UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Spring 2020

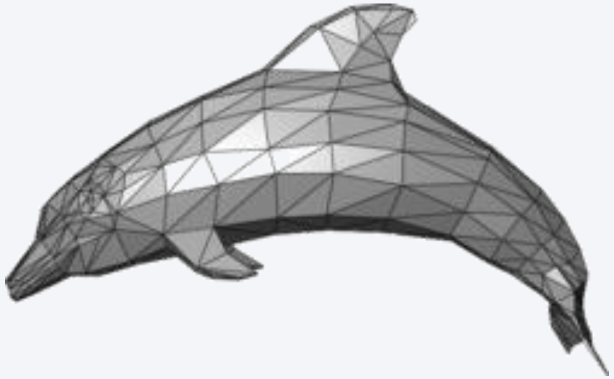
PROGRAMMING

SUMMARY

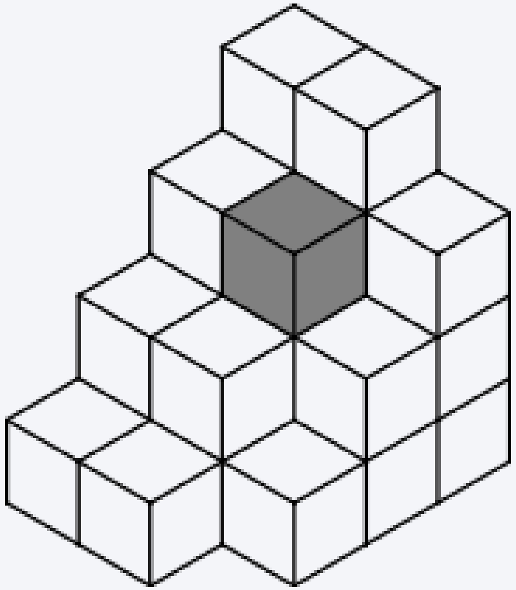
Maurizio Rigamonti

LEVEL 2

- **Polygonal**
  - Hardware!



- **Voxel (a.k.a. Boxel)**
  - Volumetric pixel
  - Regular grid in space
  - Used for medical and scientific data

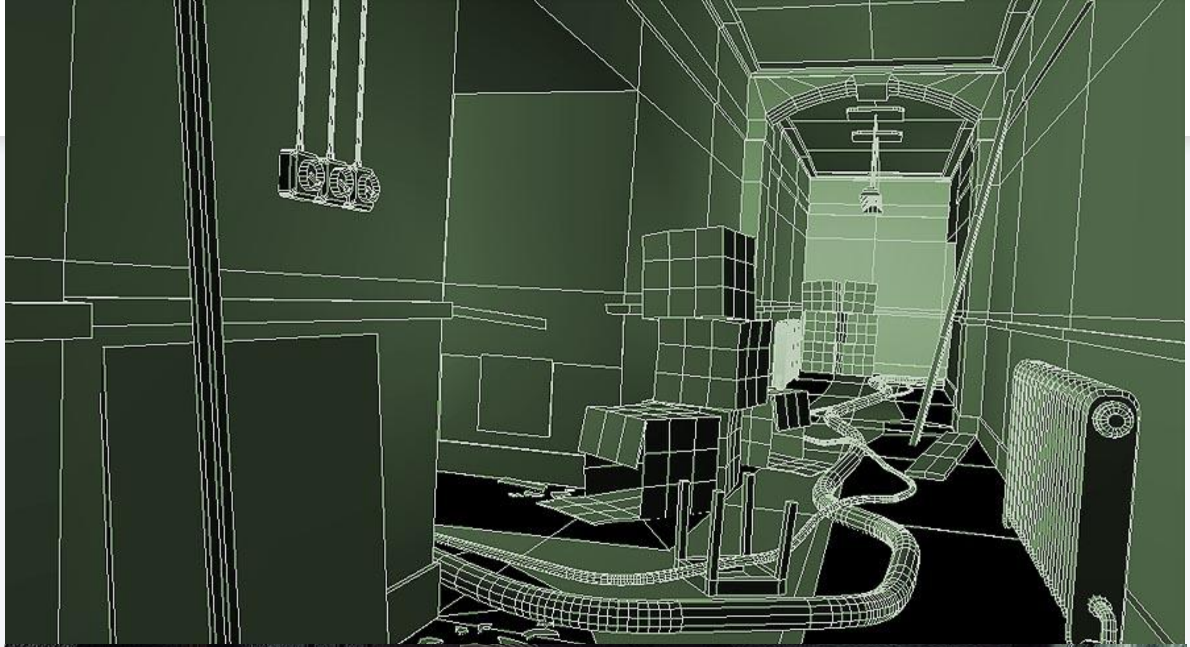


CREATING A SCENE

1. Geometry calculation
2. Texturing
3. Lighting
4. Shading

3D SCENE

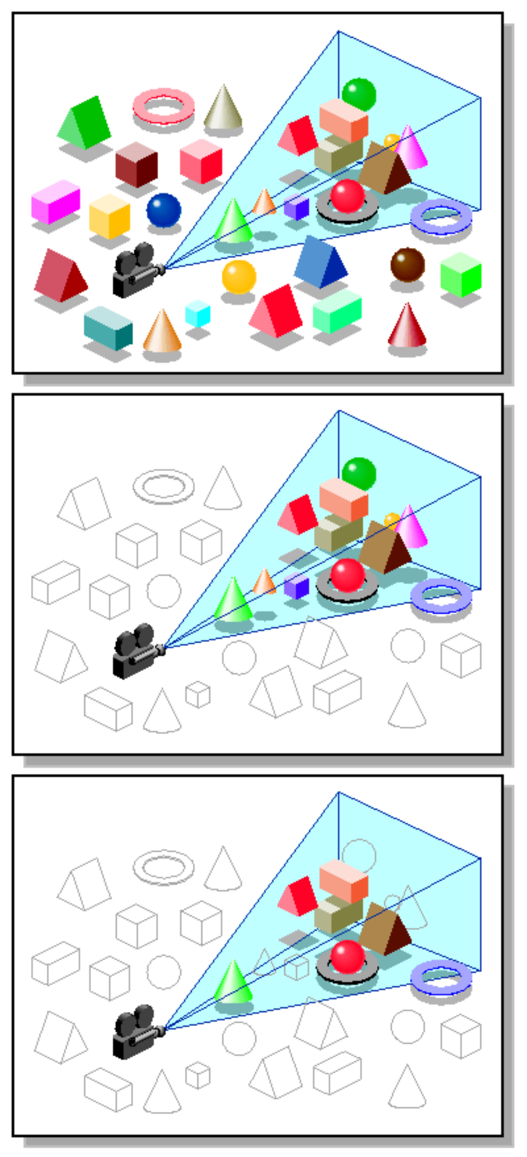
- Elements
  - Vertices, edges
  - Triangles
  - Meshes
  - Skeletons
  - Textures
  - Cameras
  - Lights
  - ….



