

Game Technology

Lecture 13 – 30.01.2018
Artificial Intelligence in Games



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Black & White (2001)

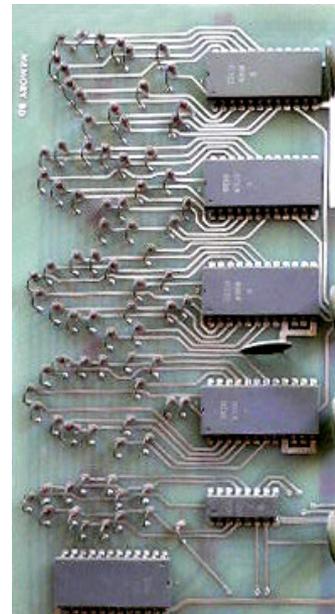
Dipl.-Inf. Robert Konrad
Polona Caserman, M.Sc.

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Introduction

Computer Space (1971)

- Joel Bushnell, Ted Dabney
- Players control a spaceship
- AI UFOs
 - Fire into the quadrant in which the player spaceship is



Introduction

Pac Man (1980)

- Tōru Iwatani
- Four different ghost „personalities“

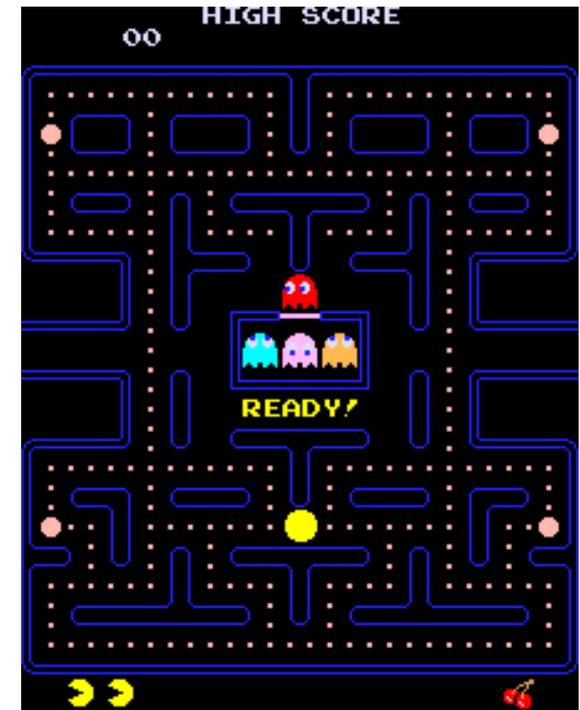
CHARACTER	NICKNAME	CHARACTER	NICKNAME
 - SHADOW	„BLINKY“	 DIKAKE - - -	„AKABEI“
 - SPEEDY	„PINKY“	 MACHIBUSE - -	„PINKY“
 - BASHFUL	„INKY“	 KIMAGURE - -	„ADSUKE“
 - POKEY	„CLYDE“	 OTOBOKE - - -	„GUZUTA“

E.g. red ghost

- „Shadow“
 - Target is always Pac-Man (vs. where he is going to)

More info

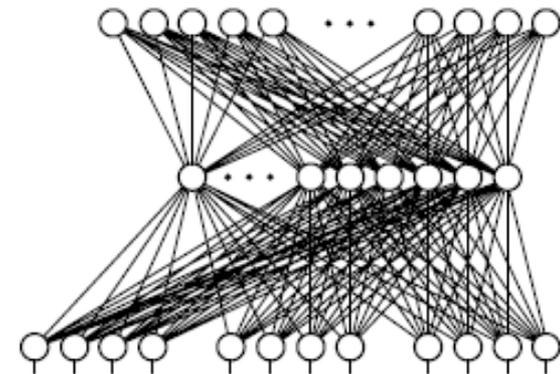
- <http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior>



Introduction

Creatures (1996)

- Steve Grand
- Train and breed creatures
- Used [neural networks](#) for AI



Introduction

Today

- AI central component in almost all games
- Gets more importance (= computation time), is integrated into game design decisions
- AI engines and AI middleware



Model

AI gets its
information

AI gets given processor time

Execution management

World interface

Group AI

Strategy

Character AI

Decision making

Movement

Content creation

Scripting

AI has implications
for related technologies

Animation

Physics

AI gets turned into on-screen action

Complexity

Simple things can look good

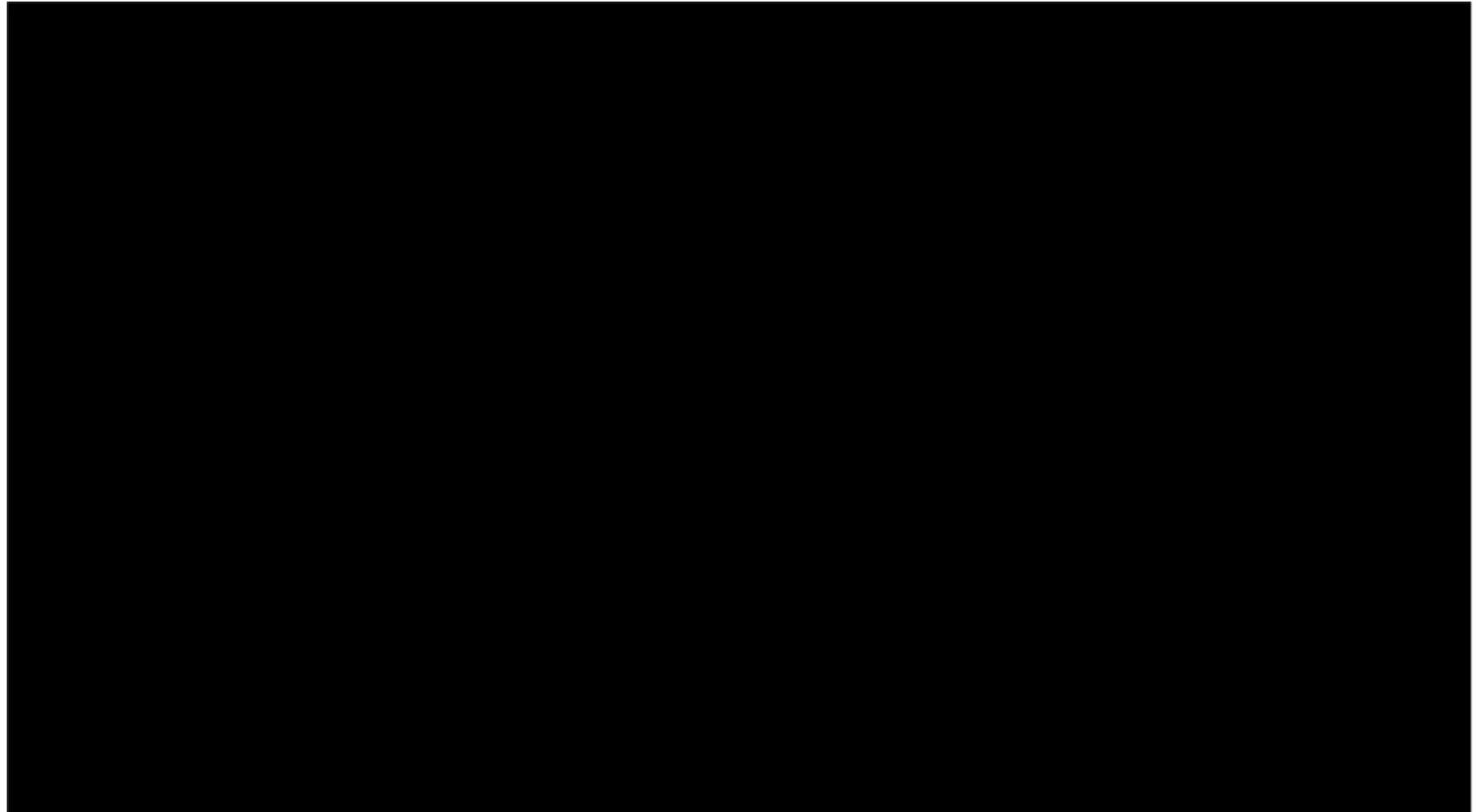
- ELIZA (1966)
 - > Hello. My name is Eliza. How may I help you?
 - * I want to give a lecture on AI for games.
 - > Why do you want to give a lecture on AI for games?
 - * It is part of a lecture series on game technology.
 - > Say, do you have any psychological problems?
 - * Maybe...
 - > You don't seem quite certain.

Pac Man

CHARACTER / NICKNAME	CHARACTER / NICKNAME
 - SHADOW "BLINKY"	 OIKAKE - - - "AKABEI"
 - SPEEDY "PINKY"	 MACHIBUSE - - "PINKY"
 - BASHFUL "INKY"	 KIMAGURE - - "AOSUKE"
 - POKEY "CLYDE"	 OTOBOKE - - - "GUZUTA"

Complexity

More complex things can easily look bad



Kinds of AI in games

„Hacks“

- Make random adjustments to movement
- Add an animation to show the emotional state instead of simulating the resulting actions

Heuristics

- Instead of pathfinding, turn and go towards the goal
- In strategy games, assign values to units instead of computing their capabilities

Algorithms

- Re-usable approaches to AI
- E.g. movement, pathfinding, decision making

Movement

AI gets its
information

AI gets given processor time

Execution management

World interface

Group AI

Strategy

Character AI

Decision making

Movement

Content creation

Scripting

AI has implications
for related technologies

Animation

Physics

AI gets turned into on-screen action

Movement

Input

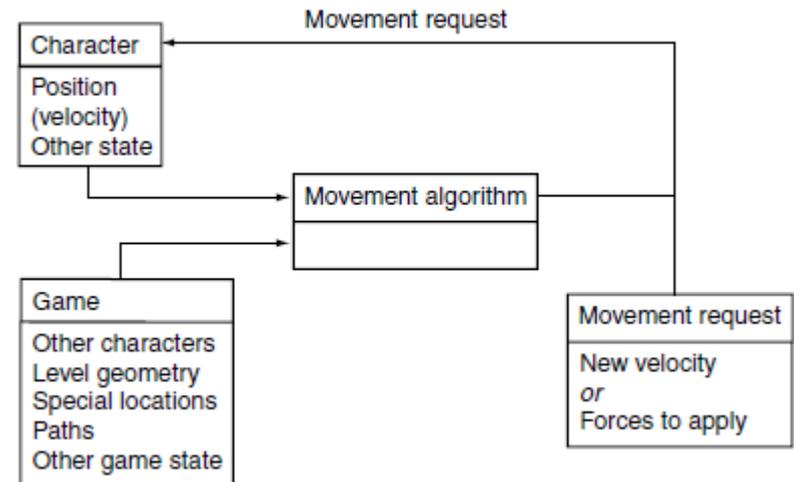
- Character's current state
- Game information, e.g. the current goal

Output

- Movement requests

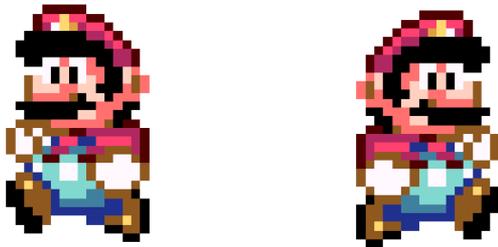
Kinematic vs. Dynamic movement

- Kinematic – Update the velocity directly
- Dynamic – Take into account velocities, change via accelerations (see physics lectures)

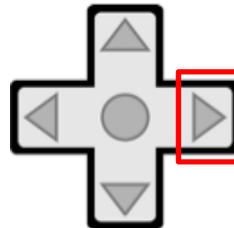


Movement comparison

Kinematic



Dynamic



Dimensionality

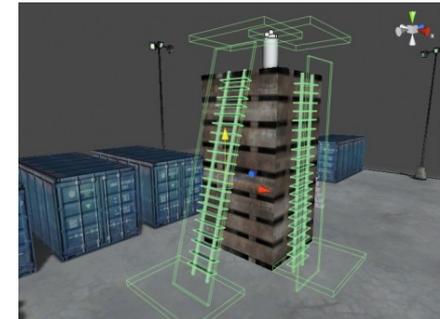
2D

- Easiest
- Often applicable to 3D rendered games
- 2D-Position + Orientation



2.5D

- 3D-Position + 2D orientation
- E.g. for characters walking and jumping/climbing ledges



