

3D VideoGames  
Università degli Studi di Milano



## Artificial Intelligence for 3D Games


---

Marco Tarini



1

## Course Plan



- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: **Game 3D Physics** ●●●●+●●
- lec. 5: **Game Particle Systems** ◐
- lec. 6: **Game 3D Models** ◐●
- lec. 7: **Game Textures** ●●
- lec. 9: **Game Materials** ◐
- lec. 8: **Game 3D Animations** ◐●●●
- lec. 10: **Networking** for 3D Games ◐
- lec. 11: **Artificial Intelligence** for 3D Games ● (highlighted with a red arrow)
- lec. 12: **3D Audio** for 3D Games ●
- lec. 13: **Rendering Techniques** for 3D Games ●

For a general, deeper discussion of many of the subjects of this lecture, see the course «AI for videogames»

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
    - data used straight as input for learning
  - e.g., in **data mining**
  - e.g., in **computer vision**
- Main reasons:
  - computational power: tera-FLOPS by GPUs
  - huge collection of data to use as the input of the Learning process!

4

## AI in games: examples of uses



- Procedural generation of...
  - levels
    - e.g., maze generation, generation of (**solvable!**) puzzles...
  - terrain
  - music, models, scenes, enemies...
- Automatic dynamic tuning of difficulty
  - 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., a non-intrusive game tutorial telling players only what they (seem to) need to get

e.g., look up "Sokoban"



6

## AI in games: one promising use (a trend in research)

- **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

Diagram: A blue arrow points from the 'Input' section to the word 'skeleton'. Another blue arrow points from the 'Output' section to the words 'skeletal animations'. A bracket groups the tasks under 'Input', with an arrow pointing to the text 'trivial to measure (scoring function)'.

7

## AI in games, the paradigmatic 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
- ...

Diagram: An arrow points from the text “flocking” algorithms or “crowd simulation” to the list item “An individual in a crowd / flock / herd”. Another arrow points from the text “the AI player in an RTS” to the list item “An (enemy) commander”.

8

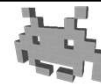
## “AI” for NPC behavior: Interactive Agents (IA)



- Many differences with “problem-solving” 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., for enemy NPCs)
  - need to interesting / fun ( ≠ optimal!)
  - need to be realistic / believable
    - not necessary, coherent / logical / optimal

9

## NPC behavior: designer perspective

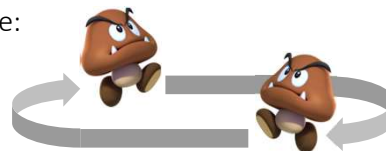


NPC behavior is not *necessarily*

- “intelligent”
- nor even complex

Rather, NPC behavior often needs to be:

- intuitable / predictable
- learnable
- understandable
- story driven
- exploitable (interesting to exploit)



Allowing game-designers to:

- tune difficulty
- elicit interesting strategies by the players
- make a given strategy rewarding

10

## 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:

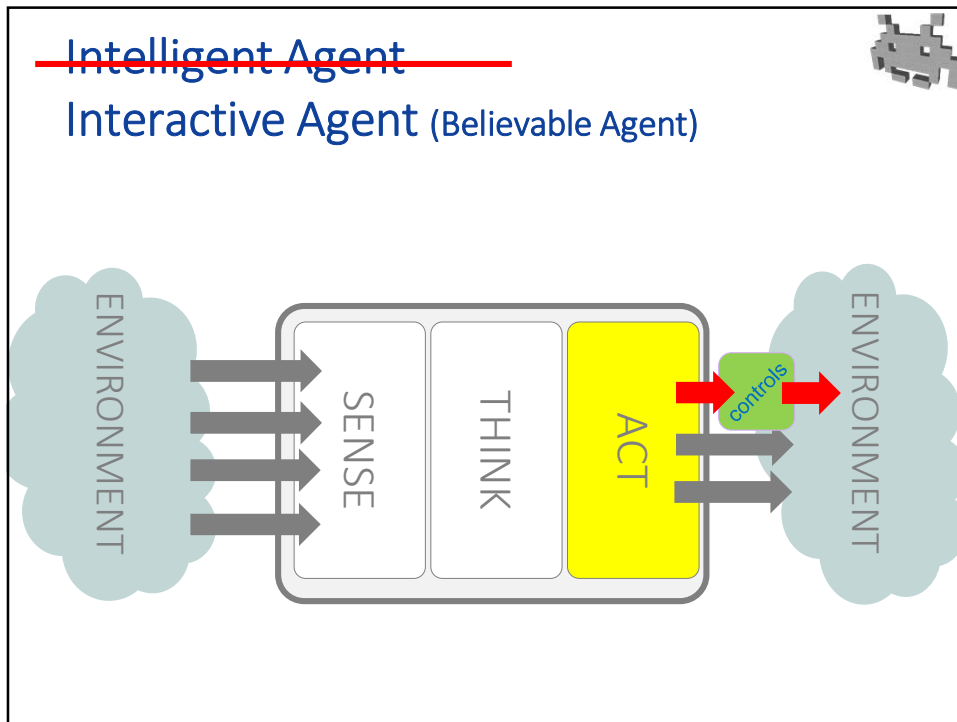
- min-max algorithms (with pruning)
- algorithms to solve complete knowledge, turn based games
- Nash equilibrium (from Game Theory)
- solution concept to address non cooperative games

11

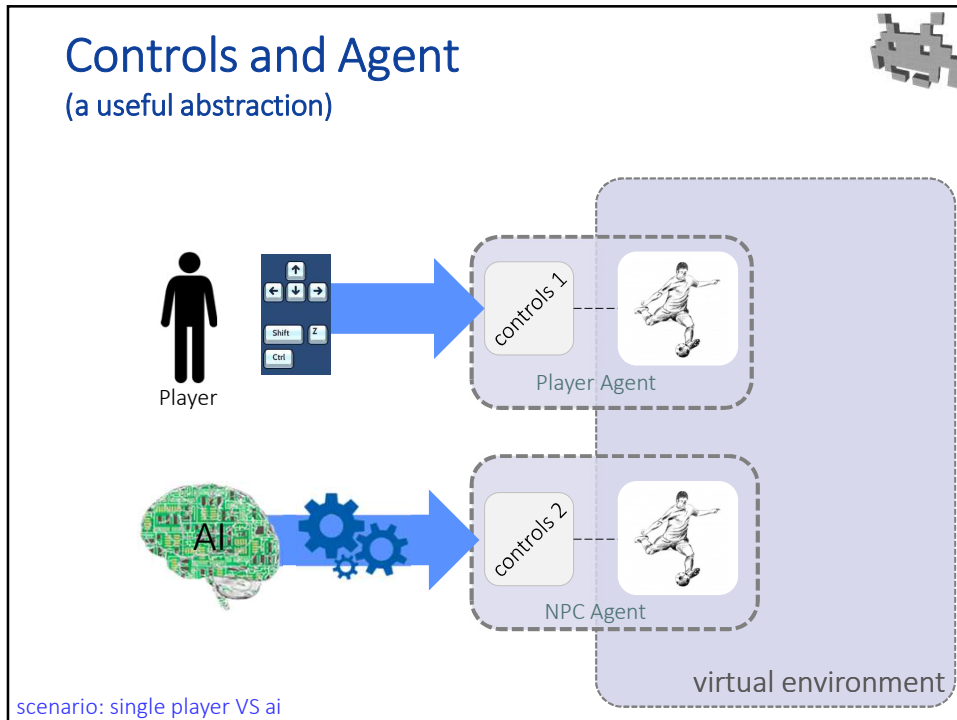
## ~~Intelligent Agent~~ Interactive Agent (Believable Agent)



12



15



19

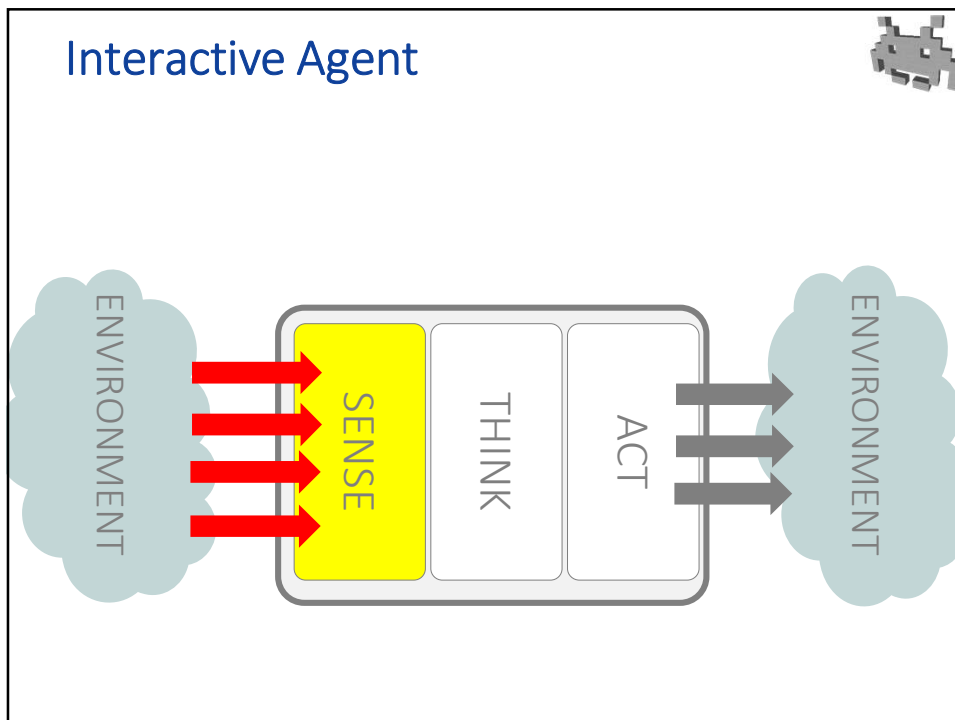
## Acts :

In robotics, “actuators”. In 3D games?

- Produce “Controls”
  - associated to the NPC character
  - a **non-cheating** AI controlled NPC (simulates a human player)
- Trigger skeletal **animations**
- Cause **movements** / displacements of IA avatar
- Play **sounds**
  - voices, yells
- Issue **orders** (to other agents)
  - e.g., in an RTS
- Trigger effects on **game-logic**
  - e.g., objects appearing, doors unlocking, HP decreased / healed, money spent / gain, etc

22

## Interactive Agent



23

## Sensing

In robotics, by “Sensors”. In games?

