DM810

Computer Game Programming II: AI

Lecture 12

Tactical and Strategic AI

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Tactical Analysis
2. Tactical Pathfinding
3. Coordinated Action

Outline

Tactical Analysis

Primarily present in RTS games

- influence mapping: determining military influence at each location
- terrain analysis: determining the effect of terrain features at each location

But other tactical information too: eg, regions with lots of natural resources to focus harvesting/mining activities.

Influence Maps

- game level split into chunks made of locations of roughly same properties for any tactics we are interested in
  (like in pathfinding: Dirichlet domains, floor polygon, tile-based grid and imposed grid for non-tile-based levels, ...)

- an influence map keeps track of the current balance of military influence at each location in the level

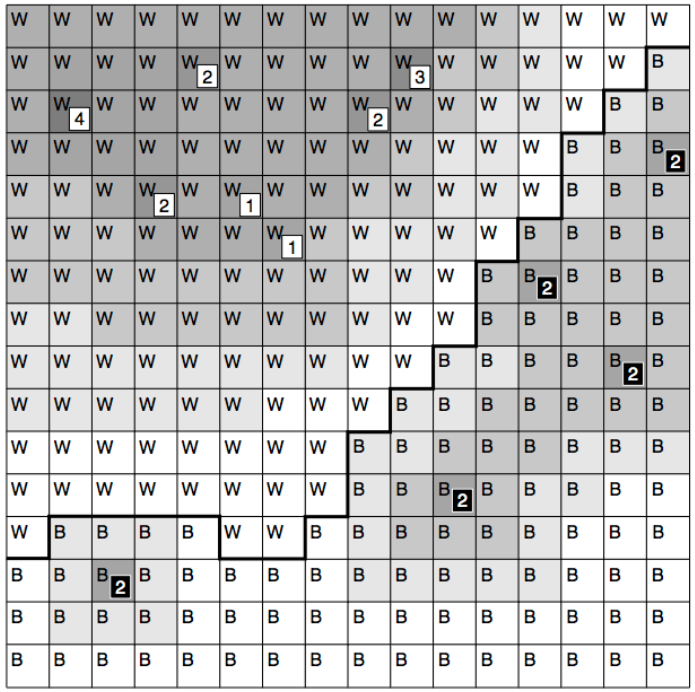
- simplifying assumption: military influence is a factor of the proximity of enemy units and bases and their relative military power (there may be many more)

- Influence is taken to drop off with distance. Let I_0 be intrinsic military power of unit

$$I_d = I_0 - d \qquad I_d = \frac{I_0}{\sqrt{1+d}} \qquad I_d = \frac{I_0}{(1+d)^2}$$

Influence Maps

- values of intrinsic influence set by level designers by visualizing the influence map, tuning needed
- influence at one location by one unit is the drop off formula
- influence of a whole side on a location is the sum of the influence of each unit belonging to that side.
- side with the greatest influence on a location has control over it
- degree of control: difference between the winning influence value and the influence of the second placed side
- degree of control high $\rightsquigarrow$ secure



Calculating Influence values

Three approaches to reduce the $O(mn)$ complexity:

- limited radius of effect
  each unit has intrinsic influence I_0 + radius of effect
  (or threshold value, $r = I_0 - I_t$)

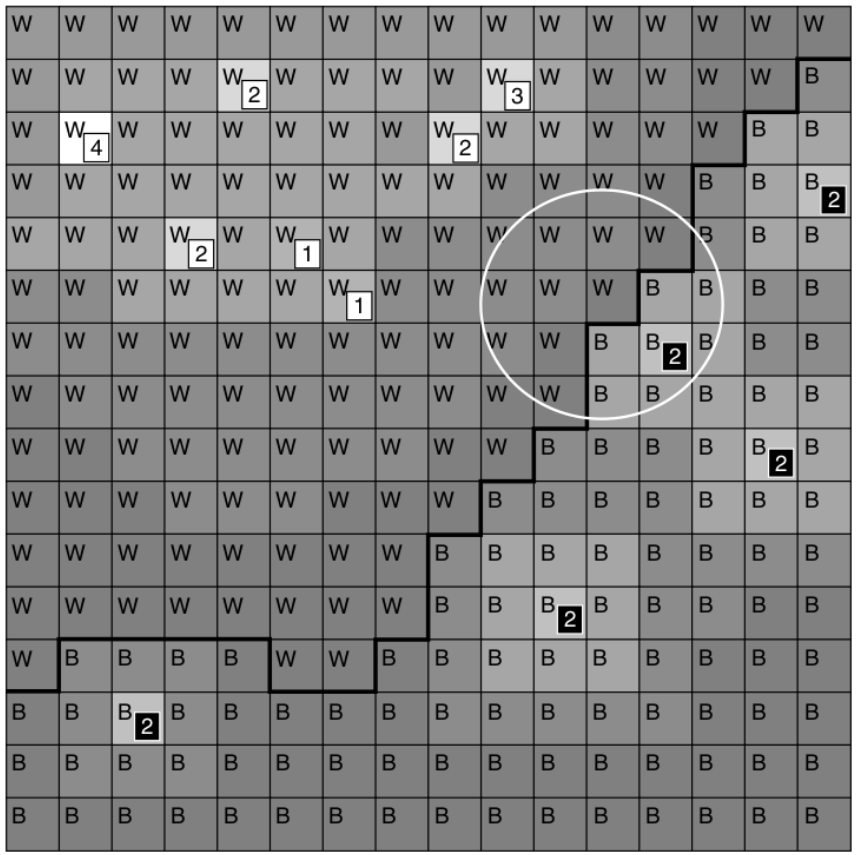
- convolution filters
  filter: a rule for how a location's value is affected by its neighbors;
  influence blurs out; expensive but graphics helps; (eg. Gaussian)

