



VK Computer Games

Mathias Lux & Horst Pichler
Universität Klagenfurt



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License. See <http://creativecommons.org/licenses/by-nc-sa/2.0/at/>

AI in Games



<http://www.uni-klu.ac.at>

Game-AI must

- ... mimick intelligence
- ... add to the gaming experience
- ... raise the level of challenge

Intelligent behaviour

- movement of bots (enemies, NPCs)
- combat tactics and planning
- interaction with the player
 - communication
 - trade
 - diplomacy
- etc.

A very short History



<http://www.uni-klu.ac.at>

In the good old times

- enemy „intelligence“ follows fixed patterns (e.g., boss fights)

In the 1990s (more processing power available)

- complex pathfinding problems
- planning & tactics
- incomplete information algorithms („fog of war“)

In the 2000s

- machine learning algorithms (e.g., neural networks)
 - „emergent behavior“ (cmp.: game of life)
 - study and design of complex/multi-agent systems
 - swarm-intelligence (flocking of birds)
- ➔ enemies without „cheating“ AI

Some Basic Rules I



<http://www.uni-klu.ac.at>

Only the player's impression counts

- even sophisticated algorithms may produce behaviour patterns, that create the impression of silly, random, or annoying behaviour
- AI which goes unrecognized is dispensable
- too smart AI or usage of complete information algorithms (cheating AI, godlike powers) results in player demotivation

Realistic AI is usually complex and expensive (CPU)

- fake intelligence whenever possible
- it is eventually not necessary to implement a goal-regression planner if the possible sequences of actions to achieve a certain goal are limited

Some Basic Rules II



<http://www.uni-klu.ac.at>

IQ must suite the enemy type






- an enemy-maintenance robot that moves randomly might still appear realistic
- an assassin should avoid the player's line of sight and try to sneak up from behind

Modify intelligence

- predictable bots are boring (unless it is an essential part of the game)
- raise the level of intelligence corresponding to the player's learning curve and/or increasing levels

Example: Ghosts in Pacman



- Each ghost has a specific „attack“ patterns
 -  Clyde – stupid, moves completely randomly
 -  Inky – random movement, starts chasing when close
 -  Pinky – ambushes (roundabout)
 -  Blinky – chases the player
- Looks simple ...
 -  ... but the designer put a lot of thought into behaviour patterns, which made the game an instant classic

Basic Ghost Modes



Pursuit / „Attack“

- ghosts move according to their „attack“ patterns

Scatter


- ghosts head for their home corners (for 5 to 7 seconds)
- triggered by a timer
- ghosts scatter up to four times per life/level
- „it seemed to be more natural than constant attack“
(Toru Iwatani, creator of Pacman)

„Blue“ mode

- when Pacman eats a power pill
- ghosts move slower with different movement patterns


Red Ghost: Blinky / Shadow



 Blinky begins each level moving at the same speed as all of the other ghosts, but after you've eaten a certain number of dots, he begins to speed up (he takes on the identity of 'Cruise Elroy').




 Blinky becomes Cruise Elroy earlier and earlier as you progress to higher and higher levels [...].


 Unlike the other ghosts, Blinky will also tend to follow close on your tail even when you turn and will often still chase you even in scatter mode.

Pink Ghost: Pinky / Speedy



 Pinky seems to have a tendency to go around blocks in an anticlockwise direction unlike Blinky and Clyde who seem to prefer going clockwise.




 This means that if Blinky and Pinky reach the opposite side of a block to where you are, they'll come at you from opposite sides of it. They can often trap you like this so be careful of this deadly duo.


Blue Ghost: Inky / Bashful



 Inky is dangerous because he's unpredictable.

 Given the same choices, he will often take different turns at different times.




 There might be rhyme and reason to his behaviour, but we haven't recognised it yet.


- One theory is that Inky's behaviour depends on his proximity to Blinky almost as if he is too afraid to act on his own (like some people who never go to a cinema by themselves).
- Another unconfirmed theory about Inky is that he will often turn off if Pac-Man charges him.

Yellow Ghost: Clyde / Pokey



 Clyde is either short-sighted or stupid.

 He will often turn off rather than approach you. His heart doesn't seem to be in it at all.

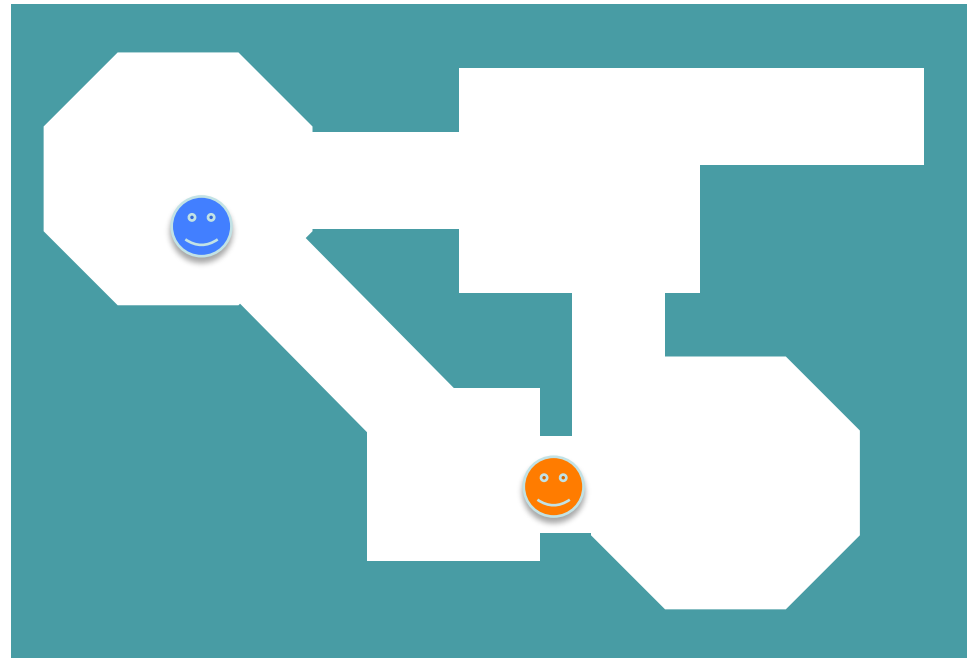
 A consequence of Clyde's unwillingness to take part is that it's often hard to round all of the ghosts up into a single cluster which is nice to do just before eating a power pill.