GEOMETRY (AND SCENE) PROCESSING

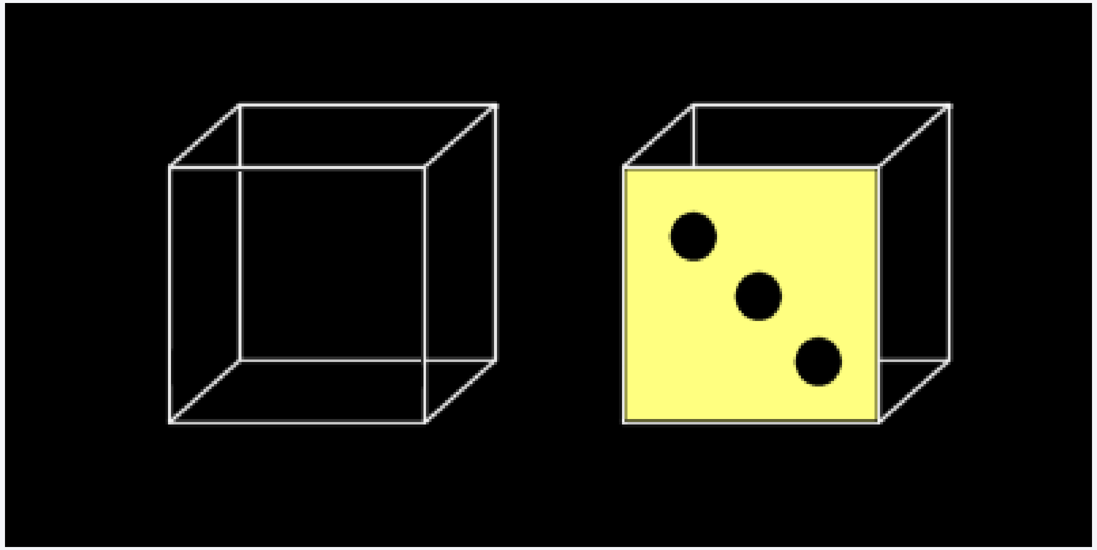
Reduction

1. Pipeline
2. Clipping
3. Culling
4. Occlusion testing
5. Resolution testing
6. Rasterization



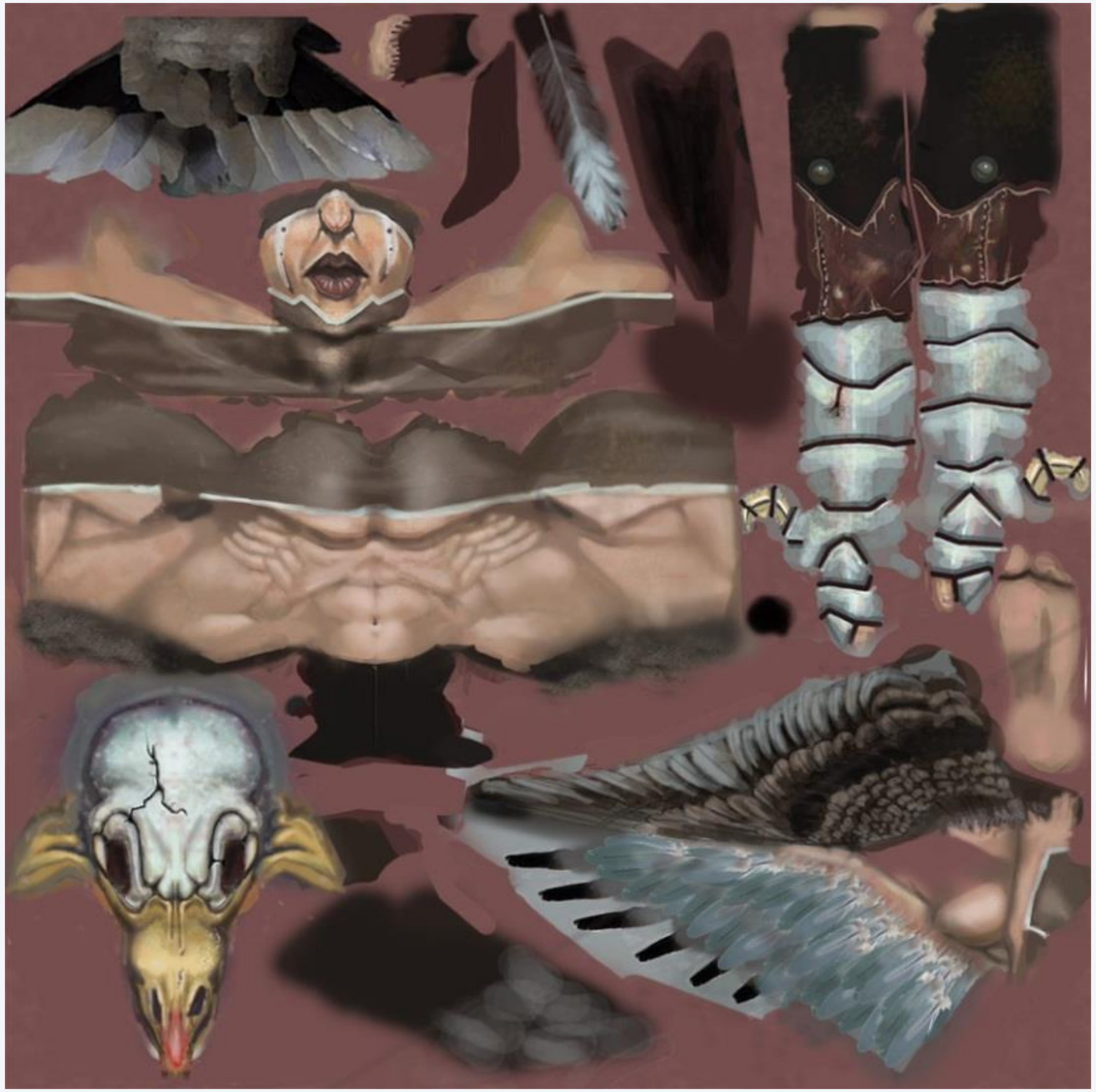
ADDING MATERIALS

- Texturing
  - Explicit VS procedural
  - Static VS dynamic



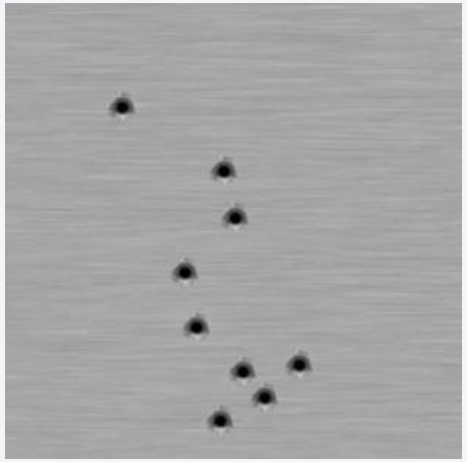
MAPPING

- Texture Mapping
  - XYZ
  - Volumes
  - Triangles
  - Tiling

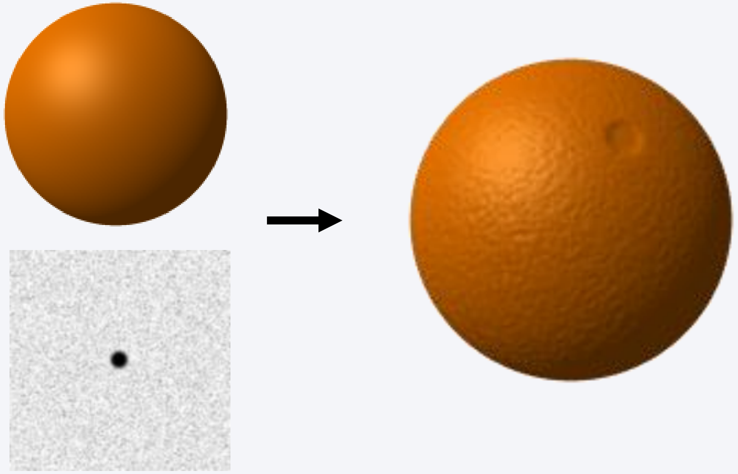


MULTIPLE TEXTURES

- Multipass techniques

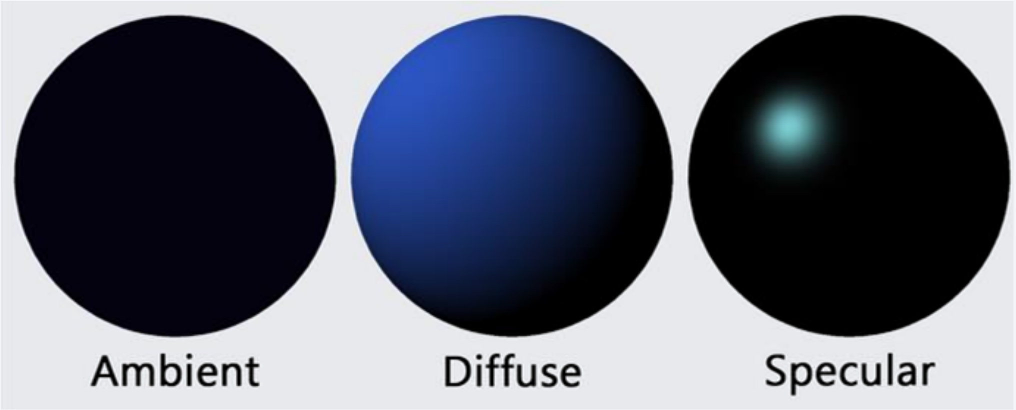


- Multitexturing
  - Environment + Gloss Mapping
  - Bump Mapping

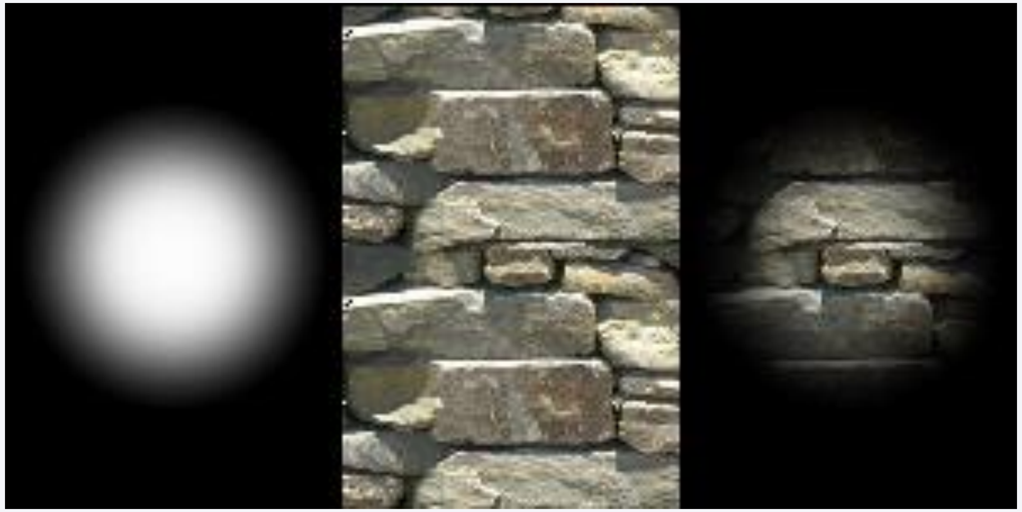


LIGHTNING

- Light
  - Ambient
  - Diffuse
  - Specular



- Light mapping



PIXEL SHADERS

- Pixel operations and functions
- Programmable
- Manipulation of
  - Light absorption and diffusion
  - Texturing
  - Reflection and refraction
  - Shadows
  - Primitive displacements
  - Post processing


SHADER 2.0
SHADER 3.0
WWW.FARCRY.NL
WWW.FARCRY.NL
SHADER 2.0
SHADER 3.0
WWW.FARCRY.NL
WWW.FARCRY.NL

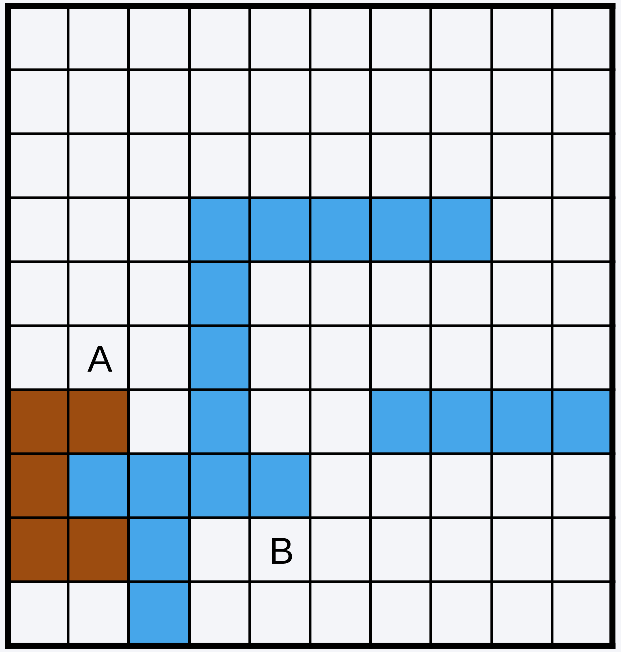
LEVEL 3

- Major component of each successful game
- More than 50 years of history
- Game AI: specific application of classic AI methods

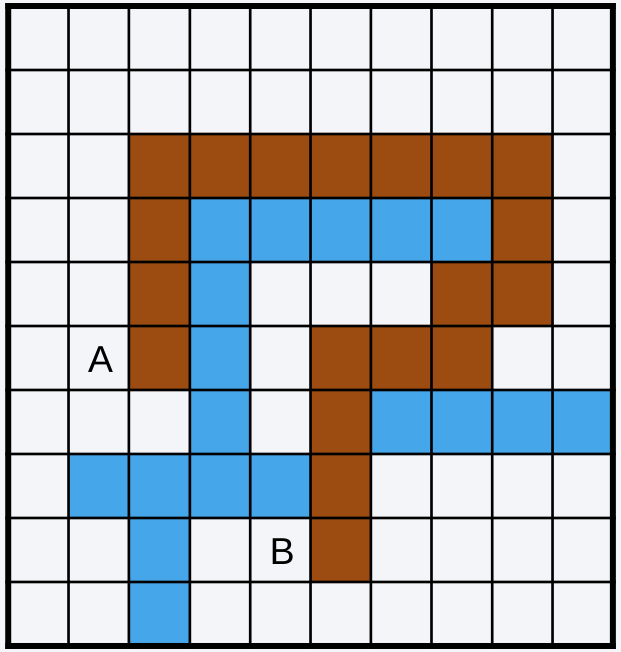
CONTEXT

- Difference between good and bad AI?
  - Bad AI: wrong goals
- AI: computer simulation of intelligent behavior
- What intelligence is?
  - (Pretty) optimal behavior?
  - Human-like behavior?
- Optimal behavior are often unrealistic

PATH FINDING



Human being



A*

STRUCTURE OF AN AI SYSTEM

- **AI entity** (agent)
  - Enemies, armies, NPC, animals, etc.
  - 4 main elements
    - A sensor (or input system)
    - A working memory
    - A reasoning core
    - An action (or output) system
- **Abstract controllers**
  - Strategies, tactics, etc.
  - Routines for group dynamics
  - Structure similar to the one of entities

Sensors depend on game type

- In general, computationally expensive

Quake enemy

- Player’s position and direction
- Map geometry
- Enemy and players weapons
- A visual system (range)

Age of Empires

- Balance of power in each map subarea
- Resources
- Breakdown of unities
- Technological status
- World geometry

MEMORY

- Individual AI
  - Points and orientations
  - Numerical values
  - States (e.g. walking, jumping, etc.)
- Abstract controllers
  - Much more complex data structures

REASONING CORE

- Subsystem **analyzing** sensors and memory to take decisions
- The speed of the decisional process depends on sensors and alternatives
  - A lot of great games use simple methods

