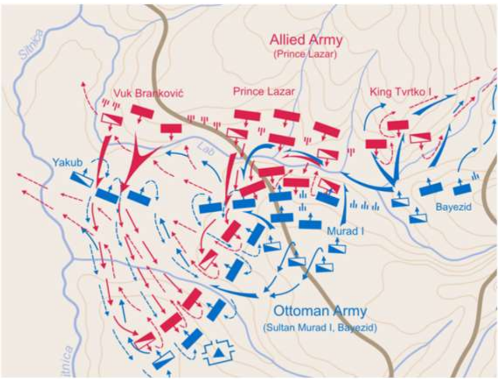
3D VideoGames 2020/2021
Università degli Studi di Milano

Artificial Intelligence in 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. 8: Game 3D Animations ●●●
- lec. 9: Game 3D Audio ●
- lec. 10: Networking for 3D Games ●
- lec. 11: Artificial Intelligence for 3D Games ●
- lec. 12: Game 3D Rendering Techniques ●●

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

2

Game Engine

- Handling common task of a game dev
  - Game logic (levels)
  - Renderer
    - Real time transoform + 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

3

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
- Reasons:
  - algorithm breakthroughs
  - computational power!!!
    - e.g., GP-GPU

4

AI in games: many uses

Main course: "Artificial Intelligence for Video Games"

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

e.g., look up "Sokoban"

5

AI in games: one important use (trending in research)

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

      (trivial to measure (score))
  - *Output:*
    - how to activate muscles to do it → skeletal animations
    - (minimizing used energy)
  - *How:*
    - genetic algorithms, Evolution strategies
    - physical simulation to score candidates

6

AI in games:
The 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

7

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

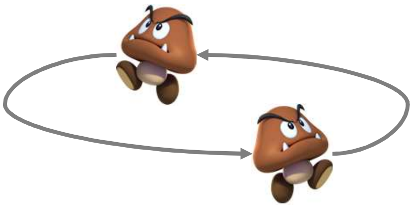
- Many differences with "problem-solving" AI:
  - "cheating" completely possible
    - e.g., info "magically" available to the Interactive Agent
  - real-time response always needed
    - very frequent decisions of the Interactive Agent (30-60 Hz!)
    - "on-line", and "soft real time"
  - sub-optimal 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

8

Designing NPC behavior: not necessarily intelligence

NPC behavior is not necessarily
- "intelligent"
- complex

Rather, NPC behavior needs be often to be:
- intuitable / predictable
- learnable
- understandable
- story driven?
- interesting to exploit
- uses:
  - tune difficulty
  - elicit interesting strategies by the player
  - make a given strategy rewarding
- etc.



9

Game AI -vs- AI to solve Games

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

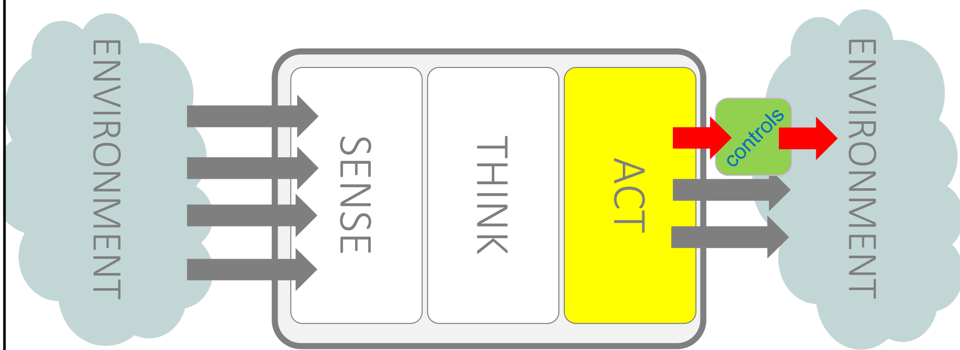
to find more about AI to (optimally) *play* games,
look for:
- min-max algorithms (with pruning)
  - algorithms to solve
    complete knowledge, turn based games
- Nash equilibrium (from Game Theory)
  - solution concept to address
    non cooperative games

10

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

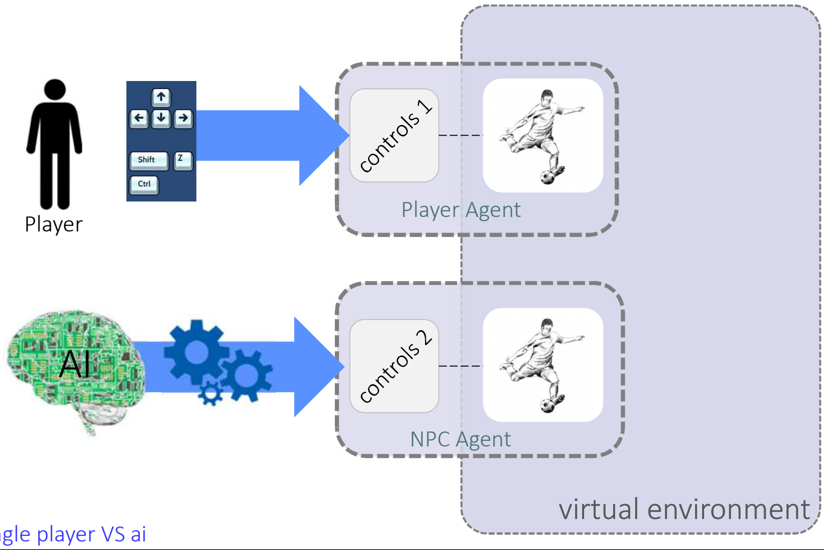
11

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



14

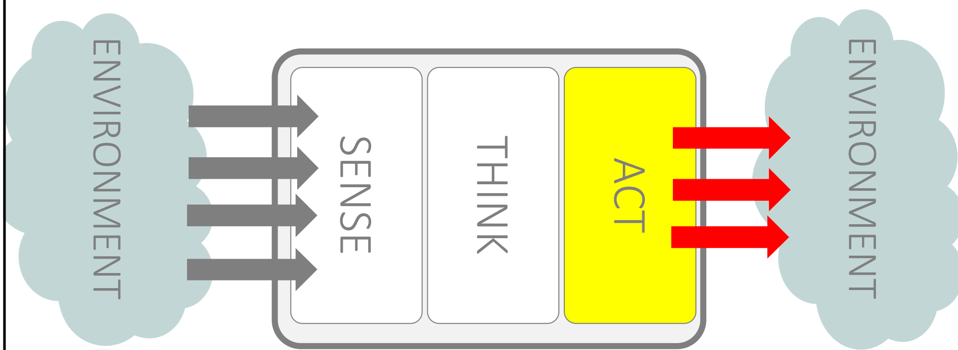
Controls and Agent
(a useful abstraction)



scenario: single player VS ai

18

Interactive Agent



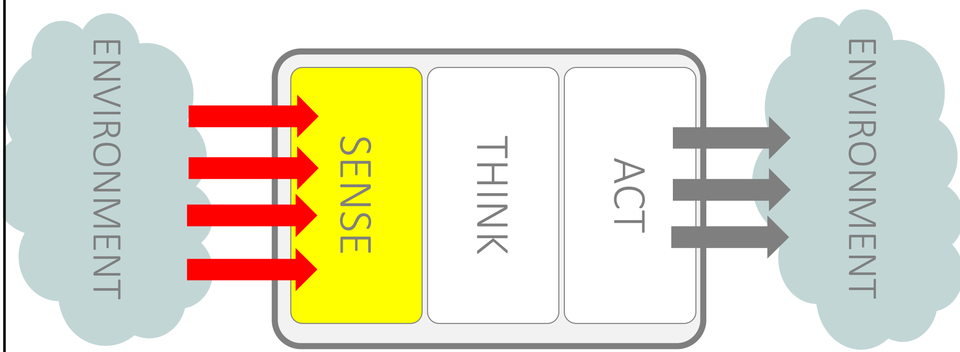
20

Acts :
In robotics, “actuators”. In 3D 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 an RTS)
- Effects on game-logic
  - e.g. objects appearing, doors unlocking,
    HP decreased / healed, money spent / gain, etc

21

Interactive Agent



22

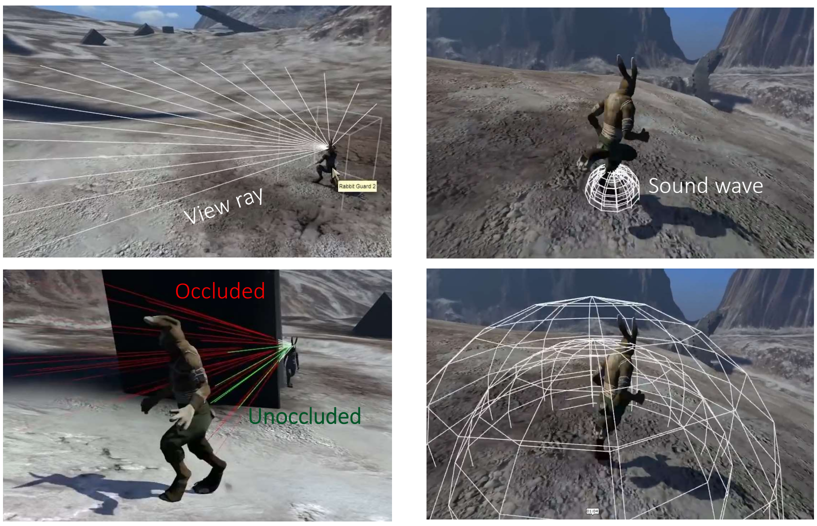
Sensing
(in robotics, by "Sensors". In games?)