3D

- Required if motion is truly 3D
- E.g. flight simulators



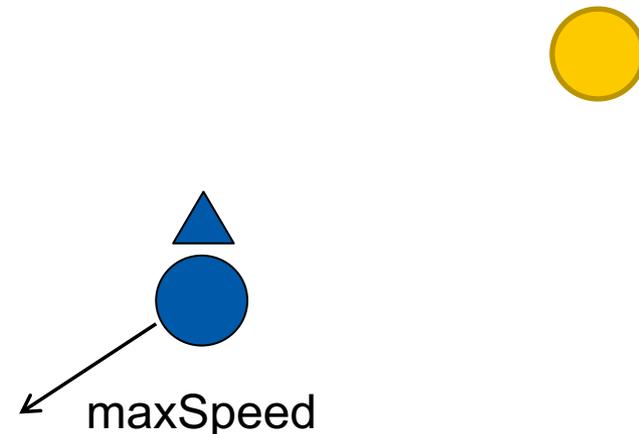
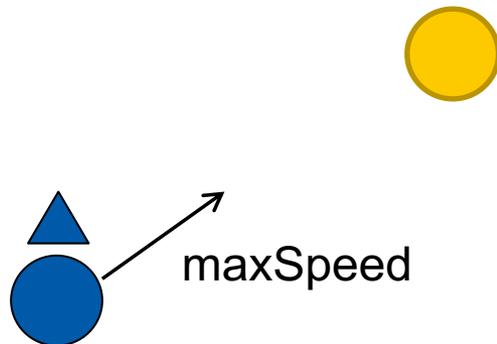
Simple movement behaviours (kinematic)

Seek

- Compare to target and current position
- Output a velocity going there directly at maximum speed

Flee

- Reverse direction from Seek



Simple movement behaviours

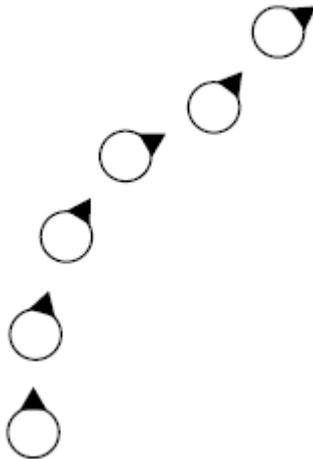
Arrive

- Define a radius when the goal is reached
- Slow down as the character gets closer
- Outside of radius
 - Calculate speed to reach target in fixed amount of time
 - Clip to maximum speed
 - → Will slow down as it gets closer
- Inside the radius
 - No movement (probably other behaviour appropriate, e.g. attacking)

Simple movement behaviours

Wander

- Change the orientation randomly in a small range each time the behaviour is called

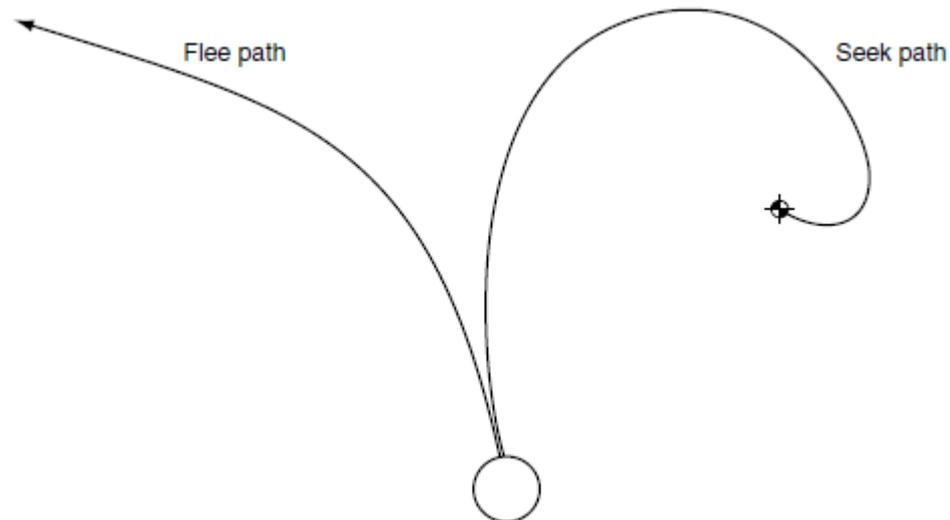


Dynamic Steering

Take into account rotation and velocity

Seek and Flee

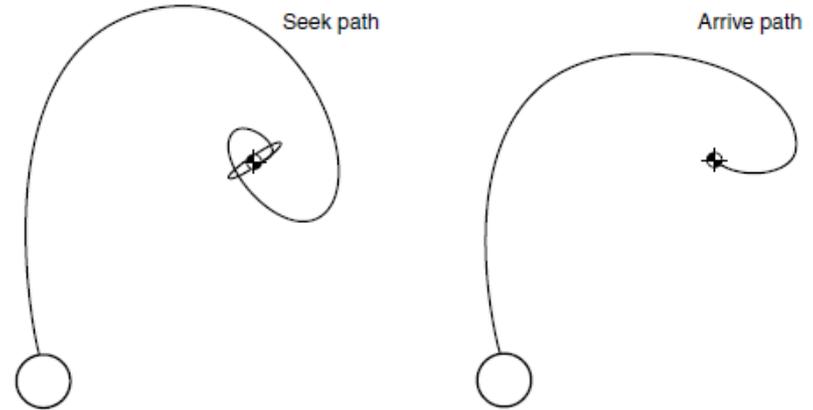
- As before: Choose direction to goal
- Output an acceleration in that direction
- Maximum speed: Either via clipping or via drag



Dynamic Steering

Arrive

- Seek can overshoot and oscillate
- Define a radius in which the character should slow down
- Slowing down is done by calculating an acceleration that leads to the target velocity



Align

- Align to the same orientation as the target

Velocity matching

- Match the velocity of the target

Delegated behaviours

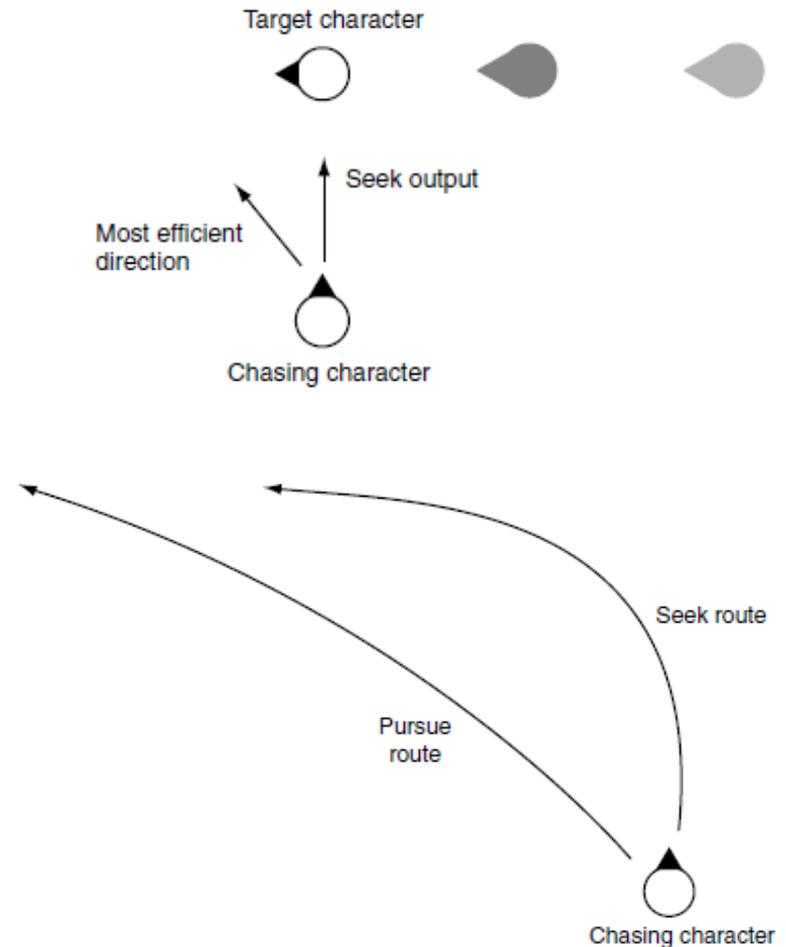
Based on the primitive behaviours we have seen until now

Pursue

- Calculate where the target will be if it continues to move with the current velocity
- A Steer behaviour is used to move towards this new target

Evade

- Opposite of pursue

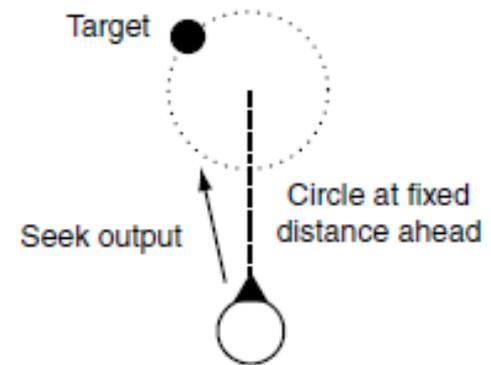


Delegated behaviours

Face

- Align to the target

Looking where you are going

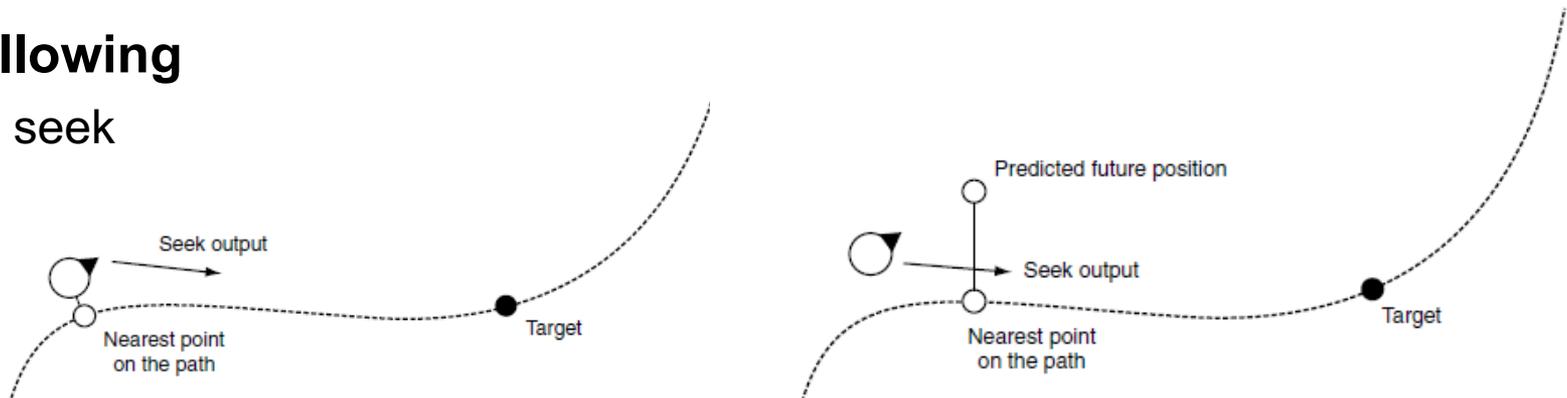


Wander

- Uses Seek

Path following

- Uses seek



Obstacle and Wall Avoidance

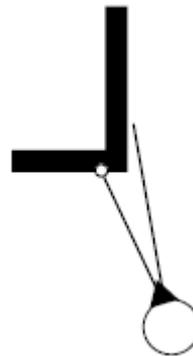
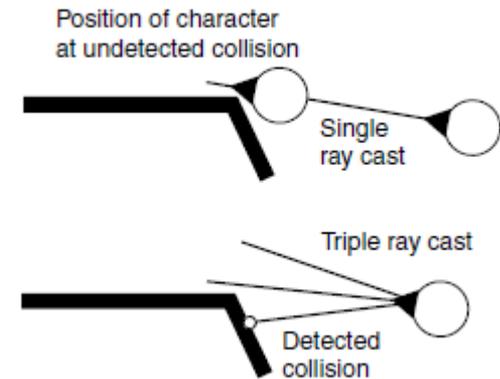
Similar, but distinct from collision detection in physics

