Game Technology

Lecture 11 – 19.12.2015
Artificial Intelligence in Games

Black & White (2001)

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

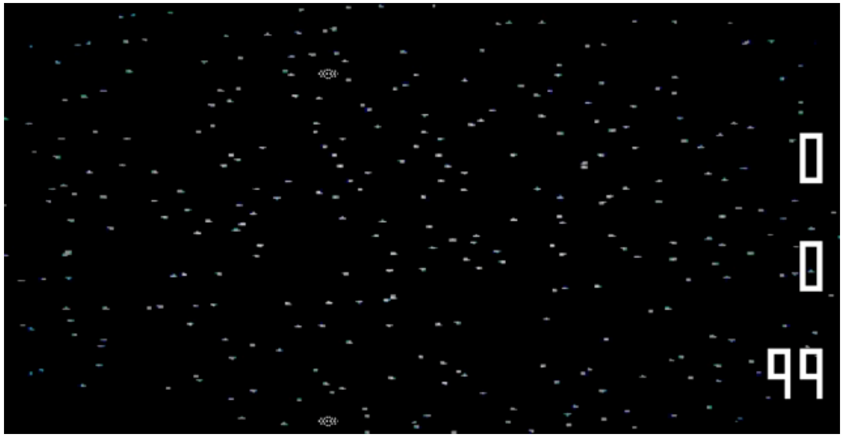
Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

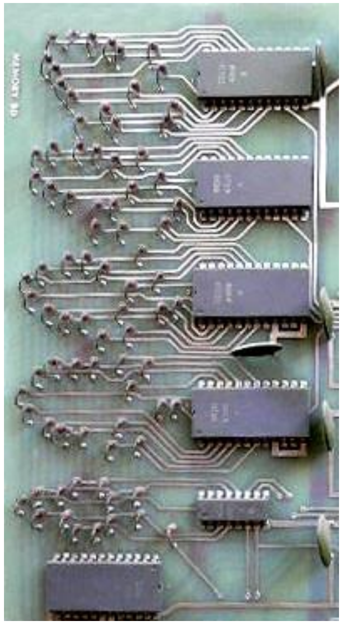
TECHNISCHE UNIVERSITÄT DARMSTADT

Introduction

Computer Space (1971)

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





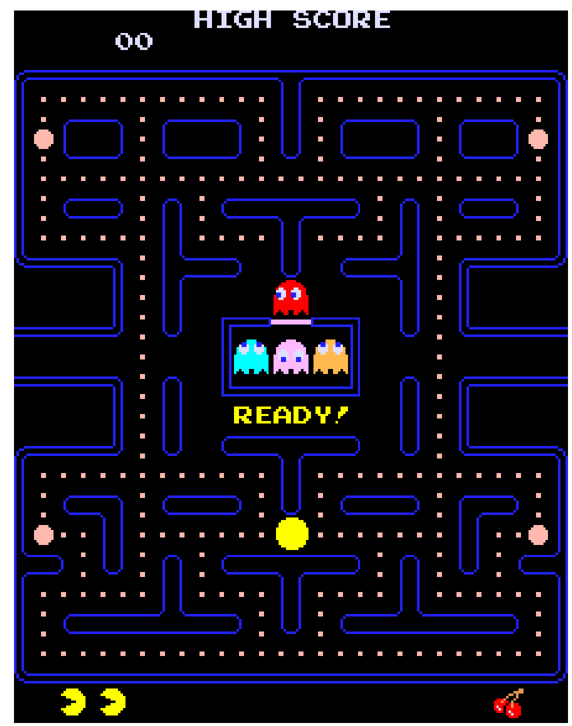
Introduction

Pac Man (1980)

- Tōru Iwatani
- Four different ghost „personalities“

E.g. red ghost

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



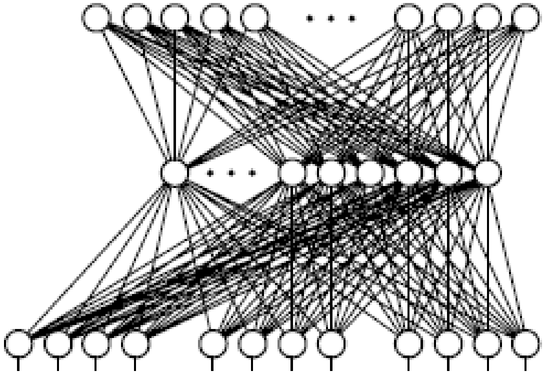
More info

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

Introduction

Creatures (1996)

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

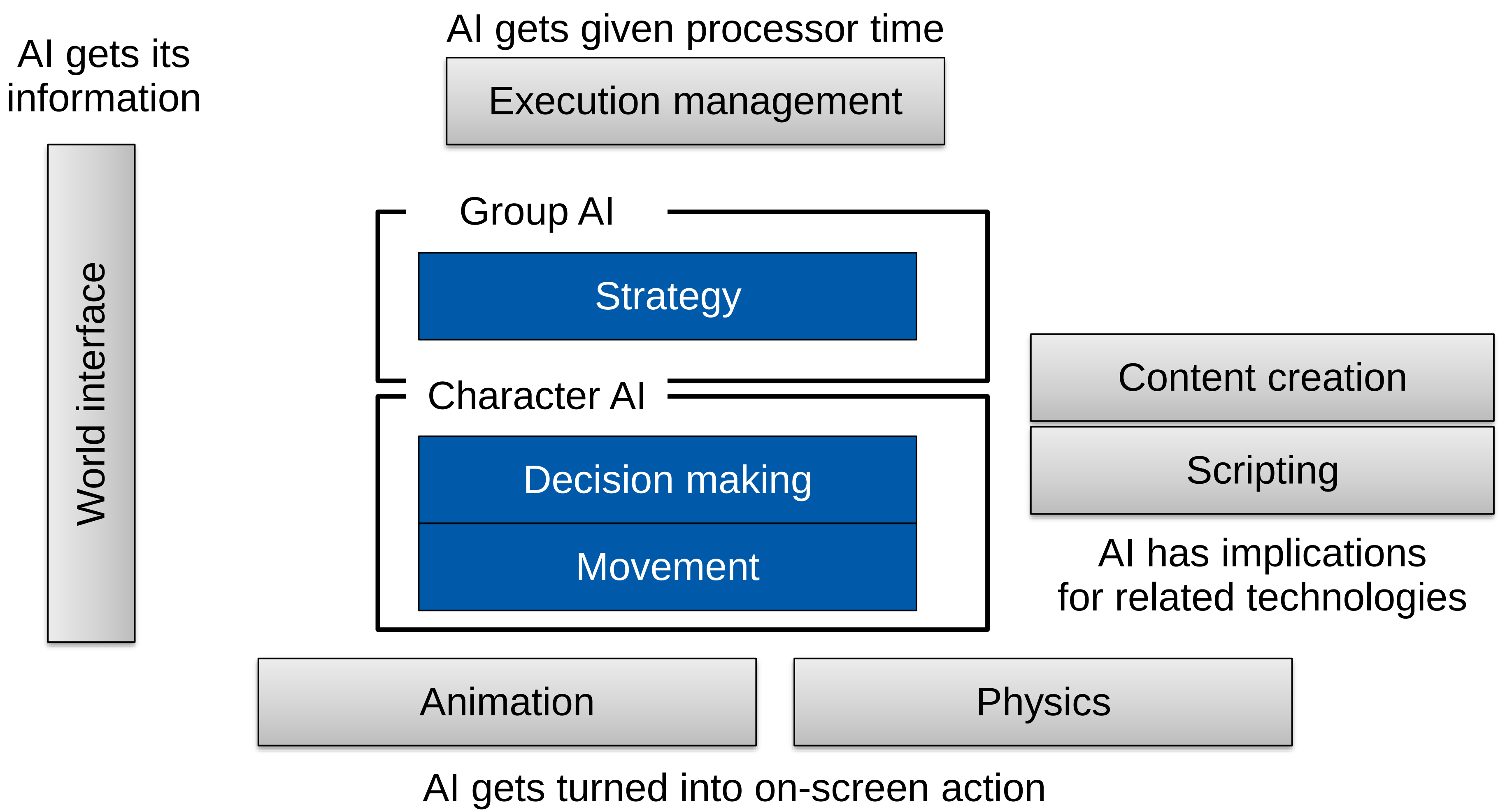


Introduction

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

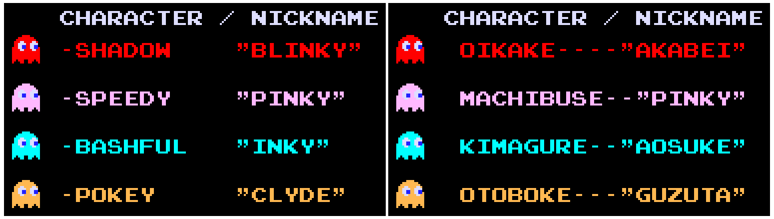

AI gets given processor time
Execution management
AI gets its information
World interface
Group AI
Strategy
Character AI
Decision making
Movement
Content creation
Scripting
AI has implications for related technologies
Animation
Physics
AI gets turned into on-screen action

Complexity

Simple things can look good

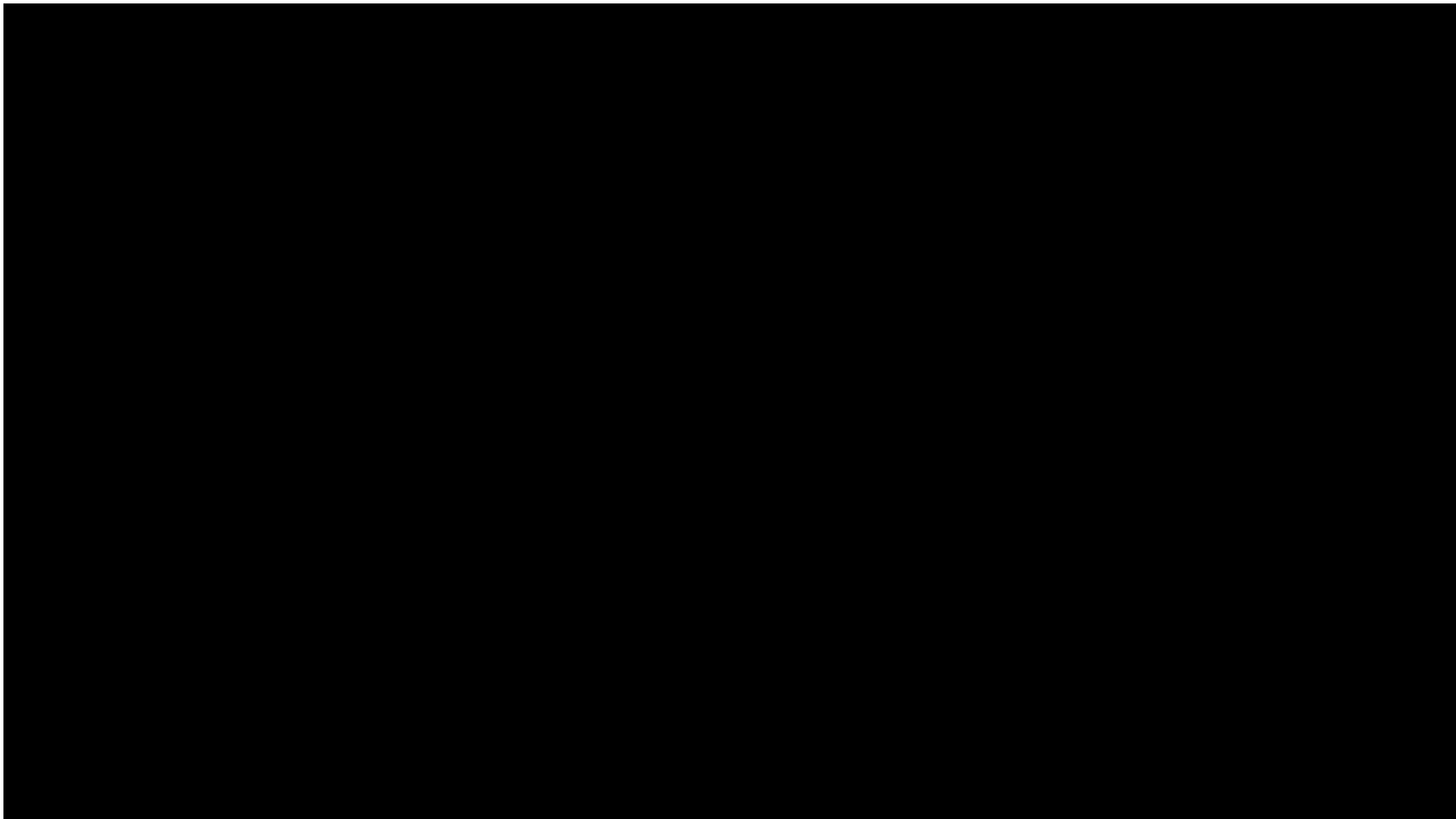
- ELIZA (1966)
- > Hello. My name is Eliza. How may I help you?
- * I want to give a lecture on AI for games.
- > Why do you want to give a lecture on AI for games?
- * It is part of a lecture series on game technology.
- > Say, do you have any psychological problems?
- * Maybe...
- > You don't seem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

