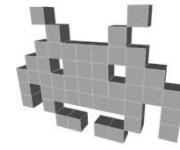
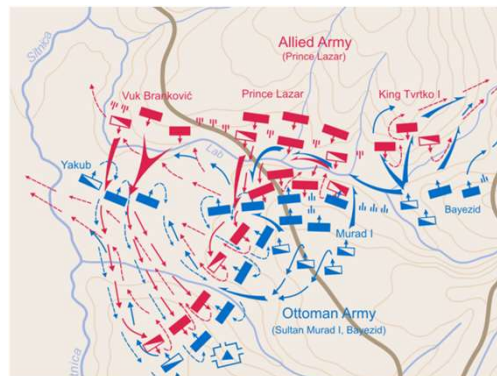


3D VideoGames 2018/2019
Università degli Studi di Milano
**Artificial Intelligence
in 3D Games**



Marco Tarini



1

Game Engine



- Handling common task of a game dev
 - Game logic (levels)
 - Renderer
 - Real time transform + lighting
 - Models, materials ...
 - Physics engine
 - (soft real-time) newtonian physical simulations
 - Collision detection + response
 - Networking
 - (LAN)
 - Sounds (mixing and "sound-rendering")
 - Handling input devices
 - Main event loop, timers, windows manager...
 - Memory management
 - Artificial intelligence module
 - Solving AI tasks
 - Localization support
 - Scripting
 - GUI (HUD)
- Animations
scripted or computed

2

AI / ML in the real world



- **Huge** advancement in recent years!
 - e.g with **deep learning**
 - (neural networks... refurbished)!
 - huge increase of manageable data size
 - very low level features used straight as input
 - e.g. in **data mining**
 - e.g. in **computer vision**
- **Reasons:**
 - algorithm breakthroughs
 - computational power!!!
 - e.g. GP-GPU

3

AI in games. Main use: NPC behavior



Widely different AIs for widely different “NPC”s!

- A wild animal
- An (enemy) soldier
- A squad leader
- An (innocent) villager / bystander
- An individual in a crowd / flock / herd
- A racing car driver
- A spaceship pilot / gunner
- A companion / buddy
- An (enemy) commander
- A zombie
- A heat seeking missile
- A WWII ace pilot
- ...

use
“flocking algorithms”
(or “crowd simulation”)

the AI player
in a RTS

4

AI in games: other uses

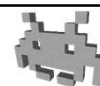


- Procedural... anything
 - terrain
 - levels
 - e.g. maze generation, generation of (solvable!) puzzles...

e.g. look up
"Sokoban"
 - music, models, etc!
- Dynamic difficulty tuning
 - learning when/how to increase/decrease difficulty
 - virtual "movie director" concept
(e.g.: "time to intensify action: spawn more zombies"
/ "time to slow down pace: spawn less zombies")
- Ranking
 - algorithms to estimate rank of players, from game outcomes
(e.g. in chess / go communities)
- An intelligent tutor / advisor
 - e.g. an non-intrusive game tutorial
telling players only what they (seem to) need to hear
- ...

5

AI in games: another (future) use



One promising *emerging* methodology:

- **Procedural Character Animations**
 - i.e. "**learn** how to run, walk, stand up, ..."
 - *Input:*
 - a character body: skeleton structure, with "muscle" actuator

muscle = springs with AI-controlled strengths
 - a given task, e.g.

go as fast as possible in this direction
 stand up from prone position
 reach the highest possible point (i.e. jump)
 ...
 - *Output:*
 - how to activate muscles to do it
 - (minimizing used energy)
 - *How:*
 - genetic algorithms, Evolution strategies
 - physical simulation to score candidates

rig

trivial to
measure
(score)

skeletal animations

6

“AI” for NPC behavior: Interactive Agents (IA)



- Some difference with real AI:
 - “cheating” completely possible
 - e.g. info “magically” available to the [Interactive Agent](#)
 - [real-time](#) response always needed
 - very frequent decisions of the [Interactive Agent](#) (30-60 Hz!)
 - “on-line”, and “soft real time”
 - [sub-optimal](#) often *required*
- NPC behavior also determined by:
 - story telling needs
 - e.g. follow designed behavior, adhere to designed personality
 - difficulty tuning (e.g. with enemy NPCs)
 - need to interesting / fun (\neq optimal!)
 - need to be realistic / believable
 - (not necessary, coherent / logical / optimal)

7

Game AI vs AI to solve Games

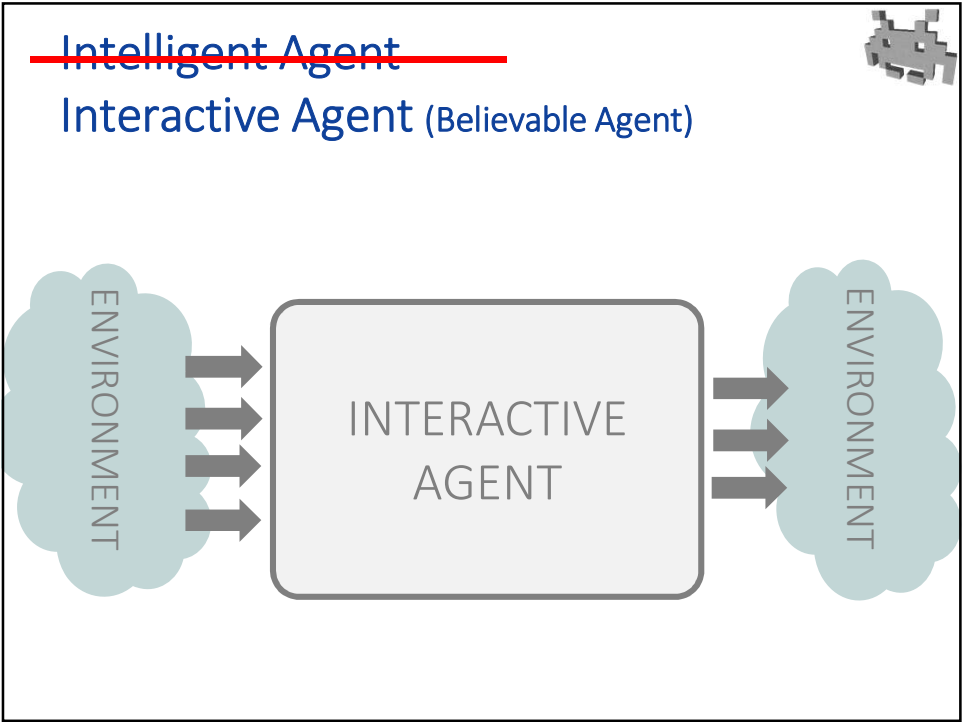


In a word:
entertainment, not problem solving !

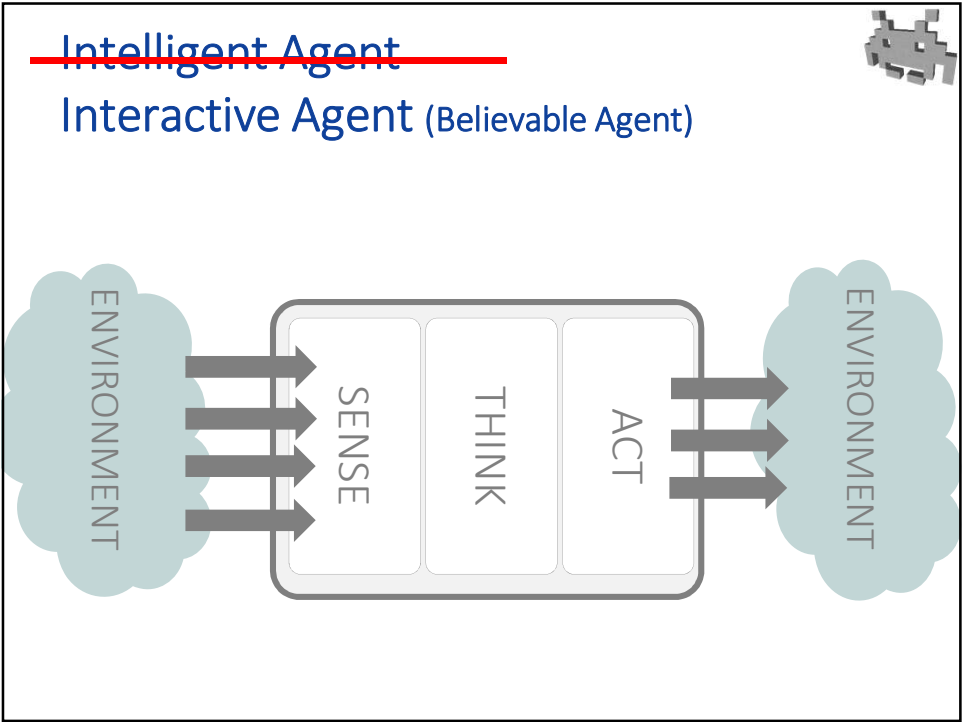
to find more about AI to (optimally) [play](#) games,
look for:

- [min-max algorithms](#) (with pruning)
 - algorithms to solve complete knowledge, turn based games
- [Nash equilibrium](#) (from [Game Theory](#))
 - solution concept to address non cooperative games

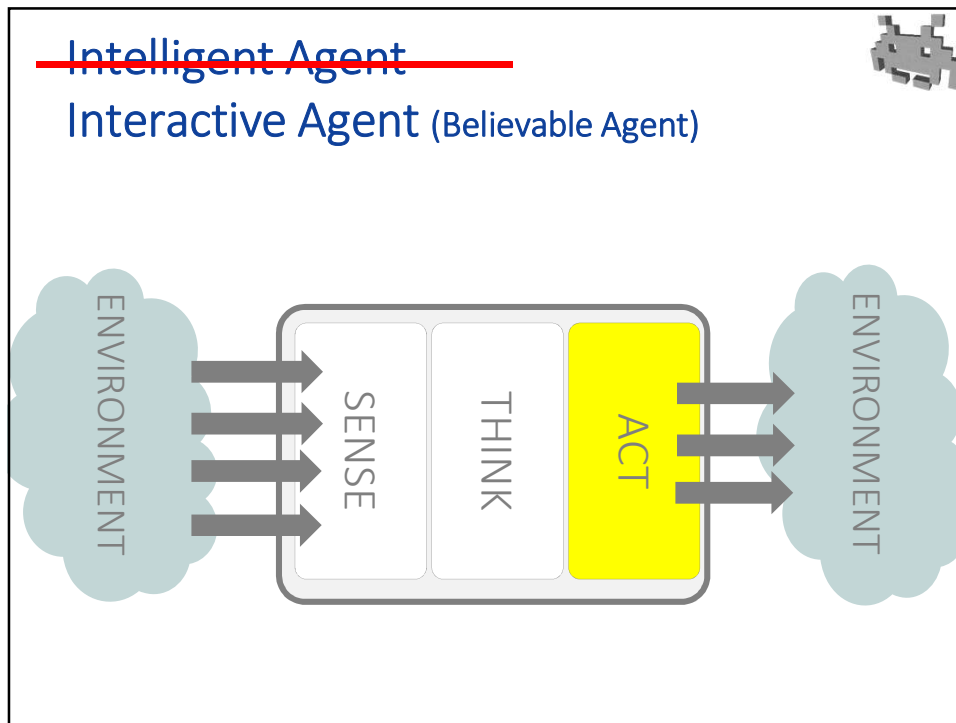
8



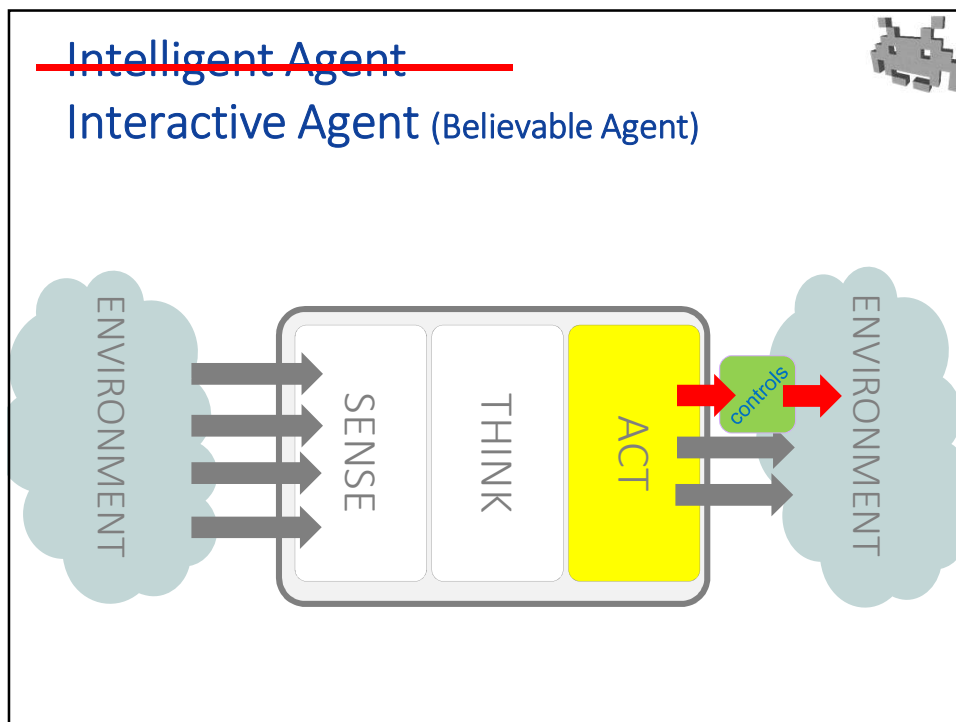
9



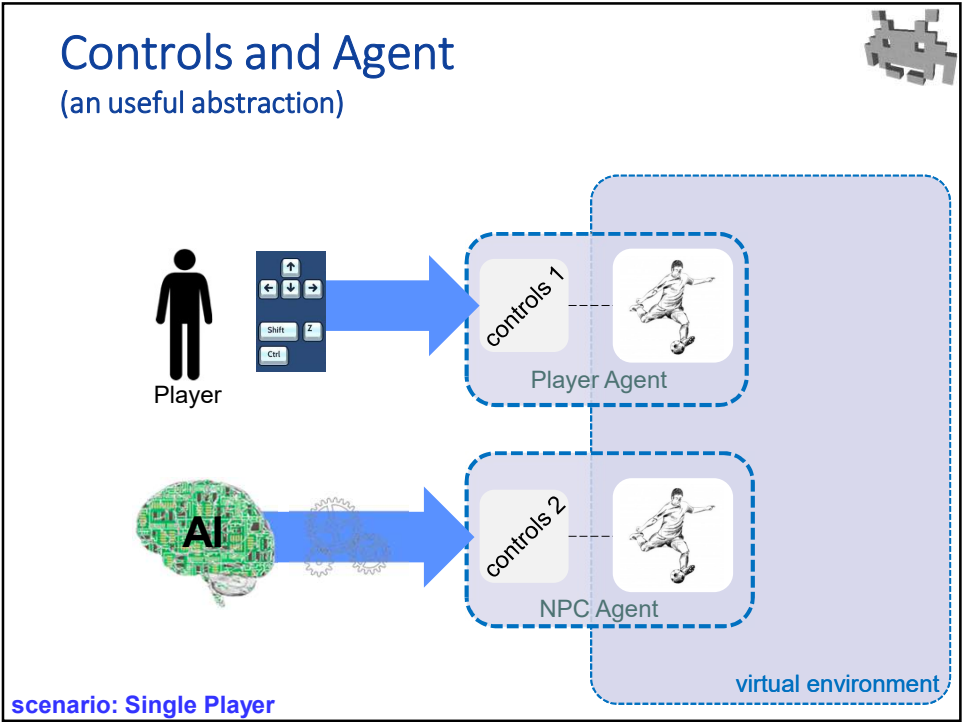
10



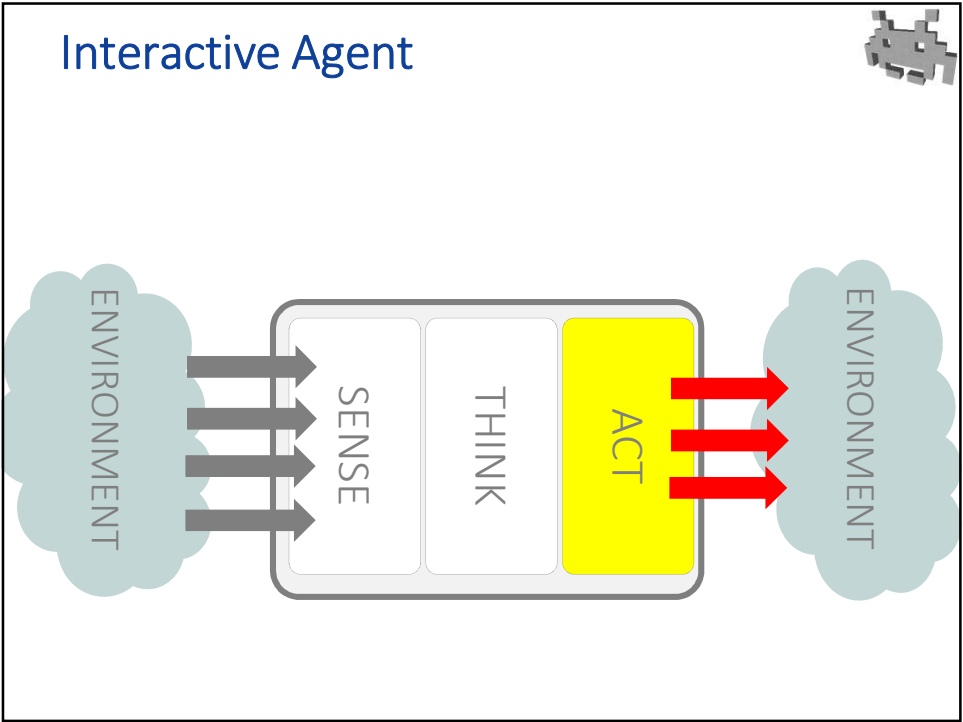
11



12



13



17

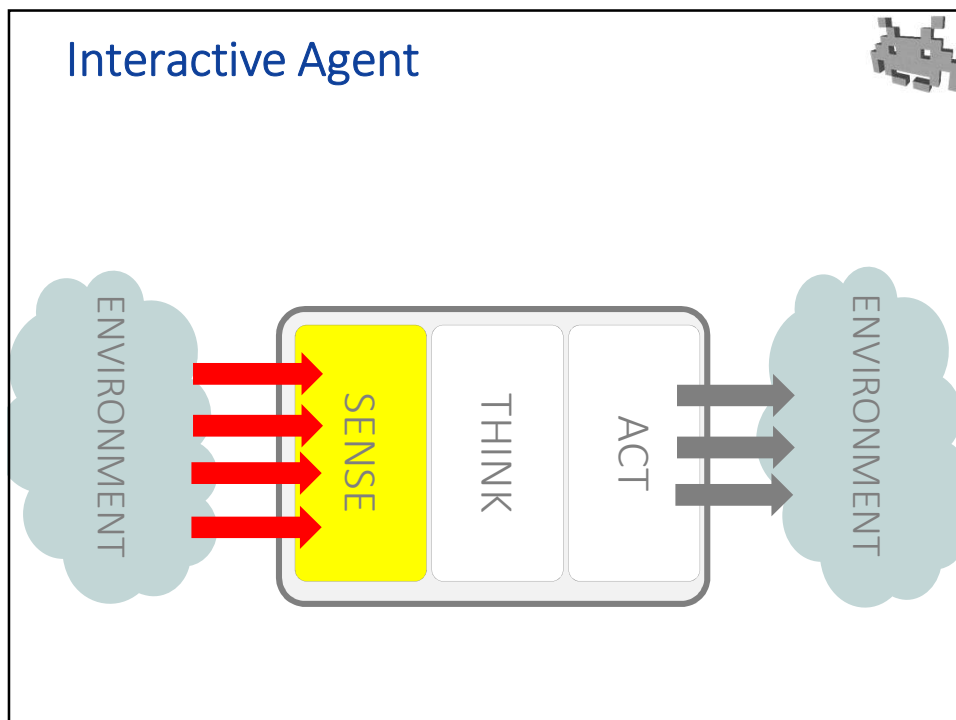
Acts

(in robotics, by “Actuators”. In games?)

- Produce “Controls”
 - associated to the NPC character
 - a **non-cheating** AI controlled NPC (simulation of a player)
- Animations
- Movements / displacements
- Sounds
 - voices, yells
- Orders (issued to other agents)
 - (e.g. in a RTS)
- Effects on **game logic**
 - e.g. objects appearing, doors unlocking, HP decreased / healed, money spent / gain, etc

18

Interactive Agent



19

Sensing

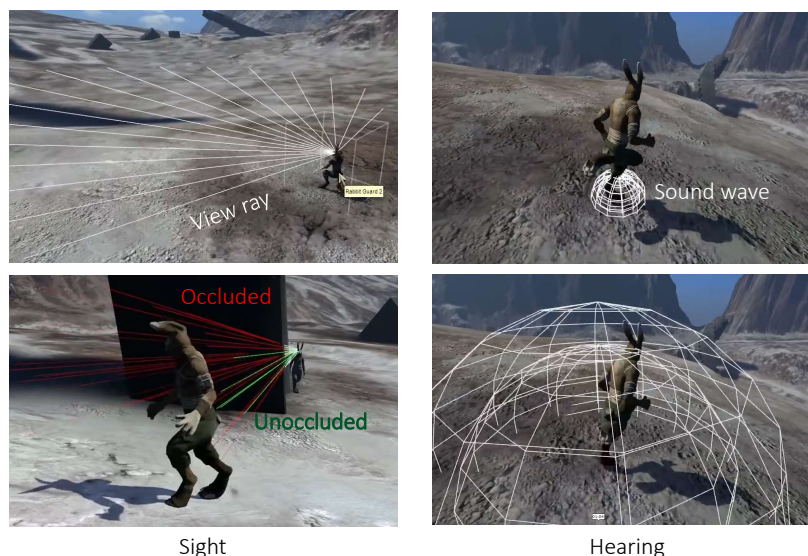
(in robotics, by “Sensors”. In games?)

- Gather info (“percepts”)
 - which will be used for the “think” phase
 - NB: this info must often persist in the “mind” of the agent!
 - more about this in the next phase
- Performed at regular intervals, or “on demand” (by the AI)
- Simulating senses in a 3D world...
 - Sight
 - way1: ray-casting
 - (uses ray-VS-hitbox collision)
 - way2: synthesize then analyze **probe renderings!** (accurate, expensive)