Basic mechanism: Use ray casts

- One ray cast often not enough
- Use multiple casts or swept volumes

Handling corners

- Deadlock even for several rays
- Possible solutions
 - Adaptive ray fan sizes
 - Special-case code for corners



Combining steering behaviours

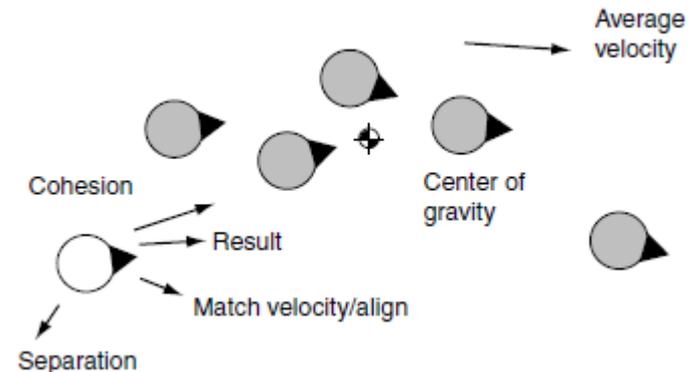
(Weighted) blending

- Take the (weighted) average of basic steerings
- Good weights? Always the same weights?

Flocking

- Simulation of fish schools, bird flocks, ...
- Simulated entities referred to as boids
- Created by Craig Reynolds, 1987

- Blend of 3 steering factors
 - Separation (move away from close boids)
 - Move in the same way as the flock (match avg. velocity and orientation)
 - Cohesion (steer towards center of the flock)



Combining steering behaviours

Equilibria

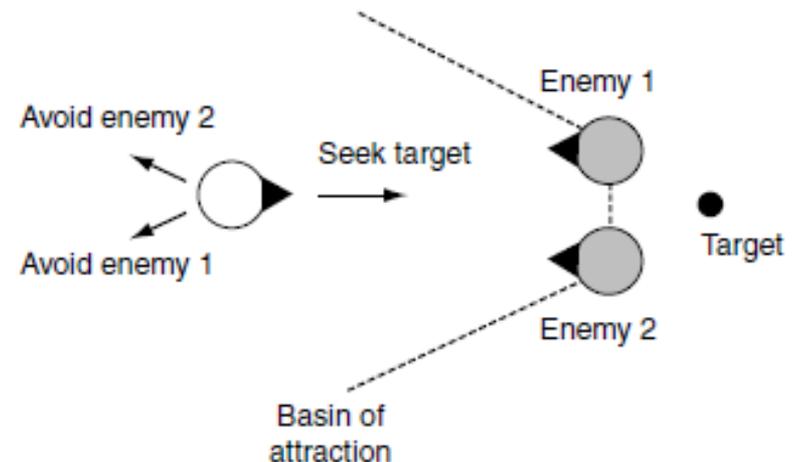
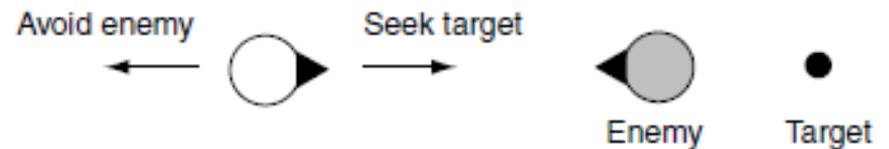
- Conflicting steering behaviours lead to deadlocks

Unstable equilibria

- Situation is inpropable
- Basin of attraction is small

Stable equilibria

- Less likely to be broken by numerical inaccuracies

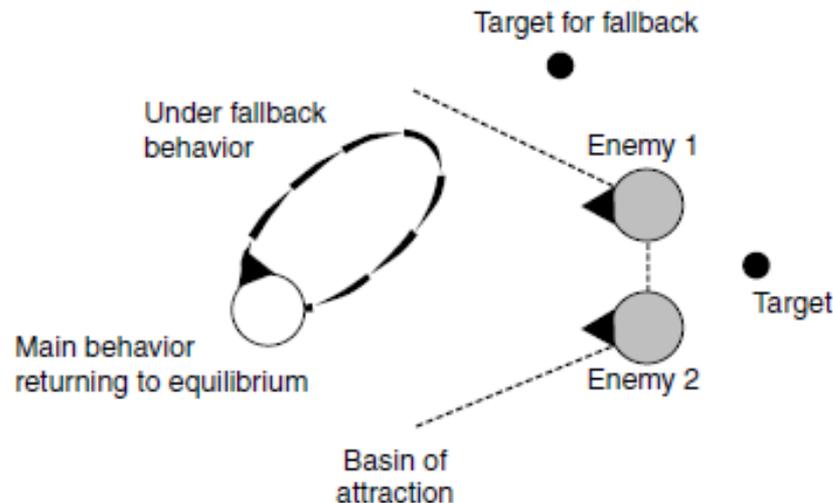


Priorities

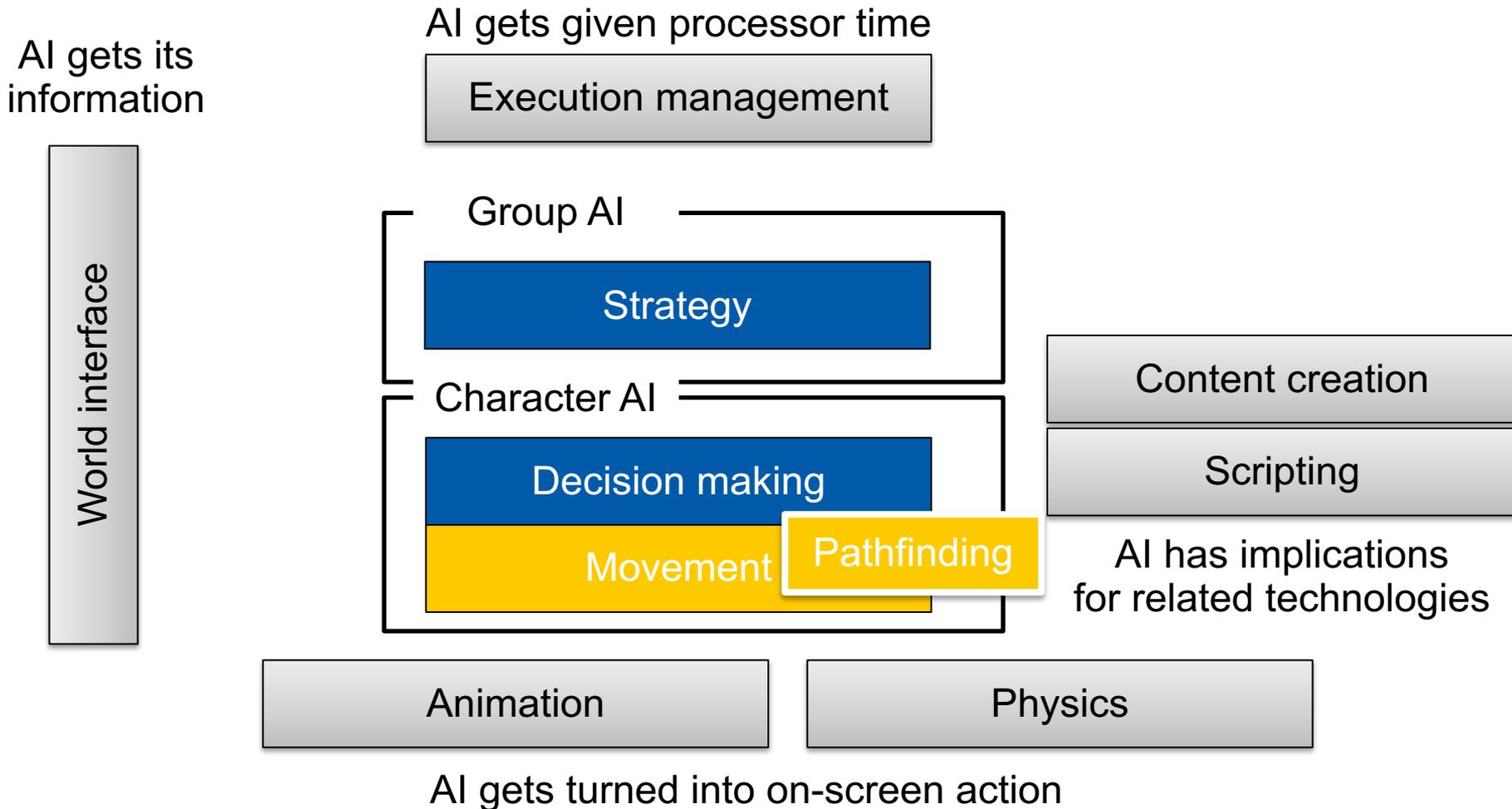
Define priorities for steering behaviours

E.g. collision avoidance has priority, if it has an output, it is taken

Can still oscillate for stable equilibria



Pathfinding



Pathfinding

Higher-level movement planning

- Generate a path for an AI to follow

Based on a graph

- Basic connectivity: point B can be reached from A if there is a connection between them
- Weights/costs: How „hard“ is it to get from A to B?
- Directions: In which direction is the connection? (E.g. jumping down an unclimbable ledge)

Finding a path

A*-Algorithm is the prevalent solution

Not required to know in this lecture

- Requires an adjacent-list and connections between nodes (e.g. tiles)

calculate cost of CurrentNode

add CurrentNode to the Open-Node-List

while (Open-Node-List not empty)

 CurrentNode = node from Open-Node-List with lowest costs

if (CurrentNode == TargetNode)

 Path completed

else

for each AdjacentNode next to CurrentNode

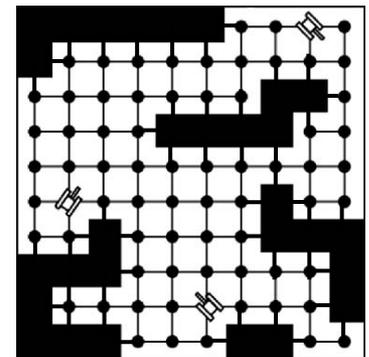
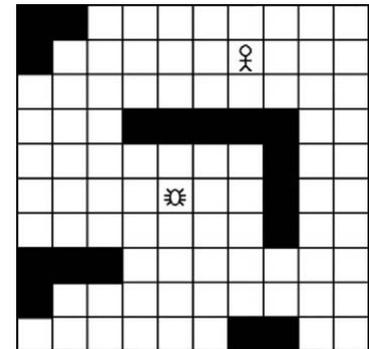
if (AdjacentNode **not** in Open-Node-List **and**
 AdjacentNode **not** in Visited-Node-List **and**
 AdjacentNode **is not** an obstacle)

 calculate cost of AdjacentNode

 link AdjacentNode to CurrentNode

 add AdjacentNode to Open-Node-List

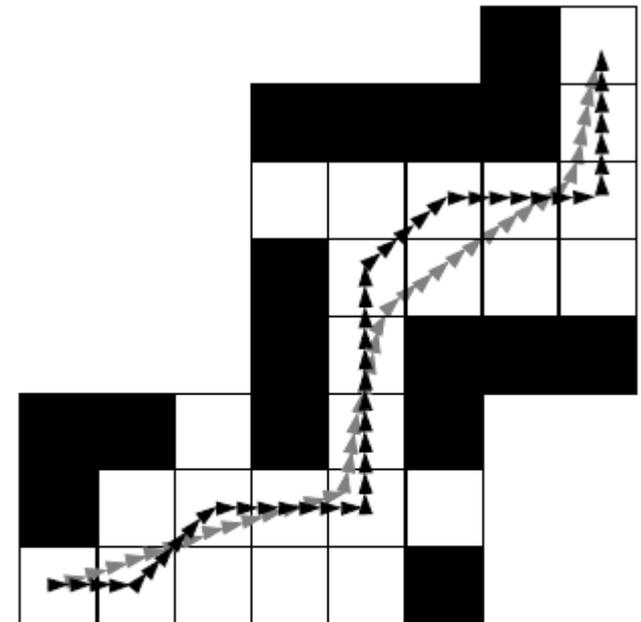
 add CurrentNode to Visited-Node-List



Tile-based

Well-suited if game world is
already represented by tiles (e.g.
heightmap)