Creating an Illusion of Intelligence



<http://www.uni-klu.ac.at>

- Seeing
- Hearing
- Reacting



The Line of Sight I

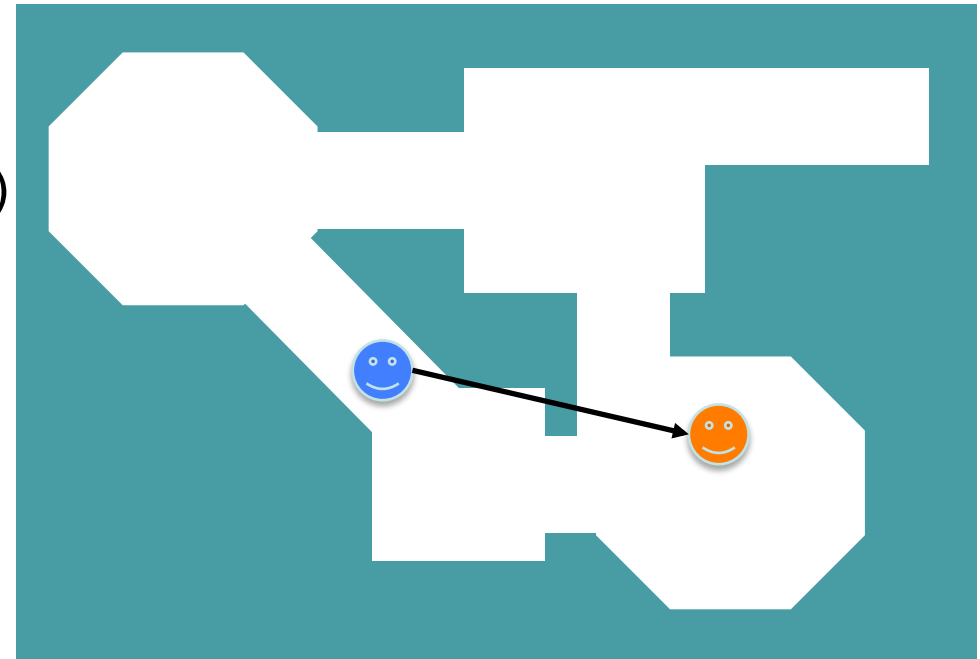


<http://www.uni-klu.ac.at>



Vision ray

- draw a line (or a rectangle) between objects (evtl. limited len)
- find all obstacles that intersect with this line (collision detect.)
- no problem for tile-based levels
 - use bounding boxes
- in this example:
 - either: invisible bounding polygons that represent obstacles
 - or: invisible monochrome bitmap (0 empty, 1 obstacle) and check bits along the line (decrease granularity if necessary)
- problem: bots have eyes in the back of their heads (no blind spots)



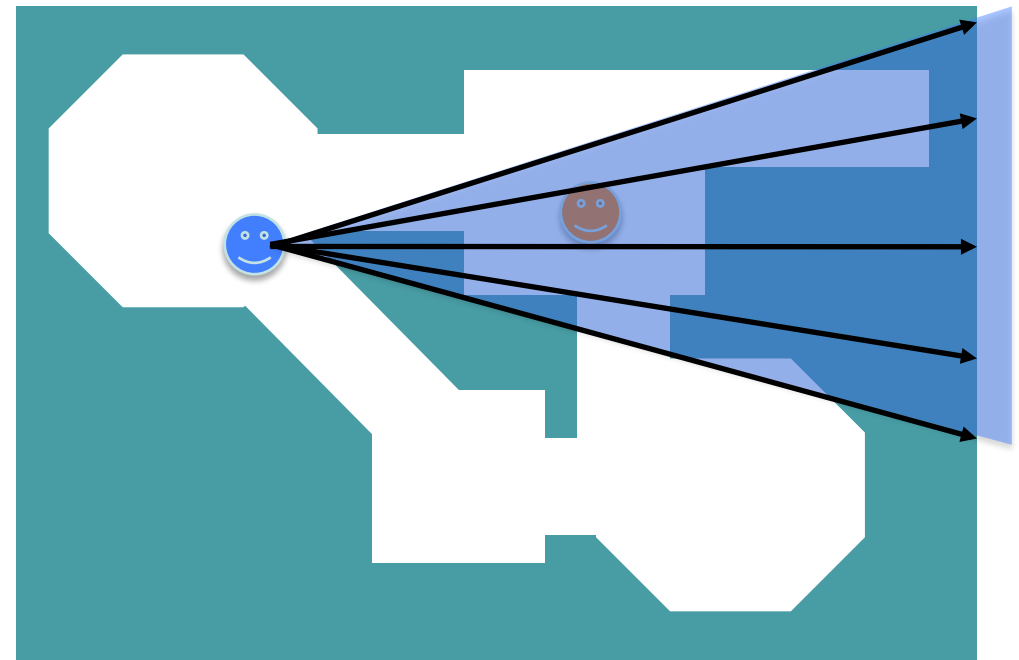
The Line of Sight II



<http://www.uni-klu.ac.at>

Vision cone (simple variant)

- create a triangle (polygon) with given „viewing“ angle
- rotate triangle into viewing direction of bot
- check if object collides with the triangle (intersect)
- if intersects:
 - check with multiple rays
(optimized: only between tangent intersection points)
- alternative: only multiple rays (no triangle)