Pac Man



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

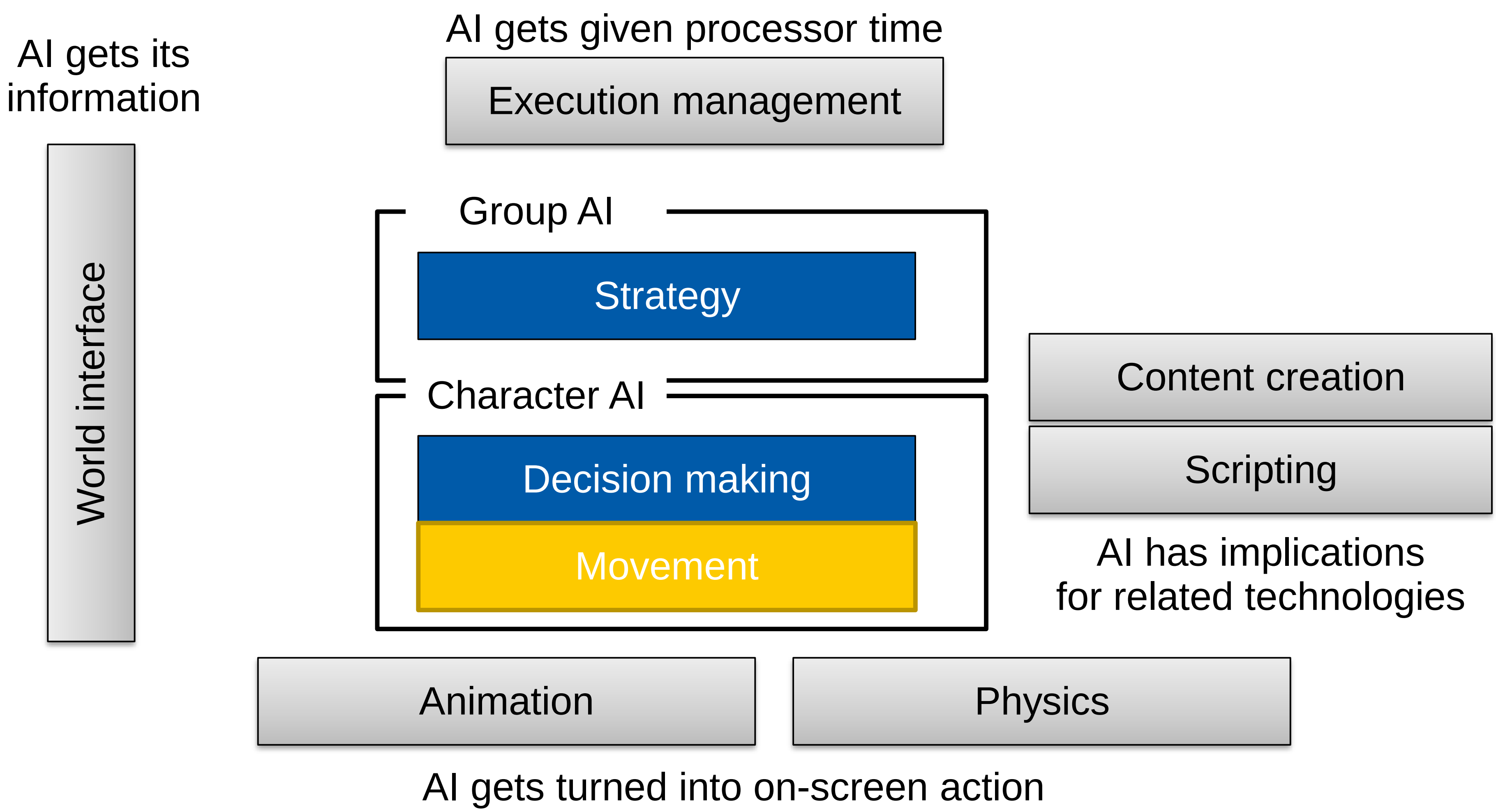
Heuristics

- 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

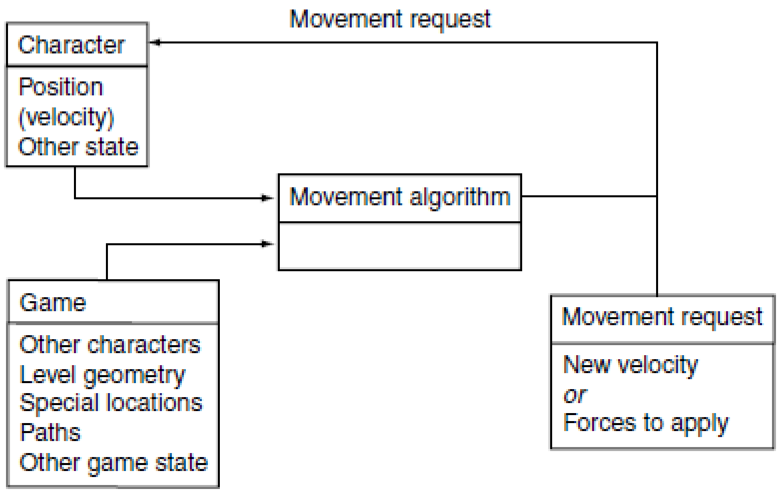


Source: Artificial Intelligence for Games, 2nd Edition

Movement

Input

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



Output

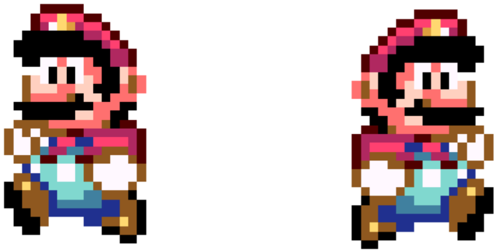
- Movement requests

Kinematic vs. Dynamic movement

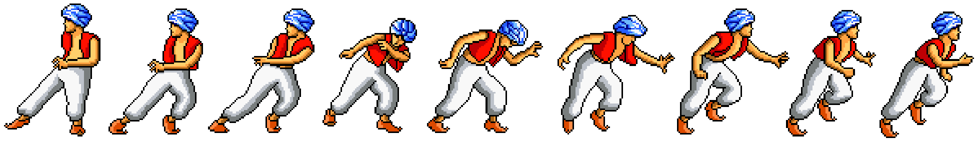
- Kinematic – Update the velocity directly
- Dynamic – Take into account velocities, change via accelerations (see physics lectures)

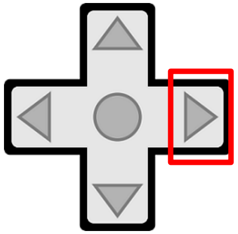
Movement comparison

Kinematic



Dynamic

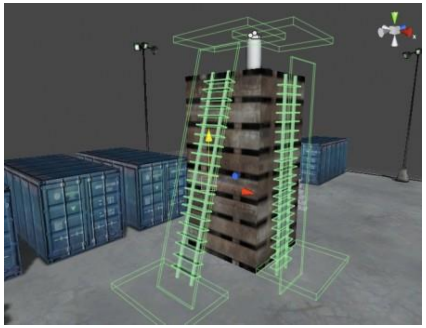




Dimensionality

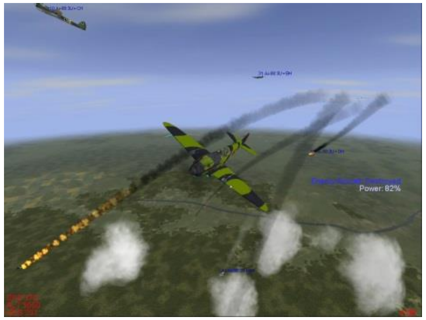
2D

- 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

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

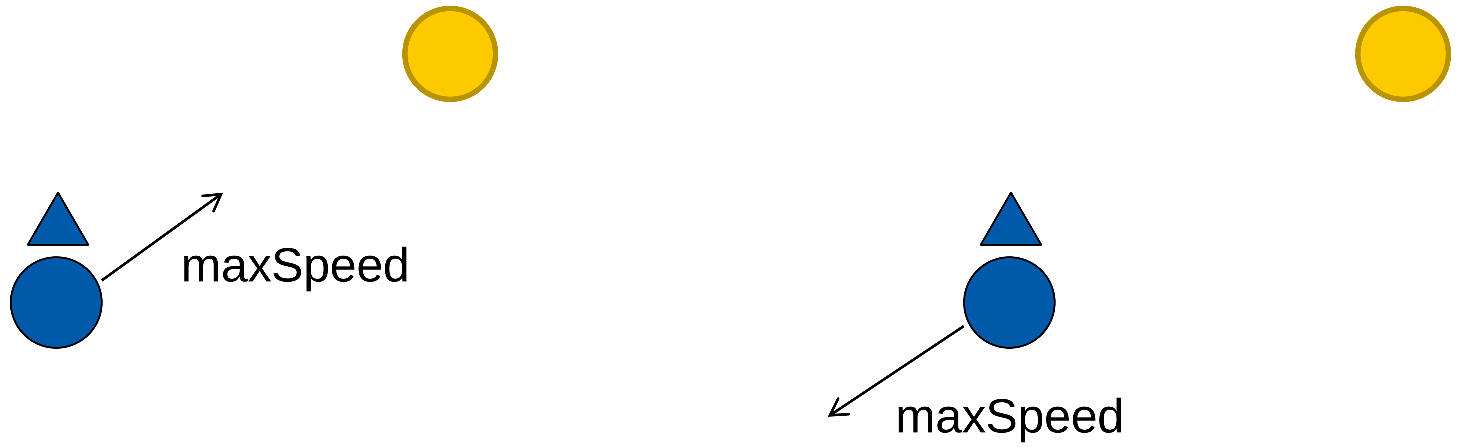
Simple movement behaviours (kinematic)