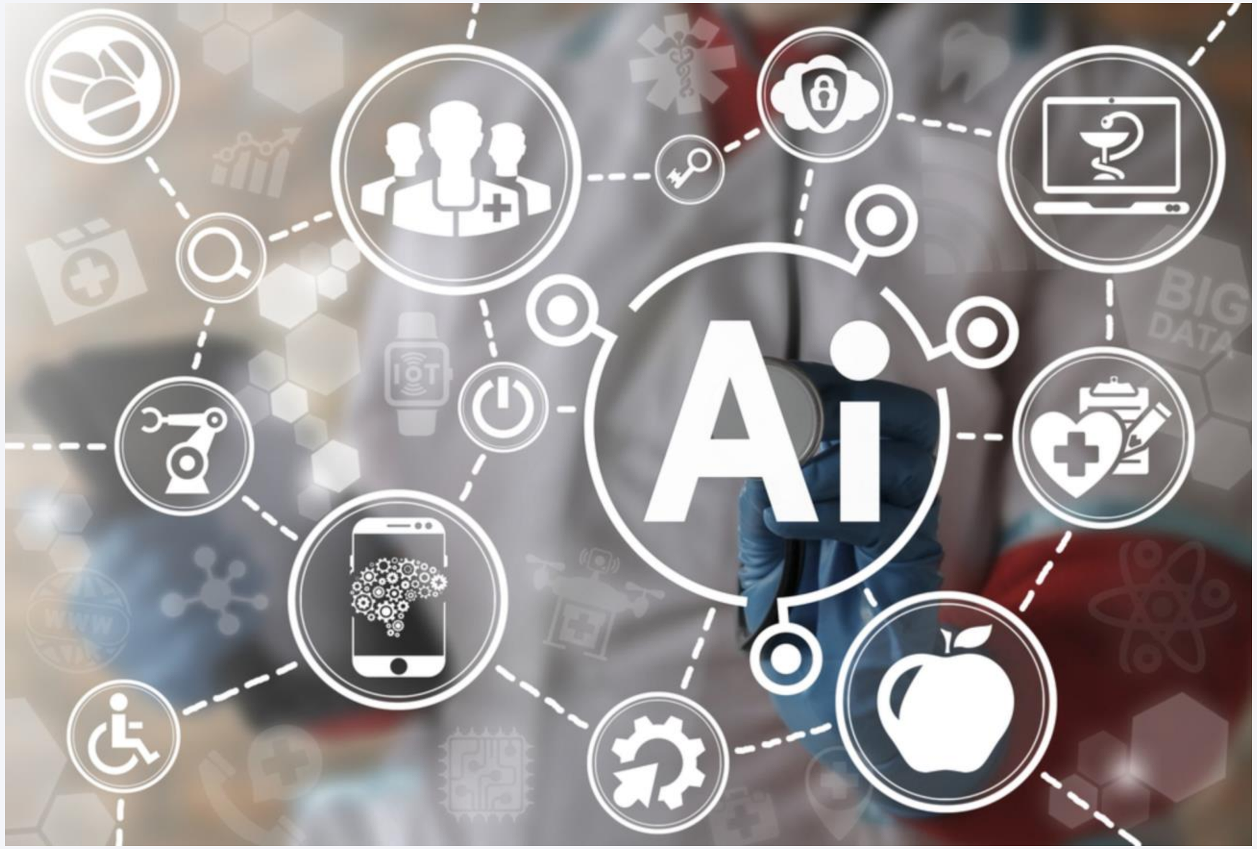
ACTION SYSTEM

- AI is coupled with **action** subroutines
- Many games exaggerate this systems
  - E.g. in Super Mario Bros, enemies have a very similar AI but different actions
  - Perceived intelligence is enhanced

- In games, 4 common methods are mainly used to simulate AI
  - State machines
  - Rule systems
  - State-space searches
  - Biology-inspired methods

LEVEL 3-1

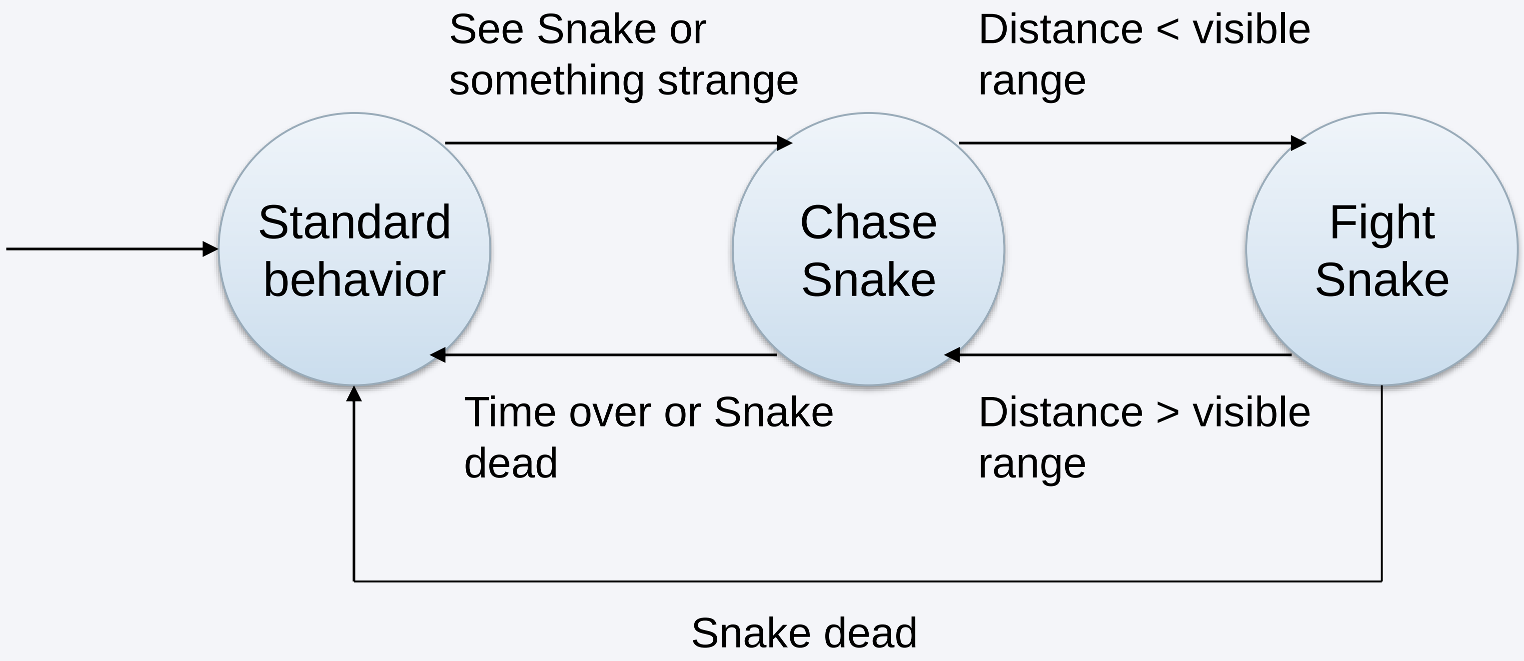
Finite state machines (FSM)



- Formalism involving 2 sets
  - A set of **states** >> AI configurations
  - A set of **transitions** >> conditions
- Most popular technique in games
- Draw the graphical representation of the FSM!
  - Difficult to change/update the AI otherwise

A BASIC EXAMPLE

- Soldiers in Metal Gear Solid?



PARALLEL AUTOMATA

- In case of complex behaviors, the FSM grows quickly
  - Cluttered
  - Difficult to manage

- Define different parallel subsystems
  - Entity with more “brains”
  - E.g. motion, fighting, collision detection, etc.

- Parallel automata works fine where behaviors are **independent** and **simultaneous**

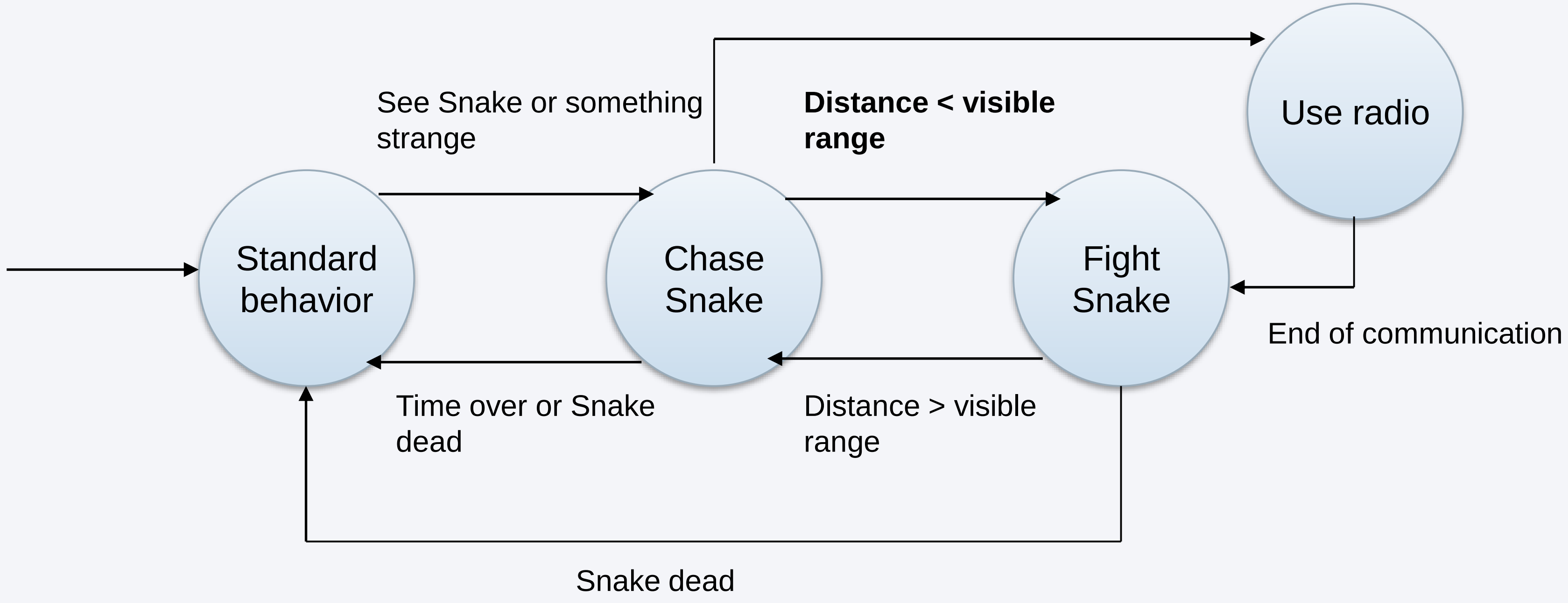
SYNCHRONIZED FSM

- Inter-automata communication
  - Enemies attacking in squad: e.g. one fixing, one advancing, and one calling for help
- Entities share a **common memory area**
- Non optimal method for strategic games with a lot of entities

NON DETERMINISTIC FINITE AUTOMATA

- FSM limitation: they are predictable
  - **Dominant strategies**!

