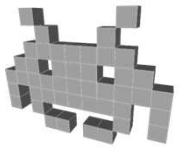
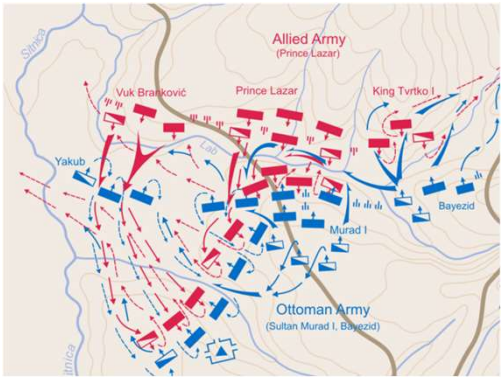
3D VideoGames
Università degli Studi di Milano



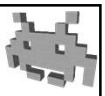
Interactive Agents for 3D Games

Marco Tarini



1

Course Plan



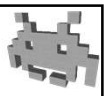
lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Models**
lec. 7: Game **Materials**
lec. 8: Game **Textures**
lec. 9: Game **3D Animations**
lec. 10: **3D Audio** for 3D Games
lec. 11: **Networking** for 3D Games
lec. 12: **Interactive Agents** 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».
For ML applications, see the course «Statistical methods for ML».

★ bridge lectures

2

AI in games: an ongoing revolution?



- Generative AI for…
  - Meshes
  - Including UV-maps, rigs, skinning
  - Material assets
  - Including textures (all kinds)
  - Levels
  - Code, scripts, logic
  - Sound effects
  - Soundtracks, scores
  - Skeletal animations
  - Dialogs, etc

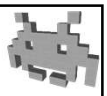
4

ML in games: one particularly 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 → skeleton
      - muscle = springs with AI-controlled strengths
    - a given task, e.g. (trivial to measure (scoring function))
      - 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 → skeletal animations
    - (minimizing used energy)
  - *How:*
    - genetic algorithms, Evolution strategies
    - physical simulation to score candidates

5

"AI" in games: examples of more traditional 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"

7

"AI" in games, the traditional meaning: NPC behavior

Widely different AIs for widely different "NPC"s!

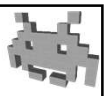
- A wild animal
- An (enemy) soldier
- A squad leader
- An (innocent) villager / bystander
- An individual in a crowd / flock / herd
- A racing car driver
- A spaceship pilot / gunner
- A companion / buddy
- An (enemy) commander
- A zombie
- A heat seeking missile
- A WWII ace pilot
- …

"flocking" algorithms
or "crowd simulation"

the AI player
in an RTS

8

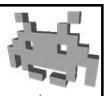
"AI" for NPC behavior: Interactive Agents (IA)



- 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

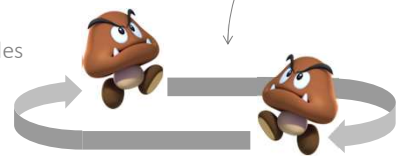
NPC behavior: designer perspective



NPC behavior is not *necessarily*
- "intelligent" / proficient ("")
- nor even complex! good at what it does

Rather, we may want NPC behavior to be (examples):
- intuitable / predictable
- learnable (by players) ← think boss battles
- understandable
- story-driven
- exploitable (interesting to exploit)
- realistic / believable ← for immersion; compare: a realistic solider (can panic), an optimal one (cannot)
- similar to human-players' behavior ← to fake players in MMO games (e.g. disconnected ones)

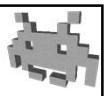
a common example of a very simple, but predicable / exploitable NPC behavior



Allowing game-designers to (examples):
- tune difficulty (by controlling NPC proficiency)
- elicit interesting strategies by the players
- make a given strategy rewarding
- give the illusions of competing against humans in MMO games …

11

AI in Games -vs- AI to solve Games



In a word:
entertainment, not *problem solving* !

Interested in AI to competently ***play*** games instead? look for:

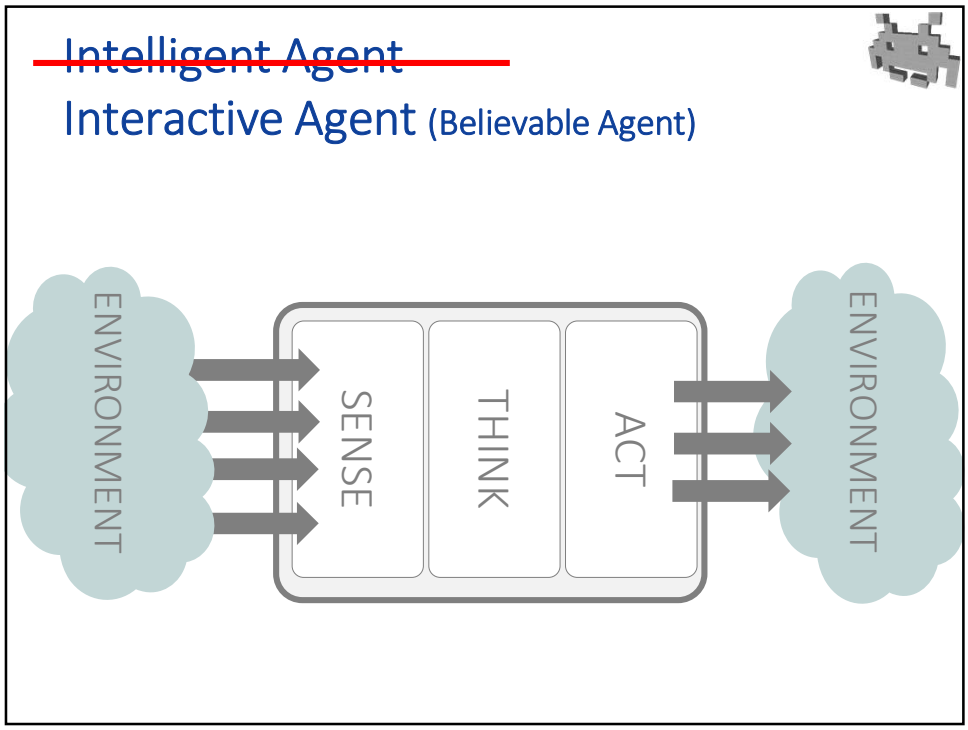
- min-max algorithms (with pruning)
  - algorithms to solve complete knowledge, turn-based games
- Nash equilibrium (from Game Theory)
  - a concept to address non cooperative games
- Machine Learning applied to generation of "actions"
  - see later!

12

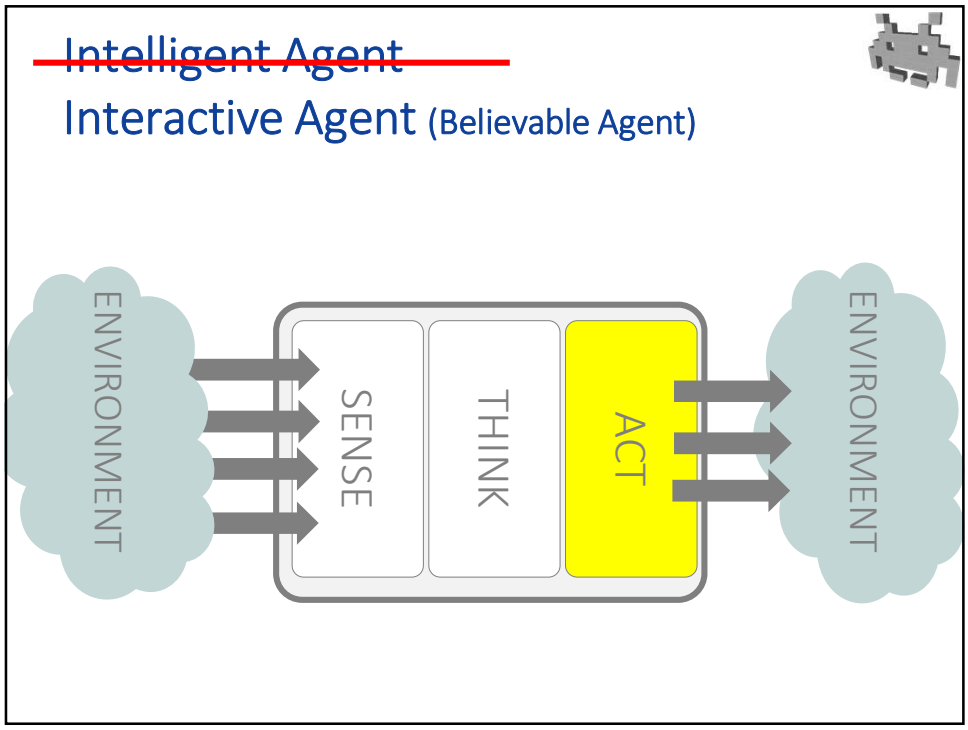
~~Intelligent Agent~~

Interactive Agent (Believable Agent)

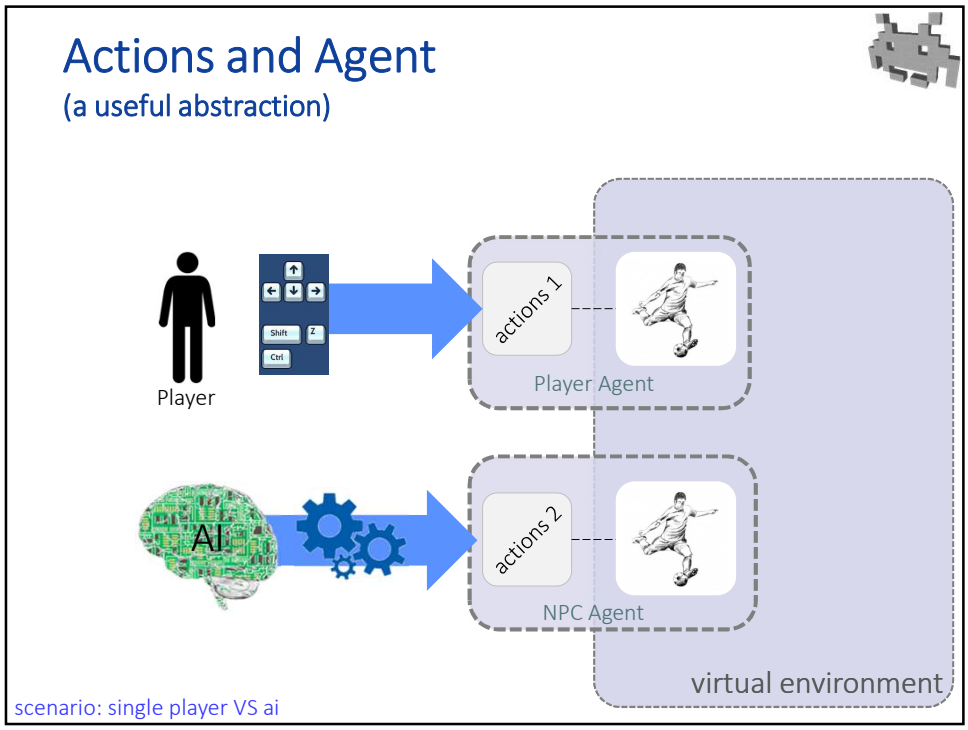
13


~~Intelligent Agent~~
Interactive Agent (Believable Agent)
ENVIRONMENT
SENSE
THINK
ACT
ENVIRONMENT

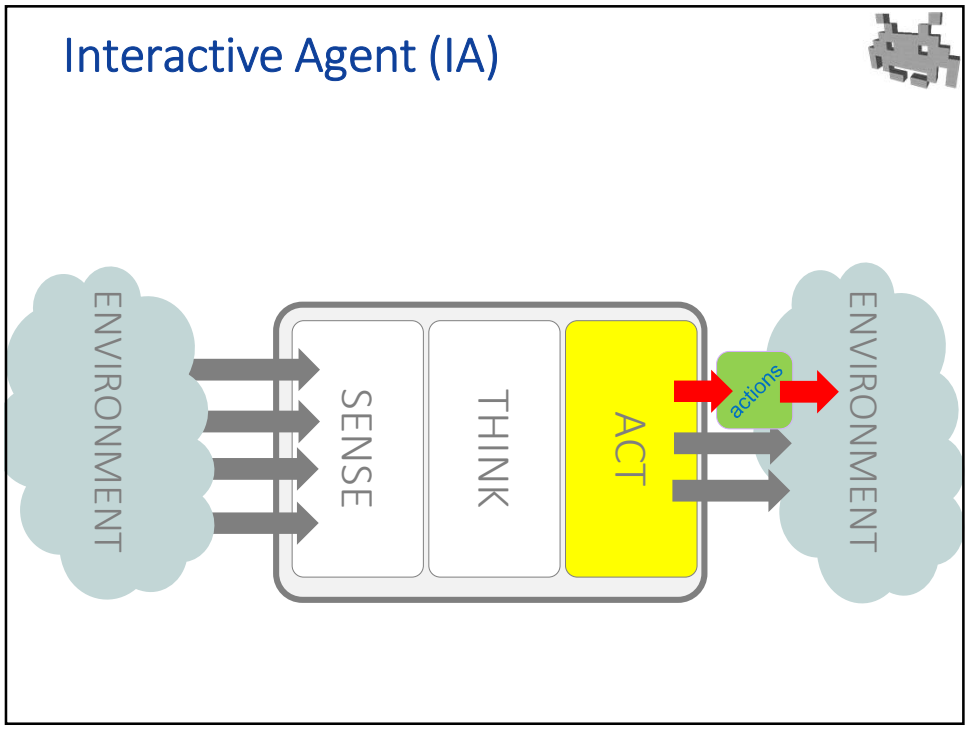
14


~~Intelligent Agent~~
Interactive Agent (Believable Agent)
ENVIRONMENT
SENSE
THINK
ACT
ENVIRONMENT

15

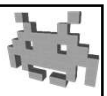

Actions and Agent
(a useful abstraction)
Player
actions 1
Player Agent
AI
actions 2
NPC Agent
virtual environment
scenario: single player VS ai

20


Interactive Agent (IA)
ENVIRONMENT
SENSE
THINK
ACT
actions
ENVIRONMENT

22

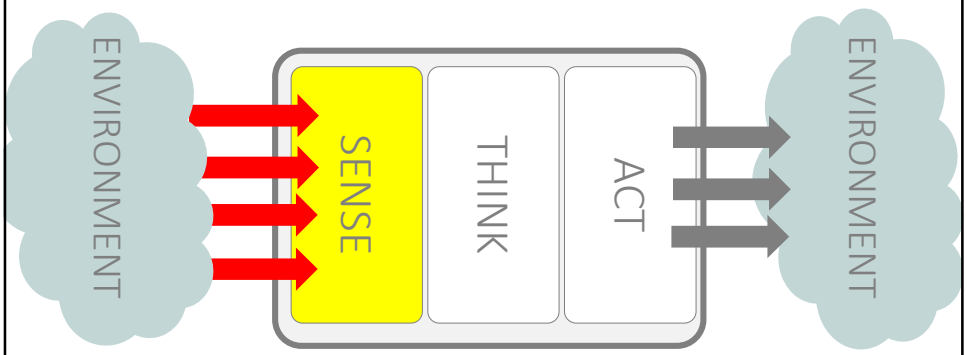
Acts: in robotics, “actuators”. In 3D games? Examples…



- Produce “actions”
  - associated to the NPC character
  - a non-cheating, symmetric NPC (simulation of a human player)
- Trigger skeletal animations / facial blend-shapes
- Cause movements / displacements of NPC 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.

