Game Technology

Lecture 14 – Artificial Intelligence in Games

Skipped

Skipped slides will be marked with this symbol and are not relevant for the exam

Dr.-Ing. Florian Mehm
Dipl-Inf. Robert Konrad

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

Final timetable

Lecture No.	Date	Topic
1	17.10.2014	Basic Input & Output
2	24.10.2014	Timing & Basic Game Mechanics
3	31.10.2014	Software Rendering 1
4	07.11.2014	Software Rendering 2
5	14.11.2014	Basic Hardware Rendering
6	21.11.2014	Animations
7	28.11.2014	Physically-based Rendering
8	05.12.2014	Physics 1
9	12.12.2014	Physics 2
10	19.12.2014	Procedural Content Generation
11	16.01.2015	Compression & Streaming
12	23.01.2015	Multiplayer
13	30.01.2015	Audio
14	**06.02.2015**	**AI**
15	13.02.2015	Scripting - Optional

Current news

Lecture and Exercise 15, Scripting

- will be optional and not part of the exercise bonus points or the exam

Exercise points and overall bonus points will be completed soon

Example exam

- Will be provided next week

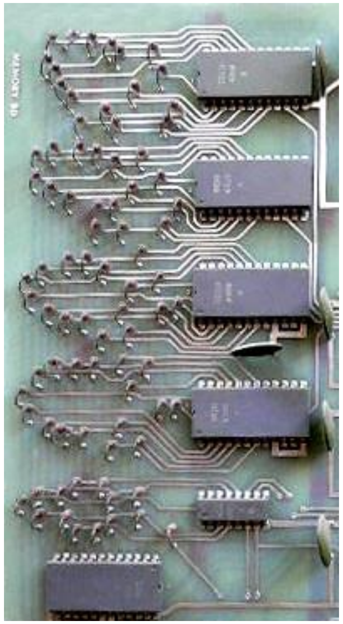
Feedback

- Thank you for filling out the feedback sheets!
- We will discuss some of the feedback and how we plan to adress it

Introduction

Computer Space (1971)

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

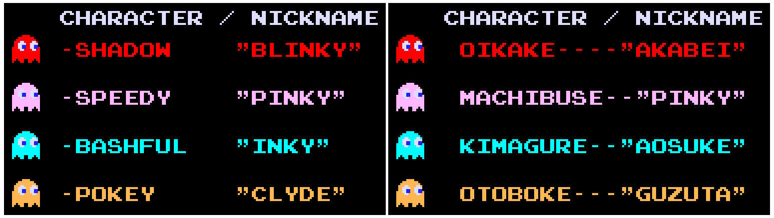


More info and image sources: http://www.technologizer.com/2011/12/11/computer-space-and-the-dawn-of-the-arcade-video-game/2/

TECHNISCHE UNIVERSITÄT DARMSTADT

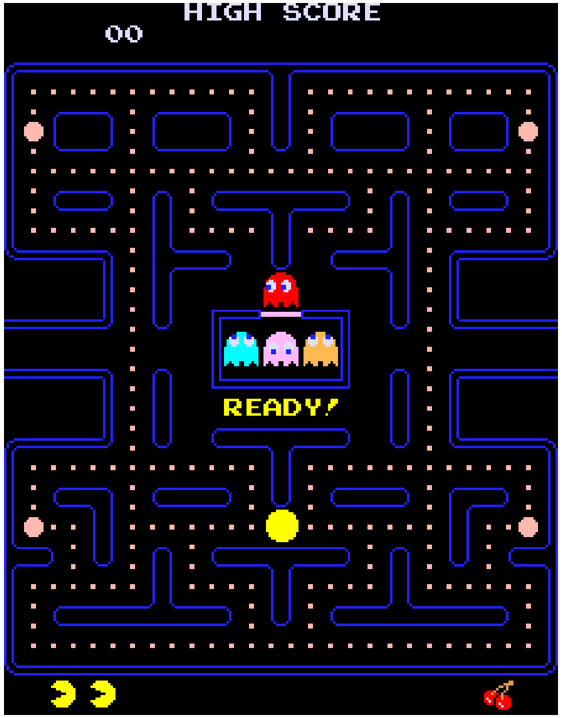
Pac Man (1980)

- Tōru Iwatani
- Four different ghost „personalities“



E.g. red ghost

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



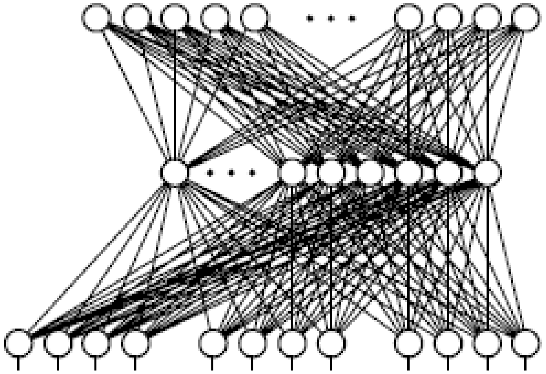
More info

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

Introduction

Creatures (1996)

- Steve Grand
- Train and breed creatures
- Used neural networks for AI

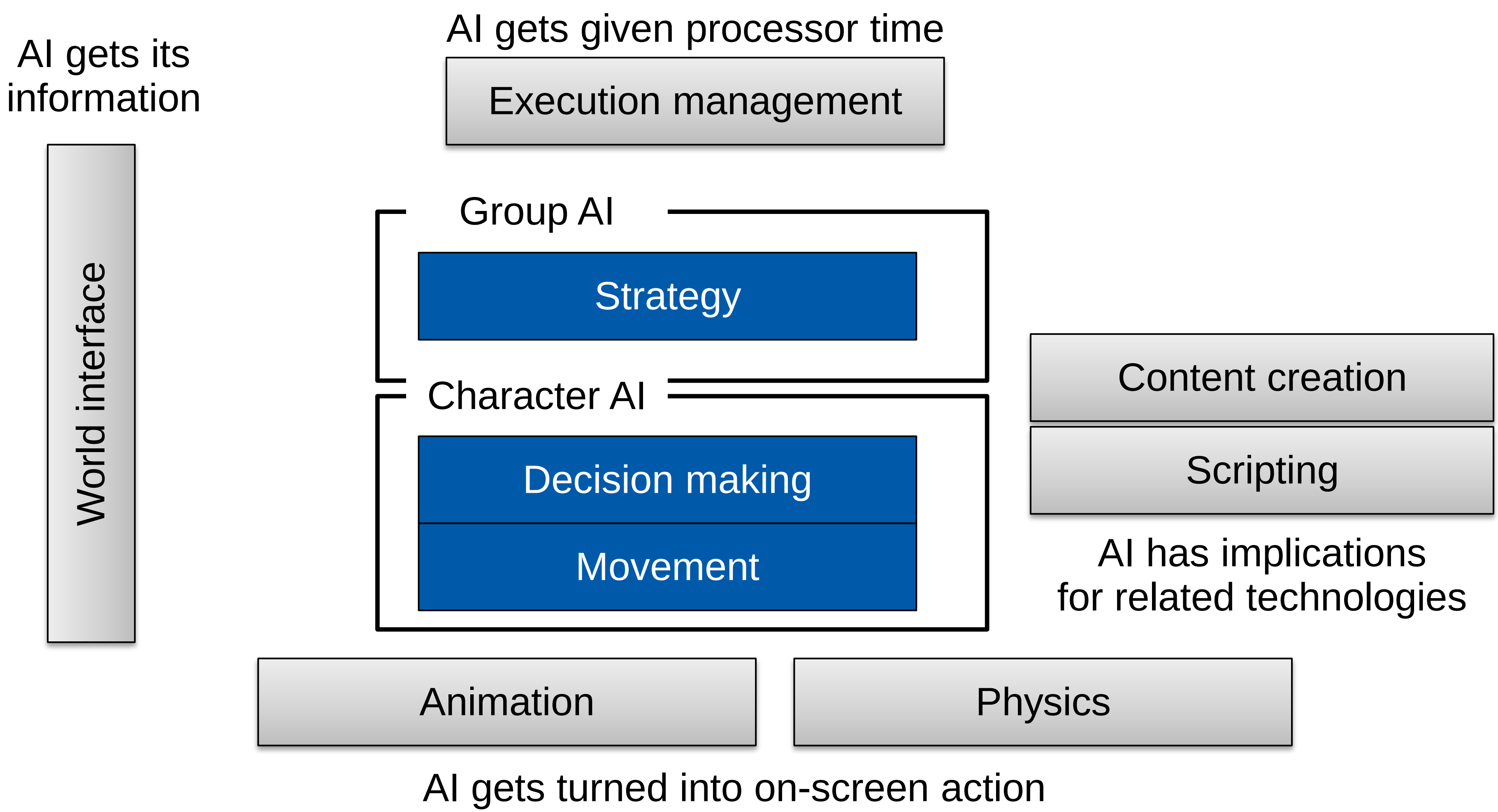


Introduction

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

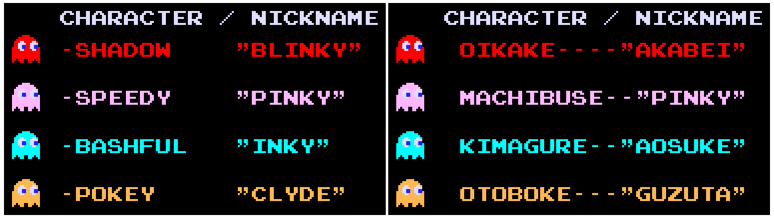

AI gets its
information
World interface
AI gets given processor time
Execution management
Group AI
Strategy
Character AI
Decision making
Movement
Content creation
Scripting
AI has implications
for related technologies
Animation
Physics
AI gets turned into on-screen action

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



Complexity

More complex things can easily look bad

Kinds of AI in games

TECHNISCHE UNIVERSITÄT DARMSTADT

„Hacks“

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

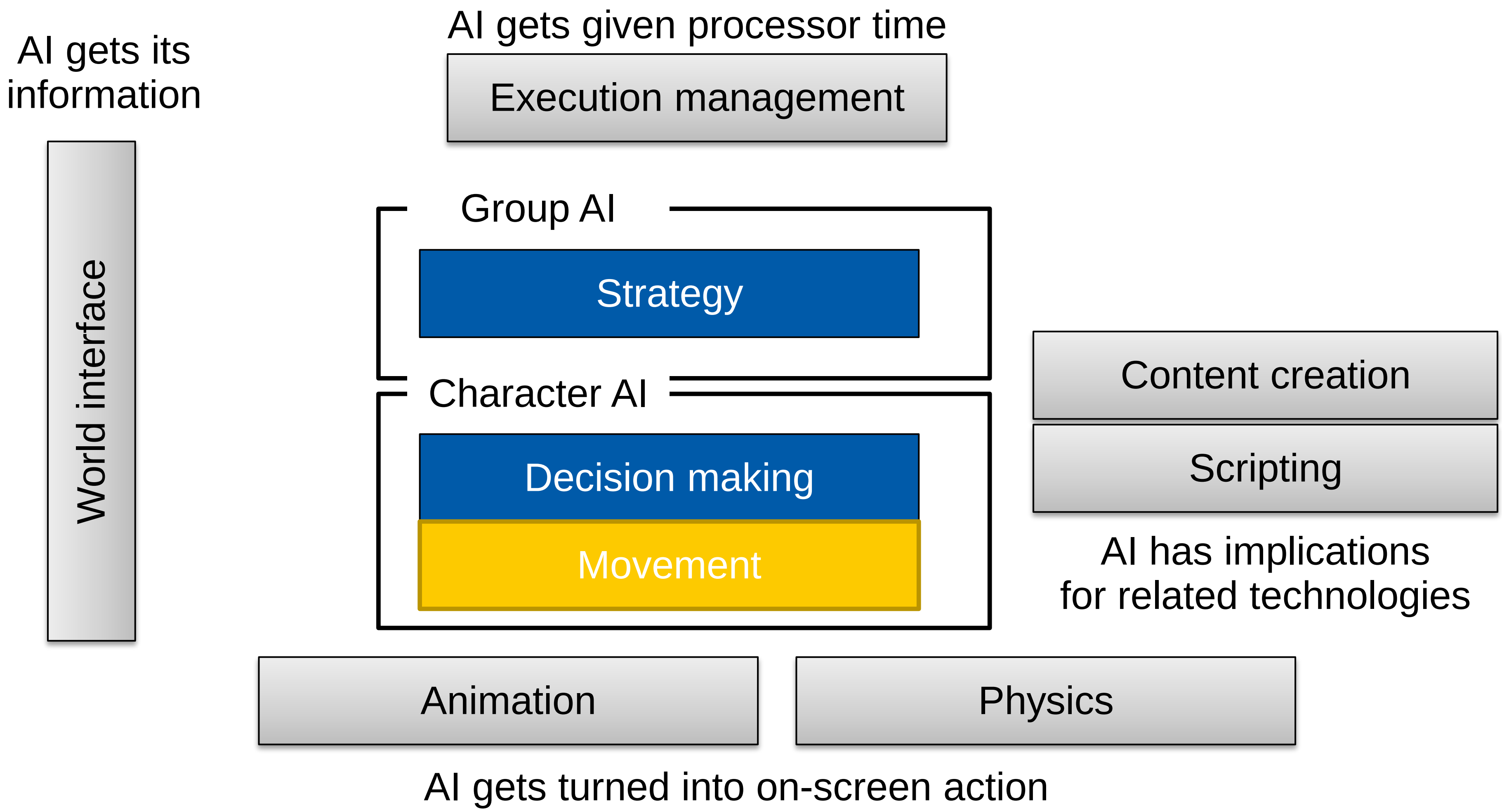
Heuristics

- 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

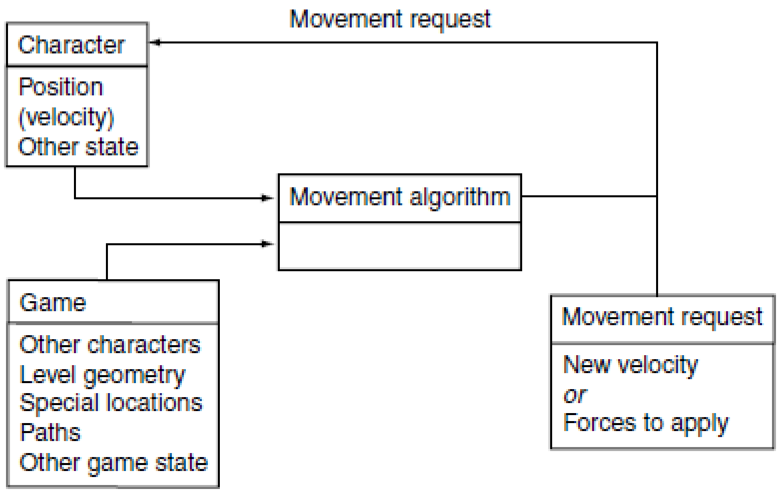
Movement


AI gets its information
World interface
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