- Add variation using non deterministic automata
  - Controlled randomness



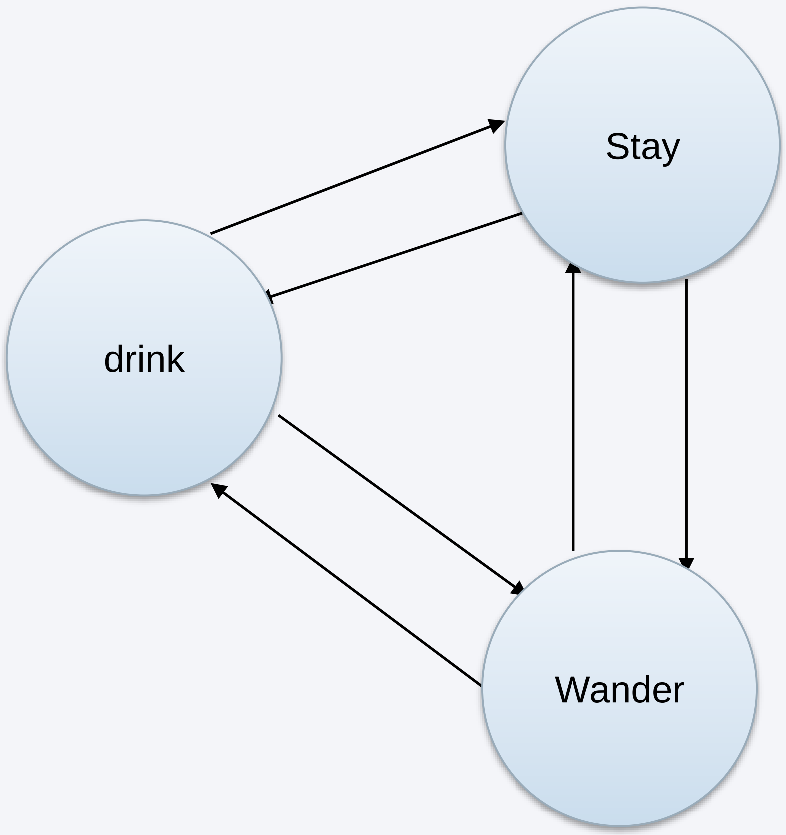
Rule Systems (RS)

LIMITS OF FSM (1/2)

- Some phenomena difficult to describe in terms of states and transitions
  - I’m injured and there is a healing potion nearby, I will drink it
  - I’m injured, but there isn’t any healing potions around, so I will wander
  - I’m not injured and there is a potion nearby, so I will stay

LIMITS OF FSM (2/2)

- Each statement implies a state of the FSM
- Each state has a possible transition to any of the others!
  - FSM are optimal for local and sequential behaviors in nature



RULES SYSTEMS

- **Prioritized** and global behaviors can be described with rule systems
- RS are set of rules of the form

Condition $\longrightarrow$ **Action**

Left-Hand Side *Right-Hand Side*

- Applying a RS to the previous example

(Injured) && (Potion nearby) $\longrightarrow$ Drink
(Injured) && (No potion) $\longrightarrow$ Wander
(Not injured) && (Potion nearby) $\longrightarrow$ Stay

RS EXECUTION

1. Test the LHS of each condition in order
2. Execute the RHS of the **first** rule that is activated

- RSs imply a sense of **priority**
  - Top rules have the precedence over bottom rules
- Global model behavior
  - At each execution time, all rules are tested
  - No sequences (FSM)

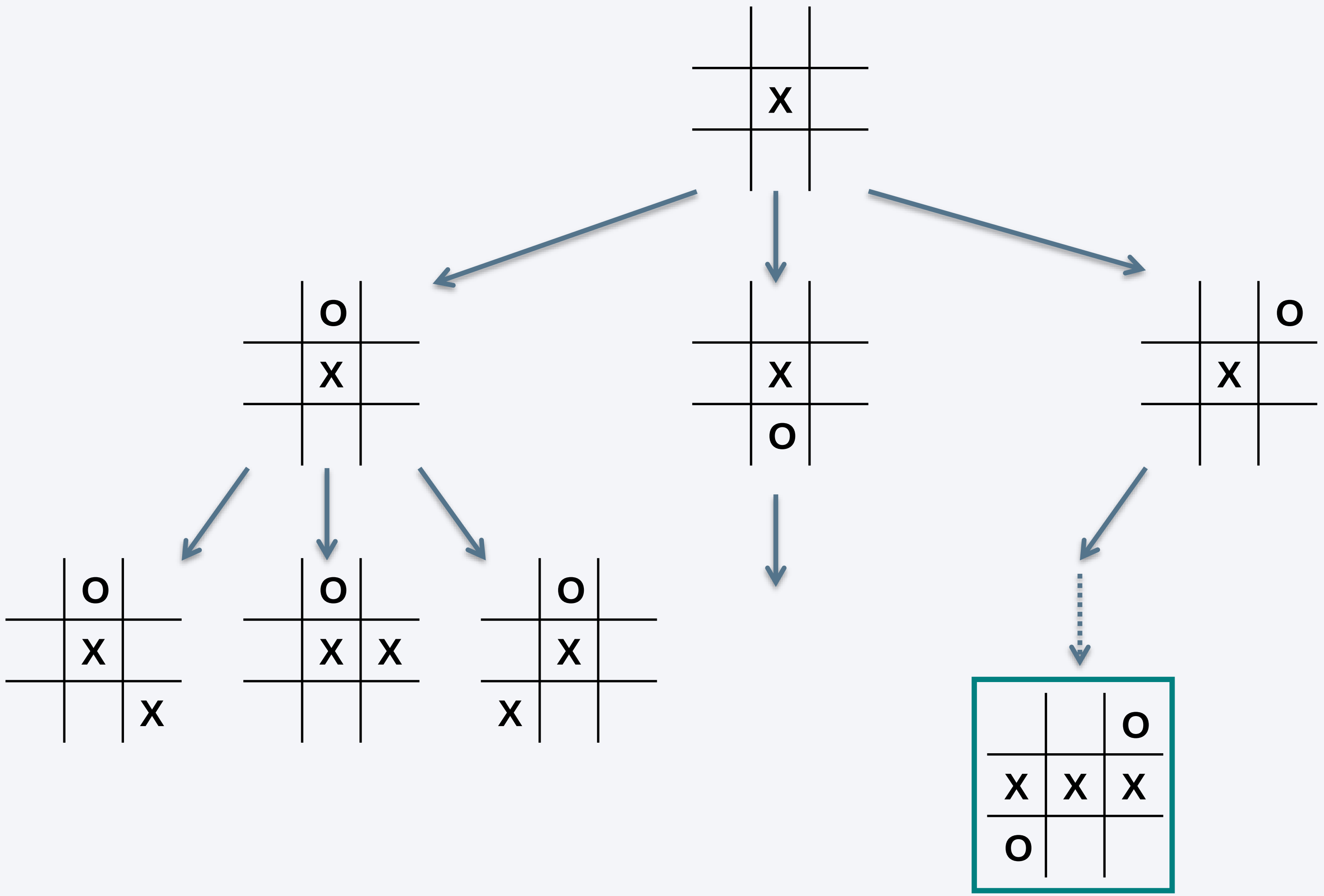
Planning and Problem Solving

- FSM and RS ideal for simple behaviors
  - Sequences and phases (FSM)
  - Priority concurrent tasks (RS)
  - Action AI
- More complex behaviors?
  - Puzzles
  - Path finding
  - Games like chess
  - Military strategies
- The AI has to “think” and “plan”

STATE-SPACE SEARCH

- Explore candidate transitions and analyze their suitability for reaching a goal
  - Complement FSM and RS
- SSS paradigm
  - Propagate each possible move into future
  - Evaluate its consequences
- The game is represented as a tree
  - Nodes: configurations
  - Branches: moves

TIC-TAC-TOE EXAMPLE



INTEGRATION OF STATE-SPACE SEARCH

- Many existing variants
  - **Brute force** (e.g. Depth-first search)
    - All possibilities
  - **Heuristic based** (e.g. A*)
    - Look for a promising candidate
- Even heuristic based methods are not able to solve all problems
  - Chess!

LEVEL 4

ACTION-ORIENTED AI

- **Action**
  - Intelligent activity with rapid behavior changes
  - Locomotion, attack, defense, etc.
- Contraposition to tactical reasoning
  - **Immediate** VS planned activity
- Simple tests
- High rhythm

LEVEL 4-1

Choreographed AI

- **Preprogrammed** action sequence
- Simple behaviors
  - Security guards walking
  - Shoot'em up ships
  - Objects (e.g. elevators)
- State machines without optional transitions
- Often simple scripting engine

THE WALKING SECURITY GUARD

- A basic script example (b: start position)

```
Walk -10
Rotate 180
Walk 10
Rotate 180
```

LEVEL 4-2

Object Tracking

OBJECT TRACKING

- **Maintaining eye contact**
  - One of the first problems in action games
- Static or moving target
  - E.g. rotating turret
- Useful for other techniques
  - Chasing
  - Evading
  - Navigation (e.g. to reach a waypoint)
  - Etc.

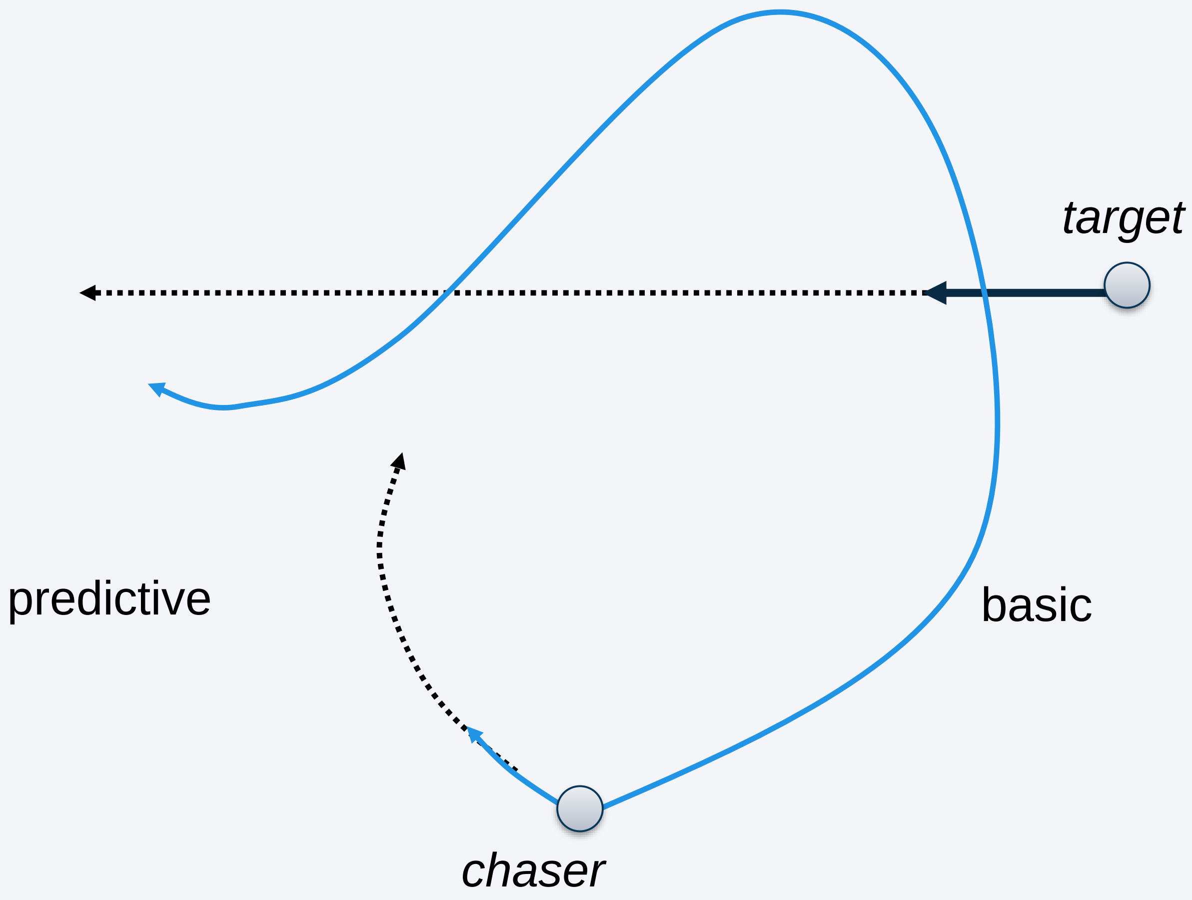
Chasing / Evasion

- Based on eye contact
  1. Advance while keep eye contact
  2. In case of lose of the target, correct orientation and keep moving