- Gather info ("percepts")
  - which will be used for the "think" phase
  - NB: this info must often persist in the "mind" of the agent!
    - more abut 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: synthetize 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

23

Simulating senses in a 3D environment

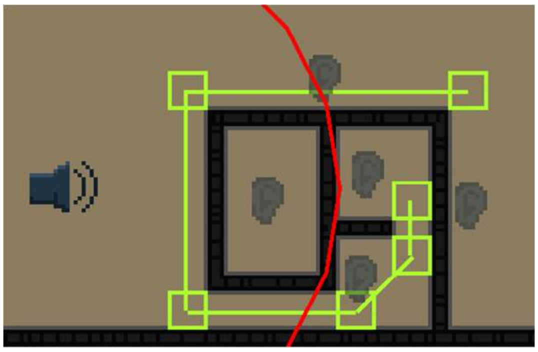


Sight Hearing

24

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

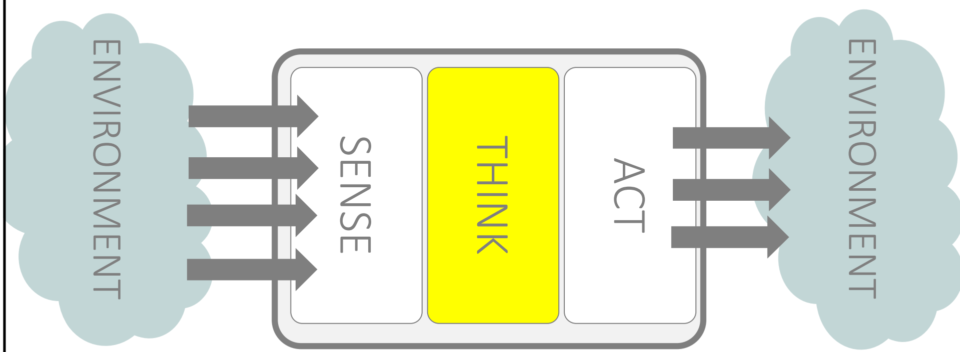
- Pathfinding for echos simulation



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

25

Interactive Agent (IA)



26

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

27

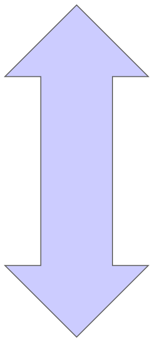
Thinking phase (aka Planning)

- 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!

28

Authoring an AI for an NPC

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



29

Authoring an AI for an NPC: *classic approach*

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

30

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)

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 "Patroling"
  - 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)

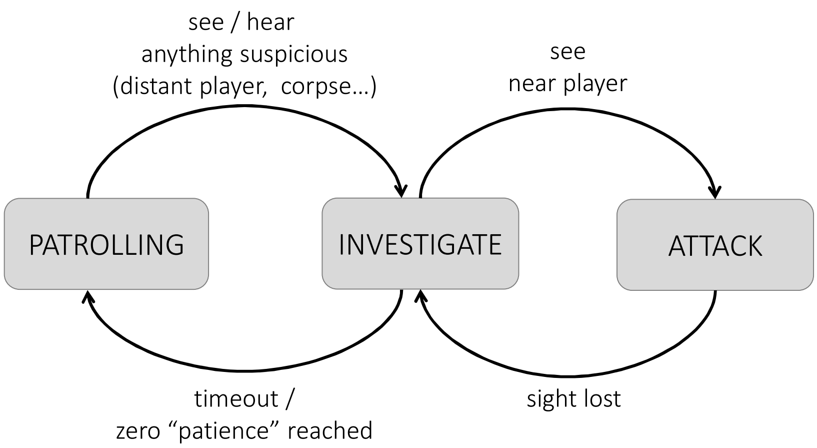
33

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

34

FSM example: a guard



36

FSM in practice

```
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;
}
```

- 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)

37

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

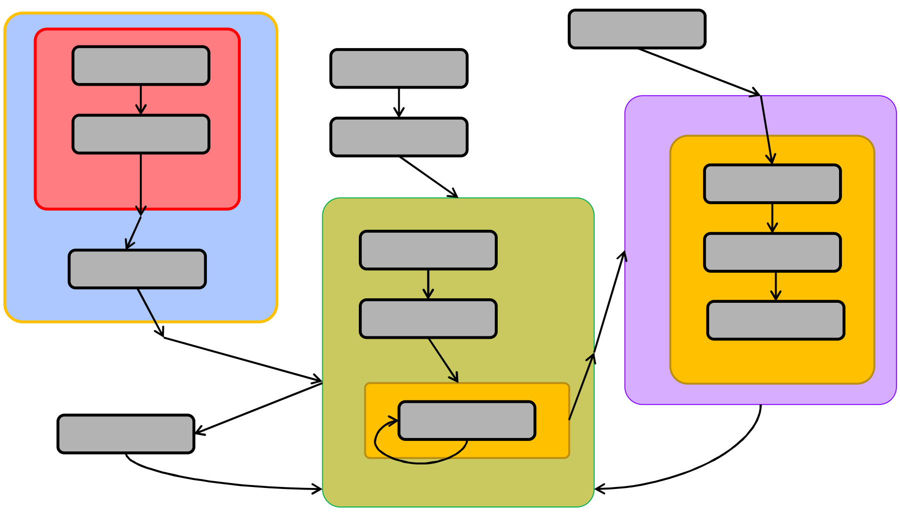
unified handling of all levels; blur classic distinction between hi-level / low-level planning.

also blur classic distinction between sensing / thinking / acting

38

HFSM

Hierarchical Finite State Machines



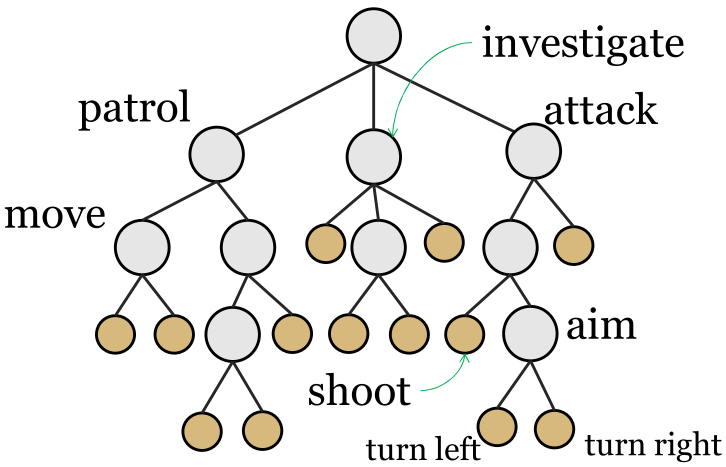
39

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)

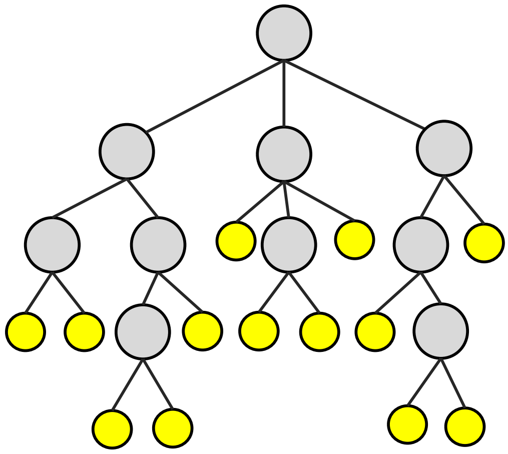
40

Behavioral Trees



41

Behavioral Trees



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

42

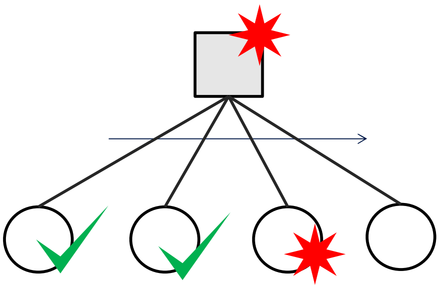
Behavioral Trees: nodes

- 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: 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)
  - the distinction not necessarily strict