Movement

Input

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



Output

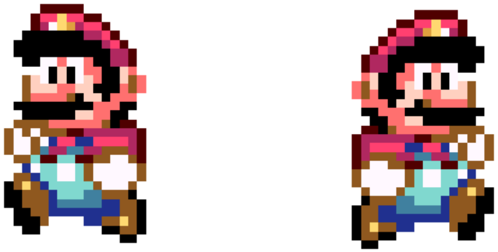
- Movement requests

Kinematic vs. Dynamic movement

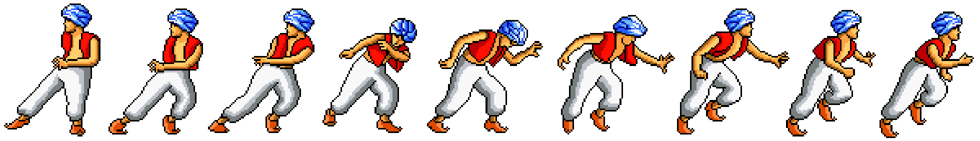
- Kinematic – Update the velocity directly
- Dynamic – Take into account velocities, change via accelerations (see physics lectures)

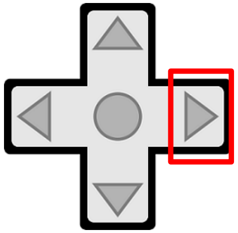
Movement comparison

Kinematic



Dynamic





https://archive.org/details/msdos_Prince_of_Persia_1990

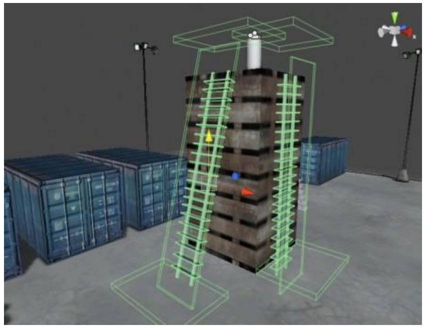
Dimensionality

2D

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

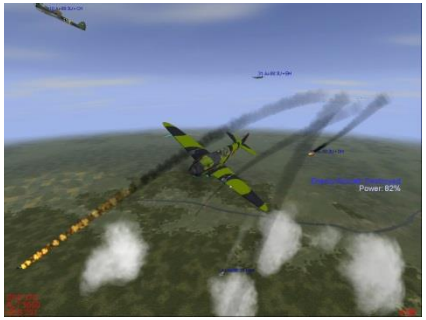
2.5D

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



3D

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



http://xn--m3h.net/umbrella/2010/11/unity3d-ladders/

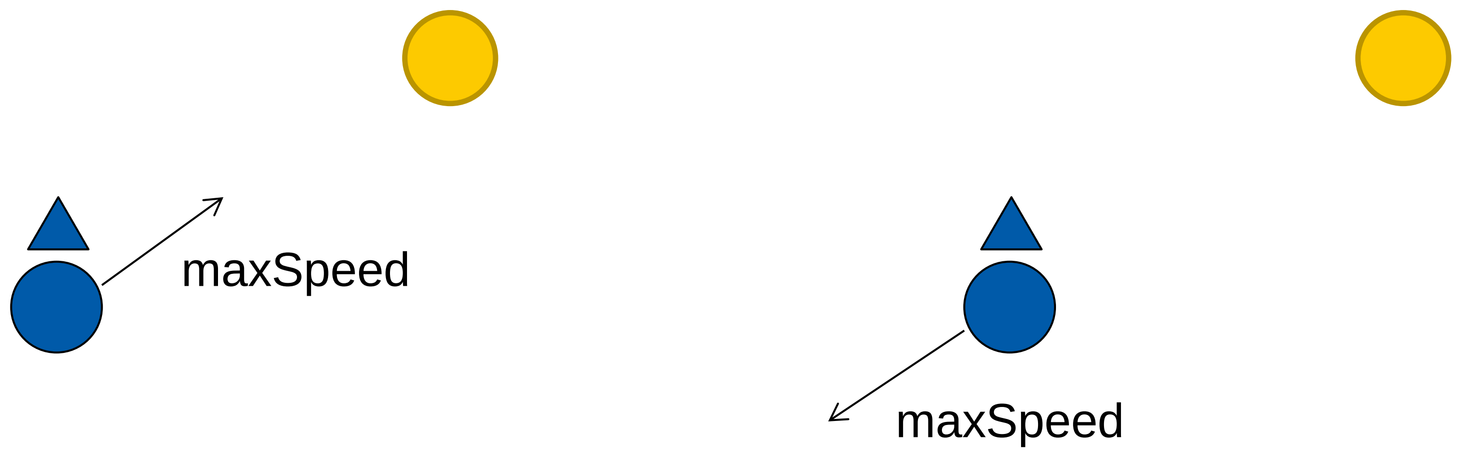
Simple movement behaviours (kinematic)

Seek

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

Flee

- Reverse direction from Seek



Simple movement behaviours

TECHNISCHE UNIVERSITÄT DARMSTADT

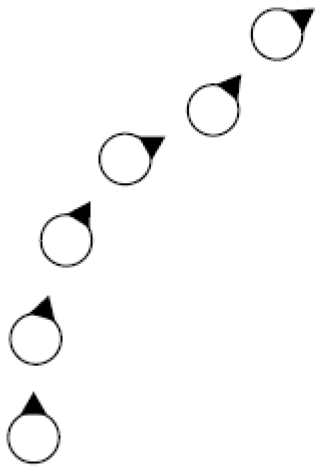
Arrive

- 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 (probaby other behaviour appropriate, e.g. attacking)

Simple movement behaviours

Wander

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

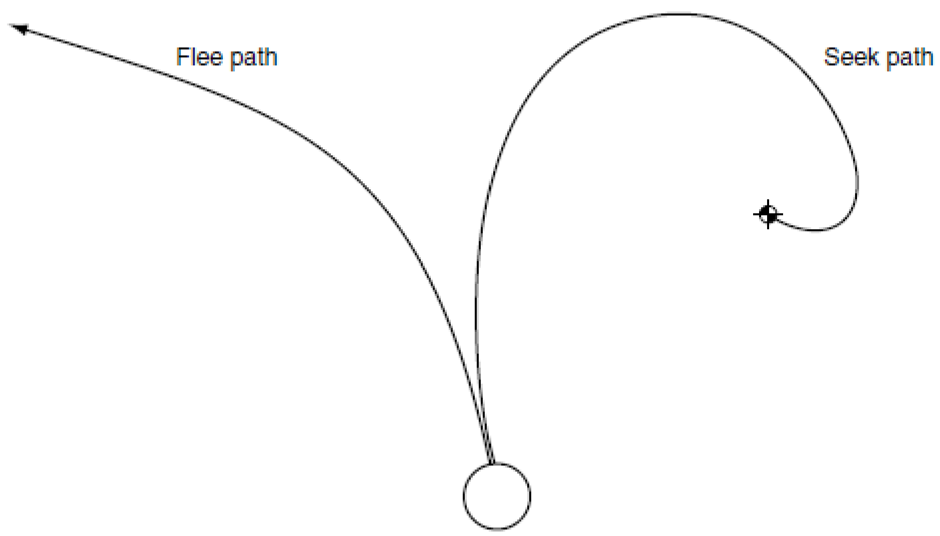


Dynamic Steering

Take into acount rotation and velocity

Seek and Flee

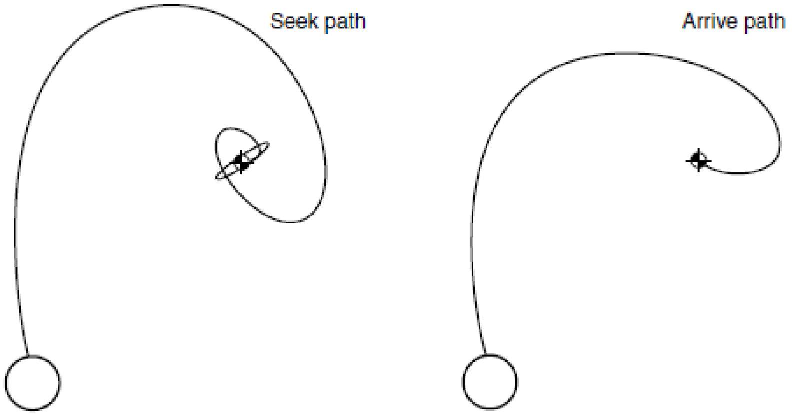
- As before: Choose direction to goal
- Output an acceleration in that direction
- Maximum speed: Either via clipping or via drag



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

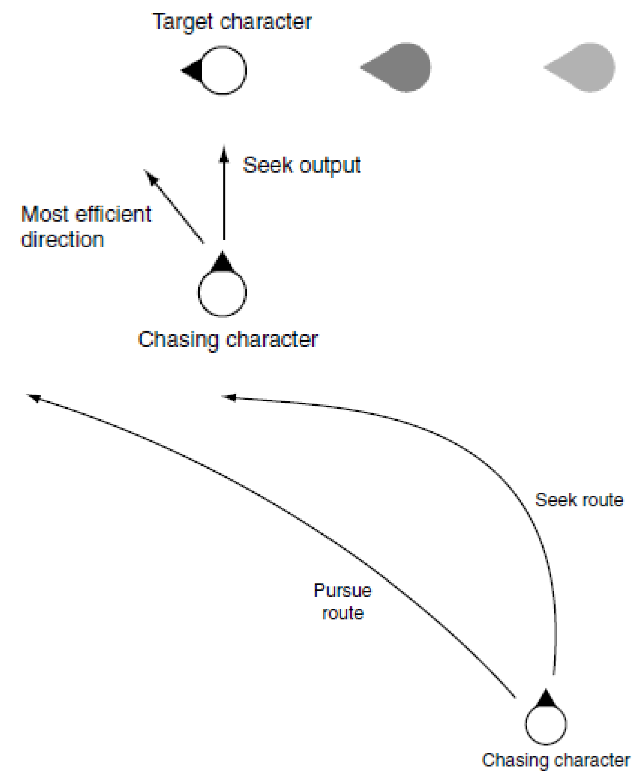
Based on the primitive behaviours we have have looked at until now

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

TECHNISCHE UNIVERSITÄT DARMSTADT

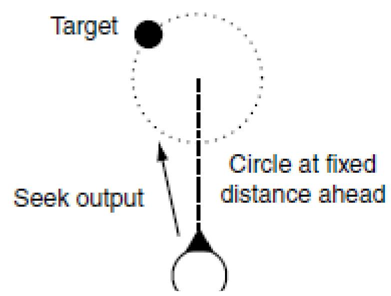
Face

- Align to the target

Looking where you are going

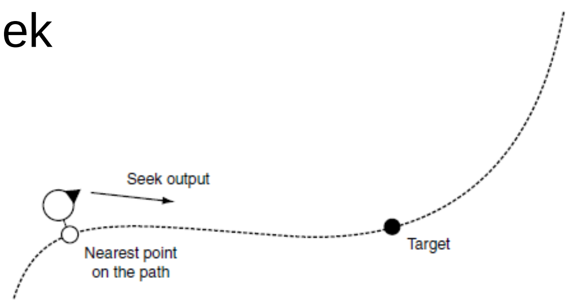
Wander

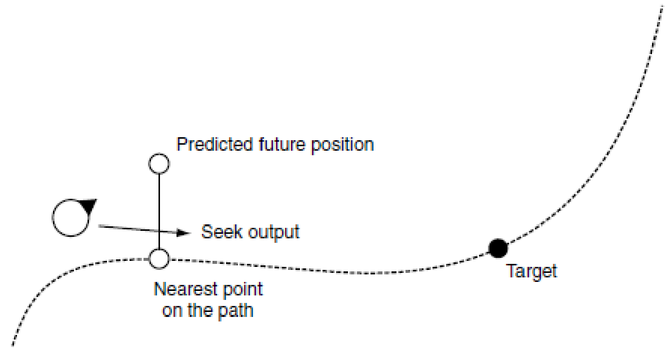
- Uses Seek



Path following

- Uses seek





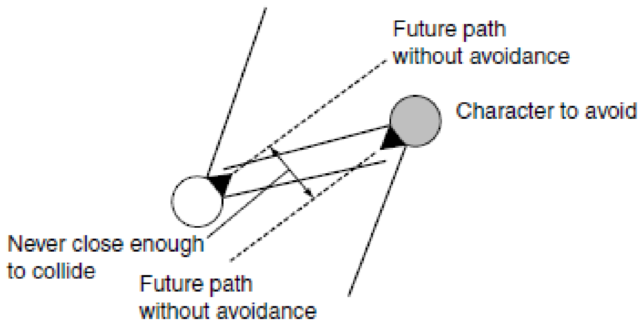
Delegated behaviours

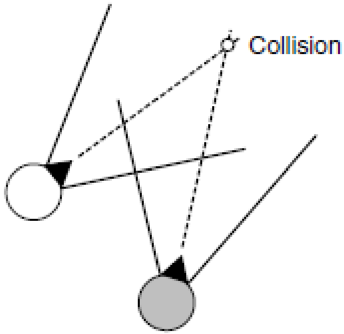
Separation

- Stay away from nearby other characters
- strength = maxAcceleration * (threshold - distance) / threshold