- Success of the technique depends
  - Hunter’s speed
  - Target’s speed
  - Turning ability

PREDICTIVE CHASING (1/2)

- Improve basic chasing
- Anticipate target movements
  - Keep history of its positions



PREDICTIVE CHASING (2/2)

- Approach
  1. Calculate a projected position
  2. Aim at that position
  3. Advance
- Various techniques to calculate the projected position
  - E.g. interpolation of the last *n* positions

EVADING

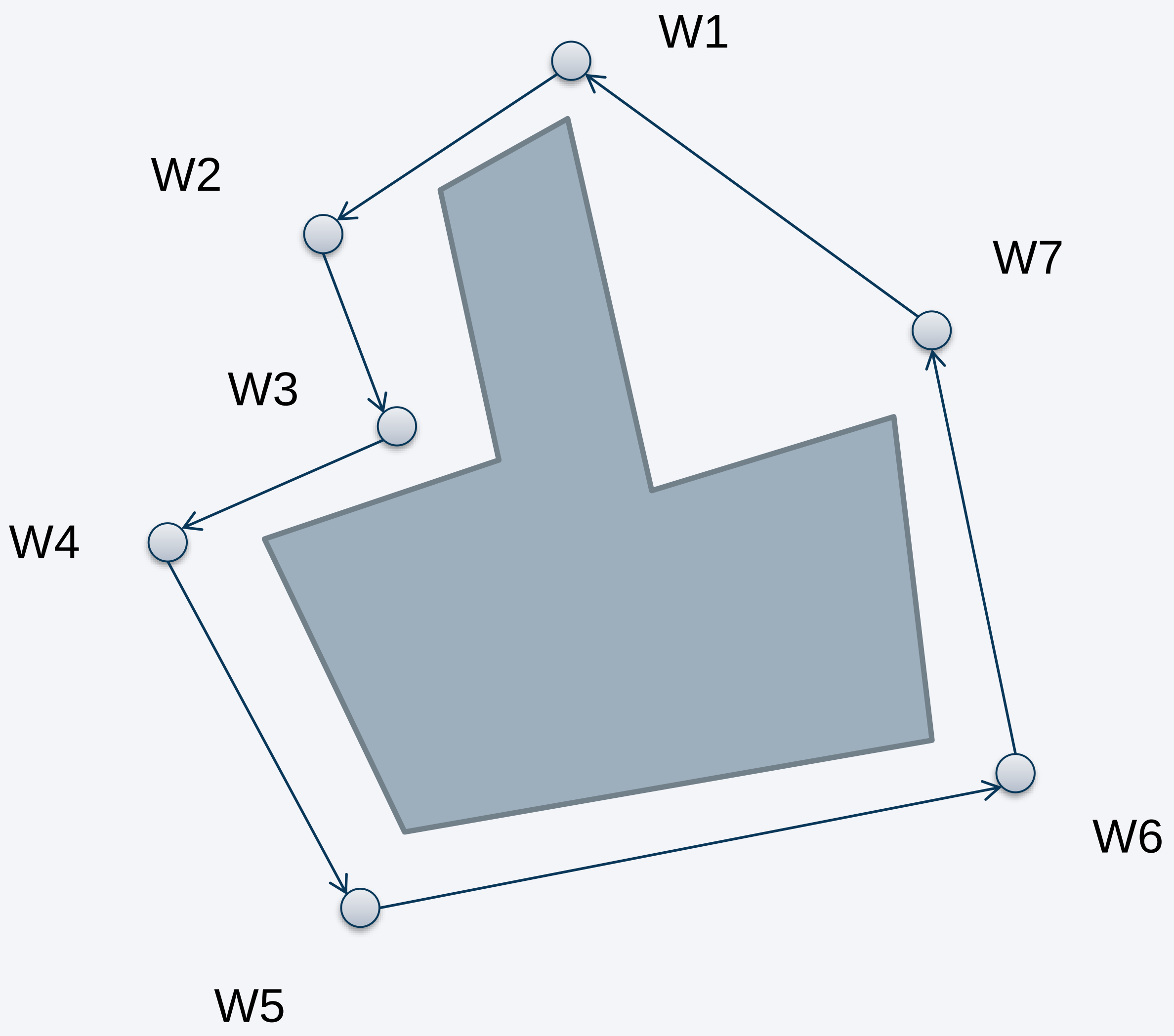
- Opposite of chasing
- Maximize the distance to the target!

Patrolling

PATROLLING

- Non playing characters
  - Policemen, dogs, etc.
- Define a set of waypoints
  - Cyclic (W1 W2 W3 W 4 W1 W2 W3 W4)
  - Ping-Pong (W1 W2 W3 W4 W3 W2 W1)
- Following a waypoint -> Chasing a sequence of targets

WAYPOINTS



Hiding and taking cover

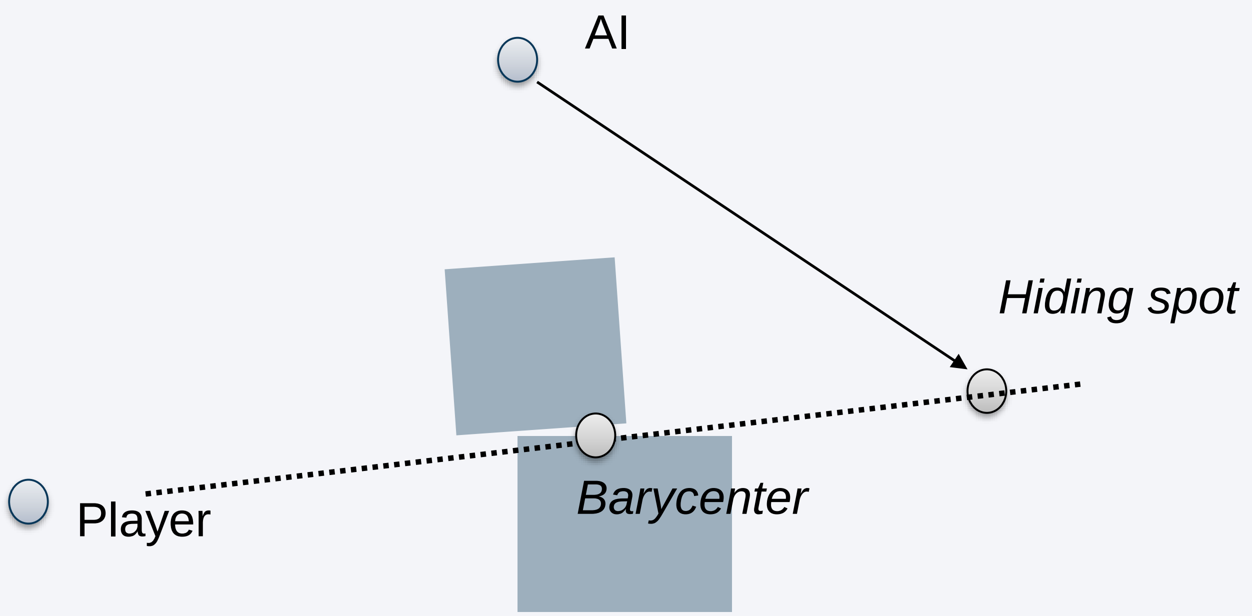
TAKING COVER

- Usually two actions
  1. Identify a hiding spot
  2. Move quickly (e.g. chasing technique)

- Detecting spots requires 3 data items
  - The position and the orientation of the player
  - The position and orientation of the AI
  - The geometry of the level (e.g. a map and a list of objects)

DETECTING A SPOT

1. Select the closest object
2. Shoot one ray from the player’s position to the barycenter of the object
3. Propagate the ray and choose a point



LEVEL 5

- Action
  - Simple methods
  - Illusion of intelligent behaviors
  - Sequential tests

- Tactical AI
  - Analysis
  - Making “right” decisions in complex scenarios
  - Paths, combat strategies, general solutions

TACTICAL THINKING

- Tactics: sequence of operations designed to achieve a goal
  - An **initial** state
  - A **goal** state
  - A **plan** to move from a state to the other
- Human brain VS machines
  - Cognitive processes VS numerical computation
- Difficult to answer questions like “who is winning the battle?”
- Tactical algorithms can have side effects
  - **Too good** to be realistic -> **frustrating** for the player

LEVEL 13-1

Path finding

PATH FINDING

- Last week: chasing
  - **No obstacles!**

- Problem: finding a path between a start and an end point

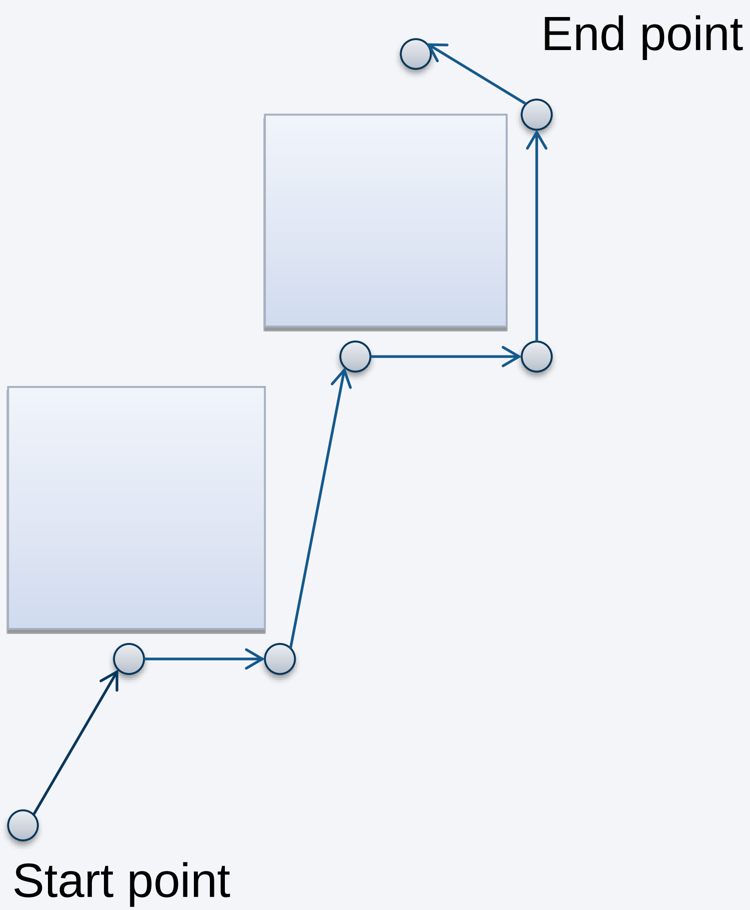
- Sequence of transitions (movements) between them

- 2 categories of path finding algorithms
  - **Local**
    - Surrounding of the current position
    - Calculated step-by-step
  - **Global**
    - Analyze the whole area
    - Precalculate the solution in one step and execute