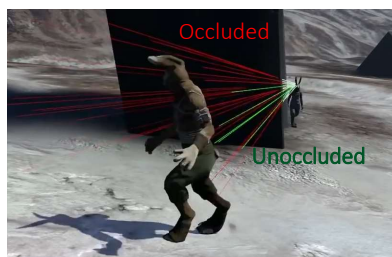
- Gather info (“percepts”)
  - which will be used for the “think” phase
  - NB: this info must often persist in the “mind” of the agent!
    - more about this in the next phase
- Performed at regular intervals, or “on demand” (by the AI)
- Simulating senses in a 3D world...
  - Sight
    - way1: ray-casting
      - (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



24

## Simulating senses in a 3D environment



Sight

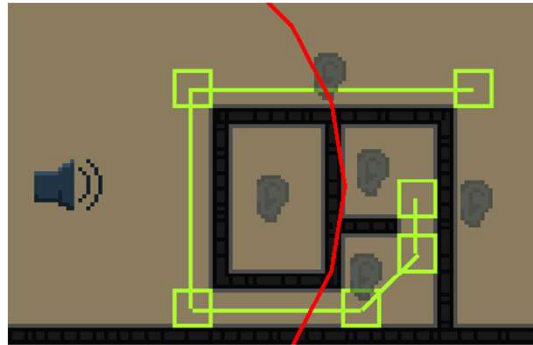
Hearing

25



## Simulating senses in a 3D env. Example: sound (with echos)

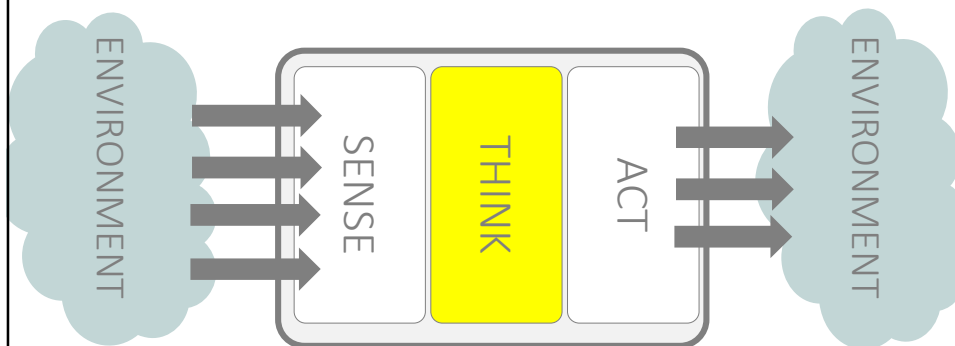
- Pathfinding for echos simulation



example from **Tendril: Echo Received** by **cepnnox** <https://forums.tigsource.com/index.php?topic=60709.0>

26

## Interactive Agent (IA)



27

## Thinking phase (aka Planning): includes status / memory of the AI



- Status of the AI: modeling the “mind” of the AI
  - current goals
    - hi-level, low-level... (more about this later)
  - internal model of the environment (as perceived by IA)
    - accumulates info gathered by senses
    - occasionally, also obtained from (simulated) communication with other NPCs
    - can be arbitrarily complicated, or very simplistic
  - moods/mindsets/dispositions (e.g., toward player)
    - 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

28

## Thinking phase (aka Planning) Goals of the AI

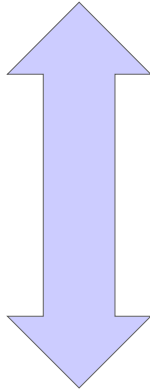


- Typically, Hierarchical Logic
  - Hi-level Decisions => Hi-Level Goals
    - update: not very often
  - ...
  - Lower-level Goals
    - update: more often
  - ...
  - Lowest-level Goals
    - solving low level tasks
  - Acts!

29

## Authoring an AI for an NPC

- Cascading goals
  - Hi-Level Goals
  - Mid-Level Goals
  - Low-level Goals
  - Lower-level Goals
  - ...
  - ...
  - Acts



30

## Authoring an AI for an NPC: *classic approach*

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

31

## Example: terrified bystander



- Cascading goals

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

32

## 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*

33

## 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

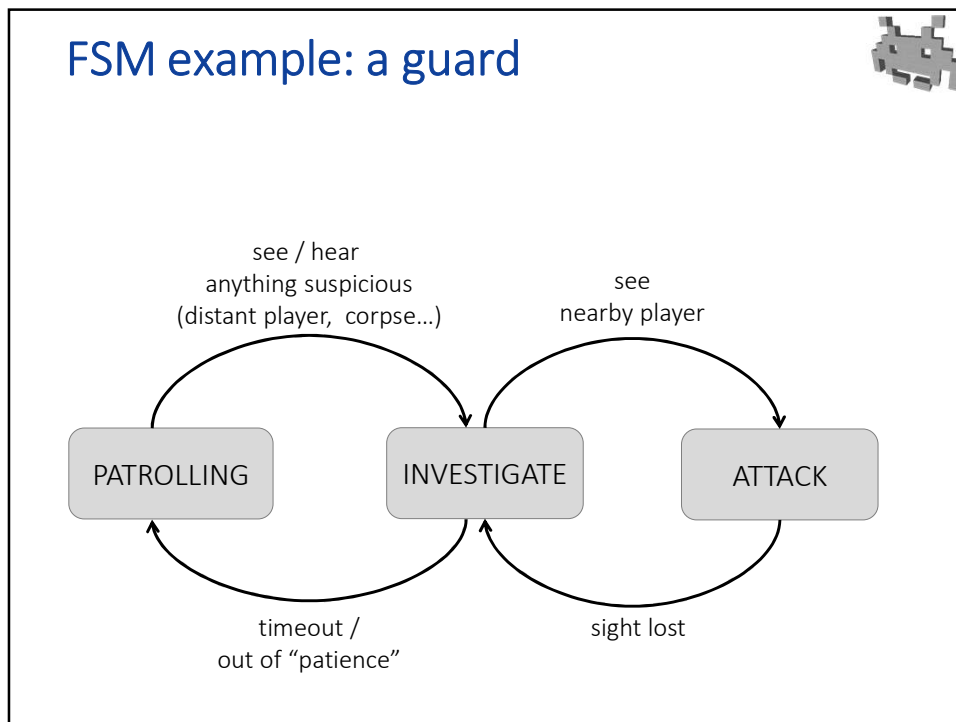
34

## FSM (more technically: Moore machines)



- Nodes = states
  - Associated to actions / behavior routines
  - Current node: current state of the IA mind
- Arches = transitions
  - associated to senses / external events  
(including time-has-passed event)

35



37

### Coding FSM

- FSM can be a coding guideline
  - use one "status" variable
  - transitions: manually coded in
- Or, a **behavior authoring tool**
  - intended for the **AI designer**
  - hardwired support, by game AI engine
  - maybe edited with WYSIWYG editor
  - transitions: conditions (to be checked automatically)
  - statuses: linked to effects (sound, animation,...)

```
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;
}
```

38

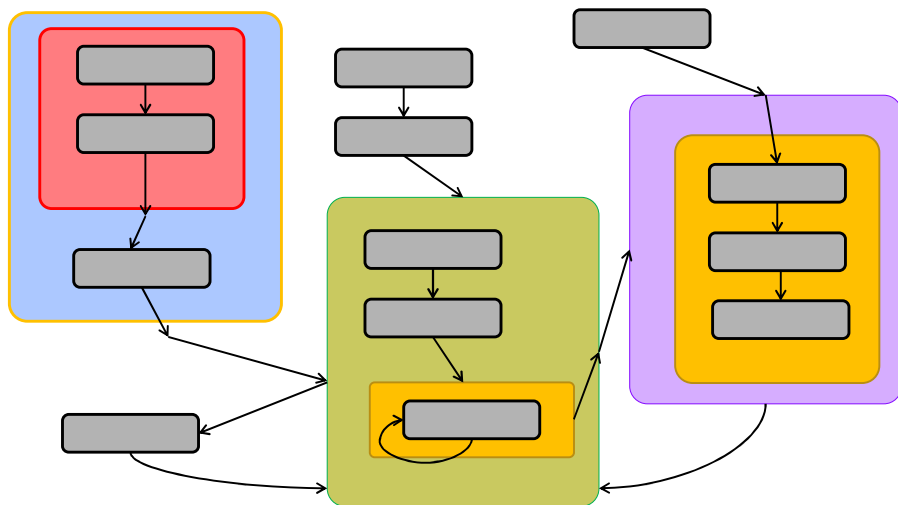
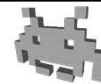
## Authoring an AI for an NPC: more tools



- Problem with the FSM approach :
    - does not scale well with world / behavior complexity
      - quickly produces very complex nets
      - (it's ok, for simple behavior)
  - Alternatives:
    - HFSM
    - Behavioral Trees
- unified handling of all levels;  
blurs classic distinction between  
hi-level / low-level goals
- also blur classic distinction between  
sensing / thinking / acting

39

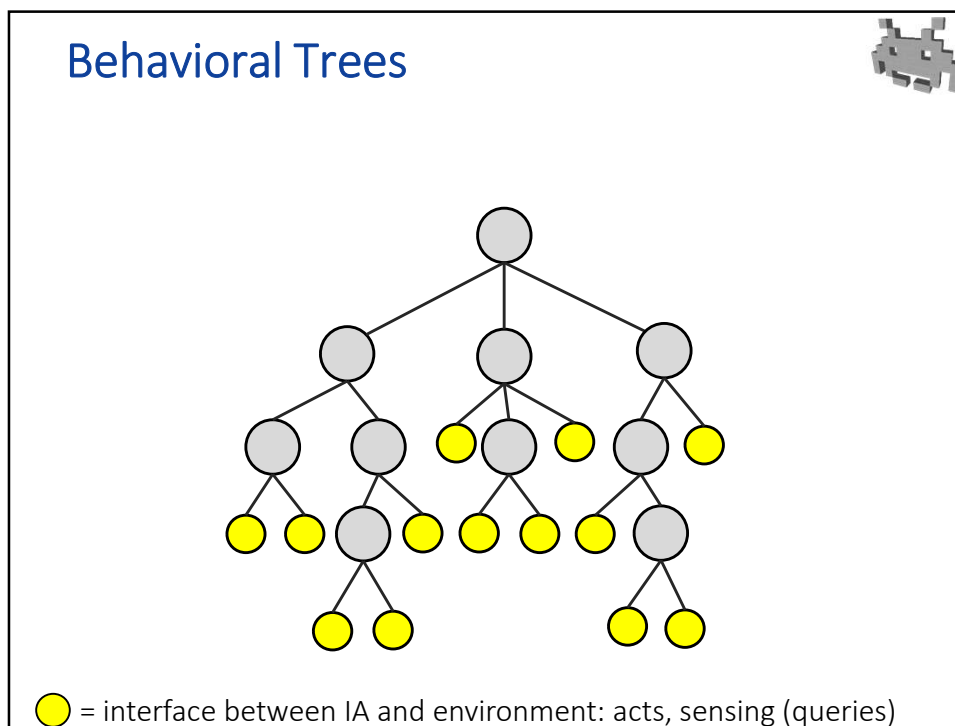
## HFSM Hierarchical Finite State Machines



40







43

### Behavioral Trees: nodes

- every node, when it has done *running*, can either have:
  - ★ failed
  - ✓ succeeded
- **leaves** are interaction with environment
  - **action** leaf:
    - animations, movements, sound, game logic...
    - Success: finished 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)
  - the distinction is not necessarily strict

44

### Behavioral Trees: nodes

- internal nodes: **sequence**

A diagram of a behavioral tree. At the top is a square internal node with a red starburst symbol. Below it are four circular leaf nodes. The first two leaf nodes have green checkmarks, and the third has a red starburst. A horizontal arrow points from the internal node to the right.

45

### Behavioral Trees: nodes

- internal nodes: **sequence**

A diagram of a behavioral tree. At the top is a square internal node with a green checkmark. Below it are four circular leaf nodes, each with a green checkmark. A horizontal arrow points from the internal node to the right.

46

### Behavioral Trees: nodes

- internal nodes: selector

A diagram of a behavioral tree. The root node is a grey circle with a green checkmark. It has four children, all white circles. The second and third children from the left have red starburst symbols. The third child also has a green checkmark.

47

### Behavioral Trees: nodes

- internal nodes: selector

A diagram of a behavioral tree. The root node is a grey circle with a red starburst symbol. It has four children, all white circles, each with a red starburst symbol.