Collision Avoidance

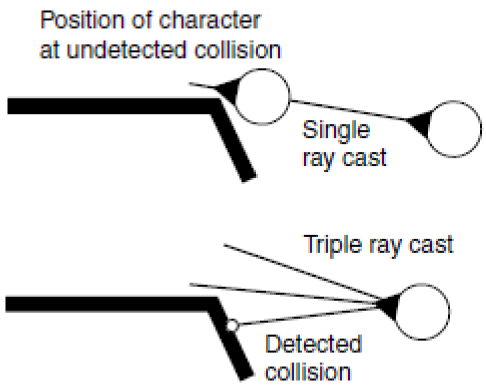
- For characters that move on crossing trajectories
- a) Only do separation inside a cone infront of the character
- b) Only handle collisions that will take place (similar to collision detection in the physics lectures) → no „panic“ reactions





Obstacle and Wall Avoidance

Similar, but distinct from collision detection in physics

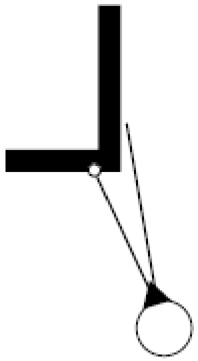


Basic mechanism: Use ray casts

- 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



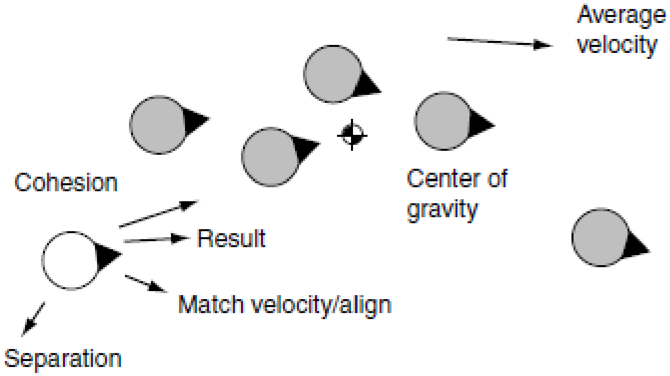
Combining steering behaviours

(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)

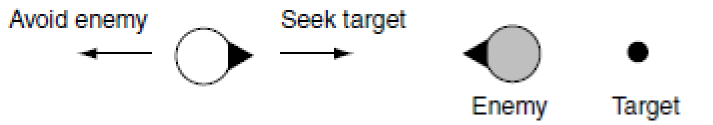
Combining steering behaviours

Equilibria

- Conflicting steering behaviours lead to deadlocks

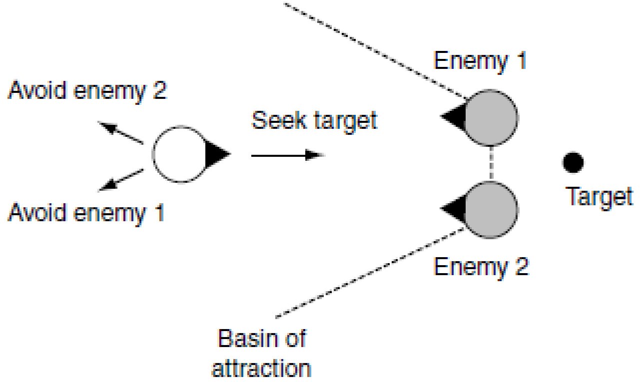
Unstable equilibria

- Situation is inpropable
- Basin of attraction is small



Stable equilibria

- Less likely to be broken by numerical inaccuracies

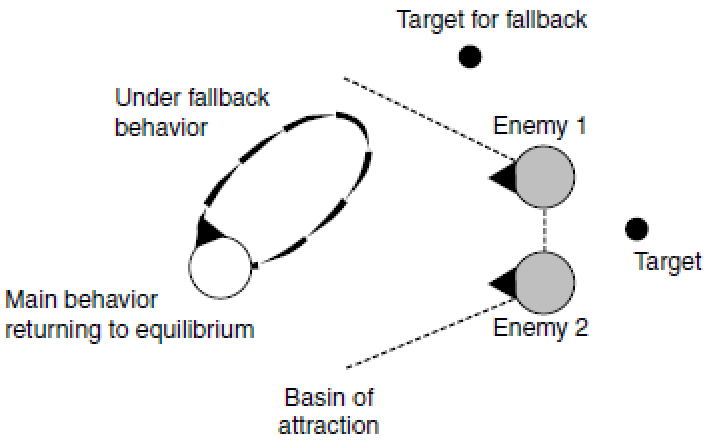


Priorities

Define priorities for steering behaviours

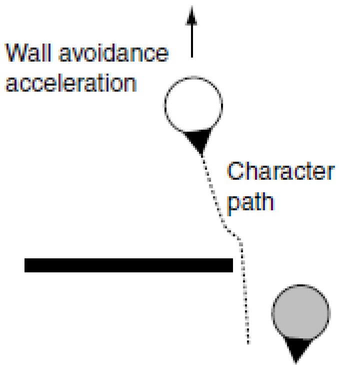
E.g. collision avoidance has priority, if it has an output, it is taken

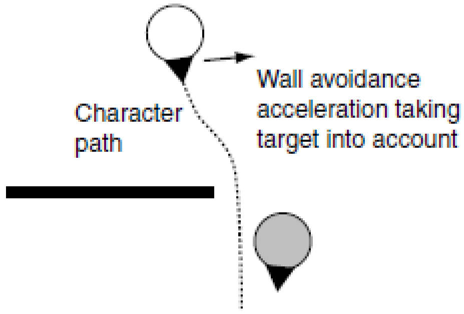
Can still oscillate for stable equilibria



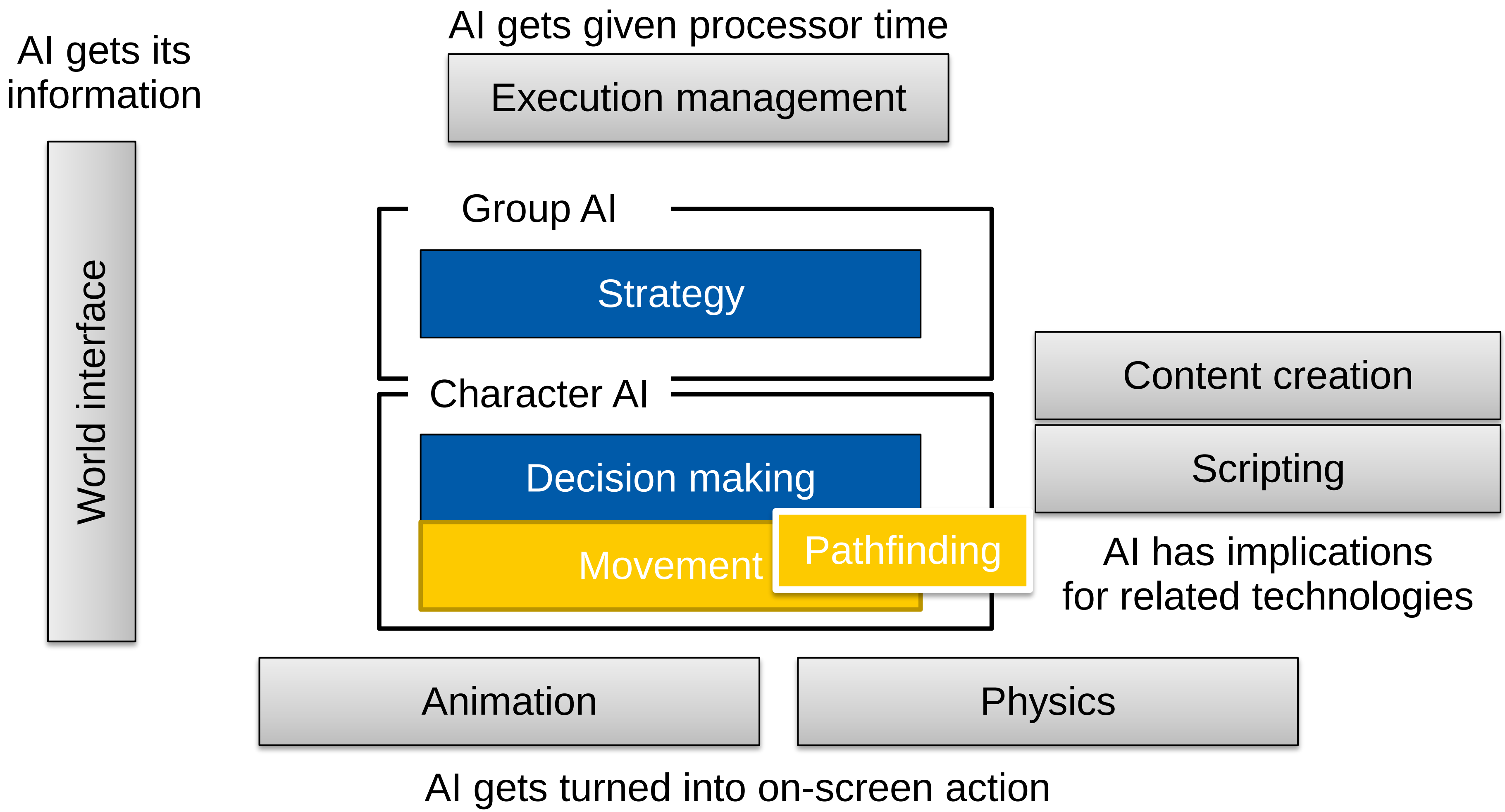
Cooperative arbitration

Communication between steering behaviours





Pathfinding



Pathfinding

TECHNISCHE UNIVERSITÄT DARMSTADT

Higher-level movement planning

- Generate a path for an AI to follow

Based on a graph

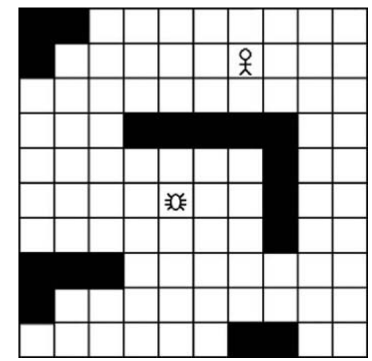
- Basic connectivity: point B can be reached from A if there is a connection between them
- Weights/costs: How „hard“ is it to get from A to B?
- Directions: In which direction is the connection? (E.g. jumping down an unclimbable ledge)