CRASH AND TURN

- Animal behavior
  - Local method
- Idea
  - Move on a straight line
  - If an obstacle is reached, turn left or right
  - Try to go around the obstacle
  - When the line of sight is open, continue on a straight line

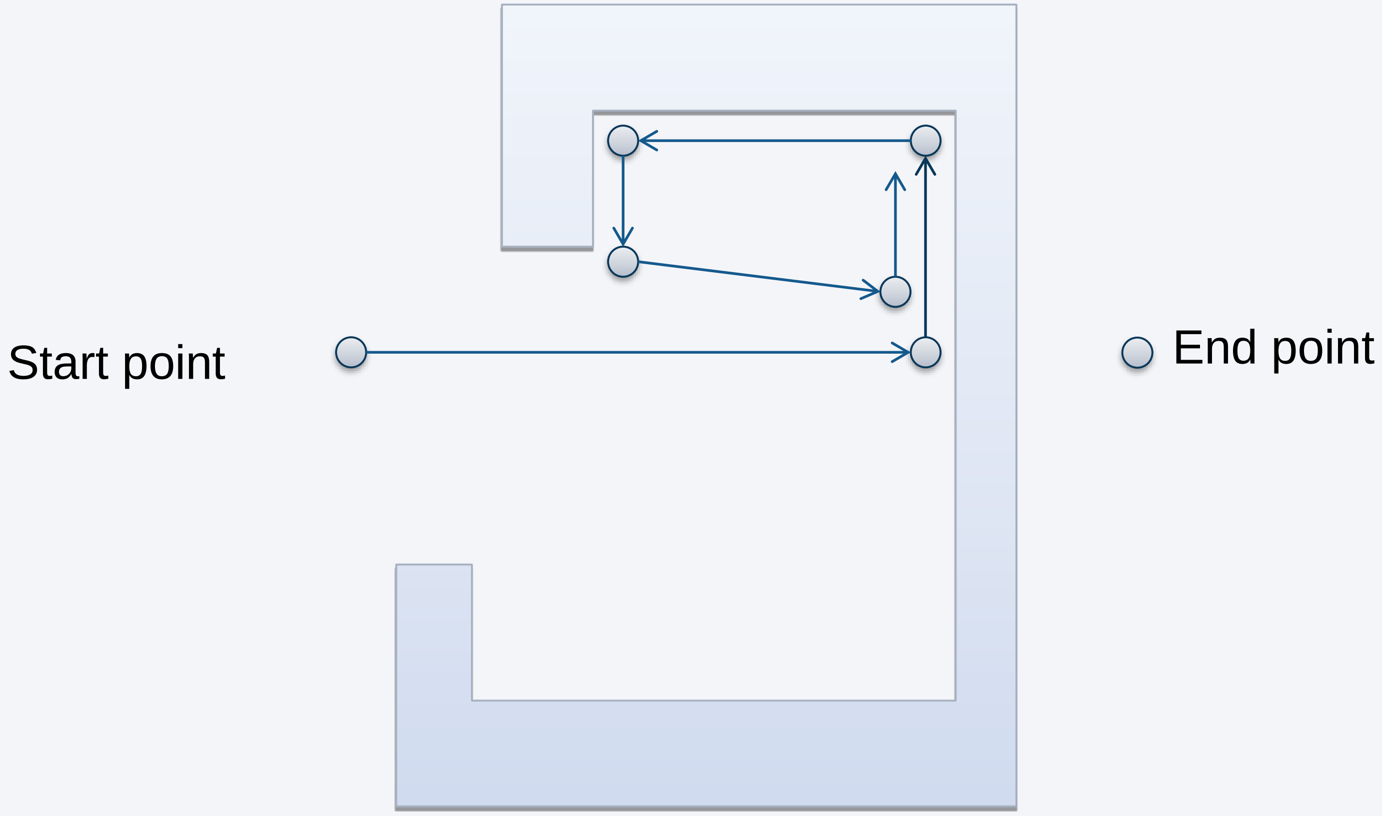
CRASH AND TURN: THE IDEA



CRASH AND TURN: PARTICULARITIES

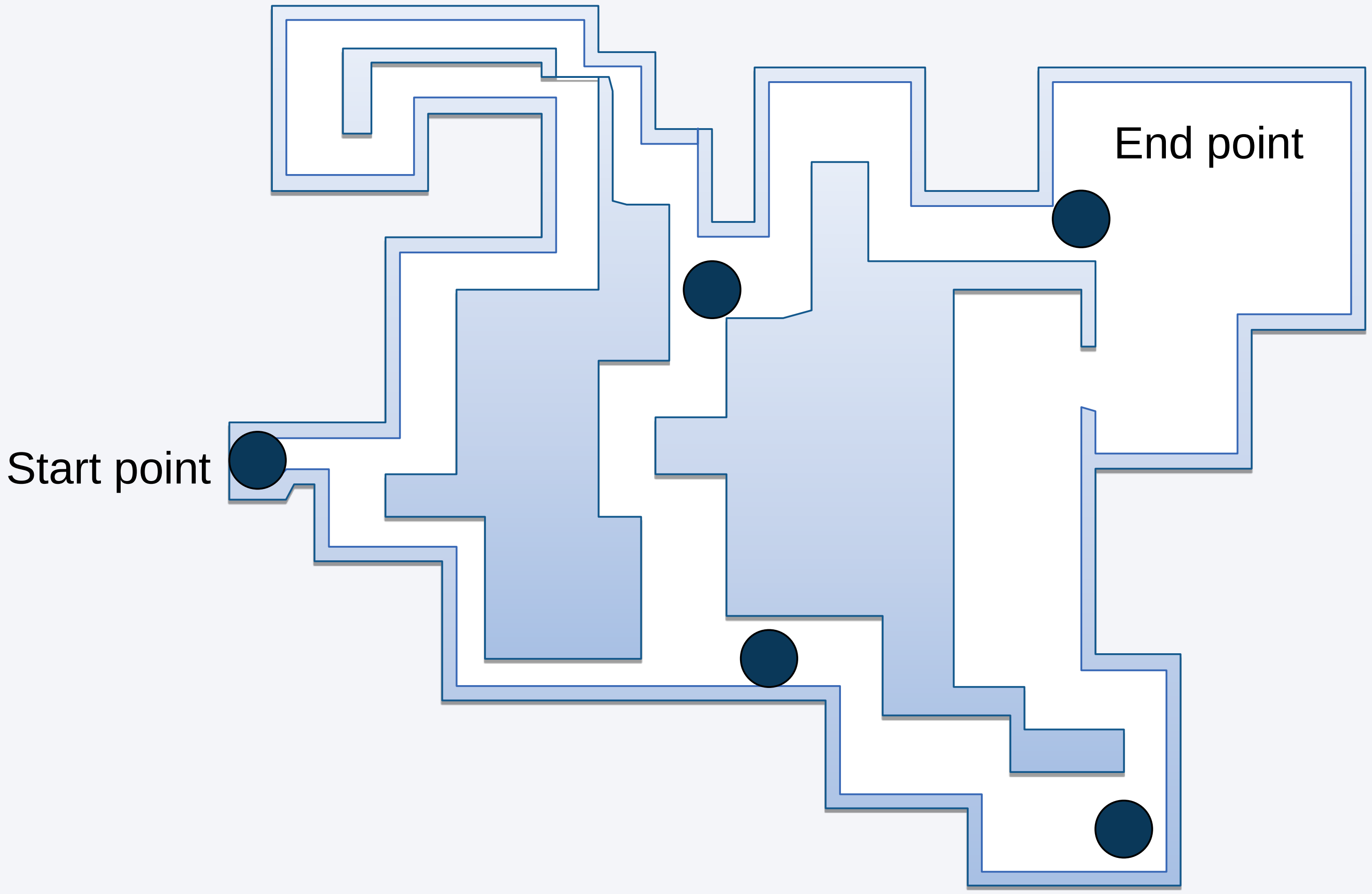
- Techniques to go around the obstacle
  - Choose side deviating the less from initial trajectory
  - Random side
- The algorithm always find a solution if all obstacles are
  - **Convex**
  - **Not connected**

CRASH AND TURN: LIMITS

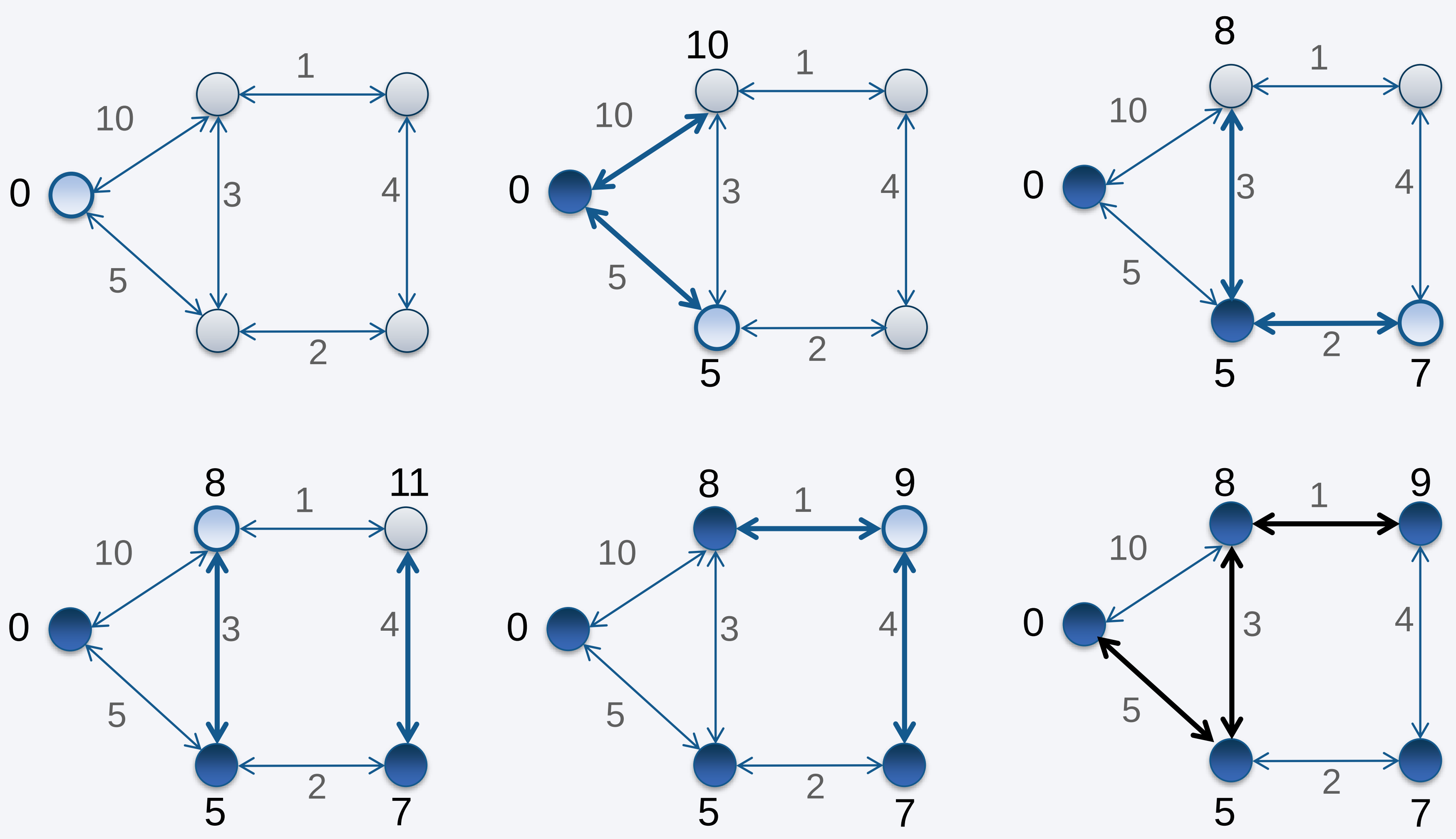


- Find optimal solutions when the geometry of the word can be described as:
  - Set of vertices
  - Weighted connections representing distances
- Popular in FPS
- Global algorithm
- Optimal to find destinations, but **not to trace** a path