48


### Behavioral Trees: nodes

- internal nodes: **inverter**

Only child

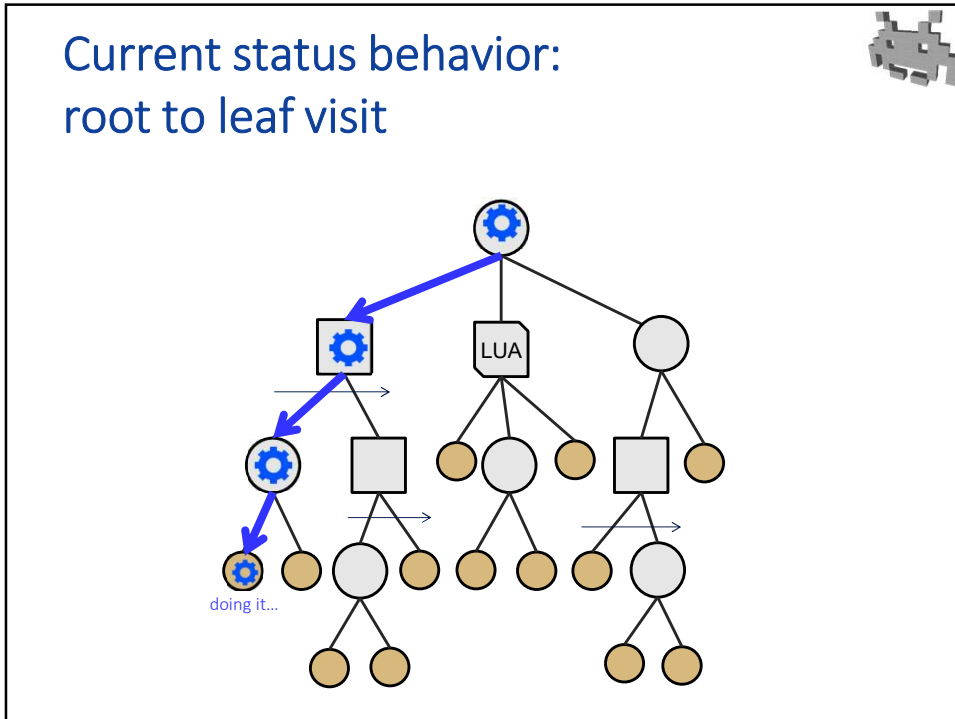
49

### Behavioral Trees: nodes

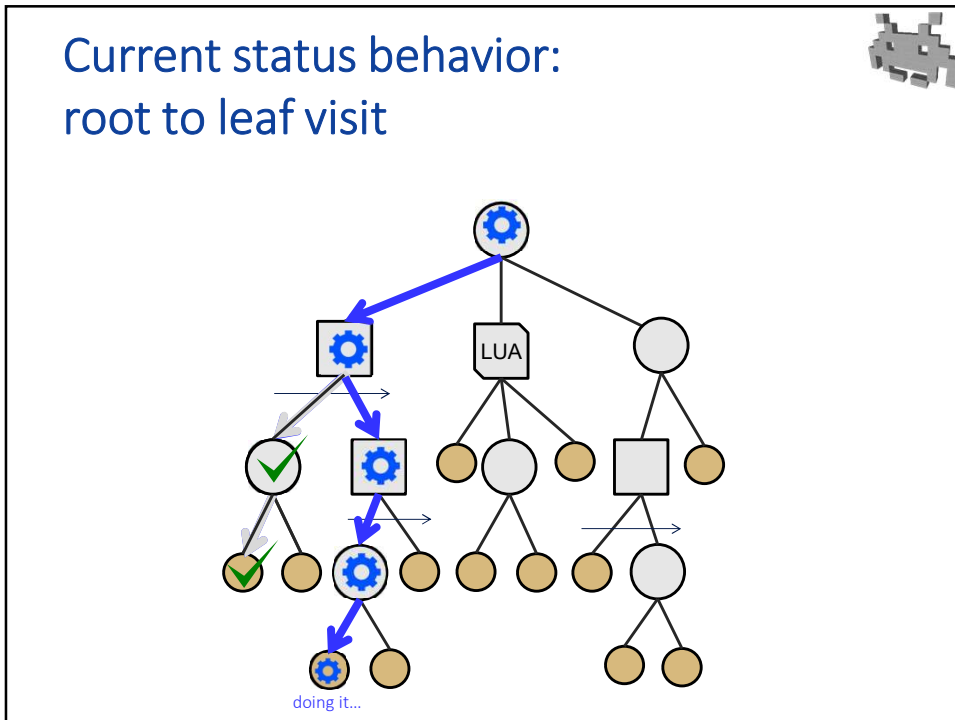
- or, nodes can be programmed arbitrarily in scripted procedure) (in , C#, JS...)
  - run children, as calls
  - fail or succeed, as returned value

BT as a framework to organize / reuse scripts

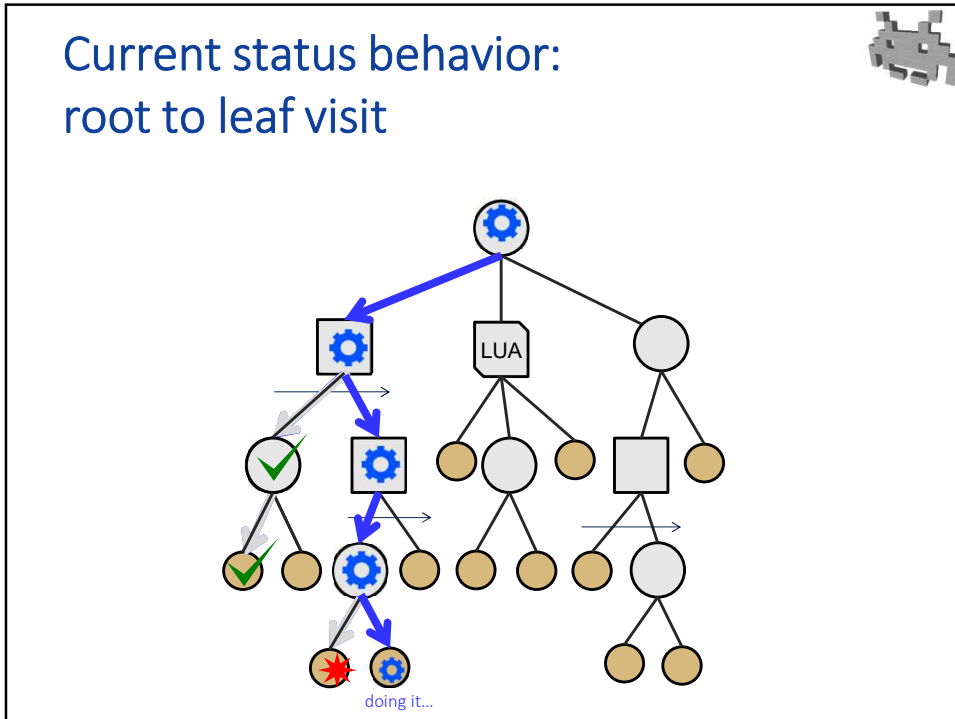
50



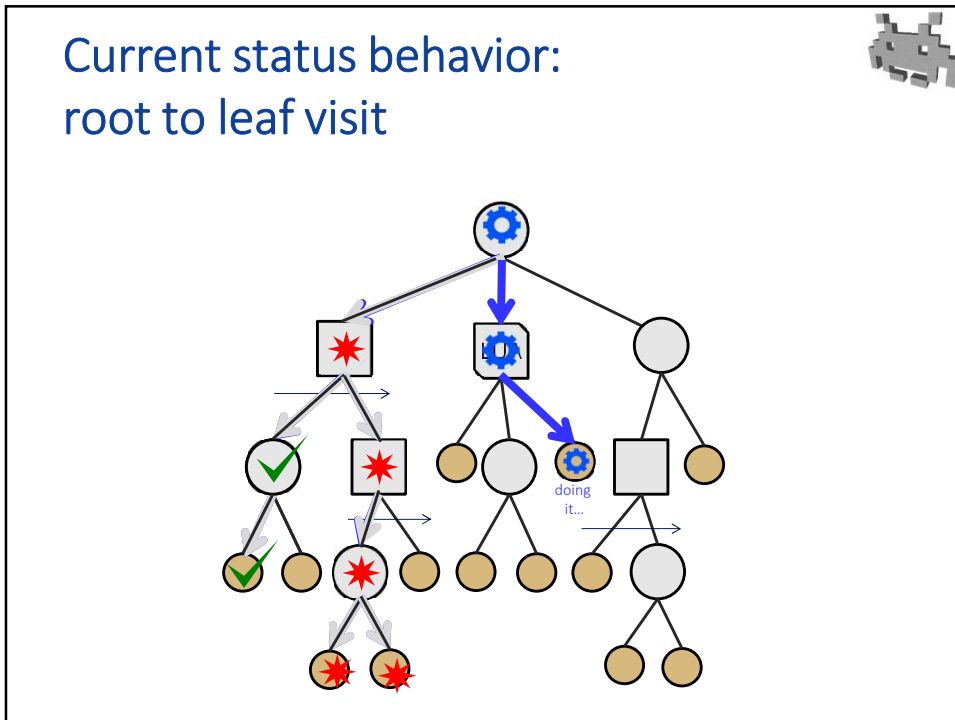
51



52



53



54

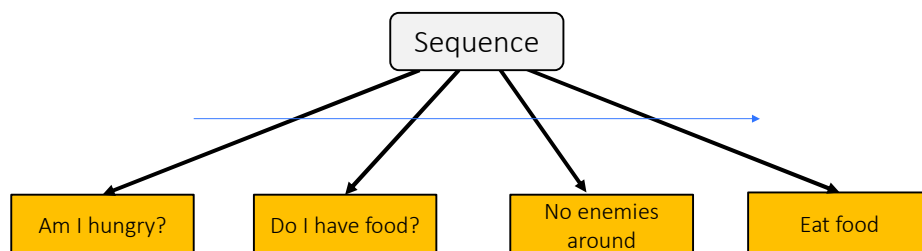
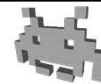
## Behavior trees: notes



- Each node can be:
  - ✖ failed
  - ✔ success
  - ⚙ in progress
    - (or still unvisited)
- Current IA-mind status:  
root-to-leaf path of ⚙ nodes
  - Low depth nodes: current high-level objectives
  - High depth nodes: current low-level objectives
  - Leaf of the path: current action / sensing action

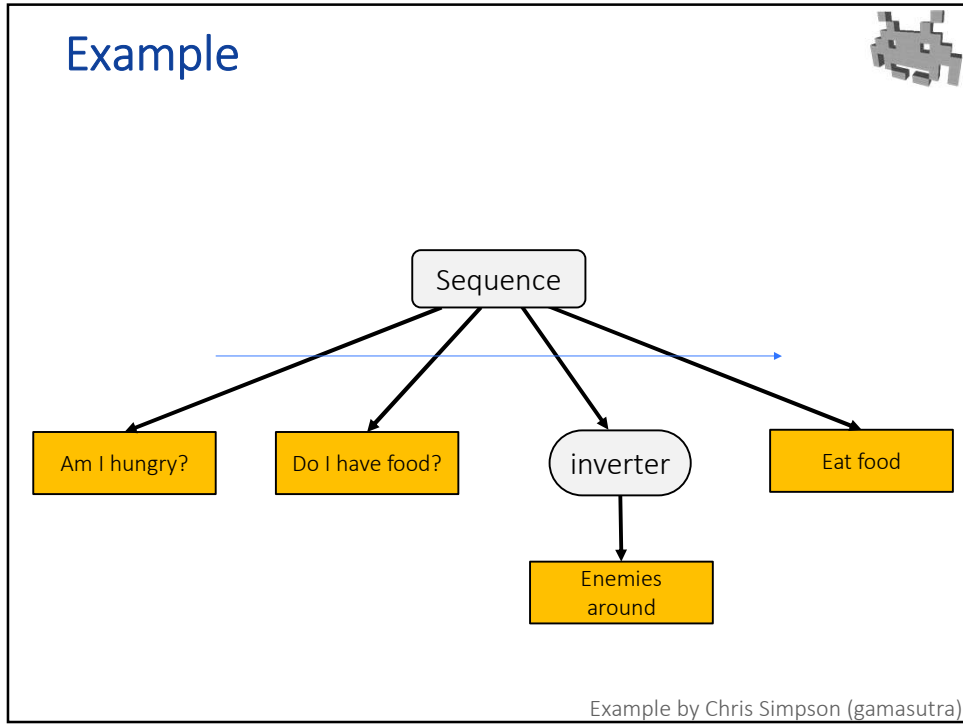
55

## Example

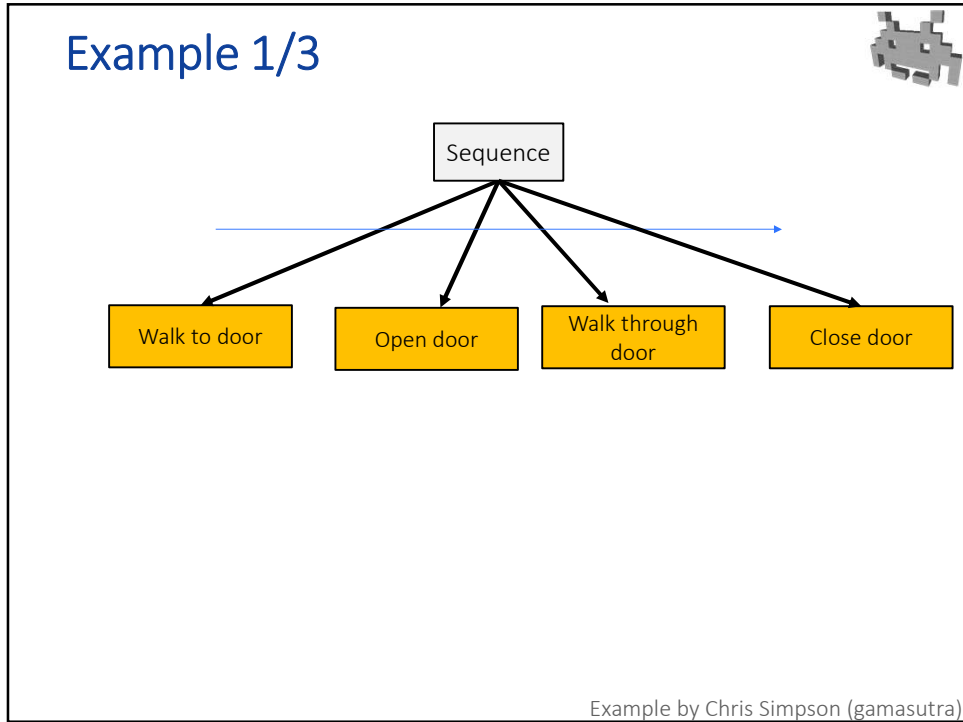


Example by Chris Simpson (gamasutra)