43

Behavioral Trees: nodes

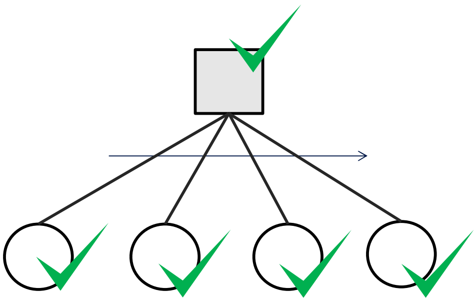
- internal nodes: sequence



44

Behavioral Trees: nodes

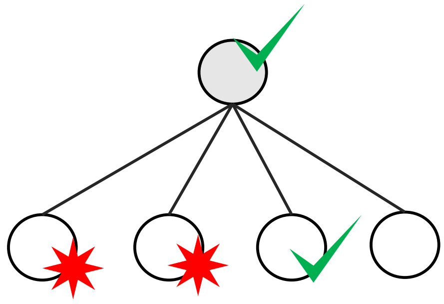
- internal nodes: sequence



45

Behavioral Trees: nodes

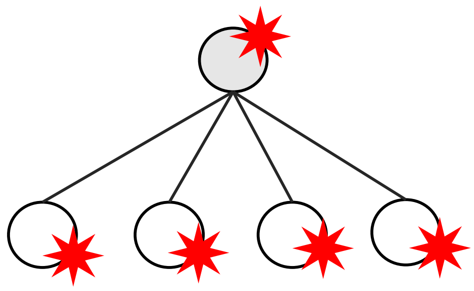
- internal nodes: selector



46

Behavioral Trees: nodes

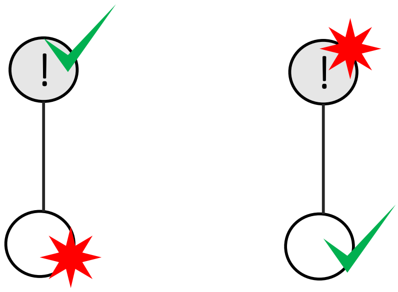
- internal nodes: selector



47

Behavioral Trees: nodes

- internal nodes: inverter

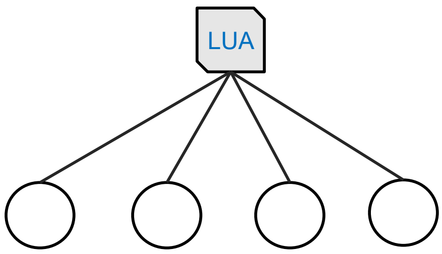


Only child

48

Behavioral Trees: nodes

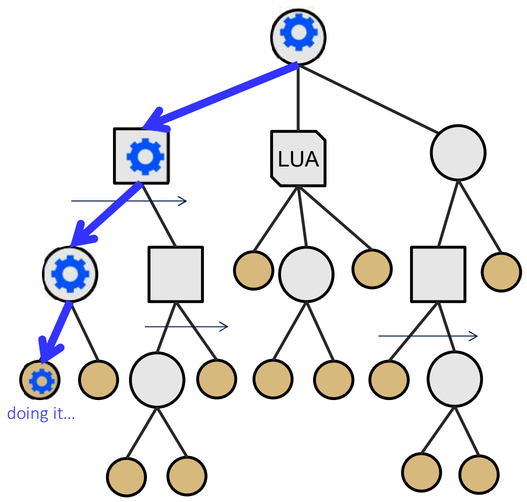
- or, nodes can be programmed arbitary (scripted procedure) (in LUA, C#, …)
  - run children, as calls
  - fail or succeed, as returned value



BT as a framework to structure / reuse / organize scripts

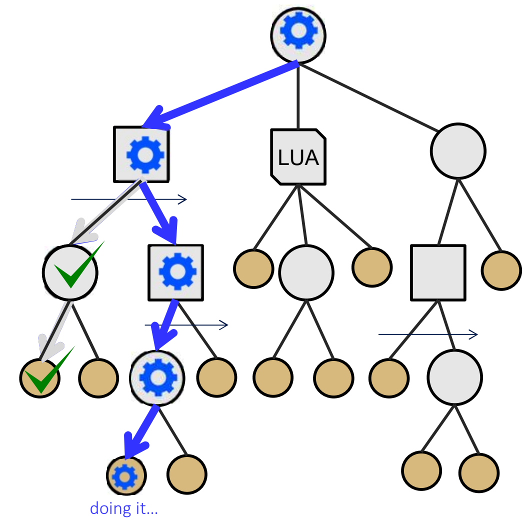
49

Compute behavior: visit tree



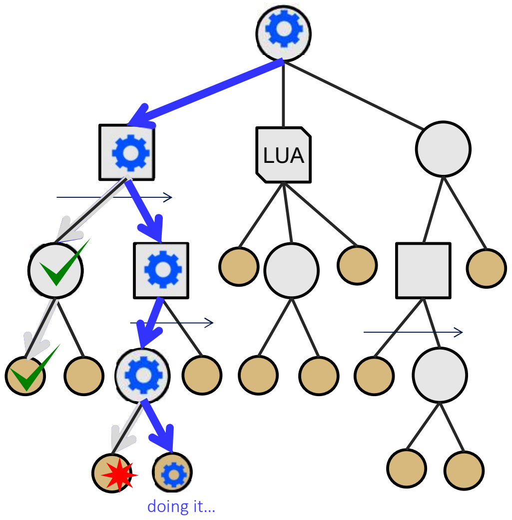
50

Compute behavior: visit tree



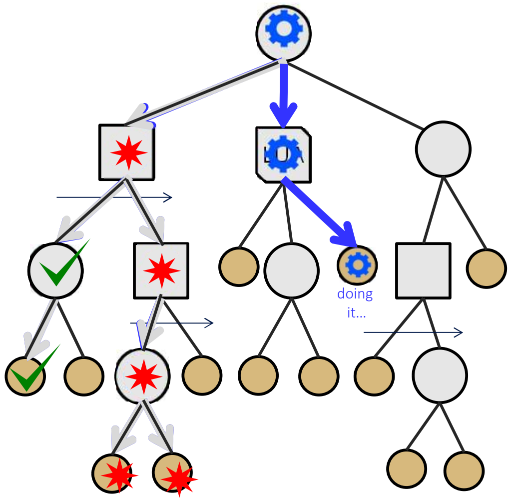
51

Compute behavior: visit tree



52

Compute behavior: visit tree



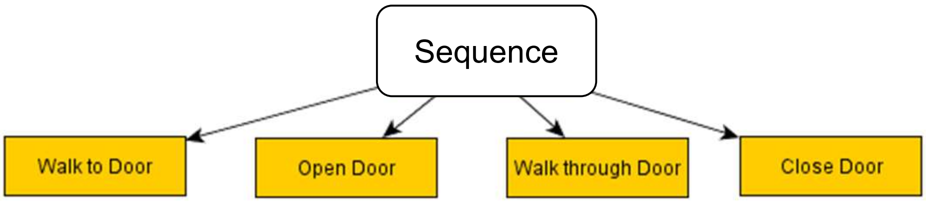
53

Behavior trees: notes

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

54

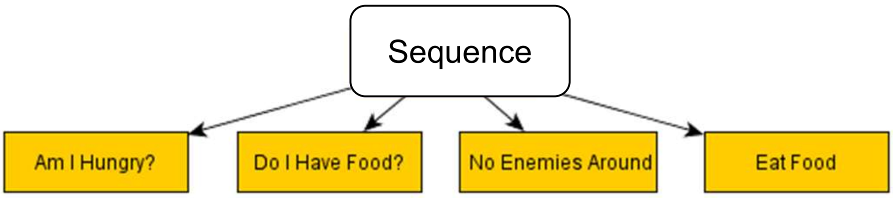
Example 1/3



Example by Chris Simpson (gamasutra)

55

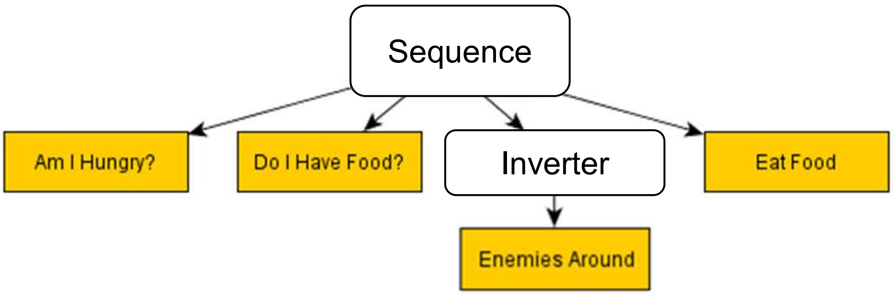
Example 1/3



Example by Chris Simpson (gamasutra)

56

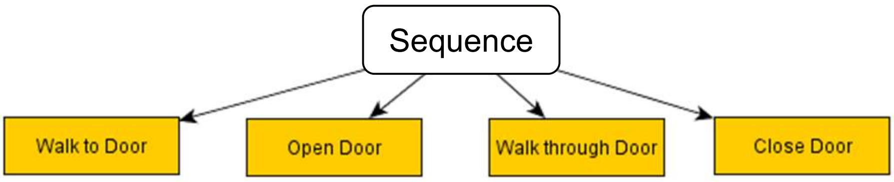
Example 1/3



Example by Chris Simpson (gamasutra)