Finding a path

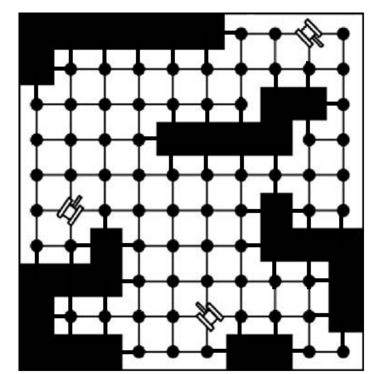
A*-Algorithm is the prevalent solution

Not required to compute it in the exam, but you should know how it works

- 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
```



World representation

Quantization

- Find the appropriate node for the current world position

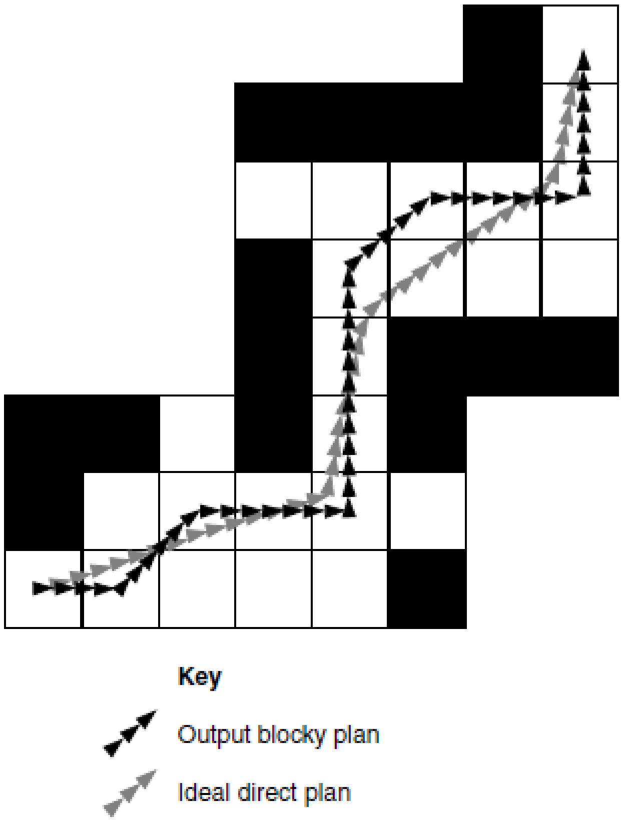
Localization

- Find the world location of a graph node

Tile-based

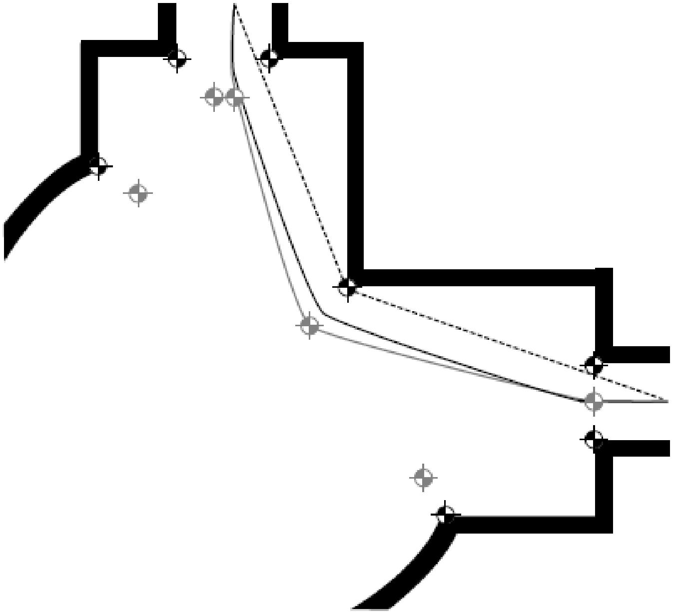
Well-suited if game world is already represented by tiles (e.g. heightmap)

Plans can be blocky due to rectangular nature



Points of visibility

Based on the edges of the world



Key

Optimal path for zero-width character

Path for character with width

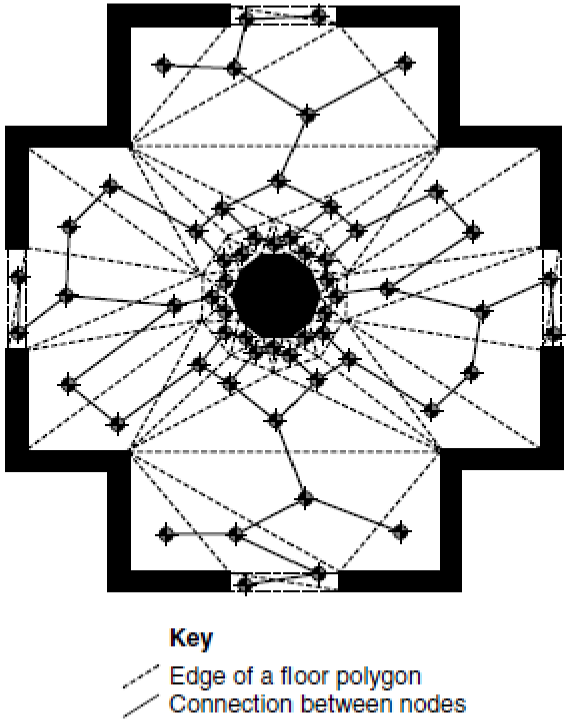
Path using vertex offsets

Original characteristic points at vertices

Offset characteristic points

Navigation mesh

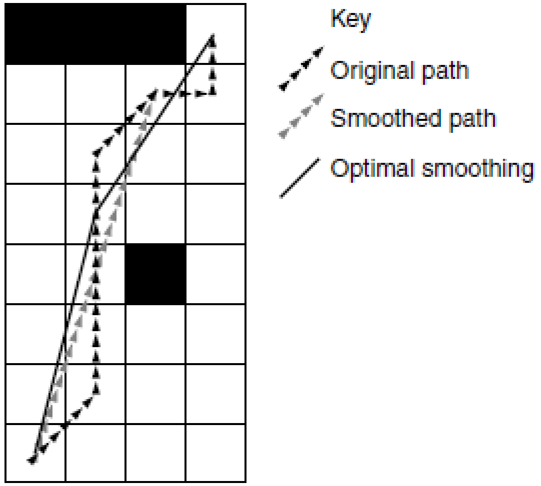
Use the mesh of the floor as the basis



Path smoothing

Filter paths to remove sharp edges

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

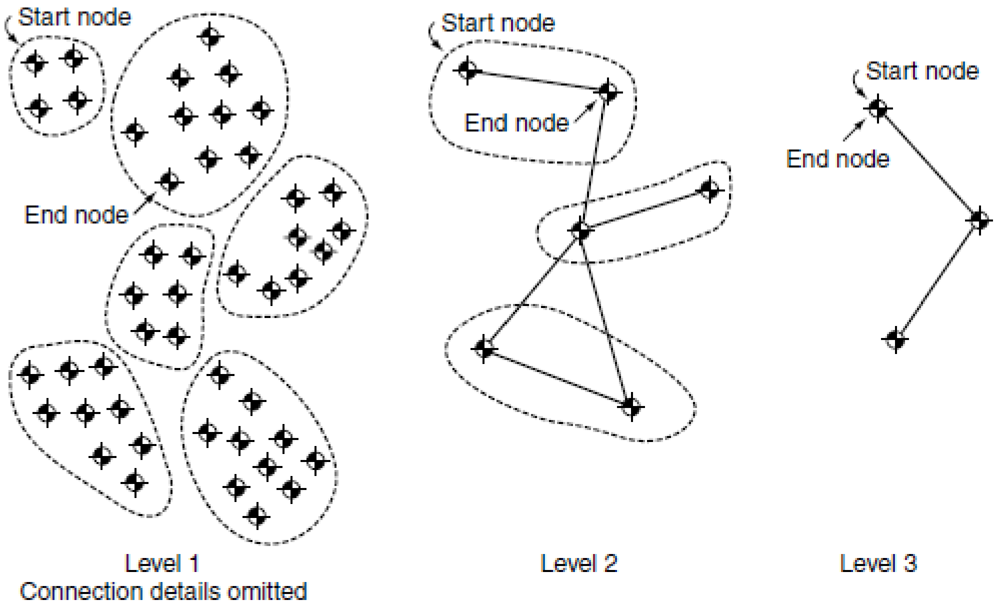


Hierarchical pathfinding

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

Compute paths between the clusters

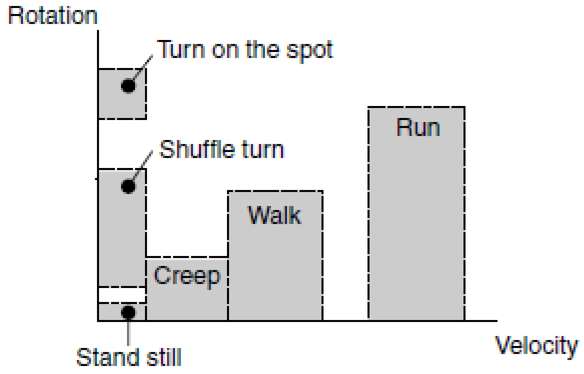
Only when the character needs to traverse a cluster, find a path through it



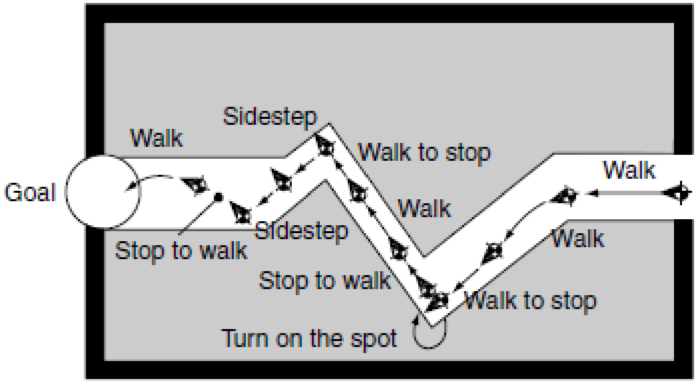
Movement planning

A character that is animated has only a finite set of animations

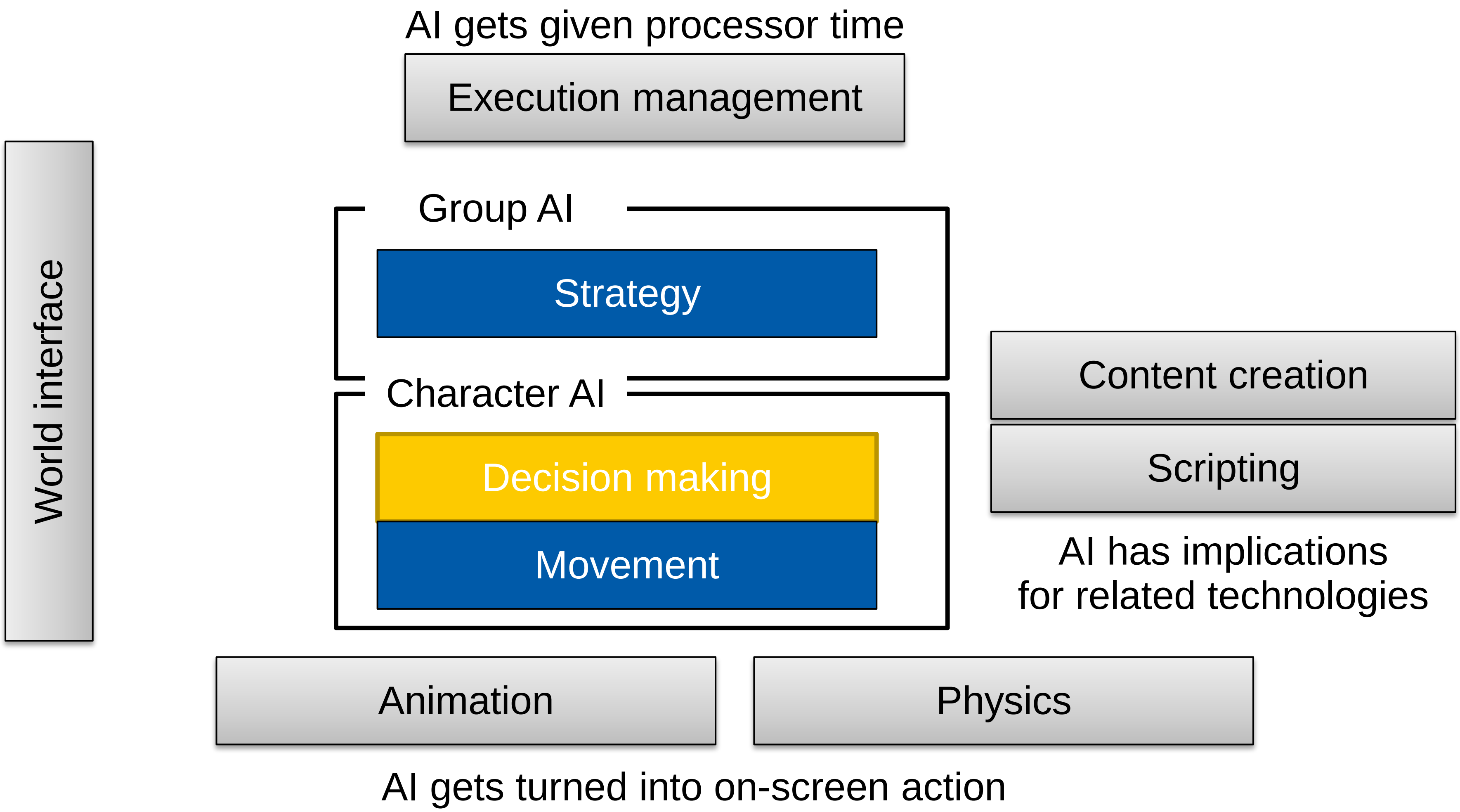
- Limited to certain directions
- Limited to certain speeds



Use A* variant to search good combinations



Decision Making

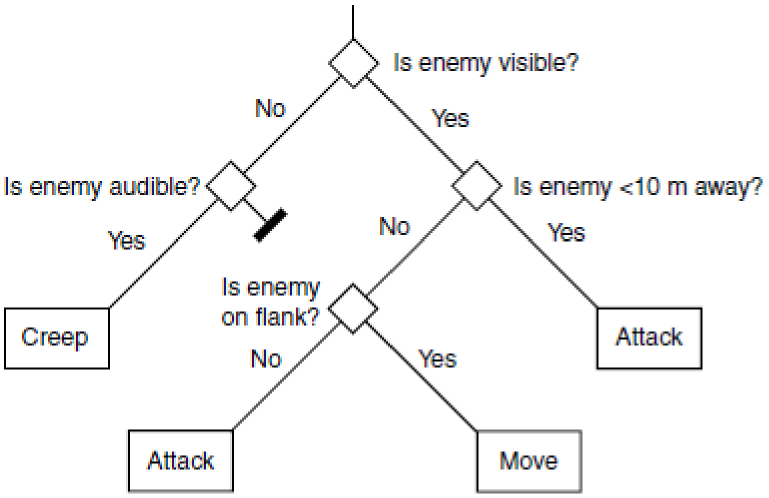


Decision making

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

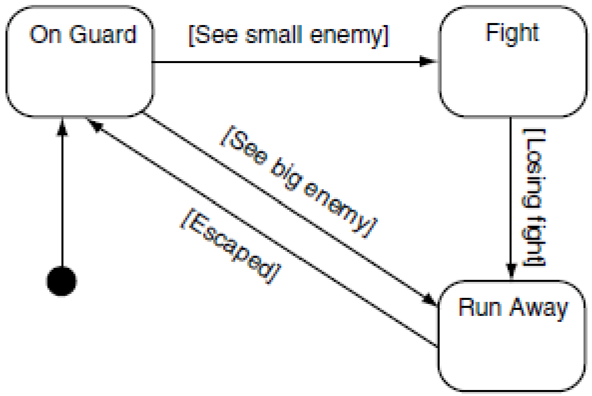
Add randomness to the graph

- 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

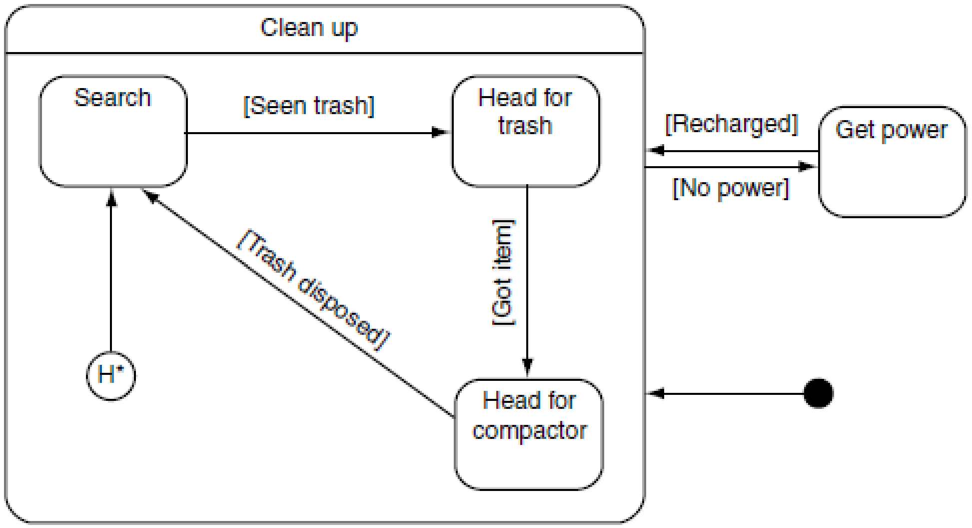
Hard-coded vs. flexible

Hierarchical state machines

TECHNISCHE UNIVERSITÄT DARMSTADT

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



Behaviour Trees

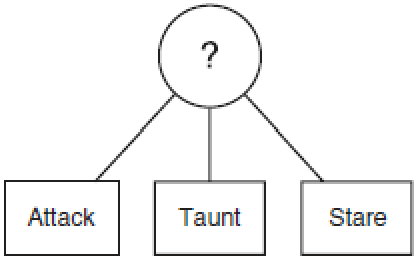
Combination of other techniques