56

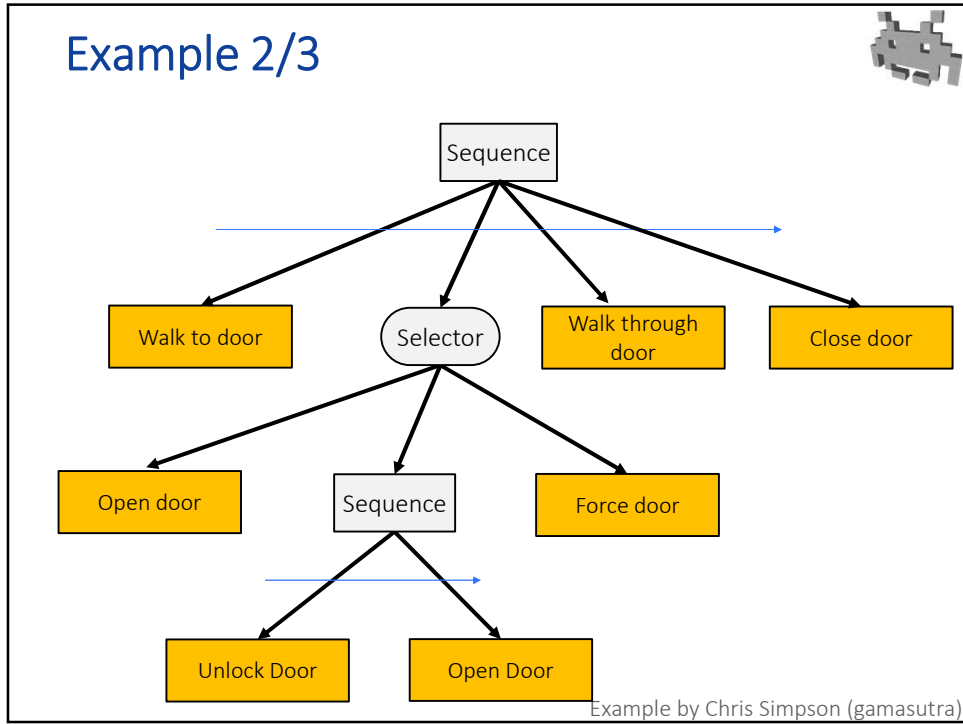


57

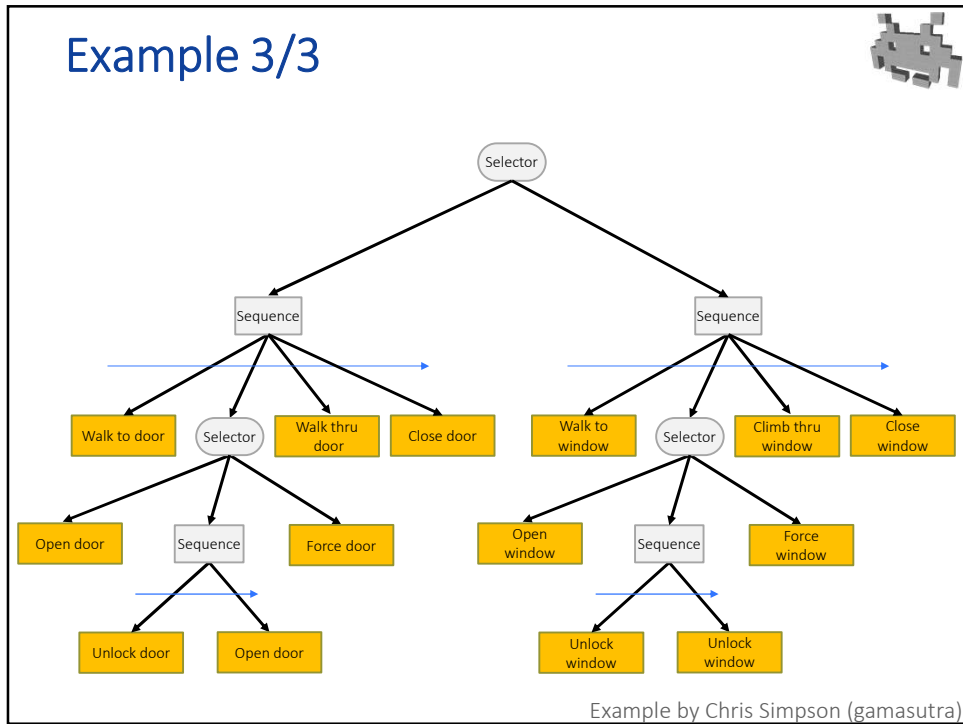


58





59



60

### Example 2

The diagram is a state machine for a patrol AI. It starts with a **Selector** node that branches into two paths:

- Loop** (Loop first node -1 times):
  - Condition: **HasPatrolPath** (Has defined patrol path)
  - Condition: **[not] SeesPlayer**
  - Path: **Sequence**
    - SelectNextPatrolTarget** (Sets next patrol target to "PatrolTarget")
    - MoveToX** (Move to blackboard key "PatrolTarget")
    - Condition: **Success**
- Sequence**:
  - Condition: **SeesPlayer**
  - Condition: **IsAlertLevel** (Is alert level "None")
  - Path: **Wait** (wait 0.2-0.2 s)
  - Path: **SetAlertLevel** (Set alert level to "Panic")
  - Path: **Wait** (wait 2-2.5 s)

example from **Tendrill: Echo Received** by **cepnox** <https://forums.tigsource.com/index.php?topic=60709.0>

61

### 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!

← such as...

62

## 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 ballistic
    - to predict, use *analytical* physics:  $pos(t) = f(t)$
  - e.g. including "leading the target"
    - i.e. aim at where target *will* be at time of impact
- Avoidance / dodging
  - of an incoming bullet
- ...

```
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 );
```

repeat a few times  
(converges really fast)

63

## Often easier to think in local object space of the IA

World space

$T$

$T^{-1}$

Agent object space

64

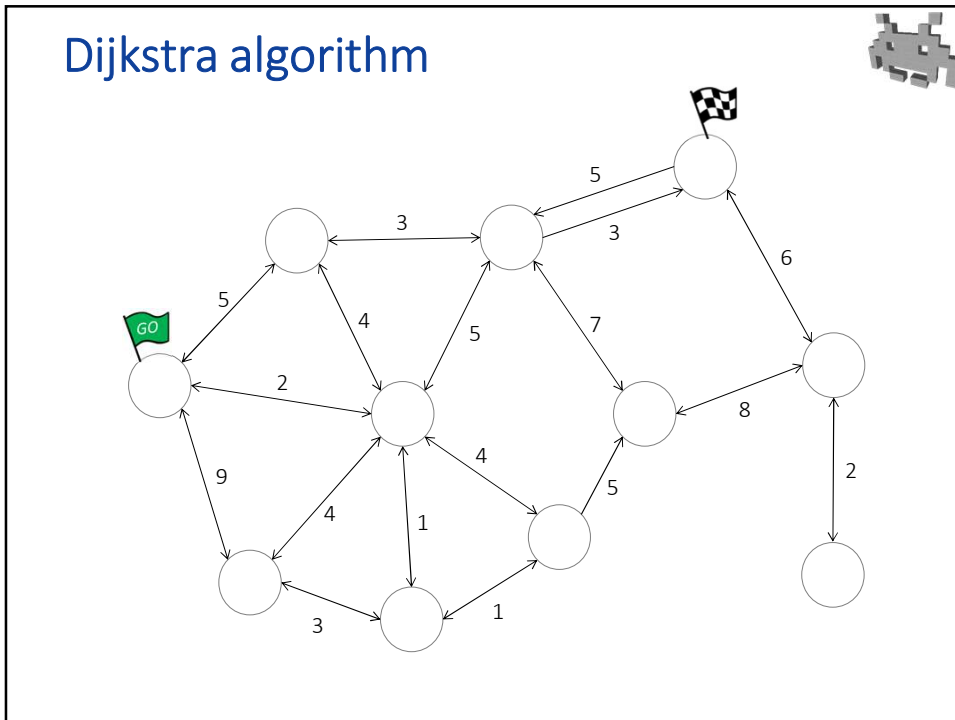
## Common lowest level tasks 2/2: Path finding

- Path finding
  - Dijkstra's algorithm
  - A\* search

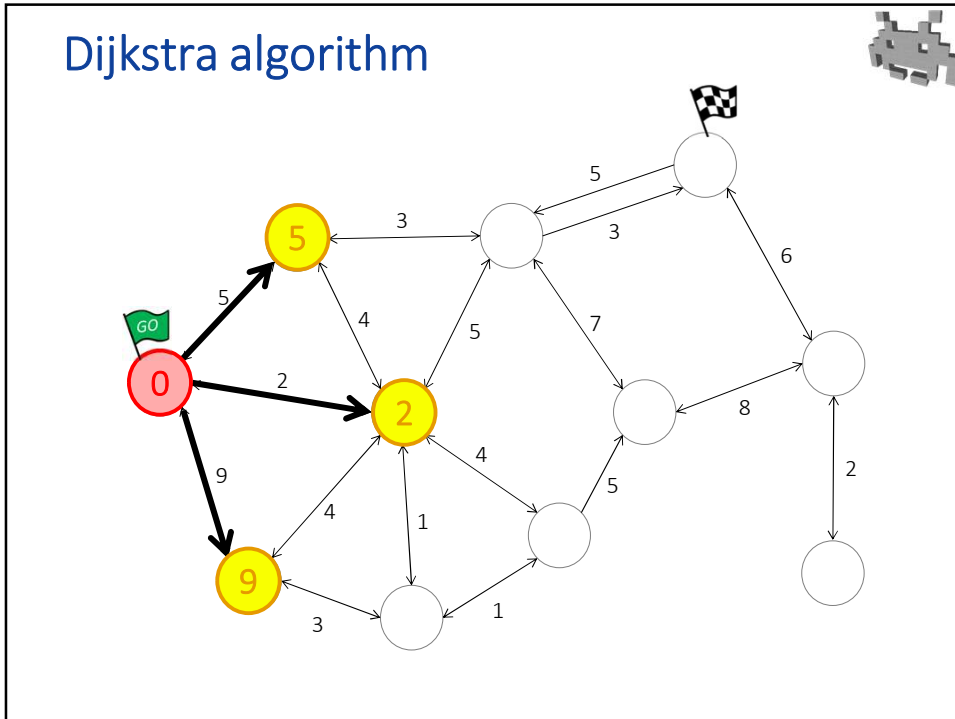


The diagram shows a 2D grid world with a black obstacle in the center. A small robot icon is located in the top right corner. The grid is composed of white circles representing nodes, with some circles missing where the obstacle is.