 - Hearing, Smell
 - simple testing against influence sphere
 - Touch / Proximity sensing:
 - collision detection / spatial queries
- ...or “cheating” (common)
 - “magically” sensing data straight from the game status
 - (simple, and often ok – when plausibility not compromised too much)

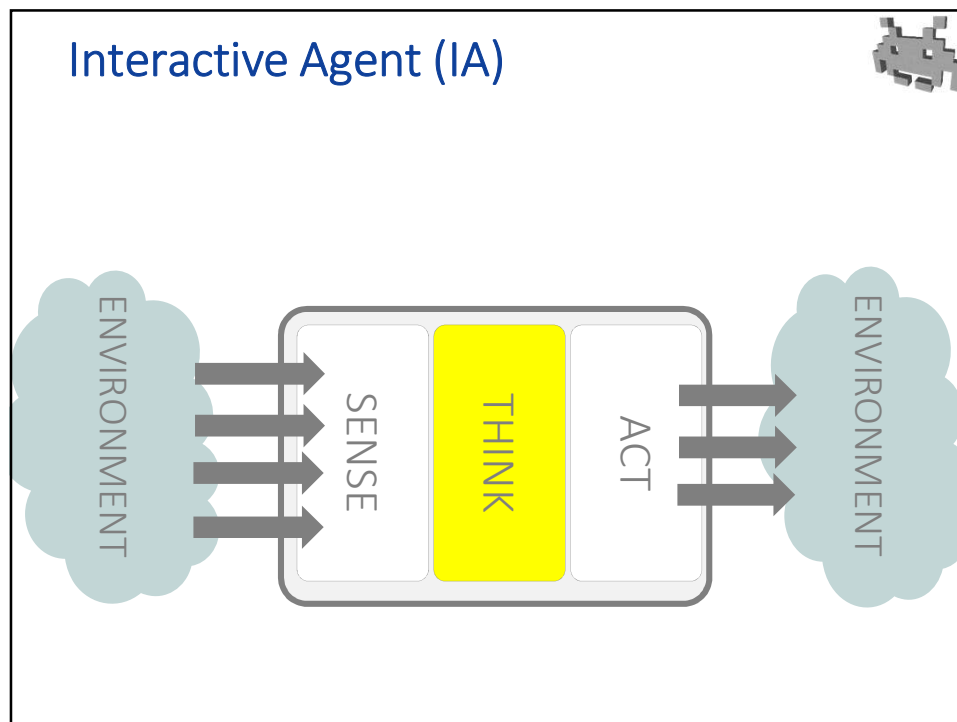
e.g. the scene graph

20

Simulating senses in a 3D environment



21



22

Thinking phase (aka planning)

- Status of the AI: modeling the “AI-mind”
 - current goals
 - hi-level, low-level... (more about this later)
 - internal model of the environment (as perceived by IA)
 - built through the sensing phase
 - occasionally, also obtained from (simulated) communication with other NPCs
 - can be arbitrarily complicated, or very simplistic
 - moods/mindsets
 - internal values modelling the varying lvl of: *fear, patience, rage, distress, confidence, hunger/thirst, fondness toward player*, etc
- persistence of these **mind** elements can be made more or less prolonged
 - e.g. deleted, to model agent forgetfulness
 - e.g. deleted, to reflect awareness that data went stale

23

Thinking phase (aka Planning)



- Typically, Hierarchical Logic
 - Hi-level Decisions => Hi-Level Goals
 - update: not very often
 - ...
 - Lower-level Goals
 - update: more often
 - ...
 - Lowest-level Goals
 - solving low level tasks
 - Acts!



