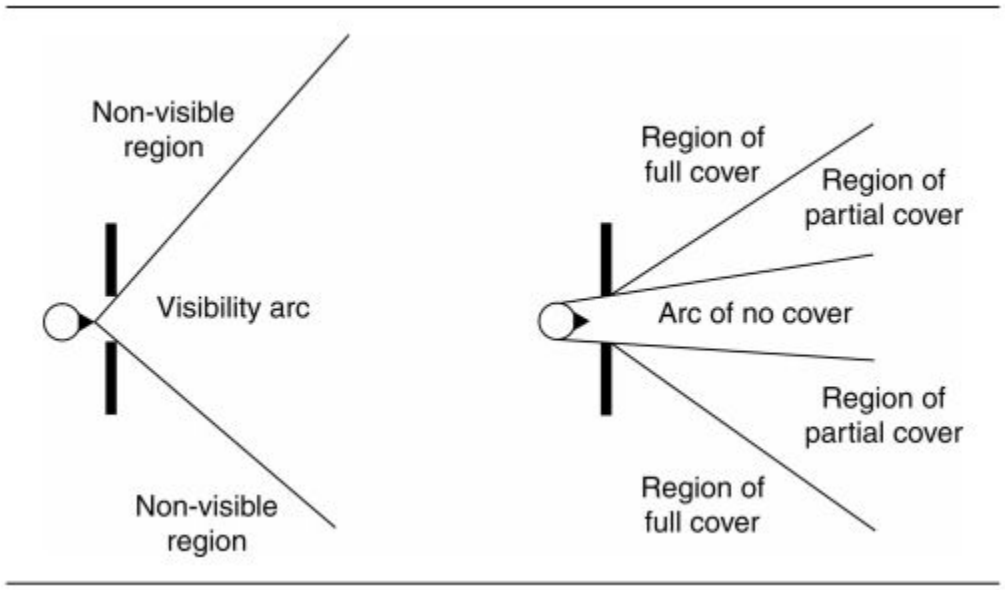
For each ray cast, we select a location…this time we shoot rays out from the waypoint.

The quality of visibility for the waypoint is related to the average length of the rays sent out

- i.e., the distance they travel before they hit an object)

We are approximating the volume of the level that can be seen from the waypoint

- a measure of how good the location is for viewing or targeting enemies.



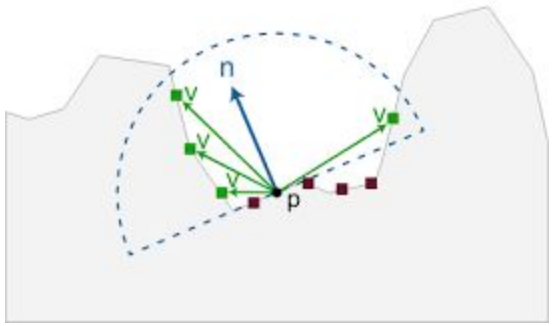
"At first glance it might seem like visibility and cover are merely opposites. If a location is a good cover point, then it is a poor visibility point. Because of the way the ray tests are performed, this isn't the case. Figure shows a point which has both good cover and reasonable visibility. It's the same logic that has people spying through keyholes: they can see a good amount while maintaining low visibility themselves." of

- Millington, Ian. AI for Games, Third Edition

ILLINOIS

Shadow

- Shadow points need to be calculated based on the lighting model for a level.
- Most studios now use global illumination lighting to preprocess light maps for in-game use.
- To determine the quality of a shadow point, sample from a character sized volume around the waypoint.
- This might involve ray casts to nearby light sources to determine if the point is in shadow,
- Or it may involve looking up data from light maps to check the strength of indirect illumination
- The quality of a hiding position is related to the visibility of the most visible part of the character
- For games with dynamic lighting, the shadow calculations need to be performed at runtime.



Sort of like this except rays will come from points $\mathbf{p}_i$ in a volume and we can direct rays to direct lighting sources and then either approximate ambient light or sample as in this diagram.

ILLINOIS

Automatic Waypoint Generation

- One approach…
  - watch human players and store the data…
  - use the data to generate tactical waypoints…
- Another approach
  - Automatically generate dense grid of points in the level
  - Compute (preprocess) metrics for each point
  - Condense (simplify) the grid

Condensing Waypoints

The condensation algorithm has tactical locations compete for inclusion into the final set.

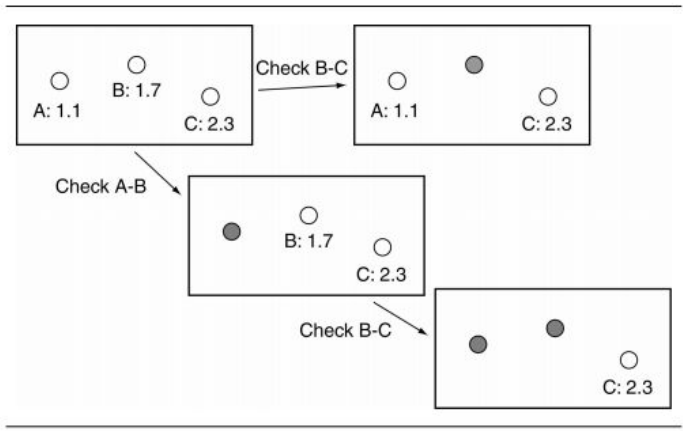
We would like to keep locations that are either:

- very high quality
- or a long distance from any other waypoint of the same type.

For each pair of locations, we first check if a character can move between them

- This is almost always done using a line-of-sight test
- If the movement check fails, then the locations can't compete with one another.
- If the movement check succeeds, the quality values for each location are compared.

If the difference between the values is greater than a weighted distance between the locations, then the location with the lower value is discarded.



This condensation algorithm is highly dependent on the order in which pairs of locations are considered. Figure shows three locations. If we perform a competition between locations A and B, then A is discarded. B is then checked against C. In this case C wins. We end up with only location C. If we first check B and C, however, then C wins. A is now too far from C for C to beat it, so both C and A remain.

Millington, Ian. AI for Games, Third Edition

ILLINOIS