- map flooding
  influence of each location is equal to the largest influence contributed by any unit
  can use Dijkstra algorithm

Calculations can be interrupted and split over frames. Not essential that they are up-to-date.

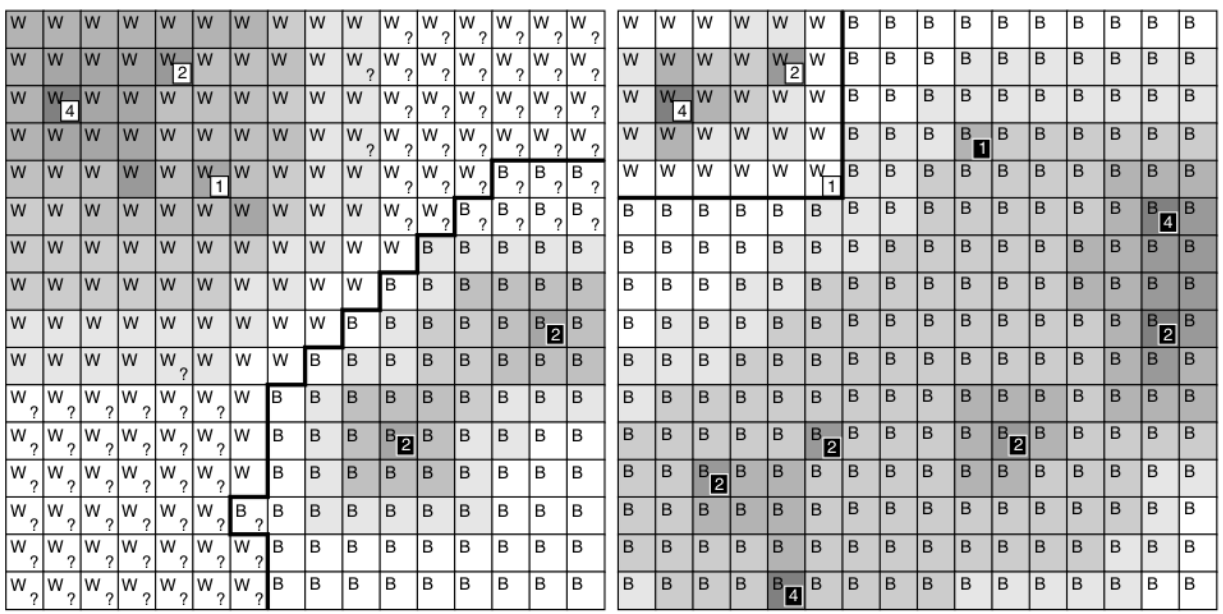
Applications

- which areas of the game are secure
- which areas to avoid
- where the border between the teams is weakest



Lack of Full Knowledge

- some units may not be in line-of-sight
- partial information $\rightsquigarrow$ the two teams may create different influence maps
  $\rightsquigarrow$ one influence map per player
- simplifications assuming full knowledge may be disappointing
- learning to predict from signs



Terrain Analysis

Similar to tactical waypoint analysis but in outdoor environments

Extract useful data from the structure of the landscape:

- difficulty of the terrain (for pathfinding or other movement)
- visibility of each location (to find good attacking locations and to avoid being seen)
- shadow, cover, ease of escape

Calculated on each location by an analysis algorithm depending on information

Terrain Difficulty

1. Each location has a type and each unit has a level of difficulty moving through terrain types.

2. ruggedness of the location: difference in height with neighboring locations (calculated offline)

Combination of 1. and 2., eg. by weighted linear sum

Visibility Maps

- check line-of-sight between location and other significant locations in the level.
- number of locations that can be seen, or average ray length if we are shooting out rays at fixed angles
- need to reduce the number of locations

Learning with Tactical Analysis

Instead of running algorithms at locations:
during the game, whenever an interesting event happens, change the values of some locations in the map.