24

Examples of common *lowest level tasks* (1/2)



- Face towards something
 - tip: remember *atan2*
 - actions: turn left or right
- Aim a weapon
 - e.g. including “walking the target”
 - i.e. aim at where target *will* be at time of impact
 - e.g. including ballistic
 - (to predict, use *analytical* physics: $\text{pos}(t) = f(t)$)
- Avoidance / dodging
 - of an incoming bullet
- ...

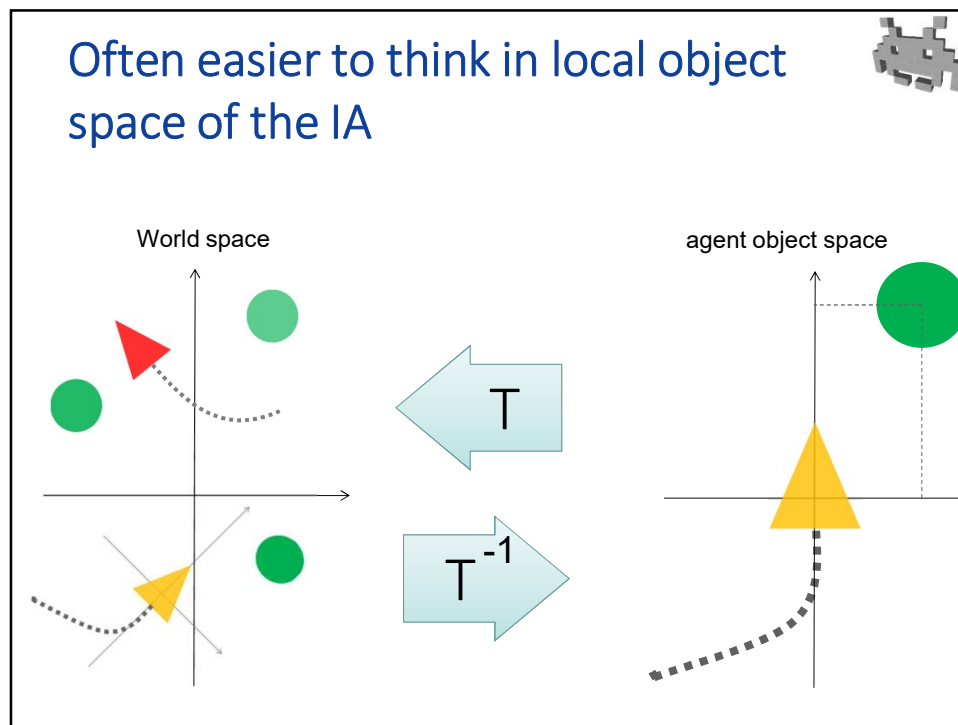
repeat a few times
(converges really fast)

```
vec3 target_pos = target.pos;

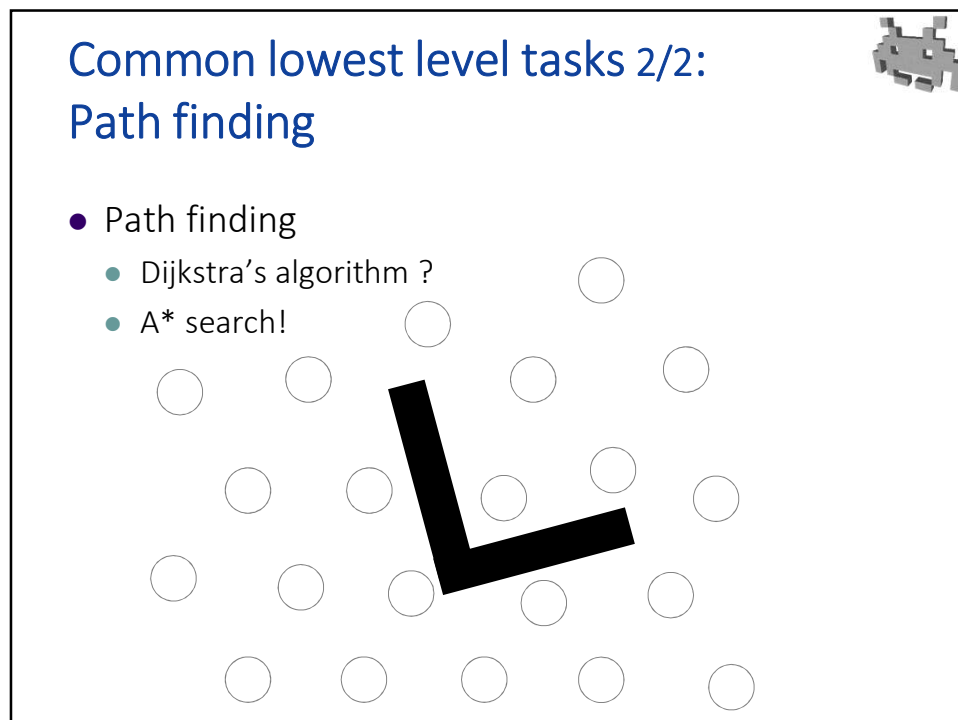
float target_dist = dist( me.pos , target_pos );
float eta = target_dist / bullet_speed;
target_pos = target.pos + target.vel * eta;

face_towards( target_pos );
```

25



26



27

Dijkstra algorithm and A* search



(part of the background, not of this course)



if you are not familiar with them,
please *do* look them up!

28

Dijkstra algorithm: notes



- input:
 - graph (**nodes**, **arches**)
 - nodes = locations where IA can be
 - arches = path to go from node A to node B, such as...
 - straight line paths A to B (to be run / walked)
 - a potential **jump** reaching B from A
 - **drop down** from A to B (note: arches are not necessarily symmetric!)
 - a (positive!) **cost**, associated to each arch
 - e.g. estimated time to go from A to B
 - in general, willingness of the IA to pass through there
 - flexible! easy to adapt costs to reflect specific scenarios, e.g.:
 - "that path is vulnerable to enemy shooting": higher cost
 - "that path is across lava. It hurts! (costs HP)": higher cost
 - "that path occludes friendly fire lines": higher cost
 - "I risk being spotted on that path (I don't want to be seen)": higher cost
 - Start node and Destination node(s)
- output:
 - path from Start to Dest
 - guaranteed to be the minimal-cost path

29

A* algorithm: (“A-star”) notes



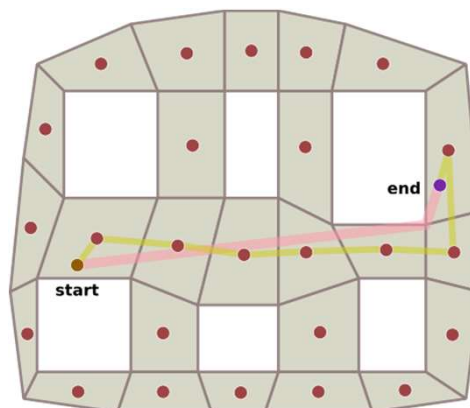
- Dijkstra not efficient enough
 - visits too many nodes
 - explores paths which are obviously wrong
 - (greedy, only guided by **distance from Start**)
- A* variation. Main idea: smarten up! with an **estimate** of the remaining **distance to Dest**
 - function $H(\text{node } x)$: an estimate of the minimal cost to go from x to Dest
 - H is user provided
 - must be: fast (constant time, possibly)
 - must be: strictly optimistic!
 - produced estimations AT MOST the real cost (never more)
 - underestimation ok, overestimation NOT OK
 - good example: simple Euclidean distance (disregarding obstacles!)
- Output: still the optimal path
 - as long as the estimator never overestimates costs
 - the better the estimations, the quicker the algorithm
 - eg: estimation always 0 (technically correct): same as Dijkstra
 - eg: perfect estimation (hypothetical case): only explore nodes in optimal path