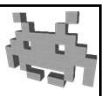
23

Interactive Agent (IA)



24

Sensing: in robotics, by “Sensors”. In 3D games?

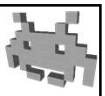


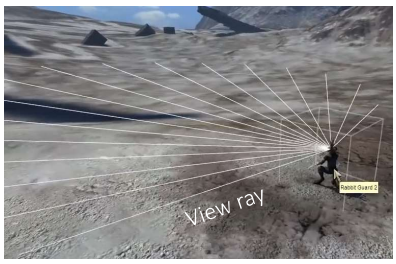
- Gather info (“percepts”)
  - which will be used for the “think” phase
  - NB: this info must often persist in the “mind” of the agent!
    - more abut this in the next phase
- Performed at regular intervals, or “on demand” (by the AI)
- Simulating senses in a 3D world…
  - Sight
    - ray-casting (uses colliders, specifically ray-VS-collider collision)
    - synthetize then analyze probe renderings! (accurate, expensive)
    - see math exercises: “testing if a position is in the cone of vision of NPC”
  - 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)

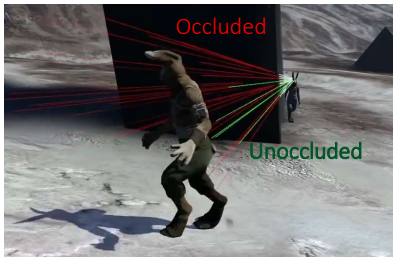
from the scene graph

25

Simulating senses in a 3D environment



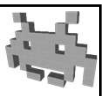




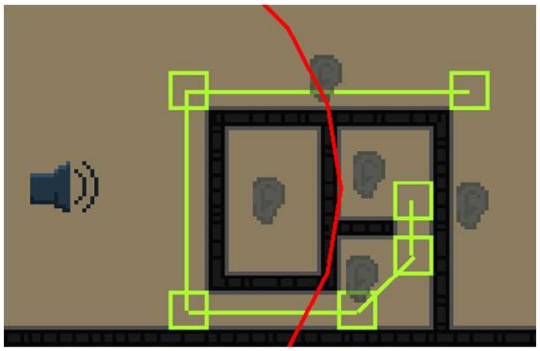
Sight Hearing

26

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



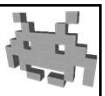
- Pathfinding for echoes simulation

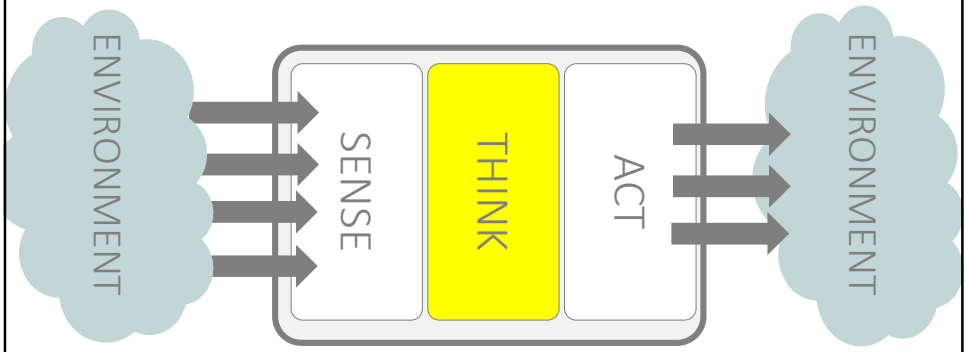


example from **Tendril: Echo Received** by **cepnox** https://forums.tigsource.com/index.php?topic=60709.0

27

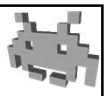
Interactive Agent (IA)





28

Thinking phase (aka Planning) Goals of the AI

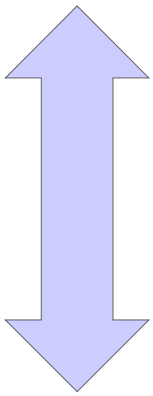


- Typically, Hierarchical Logic
  - Hi-level Decisions => Hi-Level Goals
    - update: not very often
  - …
  - Lower-level Goals
    - update: more often
  - …
  - Lowest-level Goals
    - solving low level tasks
  - Acts!

29

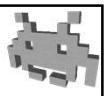
Thinking phase (aka Planning) Goals of the AI

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



30

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)

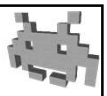
31

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

32

Example: guard



- Cascading goals
  - Hi-Level Goal — I'm "*Patrolling*"
  - Low-Level Goal — I'm going to my 3rd *nav-point*
  - Lowest-level Goal — I'm passing through *here* (find route to it – navigation)
  - Acts — actual movements + "alerted-walk" animation

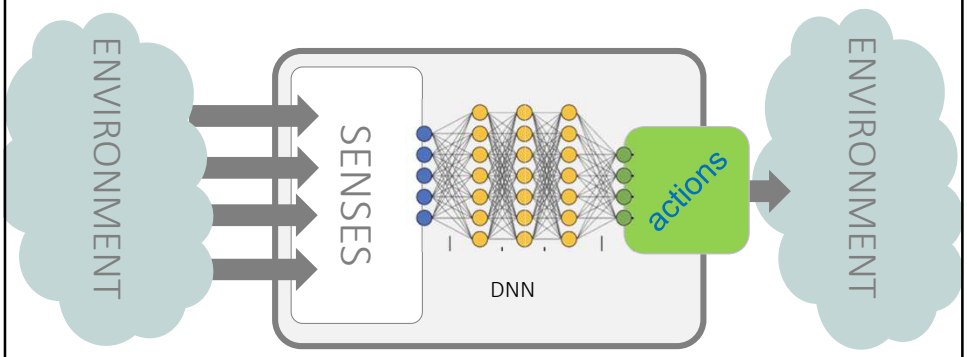
33

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

34

ML for the «think» phase (e.g. DNN) an emerging approach



35

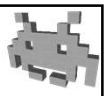
ML for the «think» phase (e.g. Deep Neural Networks)

- Sought Output: **actions** for the NPC
- **Supervised learning**
  - A classification task (assuming discrete actions e.g. keypresses): "given this input, should the fire/jump/left etc button be pressed?"
  - Technically a regression task (for continuous actions, e.g. mouse pos)
- **Data driven** (learn from examples):
  - Learning from detailed logs (frame-by-frame actions) captured from actual gameplay sessions by human players
  - This is the labelling in input to the learning process ("in this situation, human-players played this way")
  - Data can be easily collected in bulks (remember actions are light!)
- Objective: Convincingly **simulate human players**
  - in multiplayer games, with NP-NPC symmetry
  - Success can be measured by (simplified) "Turing test"

many successes reported in literature.

36

ML (e.g. Genetic Algorithms - GA) for the «think» phase



- Sought Output: **actions** for the NPC (again)
- **Genetic algorithms**
  - and other automatic optimization techniques
- Driven by a **reward function** :
  - Such as (depending on the gameplay):
    survived time, number of kills, score earned, flag captured, goals scored, (virtual) money earned, levels cleared, etc.
  - To be used as the Fitness Evaluation Function in GA
- Objective: make the NPC **competently play the game**
  - in multiplayer games, with NP-NPC symmetry
  - Success can be measured by measuring perfomances

37

ML for the «think» phase: input (e.g. fed to the DNN)

Both when learning, and when employing results at game execution

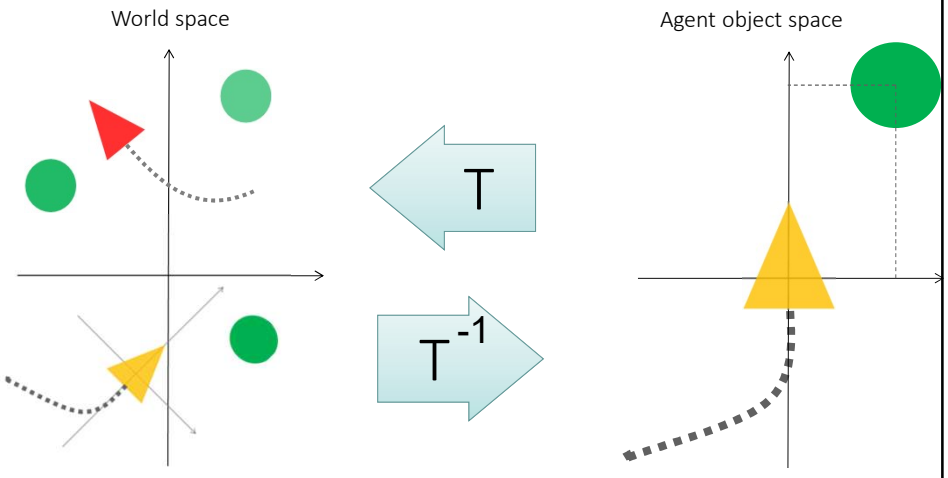
more honest but more difficult (deeper DNN, longer learning, etc)

cheating but more "learnable"

- A plain RGB rendering of the screen
  - Maybe artificially low-res for practical purposes
  - Maximal honesty: same info provided to players
  - Task becomes similar to self-driving machines with cameras
  - Basically, a gimmick (not used in practice)
- A *marked* RGB rendering of the screen
  - E.g. per pixel-coding for enemies, friends, walls, etc
  - Still a gimmick…
- Simulated percepts (visual, hearing, etc)
  - E.g. N view rays with fixed pos, a label for first hit of each
- The state of the game as known by AI
  - Positions of everything *that was observed*
  - E.g.: for each relevant object:
    last observed position / orientation, time *t* of this observation
  - *It helps if it is encoded in local NPC space! (see next slide)*
- The full state of the game
  - AI "cheats" by knowing everything
  - *It helps if it is encoded in local NPC space! (see next slide)*

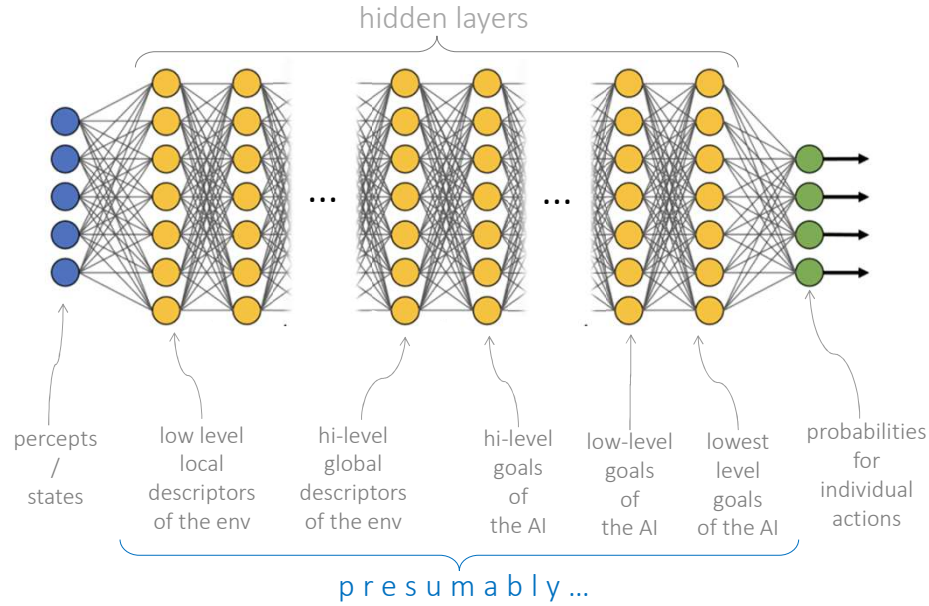
38

Often easier to think in local object space of the Agent (both for scripting AIs, and for ML)



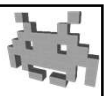
39

DNN at work (example)



40

ML for the «think» phase of IA



- A fast-advancing area of research
- Promises to cover well two cases
  - learn to mimic human players (from examples)
  - learn to be proficient at the game (autonomously)
  - Benefit: automatable

"AI training against itself"

- But let's remember "AI" (NPC behavior) serves a range of other objectives
  - we need tool to allow designers to craft NPC behavior
  - as an integral part of the game design
  - controlling proficiency is just one side of it (it's more relevant for low-level goals, for which ad-hoc approaches exist – as we are about to see)