Seek

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

Flee

- Reverse direction from Seek



Simple movement behaviours

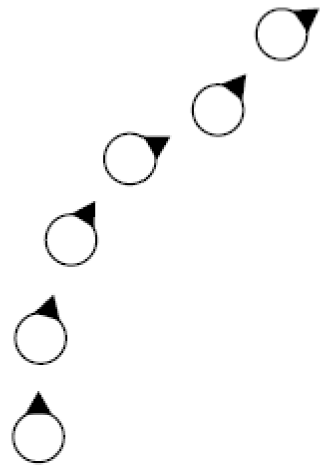
Arrive

- 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

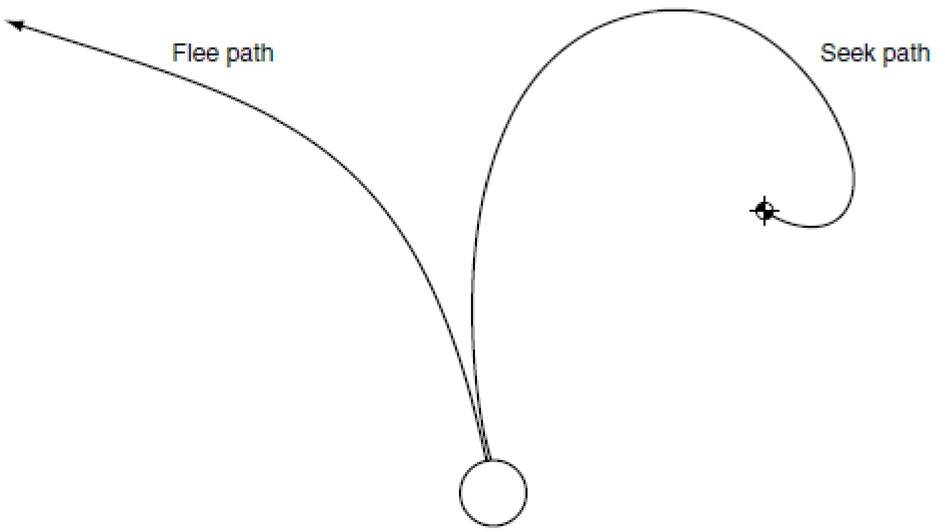


Dynamic Steering

Take into acount rotation and velocity

Seek and Flee

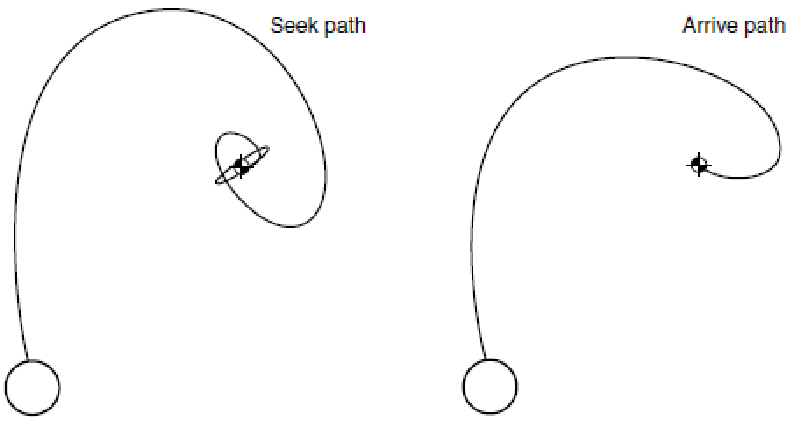
- As before: Choose direction to goal
- Output an acceleration in that direction
- Maximum speed: Either via clipping or via drag



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

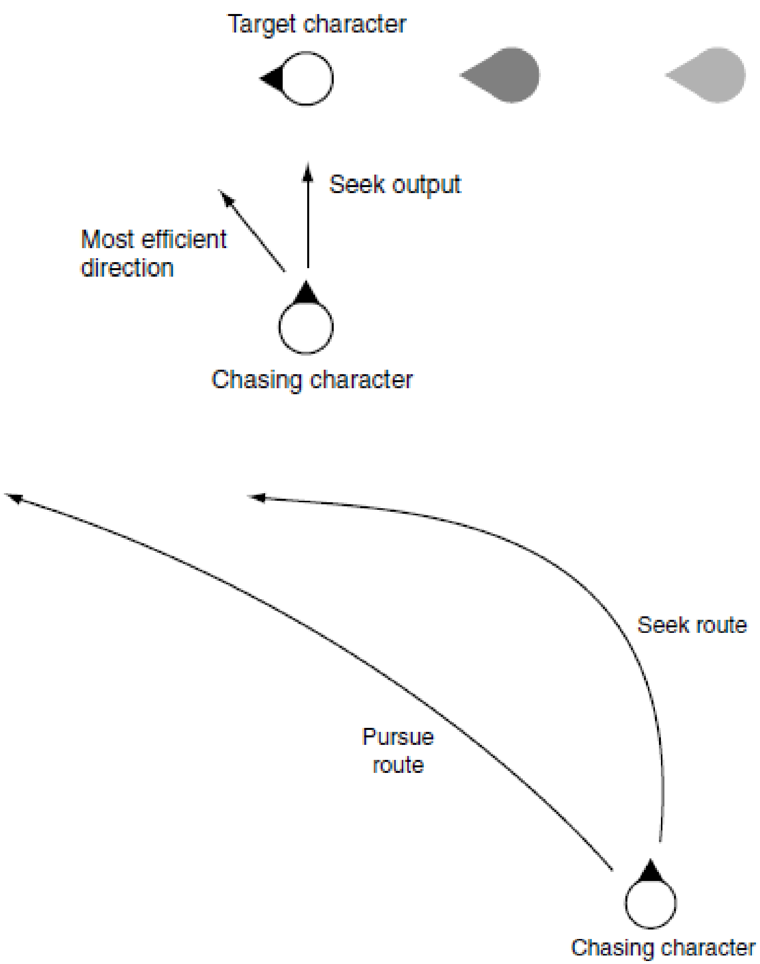
Based on the primitive behaviours we have have looked at until now

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

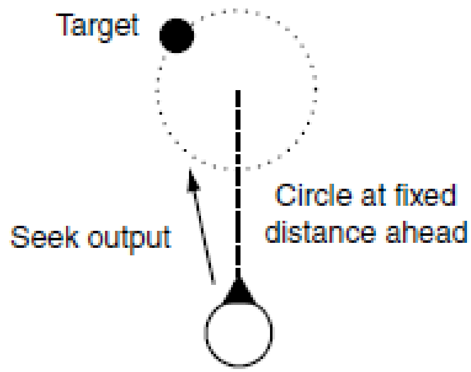
- Opposite of pursue



Delegated behaviours

Face

- Align to the target



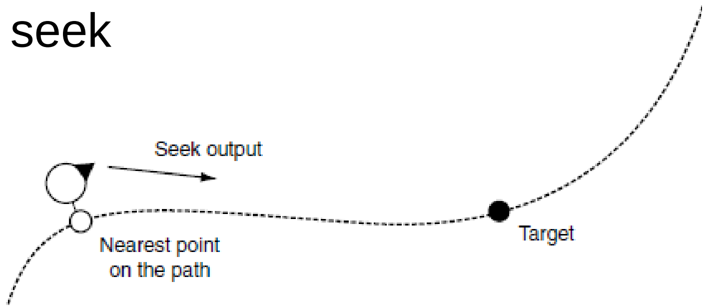
Looking where you are going

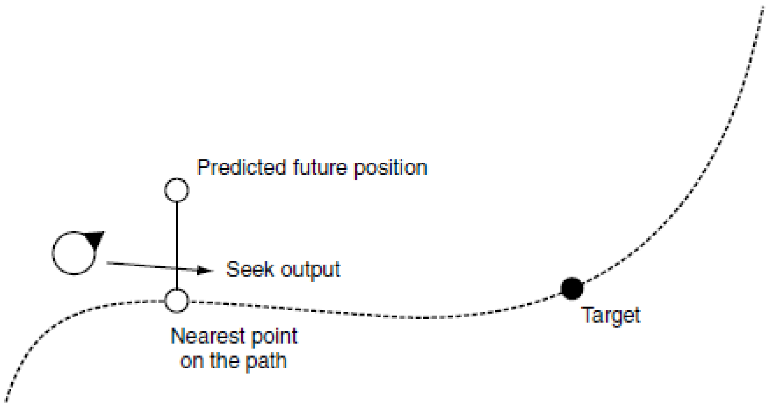
Wander

- Uses Seek

Path following

- Uses seek





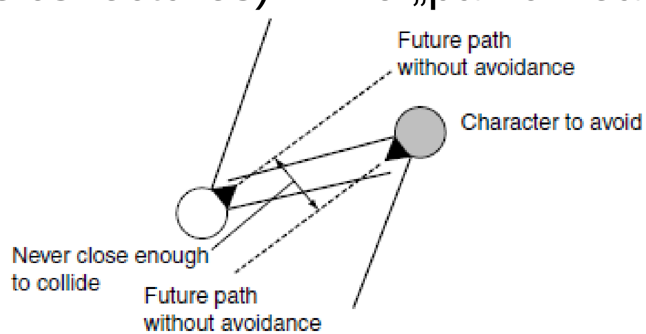
Delegated behaviours

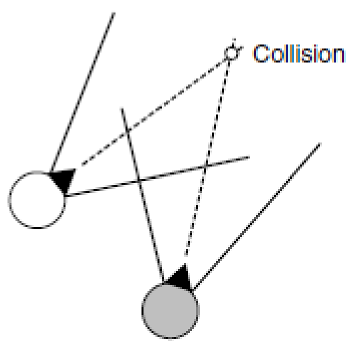
Separation

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

Collision Avoidance

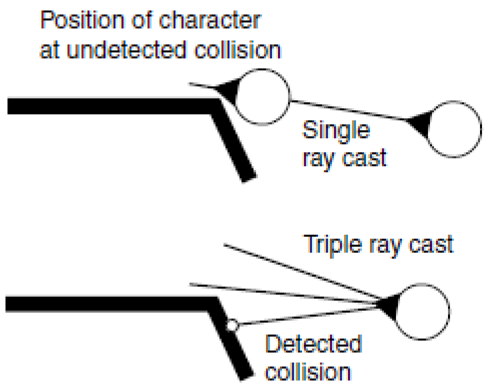
- For characters that move on crossing trajectories
- a) Only do separation inside a cone infront of the character
- b) Only handle collisions that will take place (similar to collision detection in the physics lectures) → no „panic“ reactions





Obstacle and Wall Avoidance

Similar, but distinct from collision detection in physics

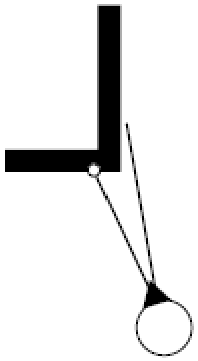


Basic mechanism: Use ray casts

- 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



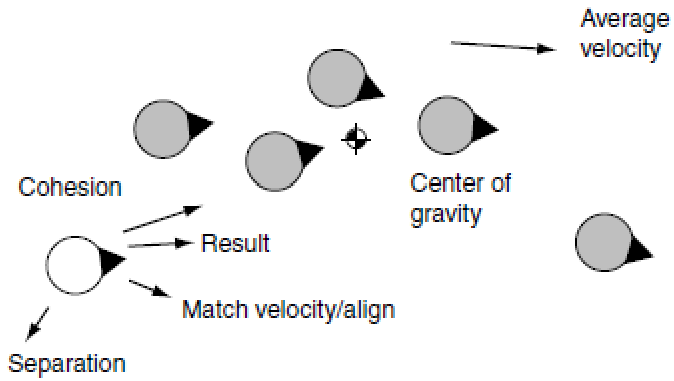
Combining steering behaviours

(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

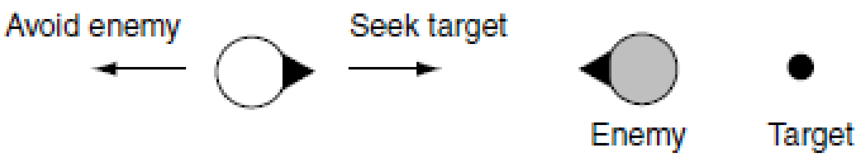


- Blend of 3 steering factors
  - Separation (move away from close boids)
  - Move in the same way as the flock (match avg. velocity and orientation)
  - Cohesion (steer towards center of the flock)