57

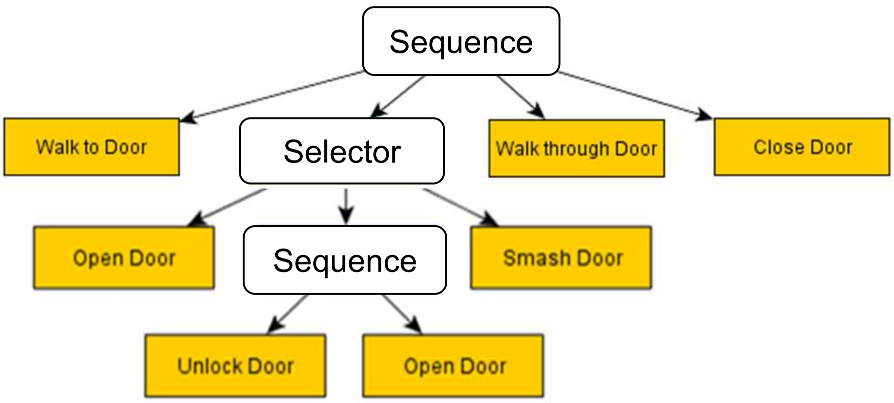
Example 1 - 1/3



Example by Chris Simpson (gamasutra)

58

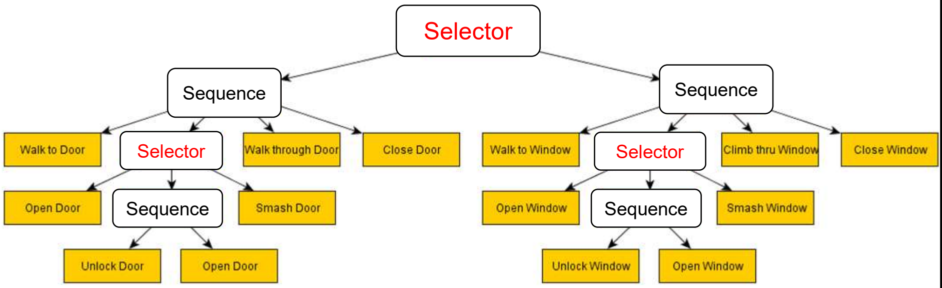
Example 1 - 2/3



Example by Chris Simpson (gamasutra)

59

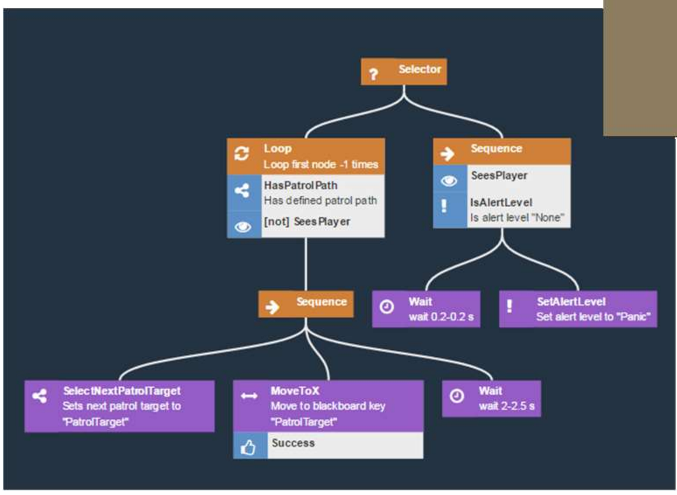
Example 1 - 3/3



Example by Chris Simpson (gamasutra)

60

Example 2



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

61

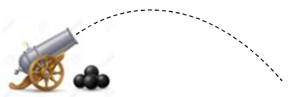
Thinking phase (aka Planning)

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

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
- ...