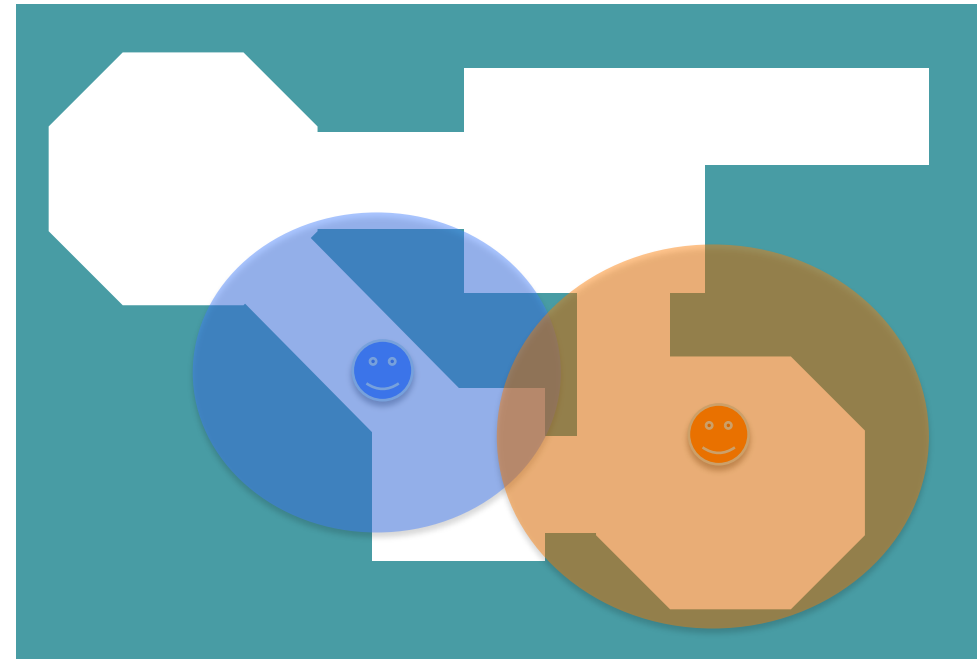
„Audio Circles“



<http://www.uni-klu.ac.at>

Noise and listening circles

- if the player does something that makes noise (shoot, reload, ...)
 - create a „noise“-circle
 - the louder the noise the greater the radius
- create „listening“-circles for bots
 - the better a bot hears the greater the circle
- collision check between noise and listening circles



Possible cases

- accurate hearing: now the bot knows player's position
- inaccurate hearing: the bot approximates the direction to the player's position (somewhere between the intersection points; randomized)

Reaction

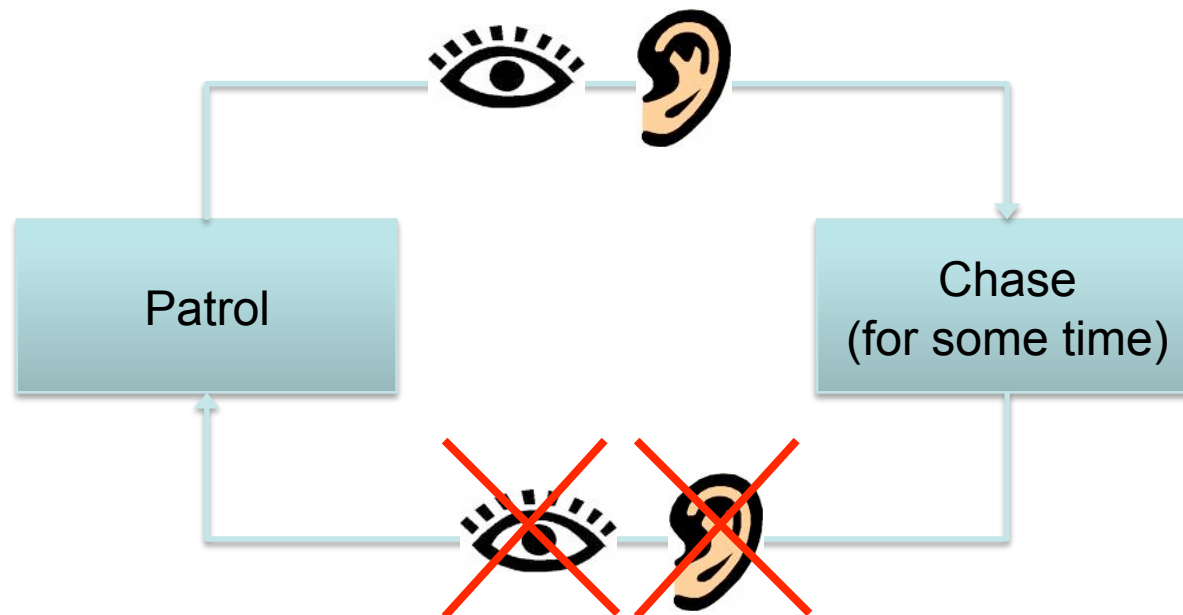


<http://www.uni-klu.ac.at>

 Bot must react according to situation and strategy

- chase, attack, flee, hide, power up, etc.

 Use finite state machine to implement reaction



Making Decisions



<http://www.uni-klu.ac.at>

Making a decision results in a state change

- triggered by interaction with player (seeing, hearing, ...)
- triggered by game world (e.g., bot spots a power-up)
- triggered by status (e.g., low health results in evasive behav.)
- triggered without obvious reason (cmp. ghosts)
 - e.g., change between wait, run, walk, patrol, ...

Do not change the state too fast

- „tweak“ your game until it feels right

Use random functions

- apply probabilities for state changes, such that certain states are more or less frequent

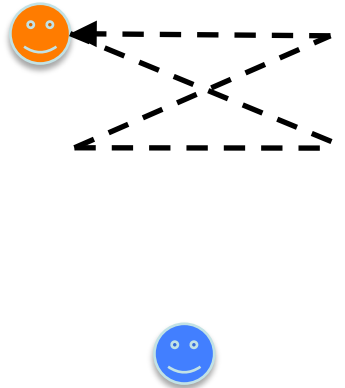
Some Behaviour Patterns



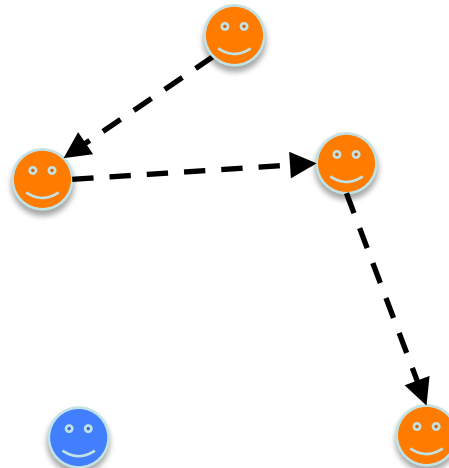
<http://www.uni-klu.ac.at>

- Dodge and Flee

Zigzag



Duck&RandomMove



Hide



Flee



Some Behaviour Patterns



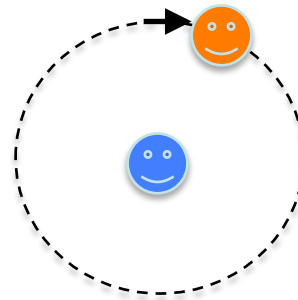
<http://www.uni-klu.ac.at>

- Attack

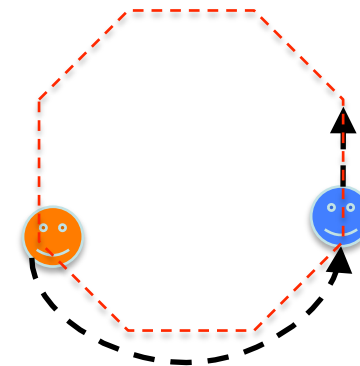
Rush



Strafe



Sneak



move along a circle,
ellipse, hexagon, octagon, ...