==> let's go back to discuss traditional approaches…

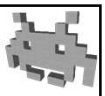
41

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

42

Thinking phase (aka Planning): about the lowest-level goals…



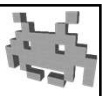
- Typically, Hierarchical Logic
  - Hi-level Decisions => Hi-Level Goals
    - update: not very often
  - …
  - Lower-level Goals
    - update: more often
  - …
  - Lowest-level Goals
    - solving low level tasks
  - Acts!

such as…

most instances are mini-tasks
we can solve with standard solutions
(supported by the game engine?),
such as…

43

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



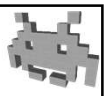
- Face towards something
  - tip: remember *atan2*
  - actions: turn left or right
- Aim a weapon
  - e.g. including ballistic
    - to predict, use *analytical* physics: pos(t) = f(t)
  - e.g. including "leading the target"
    - i.e. aim at where target *will* be at time of impact
- Avoidance / dodging
  - 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 );
```

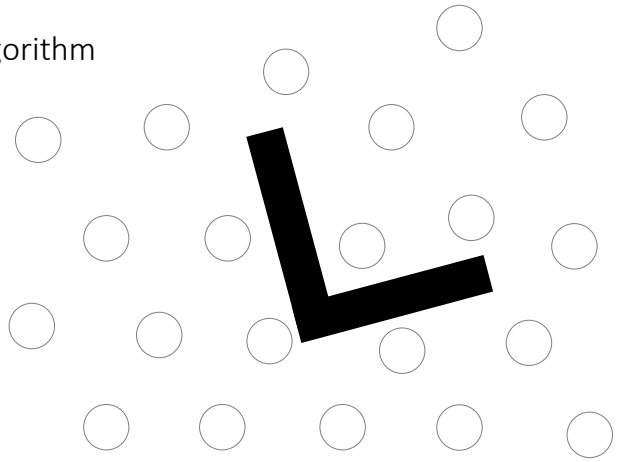
repeat a few times
(converges really fast)

44

Common lowest level tasks 2/2: Path finding



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



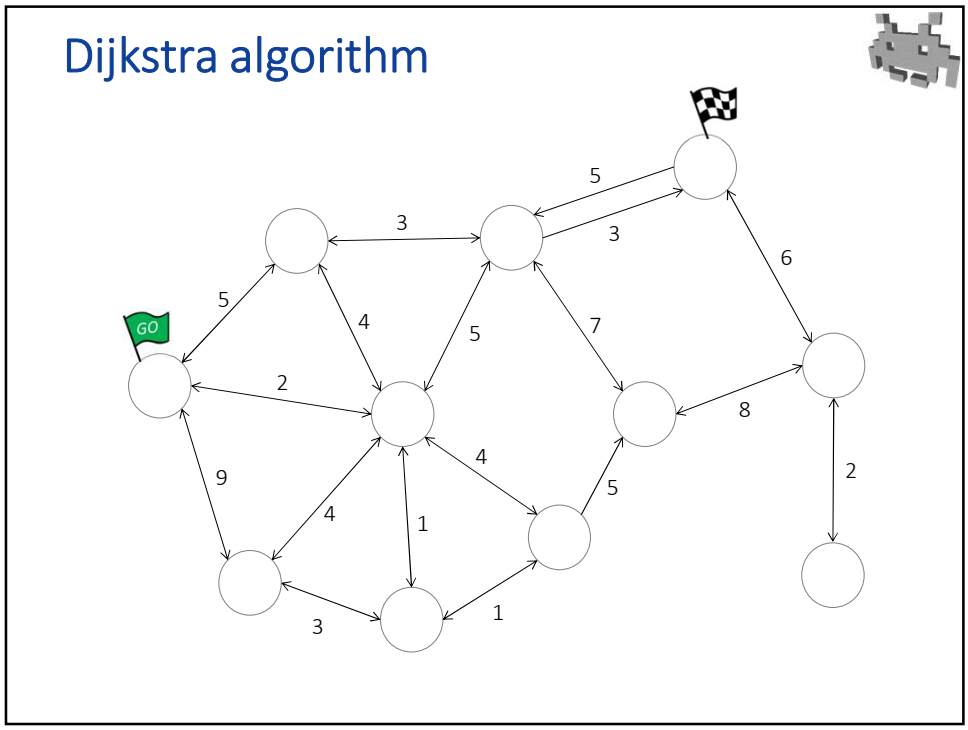
46

Dijkstra algorithm and A* search

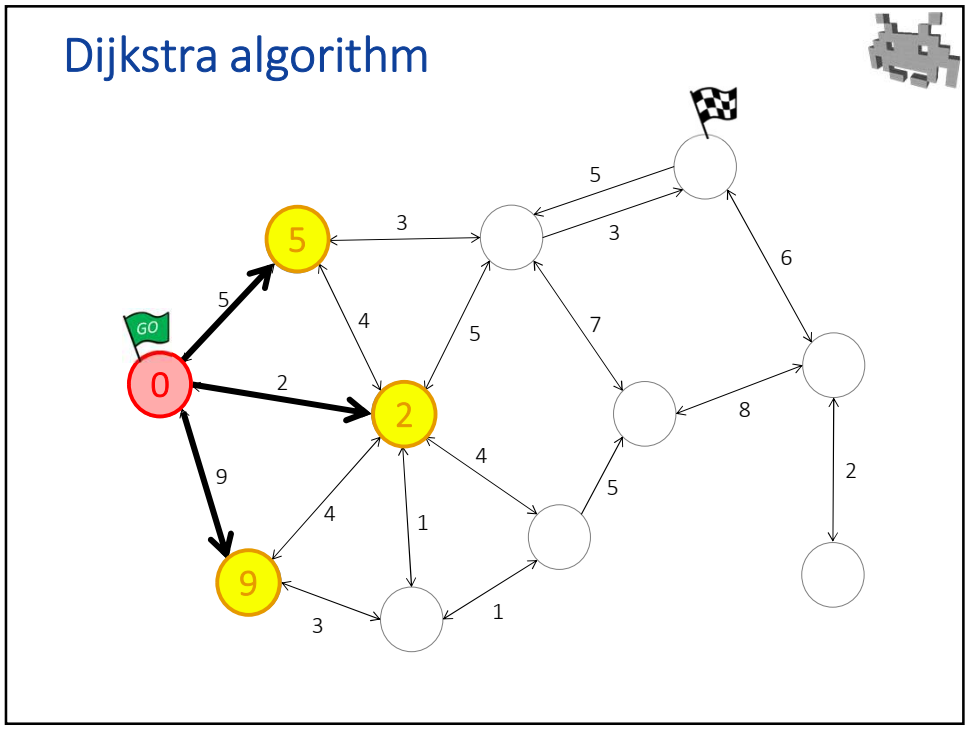
(part of the background, not of this course)

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

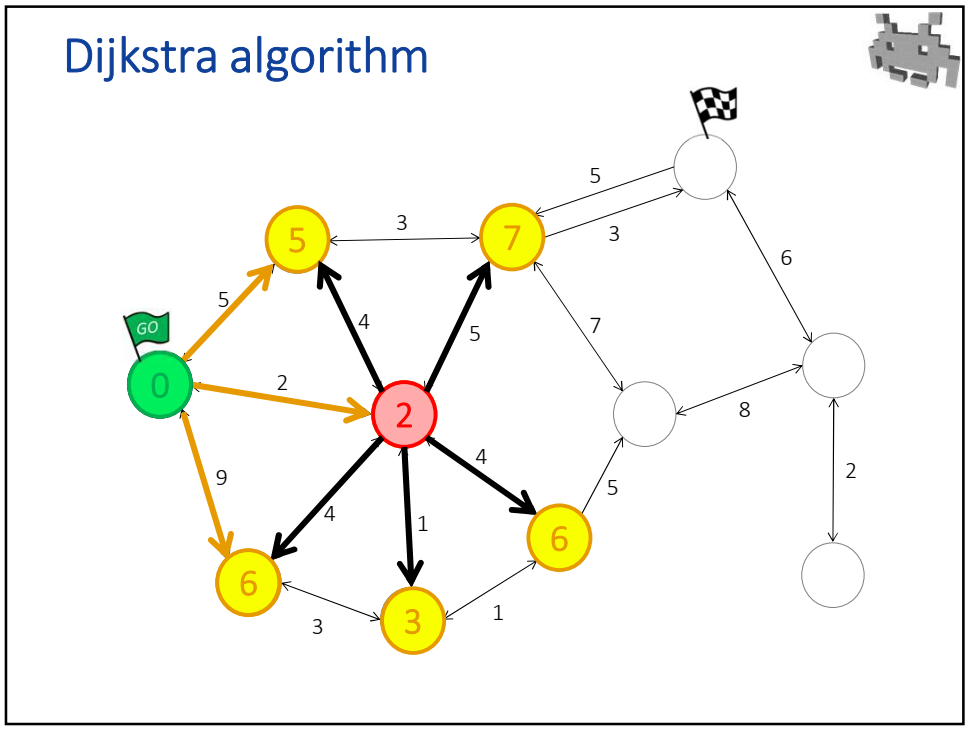
47


Dijkstra algorithm
GO
5
3
3
6
5
4
5
7
2
8
9
4
4
5
2
1
3
1

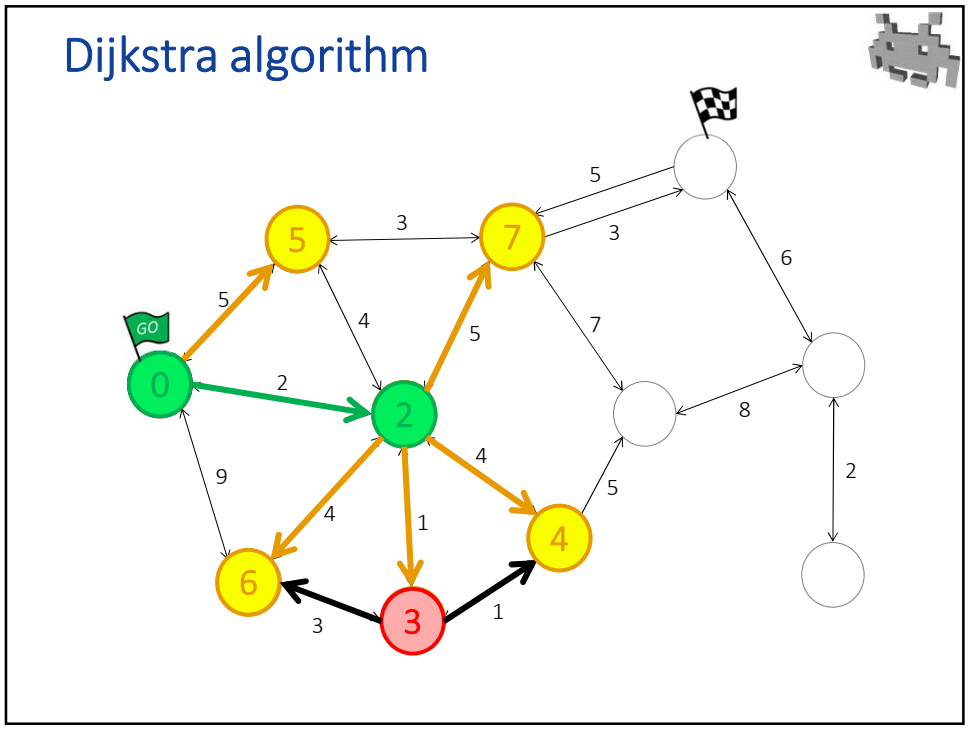
48


Dijkstra algorithm
GO
5
3
3
6
5
4
5
7
0
2
8
9
4
4
5
2
1
3
1