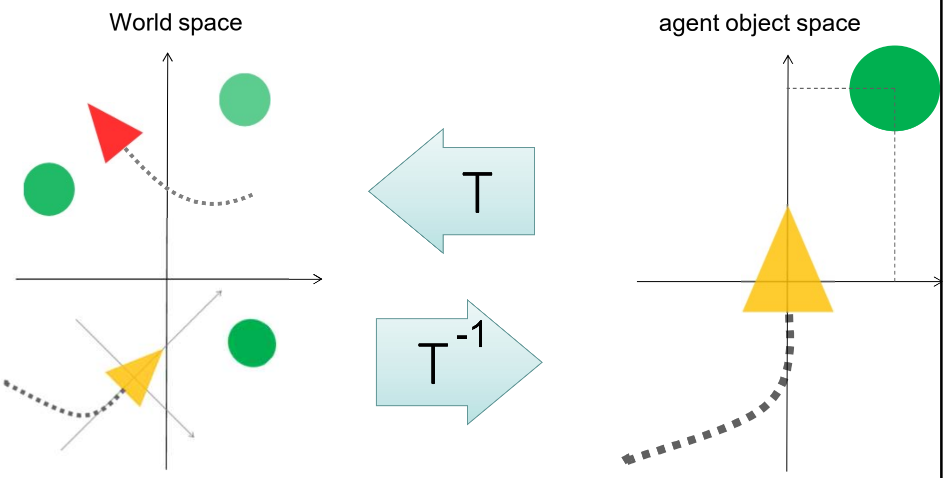


```
vec3 target_pos = target.pos;

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

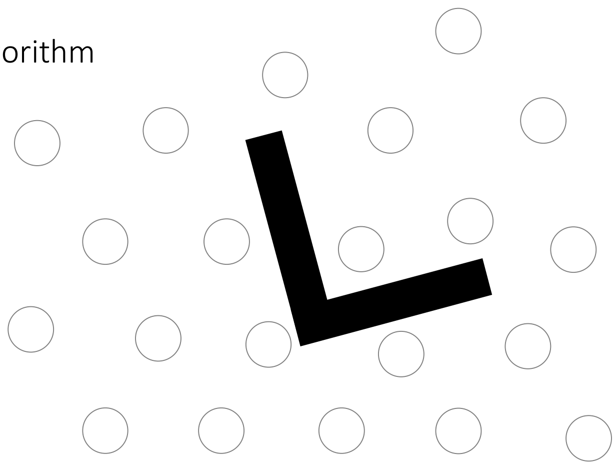
Often easier to think in local object space of the IA



64

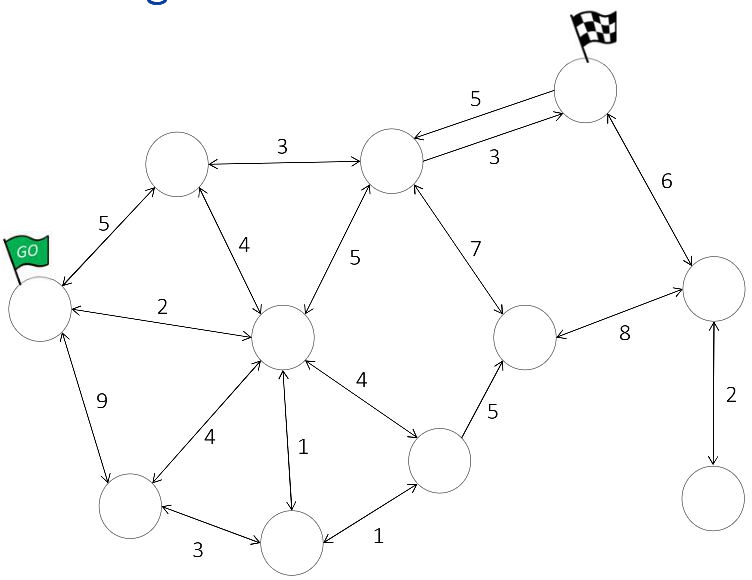
Common lowest level tasks 2/2: Path finding

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



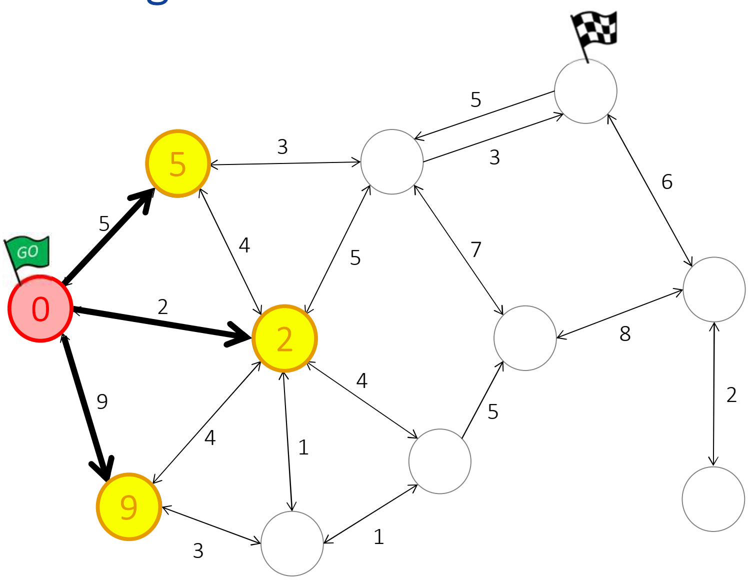
65

Dijkstra algorithm



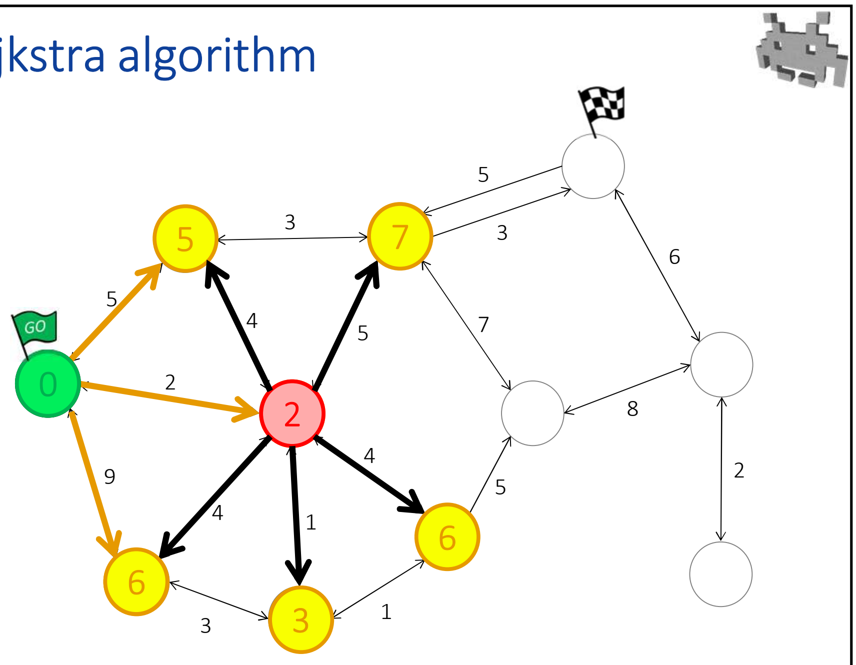
67

Dijkstra algorithm



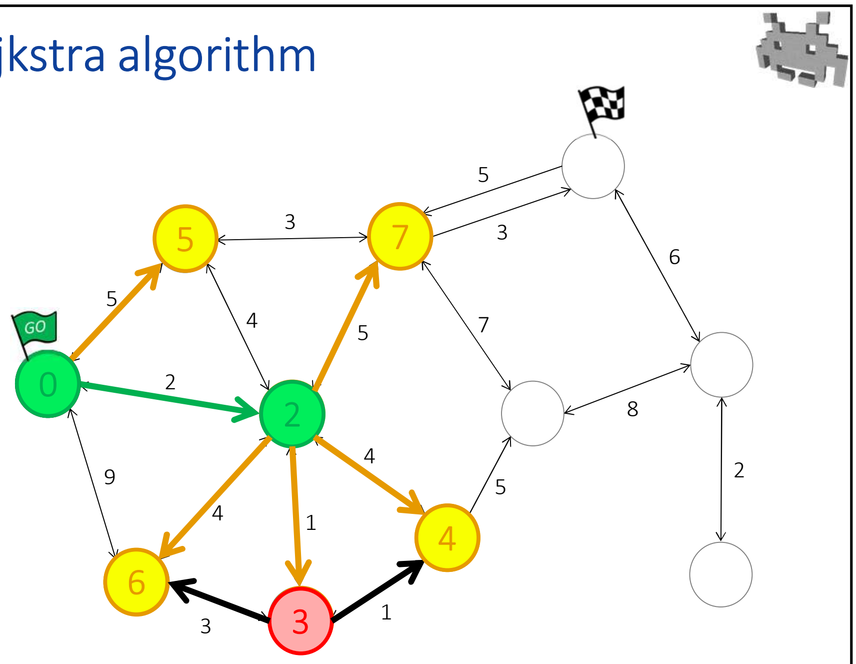
68

Dijkstra algorithm



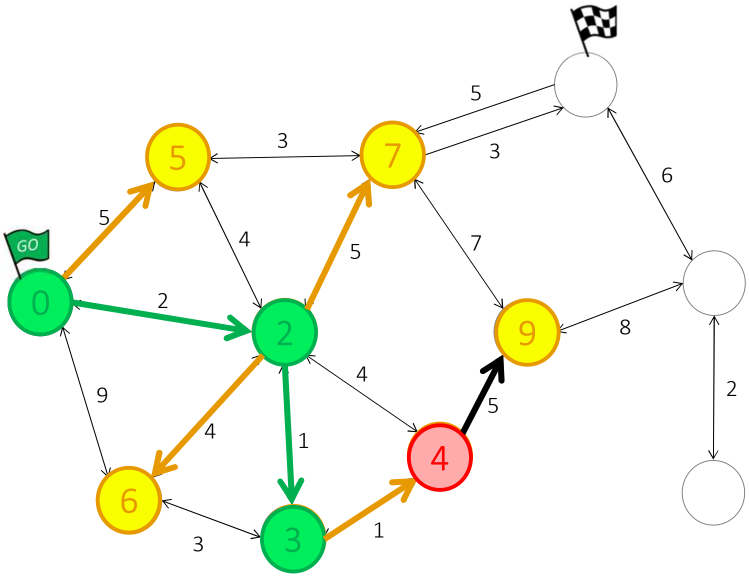
69

Dijkstra algorithm



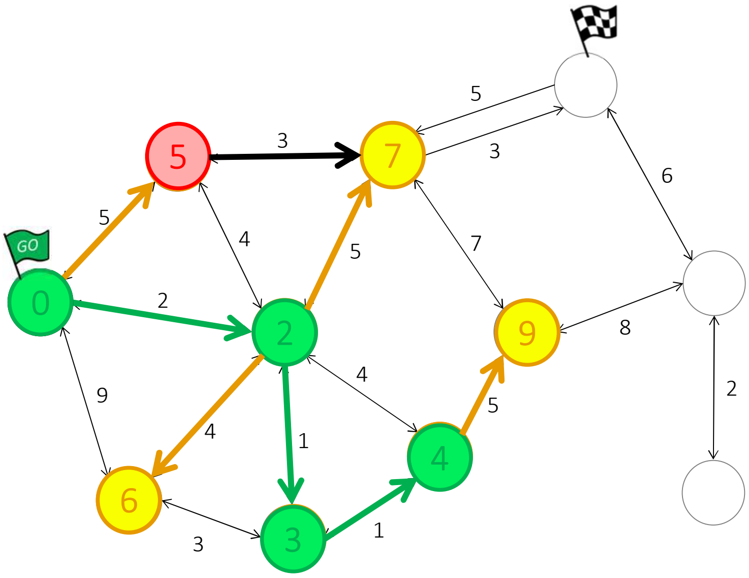
70

Dijkstra algorithm



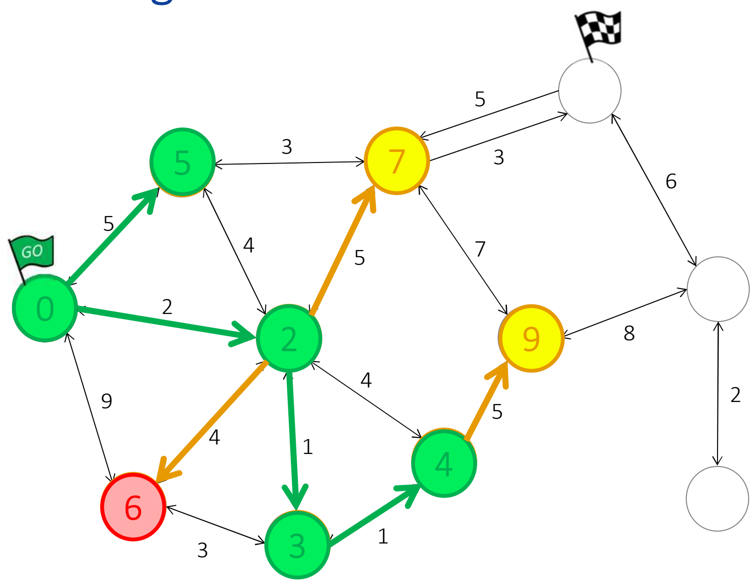
71

Dijkstra algorithm



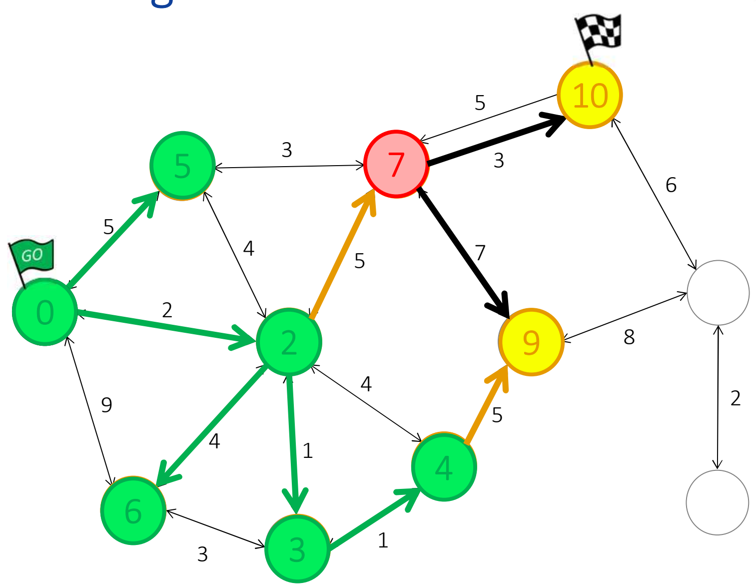
72

Dijkstra algorithm



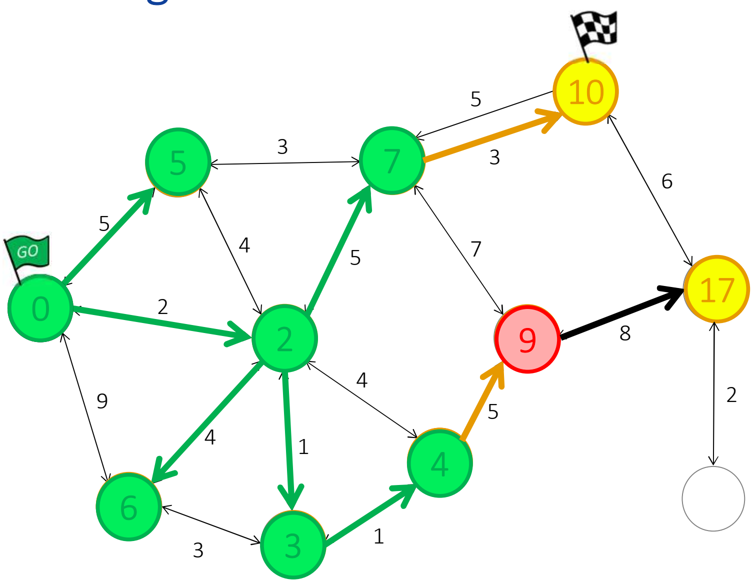
73

Dijkstra algorithm