Some Behaviour Patterns



<http://www.uni-klu.ac.at>

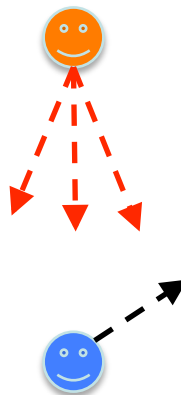
- Aim and Fire

 to consider: time between aiming and firing!

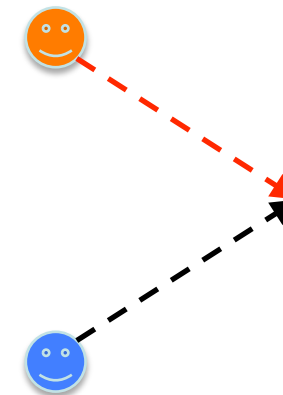
Direct



Random



Predictive



- Aiming-patterns can also be used for chasing
- Chasing may come with other changes: acceleration, rotation of turret, etc.

Pathfinding

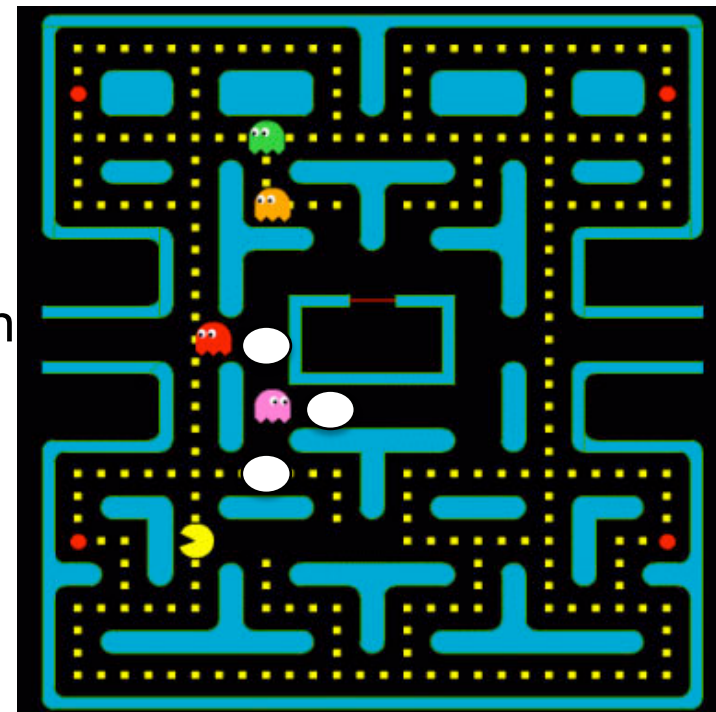


<http://www.uni-klu.ac.at>

 Find (good) path between start and goal

 Basic idea:

- assume Pinky wants to move towards Pacman
- it has 3 possibilities: up, down, or right (white circles)
- calculate Manhattan distance between player position and each of these 3 fields
 - up: 5 (fields, tiles)
 - right: 6
 - down: 3
- decision: move down!