Plans can be blocky due to
rectangular nature

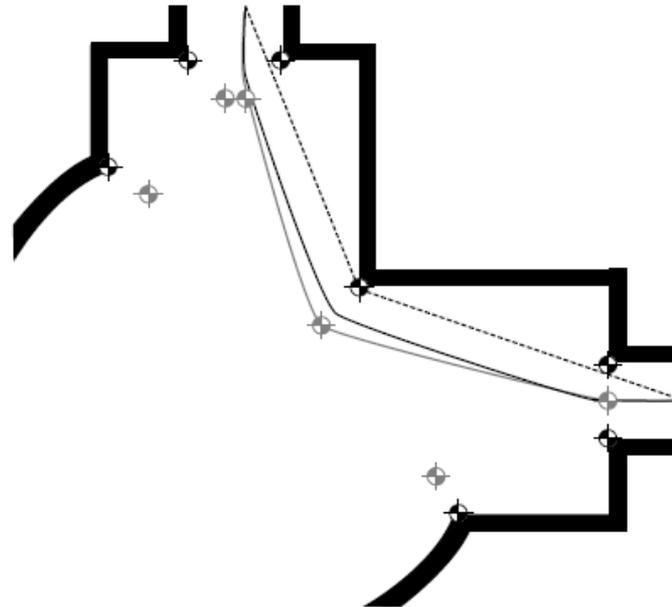


Key

-  Output blocky plan
-  Ideal direct plan

Points of visibility

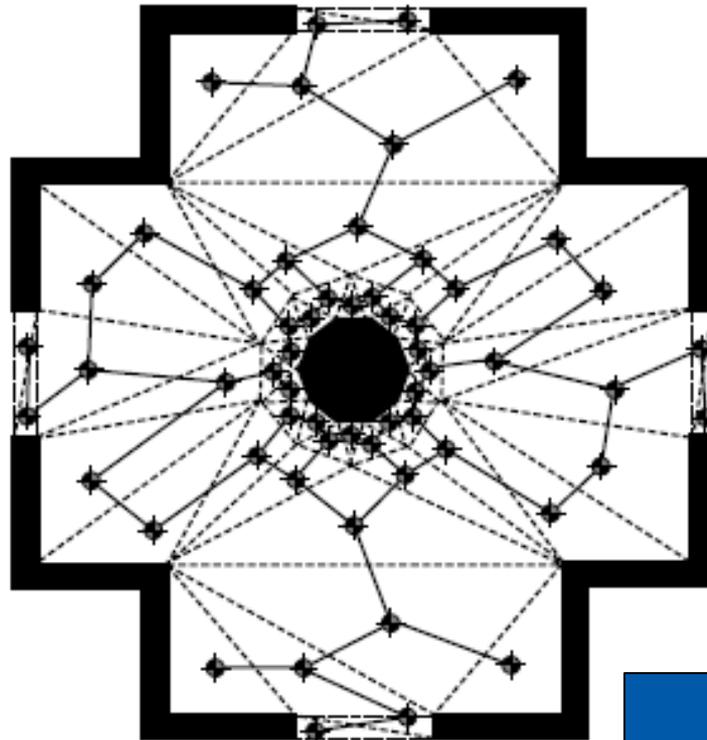
Based on the edges of the world



- Key**
- Optimal path for zero-width character
 - Path for character with width
 - Path using vertex offsets
 - ◆ Original characteristic points at vertices
 - ◆ Offset characteristic points

Navigation mesh

Use the mesh of the floor as the basis



Key

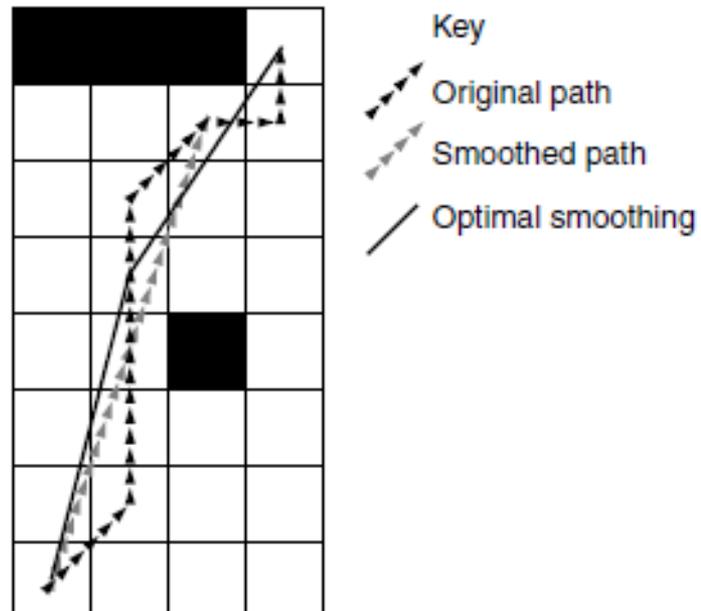
- Edge of a floor polygon
- Connection between nodes

Popular library:
Recast
<http://masagroup.github.io/recastdetour/>

Path smoothing

Filter paths to remove sharp edges

- E.g. by checking connectivity between waypoints by ray casts

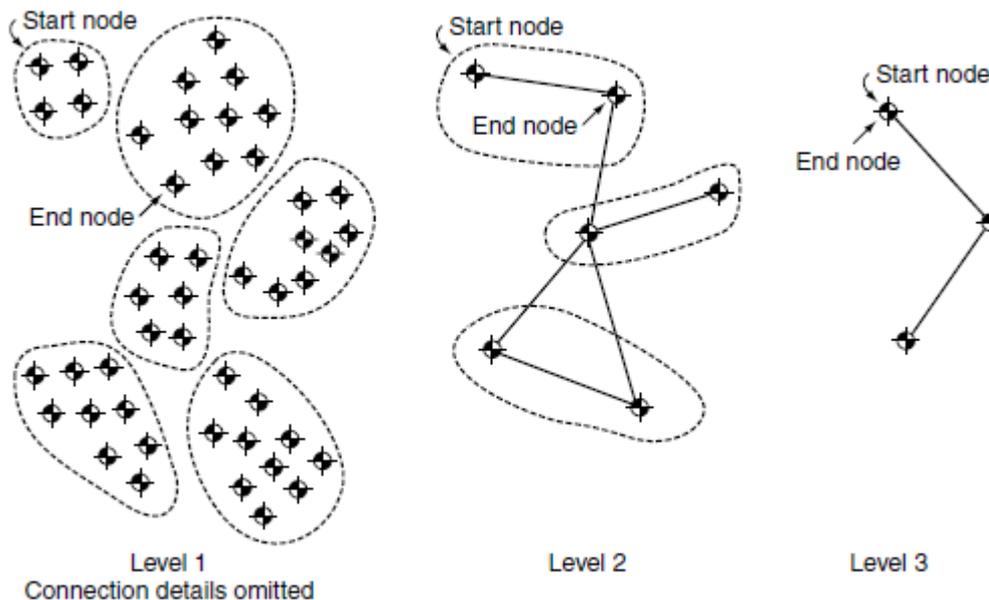


Hierarchical pathfinding

Cluster waypoints (e.g. by rooms, areas, ...)

Compute paths between the clusters

Only when the character needs to traverse a cluster, find a path through it



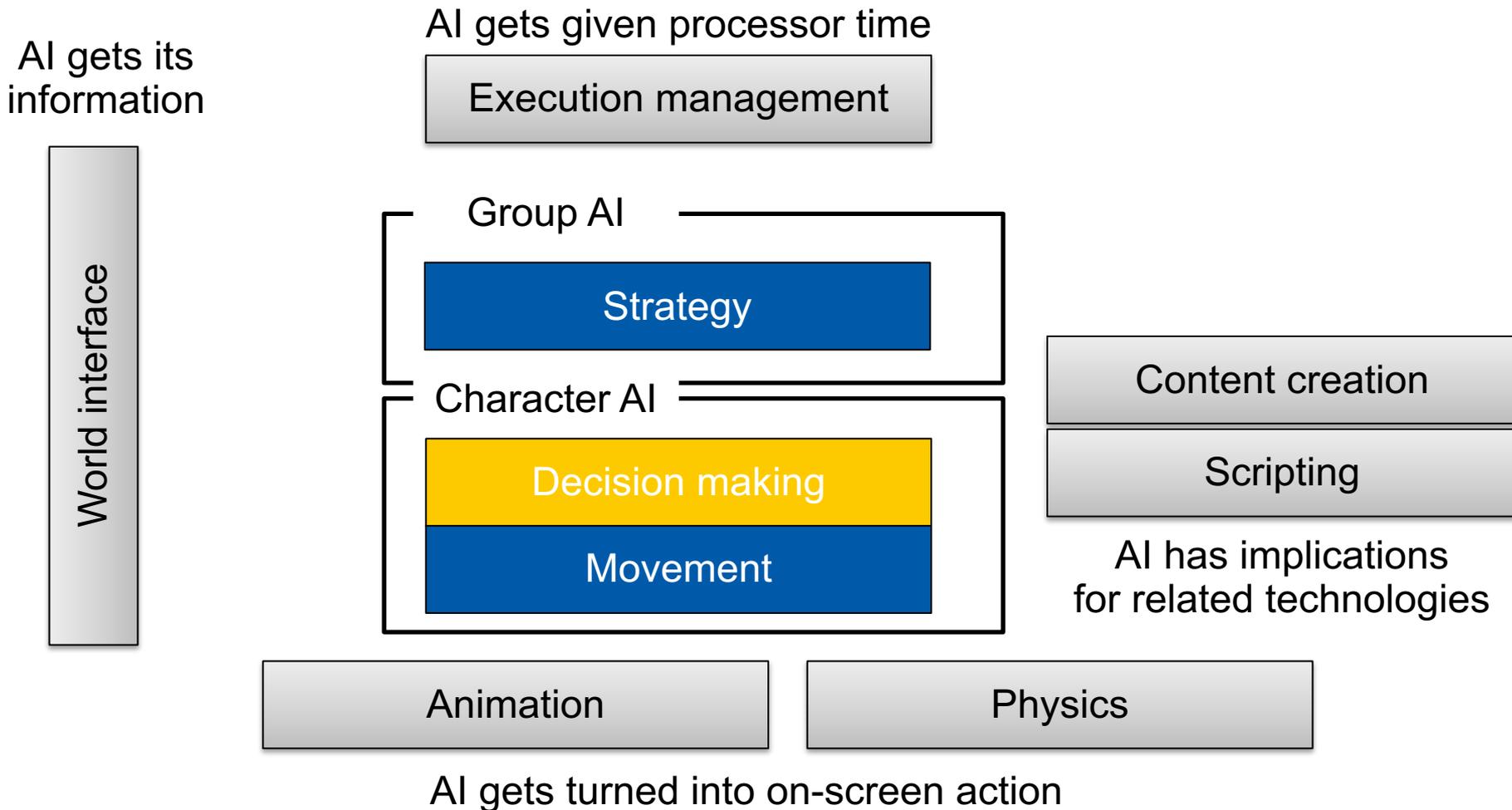
Decision Making



TECHNISCHE
UNIVERSITÄT
DARMSTADT



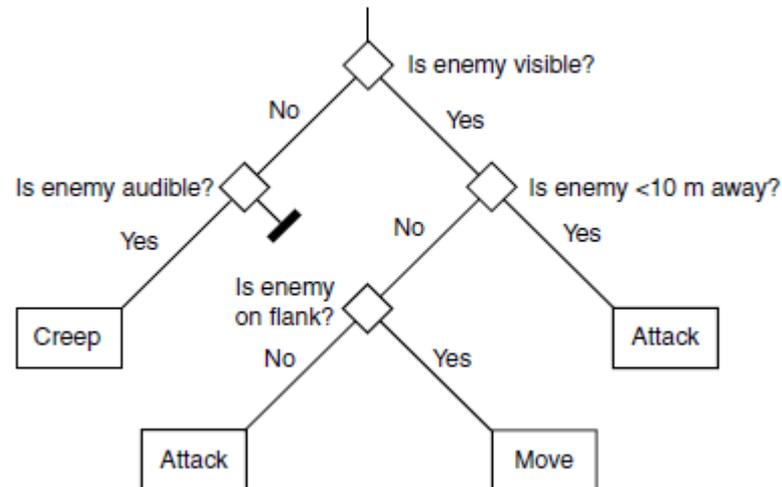
Decision Making



Decision Trees

Structure decisions as a series of conditions and actions as leaves

- Created by designer
- Condition nodes can have more than 2 branches
- Performance
 - Try to keep trees balanced
 - Use computation-heavy conditions further down in the tree