Combining steering behaviours

Equilibria

- Conflicting steering behaviours lead to deadlocks

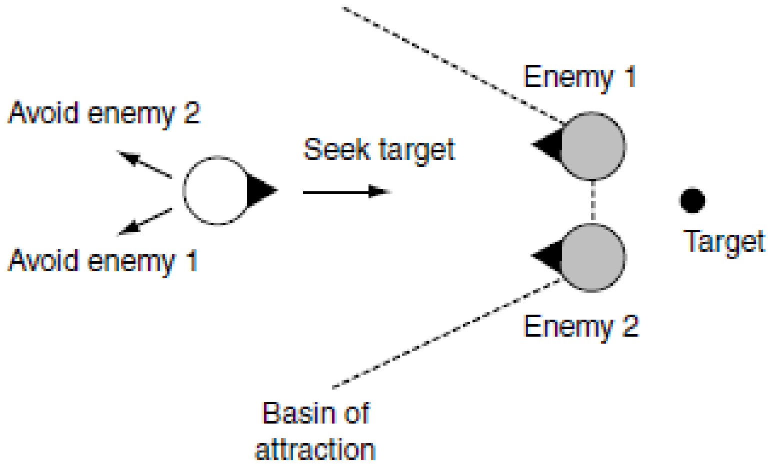


Unstable equilibria

- Situation is inpropable
- Basin of attraction is small

Stable equilibria

- Less likely to be broken by numerical inaccuracies

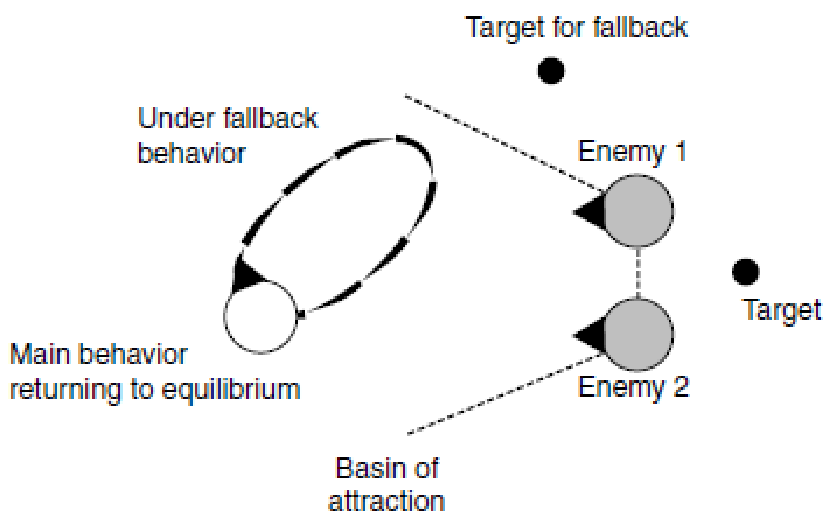


Priorities

Define priorities for steering behaviours

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

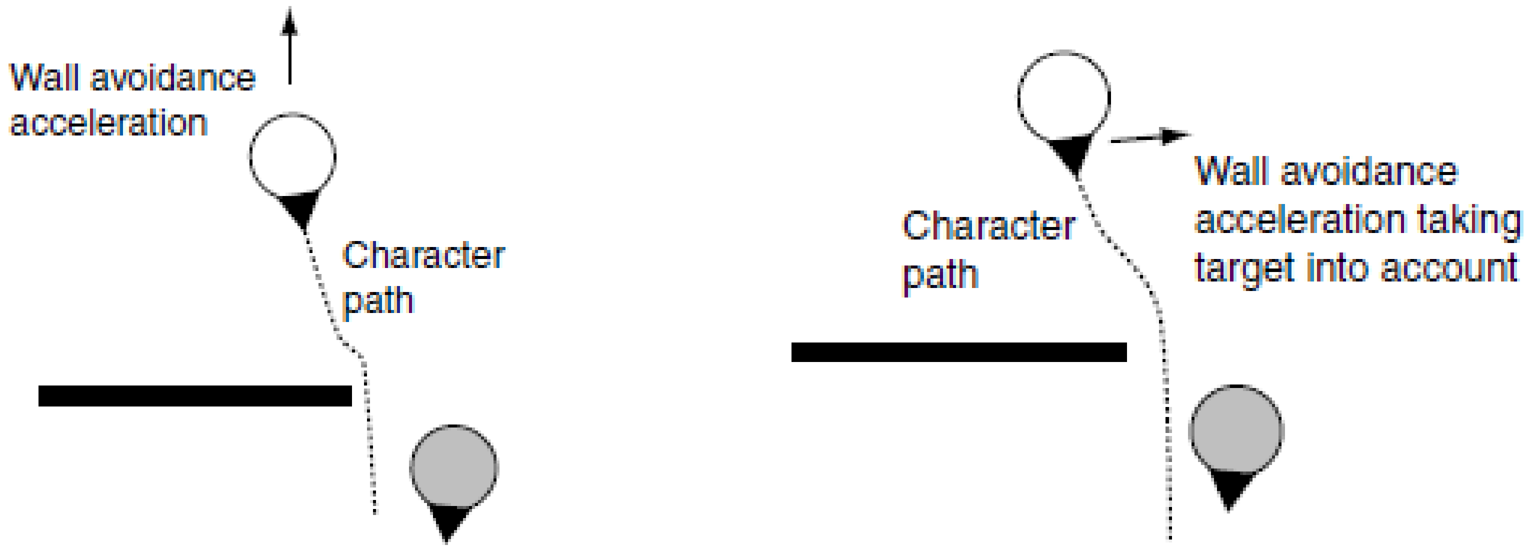
Can still oscillate for stable equilibria



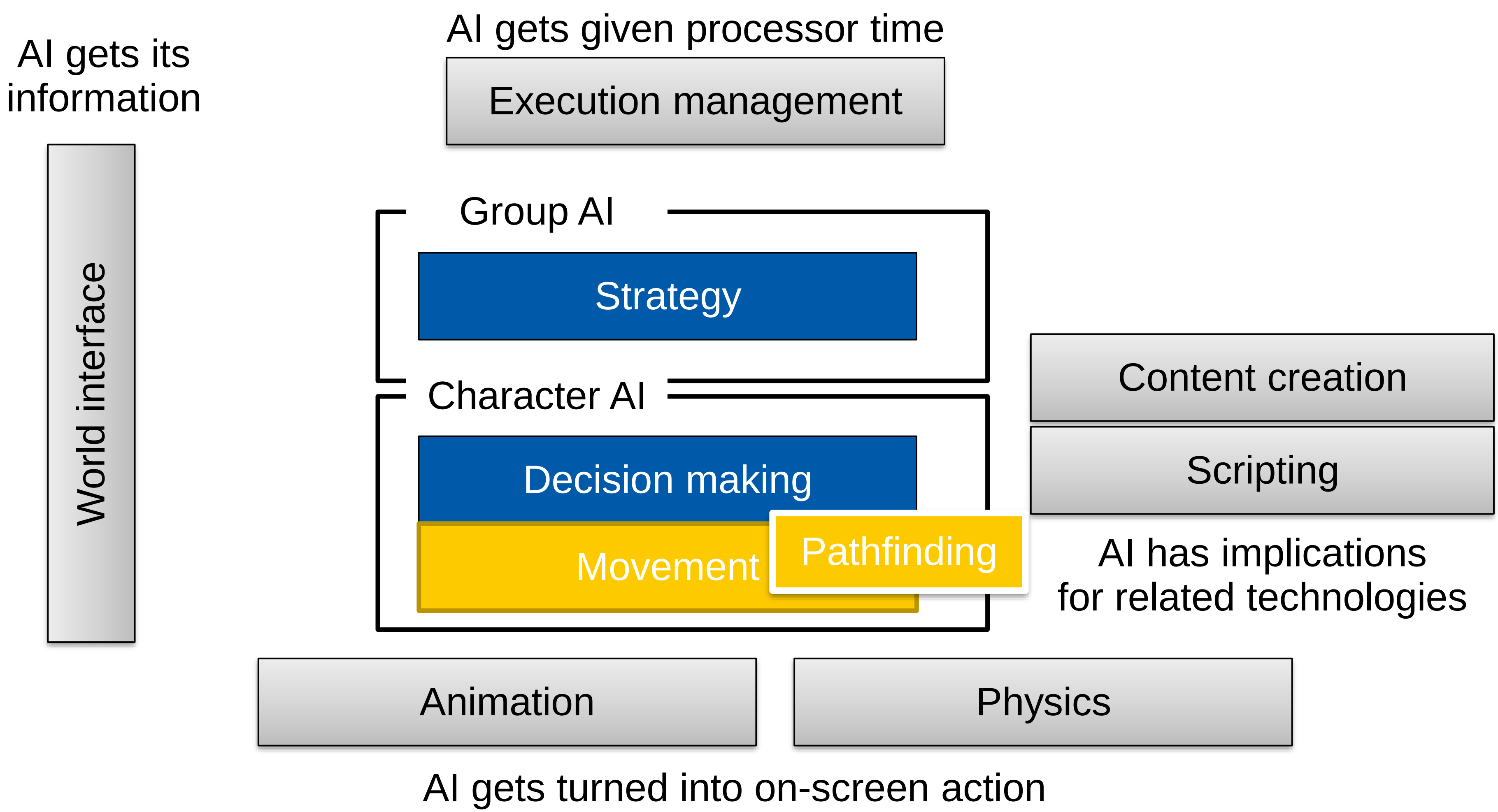
Cooperative arbitration

TECHNISCHE UNIVERSITÄT DARMSTADT

Communication between steering behaviours



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)

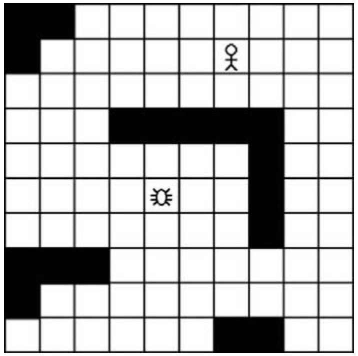
Finding a path

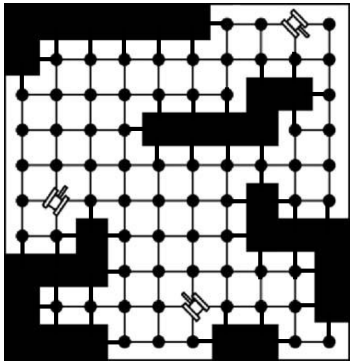
TECHNISCHE UNIVERSITÄT DARMSTADT

A*-Algorithm is the prevalent solution

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

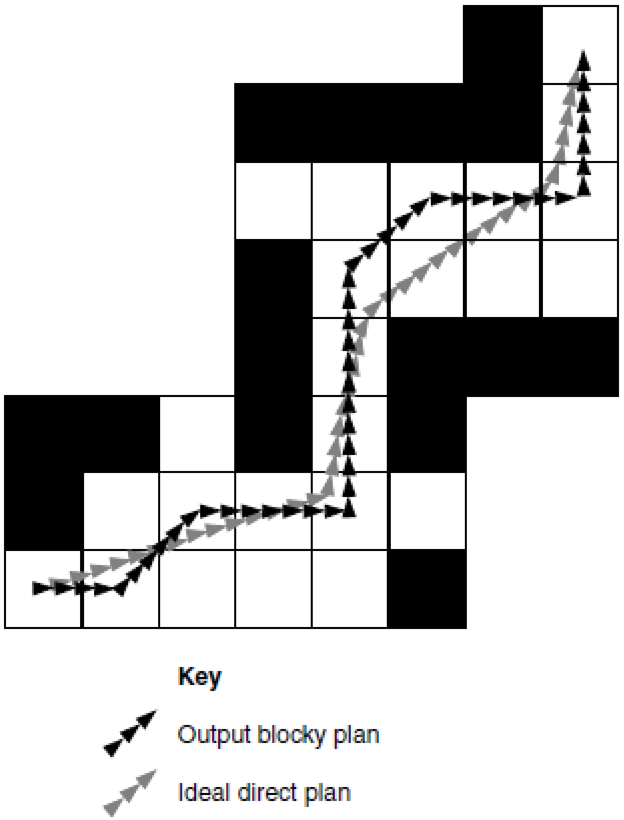
Localization

- Find the world location of a graph node

Tile-based

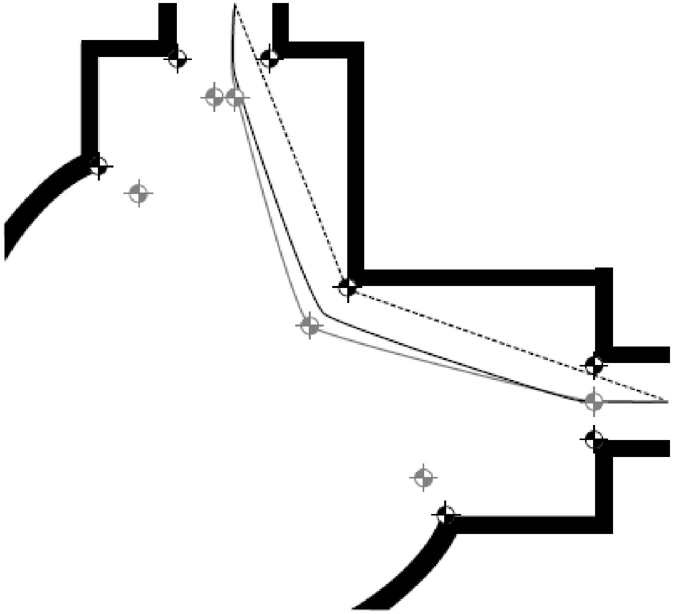
- Well-suited if game world is already represented by tiles (e.g. heightmap)