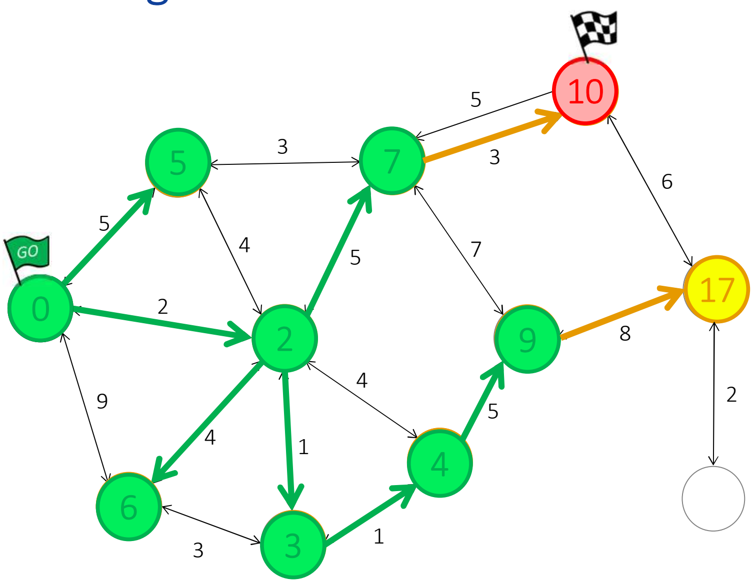
75

Dijkstra algorithm



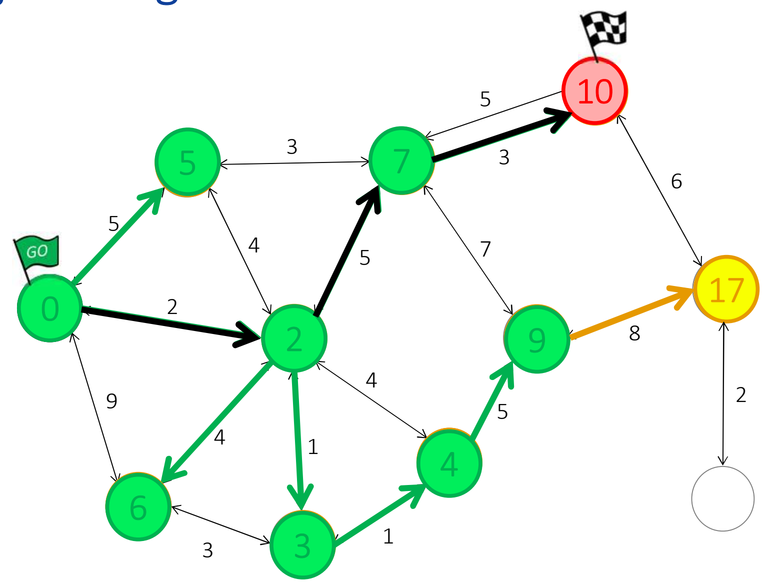
76

Dijkstra algorithm



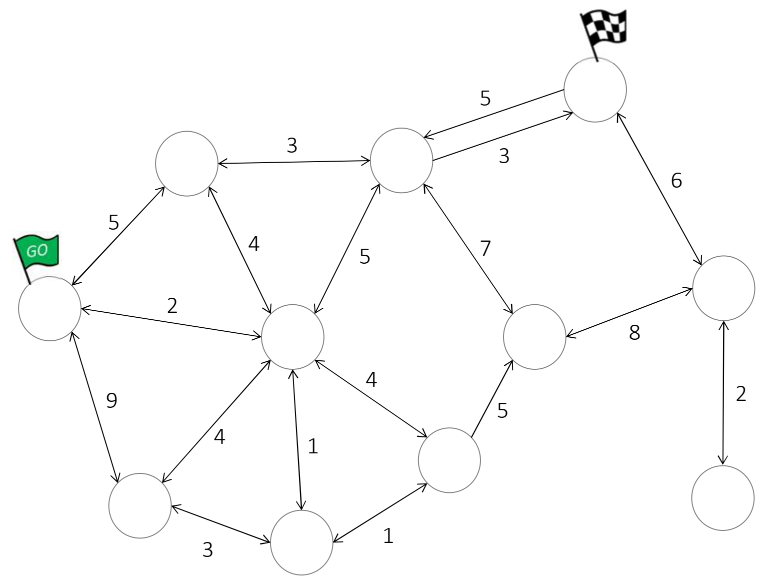
77

Dijkstra algorithm



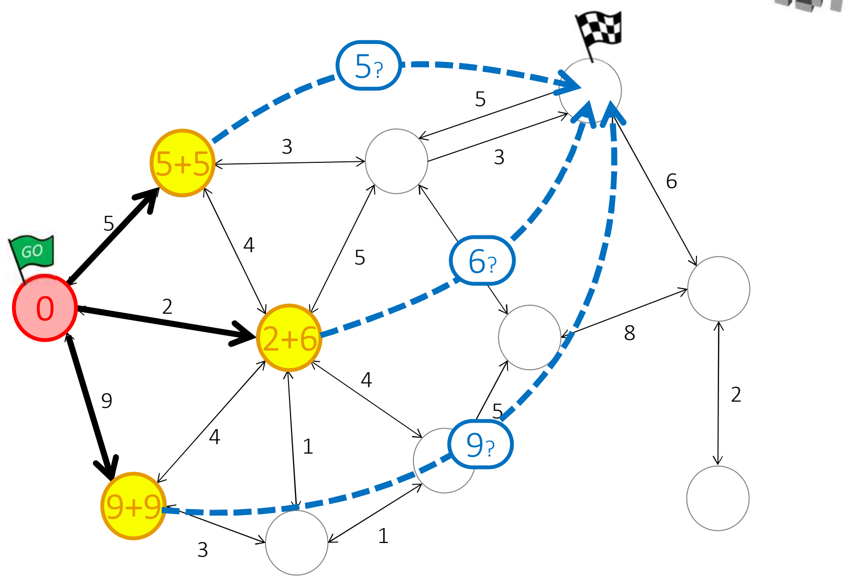
78

A* search



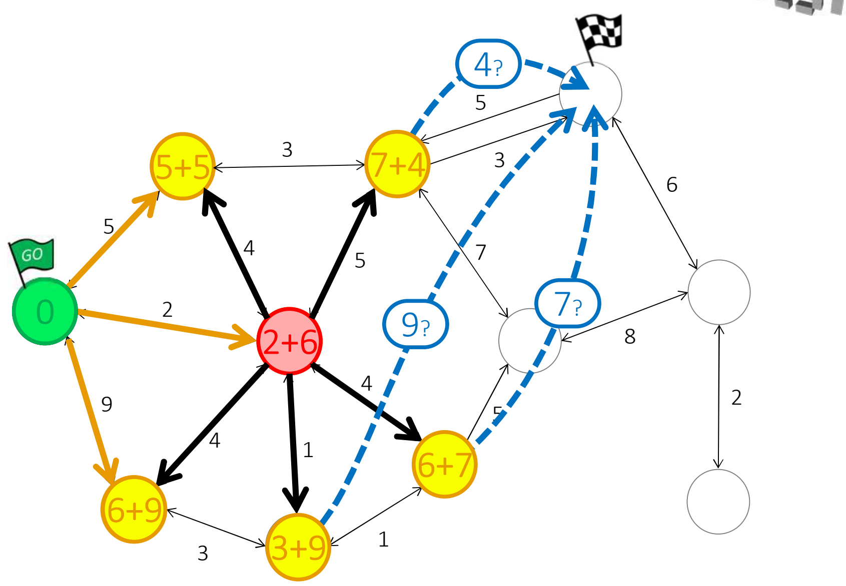
79

A* search



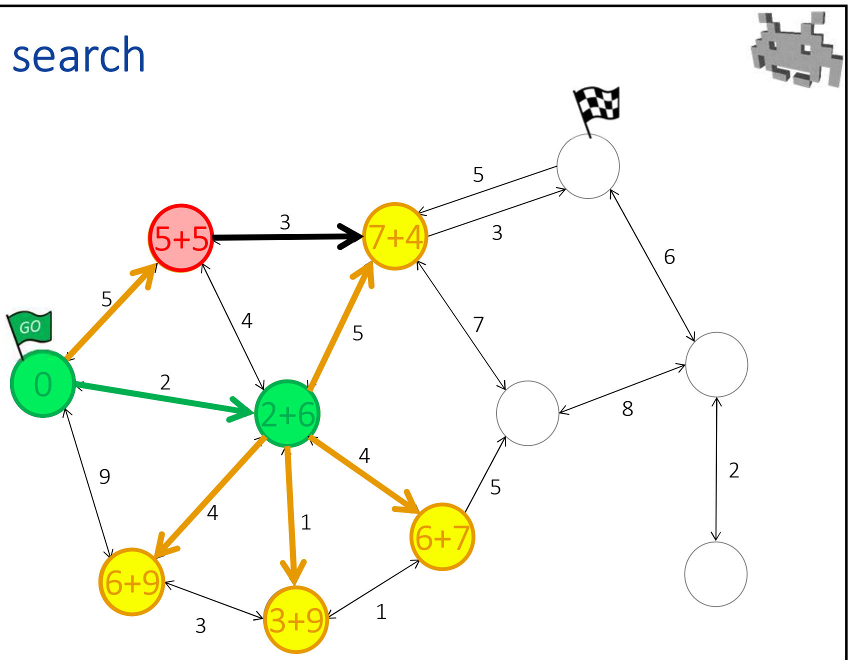
80

A* search



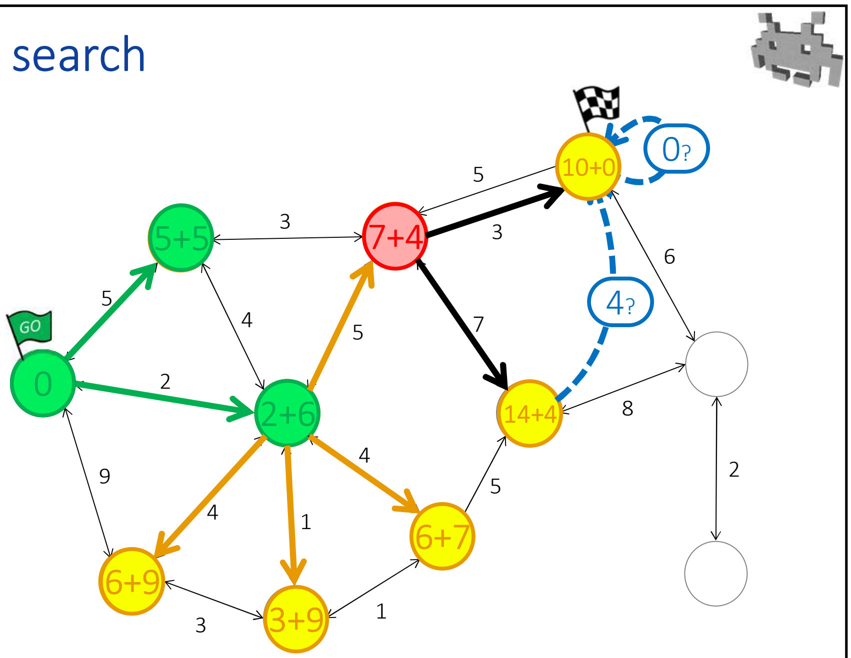
81

A* search



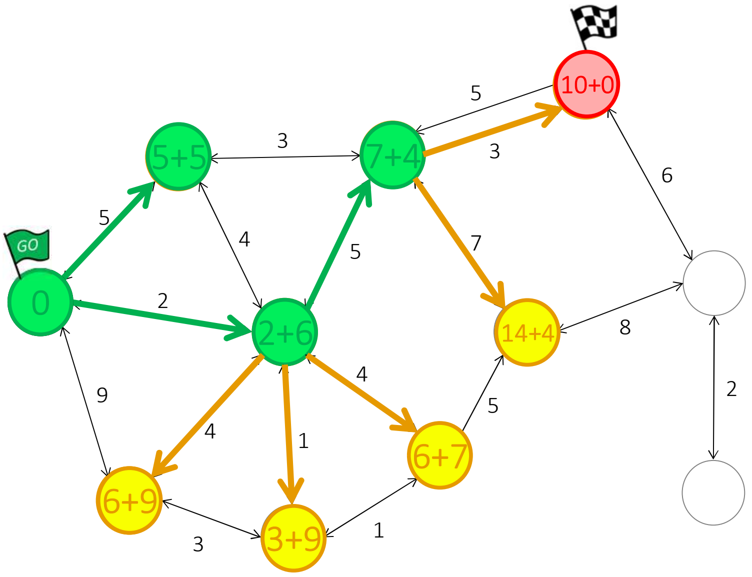
82

A* search



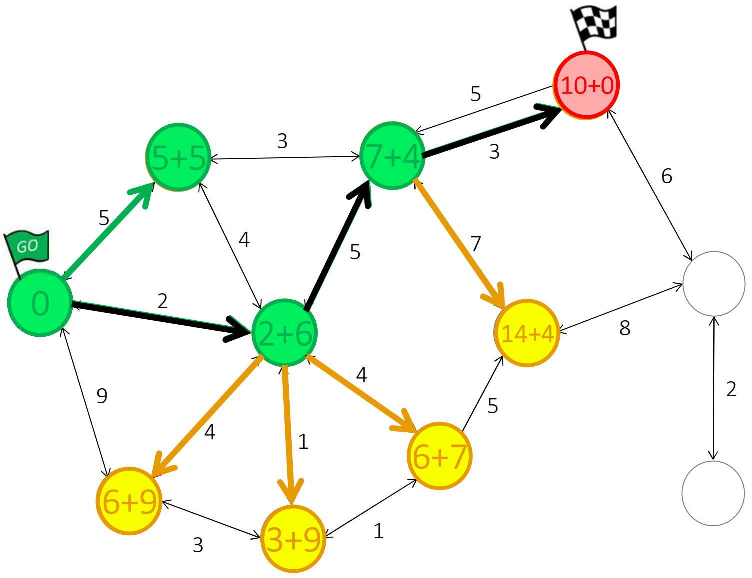
83

A* search



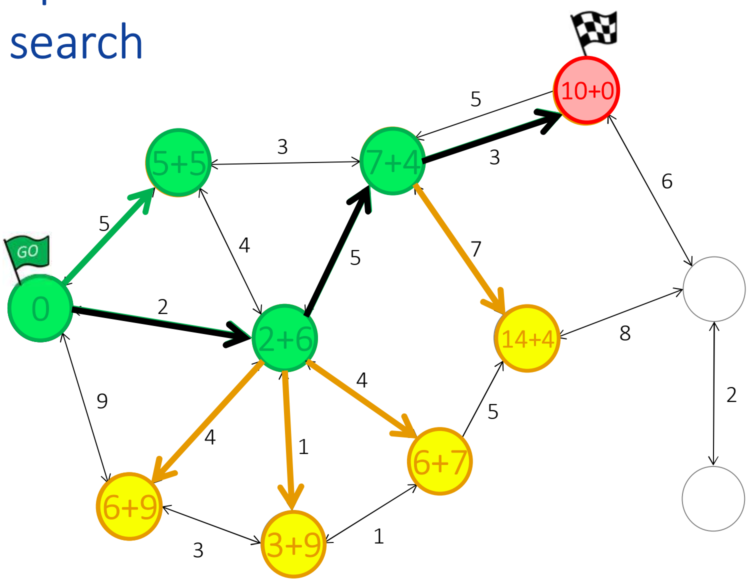
84

A* search



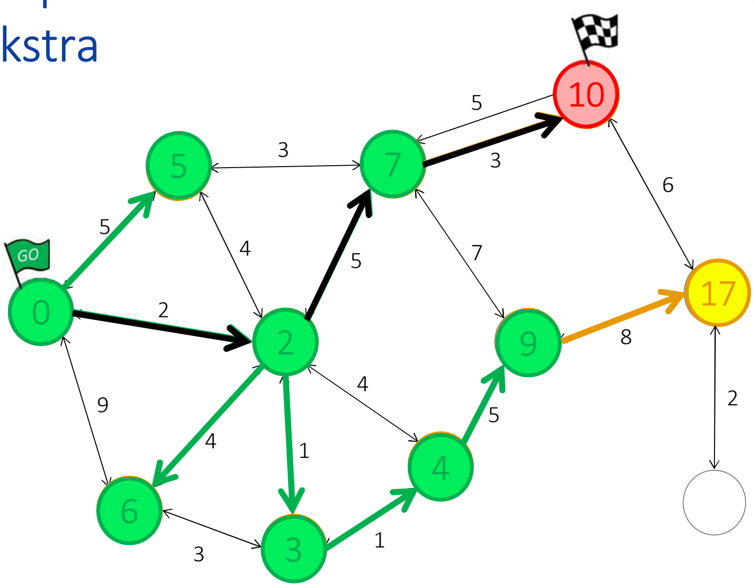
85

Compare: A* search



86

Compare: Dijkstra



87

Input of Dijkstra algorithm: notes.

- 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 bidirectional!)
- a (positive!) cost, associated to each arch
  - e.g., estimated time 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