Random Decision Trees

We do not want to have the same (predictable) behaviour each time

Add randomness to the graph

- Watch out for oscillations
- Save previous choices and only change after a timeout

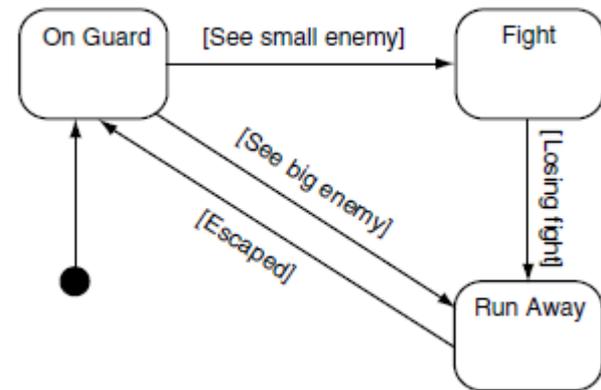
State Machines

Generalization of the decision trees

- States govern the behaviour of the agent
- Transitions are triggered if the condition matches

Created by designers

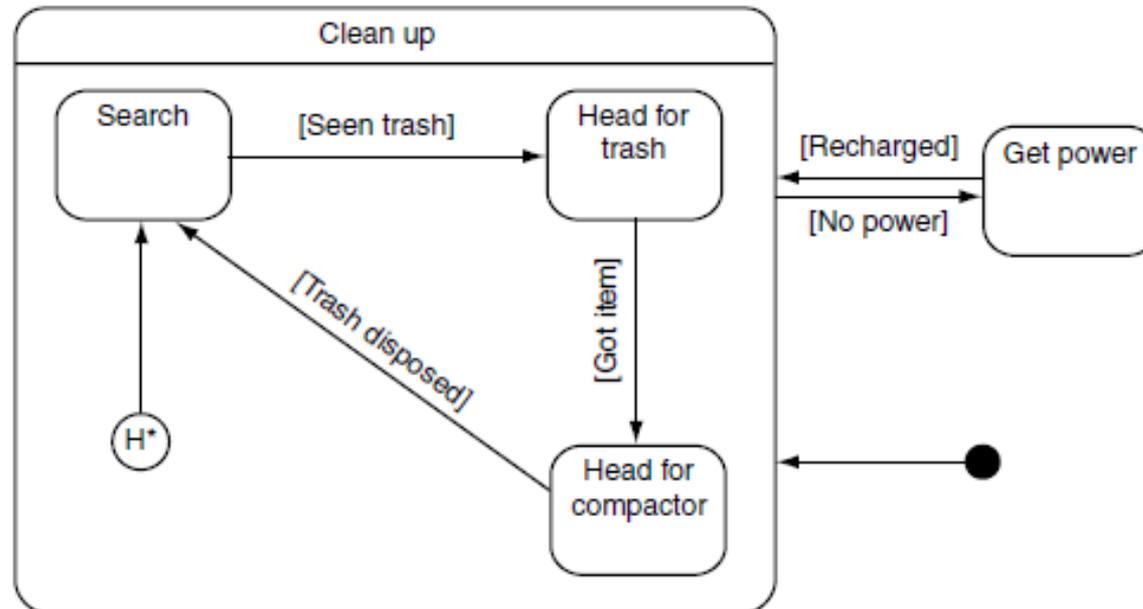
Hard-coded vs. flexible



Hierarchical State Machines

Problem: „Alarm“ states

- Normal behaviour is interrupted, e.g. by seeing the player
- Can be from any of the normal states → many transitions to alarm state
- Later return to normal state



Behavior Trees

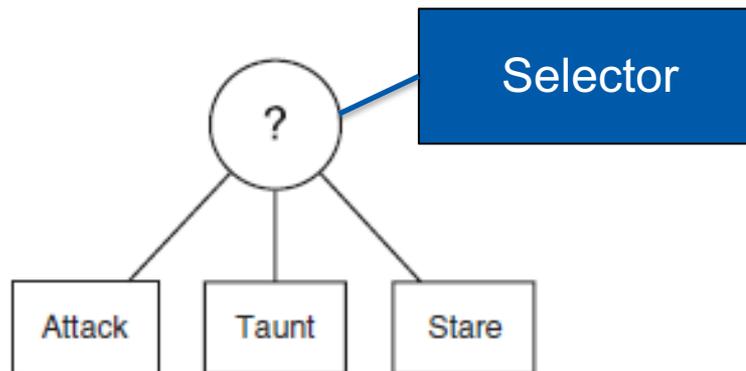
Combination of other techniques

Easy to understand and create for non-programmers (?)

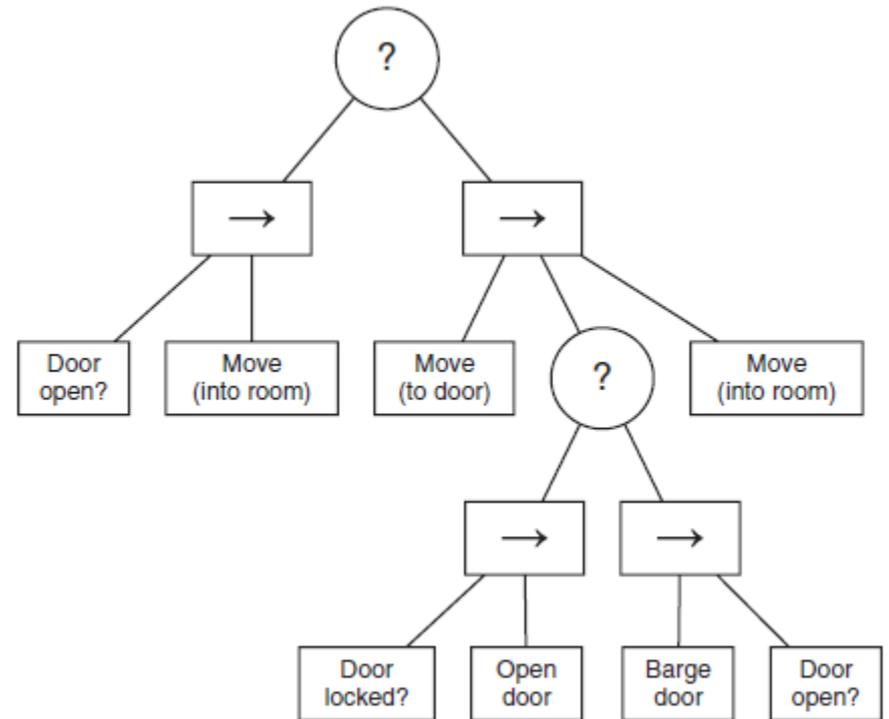
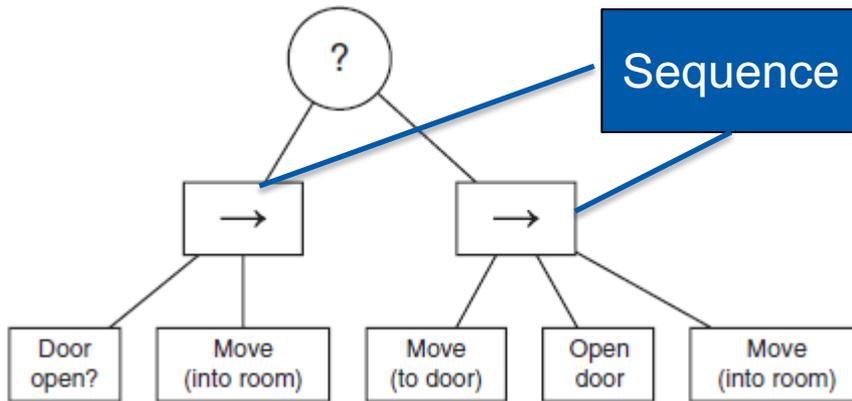
- First description in Halo 2 (2004)

Tasks instead of states

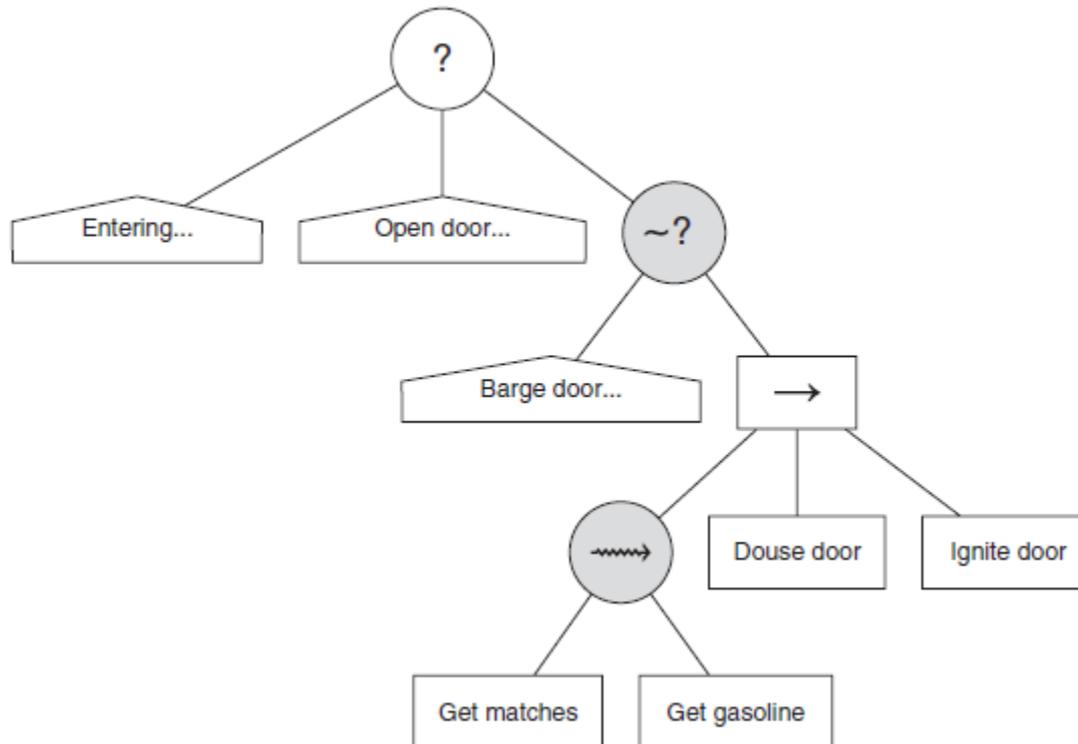
- Conditions
- Actions
- Composites
 - Selector
 - Sequence