- Plans can be blocky due to rectangular nature



Points of visibility

Based on the edges of the world



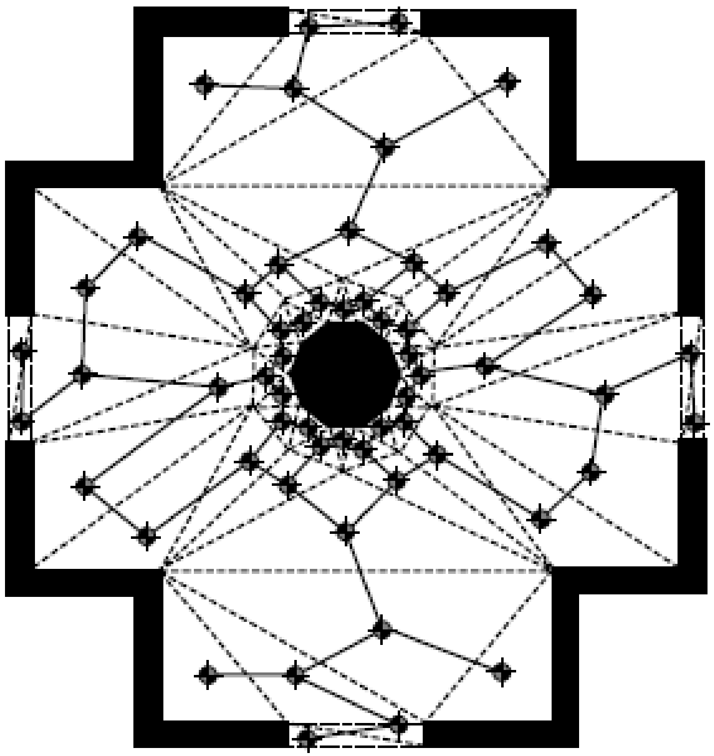
Key

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

Use the mesh of the floor as the basis



Key

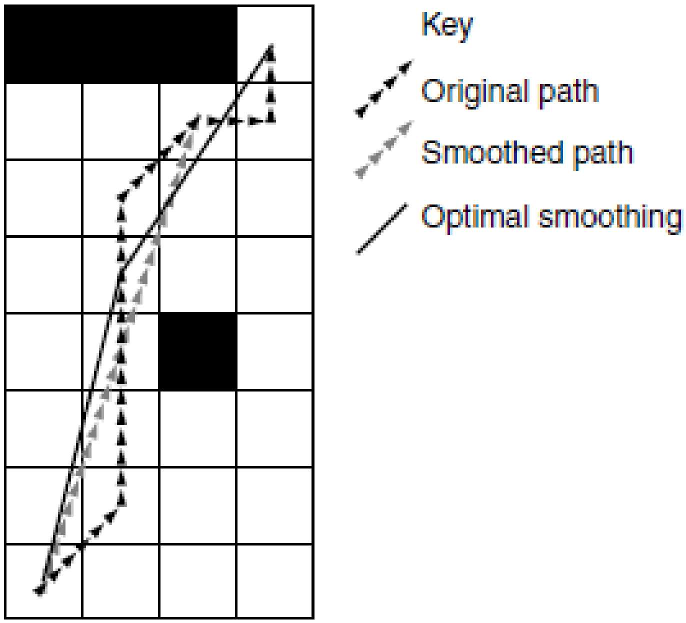
Edge of a floor polygon

Connection between nodes

Path smoothing

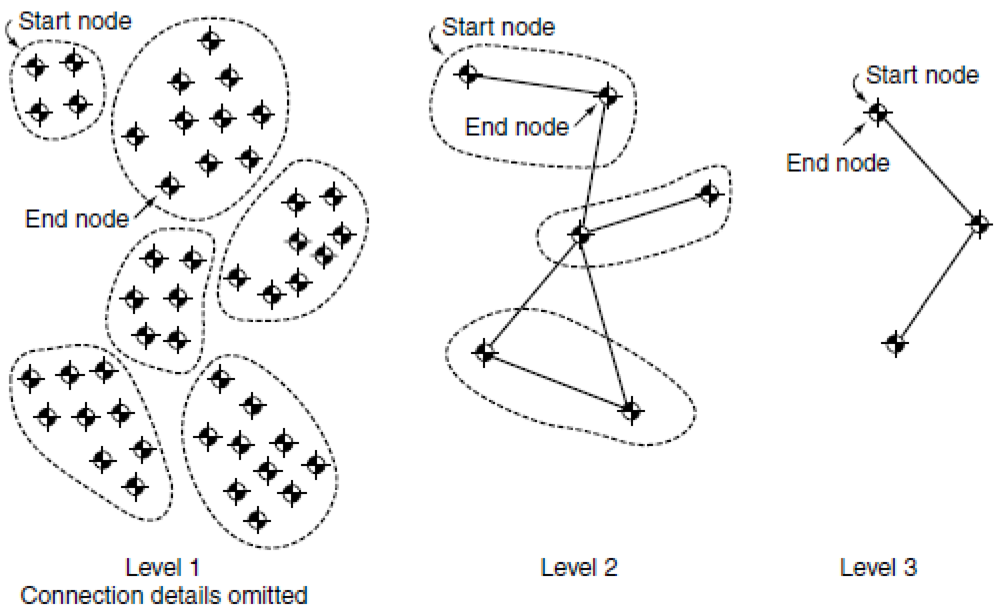
Filter paths to remove sharp edges

- 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

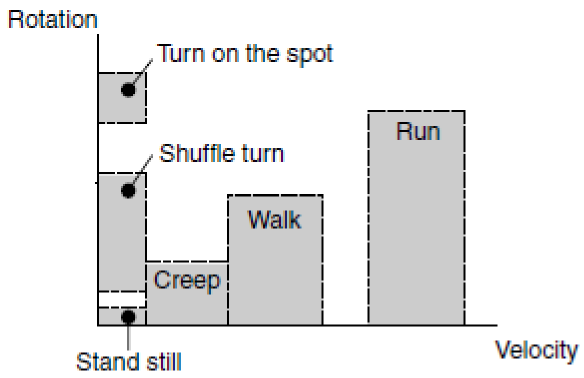


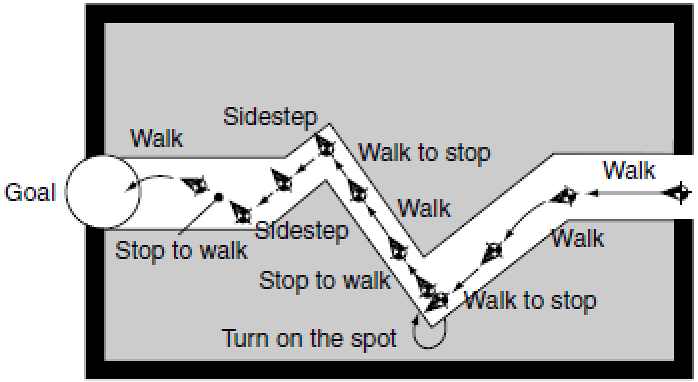
Movement planning

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

- Limited to certain directions
- Limited to certain speeds

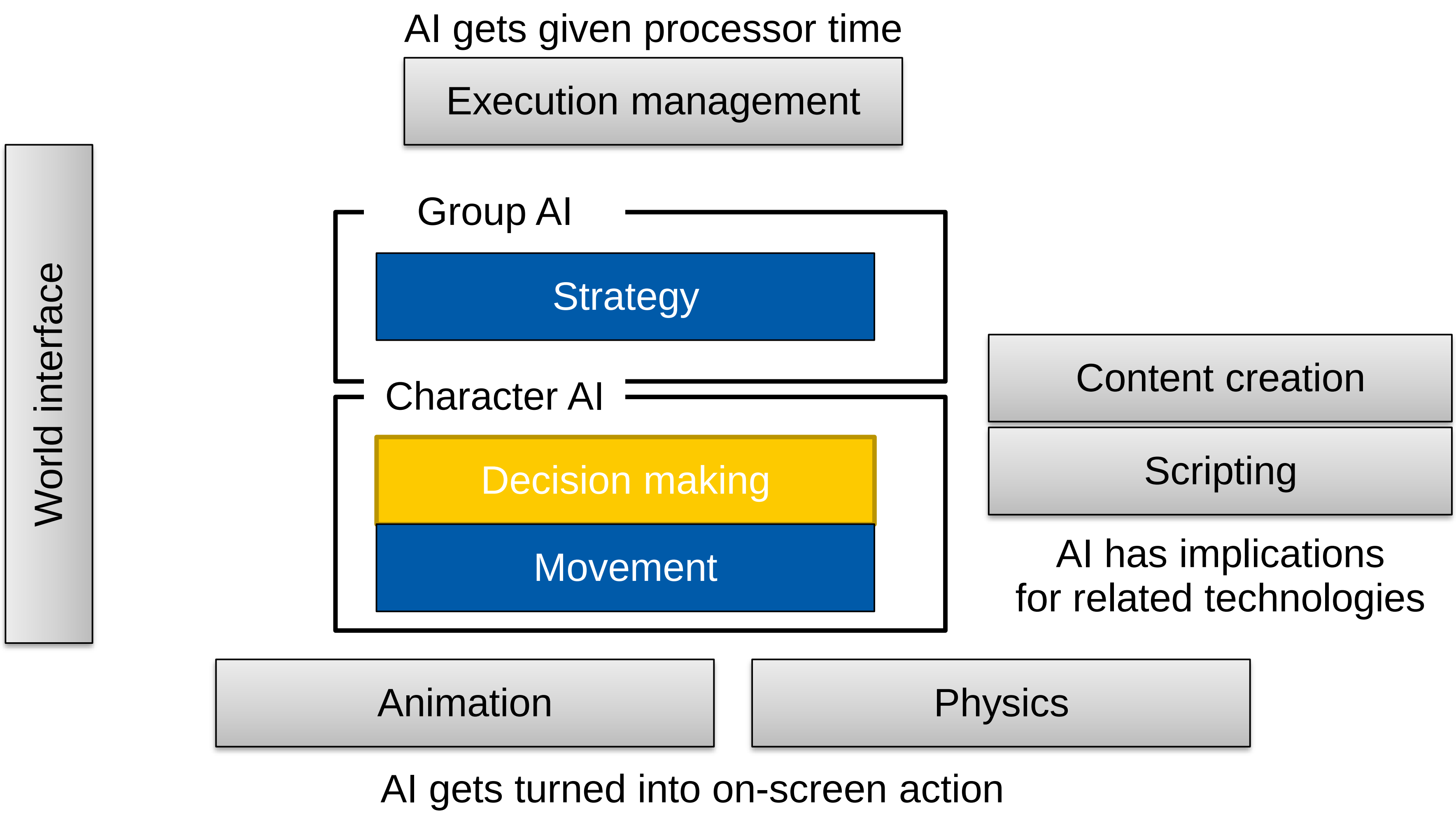
Use A* variant to search good combinations