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 of the 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

6+7

Minimal cost (e.g. time) to go from Start to here

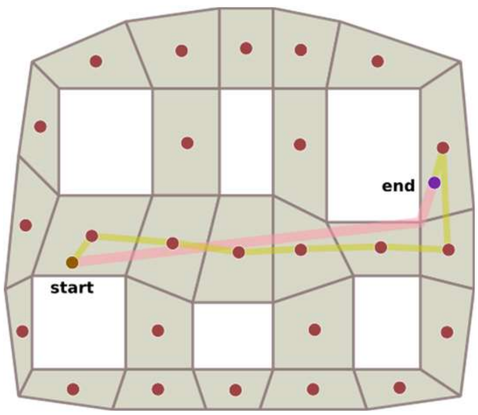
Estimated (minimal!) cost to go from here to Dest

- 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 correct): same as Dijkstra
    - e.g.: perfect estimation (hypothetical case): only explore nodes in optimal path

90

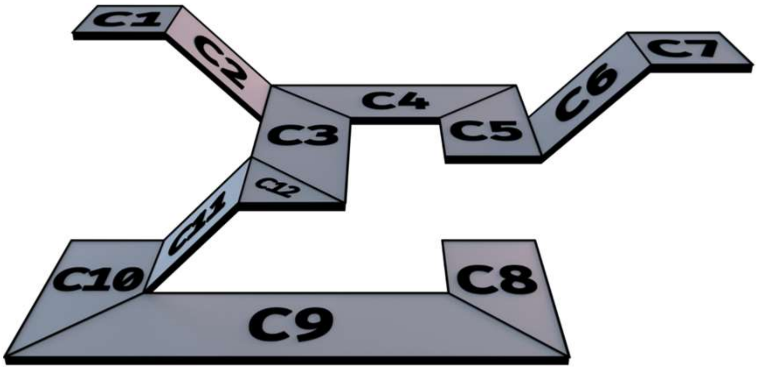
Which graph to use for A* / Dijkstra in a 3D game?

- 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)



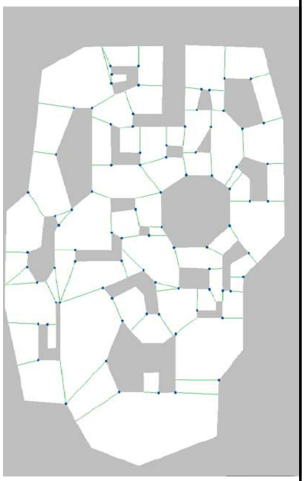
91

Nav Mesh (manually designed)



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

94

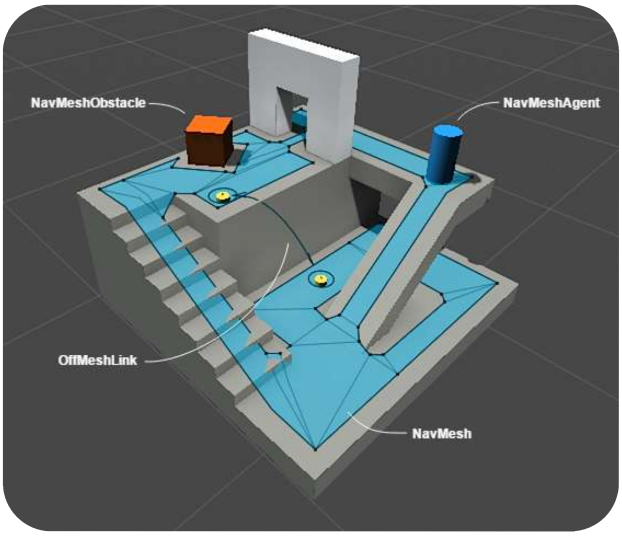
Customizing A* / Dijkstra

- 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 caught in the open")
- Remember: A* needs underestimations
  - Decreasing costs requires care