30

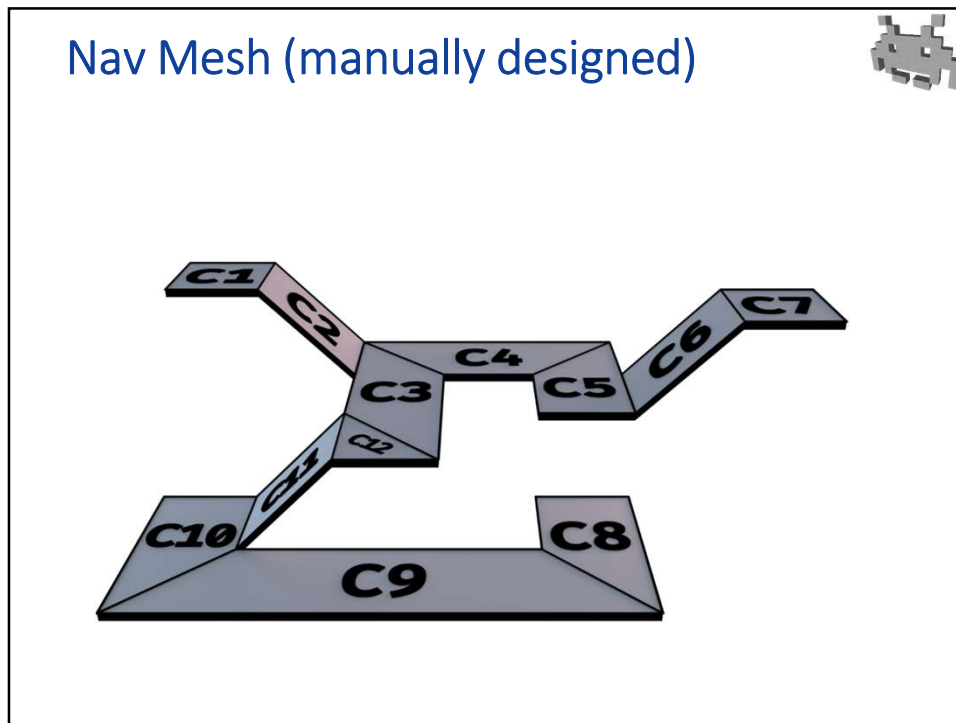
Which graph to use for A* / Dijkstra in a 3D game?



- Answer: Nav-meshes (“Navigation meshes”) or AI meshes
 - a polygonal mesh
 - faces: graph nodes (places where the NPC can stand)
 - edges between faces: graph arches (passage the NPC can traverse)



31

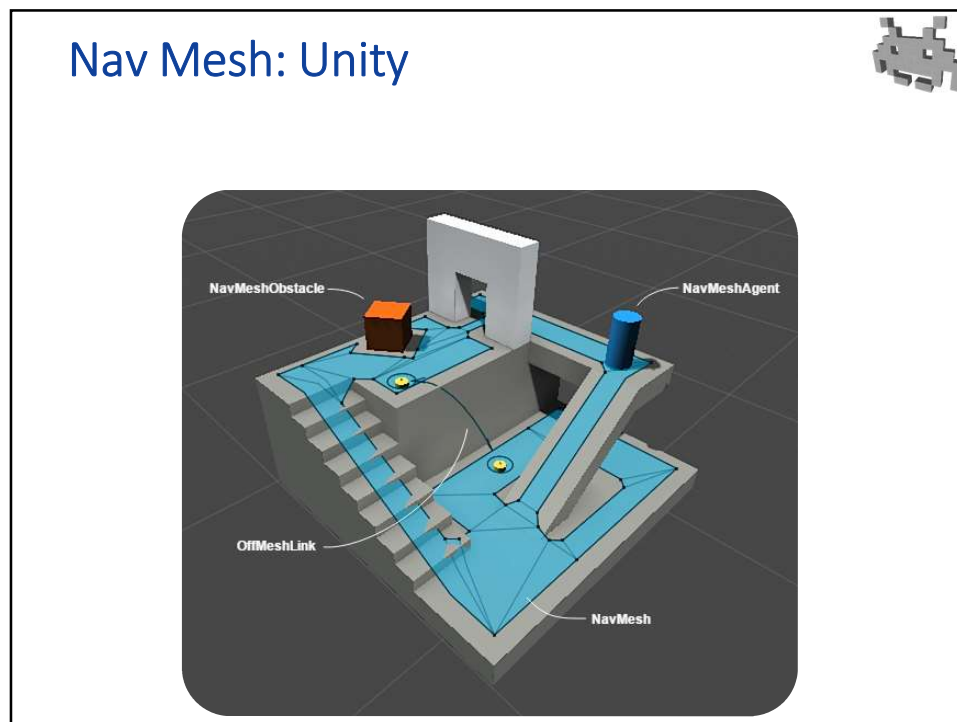


32

Baking a 3D Nav-Mesh

- Input:
 - the scene graph
 - static 3D collision proxies in its nodes
 - a proxy for the NPC (e.g. a capsule)
- Baking
 - Find nodes
 - places where an NPC can stand. How: collisions tests
 - Find arches, for each type of movement
 - Walk: dynamic collision test to determine if it is possible to go from A to B
 - Jump up: heuristics about height differences
 - Jump down: other 3D spatial heuristics
 - Add costs (e.g. time estimations)
- Add ad-hoc or dynamic behavior
 - E.g. add/remove arches when a door gets unlocked/locked,
 - Add/remove arches when a magic teleport portal is activated/deactivated,
 - etc

34



35

Customizing A* / Dijkstra

- Cost function \neq time or distance
- Customize the costs freely
 - E.g. doors: add cost to open them
 - E.g. in a shooter:
 - Increase cost of nodes currently “under friendly fire” (“don’t get in the line of fire of your friends”) ← find out with 3D raycasts
 - Increase cost of exposed nodes (“don’t get caught in the open”)
- Remember: A* needs underestimations
 - Decreasing costs requires care
 - E.g. add teleport doors? Be careful

36

Flocking algorithms



- A mid-level objective: “stay with the group”
 - but “not too close”
- Each element of the swarm targets the position of the 3D barycenter swarm
 - But avoids collision with closer members
- ==> decent flocking behavior emerges
 - E.g. flock of birds, school of fishes
 - But this is just the ABC of flocking algorithms
 - Many subtilities can be added

37


Other mid-level objectives in 3D games



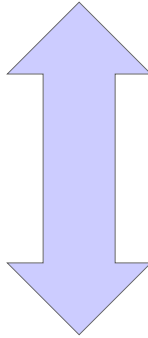
- Often, completely ad-hoc strategies:
 - E.g. driving games:
 - compute-and-bake (or manually edit)
the optimal 3D path in each racing circuit
 - e.g. as a b-spline curve or as a segmented curve
 - Just make NPC cars target the path position ahead of them (mid level), but avoid collisions (low level)
 - => decent racer behavior emerges

38

Authoring an AI for an NPC




- Cascading goals
 - Hi-Level Goals
 - Low-Level Goals
 - Lowest-level Goals
 - Acts



39

Authoring an AI for an NPC: *classic approach*



- Cascading goals
 - Hi-Level Goals ← FSM
 - Low-Level Goals ← Scripts
 - Lowest-level Goals ← Scripts /
Hard-Wired Subroutines
(by the AI engine)
 - Acts

40

Example: terrified bystander



- Cascading goals

- Hi-Level Goal *I'm "Escaping"*
- Low-Level Goal *I'm going to *that* hiding spot*
- Lowest-level Goal *I'm passing through here*
(find route to it -- navigation)
- Acts *(actual movements + "panicked-run" animation)*

41

Example: WWII soldier



- Cascading goals

- Hi-Level Goal *I'm Sniping*
- Low-Level Goal *I'm going for *that* enemy soldier*
- Lowest-level Goal *I'm aiming at *this* (x,y,z)*
(the center of his exposed head)
- Acts *crouched-aim animation*
+ turn left by 2.5 deg
+ IK to re-orient rifle vertically

42

Example: guard



- Cascading goals

- Hi-Level Goal

I'm "Patrolling"

- Low-Level Goal

I'm going to
3rd *Nav point*

- Lowest-level Goal

I'm passing through *here*
(find route to it -- navigation)

- Acts

(actual movements +
"alerted-walk" animation)