Decision Making

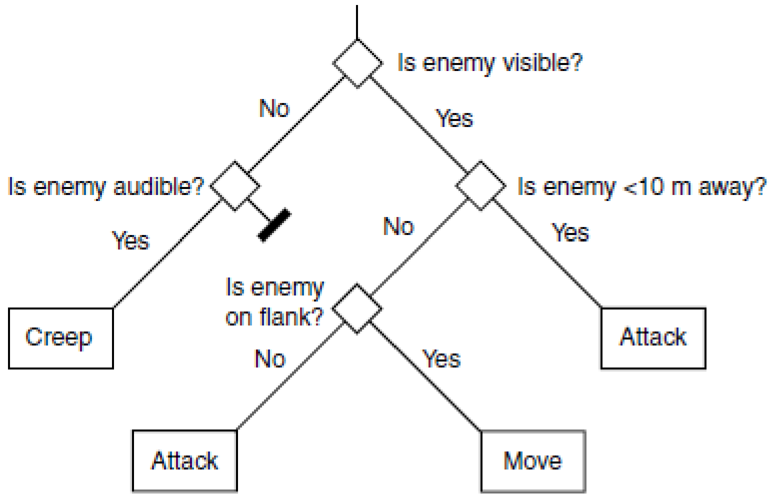
Decision Making



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

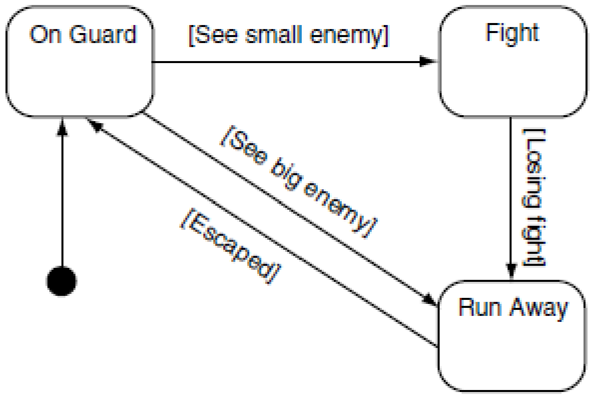
State machines

Generalization of the decision trees

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

Created by designers

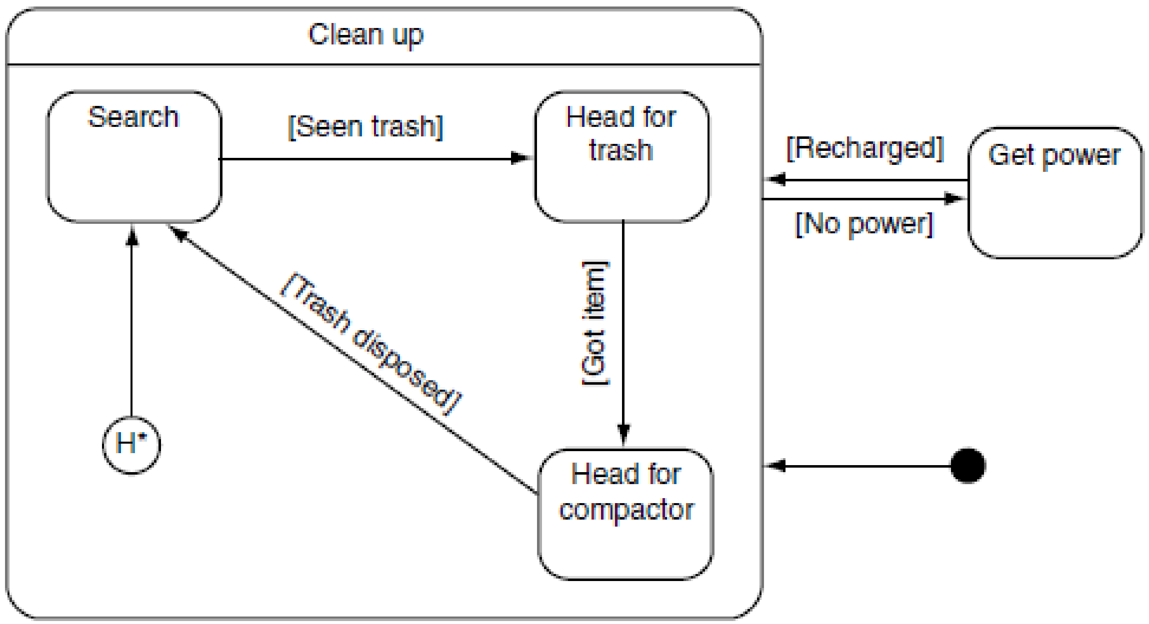
Hard-coded vs. flexible



Hierarchical state machines

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

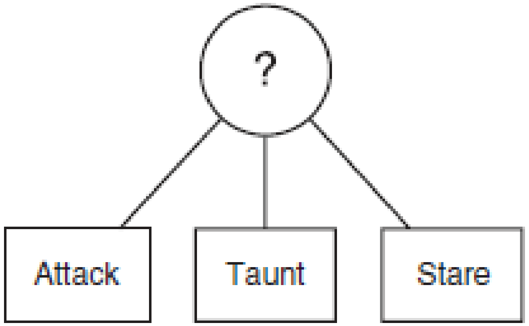
Combination of other techniques

Easy to understand and create for non-programmers

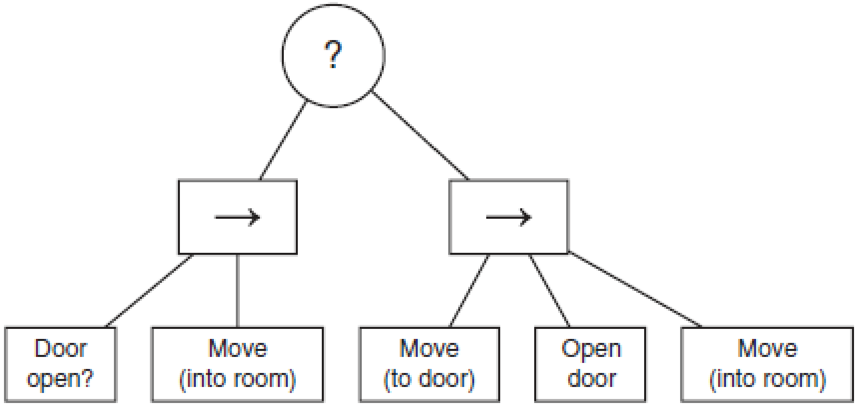
- First description in Halo 2 (2004)

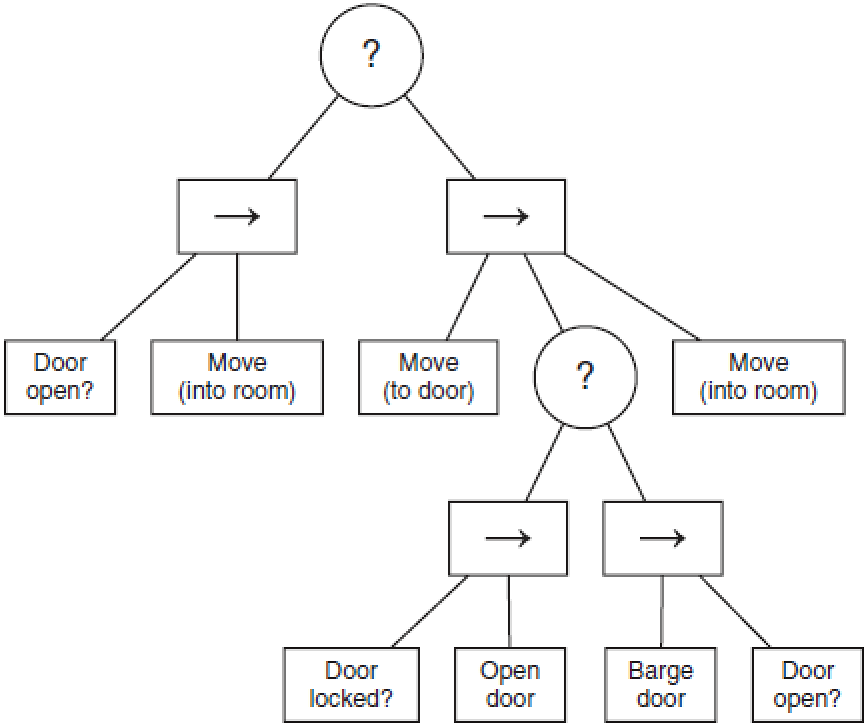
Tasks instead of states

- Conditions
- Actions
- Composites
  - Selector
  - Sequence

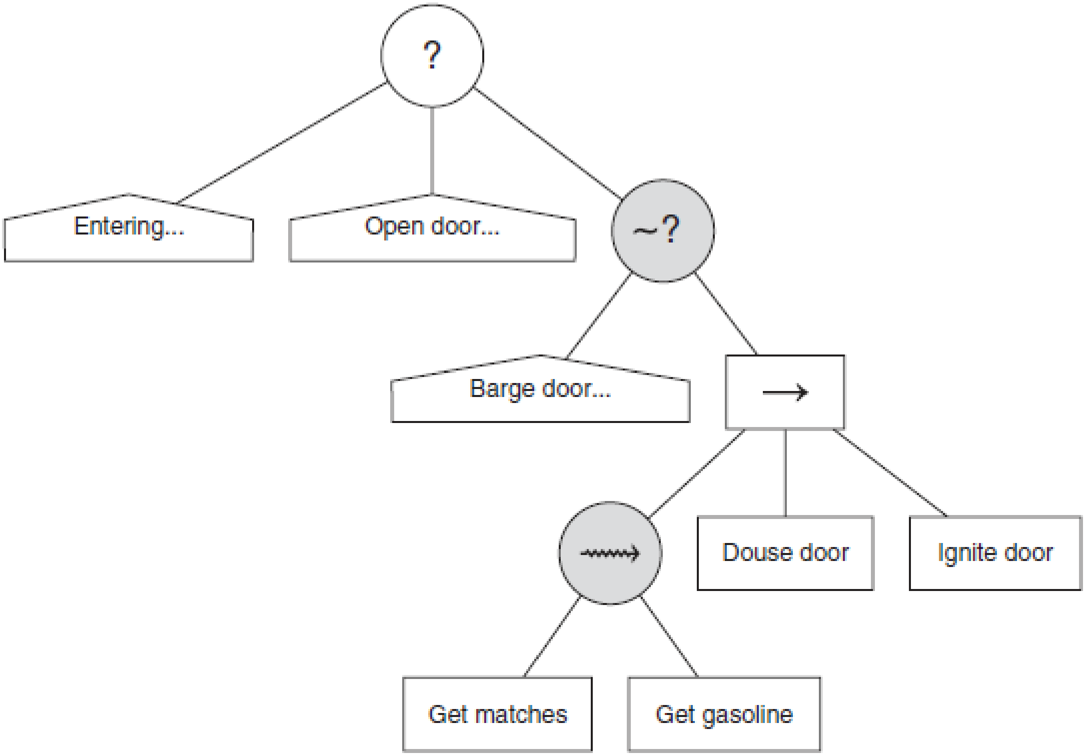


Behaviour trees





Non-determinism



Fuzzy logic

Game states can be ambiguous

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

Fuzzy logic

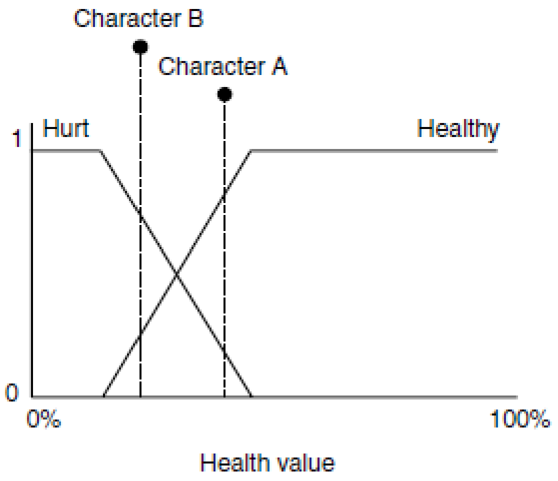
- Instead of being or not being in a set, fuzzy logic states that an object has a set degree of membership in the range [0,1]

