Lezione 11 30/04/2025

Artificial Intelligence for Games

Si intende di solito qualcosa che ha un ambito applicativo molto limitato, come per esempio gli scacchi.

Tipicamente, l'AI in ambito accademico, quando si vuole sviluppare un meccanismo di AI tipicamente:

- va a trovare un problema
- si cerca di capire come questo problema è affrontato in letteratura
- se ci sono problemi, trovare altre strade e cercare di risolverli
- pubblicare articoli scientifici con le migliorie rispetto a soluzioni precedenti

quindi si ignorano problemi come **risorse computazionali**, si è interessati più ad un problema teorico. La fase di ottimizzazione arriva dopo.

I giochi sono un ambiente che può essere sfruttato per risolvere problemi, virtualizzandoli. Il successo può essere poi traslato nel mondo reale.

Nei giochi però l'obiettivo è il tempo reale, non possiamo avere algoritmi eccessivamente pesanti. L'importante è che faccia quello che deve fare dal punto di vista del giocatore. L'AI deve anche essere molto stabile e non deve essere troppo forte altrimenti il giocatore diventa frustrato.

Può essere usata per la **generazione procedurale** di livelli (risolvibili), terreno, musica, modelli, scene, nemici...

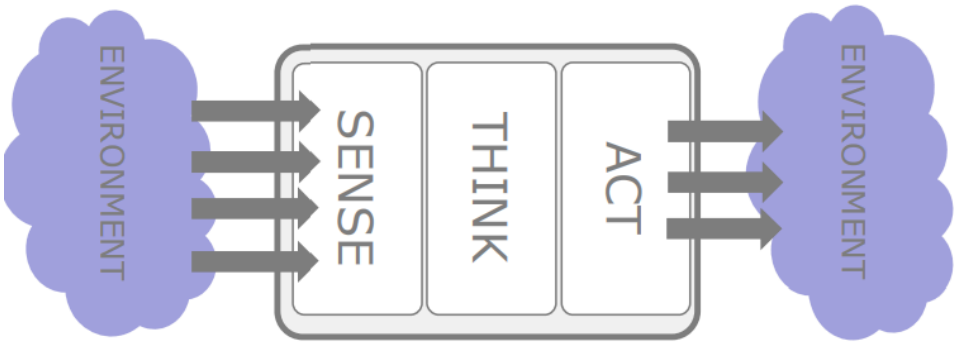
Può essere usata anche per cambiare la difficoltà in modo automatico.

L'uso principale che vedremo noi è quello di definire il **comportamento degli NPC**, vogliamo che siano personaggi plausibili, intelligenti ma non troppo, in certi casi magari anche prevedibili.

- Rather, NPC behavior often needs to be:
  - intuitable / predictable
  - learnable
  - understandable
  - story driven
  - exploitable (interesting to exploit)
    - elicit interesting strategies by the players
    - make a given strategy rewarding

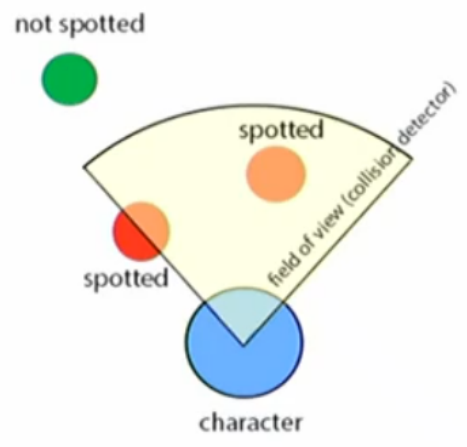
Inoltre, gli algoritmi che vengono sviluppati, non sono ottimali perchè questo richiederebbe computazione eccessiva per un gioco real time, e spesso non sono realistici perchè magari potrebbero sfruttare informazioni che normalmente non sarebbero accessibili, che però sono necessarie al funzionamento dell'algoritmo.

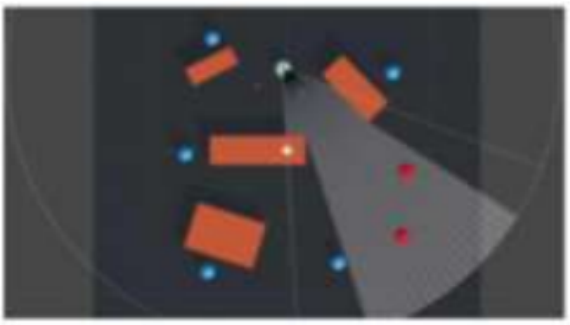
Possiamo chiamare gli NPC degli **agenti interattivi** (non intelligenti) perchè rispondono a quello che succede nell'ambiente esterno.



Le informazioni in input come quello che vede e sente, va ad attivare una serie di azioni che deve compiere.