Easy to understand and create for non-programmers

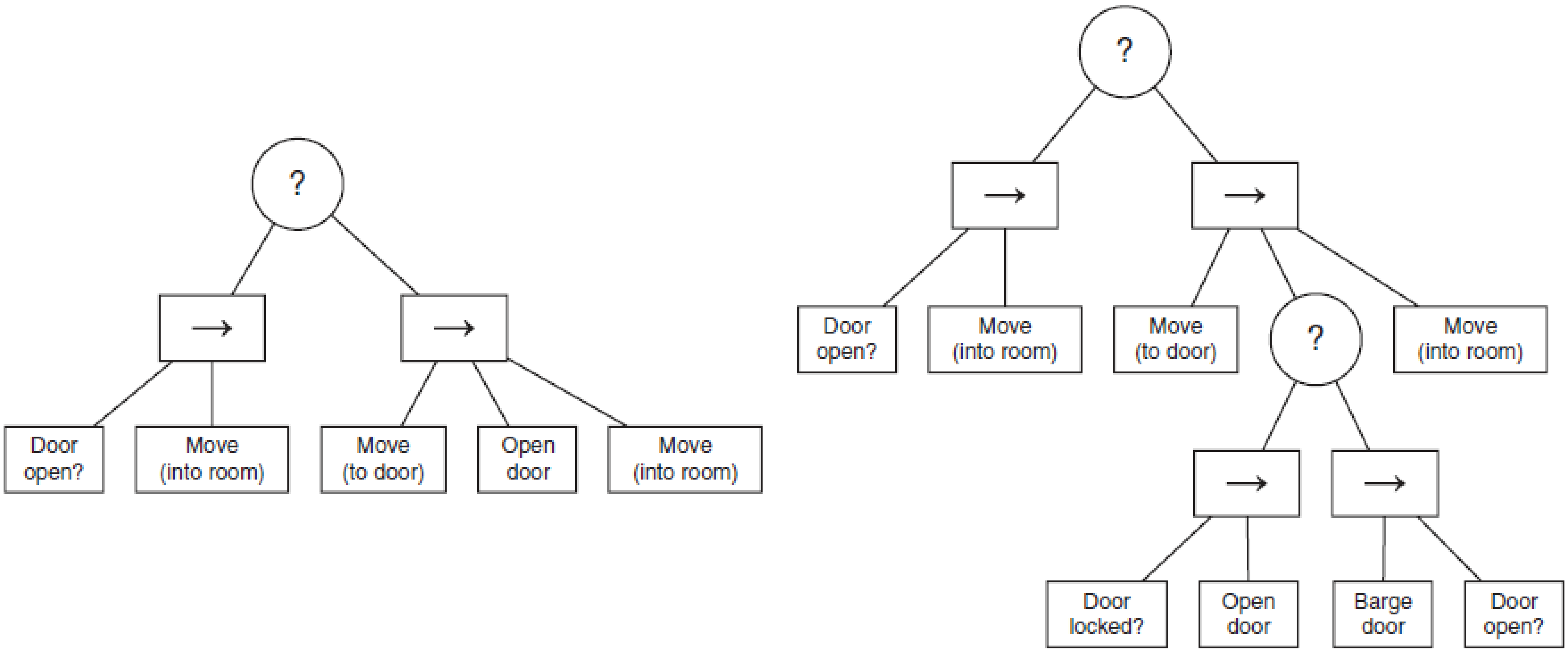
- First description in Halo 2 (2004)

Tasks instead of states

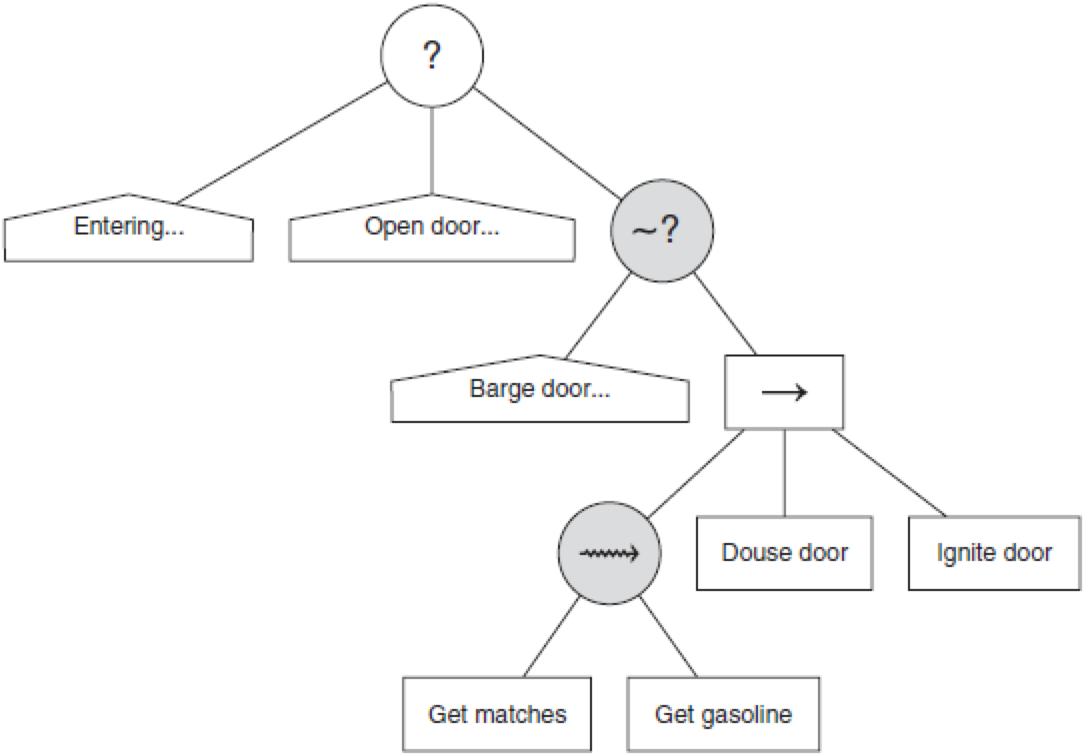
- Conditions
- Actions
- Composites
  - Selector
  - Sequence



Behaviour trees



Non-determinism



Fuzzy logic

Game states can be ambiguous

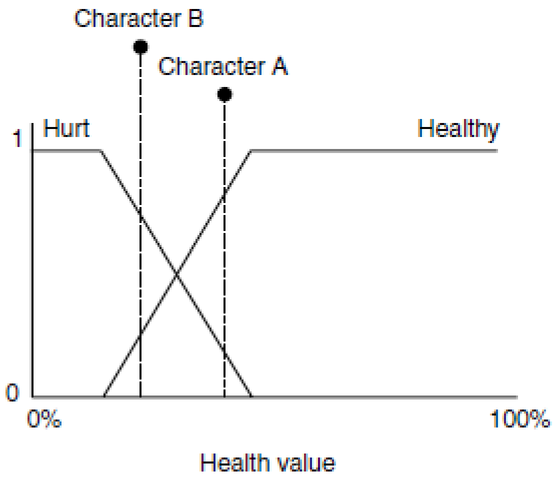
- How much health is „injured“?
- How close is „close“?

Fuzzy logic

- Instead of being or not being in a set, fuzzy logic states that an object has a set degree of membership in the range [0,1]

Fuzzyfication

- Turning game data into set degree of membership



Defuzzyfication

- Mapping fuzzy data to game data

Fuzzy operators and rules

Operators: Similar to Boolean operators

Expression	Equivalent	Fuzzy Equation
NOT A		$1 - m_A$
A AND B		$\min(m_A, m_B)$
A OR B		$\max(m_A, m_B)$
A XOR B	NOT(B) AND A	$\min(m_A, 1 - m_B)$
	NOT(A) AND B	$\min(1 - m_A, m_B)$
A NOR B	NOT(A OR B)	$1 - \max(m_A, m_B)$
A NAND B	NOT(A AND B)	$1 - \min(m_A, m_B)$

Rules

- Define membership in fuzzy sets based on membership in other sets

$$m_{\text{(Should Brake)}} = \min(m_{\text{(Close to the Corner)}}, m_{\text{(Traveling Quickly)}}).$$

Example

corner-entry AND going-fast THEN brake
corner-exit AND going-fast THEN accelerate
corner-entry AND going-slow THEN accelerate
corner-exit AND going-slow THEN accelerate

Corner-entry: 0.1
Corner-exit: 0.9
Going-fast: 0.4
Going-slow: 0.6

$$\begin{aligned}
\text{Brake} &= \min(0.1, 0.4) = 0.1 \\
\text{Accelerate} &= \min(0.9, 0.4) = 0.4 \\
\text{Accelerate} &= \min(0.1, 0.6) = 0.1 \\
\text{Accelerate} &= \min(0.9, 0.6) = 0.6
\end{aligned}$$

Take maximum:
Accelerate = 0.6

Goal-oriented behaviour

Goals

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

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?

Rule-based systems

Skipped

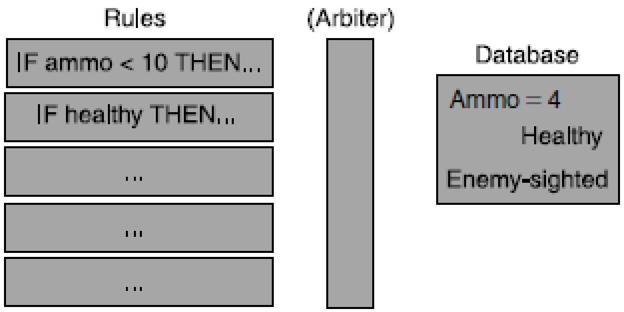
Aka. Expert systems

Set of rules in the form IF condition THEN action

Created by a designer

Arbitration and variable matching

- Which rule to use if several fit?
- Variable matching: When rules have (connected) variables



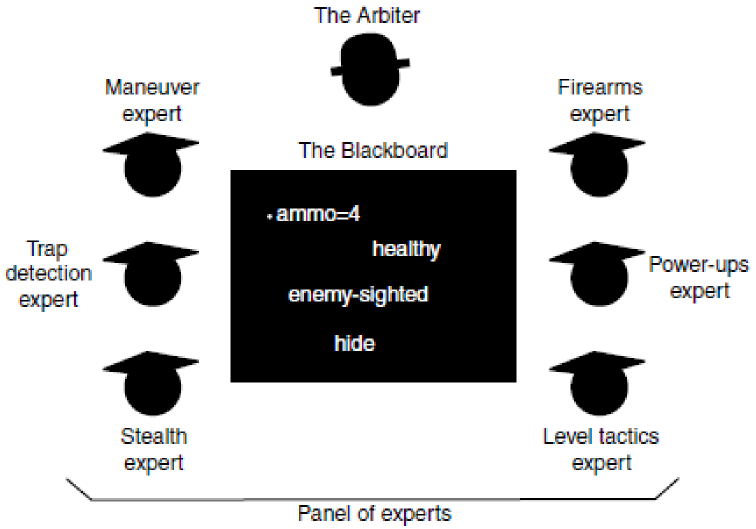