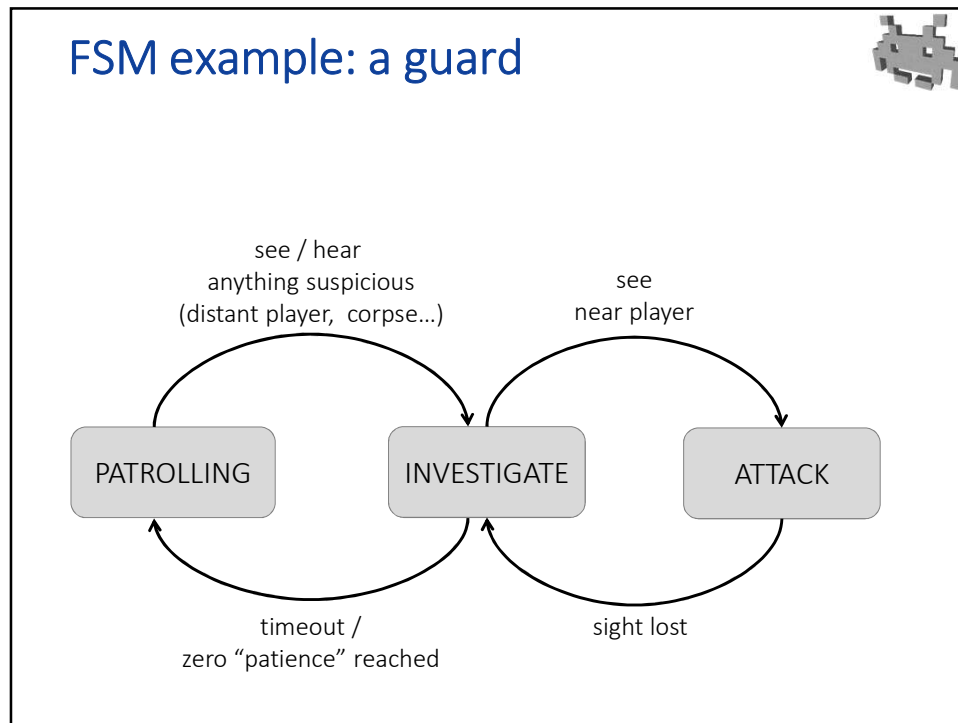
43

Background FSM (more technically: Moore machines)



- Nodes = states
- Arches = transitions
 - associated to arches: input (senses, events)
 - associated to states: output (actions)
 - current state: state of the IA mind

44



46

FSM in practice

- Just a scripting guideline
 - one "status" variable
 - transitions: manually coded in
- Or, a **behavior authoring tool**
 - intended for the **AI designer**
 - hardwired support, by game AI engine
 - maybe WYSIWYG editor
 - transitions: conditions (to be checked automatically)
 - statuses: linked to effects (sound, animation,...)
 - (small advantage: avoids real time script interpretation ==> can be more efficient)

```
if (status==PATROLLING)
then doPatrolling();
if (status==ATTACK)
then doAttack();

procedure doPatrolling(){
// ...
if next_nav_point reached ...

// state transitions
if (target_in_sight)
then status = ATTACK;
}
```

47

Authoring an AI for an NPC: more tools

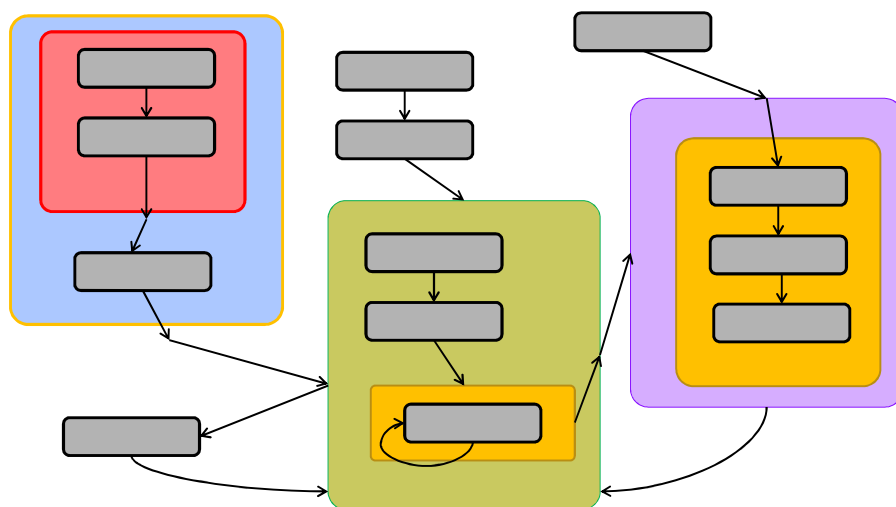
- Problem with the **FSM** approach :
 - does not scale well with world / behavior complexity
 - quickly produces very complex nets
 - (ok, for simple behavior)
- Alternatives:
 - **HFSM**
 - **Behavioral Trees**

unified handling of all levels;
blur classic distinction between
hi-level / low-level planning.

also blur classic distinction between
sensing / thinking / acting

48

HFSM Hierarchical Finite State Machines



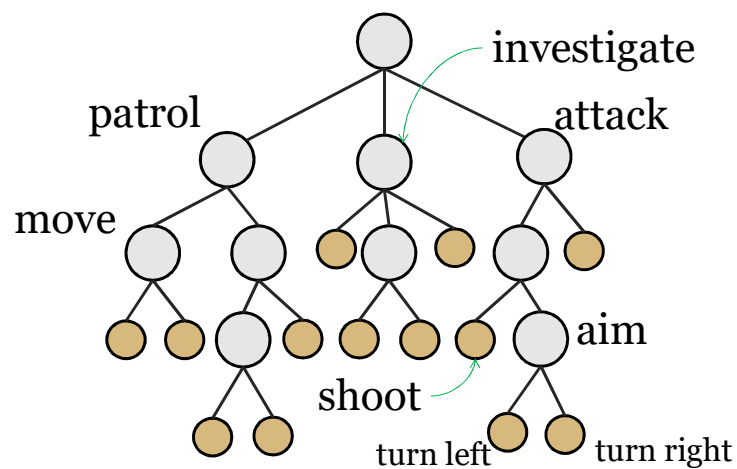
49

HFSM: concept

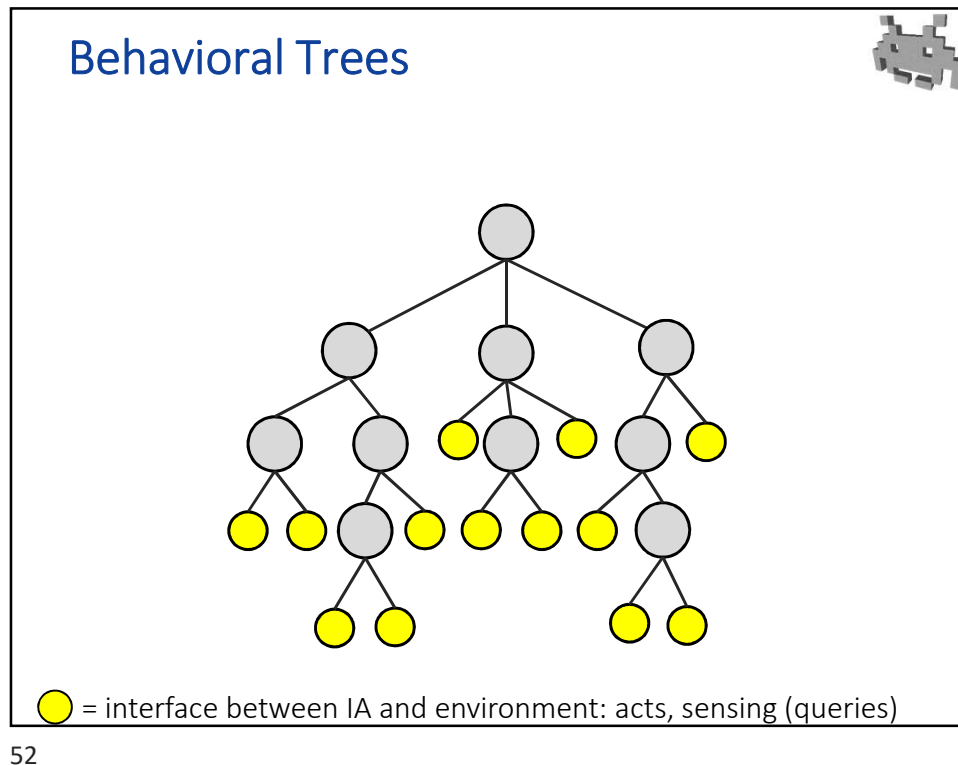
- A FSM where a state can be a sub-FSM
 - meta-state = sub-FSM
 - meta-transitions =
checked from any state of the current sub FSM
 - recursive (multiple levels)
- Advantages:
 - easier design
 - aids reusing chunks of behavior
(from an AI to another)

50

Behavioral Trees



51



52

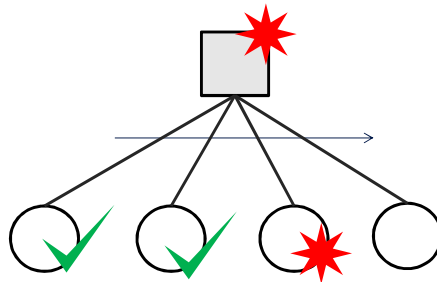
Behavioral Trees: nodes

- every node, when it has done *running*, can either:
 - ✖ fail
 - ✓ succeed
- **leaves**: interaction with environment
 - **action** leaf:
 - animations, movements, sound, game logic...
 - Success: done it.
 - Failure: could not do it
 - (e.g. movement negated by obstacle, object not in inventory...)
 - **sense** leaf:
 - queries on senses, on game status, ...
 - Success / Failure: query result
 - (e.g see / not see an obstacle in front of IA)
 - distinction not necessarily strict