Behavior Trees



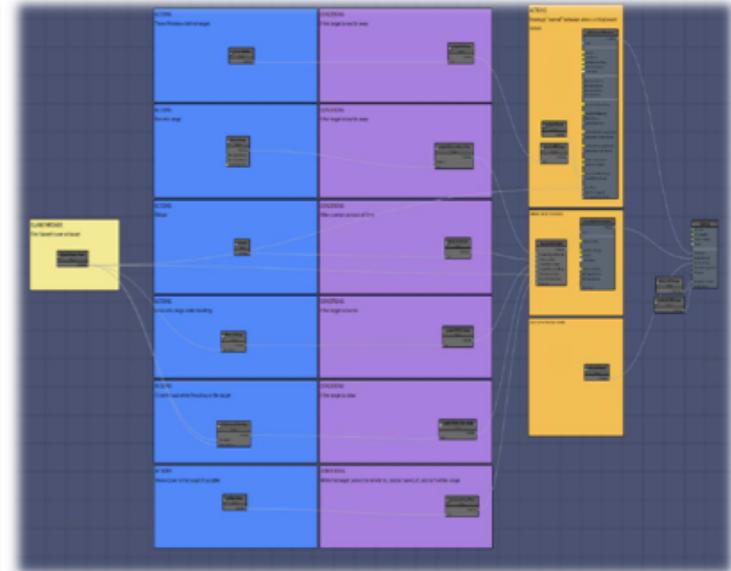
Non-determinism



Behavior Tree shortcomings

Behavior Trees can become hard to manage and understand

Eg. Behavior Tree in „The Division“:



Utility-Based AI

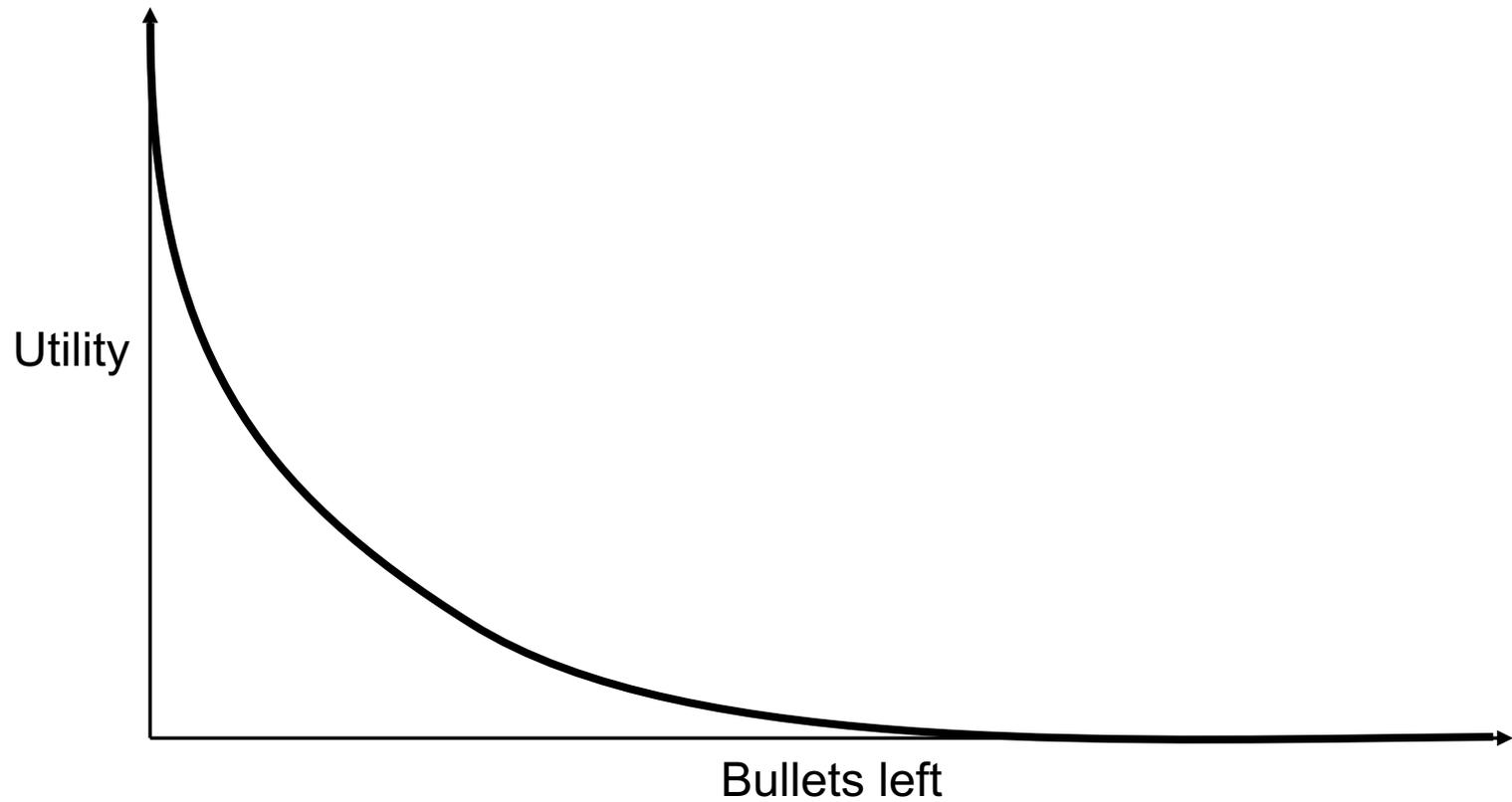
Don't explicitly „script“ AI behavior
Instead, create **options** for AIs

- Patrol
- Sleep
- Attack
- Reload
- ...

Which option to choose?

- The one that is the most useful at the moment → Utility
- E.g. no ammo in magazine and want to shoot → Utility of „Reload“ is high
- E.g. ammo in magazine is full → Utility of „Reload“ is low

Example: Reloading



Considerations

Even if AIs have many options, they need similar checks for most of them

- We can implement re-usable **considerations**

Examples

- Distance to enemy
- Current health
- Current ammo
- ...

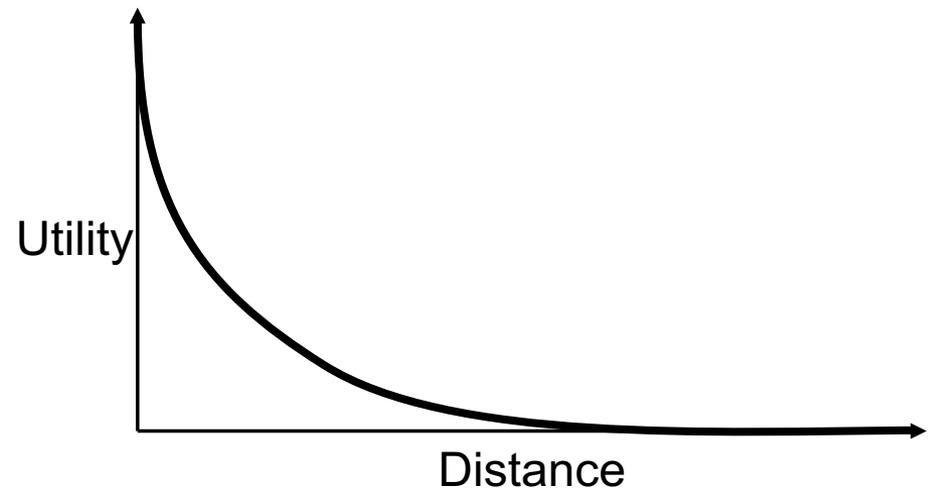
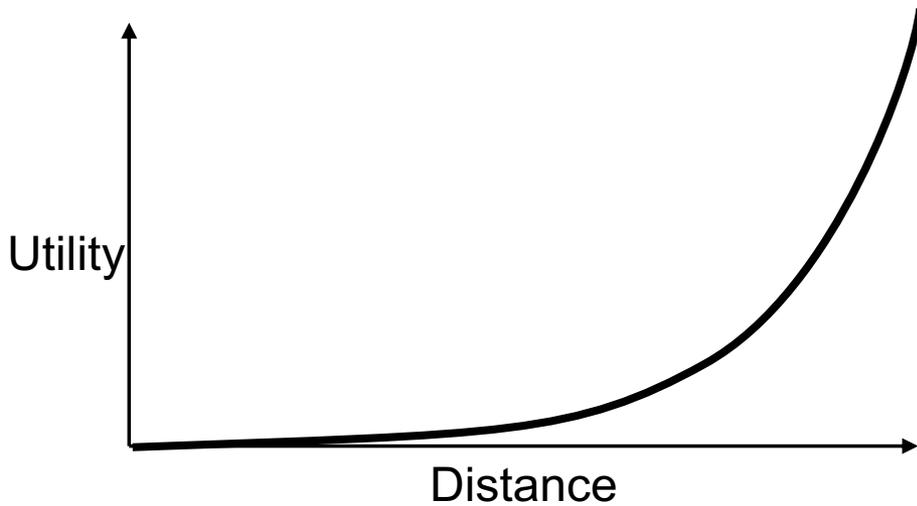
Reflective considerations

- Time since last execution
- Number of times executed
- ...

Example – „Distance to enemy“-consideration

If we want to flee

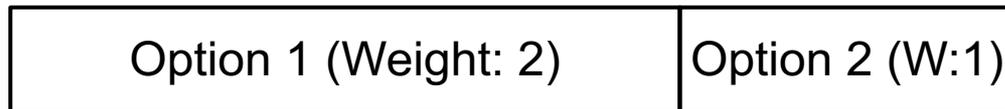
If we want to punch



Choosing an option to use

Choose between different options

- Supply a weight per option
 - Do weight-based random
 - E.g. Option 1 is weighted with weight 2, option 2 with weight 1
- Option 1 will be executed 2/3 of the time



Believable

- More important options are used more often
- Non-deterministic: Not every time the same option

Override Behaviors

Problem with weight-based random: We are allowed to choose bad options, even if their weight is really low

E.g. a grenade is thrown next to an AI

- Throwing it away is a good option
- Having a chat with your teammate is a really bad option

Solutions

- Do not consider options with weight 0 at all
- Define a cutoff: Options further away than x % from the best option will not be considered
- Ranks

Ranks

Let considerations also rank their options

Only consider valid options of the highest overall rank

Handle the grenade: Rank 5

Have a chat with a teammate: Rank 1

If the grenade example happens, the option „Handle the grenade“ gets priority, regardless of the other option’s weights

Goal-oriented behaviour

Goals

- A.k.a. motives or insistence
- Something the character wants to achieve
- E.g. (Sims) Hunger, bathroom distress, ...