Frag-maps learned offline during testing. In the final game they will be fixed.

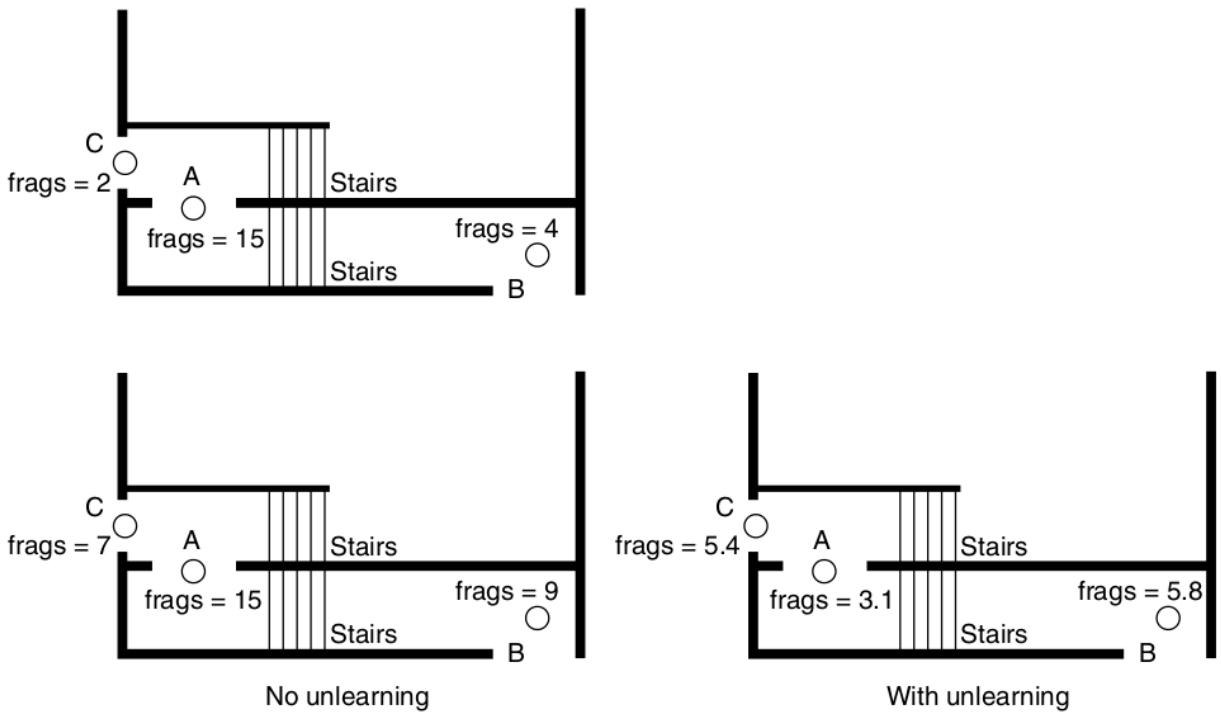
Frag-map per character:

- initially each location gets a zero value
- each time a character gets hit (including the char. itself), subtract a number from the location in the map corresponding to the victim
- if a character hits another character, increase the value of the location corresponding to the attacker.

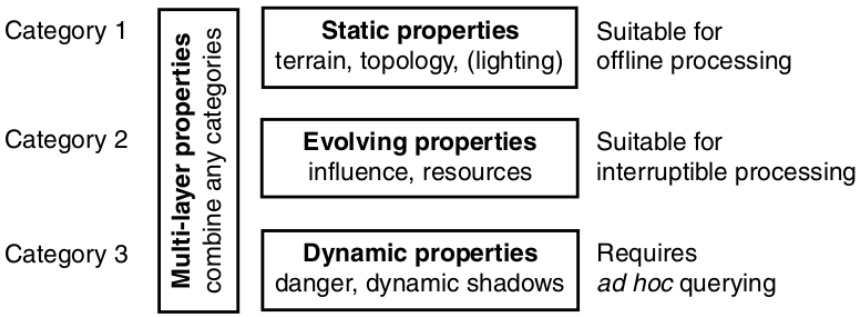
filtering can be used to expand values out to form estimates for locations we have no experience of.