53

Behavioral Trees: nodes

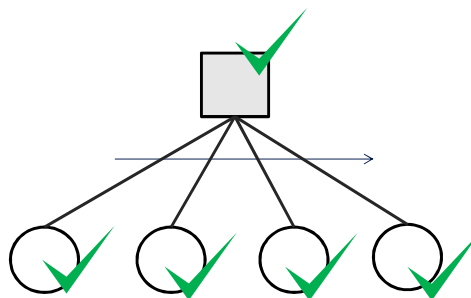
- internal nodes: **sequence**



54

Behavioral Trees: nodes

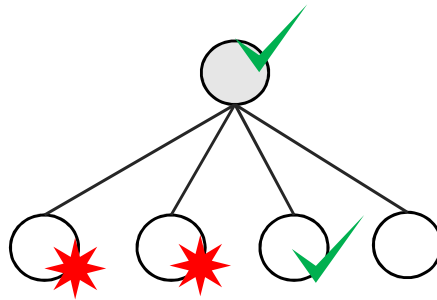
- internal nodes: **sequence**



55

Behavioral Trees: nodes

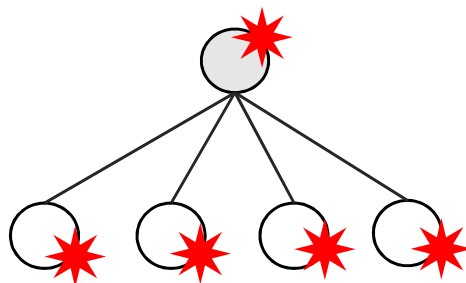
- internal nodes: **selector**



56

Behavioral Trees: nodes

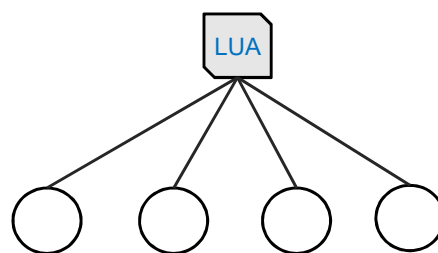
- internal nodes: **selector**



57

Behavioral Trees: nodes

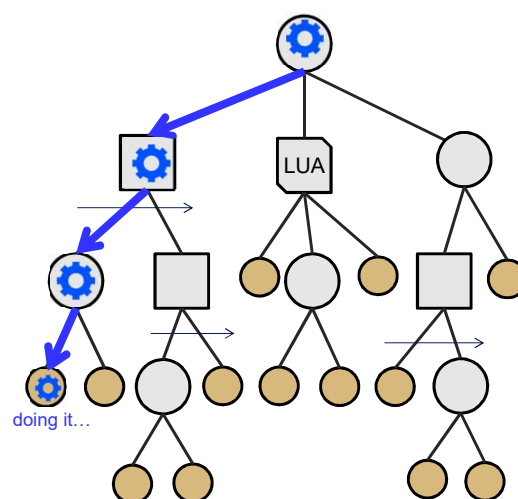
- or, nodes can be programmed arbitrary (scripted procedure) (LUA, C#, ...)
 - run children, as calls
 - fail or succeed, as returned value