The Sims (2000)

Actions

- Anything the character can do
- Depends on the game state (e.g. is the food it wants to eat raw or cooked?)
- Changes the goal levels (eating → less hunger)

Selection

Just choose the current optimum

- Can be sufficient
- But will not lead to longer strategies

Goal: Eat = 4 Goal: Sleep = 3

Action: Get-Raw-Food (Eat – 3)

Action: Get-Snack (Eat – 2)

Action: Sleep-In-Bed (Sleep – 4)

Action: Sleep-On-Sofa (Sleep – 2)

Overall utility

- Add a distress/energy metric
- Choose the option that leaves the character with the best overall result

Timing

- Also keep time in the view
- Short, ineffective vs. Long and effective actions

Planning

Goal-Oriented Action Planning (GOAP)

- Look at sequences of actions
- Keep a world model and extend it into the hypothetical future
- Rank the actions by the overall utility they produce in the long run

Use A* (variants)

- Instead of depth-first search, use a heuristic
- Use A* variants that can cope with infinite graphs
 - E.g. IDA* (Iterative Deepening A*)

Horizon effect

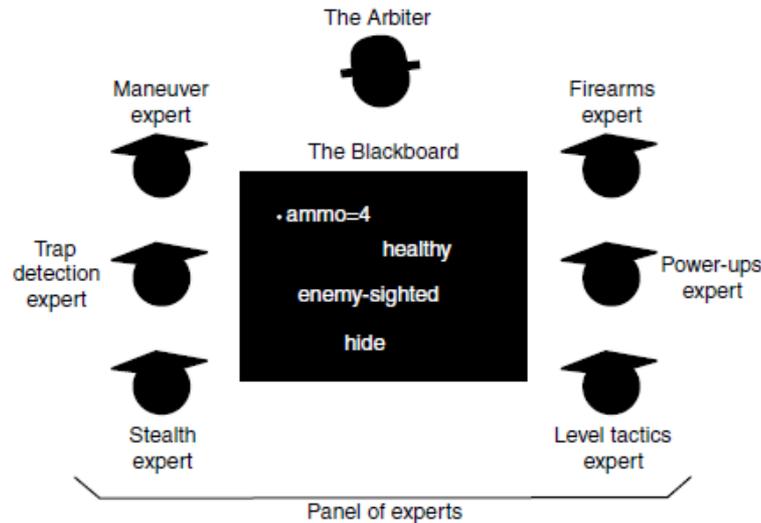
- How many actions do we look ahead?
- What if n actions lead make only slow progress, but $n+1$ (beyond our horizon) is the best option?

Blackboard architectures

Every AI can write to and read from the blackboard
Efficient way of communicating with other AIs

For decision making

- AI components estimate if they can act on the current blackboard state
- When given control by an arbiter, they control the character



Tactical and Strategical AI

AI gets its
information

AI gets given processor time

Execution management

World interface

Group AI

Strategy

Character AI

Decision making

Movement

Content creation

Scripting

AI has implications
for related technologies

Animation

Physics

AI gets turned into on-screen action

Tactical and Strategic AI

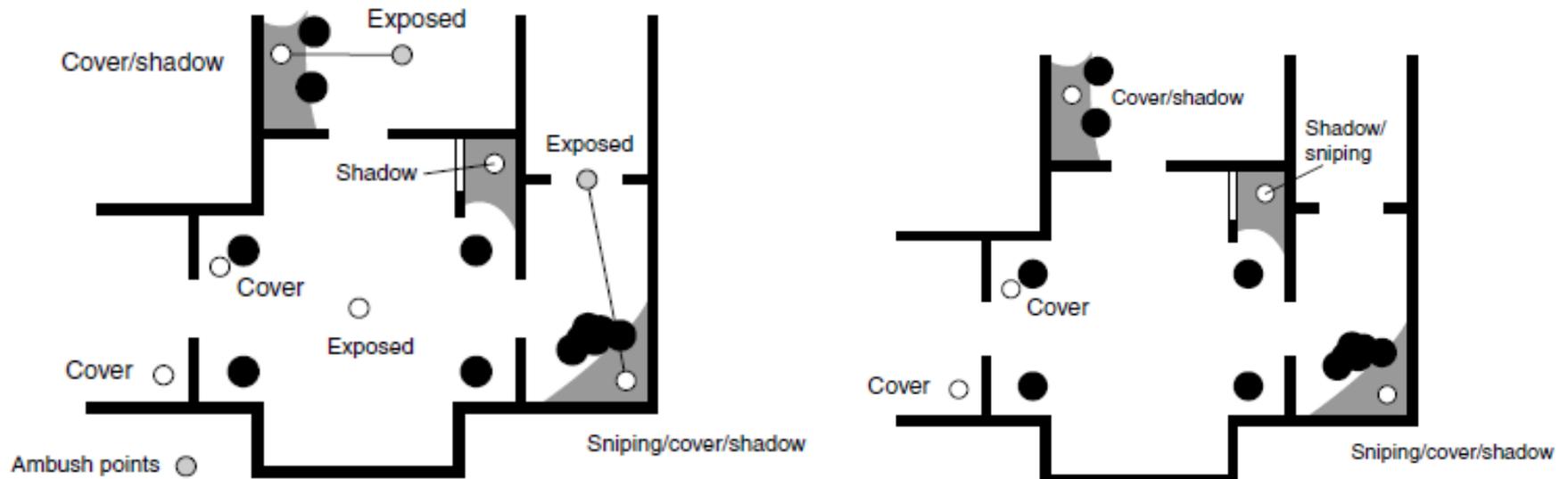


VS



Waypoint Tactics

Find points that are suited for different actions



Tactical Analyses

Analyze the game world

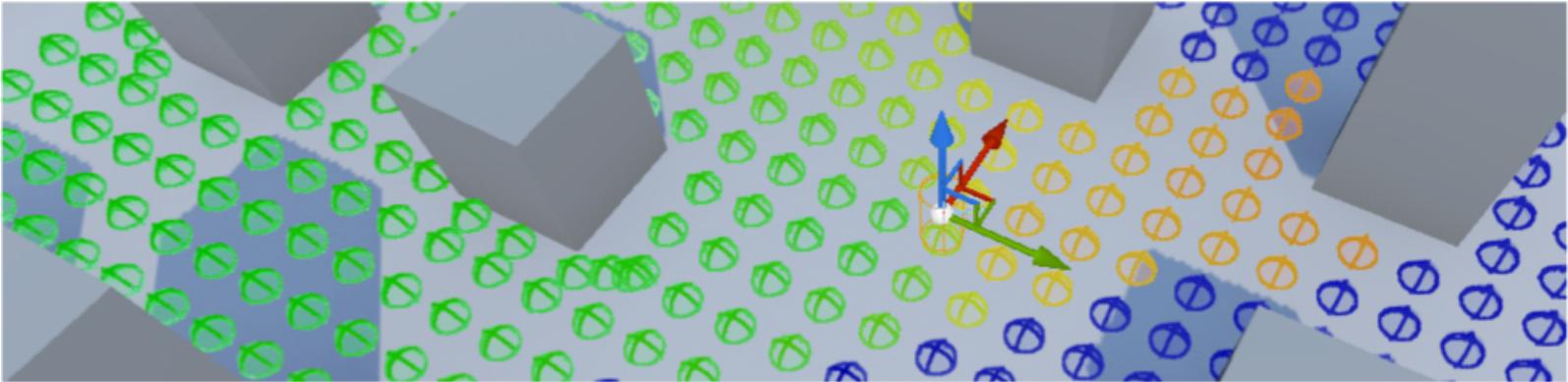
Influence maps

- How strong is the influence of each player?
- Based on unit and building strength values
- Distance attenuation

W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
W	W	W	W	W	2	W	W	W	W	W	3	W	W	W	W	B
W	W	4	W	W	W	W	W	W	W	2	W	W	W	W	B	B
W	W	W	W	W	W	W	W	W	W	W	W	W	W	B	B	2
W	W	W	W	2	W	1	W	W	W	W	W	W	W	B	B	B
W	W	W	W	W	W	W	1	W	W	W	W	W	B	B	B	B
W	W	W	W	W	W	W	W	W	W	W	W	B	B	2	B	B
W	W	W	W	W	W	W	W	W	W	W	W	B	B	B	B	B
W	W	W	W	W	W	W	W	W	W	W	B	B	B	B	B	2
W	W	W	W	W	W	W	W	W	W	W	B	B	B	B	B	B
W	W	W	W	W	W	W	W	W	W	B	B	B	B	B	B	B
W	W	W	W	W	W	W	W	W	W	B	B	B	B	B	B	B
W	W	W	W	W	W	W	W	W	W	B	B	B	B	B	B	B
W	B	B	B	B	W	W	B	B	B	B	B	B	B	B	B	B
B	B	B	2	B	B	B	B	B	B	B	B	B	B	B	B	B
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Unreal Engine Environmental Queries

- Generates query points (e.g. around the AI)
- Tests the points and returns the best one(s)



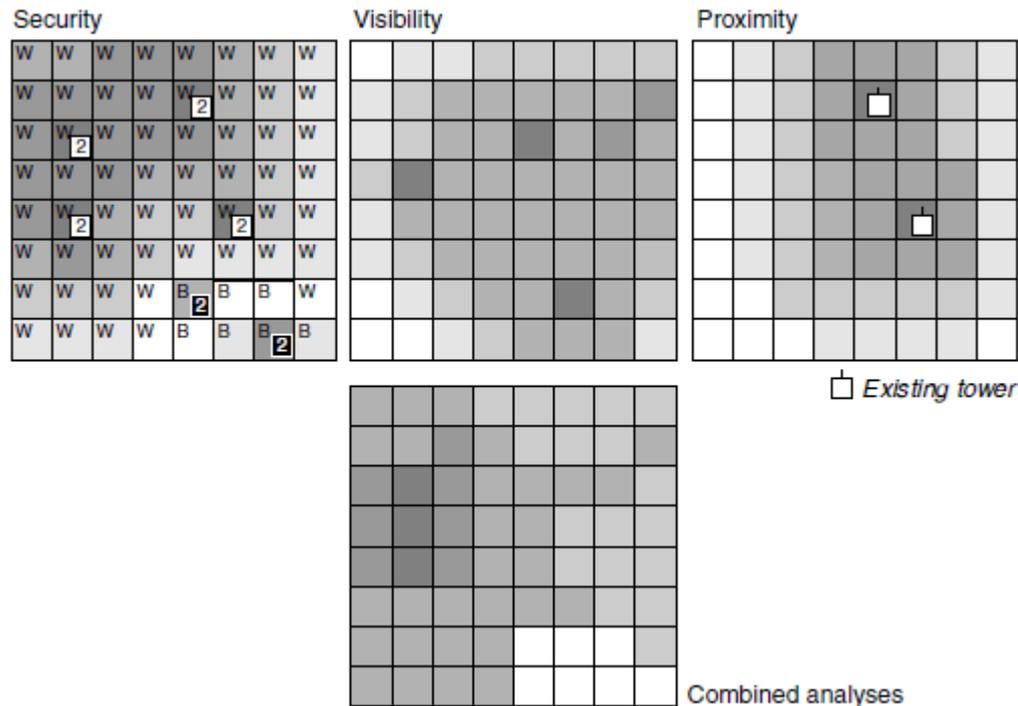
Learning influence maps

Frag Maps



Multiple Layer Analysis

Combine different tactical analyses into one



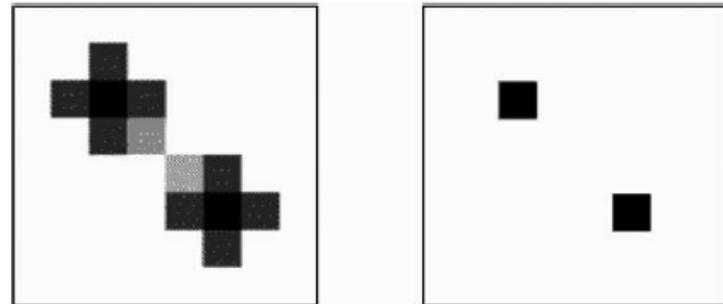
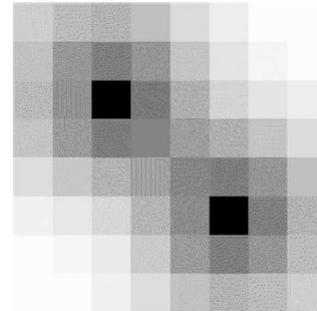
Generating influence maps

Map Flooding

- Use Dijkstra Algorithm variant
- If influences overlap, the stronger influence wins
- Leads to Voronoi Diagram-like map