Blackboard architectures

Every AI can write to and read from the blackboard

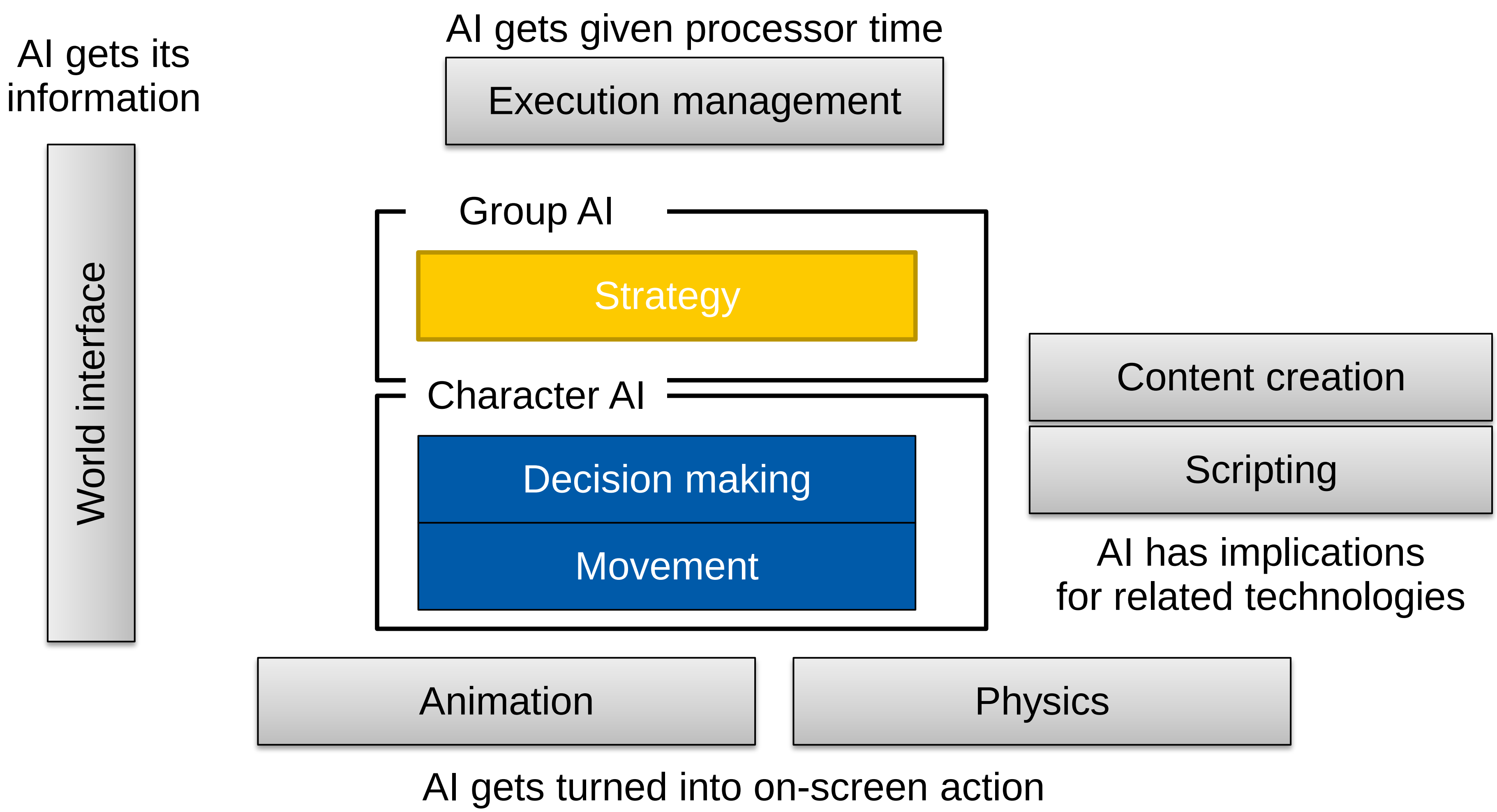
Efficient way of communicating with other AIs

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

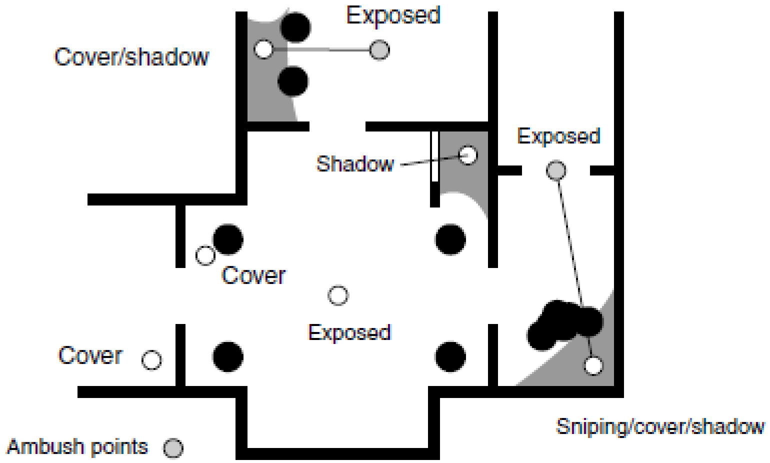


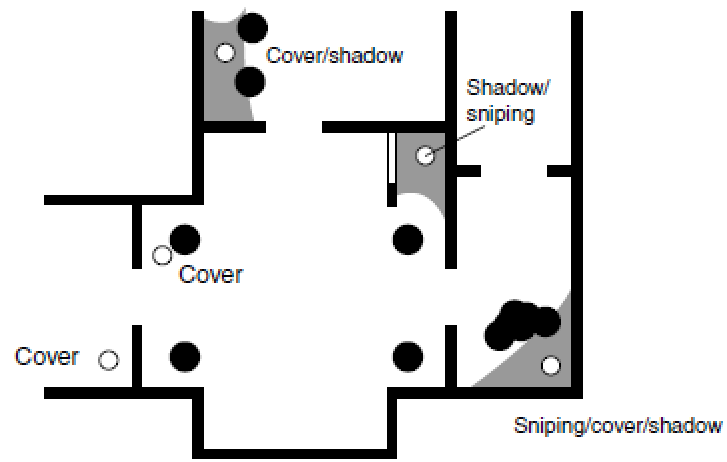
Tactical and Strategical AI

vs

Waypoint Tactics

Find points that are suited for different actions





Context Sensitivity

Skipped

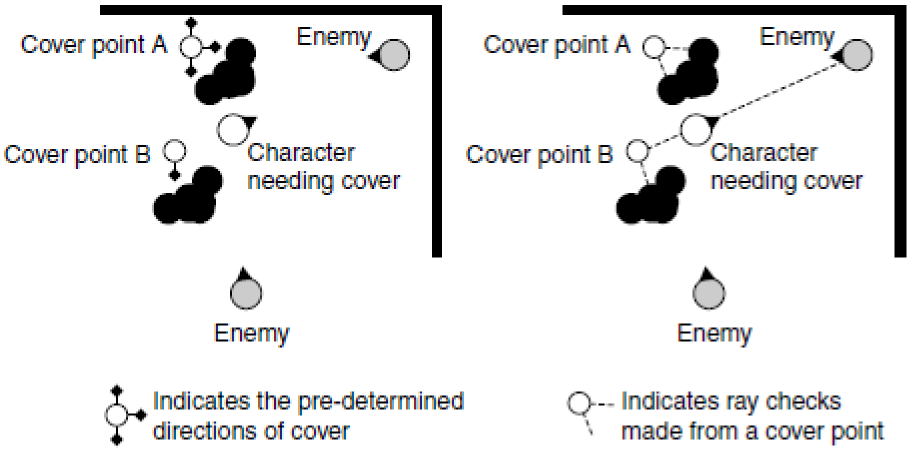
TECHNISCHE UNIVERSITÄT DARMSTADT

A tactical waypoint can depend on the actions of other AIs

- Example: Cover depends on where the enemy is

Precompuation step: Find directions that the point provides cover for

At runtime: Find out if the currently relevant direction is among them

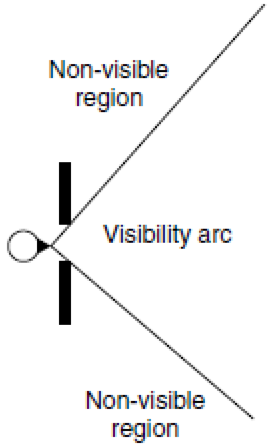


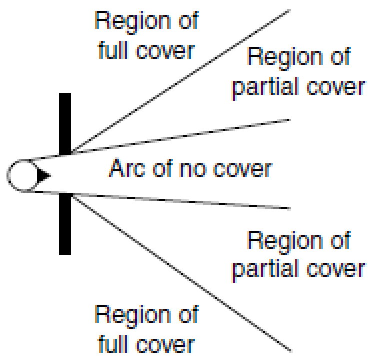
Generating the points

Skipped

Using visibility checks

- Visibility: How much space can be overlooked?
- Cover: How well is a character covered?





Shadow points

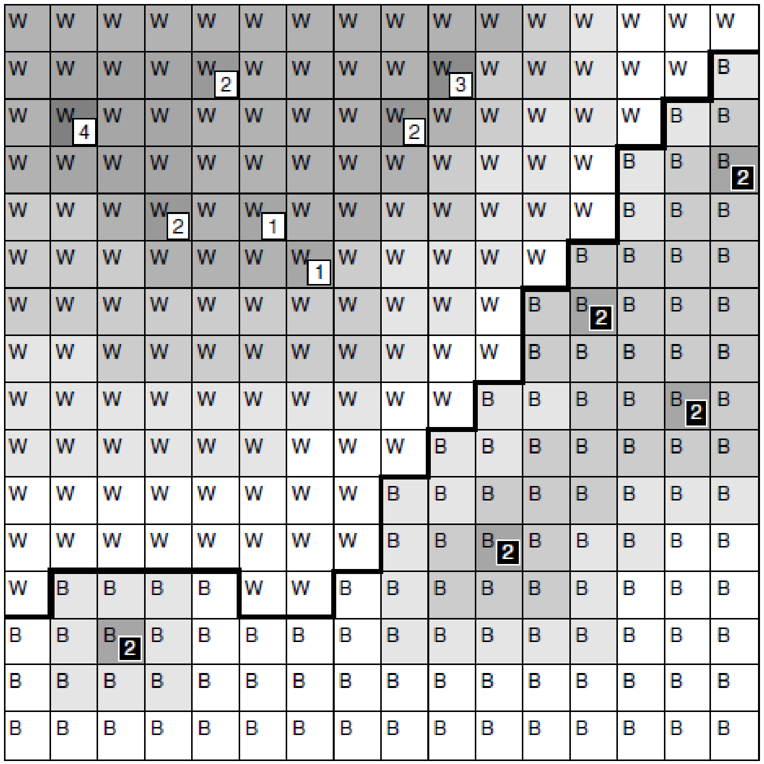
- Cast rays to light sources
- Use an existing global lighting solution

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

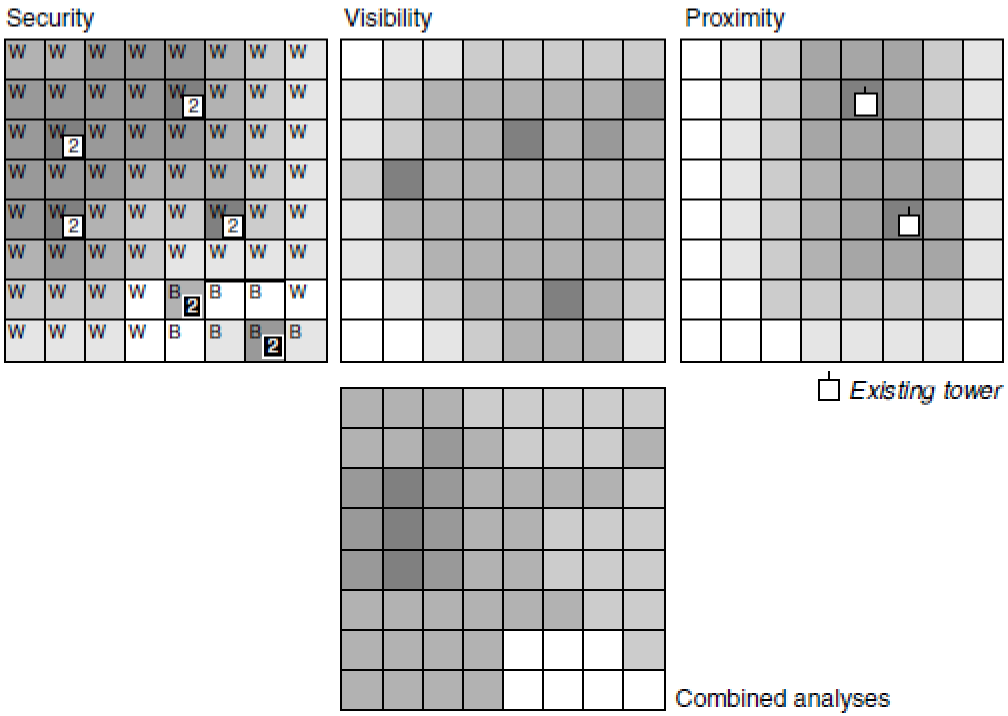


Learning influence maps

Frag Maps

Multiple Layer Analysis

Combine different tactical analyses into one



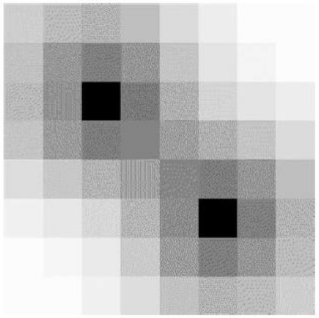
Generating influence maps

Map Flooding

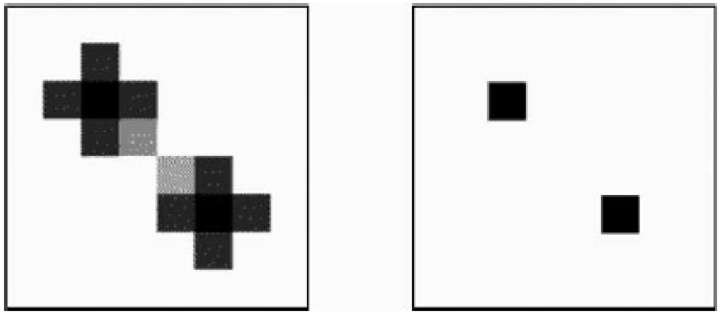
- Use Dijkstra Algorithm variant
- If influences overlap, the stronger influence wins
- Leads to Voronoi Diagram-like map

Filters

- Blur



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

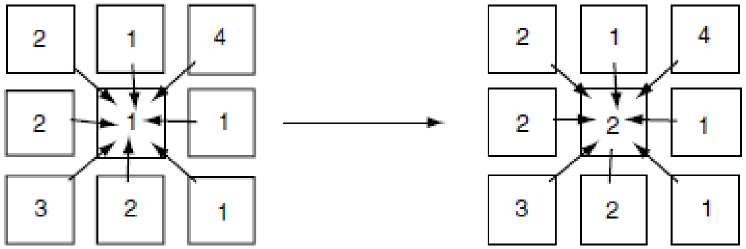


Generating influence maps

Skipped

Cellular automata

- Define rules how tiles are filled based on neighbors
- Used in Sim City



IF two or more neighbors with higher values, THEN increment

IF no neighbors with as high a value, THEN decrement

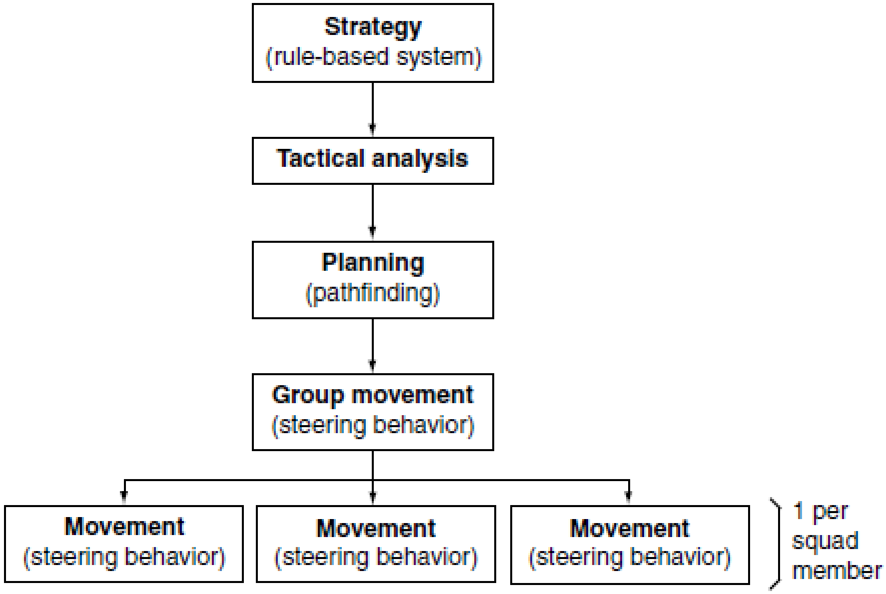
Coordinated Action

Bottom up

- AIs query higher-level systems for tactical data

Top-down

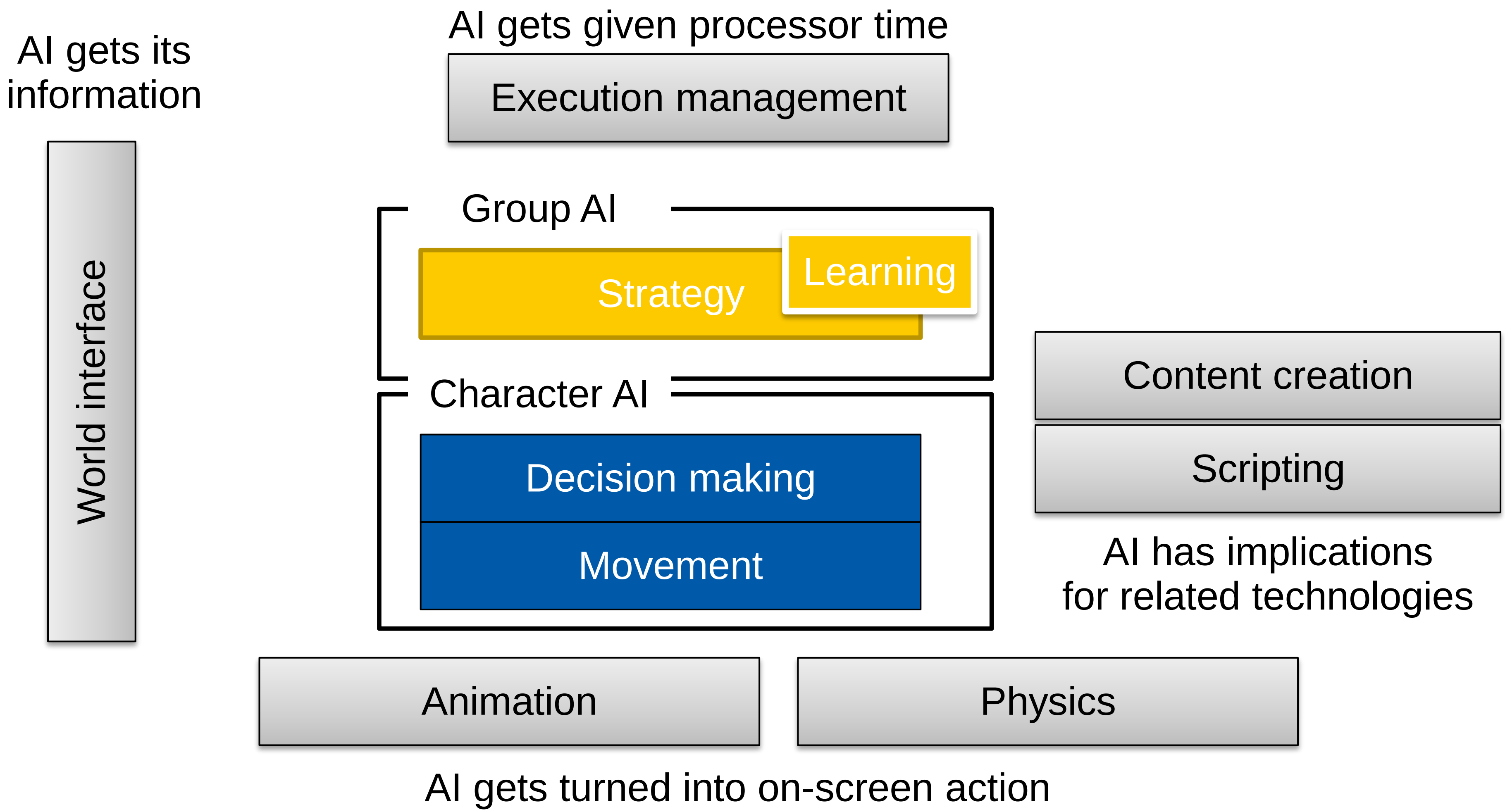
- High-level commands passed downwards