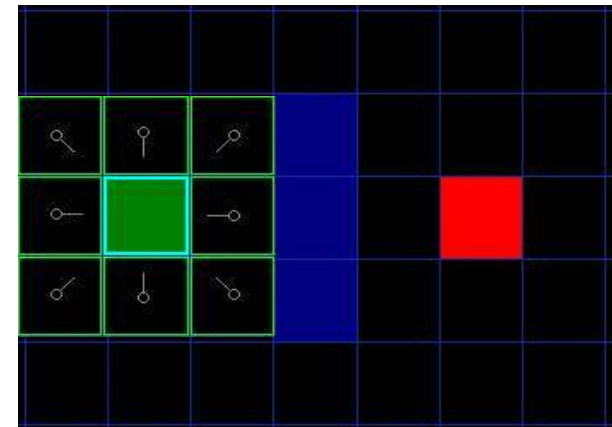


Pathfinding with A*



<http://www.uni-klu.ac.at>

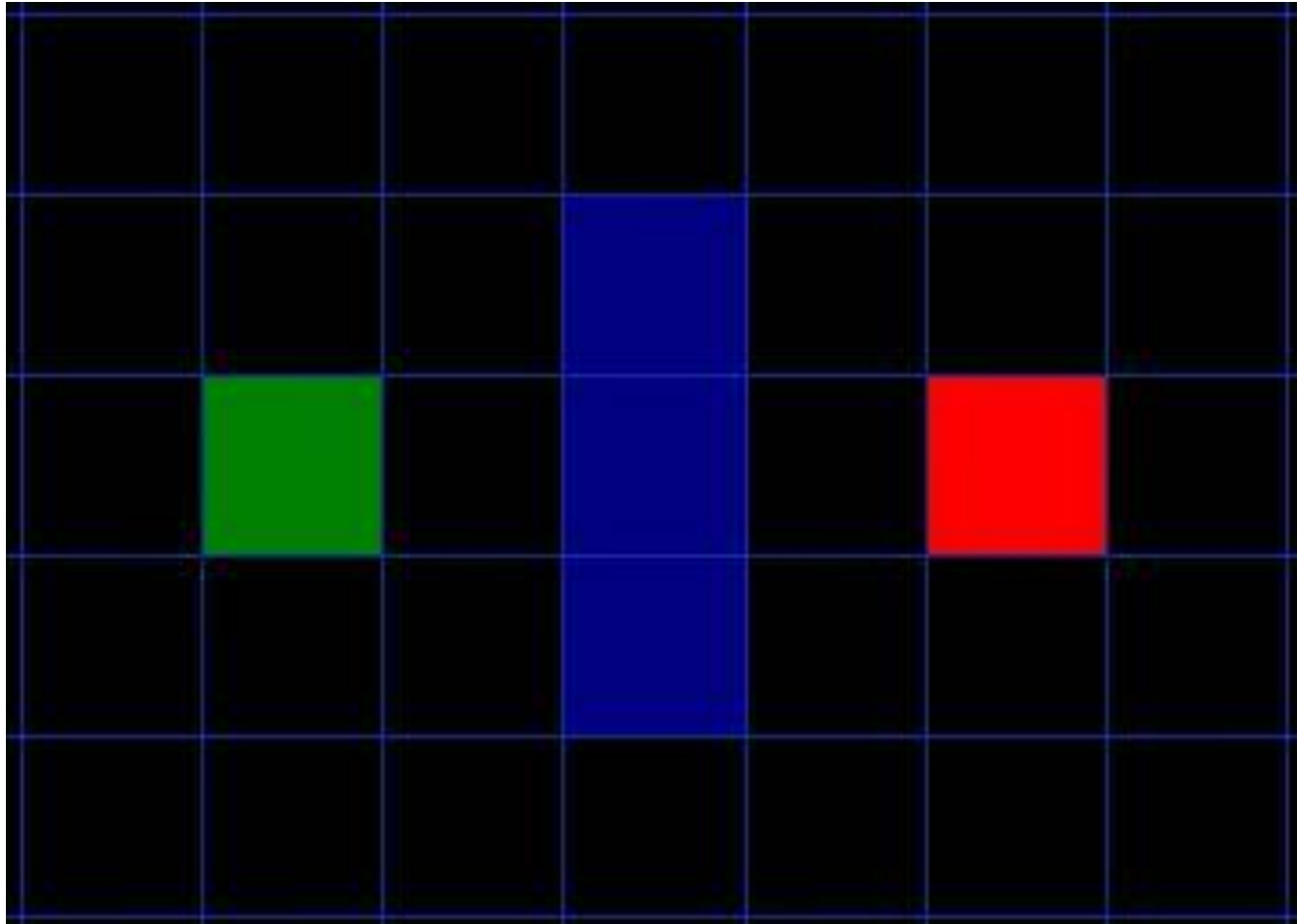
- Given
 - ☒ terrain
 - ☒ starting position
 - ☒ goal position
- Find shortest path
 - ☒ span tree
 - calculate new position according to possible movement
 - calculate $f = g + h$
 - g ... cost from starting position (covered distance)
 - h ... heuristic: estimated distance to goal
 - span tree from new position
 - stop
 - if shortest path found
 - depth or time limit reached
 - ☒ Move to first position on calculated shortest path
- Admissibility Theorem
 - ☒ h must be less or equal to the real rest-distance!
 - ☒ for our example: h calculated by the manhattan distance