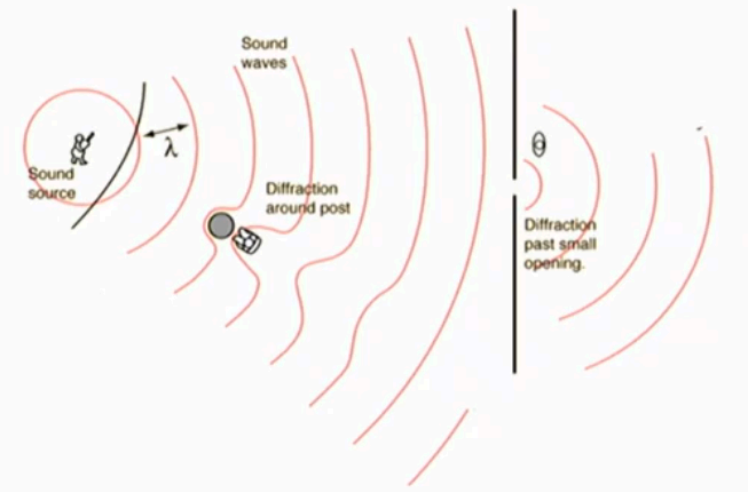
Percepire l'ambiente significa ottenere delle informazioni vicine al soggetto, tipicamente per fare questo si usano dei "sensori". Per esempio se ci interessa capire se c'è un nemico nelle vicinanze si può usare la vista, e si modella il campo visivo definendo questi coni di vista.





Si devono tenere in conto anche oggetti che possono occludere.

Nel caso dell'audio invece si definisce un volume sferico della regione di interesse, e all'intorno di quel volume vengono generate delle onde sonore (invisibili) che possono essere intercettate da un altro personaggio e dal suo volume sferico di udito.



Se il personaggio ha percepito qualcosa, allora si passa alla fase di **ragionamento**. Quindi ci deve essere un algoritmo che prende delle decisioni, che dipendono dall'obiettivo dell'agente.

Possono anche essere algoritmi ad hoc, in base all'obiettivo.

Un personaggio di solito ha un macro obiettivo, e anche una serie di sotto obiettivi che servono a raggiungere quel macro obiettivo.

Per esempio qui abbiamo un personaggio spaventato. Per esempio il macro stato cambia in “devo scappare”, quindi deve cambiare in una versione di se stesso che si comporta in un modo diverso. Poi magari vuole nascondersi, questo è un secondo obiettivo, deve trovare un posto dove nascondersi. Il terzo obiettivo può essere il come arrivare al rifugio per nascondersi, questo è un problema di path finding.

◆ Goals of the AI: typically, hierarchical logic

- Hi-Level Goals
  - “I’m escaping”
- Mid-level Goals
  - “I’m going for that hiding spot”
- Lowest-level Goals
  - “I’m passing through here” (escape route)
- Acts!
  - Actual movements + “panicked-run animation”

Un altro esempio di un cecchino:

◆ Goals of the AI: typically, hierarchical logic

- Hi-Level Goals
  - “I’m sniping”
- Mid-level Goals
  - “I’m going for that enemy soldier”
- Lowest-level Goals
  - “I’m aiming at this coordinates (x,y,z)” (the center of his exposed head)
- Acts!
  - Crouched-aim animation + IK to re-orient rifle

◆ Goals of the AI: typically, hierarchical logic

- Hi-Level Goals
  - "I'm Patrolling"
- Mid-level Goals
  - "I'm going to my 3rd nav-point"
- Lowest-level Goals
  - "I'm passing through here" (find route to it – navigation)
- Acts!
  - actual movements + "alerted-walk" animation

Gli **obiettivi di alto livello** sono di solito gestiti da automi a stati finiti, per cambiare lo stato.

Gli **obiettivi intermedi** sono delle azioni con eventi scriptati.

Se poi il personaggio deve muoversi, subentrano degli algoritmi specifici come quello per trovare il cammino minimo, evitare gli ostacoli...

◆ Goals of the AI: typically, hierarchical logic

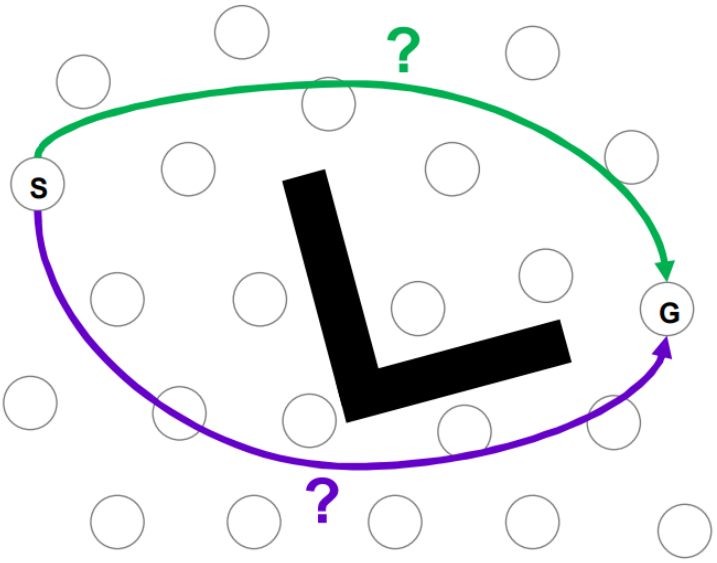
- Hi-Level Goals
  - update: not very often
  - ⟵ E.g. Finite State Machine (FSM)
- ...
- Mid-level Goals
  - update: more often
  - ⟵ E.g. Scripts
- ...
- Lowest-level Goals
  - solving low level tasks
  - ⟵ E.g. Scripts, task-specific algorithms
- Acts!