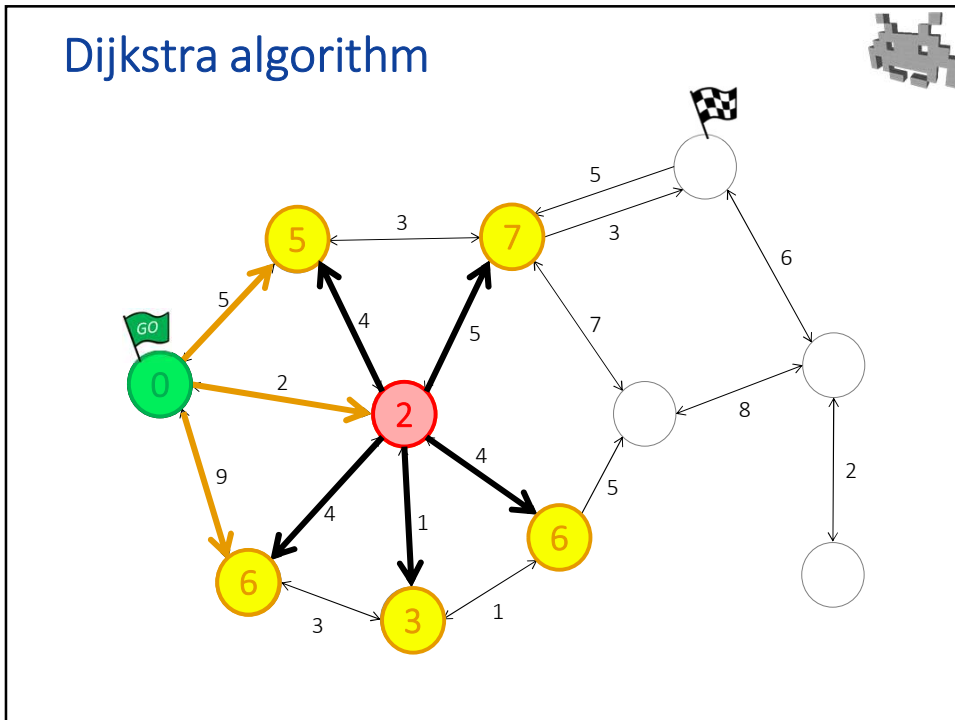
65



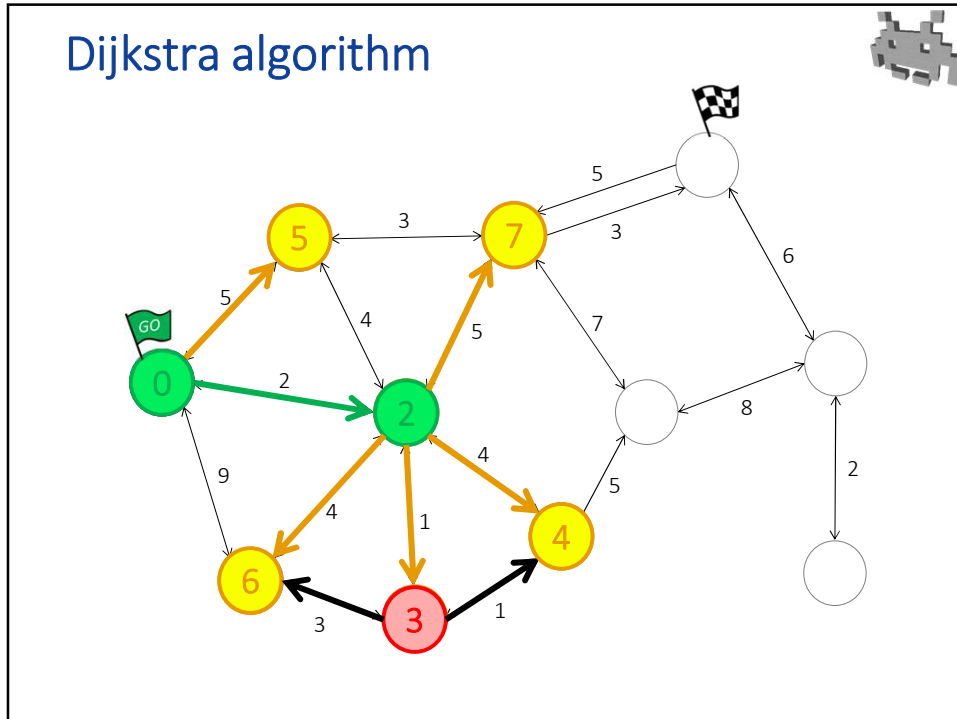
67



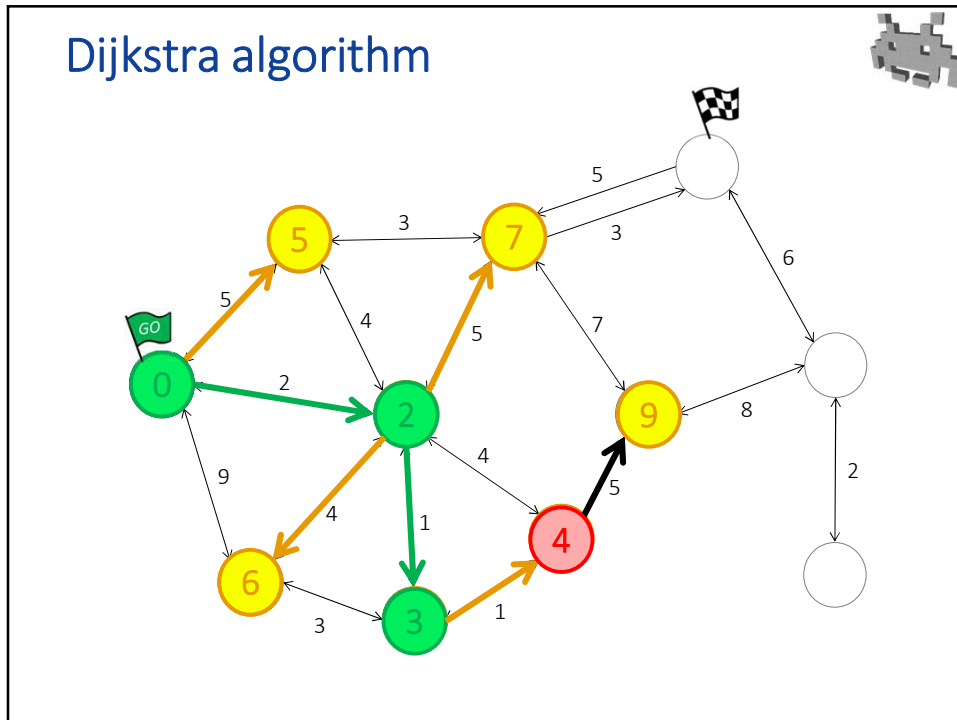
68



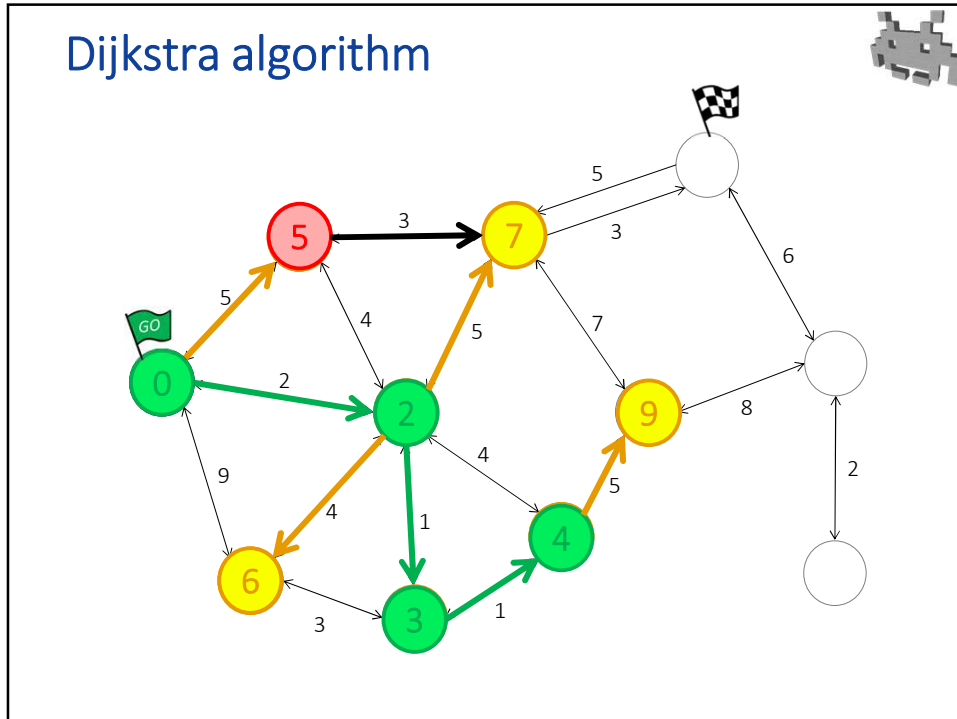
69



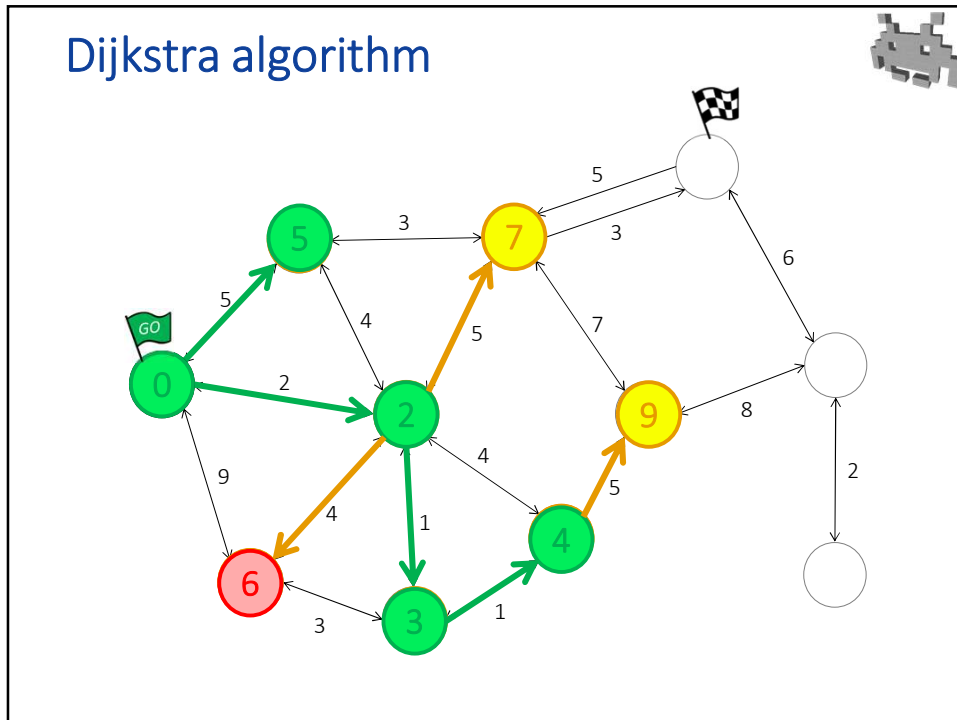
70



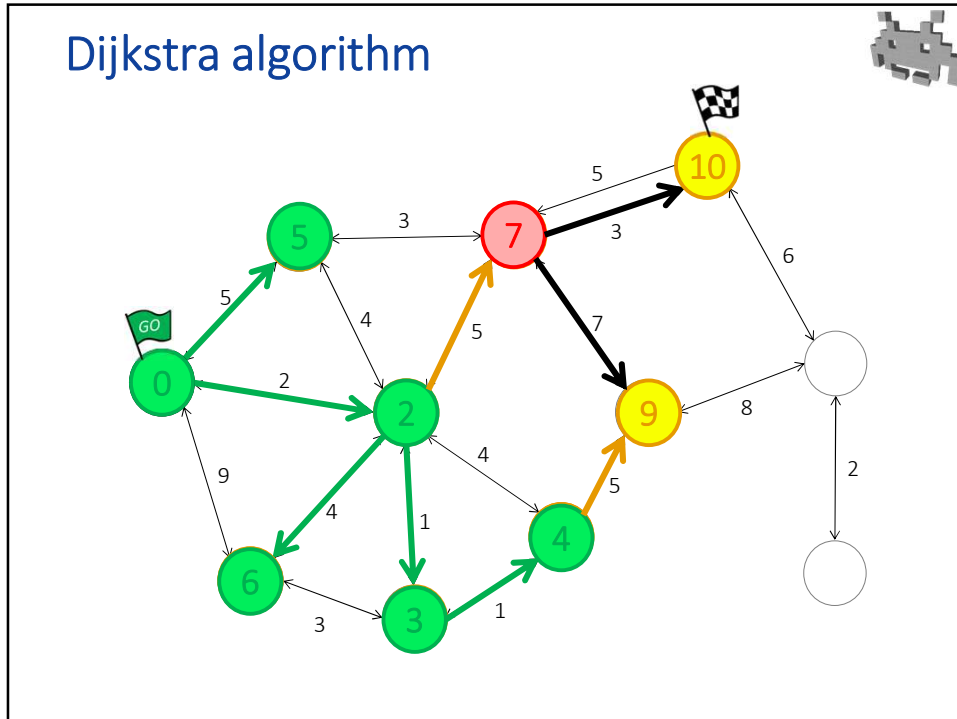
71



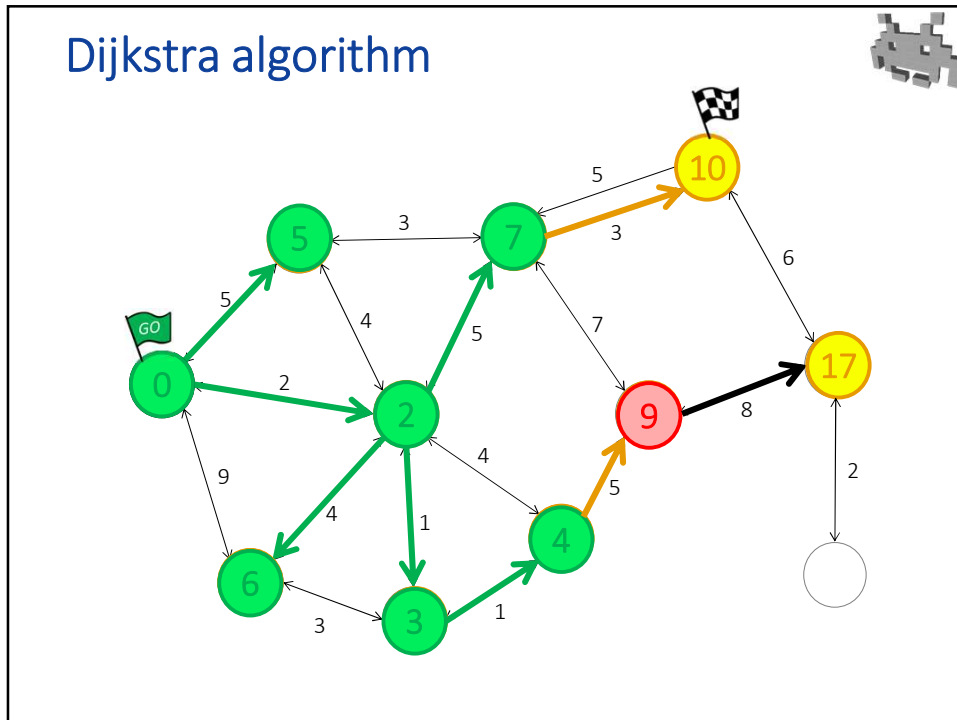
72



73

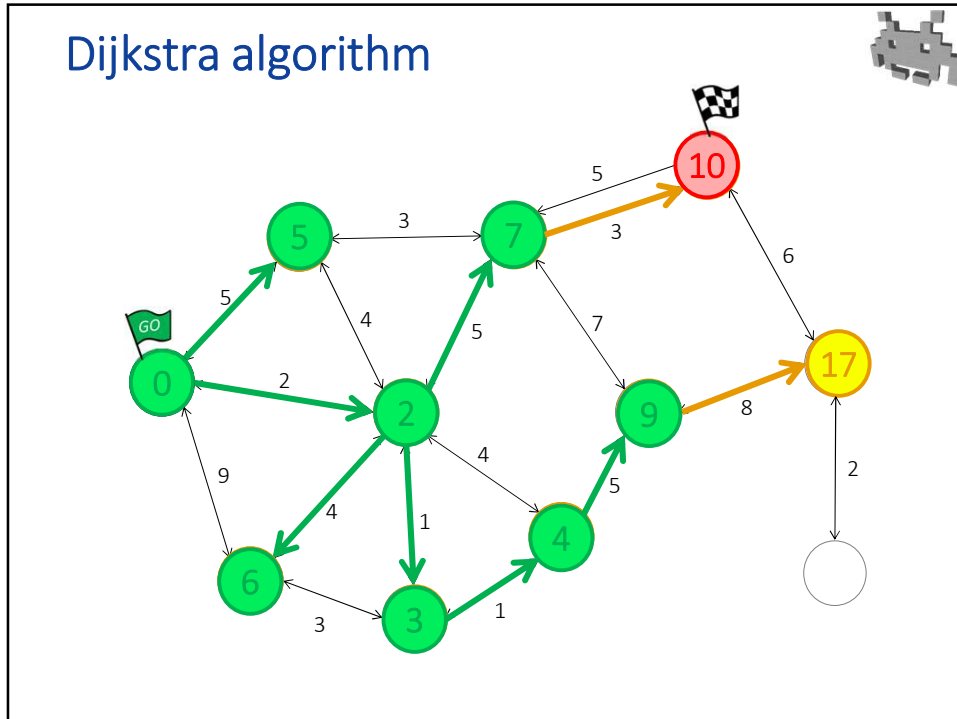


75

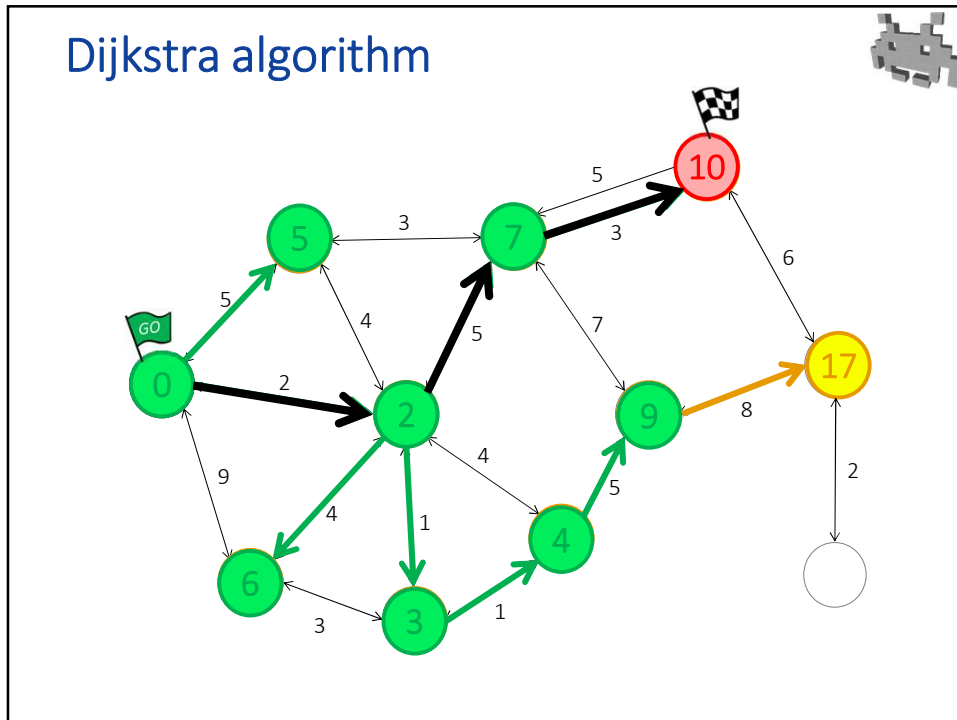


76

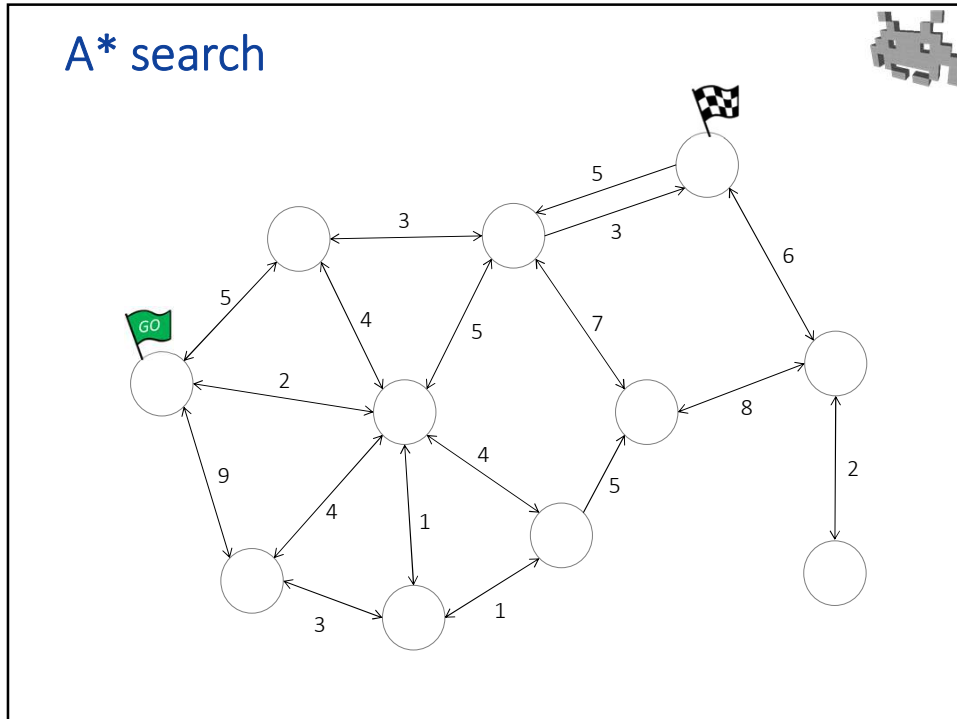




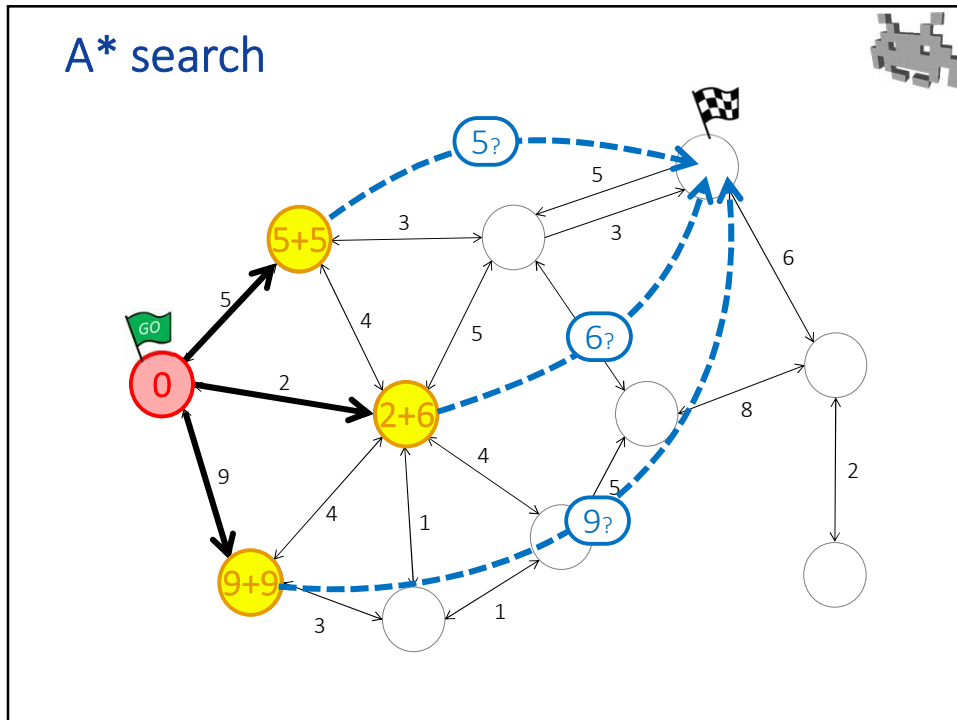
77



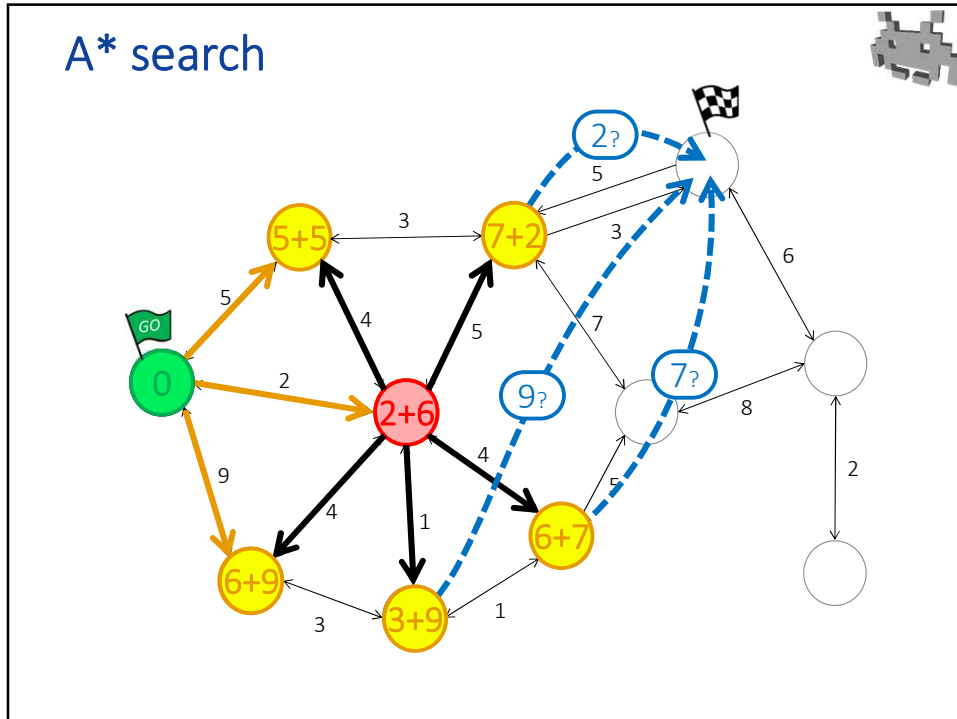
78



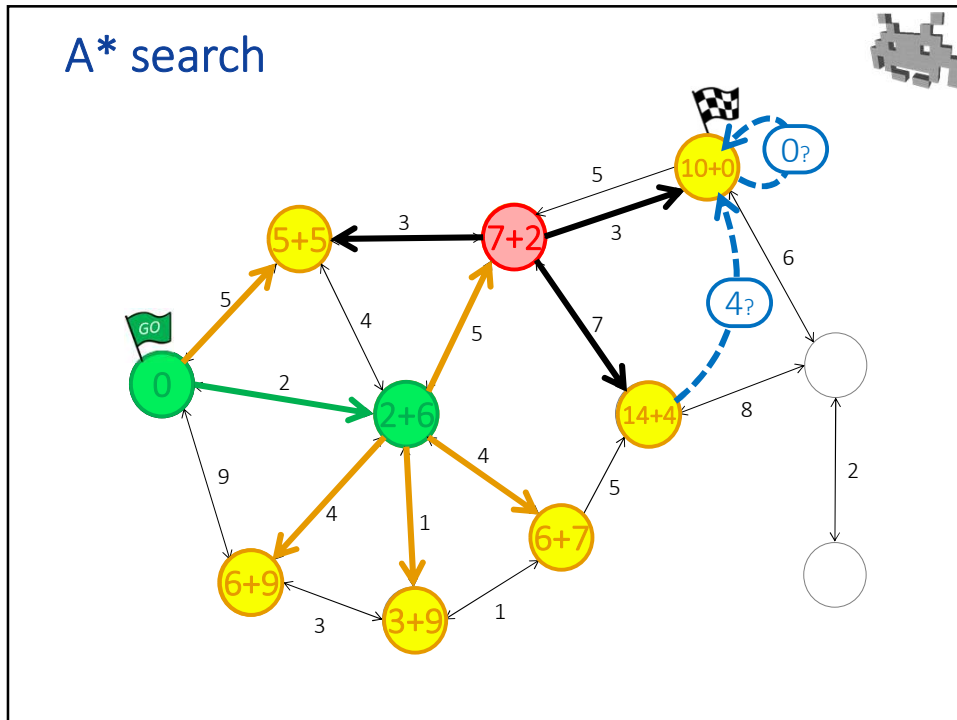
79



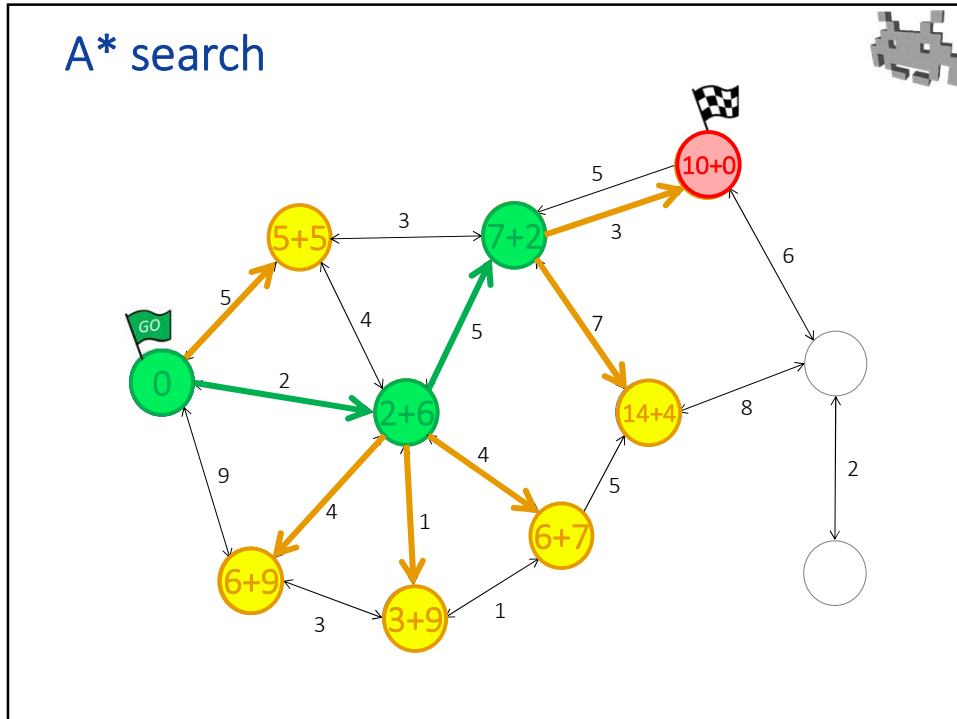
80



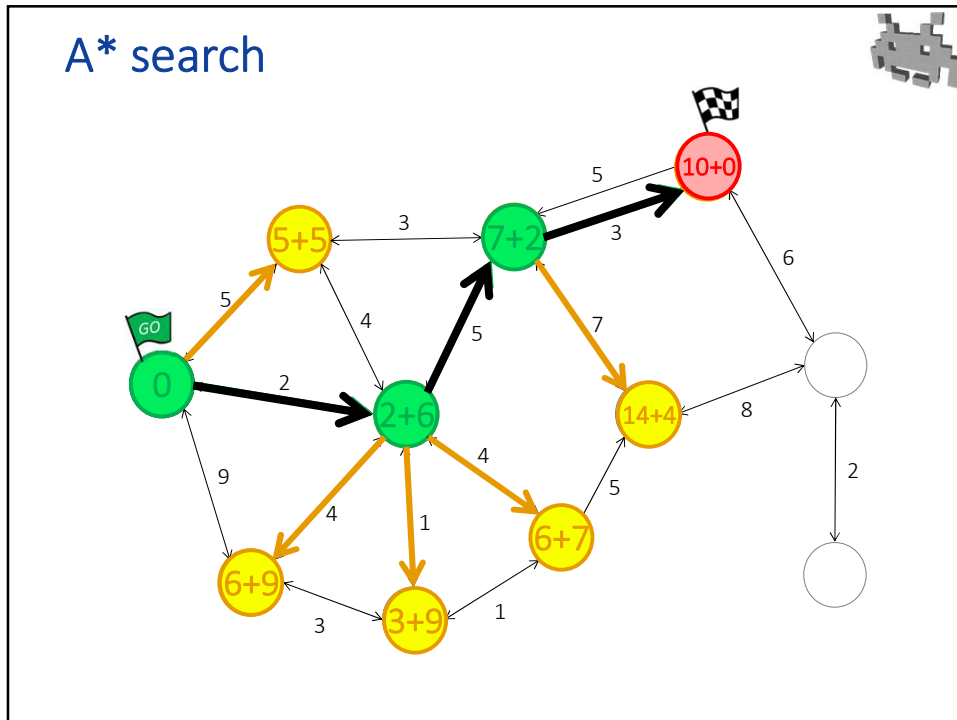
81



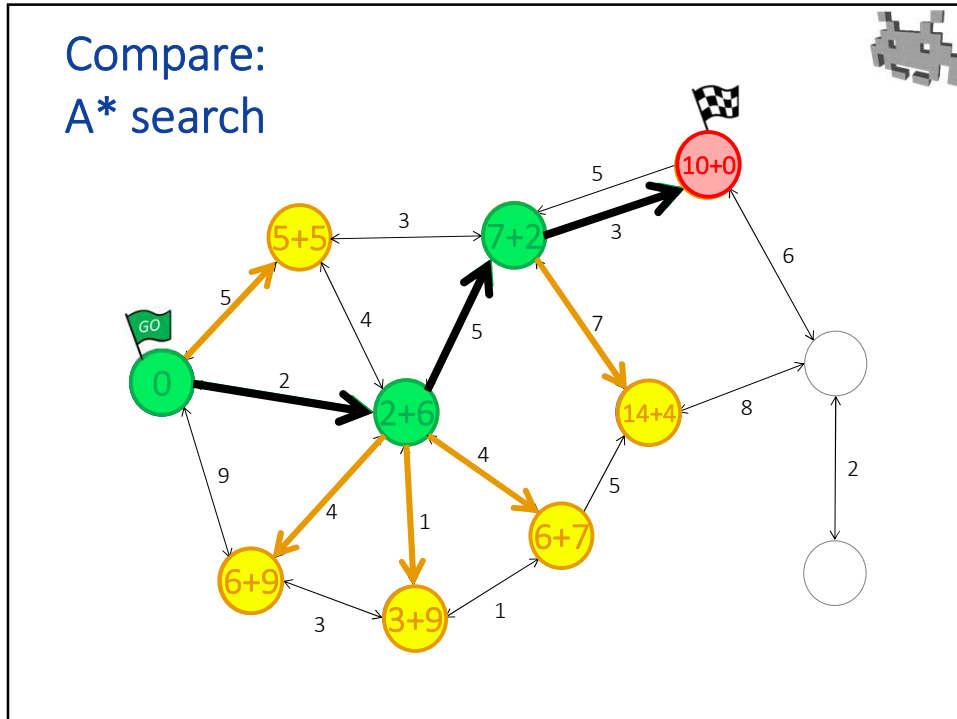
82



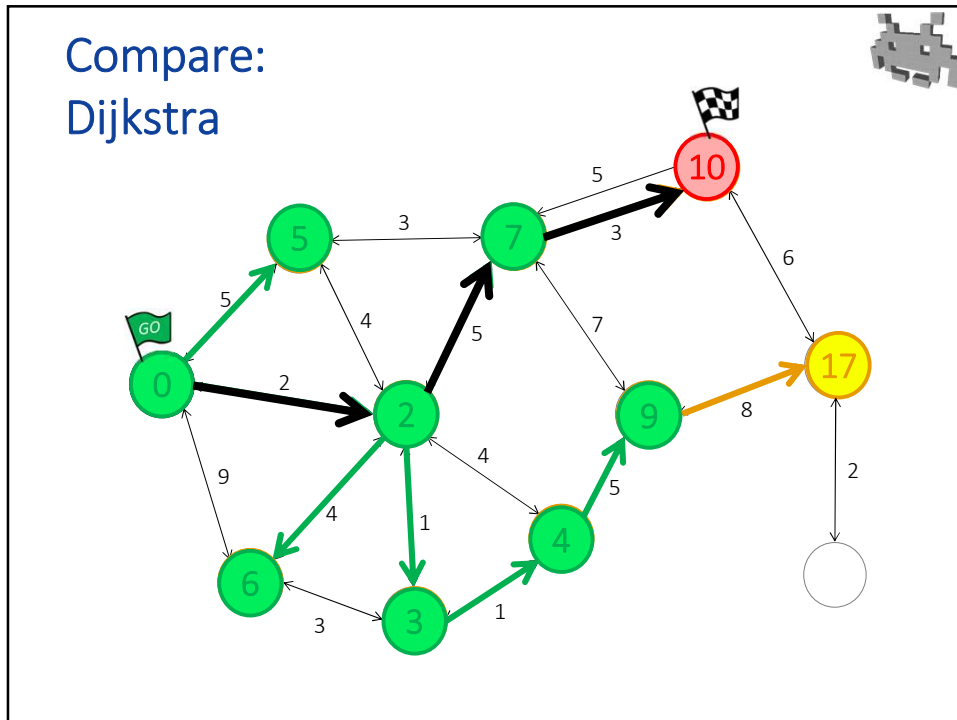
83



84



85



86

## Input of Dijkstra algorithm: notes.



- graph (nodes, arches)
  - nodes = locations where the Interactive Agent can be
  - arches = paths to go from node A to node B, for example...
    - ...a straight path from A to B (to be run / walked)
    - ...a potential **jump** reaching B from A
    - ...a **drop down** from A to B (note: arches are not necessarily bidirectional!)