They can be adapted online, forgetting old information

- best location to ambush from
- A is exposed from two directions (locations B and C).
- character gets killed 10 times in location A by 5 attacks from B and C.
- forgetting factor 0.9



Structure for Tactical Analysis



Updating tactical analysis for the whole level at each frame is too time consuming.

limit the recalculation to those areas that we are planning to use:

- neighborhood of the characters
- use second-level tactical analysis

Multi-Layer Analysis

Combine tactical information

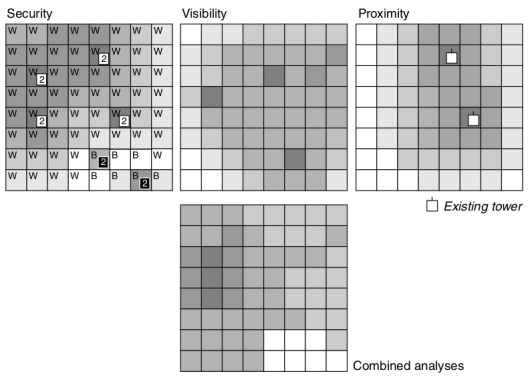
Example: RTS game where the placement of radar towers is critical to success
Relevant properties:

- Wide range of visibility (to get the maximum information)
- In a well-secured location (towers are typically easy to destroy)
- Far from other radar towers (no point duplicating effort)

Combination:

$$\text{Quality} = \text{Security} \times \text{Visibility} \times \text{Distance}$$

$$\text{Quality} = \frac{\text{Security} \times \text{Visibility}}{\text{Tower Influence}}$$



Map flooding

Map flooding: calculates Dirichlet domains on tile-based levels. which tile locations are closer to a given location than any other

Example: a location in the game belongs to the player who has the nearest city to that location
- each city has a strength
- the region of a city's influence extends out in a continuous area.
- cities have maximum radius of influence that depends on the city's strength.

Dijkstra algorithm

- open list: set of city locations.
- labels: controlling city + strength of influence
- process location with greatest strength
- processing: calculate the strength of influence for each location's neighbor for just the city recorded in the current node.
- updates: highest strength wins, and the city and strength are set accordingly; processed locations moved in closed list; changed strength moved to open list.

Convolution Filters

- update matrix
- size of the filter: number of neighbors in each direction.
- new value by multiplying each value in the map by the corresponding value in the matrix and summing the results.
- two copies of the map. One to read and one to write.
- If the sum total of all the elements in our matrix is one, then the values in the map will eventually settle down and not change over additional iterations.
- processing several locations at the same time (SIMD)
- in games we limit the number of pass (one per frame) for speed
- **boundaries**: adapt the matrix but matrix switching is not good.
  adapt the map adding margin numbers that are updated as well

Gaussian blur

smooths out differences

elements of the binomial series

$$
\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}
$$

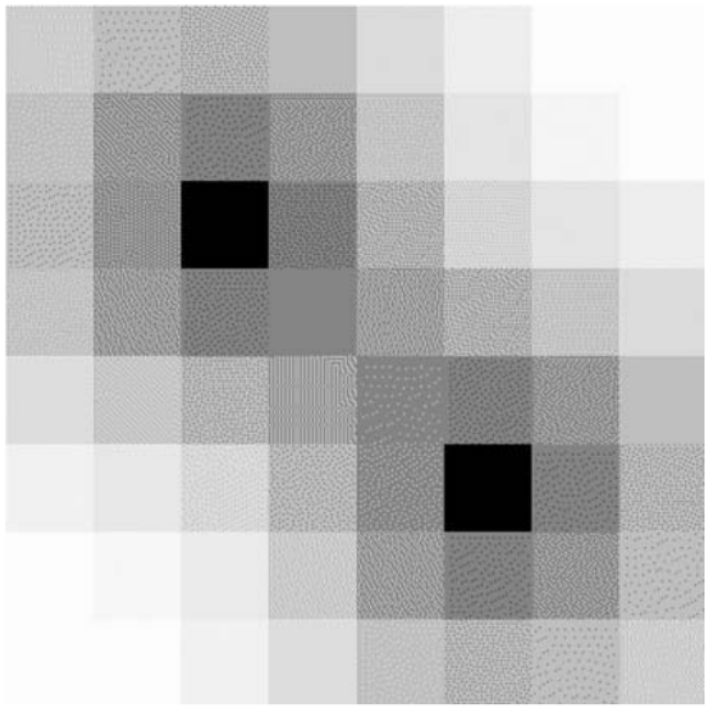
$$
\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} 1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1 \end{bmatrix}.
$$