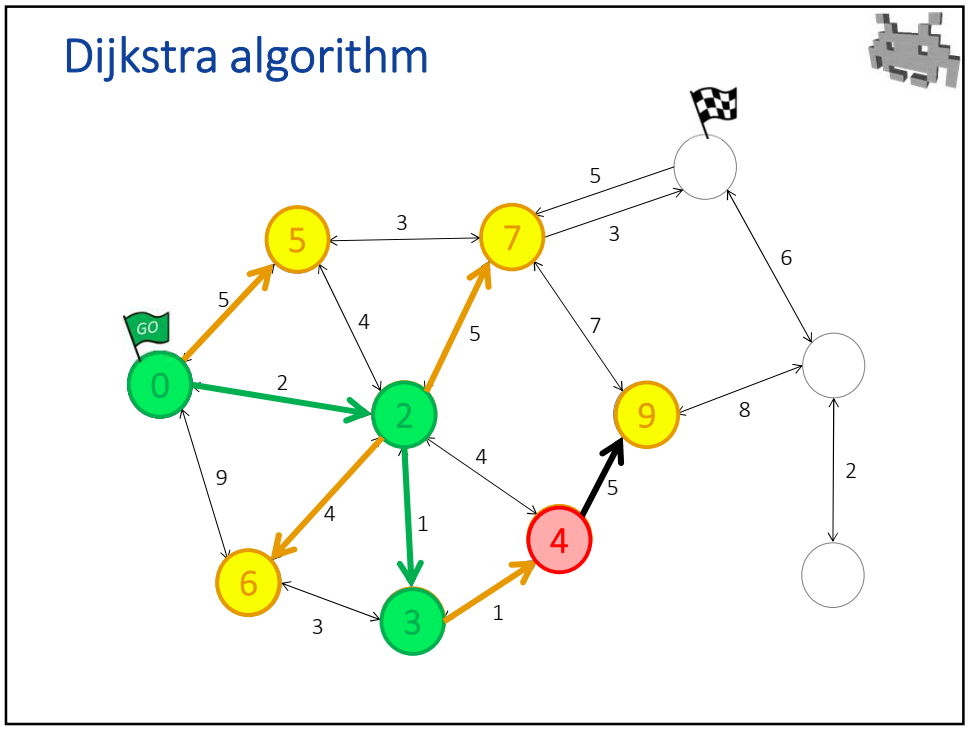
49


Dijkstra algorithm
GO
0
2
5
7
6
3
5
3
6
5
4
5
7
8
2
9
4
4
1
5
3
1

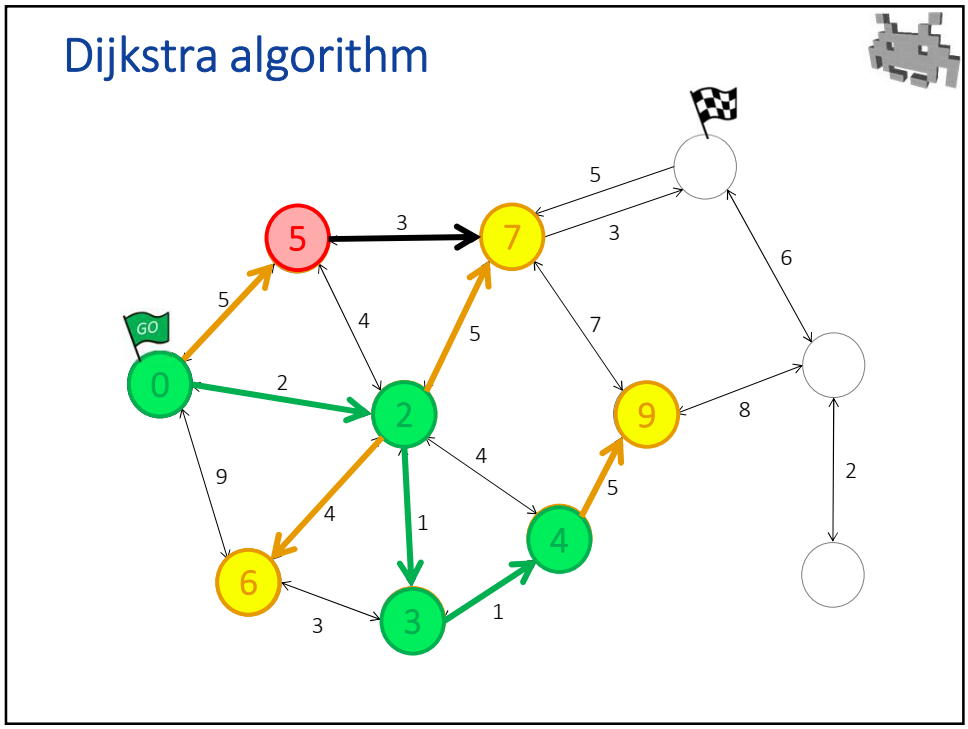
50


Dijkstra algorithm
GO
0
2
5
7
6
3
4
5
3
6
5
4
5
7
8
2
9
4
4
1
5
3
1

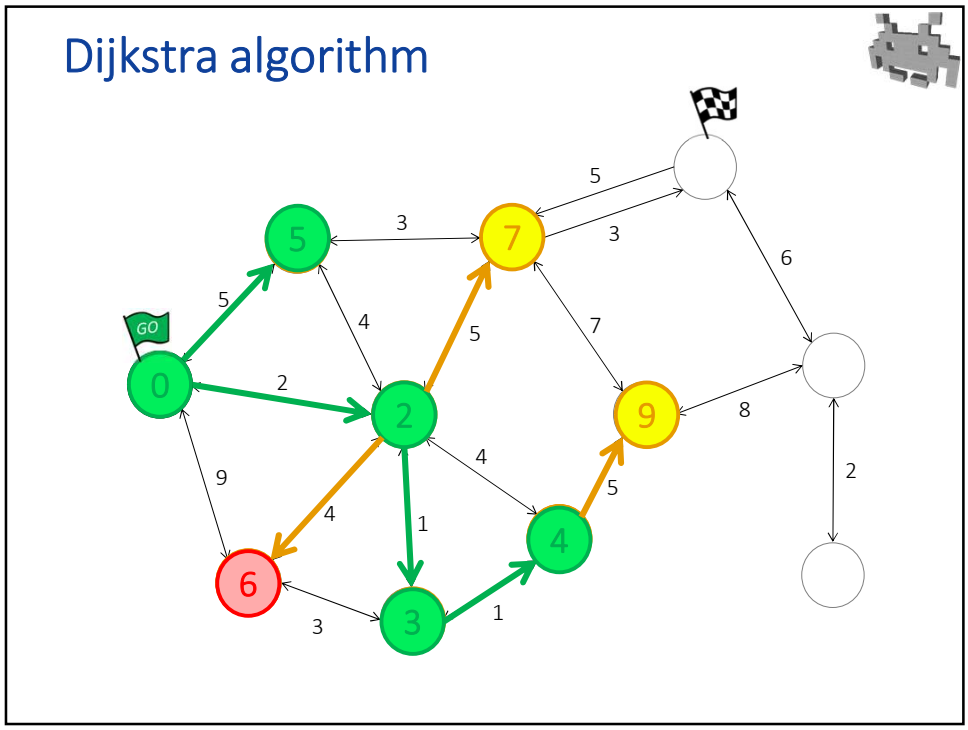
51


Dijkstra algorithm
GO
0
2
5
7
9
6
3
4
5
3
5
4
5
7
6
2
9
4
1
4
5
3
1
8
2

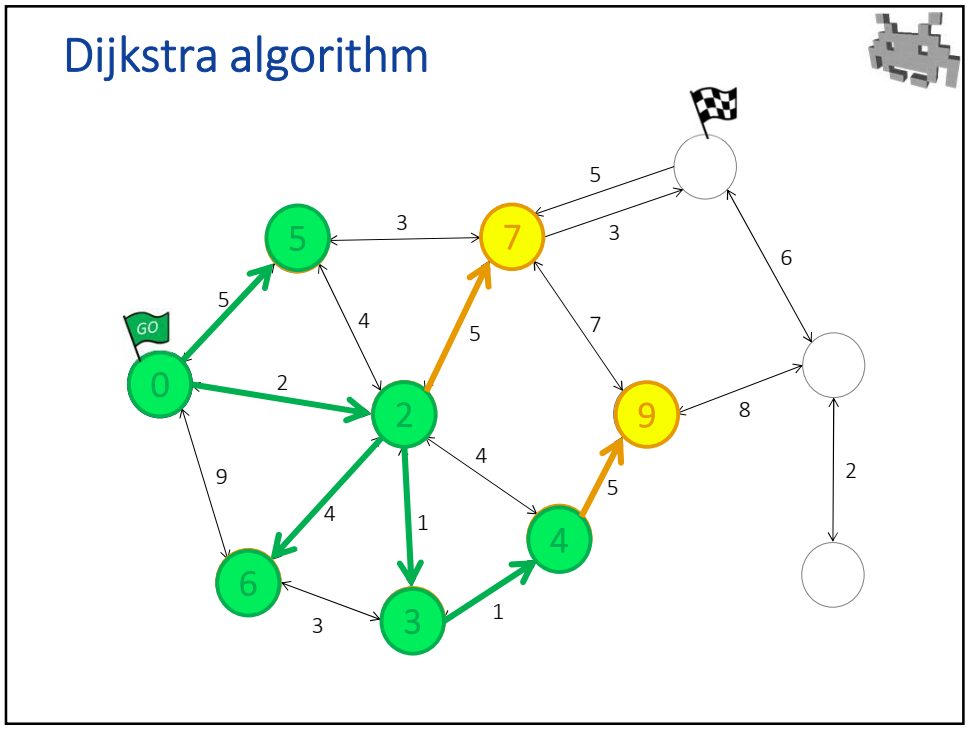
52


Dijkstra algorithm
GO
0
2
5
7
9
6
3
4
5
3
5
4
5
7
6
2
9
4
1
4
5
3
1
8
2

53

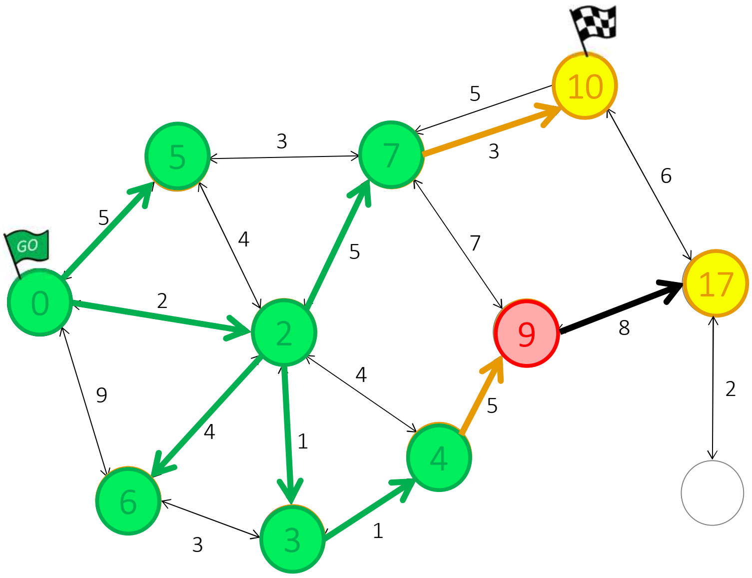

Dijkstra algorithm
GO
0
2
5
6
7
9
3
4
5
3
5
4
5
7
6
8
2
9
4
1
3
1

54


Dijkstra algorithm
GO
0
2
5
6
7
9
3
4
5
3
5
4
5
7
6
8
2
9
4
1
3
1

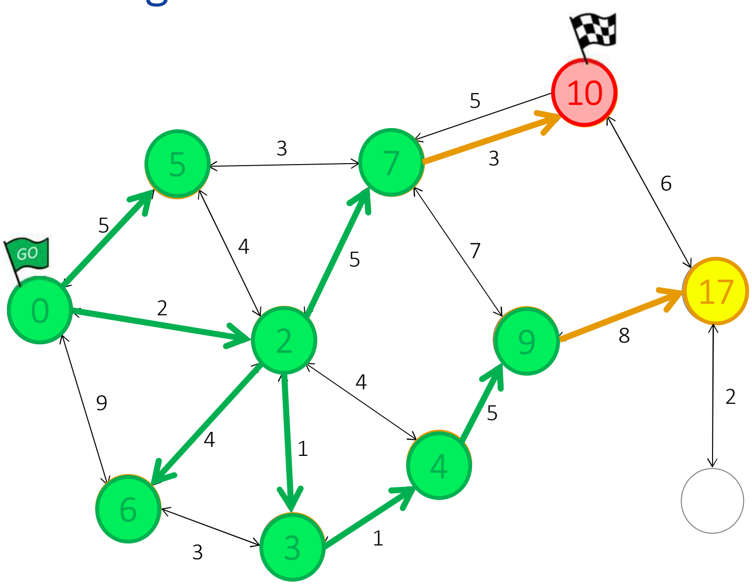
55

Dijkstra algorithm

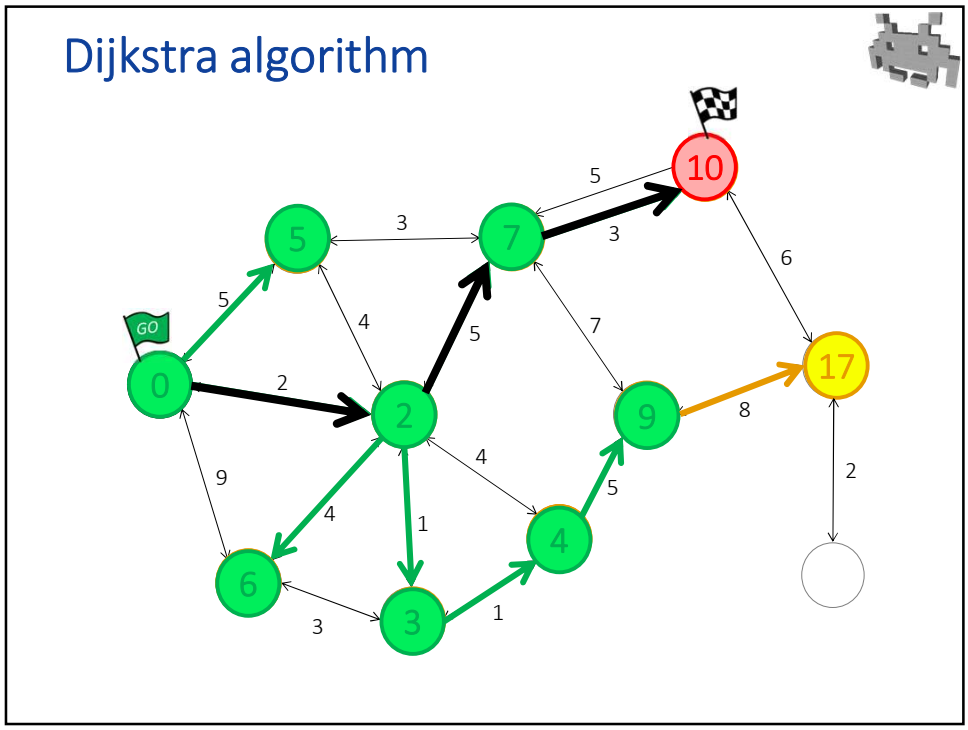


57

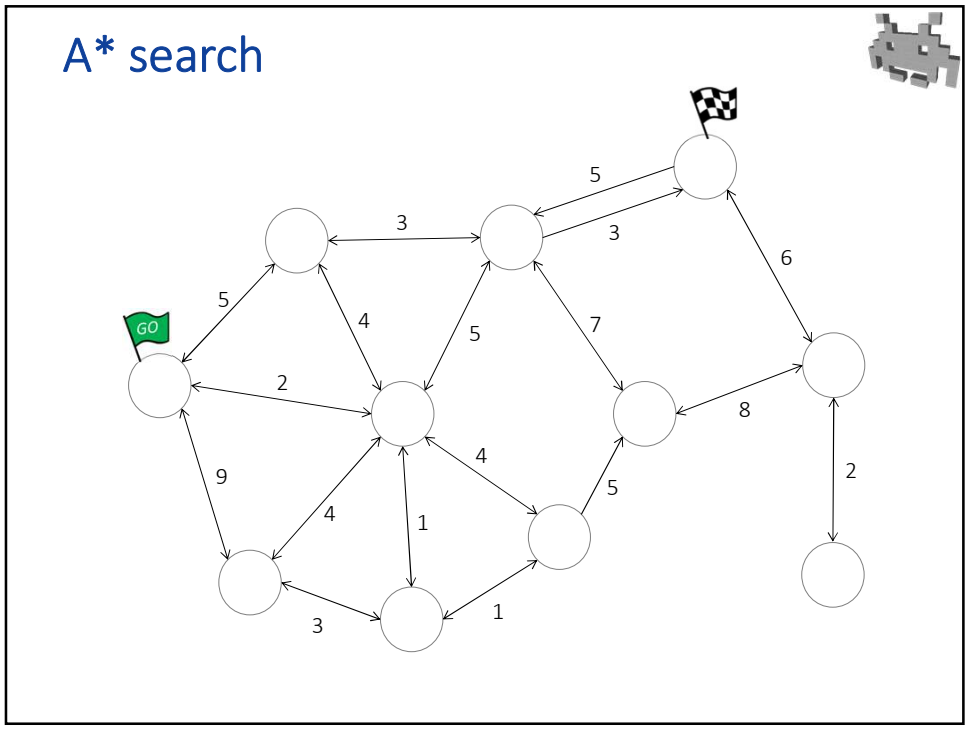
Dijkstra algorithm



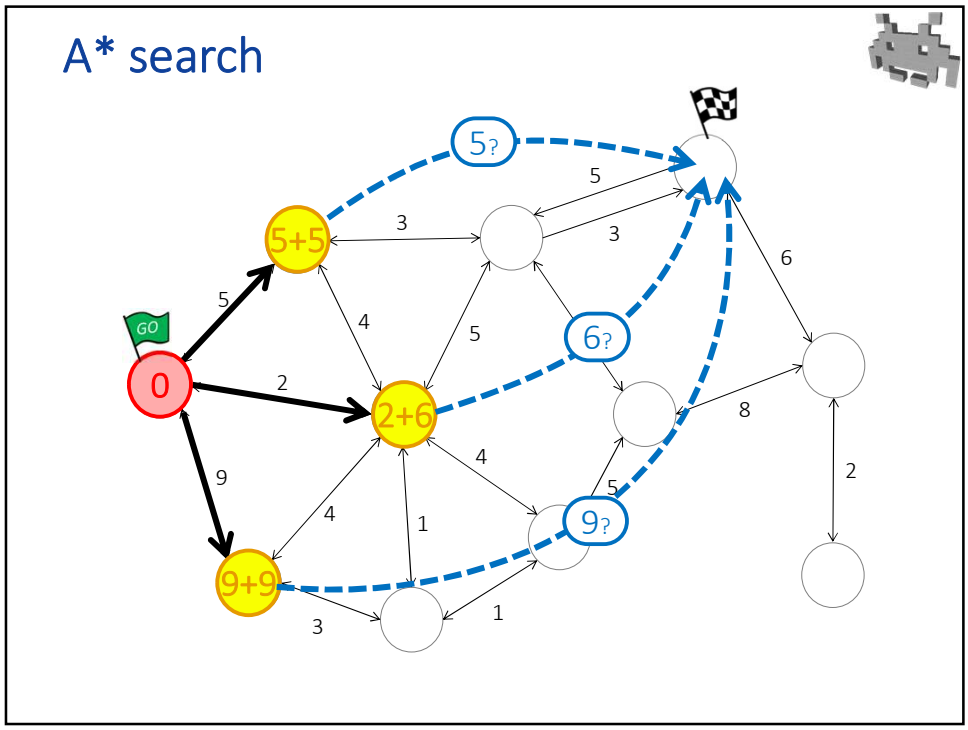
58


Dijkstra algorithm
GO
0
2
5
7
10
17
9
4
6
3
5
3
5
3
6
5
4
5
7
9
8
4
2
4
1
3
1

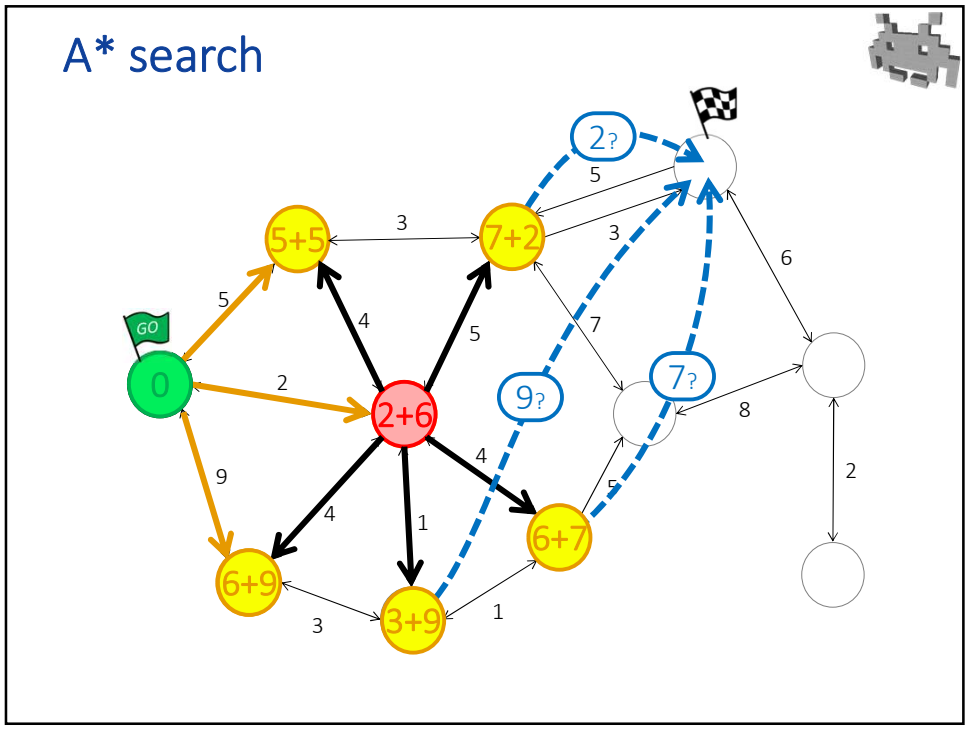
59


A* search
GO
5
3
5
3
6
5
4
5
7
2
8
9
4
5
2
4
1
3
1

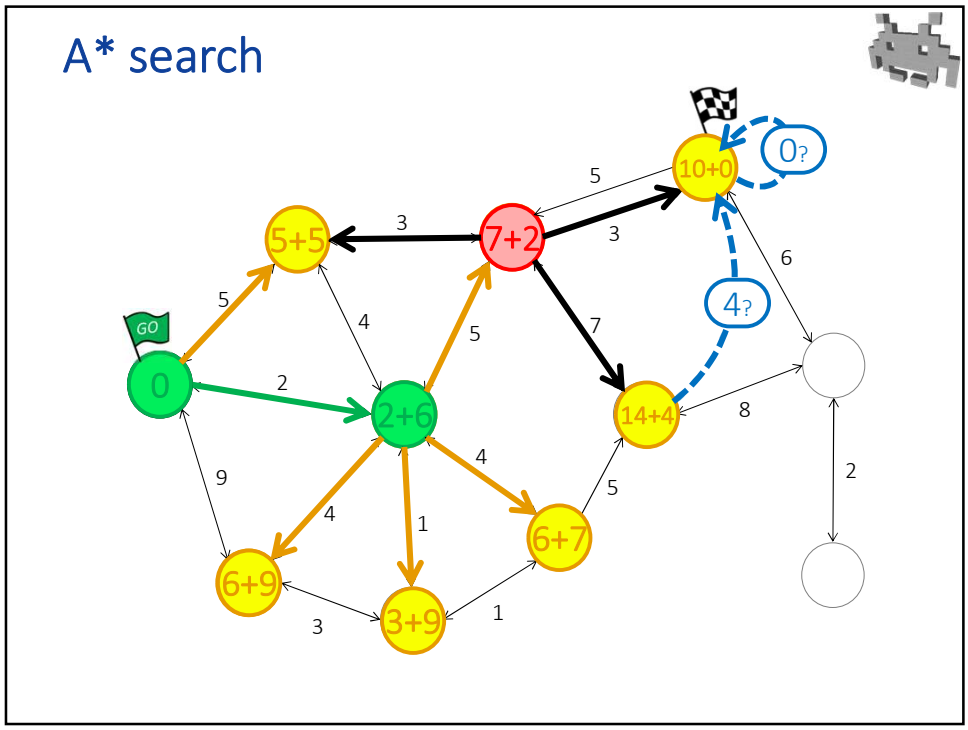
60


A* search
GO
0
5+5
2+6
9+9
5?
6?
9?
5
3
4
2
9
1
6
8

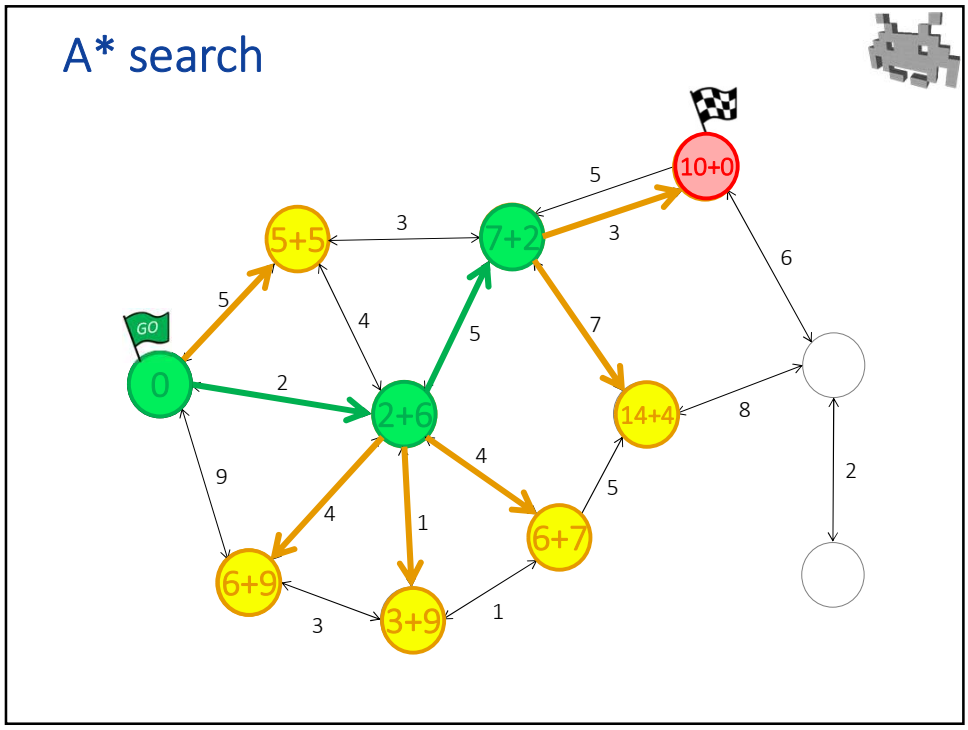
61


A* search
GO
0
5+5
7+2
2+6
6+9
3+9
6+7
2?
9?
7?
5
3
4
2
9
1
6
8
7

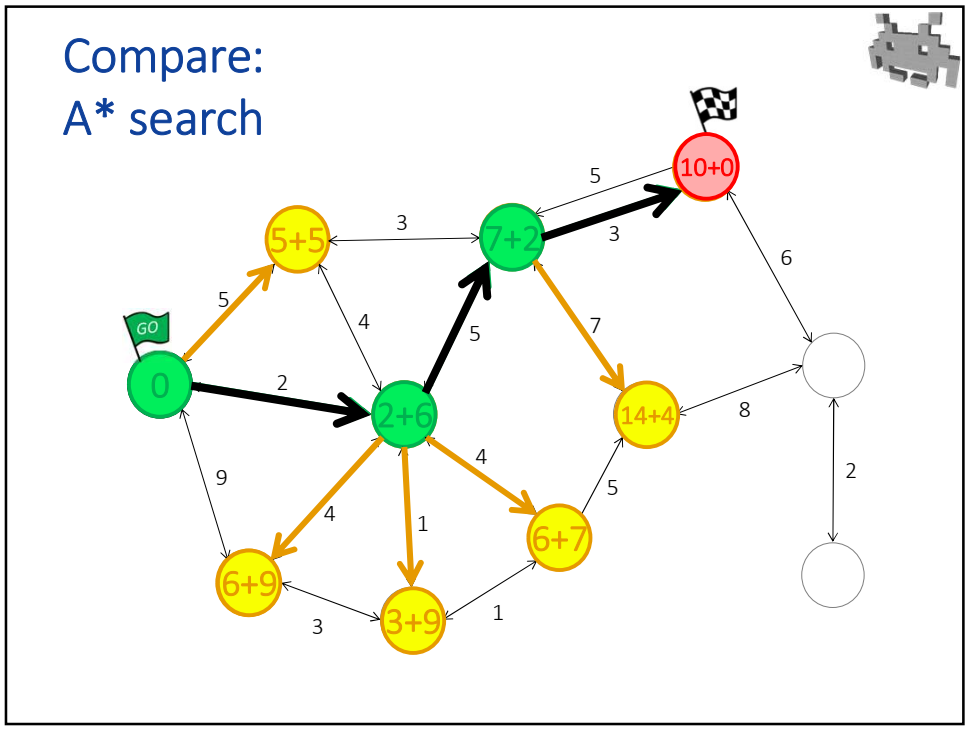
62


A* search
GO
0
5+5
7+2
10+0
0?
4?
2+6
14+4
6+7
6+9
3+9
5
3
5
4
5
7
6
2
9
4
1
4
5
8
2
3
1

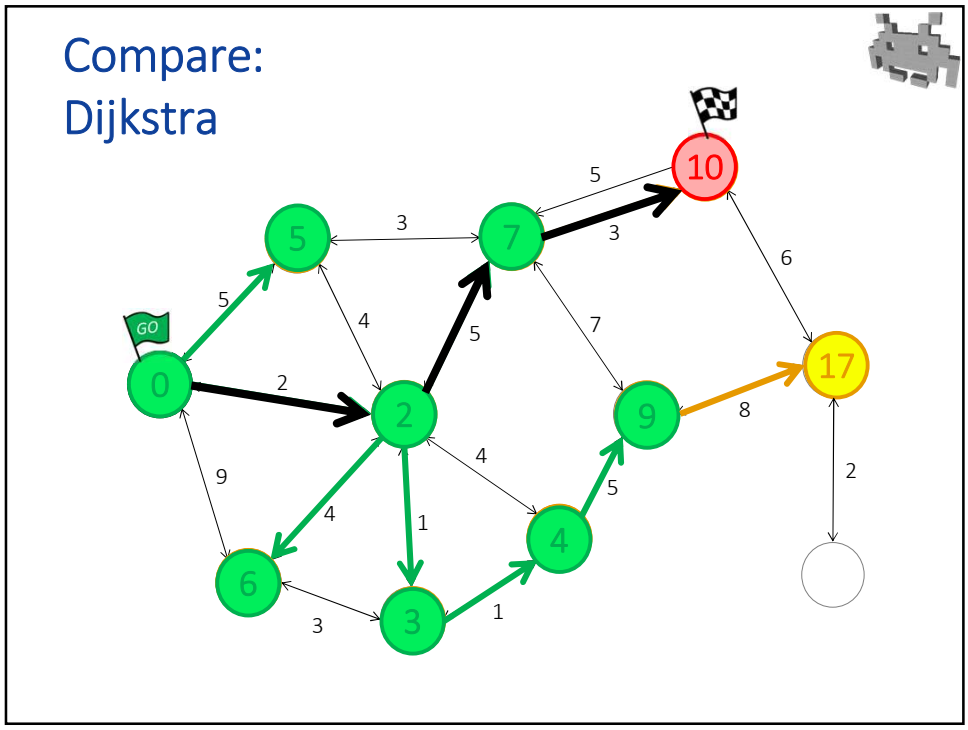
63


A* search
GO
0
5+5
7+2
10+0
2+6
14+4
6+7
6+9
3+9
5
3
5
4
5
7
6
2
9
4
1
4
5
8
2
3
1

64

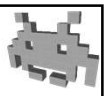

Compare:
A* search
GO
0
5+5
7+2
10+0
2+6
14+4
6+7
6+9
3+9
5
3
5
4
5
7
6
2
8
9
4
4
1
5
3
1
2

66


Compare:
Dijkstra
GO
0
5
7
10
2
9
17
6