- a (non-negative!) **cost**, associated to each arch
  - e.g., estimated time needed to go from A to B
  - in general, this reflects the 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)
  - Destination nodes can be multiple

88


## Dijkstra algorithm: notes.




- Any nodes is visited / processed only once
  - Or zero times! Not all nodes are visited
- The algorithm requires to keep track of a set of “active” nodes
  - (in yellow, in the graph)
  - nodes are removed and added to this set
  - it is necessary to find the minimal element in this set
  - → ideal data structure for this : heap (priority queue)
- Output: path from Start node to Dest node
  - it’s guaranteed to be the minimal-cost path
  - the path with the minimum associate cost
  - also, the cost of this path
  - also, a minimum span tree of all visited nodes (results can be reused for all visited nodes)

89

## A\* algorithm: (“A-star”) notes




- Dijkstra not efficient enough
  - visits too many nodes
  - explores paths which are obviously wrong
  - it’s greedy, only guided only by **distance from Start**)
- “A\* search” is a variation. Main idea: smarten up! with an **estimate** of the remaining **distance to Dest**
  - function  $h(X)$  – with  $X$  being a node: returns an estimate of the *minimal* cost to go from  $x$  to Dest
    - $h$  is provided by the user
    - it must be: fast (constant time, possibly)
    - it 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 quickest the algorithm
    - e.g.: if  $h(X)$  is always 0 (technically, still correct): A\* does the same as Dijkstra
    - e.g.: perfect estimation (hypothetical case): A\* only explores nodes in optimal path

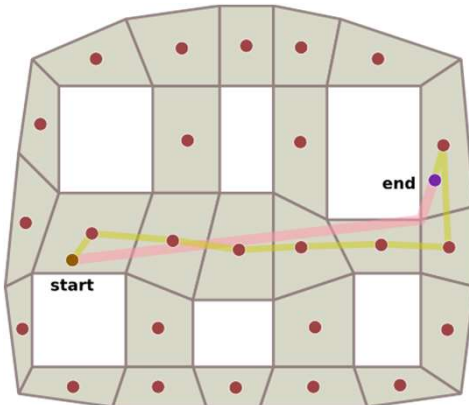


90

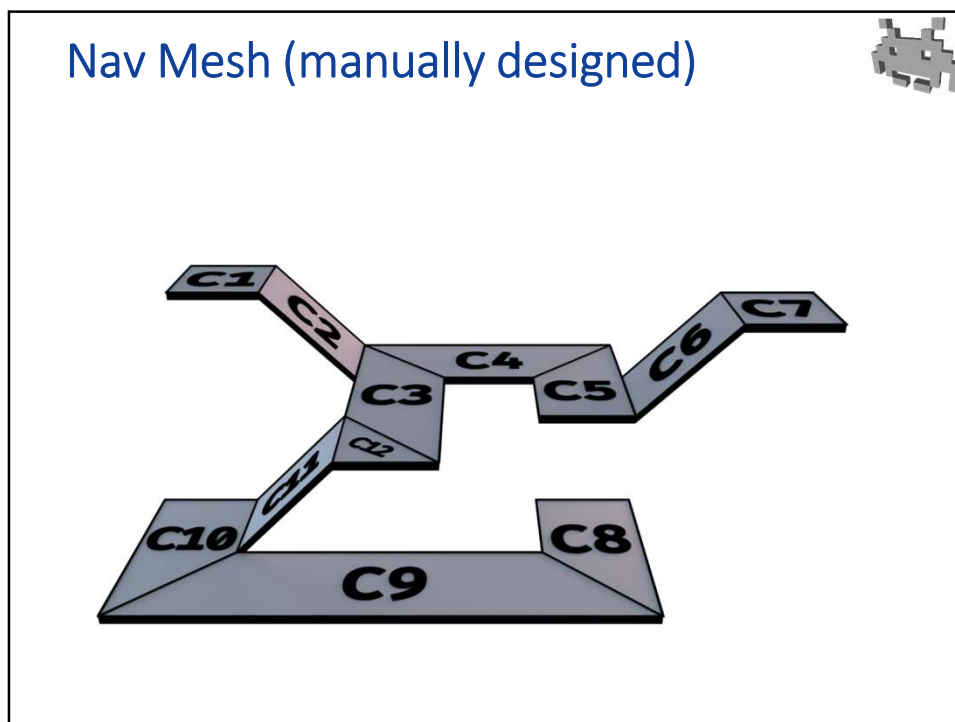
## Which graph to use for A\* / Dijkstra in a 3D game?



- 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)



91



92

### 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

93



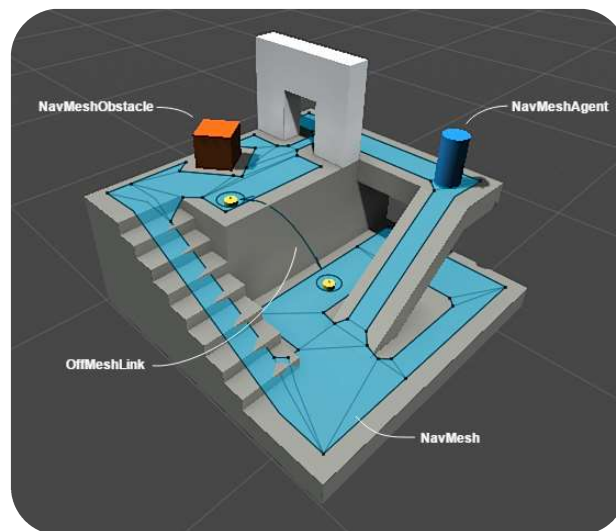
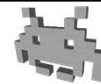
## Customizing A\* / Dijkstra



- 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

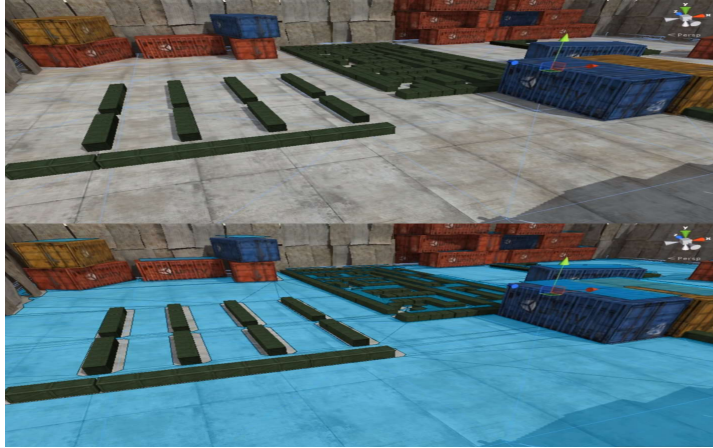
94

## Nav Mesh: Unity



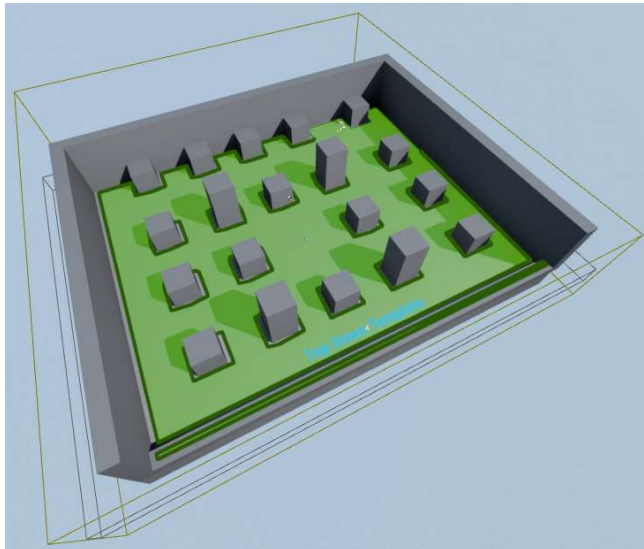
95

## Nav-Mesh baking: example in Unity



96

## Nav-Mesh baking: example in Unreal



97