$$
\begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}
$$

$$
M = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}
$$



Separable: first only vertical and then only horizontal vector

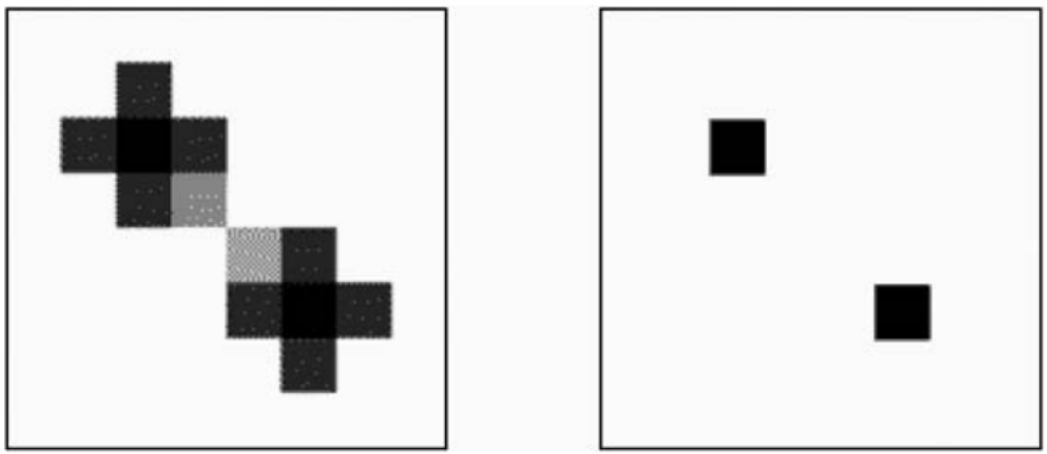
Evaporation at each iteration. The total removed influence is the same as the total influence added

Sharpening filter

concentrates the values in the regions that already have the most

central value will be positive, and those surrounding it will be negative

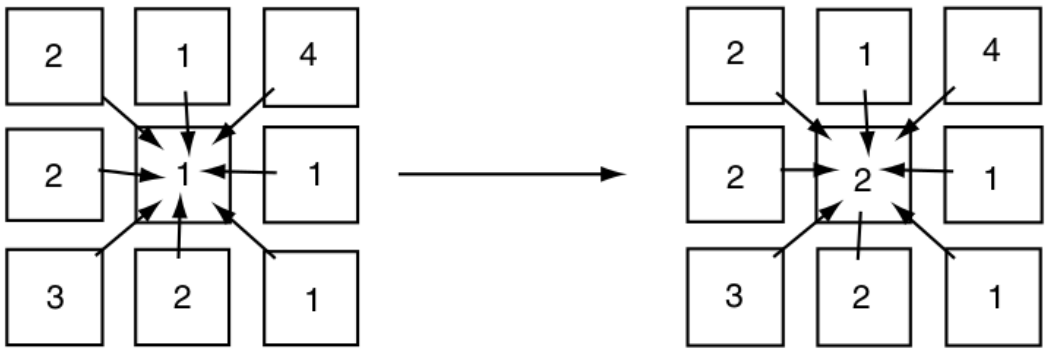
$$\frac{1}{a}\begin{bmatrix} -b & -c & -b \\ -c & a(4b+4c+1) & -c \\ -b & -c & -b \end{bmatrix}$$



Never run to steady state

Cellular Automata

- update rules generate the value at one location in the map based on values of other surrounding locations (eg. Conway's"The Game of Life")
- at each iteration values are calculated based on the surrounding values at the previous iteration.
- values in each surrounding location are first split into discrete categories
- update for one location depends only on the value of locations at the previous iteration ⇝ need two copies of the tactical analysis



IF two or more neighbors with higher values,
THEN increment

IF no neighbors with as high a value,
THEN decrement

Rules

- output category, based on the numbers of its neighbors in each location (If two neighboring locations change their category based on each other, then the changes can oscillate backward and forward)

- in other applications maps are considered to be either infinite or toroidal but not in games

- rules that are based on larger neighborhoods (not just locations that touch the location in question) and proportions rather than absolute numbers.
  Eg: if at least 25% of neighboring locations are high-crime areas then a location is also high crime

- In most cases, rules are heavily branched: lots of `switch` or `if` statements, which are not easily parallelized

- rule of thumb to avoid dynamism: set only one threshold

Applications

Areas of Security

A location is secure if at least four of its eight neighbors (or 50% for edges) are secure