Learning

Skipped

TECHNISCHE UNIVERSITÄT DARMSTADT

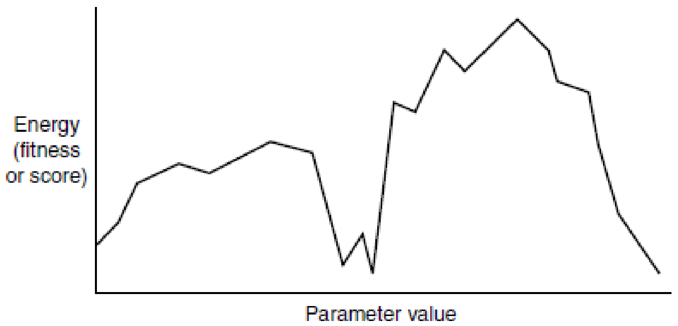


Parameter learning

Skipped

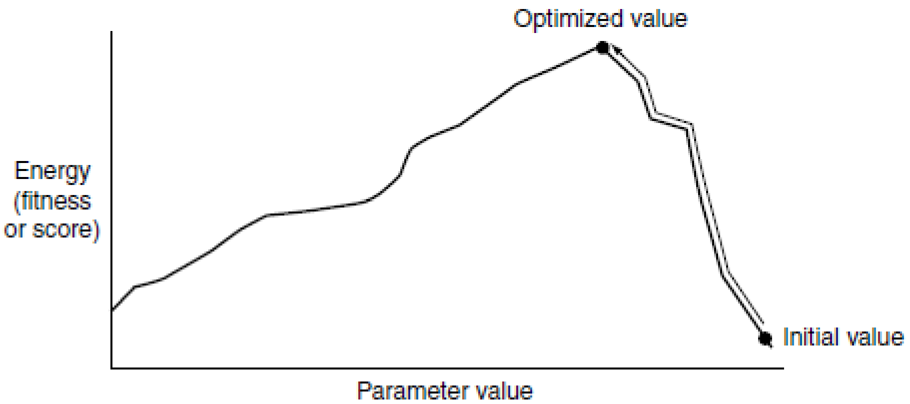
Learn the optimal value of a parameter

- E.g. optimal driving speed, number of zergs in a rush, …



Hill climbing

- Move towards ascending gradient
- Stop when a maximum is reached

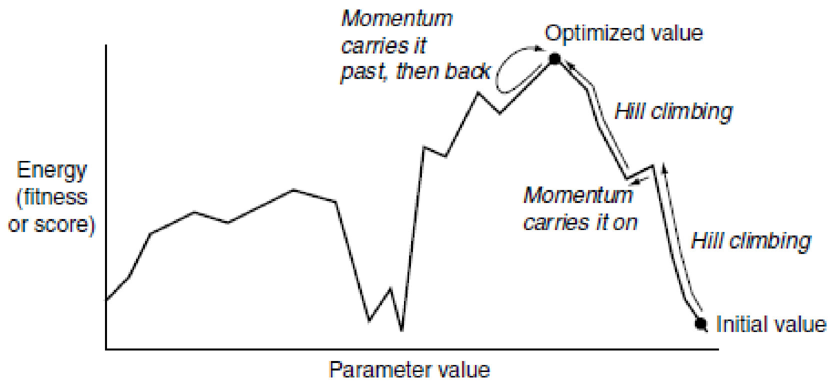


Finding global maxima

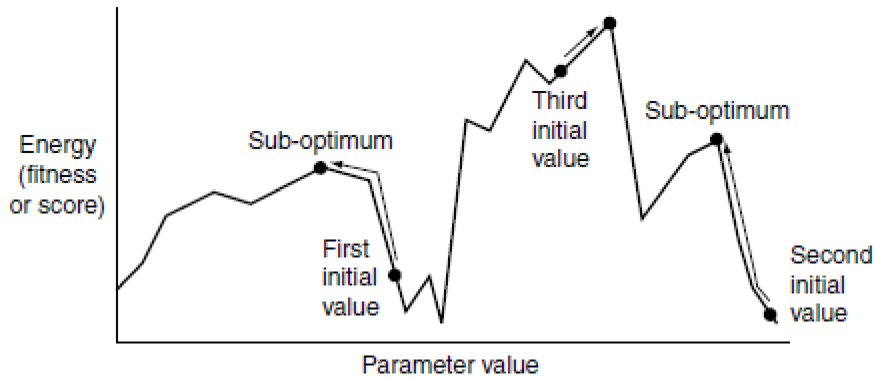
Skipped

TECHNISCHE UNIVERSITÄT DARMSTADT

Momentum



Multiple trials



Action Prediction

Skipped

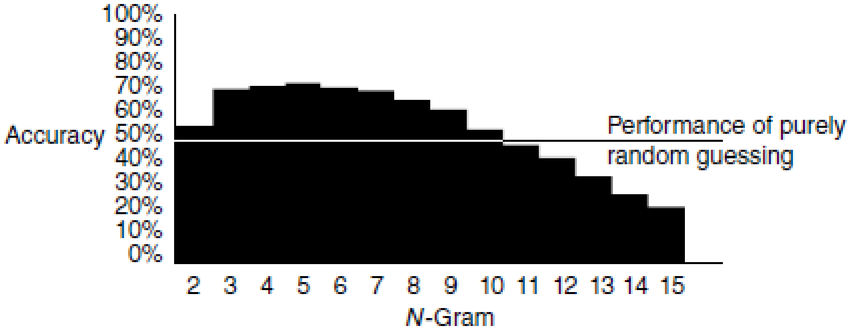
TECHNISCHE UNIVERSITÄT DARMSTADT

Raw probability

- Based on counting previous examples

N-Grams

- Take strings of several characters
- E.g. 2-gram for buttons A and B
  - AA
  - AB
  - BA
  - BB



Window Size

- Quality highly dependent on N

Further learning topics

Skipped

Naive Bayes

- Combine probabilities

Learn decision trees

- ID3
  - Entropy and information gain
- ID4
  - Incremental learning

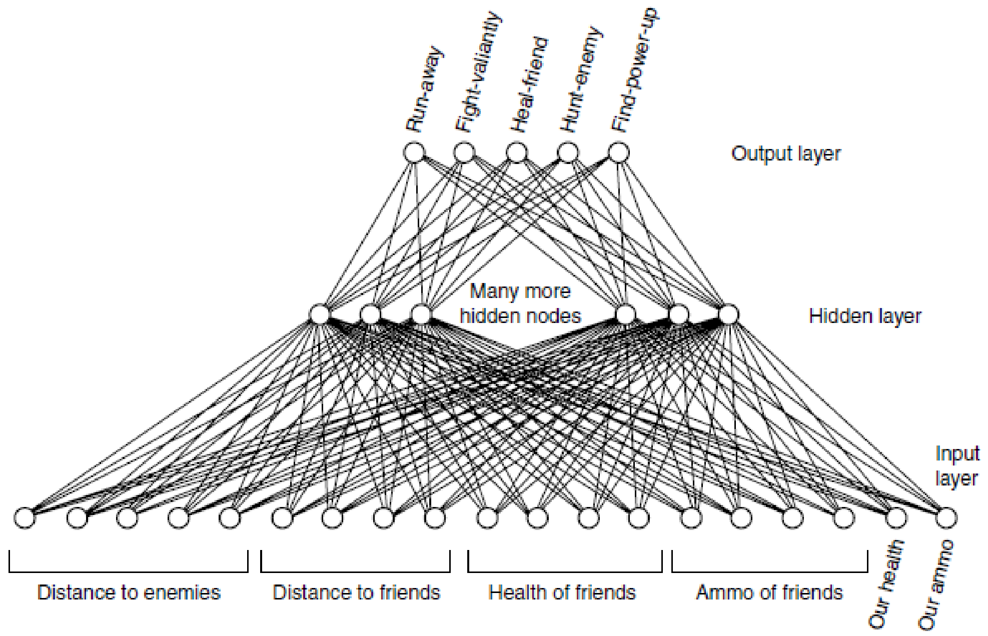
Reinforcement learning

- Learn qualities (Q-Learning) for each state and action

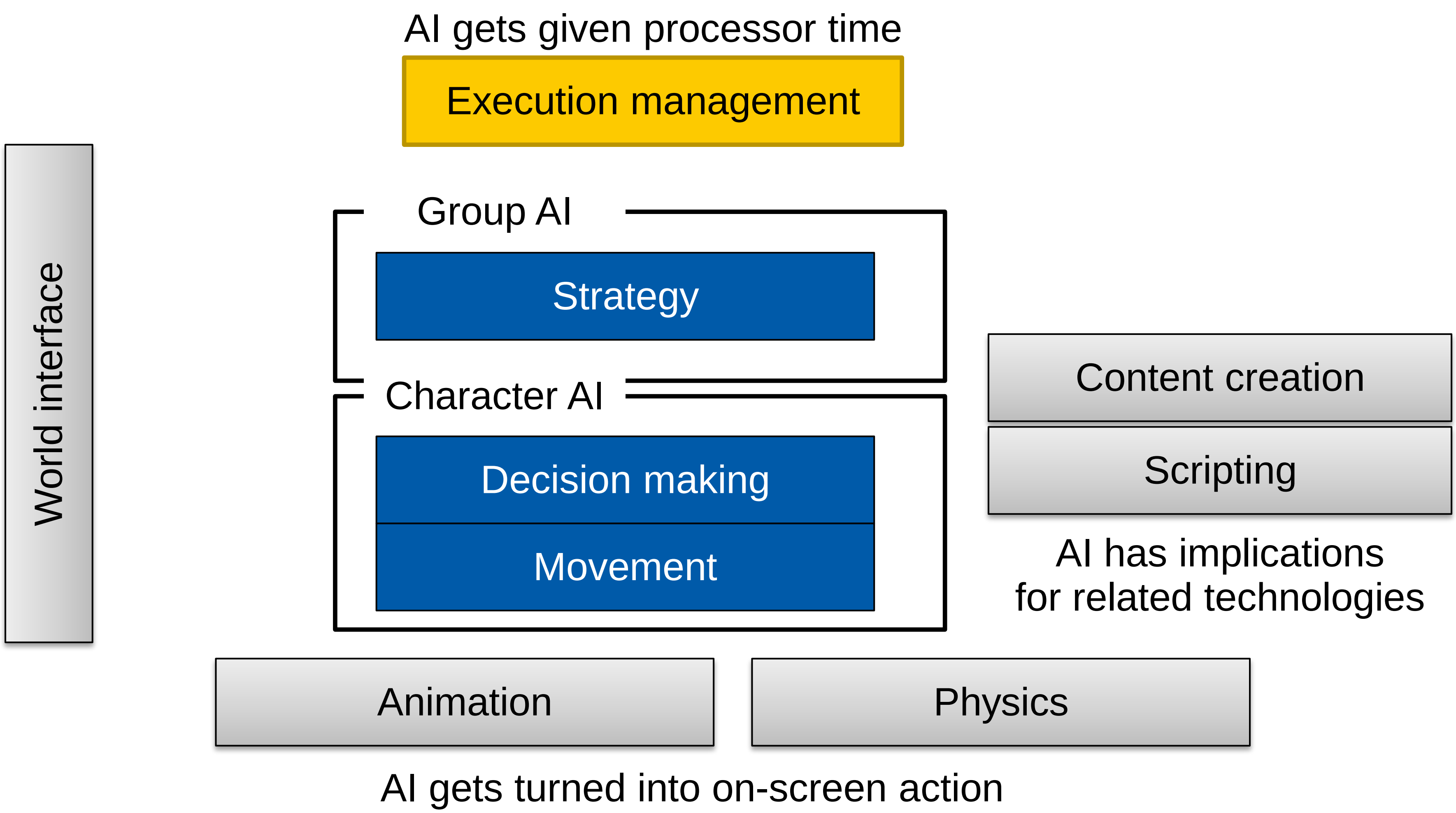
Neural Networks

Skipped

- Model networks of neurons that each have specific values determining the output as a function of the inputs
- Decision-making: Feed a state‘s variables in and observe the output
- Learning: Feed an example state in, compare the output to the intended output, and adjust the values accordingly



Execution management



Scheduling

Frequency

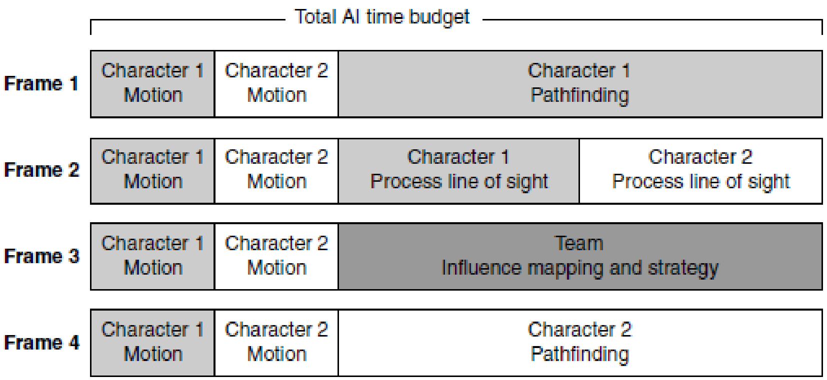
- How often should the AI task be called?

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

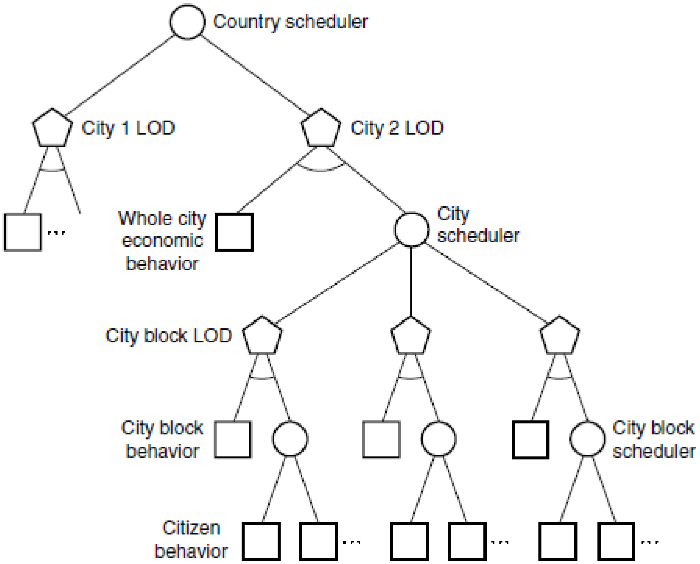
TECHNISCHE UNIVERSITÄT DARMSTADT

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



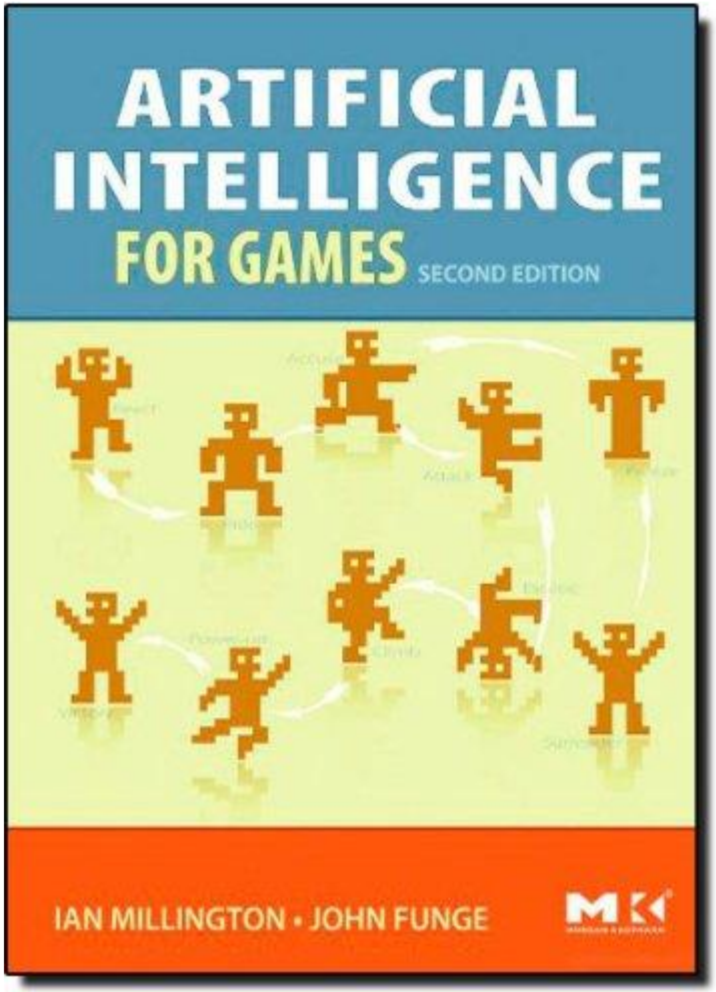
Literature

Artificial Intelligence for Games

- Ian Millington, John Funge

AI Game Programming Wisdom Series

http://aigamedev.com/



Exercise 13 - Sound

TECHNISCHE UNIVERSITÄT DARMSTADT

2.1 Doppler Effect