Pathfinding with A*



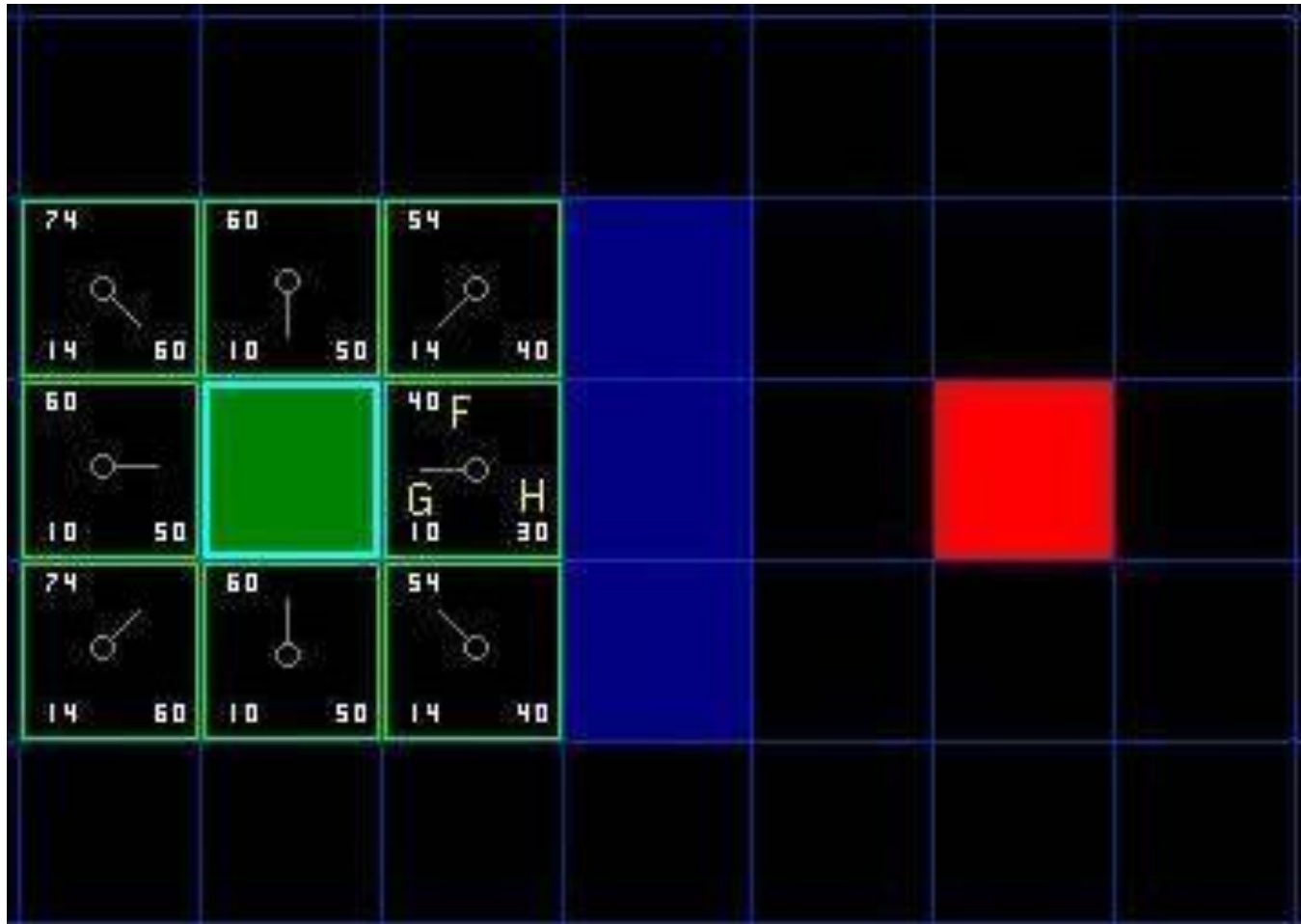
<http://www.uni-klu.ac.at>



Pathfinding with A*



<http://www.uni-klu.ac.at>

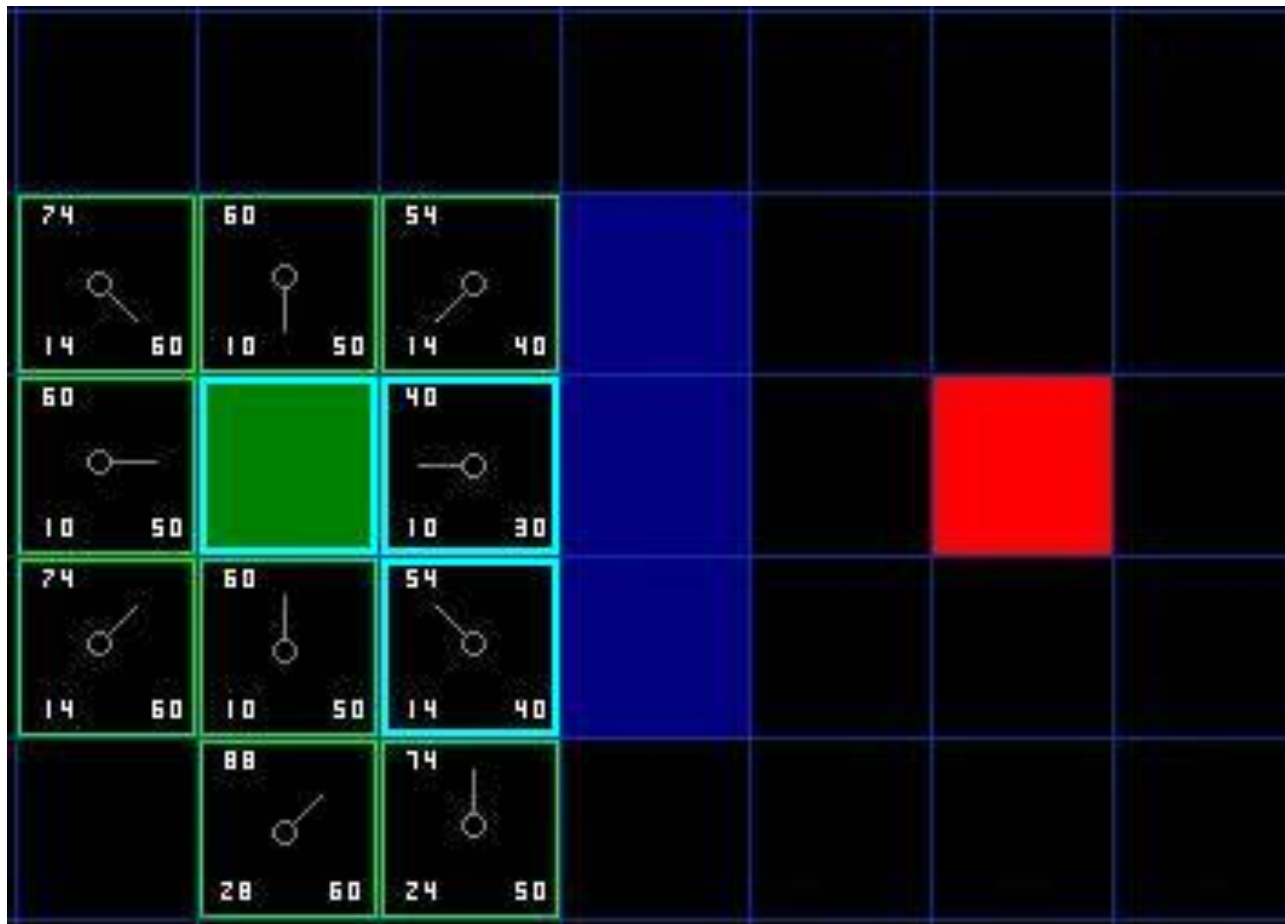


Movement cost:
horizontal 10
vertical 10
diagonal 14

Pathfinding with A*



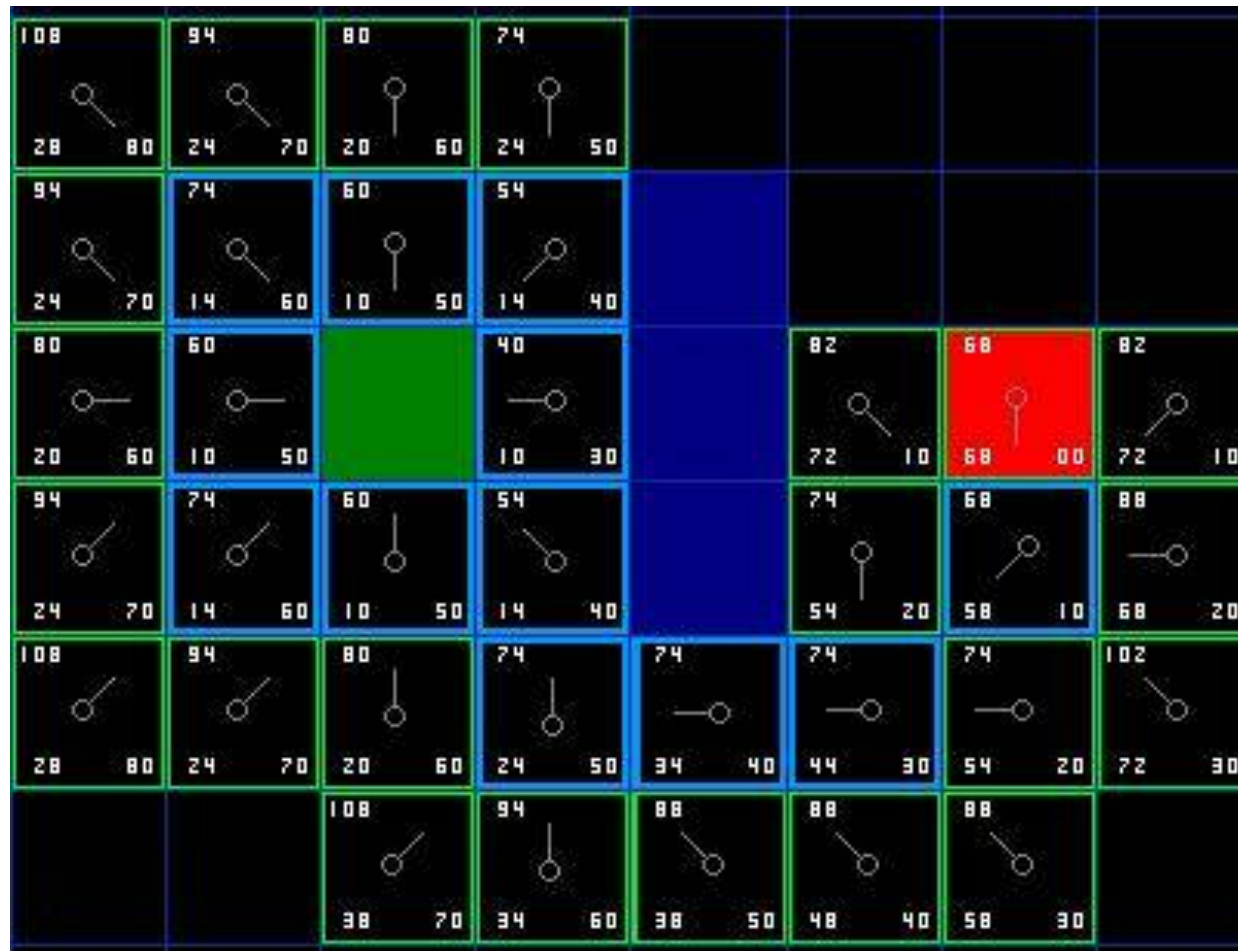
<http://www.uni-klu.ac.at>



Pathfinding with A*



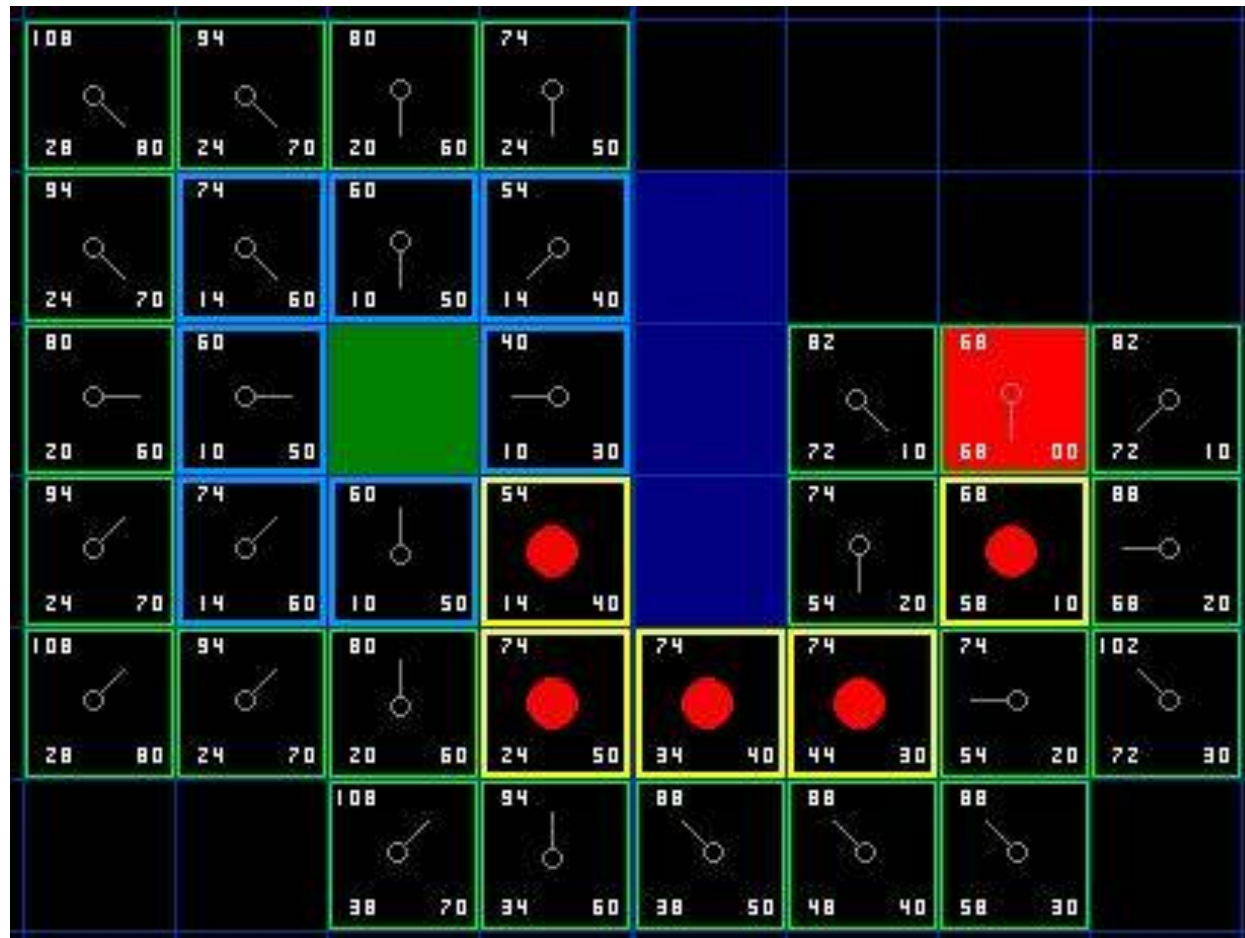
<http://www.uni-klu.ac.at>



Pathfinding with A*



<http://www.uni-klu.ac.at>



A* Performance



<http://www.uni-klu.ac.at>



Performance problem

- for many bots and/or huge search spaces A* might slow down the game










Performance improvement

- recalculate only if goal-position changes
- do not recalculate too often (apply wrong direction for some time)
- limit search time (if not finished: choose currently best direction)
- if reusable: store and reuse search results
- divide search (solve subgoals)
 - use build time information (graph with selected locations on the map)
 - if it is known that for path from A to Z, one must pass O, then calculate A to O first