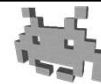
## Flocking algorithms



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

99

## Other mid-level goals for AI in 3D games



- Often, completely ad-hoc strategies:
  - E.g., in a car-driving racing game:
    - compute-and-bake (or, manually edit) the *optimal* path for each racing circuit e.g., as a b-spline curve or as a segmented curve
    - make NPC cars target the path position ahead of them (mid level), but avoid collisions (low level)
    - result: decently competent car-racer behavior

100

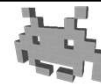
## AI support in a game engine: a summary



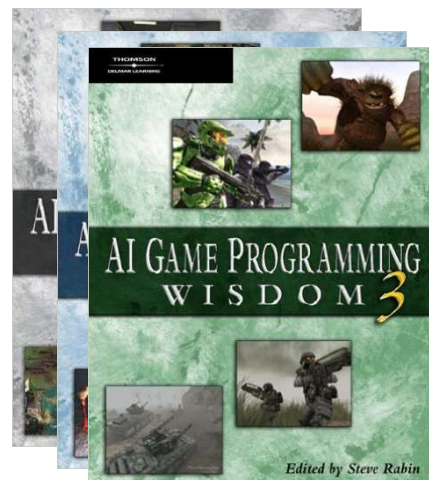
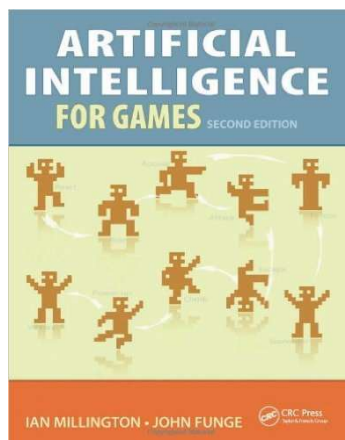
- **Assets** for (NPC) AI:
  - for *behavior modelling*:
    - **Scripts** (can well be the only one)
    - **FSM**
    - **HFSM**
    - **BT**
  - for *navigation*:
    - **nav-meshes** (aka **AI-meshes**)
  - for *sensing / queries*:
    - **hit-boxes**, **bounding volumes**, **spatial indexing**
    - 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 on a 3D environment

101

## To investigate further



- AI for VideoGames course!
- Books:



102