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

$$\text{Brake} = \min(0.1, 0.4) = 0.1$$

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

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, …

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

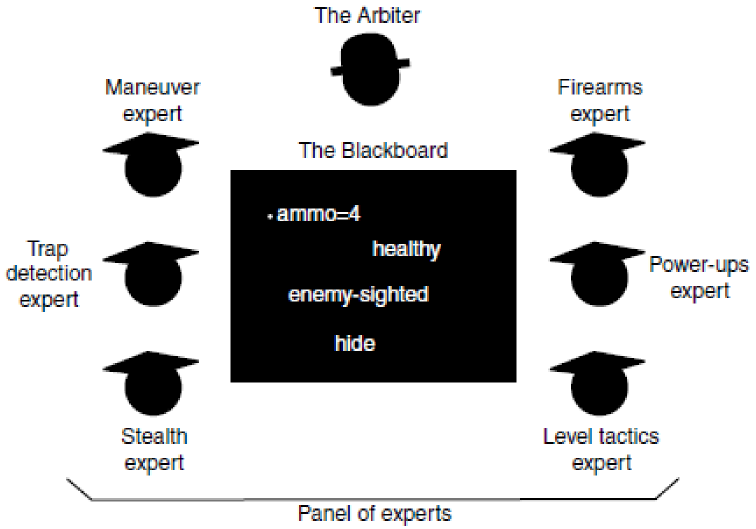
TECHNISCHE UNIVERSITÄT DARMSTADT

Every AI can write to and read from the blackboard

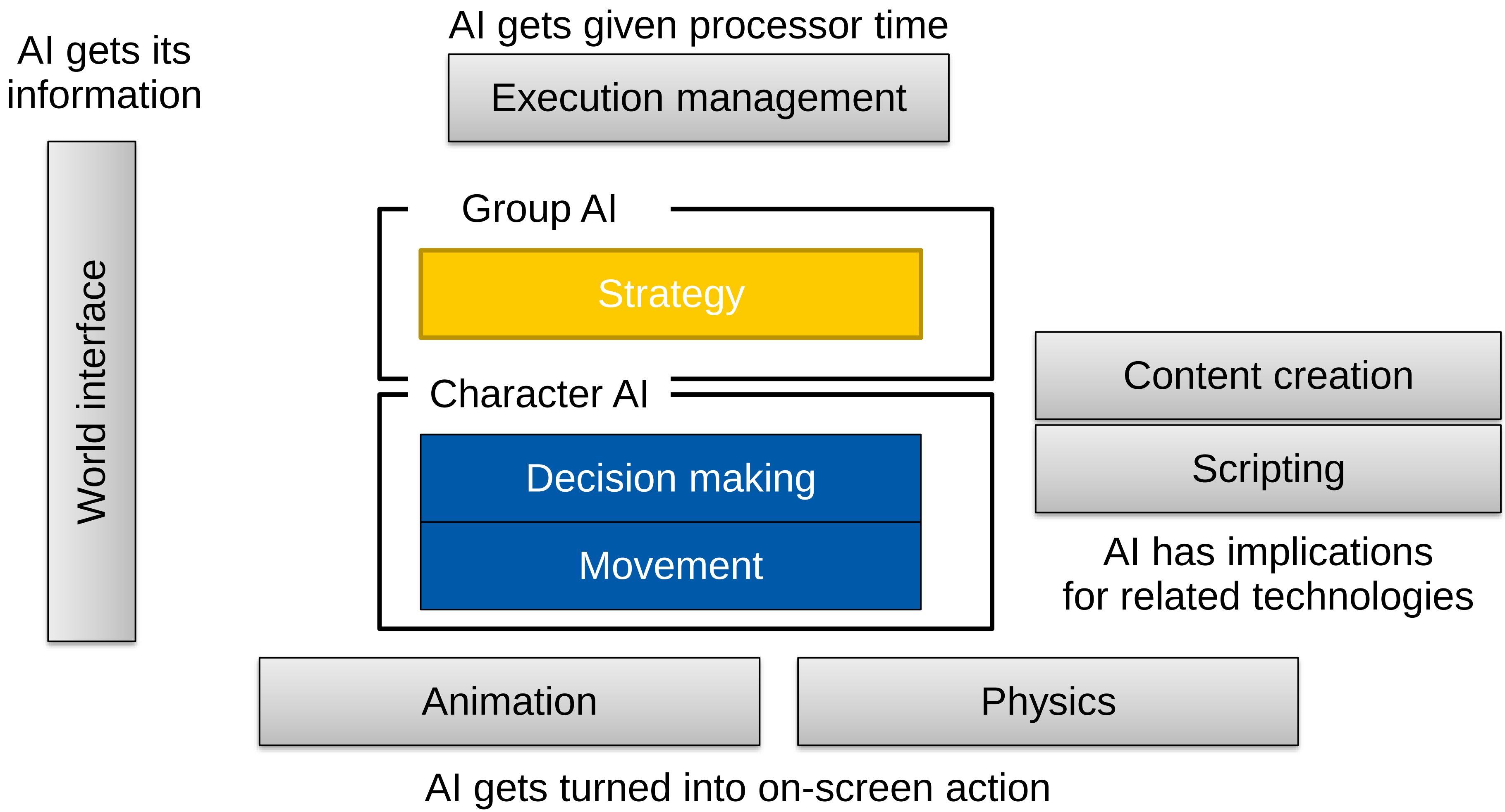
Efficient way of communicating with other AIs

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

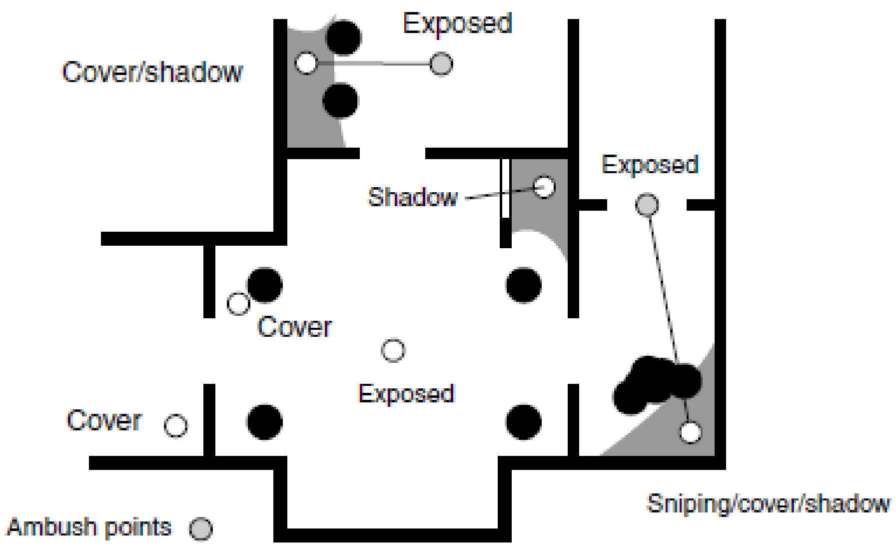


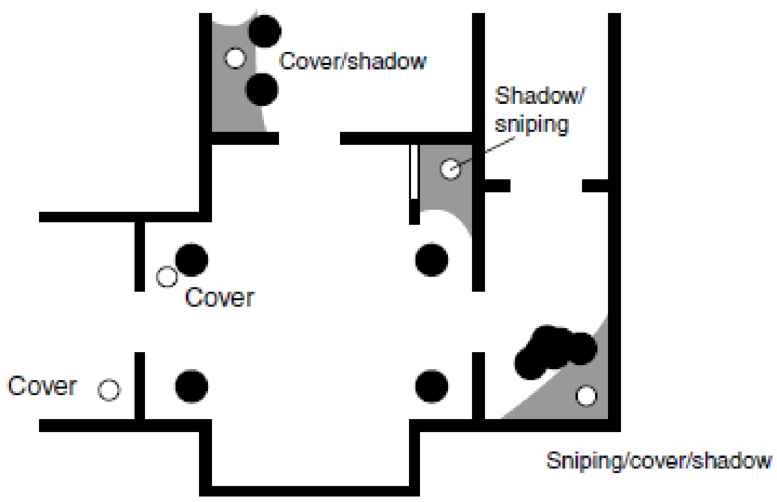
Tactical and Strategical AI

vs

Waypoint Tactics

Find points that are suited for different actions



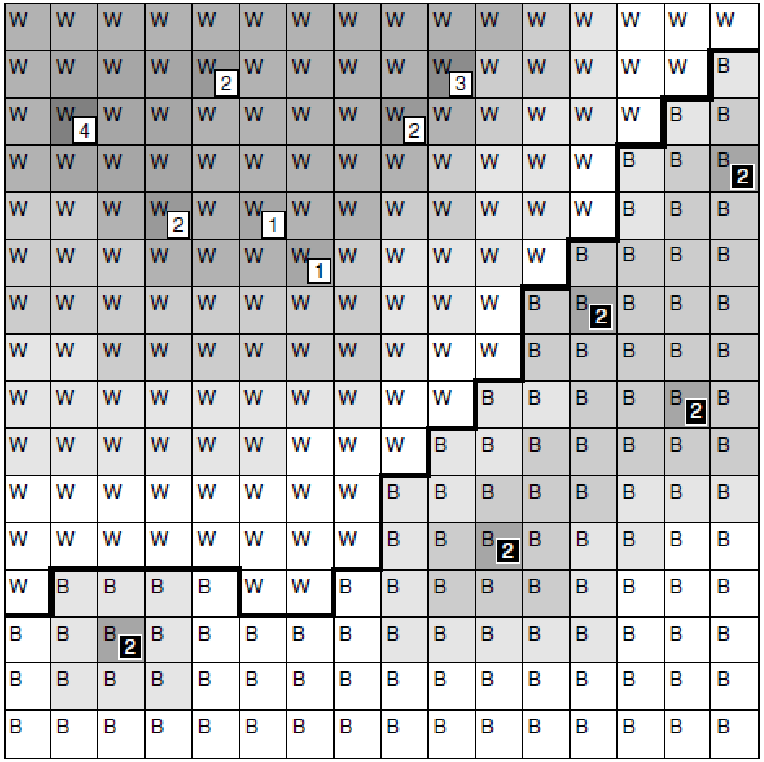


Tactical Analyses

Analyze the game world

Influence maps

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

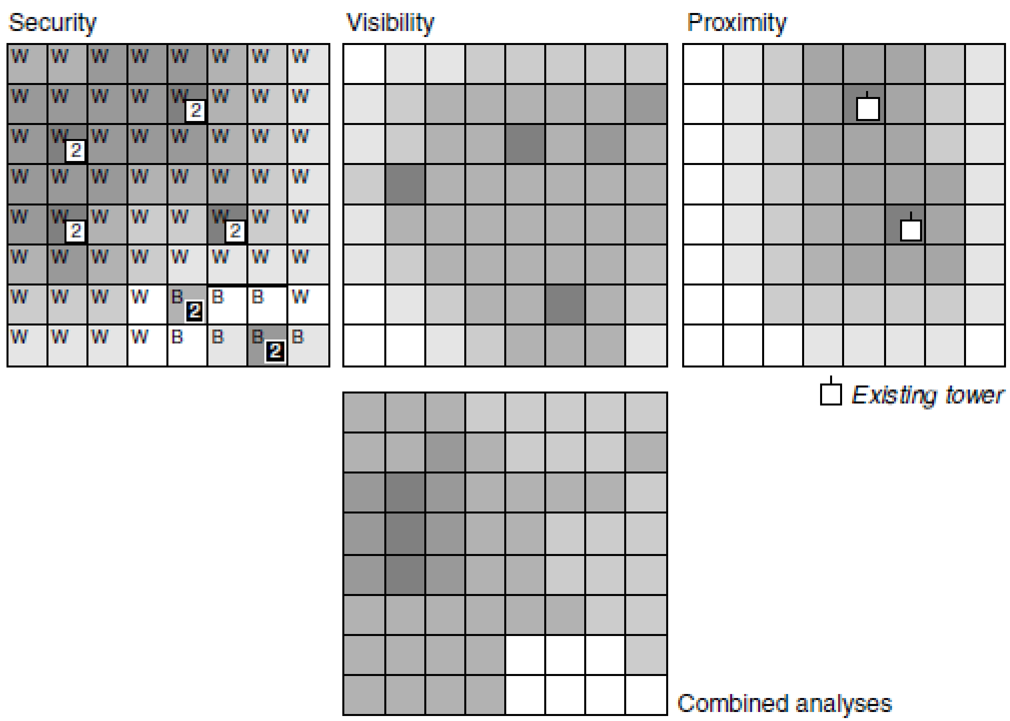


Learning influence maps

Frag Maps

Multiple Layer Analysis

Combine different tactical analyses into one



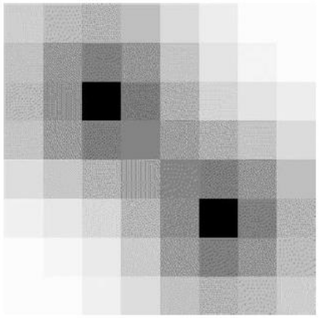
Generating influence maps

Map Flooding

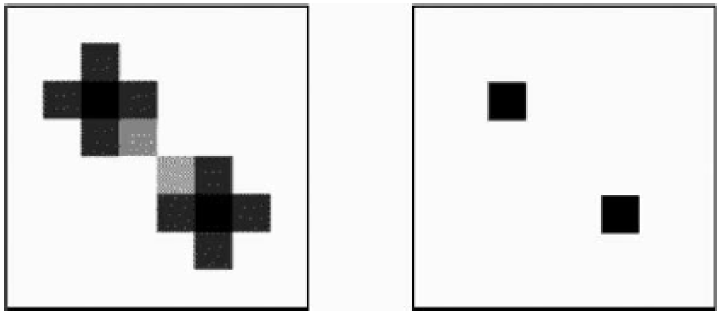
- Use Dijkstra Algorithm variant
- If influences overlap, the stronger influence wins
- Leads to Voronoi Diagram-like map