  - E.g. add teleport doors? Be careful

find out with 3D raycasts

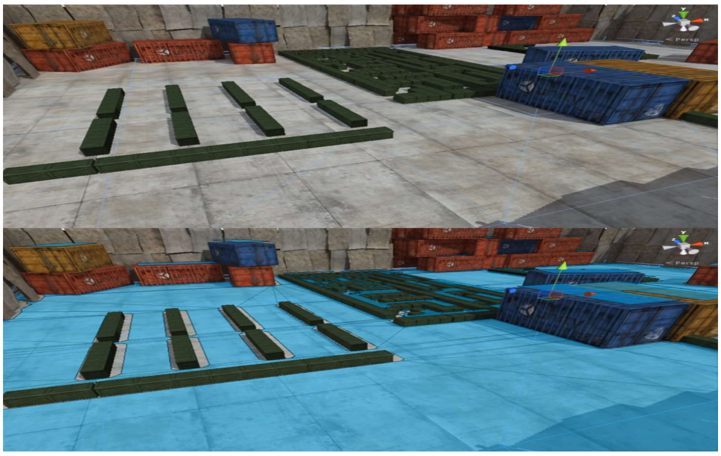
95

Nav Mesh: Unity



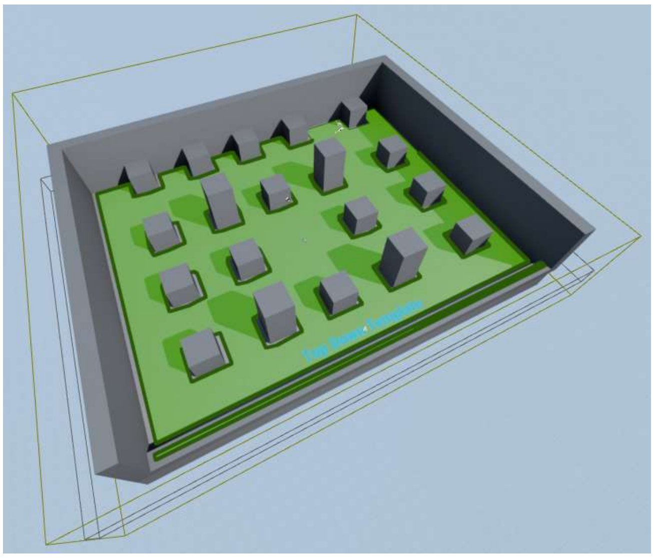
96

Nav-Mesh baking: example in Unity



97

Nav-Mesh baking: example in Unreal



98

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

99

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

100

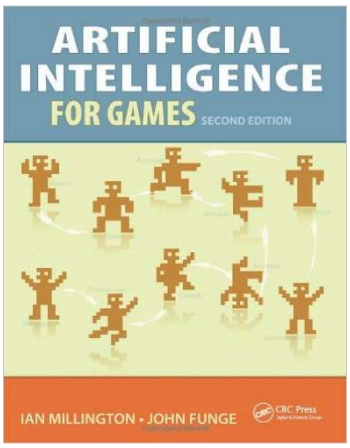
AI support in a game engine: a summary

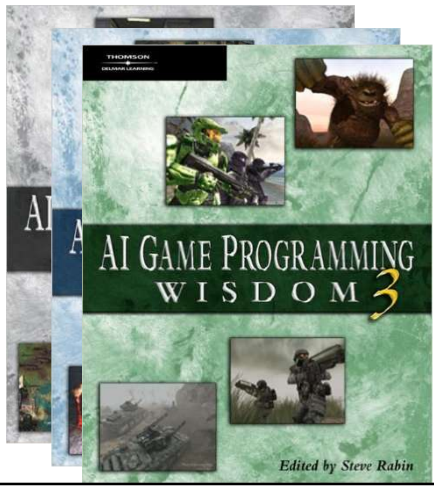
- Assets for (NPC) AI:
  - for *behavior modelling*:
    - Scripts (can well be the only one)
    - FSM
    - HFSM
    - BT
  - for *navigation*:
    - nav-meshes (aka AI-meshes)
  - for *sensing / queries*:
    - hit-boxes, bounding volumes, spatial indexing
    - the same ones used by physic engine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

101

To investigate further

- AI for VideoGames course!
- Books:





102