Filters

- Blur
- Sharpen
- E.g. to find the most important location



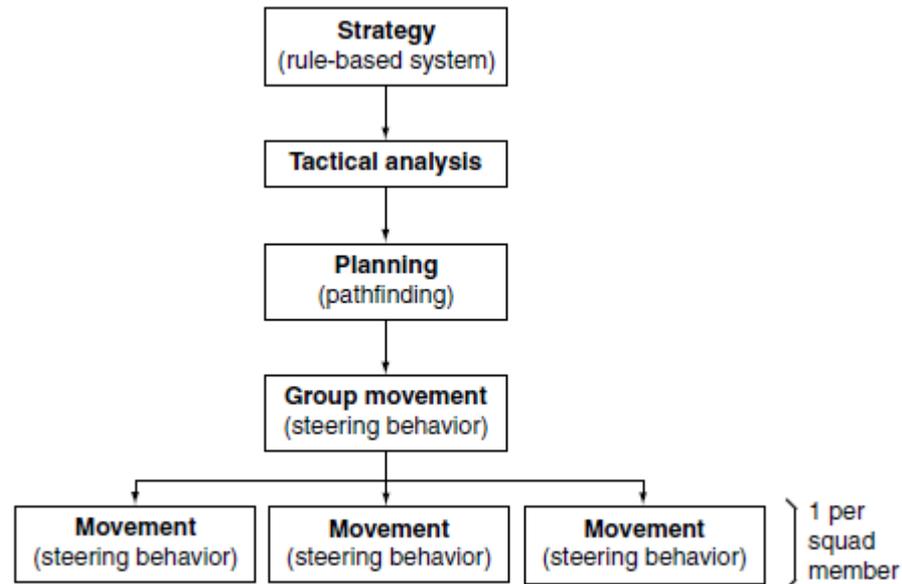
Coordinated Action

Bottom up

- Als query higher-level systems for tactical data

Top-down

- High-level commands passed downwards



Execution management

AI gets its
information

AI gets given processor time

Execution management

Group AI

Strategy

Character AI

Decision making

Movement

Content creation

Scripting

AI has implications
for related technologies

Animation

Physics

AI gets turned into on-screen action

World interface

Scheduling

Frequency

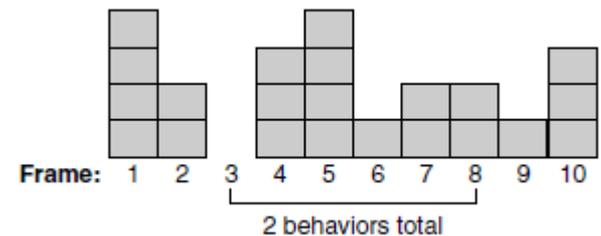
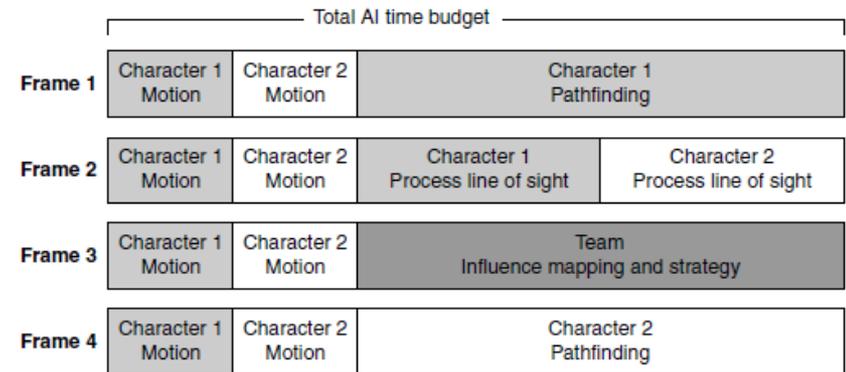
- How often should the AI task be called?

Phase

- Offset the task's execution by n frames
- If several tasks have the same phase, spread them out by changing phases

Choose a good phasing

- Wright's method
 - Simulate (only counting) ahead N frames
 - Choose the frame with the lowest number of tasks and set the phasing accordingly



Interruptible tasks

Allow AI tasks to run over several frames

E.g. pathfinding across the whole map

Continue where it stopped the last time



Anytime Algorithms

Be interruptible

Always provide a solution

- Can be used as a good starting point
- Will be refined the longer the algorithm runs in total

Also remember hacks

- Just start going into the direction of the target
 - Can always backtrack later
 - Always better than not moving for some seconds
- Carry out a „order received“ animation

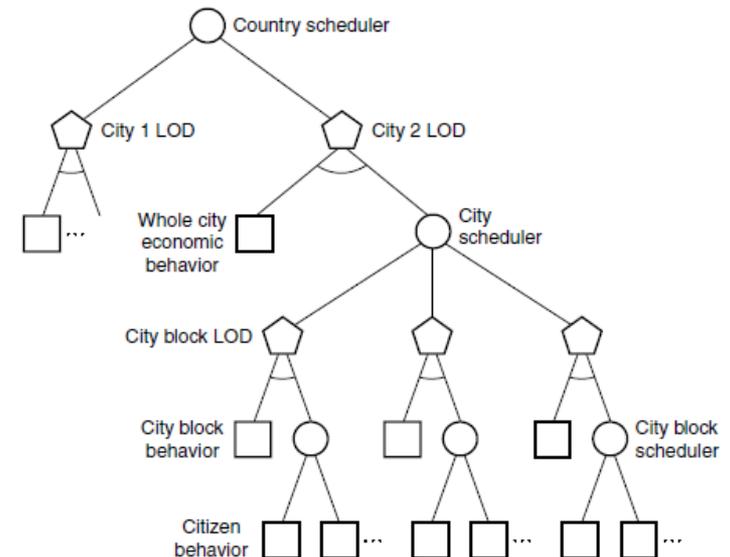
AI Level of Detail

AI that is imperceptible gets less time and simulates less

Hysteresis

- Characters moving in and out of the radius

Phase out areas that are not nearby



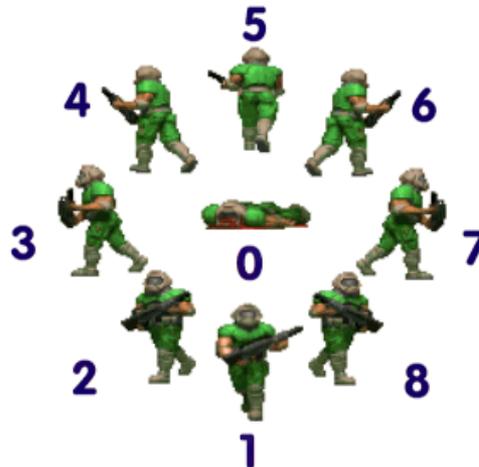
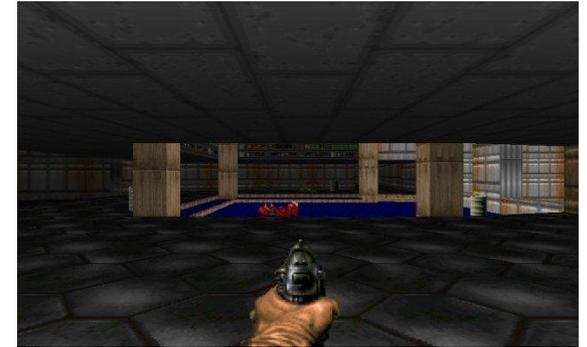
AI Recap: Doom (1993)

Source code available at

<https://github.com/id-Software/DOOM>

Basics

- Characters displayed as sprites with 8 directions



Historical AI: Doom (1993)

Movement → Kinematic

Basic moving

- Character is given a move direction and a speed
- If valid (no wall, no falling, ...) character is moved
- If bumps into an openable door, game opens it
- Facing the move direction: adjusted in 90 degree steps

Chasing

- Find a chase direction
- Followed for a fixed number of frames (movecount), then re-evaluated
 - Direct path if available
 - If no direct path, try random

Historical AI: Doom (1993)

Pathfinding

- Nada
- Not implemented until Quake 3?

Decision Making

- Finite State Machine
- Also combined with the animation sprite handling
- Callback to A_XYZ functions for actions

```
typedef struct
{
    spritenum_tsprite;
    longframe;
    longtics;
    // void(*action) ();
    actionf_taction;
    statenum_tnextstate;
    longmisc1, misc2;
} state_t;
```

```
{SPR_POSS,0,10,{A_Look},S_POSS_STND2,0,0},// S_POSS_STND
{SPR_POSS,1,10,{A_Look},S_POSS_STND,0,0},// S_POSS_STND2
{SPR_POSS,0,4,{A_Chase},S_POSS_RUN2,0,0},// S_POSS_RUN1
{SPR_POSS,0,4,{A_Chase},S_POSS_RUN3,0,0},// S_POSS_RUN2
{SPR_POSS,1,4,{A_Chase},S_POSS_RUN4,0,0},// S_POSS_RUN3
{SPR_POSS,1,4,{A_Chase},S_POSS_RUN5,0,0},// S_POSS_RUN4
{SPR_POSS,2,4,{A_Chase},S_POSS_RUN6,0,0},// S_POSS_ATK2
```

Historical AI: Doom (1993)

Strategy

- Nothing here yet

World Interface

- Checking if player is visible
 - Distance, 180 degrees vision, Obstructed by geometry?
- Activated by sounds
 - AI sounds alert them to the player, sound propagation approximated

Execution Management

- „Thinkers“ for game objects (including AI) called once per tic

Content Creation, Scripting

- Scripting implemented in Hexen
- Content Creation: Doomed Editor, add objects with special relevance (triggering events), but event itself hardcoded

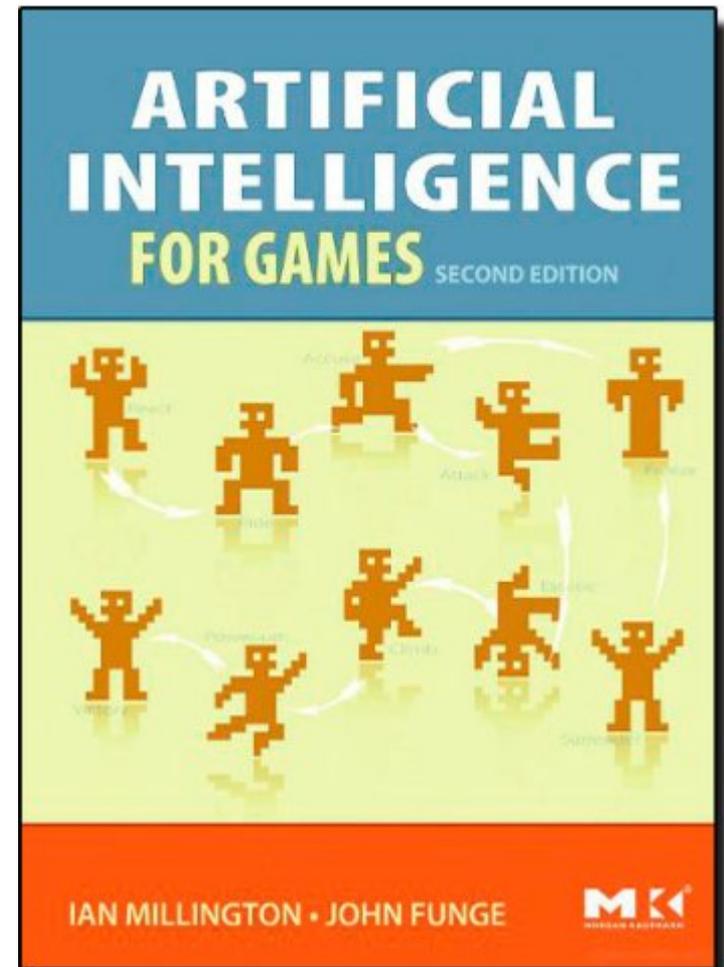
Literature

Artificial Intelligence for Games

- Ian Millington, John Funge

AI Game Programming Wisdom Series

<http://aigamedev.com/>



Conclusion

Different levels of AI control

- Basic movement algorithms
- Choosing where to move and on which path
- Higher-level behaviours
- Planning

Approaches

- Hacks
- Heuristics
- Algorithms