Consider a car driving with 150 km/h and a person running away from the car with 15 km/h. The car emits lots of different sound effects. How much does the frequency of those sound effects change for the person when the car passes him?

f1 = f * (c - 15 km/h) / (c - 150 km/h)

f2 = f * (c + 15 km/h) / (c + 150 km/h)

fdif = abs(f1 - f2)

Exercise 13 - Sound

2.2 Sound location simulation without headphones

Directional sound can be simulated effectively using headphones. Can this also be done using regular speakers? What are the expected limitations?

Only for one person.

Exercise 13 - Sound

TECHNISCHE UNIVERSITÄT DARMSTADT

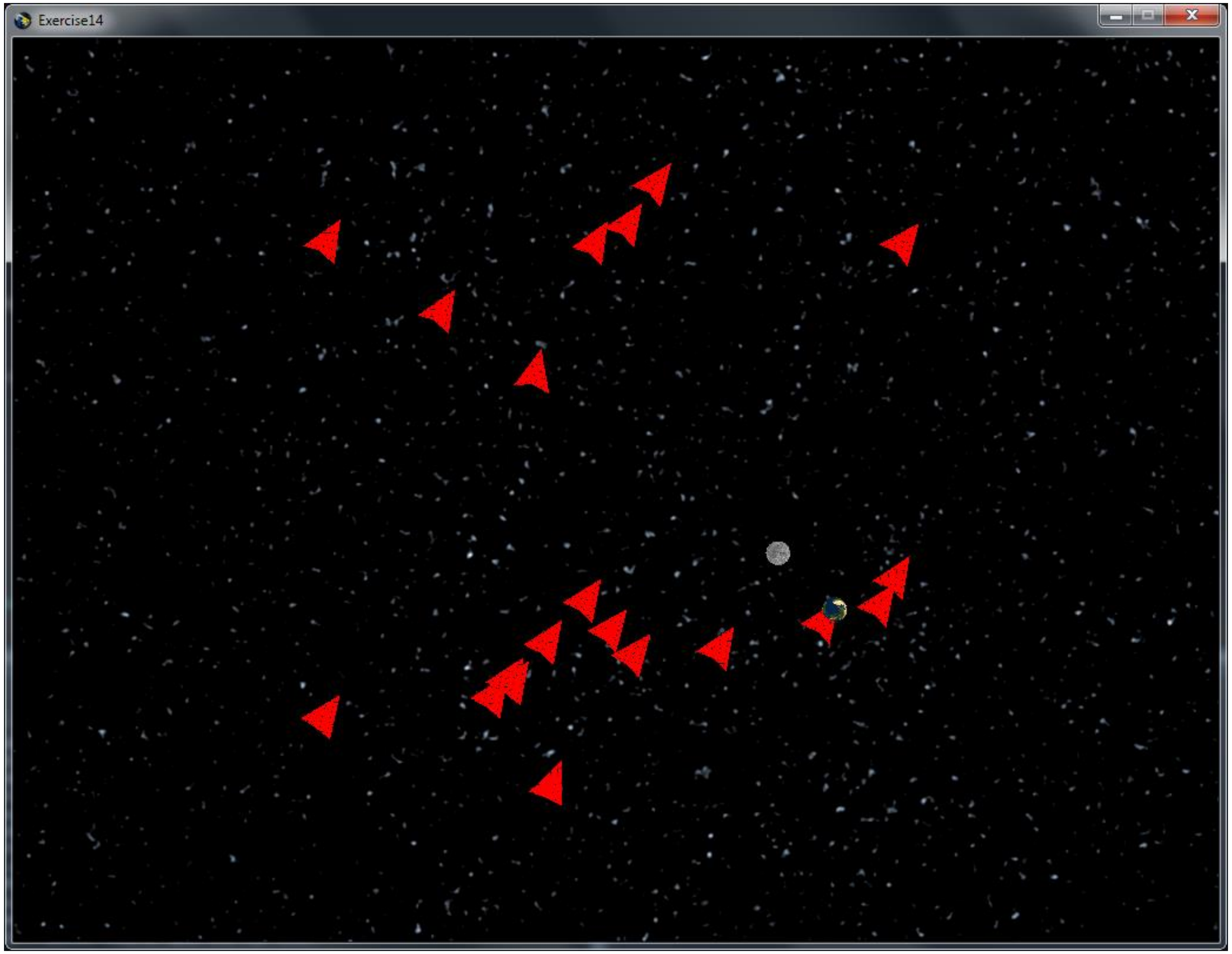
2.3 Sound reflection data

Considering the data available to a physically based rendering engine – what data can be reused to simulate realistic sound reflections?

Geometry, normal maps, roughness maps

Exercise 14 - AI

Exercise 14 – AI – Practical Tasks



Exercise 14 – AI – Practical Tasks

TECHNISCHE UNIVERSITÄT DARMSTADT

Flocking

- Implement Flocking as a blended steering behaviour involving
  - Separation
  - Cohesion
  - Alignment of velocity

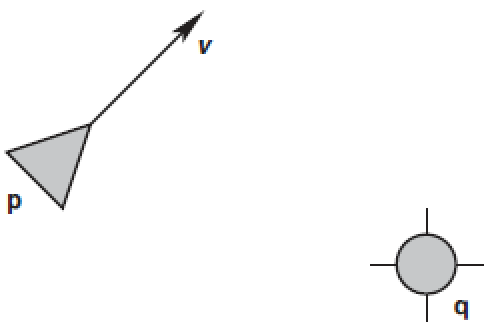
Decision Making

- Implement a finite state machine for the moon
  - If it is not close to the earth: Wander
  - If it is close to the earth: Seek

Exercise 14 – Theoretical Tasks

Task 2.1

- For a given configuration, give the steering values for
  - Seek
  - Flee
  - Pursuit
  - Evade



Exercise 14 – Theoretical Tasks

TECHNISCHE UNIVERSITÄT DARMSTADT

Task 2.2

a) Can A* be interrupted during its execution and continued at a later time? If yes, describe why. If not, give a counterexample.

b) Are there problems to be expected if the next time the pathfinding continues is a certain amount of time (e.g. 1 second) later?

c) If a) is true, does this make A* an Anytime algorithm?

Exercise 14 – Theoretical Tasks

TECHNISCHE UNIVERSITÄT DARMSTADT

Task 2.3

Imagine a scene with many characters walking in random directions. In this scene, an AI LOD system is used. Nearby characters use collision avoidance, far away characters may freely interpenetrate.

What problems could arise if characters exploded when they collided?

Questions & Contact

game-technology@kom.tu-darmstadt.de