Filters

- Blur



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



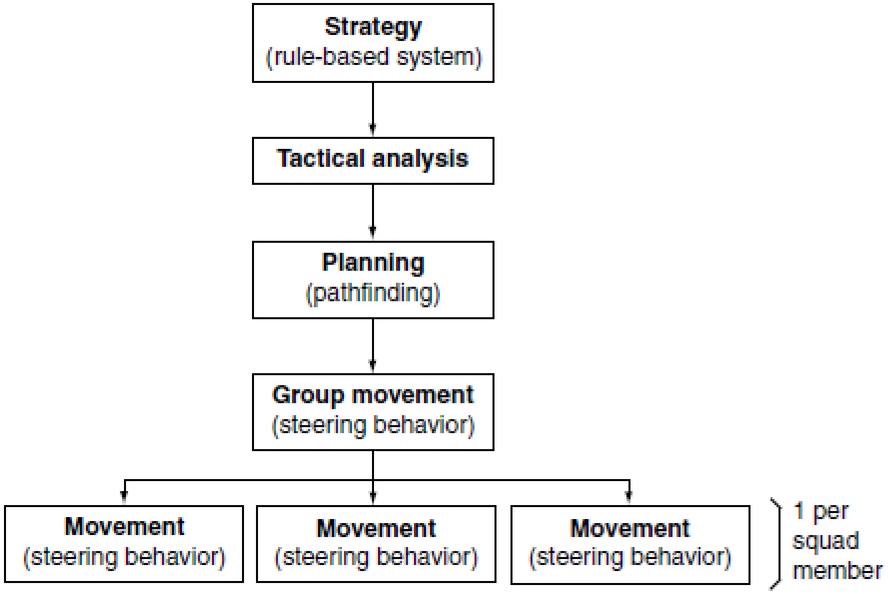
Coordinated Action

Bottom up

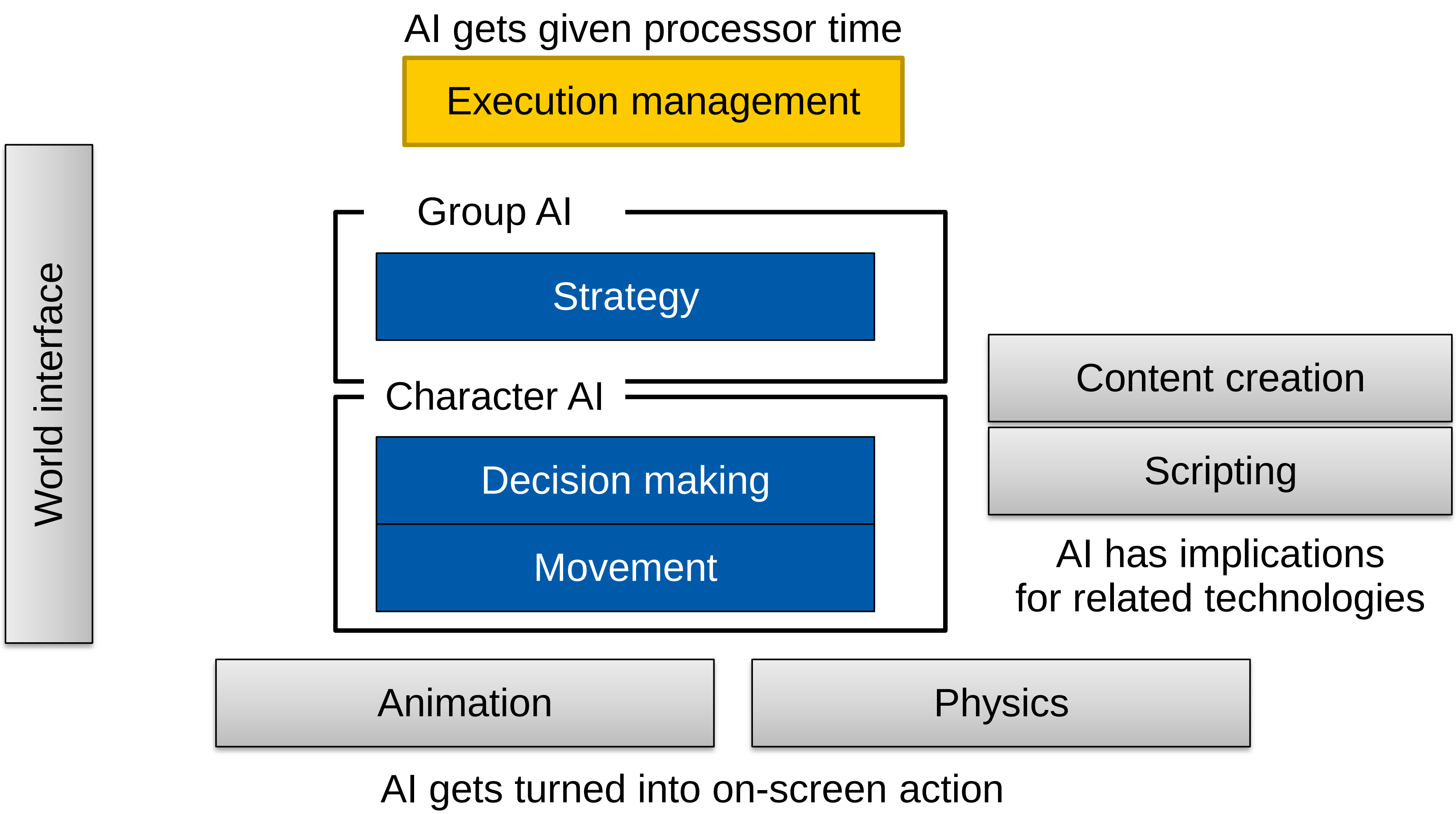
- AIs query higher-level systems for tactical data

Top-down

- High-level commands passed downwards



Execution management



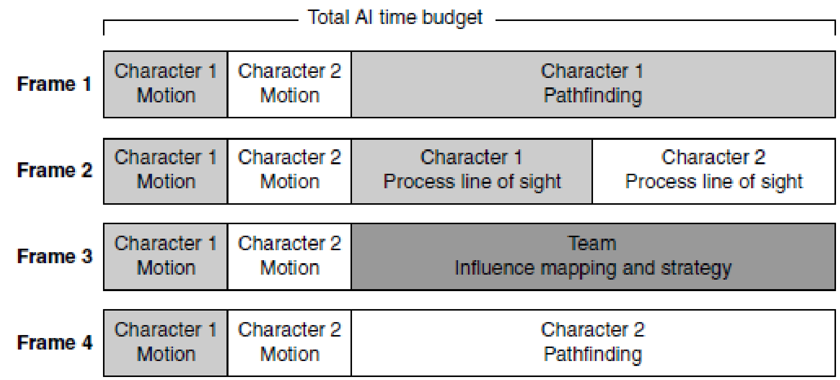
Scheduling

Frequency

- How often should the AI task be called?

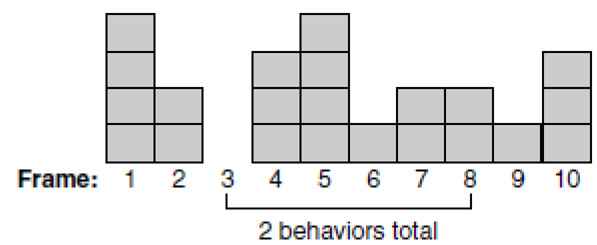
Phase

- 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

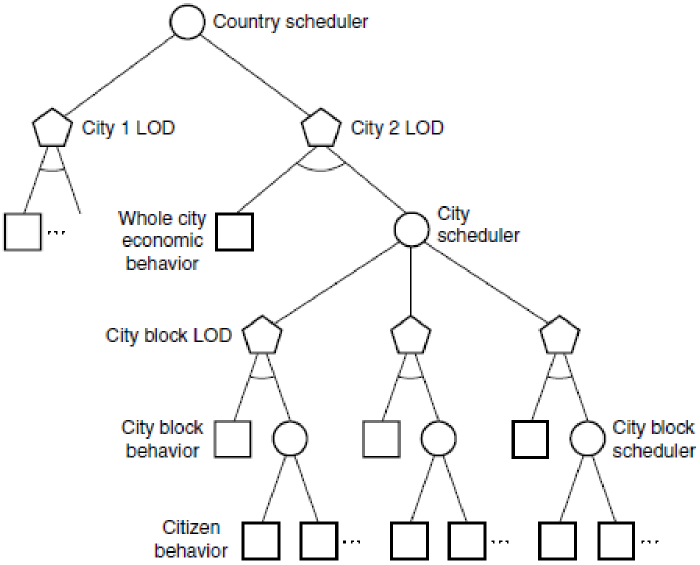
AI Level of Detail

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

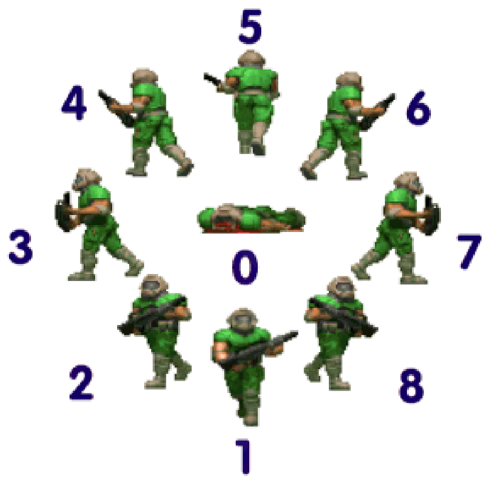


AI Recap: Doom (1993)

Source code available at
https://github.com/id-Software/DOOM

Basics

- Characters displayed as sprites with 8 directions



AI Recap: 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

AI Recap: 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
```

AI Recap: Doom (1993)

TECHNISCHE UNIVERSITÄT DARMSTADT

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

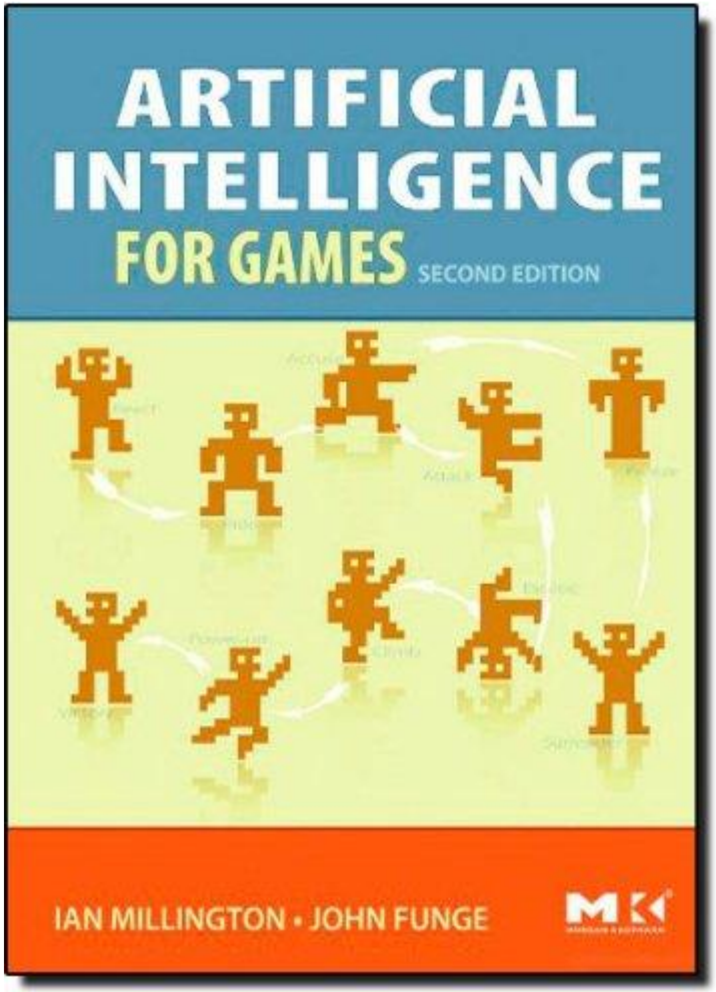
Literature

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