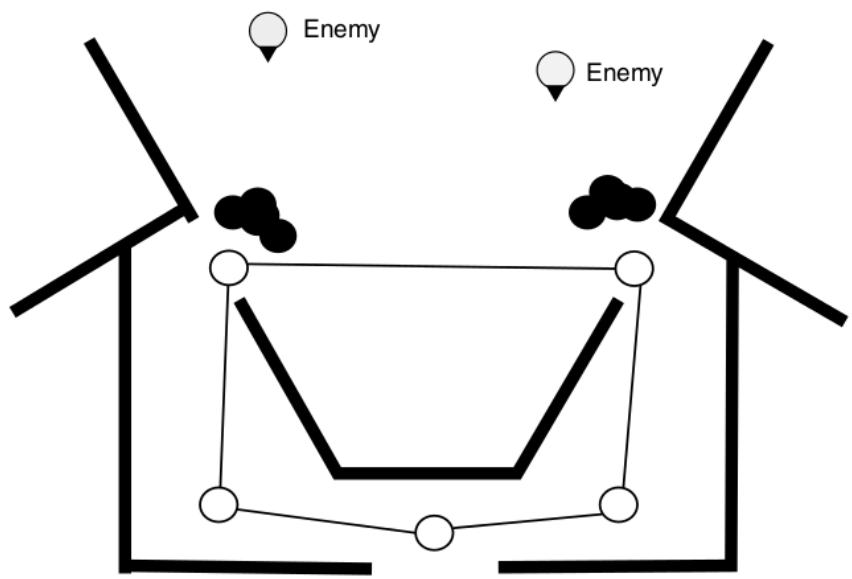
Building a City

way buildings change depending on their neighborhood
a building appears on an empty patch of land when it has one square containing water and one square containing trees
Taller buildings come into being on squares that border two buildings of the next smaller size, or three buildings of one size smaller, or four buildings of one size smaller still.
buildings are not removed
In RTS reasoning on a flow of resources

Outline

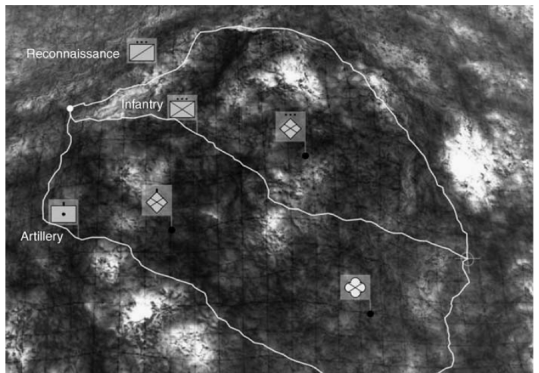
Tactical Pathfinding

- when characters move, taking account of their tactical surroundings, staying in cover, and avoiding enemy lines of fire and common ambush points
- same pathfinding algorithms as saw are used on the same kind of graph representation.
- cost function is extended to include tactical information
  $C = D + \sum_i w_i T_i$
- problem: tactical information is stored on nodes
  $\rightsquigarrow$ average the tactical quality of each of the locations incident to the arc



Tactic Weights and Concern Blending

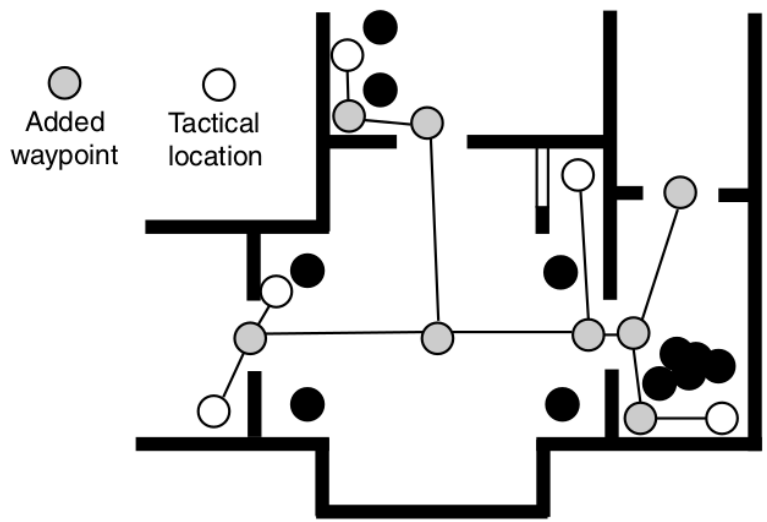
- If a tactic has a high weight, then locations with that tactical property will be avoided by the character
- If large negative value, then the character will favor locations with a high value for that property
- but! negative costs are not supported by normal pathfinding algorithms such as A*; use `assert` to check
- Performing the cost calculations when they are needed slows down pathfinding significantly.
  Is the advantage of better tactical routes for your characters outweighed by the extra time they need to plan the route in the first place?
- Weights have an impact on character personalization
  also they are not static



- Issue: heuristic may become invalid (not admissible)
  Eg: Euclidean distance not anymore valid; maybe multiplying it by a factor?