3
4
5
3
5
4
5
7
6
2
8
9
4
4
1
5
3
1
2

67

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 reflets 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

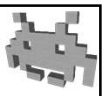
69

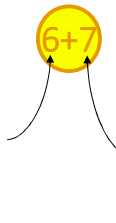
Dijkstra algorithm: notes.

- Any nodes is visited / processed only once
  - Or zero times! Not all nodes are visited
- Can be trivially adapted for multiple initial and target nodes
  - Returns the shortest-path between any initial and any target nodes (the smallerst!)
- The algorithm requires to keep track of a set of "active" nodes
  - (in yellow, in the graphs above)
  - 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
  - if negative costs: still terminates, only not guaranteed to find the optimum

70

A* algorithm: ("A-star") notes

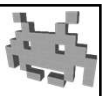




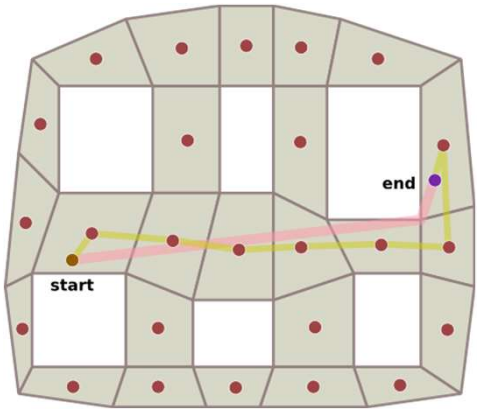
- Dijkstra not efficient enough
  - visits too many nodes
  - explores paths which are obviously wrong
  - it's greedy, only guided only by distance from Start)
- "A* search" is a variation. Main idea: smarten up! with an estimate of the remaining distance to Dest
  - function $h(X, D)$ – with X, D being nodes: returns an estimate of the *minimal* cost to go from x to Destination node D
    - Function h must be provided to the algorithm
    - 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

71

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

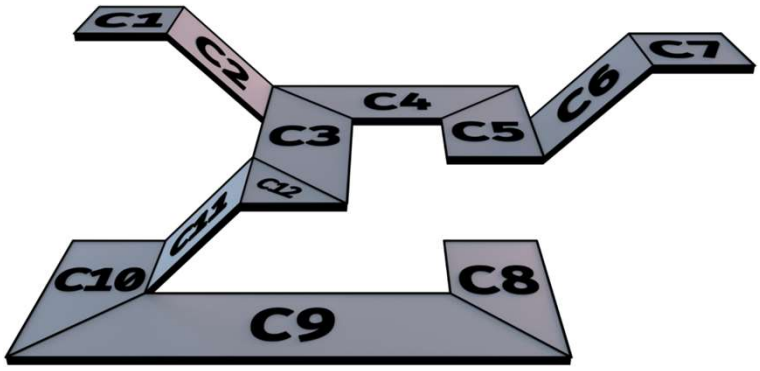


- Answer: Nav-meshes ("Navigation meshes") or AI meshes
  - a polygonal mesh
  - faces: graph nodes (places where the NPC can stand)
  - edges between faces: graph arches (passage the NPC can traverse)