DIJKSTRA: MAPS AND GRAPHS



DIJKSTRA: THE IDEA



DIJKSTRA: THE ALGORITHM

```
graph     //graph
start     //starting vertex
end       //ending vertex
distance  //array of distances
previous  //array of previous distances for each vertex

function Dijkstra()
    create vertex set Q
    for each vertex v in graph
        distance[v] = infinity
        previous[v] = -1
        add v to Q
    distance[start] = 0
    while Q not empty
        v1 = a vertex in Q with min distance[v1]
        remove v1 from Q
        for each neighbor v2 of v1
            newDistance = distance[v1] +  weight(v1, v2)
            if newDistance < distance[v2]
                distance[v2] = newDistance
                previous[v2] = v1

function buildPath()
    create empty set P
    v = end
    while previous[v] exists //(e.g. previous[v] != -1)
        insert v at the beginning of P
        v = previous[v]
    insert v at the beginning of P
```

A* (PRONOUNCED A-STAR)

- Space-search algorithm
  - Real-time strategy games
- Path finding method
  - Chessboard
  - States: locations
  - Transitions: unitary movements
- A* evaluates alternative from best to worst
  - Convergence to the best solution

A*: THE GENERAL IDEA

- Init
  - A **base** node
  - A **destination** node
  - A set of **movement** rules (e.g. left, right, up, down)
- At each step, compute possible movements until
  - All movements have been tried
  - We reach the destination
- Necessary to **evaluate** how each node is good
  - Explore better paths first

A*: RATING NODES

- Rating process

$$f(node) = g(node) + h(node)$$

`f(node)`: **total** score for a node
`g(node)`: cost of path **already** traversed
`h(node)`: heuristic estimating the **future**, moves still needed

- g can be the number of steps taken
- h can be for instance a Manhattan distance

```
Manhattan(p1,p2)=abs(p2.x-p1.x)+abs(p2.y-p1.y)
```

- A* produces optimal results with underestimating heuristics
  - Manhattan distance with four-connected worlds
  - Euclidean distance with eight-connected worlds

A*: THE ALGORITHM

```
s.g = 0 //start node
s.h = dist(s) //distance between the goal and s
s.f = s.g + s.h
s.parent = null
push s on open //open is a priority queue with possible candidates

while open is not empty
    pop n from open //n is a node with the lowest f
    if n is the goal node
            construct path and return success
    for each successor n1 of n
            newg = n.g + cost(n, n1) //cost is the distance between n and n1
            if n1 is in (open or closed) and n1.g <= newg
                    skip
            n1.parent = n
            n1.g = newg
            n1.h = dist (n1)
            n1.f = n1.g + n1.h
            if n1 is in closed //closed is a list of already explored nodes
                    remove n1 from closed
            if n1 is not in open
                    push n1 on open
    push n on closed
return failure
```

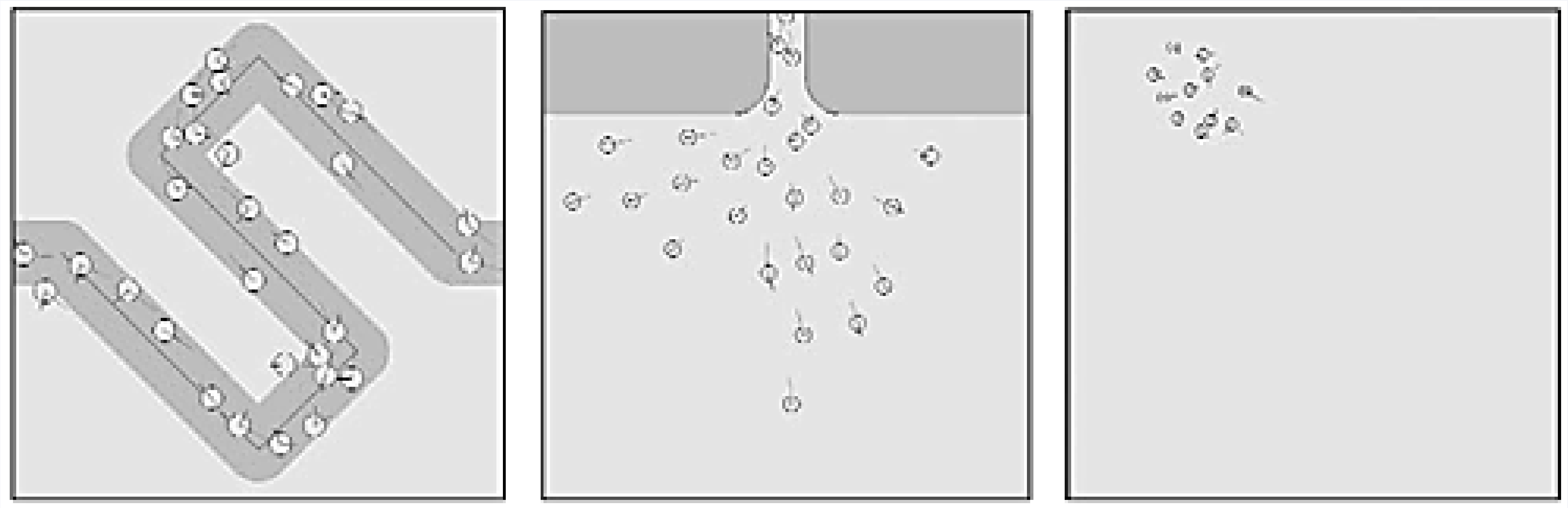
A*: CONCLUSIVE WORDS

- Most widely used path finding algorithm
- Problems
  - Precompute the whole path in one step
    - Unusable with dynamic geometries
    - How to blend it with fog-of-war?
  - A* always produces “optimal” results: **unrealistic**
  - Memory problems
- Improvements
  - Region-based A* (e.g. rooms connected by edges)
  - Interactive-Deepening A* (compute the whole path in small pieces)

Group Dynamics

BOIDS

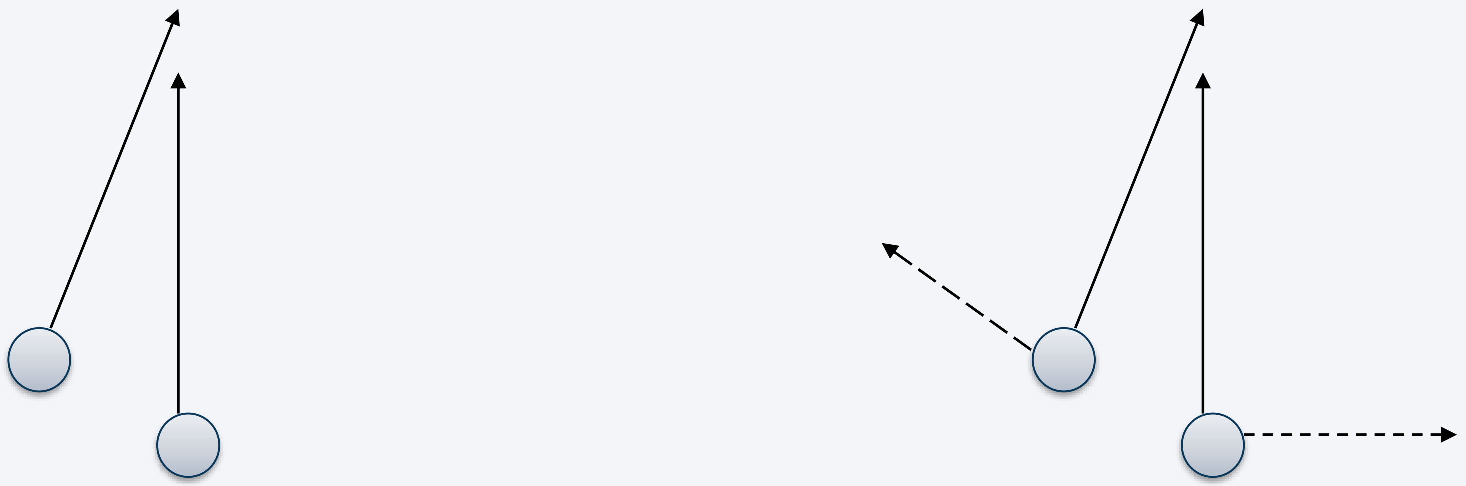
- Algorithm designed to create movies special effects (C. W. Reynolds, 1990s)
  - Used in *The Lion King*



- Boids are a real-time simulation of human and animals groups
- *Core hypothesis*: the behavior of a group is governed by a small set of simple rules
- Flocking behavior modeled with 3 rules
  - **Separation**
  - **Alignment**
  - **Cohesion**

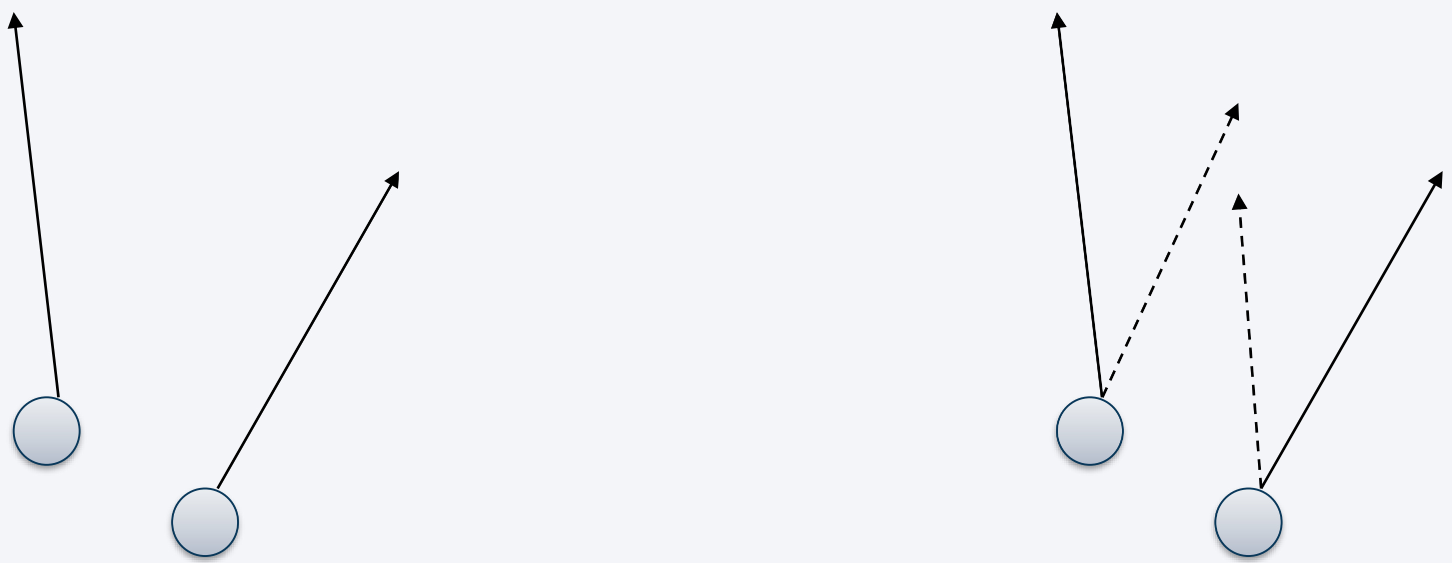
SEPARATION

- Avoid collision in the group
  - A distance threshold between members
  - If the distance is lower than the threshold, both members change their orientation