- graph: nodes derived from influence maps for outdoor and tactical waypoints for indoor
  connections: generate them by running movement checks or line-of-sight checks between map locations or waypoints
  if grid based: adjacent locations are connected + gradient threshold considerations

- necessary to add additional waypoints at locations that do not have peculiar tactical properties. Superimpose the tactical waypoints onto a regular pathfinding graph.



Outline

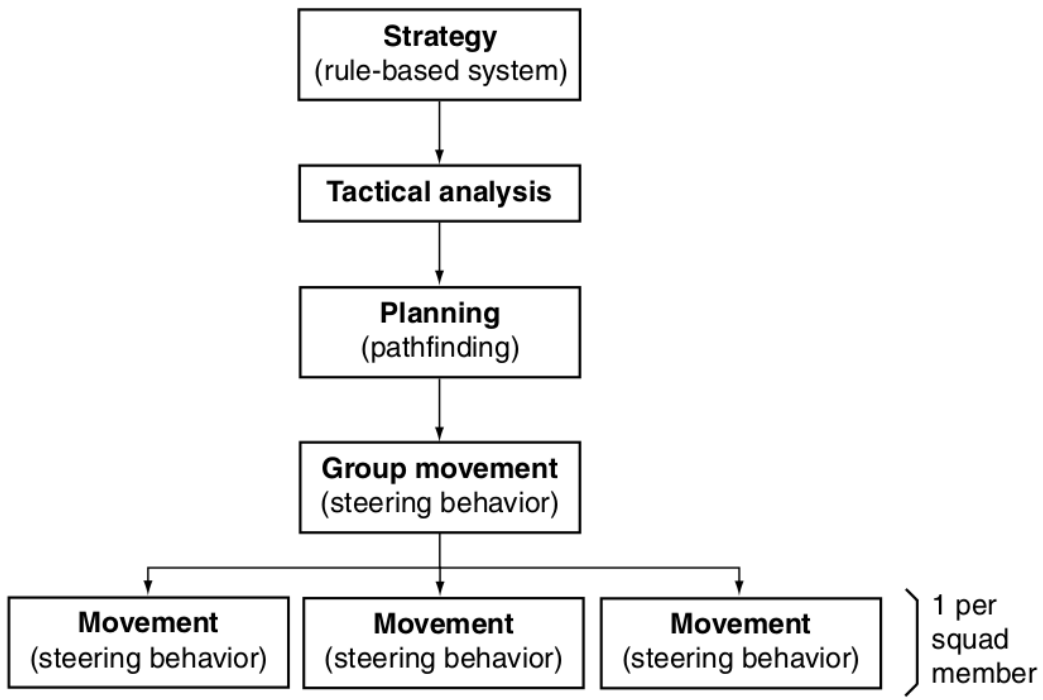
Coordinated Action

- a whole side in a real-time strategy game
- squads or pairs of individuals in a shooter game.
- AI characters acting in a squad led by the player: cooperation occurs without any explicit orders being given.
  Characters need to detect the player's intent and act to support it.

AI characters need to tell each other what they are planning through some kind of messaging system,

Multi-Tier AI

top-down vs bottom-up (emergency) approach
Hierarchy of behaviors: at some levels decisions shared among multiple characters

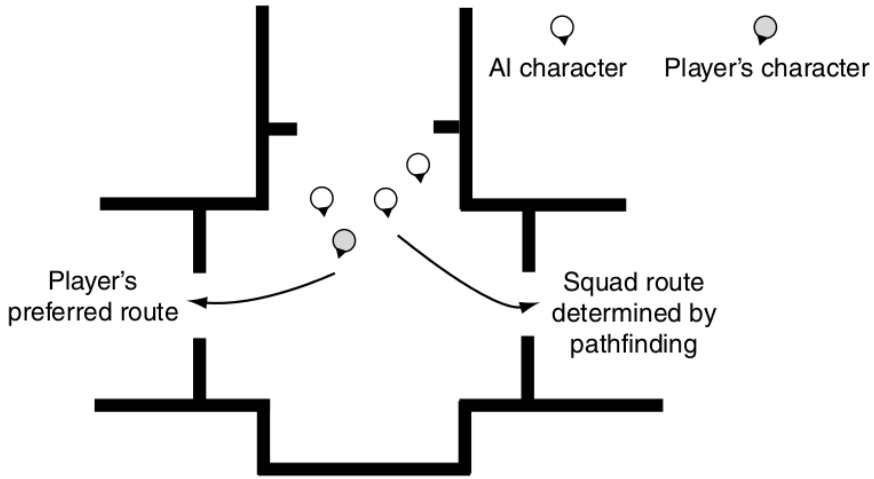


- Group Decisions
  standard expert systems or state machines, tactical analysis or waypoint tactic algorithms
  difference: actions are not scheduled for execution by the character, but are orders passed down

- Group Movement
  steering behaviors (feasibility issues)

- Group Pathfinding
  different blend of tactical concerns for pathfinding than any individual would have alone
  heuristic of weakest character
  decision tree, expert system, or other decision making technology to detect constraints and separate the group

- Including the Player
  high-level decision making may make a decision that the player subverts



  The game design may involve giving the player orders, but ultimately it is the player who is responsible for determining how to carry them out.