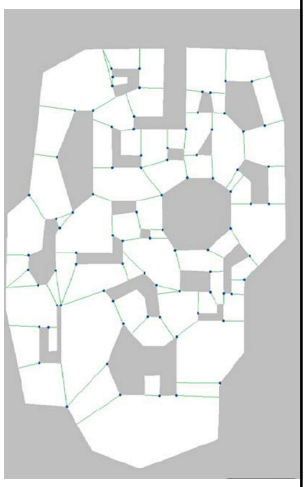
72

Nav Mesh (manually designed)



73

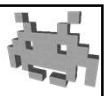
Baking a 3D Nav-Mesh



- 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

74

Customizing A* / Dijkstra



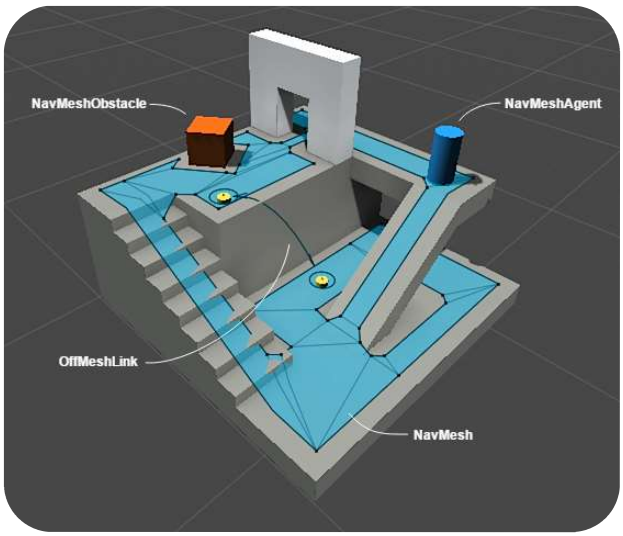
- Cost function ≠ time or distance
- Customize the costs freely
  - E.g. doors: add cost to open them
  - E.g. in a shooter:
    - Increase cost of nodes currently "under friendly fire" ("don't get in the line of fire of your friends"
    - Increase cost of exposed nodes ("don't get caught in the open")

find out with 3D raycasts

- Remember: A* needs underestimations
  - Decreasing costs requires care
  - E.g. add teleport doors? Be careful

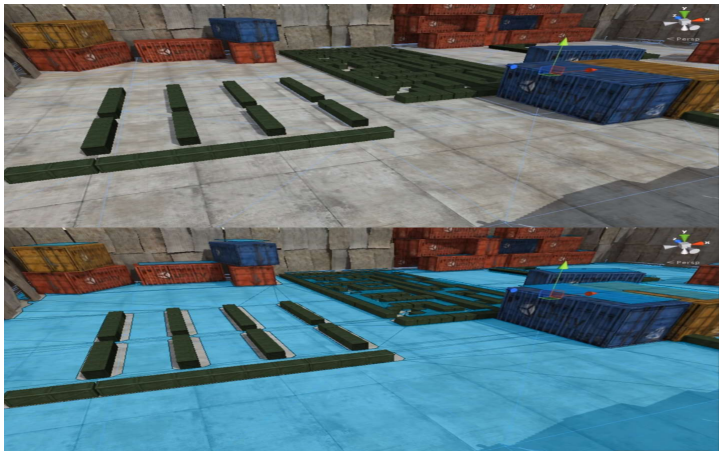
75

Nav Mesh: example in Unity



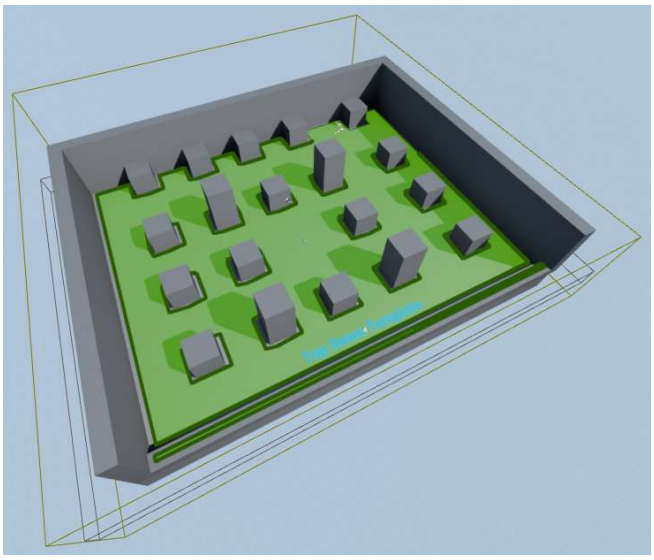
76

Nav-Mesh baking: example in Unity



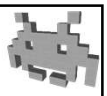
77

Nav-Mesh baking: example in Unreal



78

Flocking algorithms

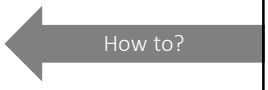


- A mid-level objective: “stay with the group”
  - but “not too close to anyone” (avoid collisions with peers)
- Each element of the swarm is attracted to the position of the 3D barycenter of the swarm
  - but avoids collision with closer members
- ==> decent flocking behavior emerges
  - e.g. flock of birds, school of fishes

80

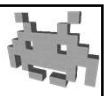
Thinking phase (aka Planning): about the mid-to-high level goals

- 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!



81

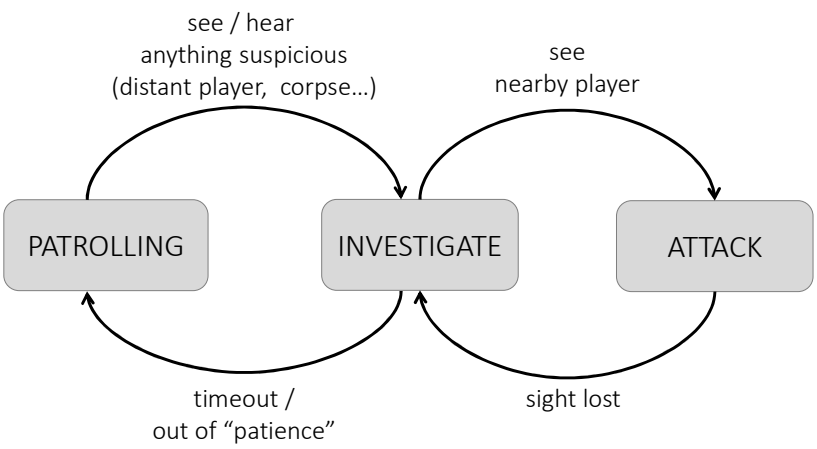
FSM
(more technically: Moore machines)



- 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)

82

FSM example: a guard



84

Implementing FSM