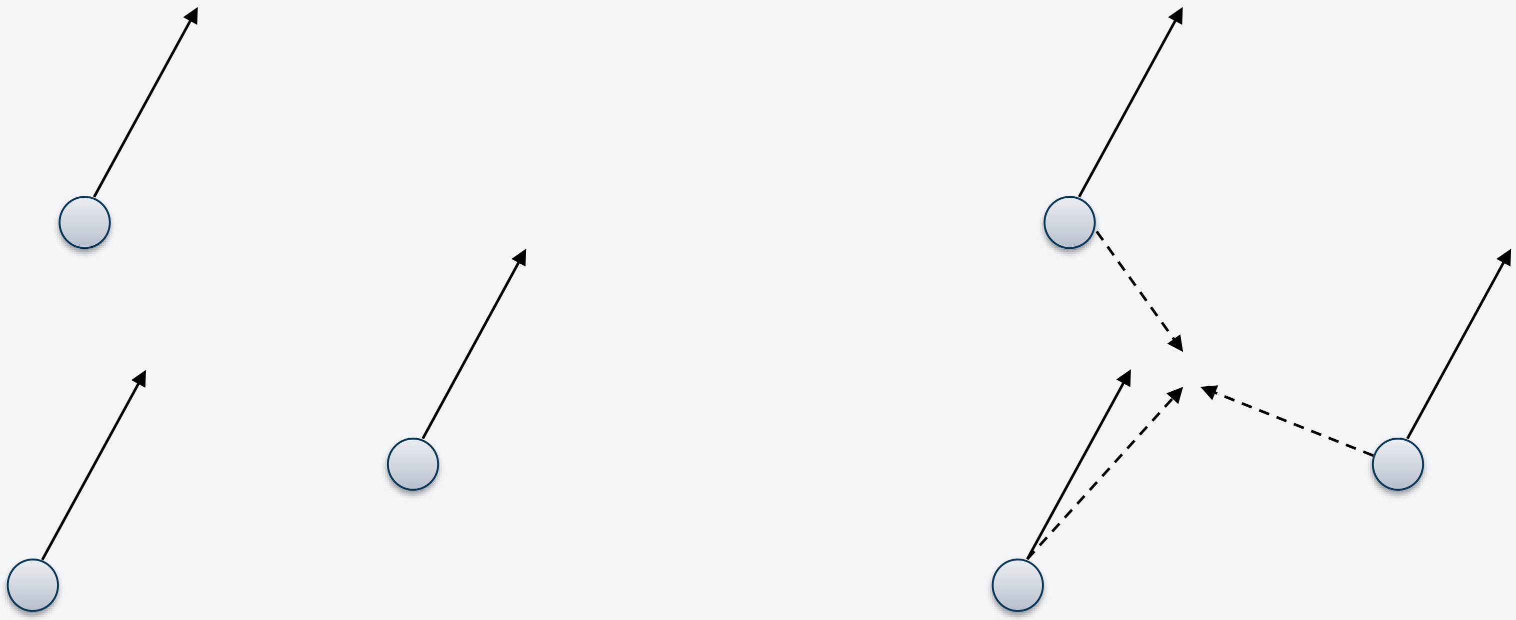
ALIGNMENT

- All members will aim in the same direction



COHESION

- All members try to stay close to the barycenter of the group
  - In respect of the separation rule!



BOIDS IMPLEMENTATION

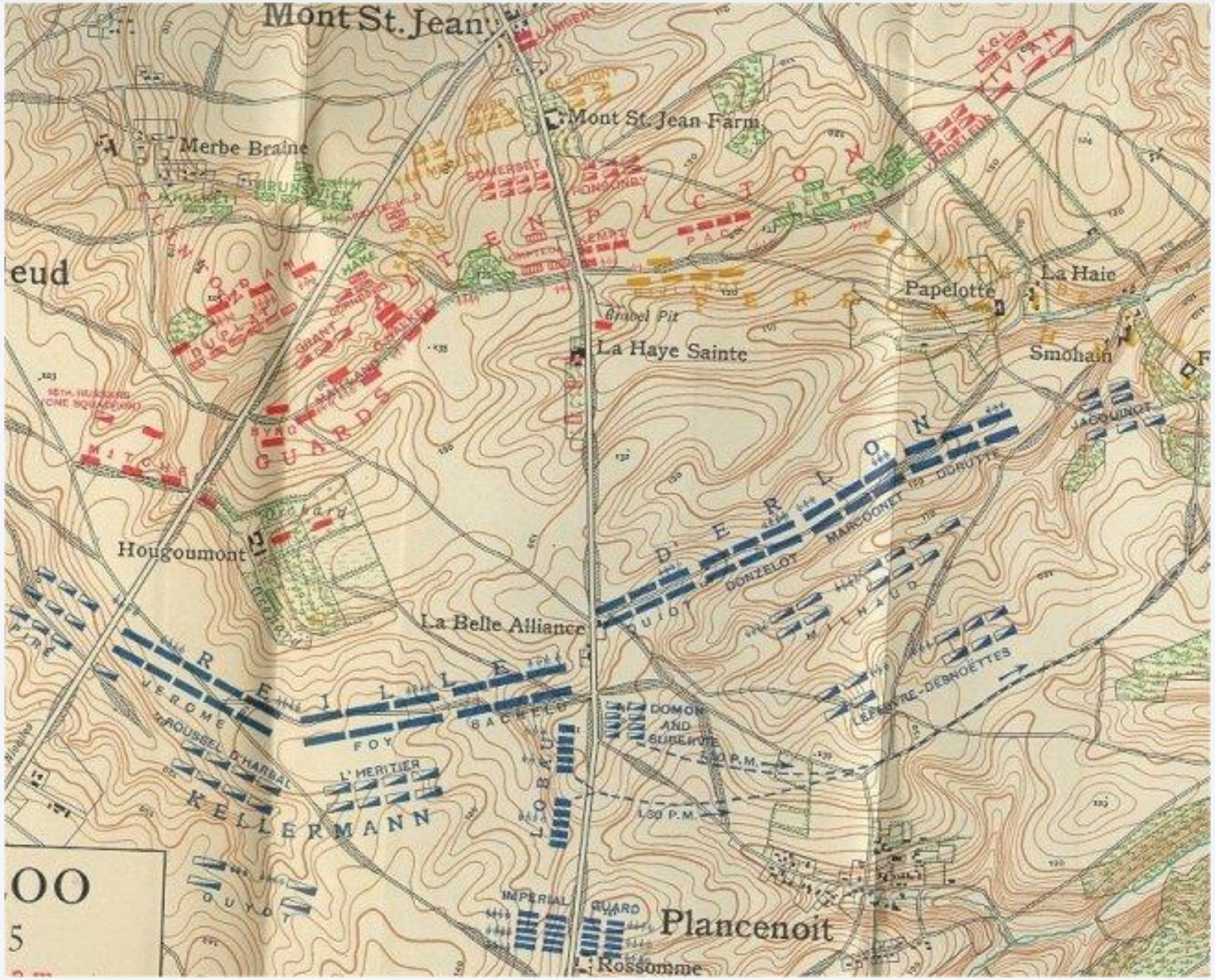
- Boids do **not** need
  - An internal state
  - A working memory
- One of the member is an AI
  - The others will adapt to the “leader”
- Algorithms based on attraction and repulsion laws
- Possible to add additional rules
  - Simulate two populations
  - Obstacles can also be represented as boids

FORMATION-BASED MOVEMENTS

- Human groups dynamic can also be simulated with summations of fields

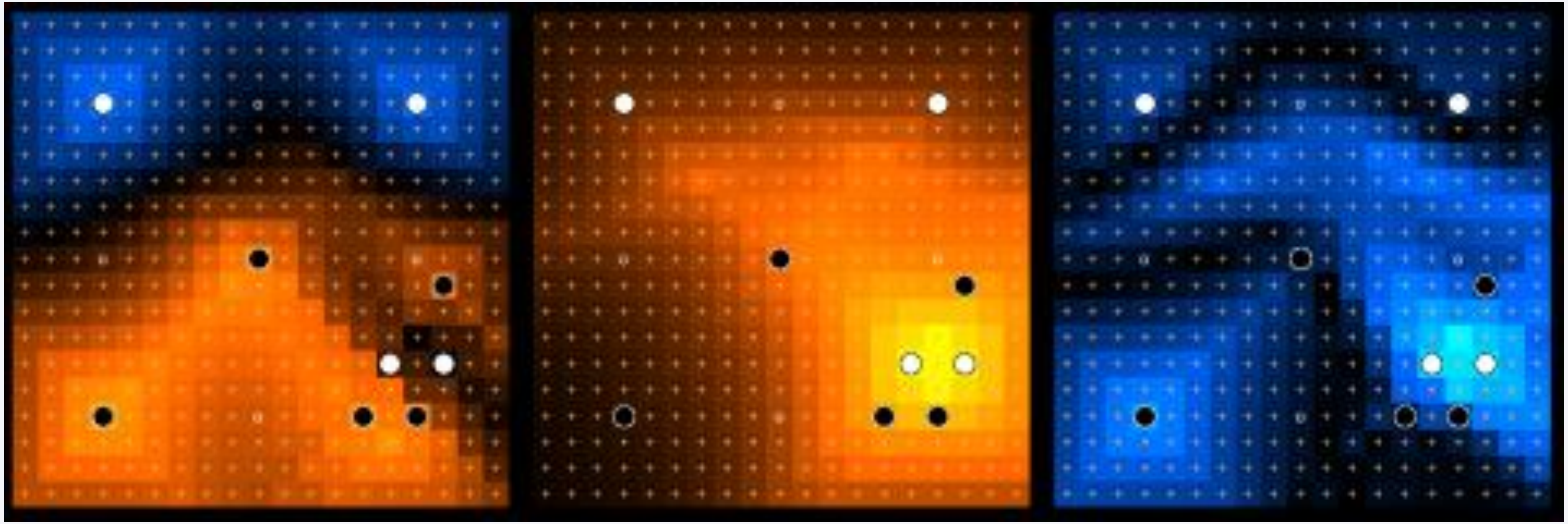
- Military formation
  - 1 field separating units (repulsive)
  - 1 field keeping the position in the formation (attractive)
  - 1 field detecting collisions with obstacles (repulsive)

Military analysis



INFLUENCE MAPS

- Data structures useful to represent balances of power, frontlines, etc.
  - Simple to consult
  - Dynamic



INFLUENCE MAPS USAGE

- Creation of a map for 2 armies (matrix)
  - +1 represent the soldiers of the first army
  - -1 represent the units of the second army
  - Empty cells contain interpolated values
- The size of the IM can be smaller than the size of the world
- IM can also map several values
- Useful tests
  - Frontlines between armies are extracted from zero values
  - The sum of all values indicates the winning army
  - Other tests include breakability, weakest enemy, etc.