Some Thoughts on Tactical AI



<http://www.uni-klu.ac.at>

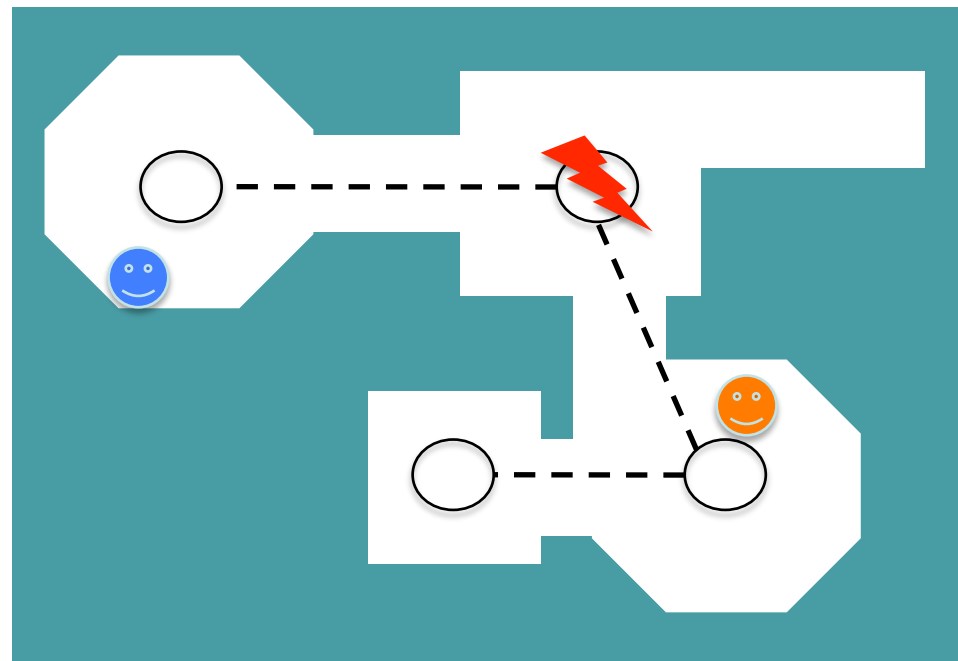
- Rational movement
 -  avoid cones of fire
 -  do not move in „lines of sight“
 -  seek cover: move to cover locations
- Choke point analysis
 -  choke-point = points where a player must go through
 -  use navigation graph to find choke points
 - graph expresses connections between „rooms“ and „corridors“
 -  go to choke points
 -  wait for player, prepare, and ambush

Some Thoughts on Tactical AI



<http://www.uni-klu.ac.at>

- Choke points



Some thoughts on Tactical AI

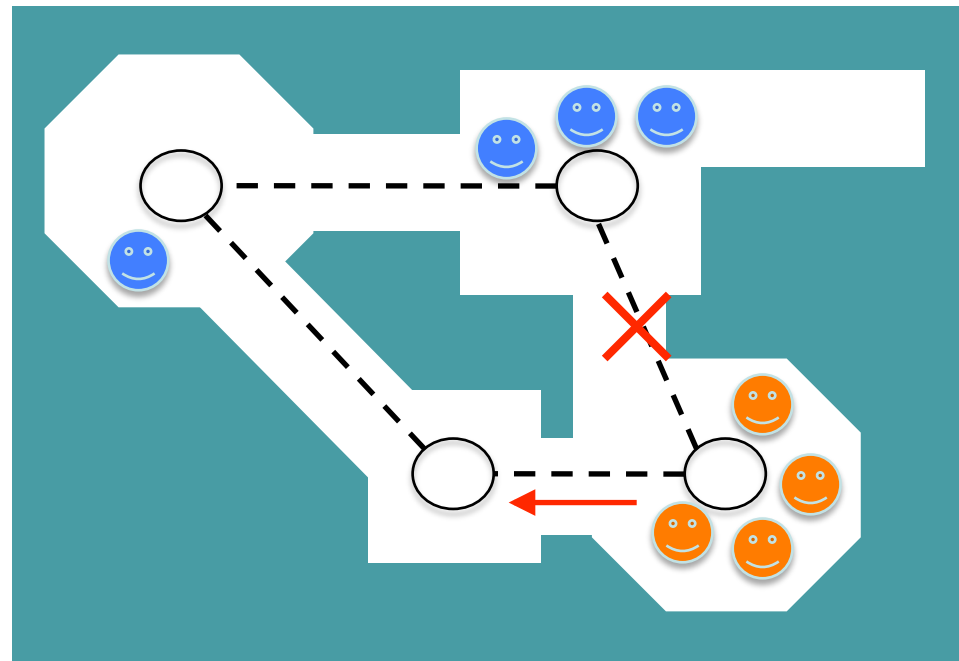


<http://www.uni-klu.ac.at>

- Influence Maps

- dominance of teams per area

- needed for tactical reasoning



Some thoughts on Tactical AI



<http://www.uni-klu.ac.at>

- Commander AI

- ☒ know goals

- conquer point A, destroy Player X, ...

- ☒ update and assess current situation

- enemy unit types, area domination, no. units available, required resources, unknown areas, ...

- ☒ select (pre-defined) tactics

- e.g., scout/gather/construct, „tank-rush“ (in C&C)

- ☒ planning

- determine necessary actions (to achieve the goal)

- ☒ perform actions

- send scouts, produce needed units, command units to hot spots/choke points/under-dominated areas, ...

- ☒ etc.

Further AI Algorithms



<http://www.uni-klu.ac.at>

Planning Algorithms

- find a sequence of actions to achieve a goal
 - example: monkey and banana
 - see Means Ends Analysis
 - see Goal Regression Planning

Constraint Satisfaction

- find a solution (assignment of variables) within given constraints
 - example: solve a Sudoku
 - see Constraint Satisfaction Problem

Further AI Algorithms



<http://www.uni-klu.ac.at>

Learning

- adapt behaviour based on prior experience
 - example: best reaction on typical player movement
 - see Learning with decision trees

Game Playing

- find best next move
 - example: turn-based two player board games
 - see Min-Max Search
 - see Alpha-Beta Search