```
if (status==PATROLING)
  then doPatroling();
if (status==ATTACK)
  then doAttack();

procedure doPatroling(){
   //  …
   if next_nav_point reached …

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

- FSM can serve as a coding guideline
  - use one “status” variable
  - transitions: manually coded in
- Or, as a behavior authoring tool
  - intended for the AI designer
  - supported by game engine
  - WYSIWYG editors possible
  - transitions: conditions (to be checked automatically)
  - statuses: linked to effects (sound, animation,…)
- Limitation: scalability (with complexity)
  - only good for simple behavior
  - quickly produces intricated nets
- Let’s see alternatives:
  - HFSM
  - Behavioral Trees

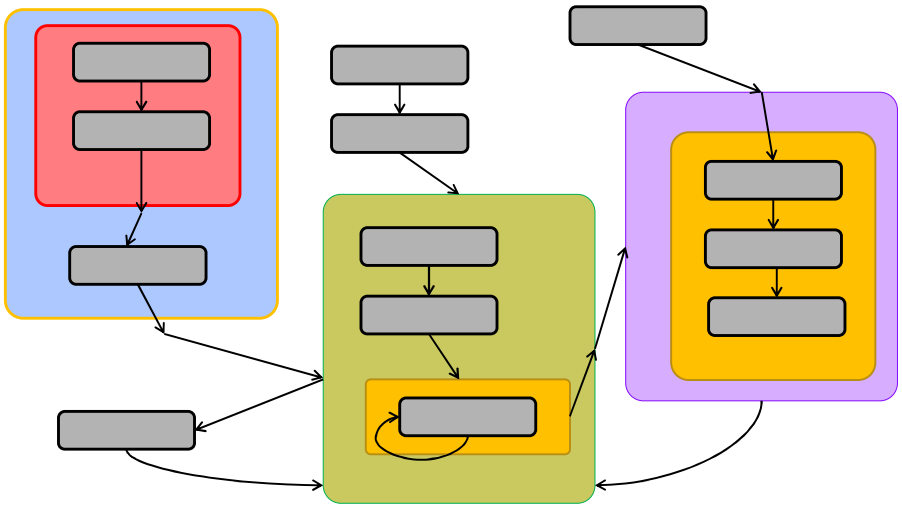
blur distinction between hi-level / low-level goals

also blur classic distinction between sensing / thinking / acting

86

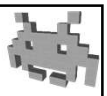
HFSM

Hierarchical Finite State Machines



89

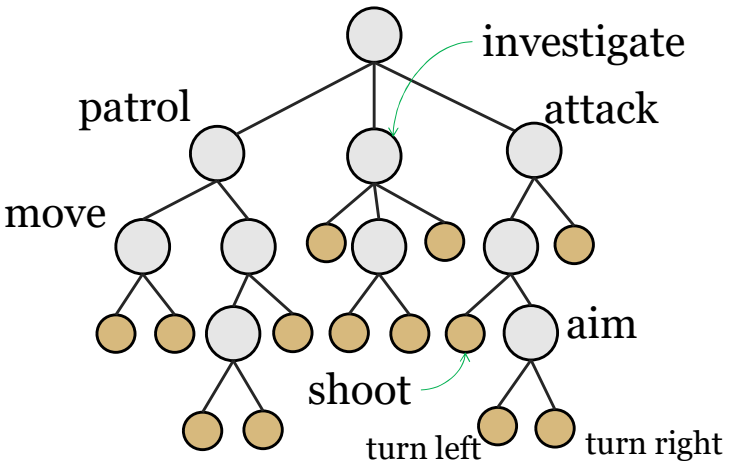
HFSM: concept



- An 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 a designed AI to another)

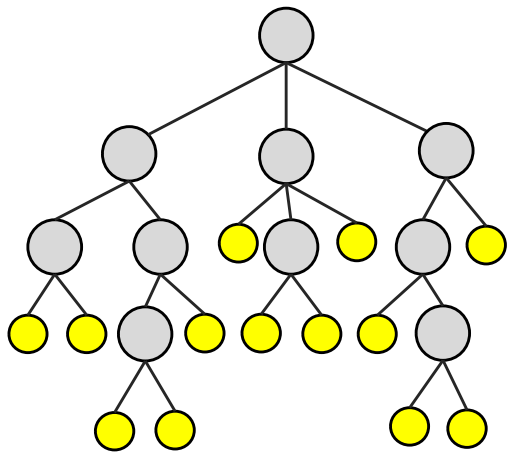
90

Behavioral Trees



91

Behavioral Trees



= interface between IA and environment: acts, sensing (queries)

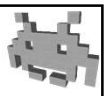
92

Behavioral Trees: nodes

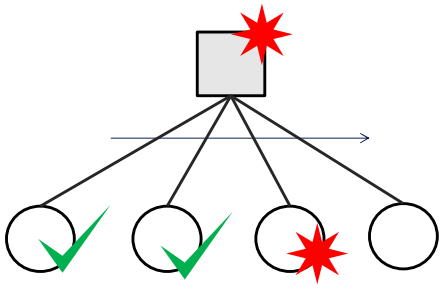
- every node, when it has done *running*, can either have:
  - failed
  - succeeded
- leaves are interaction with environment
  - action leaf:
    - animations, movements, sound, game logic…
    - Success: finished it.
      Failure: could not do it
      - (e.g. movement negated by obstacle, object 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

93

Behavioral Trees: nodes



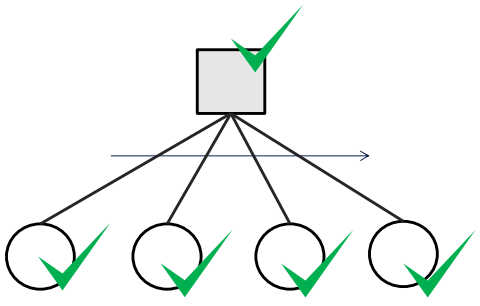
- internal nodes: sequence



94

Behavioral Trees: nodes

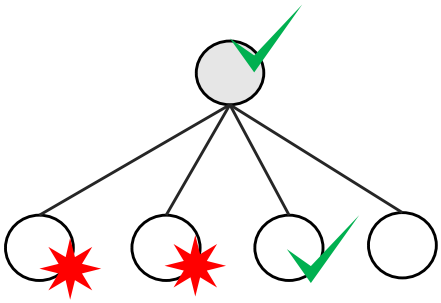
- internal nodes: sequence



95

Behavioral Trees: nodes

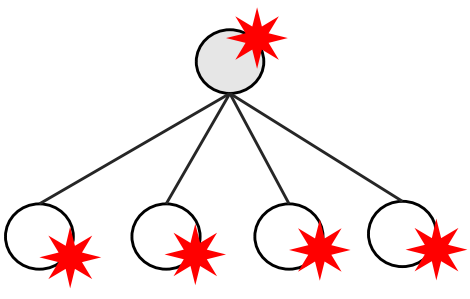
- internal nodes: selector



96

Behavioral Trees: nodes

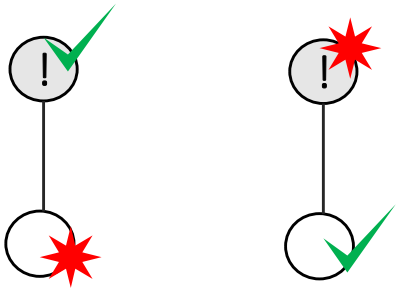
- internal nodes: selector



97

Behavioral Trees: nodes

- internal nodes: inverter

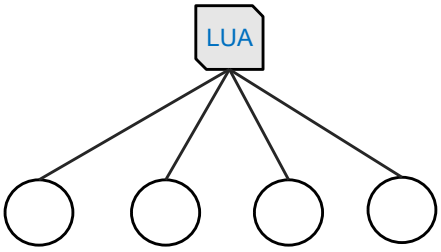


Only child

98

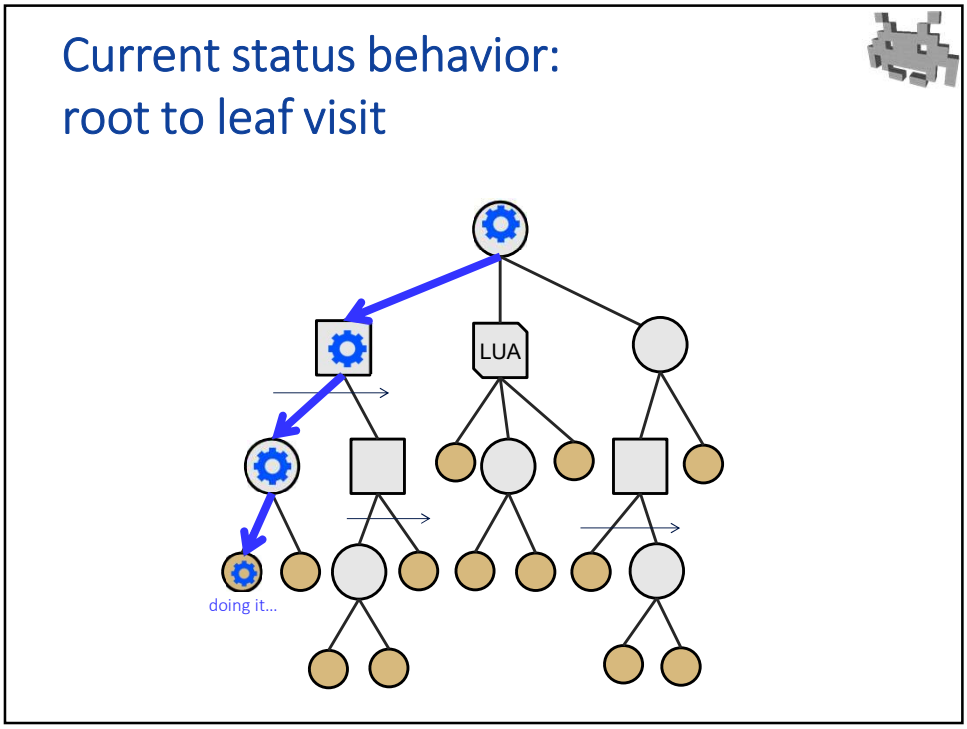
Behavioral Trees: nodes

- or, nodes can be programmed arbitrarily in scripted procedure (in , C#, JS…)
  - run children in some prescribed order
  - fail or succeed, as returned value



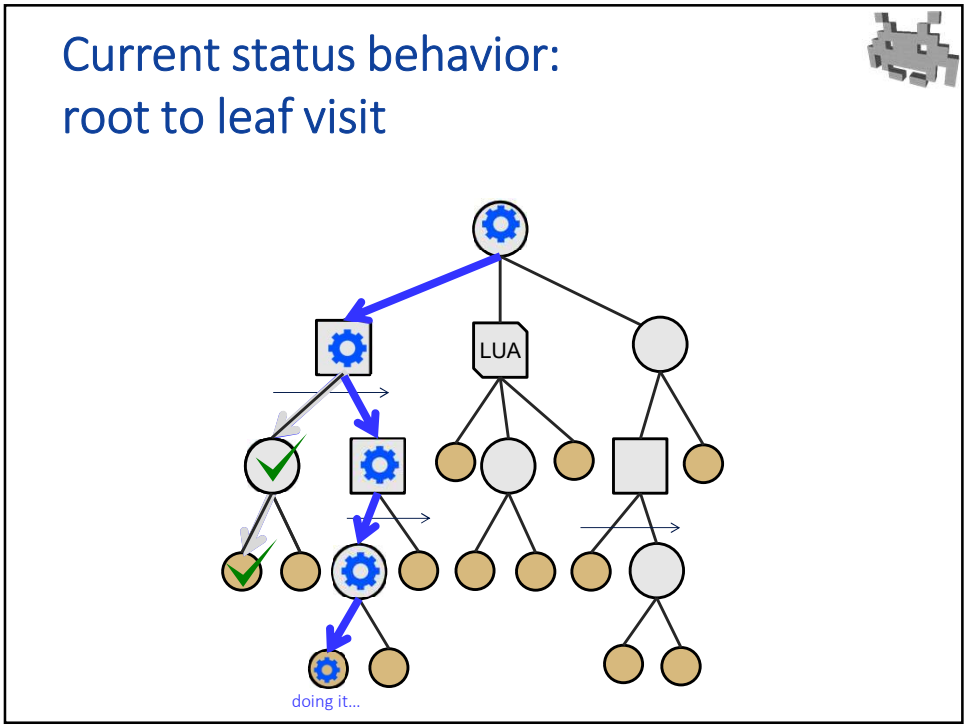
99

Current status behavior: root to leaf visit



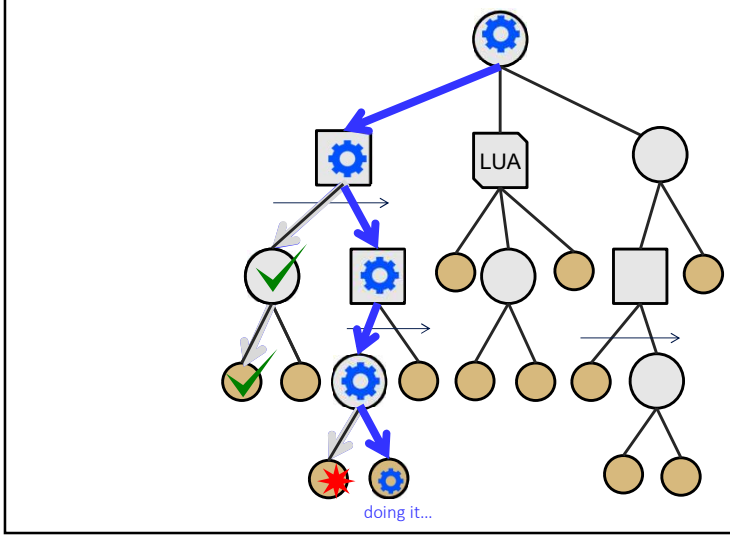
100

Current status behavior: root to leaf visit



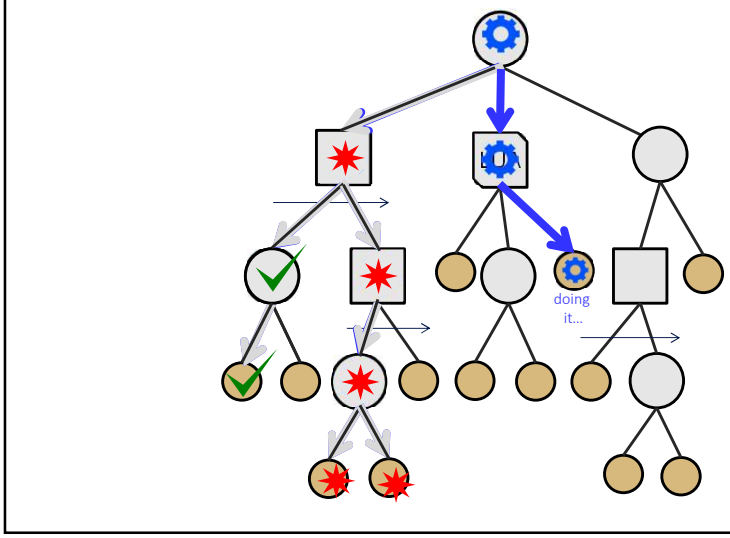
101

Current status behavior: root to leaf visit



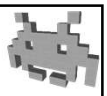
102

Current status behavior: root to leaf visit



103

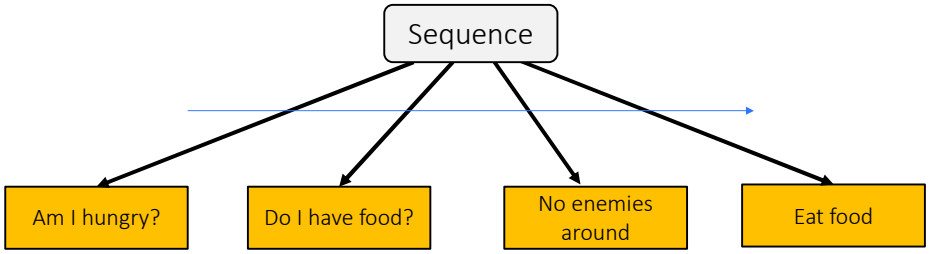
Behavior trees: notes



- Each node can be:
  - ✸ failed
  - ✓ success
  - ⚙ in progress
    - (or, still unvisited)
- Current IA-mind status:
  root-to-leaf path of ⚙ nodes
  - Shallow nodes: current high-level objectives
  - Deeper nodes: current low-level objectives
  - Leaf at the end of the path: current action/sensing action