BT as
a framework to
structure /
reuse /
organize
scripts

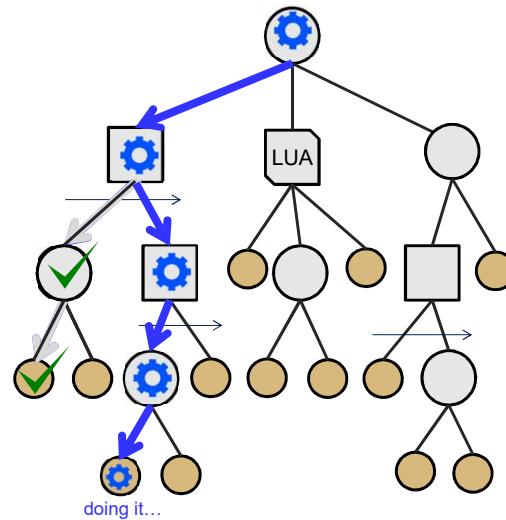
58

Compute behavior: visit tree



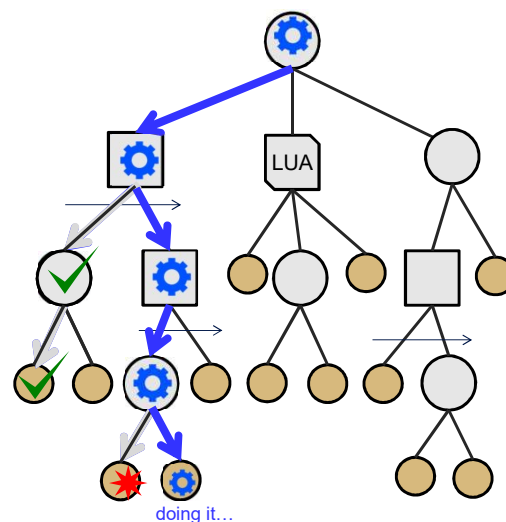
59

Compute behavior:
visit tree



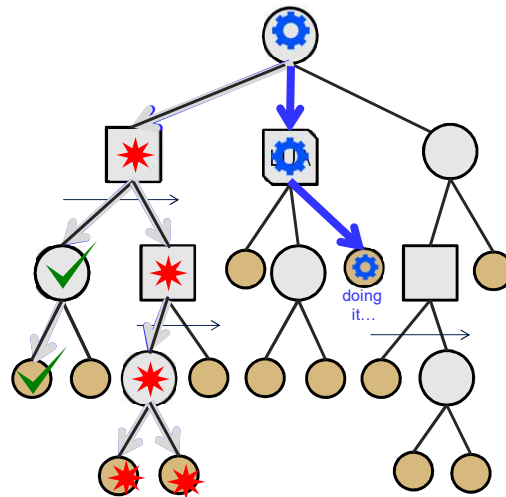
60

Compute behavior:
visit tree



61

Compute behavior: visit tree

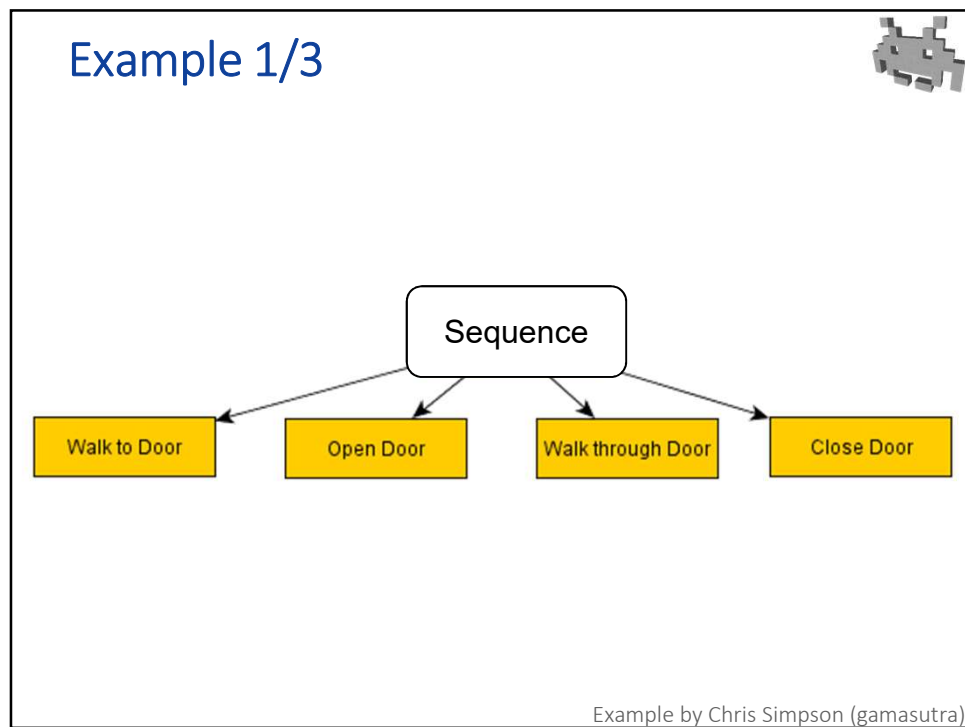


62

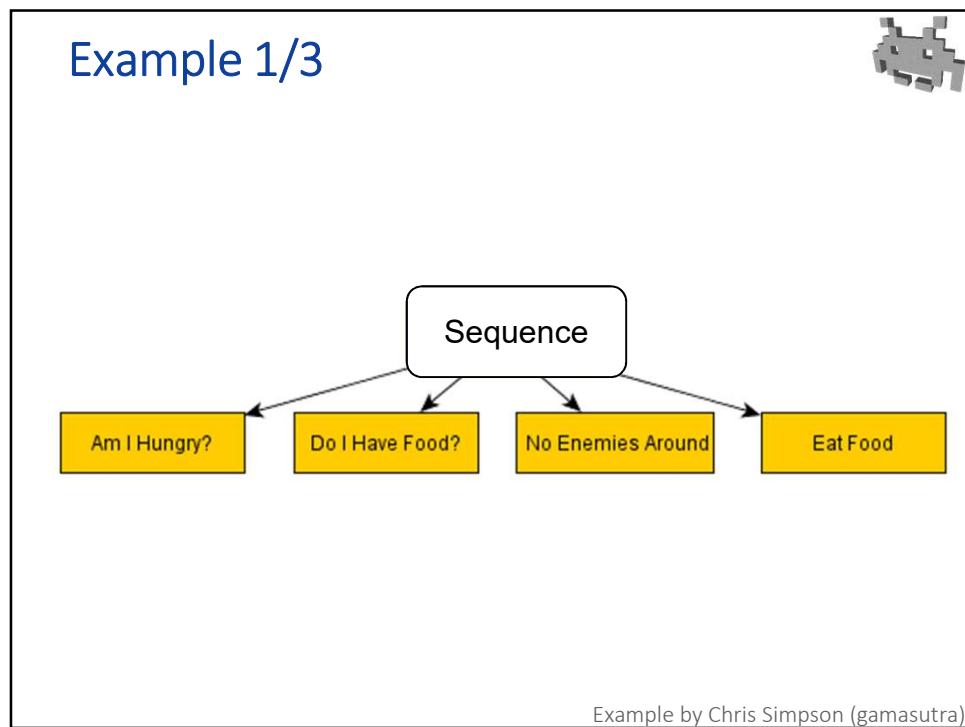
Behavior trees: notes

- Each node can be:
 - failed (red star)
 - success (green checkmark)
 - in progress (blue gear)
 - (or still unvisited) (grey dot)
- Current IA-mind status: path from root to leaf
 - Nodes in the path are (blue gear)
 - Low depth nodes: high-level objectives
 - High depth nodes: low-level objectives
 - Leaf of the path: current action / sensing action

63

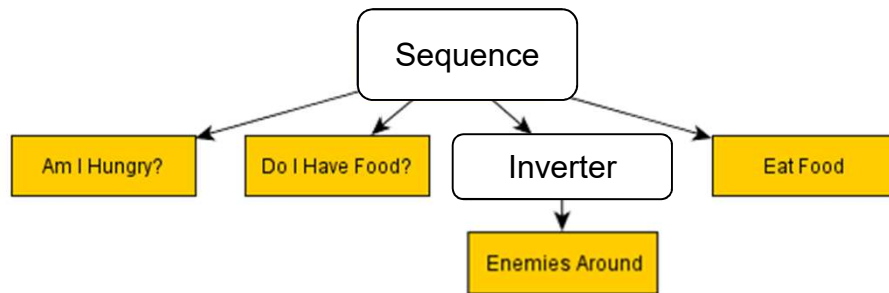


64



65

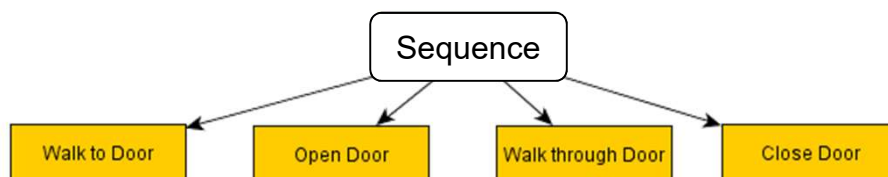
Example 1/3



Example by Chris Simpson (gamasutra)

66

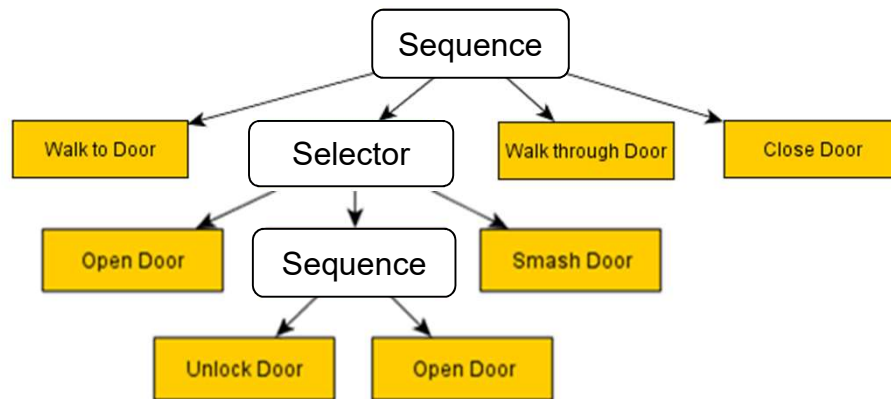
Example 1/3



Example by Chris Simpson (gamasutra)

67

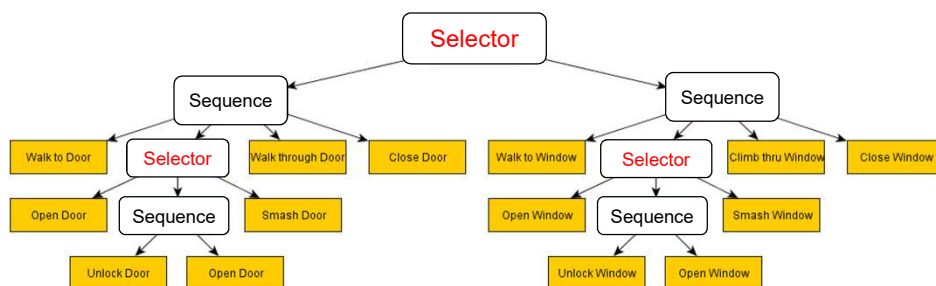
Example 2/3



Example by Chris Simpson (gamasutra)

68

Example 3/3



Example by Chris Simpson (gamasutra)

69

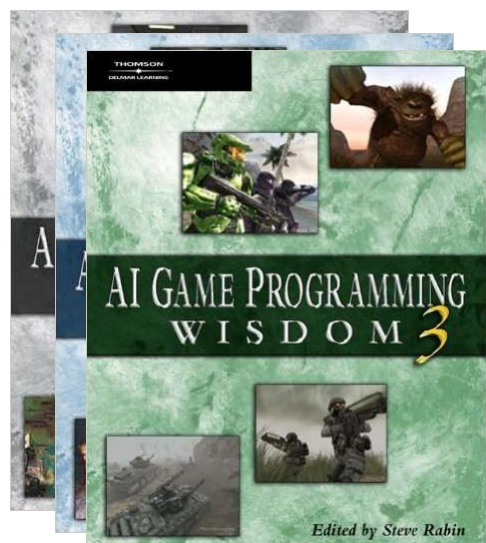
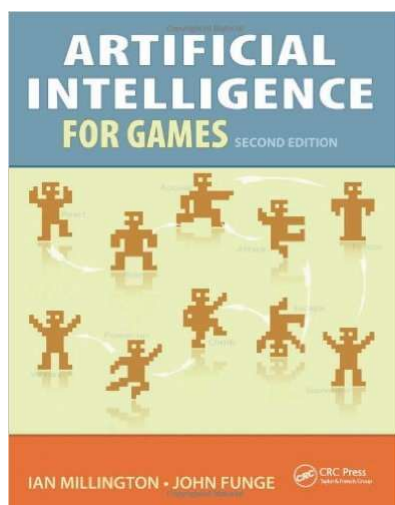
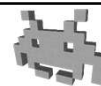
AI support in a game engine: a summary



- **Assets** for (NPC) AI:
 - for *behavior modelling*:
 - **Scripts** (can well be the only one)
 - **FSM**
 - **HFSM**
 - **BT**
 - for *navigation*:
 - **nav-meshes** (aka **AI-meshes**)
 - for *sensing / queries*:
 - **hit-boxes**, **bounding volumes**, **spatial indexing**
 - the same ones used by **physic engine** for **collision detection**
- **Game tools**
 - to assist their construction (by AI designer)
- **Support for a few hard-wired functions**
 - to solve lowest level tasks

70

To investigate further



71