  Alternatively Player at the top level of the hierarchy and conflicts avoided.

- Explicit Player Orders, eg. in RTS games
  intent-identification problem

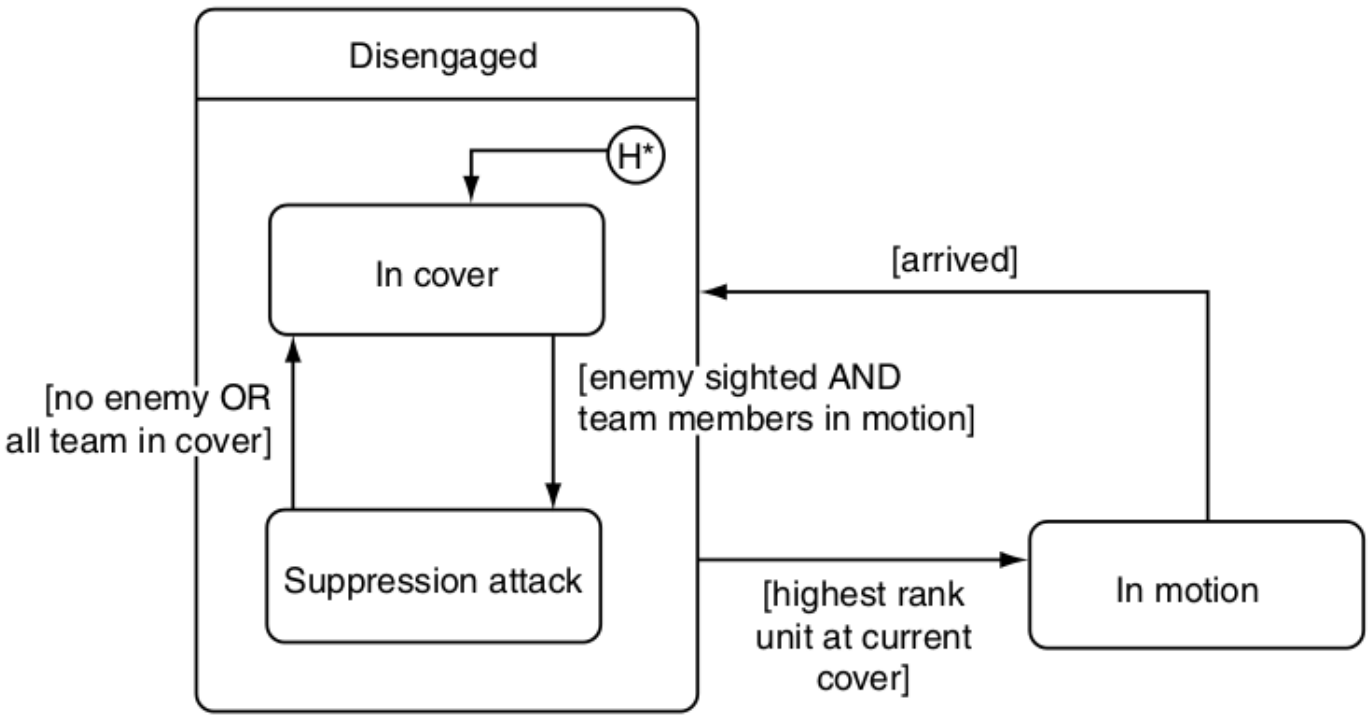
Structuring Multi-Tier AI

- communication mechanism that can transfer orders from higher layers in the hierarchy downward. This needs to include information about the overall strategy, targets for individual characters, and typically other information (such as which areas to avoid because other characters will be there, or even complete routes to take).

- A hierarchical scheduling system that can execute the correct behaviors at the right time, in the right order, and only when they are required.

Emergent Cooperation

- less centralized techniques to make a number of characters appear to be working together.
- Each character has its own decision making
- the decision making takes into account what other characters are doing
- If any member of the team is removed, the rest of the team will still behave relatively efficiently
- most squad based games


Disengaged
H*
In cover
[arrived]
[no enemy OR
all team in cover]
[enemy sighted AND
team members in motion]
Suppression attack
[highest rank
unit at current
cover]
In motion

- Scalability
- Predictability