104

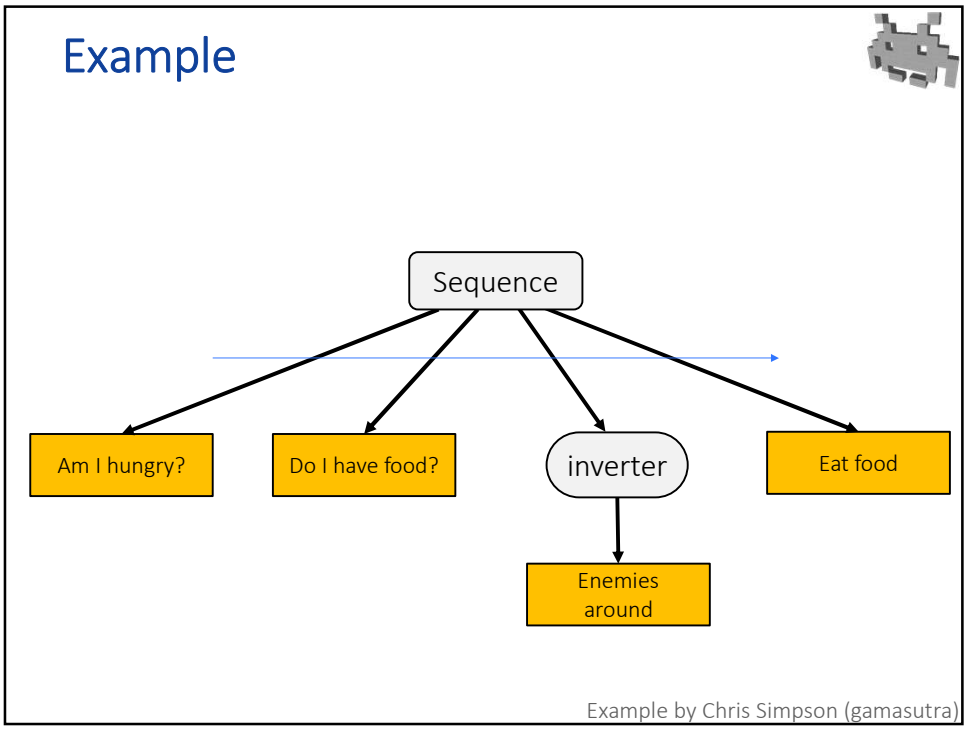
Example



Example by Chris Simpson (gamasutra)

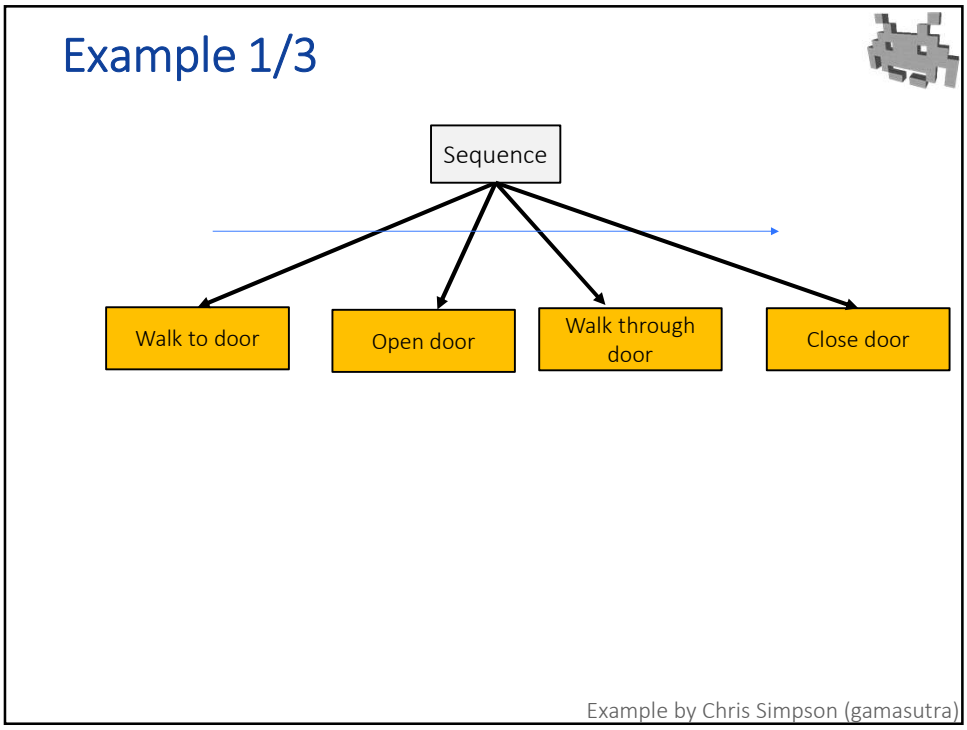
105

Example



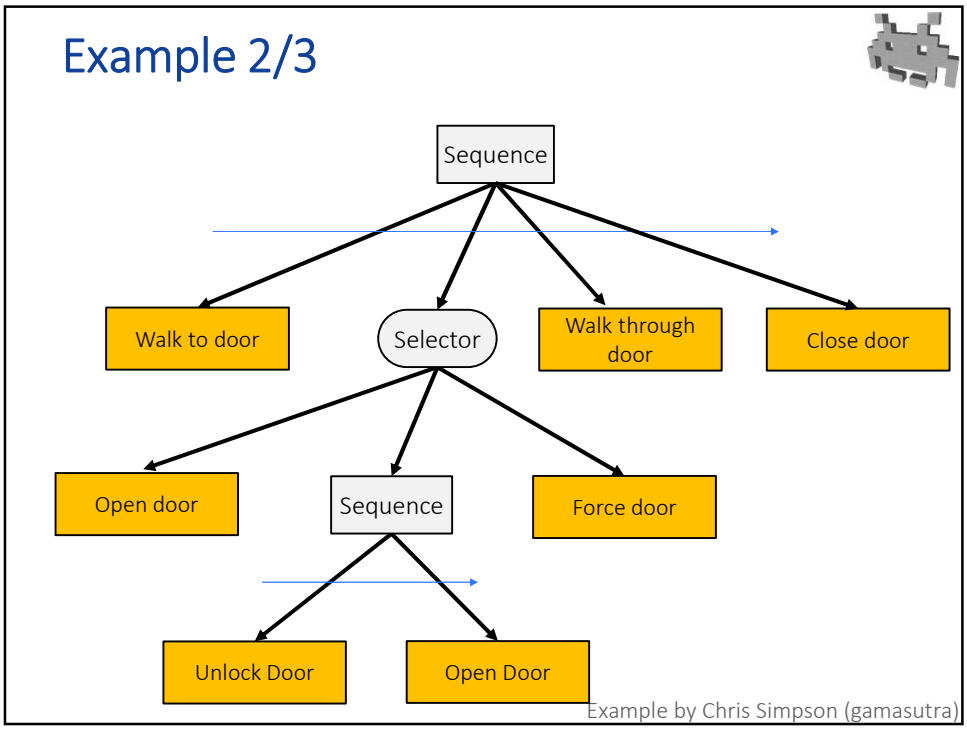
106

Example 1/3



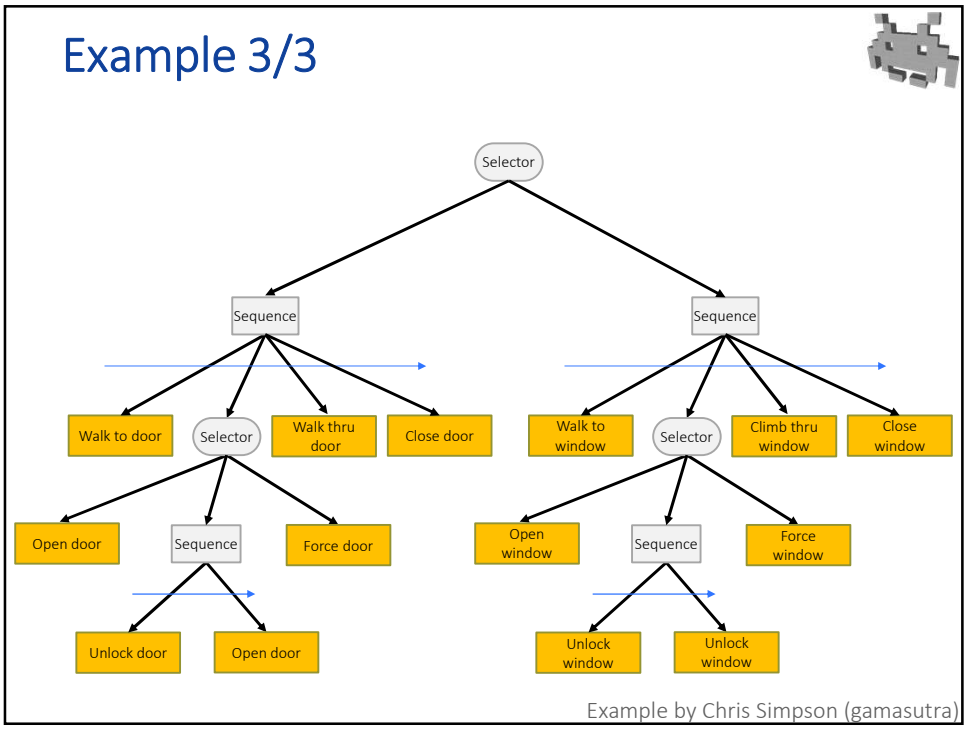
107

Example 2/3



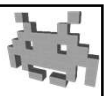
108

Example 3/3



109

Behavioral tree: advantages



- Intuitive to model
  - even for designers without a CS background?
- Subtrees (intermediate goals) can be reused for similar situations across NPCs / games
  - E.g. anything than needs to enter the house uses the same "try to enter house" subtree
- Scripted nodes (e.g. LUA) modelling more complex decision making can also be reused
  - BT serves as a framework to organize / reuse scripts
- Simpler behaviors = building blocks of more complex behaviors
  - Top-down or bottom-up approach to NPC behavior design

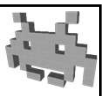
111

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

112

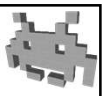
Support for NPC-behavior in a game engine: a summary



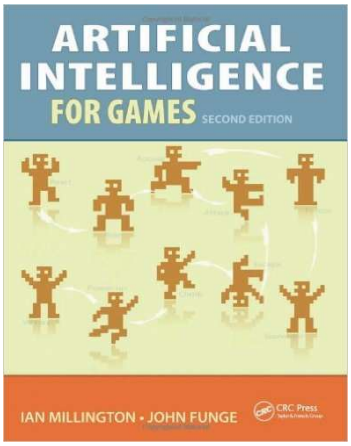
- Assets for NPC behaviors:
  - for *behavior modelling*:
    - Scripts (can well be the only one)
    - FSM
    - HFSM
    - BT
  - for *navigation*:
    - nav-meshes (aka AI-meshes)
  - for *sensing / queries*:
    - hit-boxes, bounding volumes, spatial indexing
    - the same ones used by physic engine for collision detection
- Game tools
  - to assist their construction (by AI designer)
- Support for a few hard-wired functions
  - to solve lowest level tasks om a 3D environment

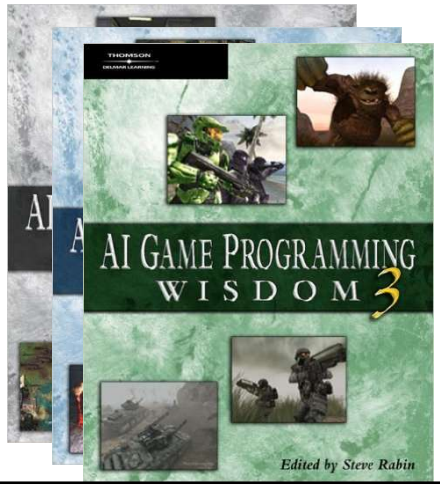
113

To investigate further



- AI for VideoGames course!
- Books:





114