Pathfinding

Di solito si va a definire nella scena una griglia di punti approssimabile con un grafo connesso, dove ciascun punto determina una posizione che può raggiungere il personaggio. La questione da risolvere è quella di capire come passare da un punto all'altro, tenendo conto dei costi a livello di tempo, etc.

Ci sono diversi algoritmi:

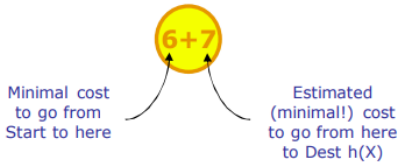
- Different algorithms available
  - Johnson's Algorithm
  - Floyd-Warshall Algorithm
  - Shortest Path Dijkstra
  - Shortest Path A*
  - ...



(qua ci sono un tot di slides sull'algositmo di Dijkstra)

- Dijkstra not efficient enough
  - visits too many nodes
  - explores paths which are obviously wrong
  - it's greedy, only guided by distance from Start

- Use 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!
      – underestimation ok, overestimation NOT OK
    - E.g.: simple Euclidean distance (disregarding obstacles!)

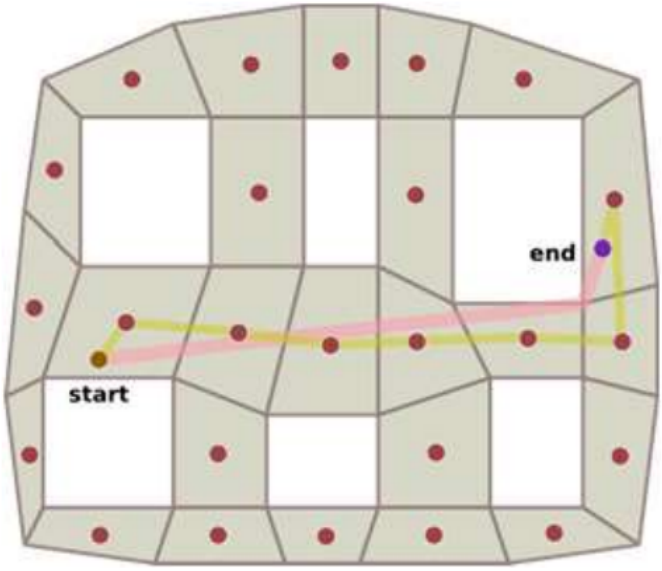


- Output: *still* the optimal path
  - as long as the estimator never overestimates costs
  - the better the estimations, the quickest the algorithm
    - e.g.: if $h(X)$ is always 0 (technically, still correct): A* does the same as Dijkstra
    - e.g.: perfect estimation (hypothetical case): A* only explores nodes in optimal path

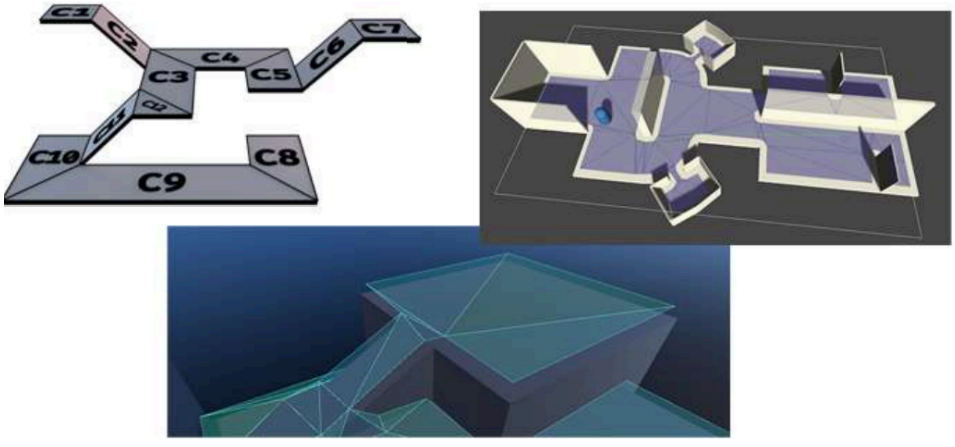
A* invece fa una stima che deve essere veloce.

Il costo dei nodi ha 2 parti, quello a sinistra è quello effettivo per raggiungere il nodo, mentre invece quello a destra è una stima per raggiungere il punto finale.

Come possiamo usare A* / Dijkstra in un gioco? Il terreno è scomposto in pezzi di mesh, in diverse regioni. Ciascuna regione costituisce l'equivalente del nodo di prima. Il passaggio da una mesh all'altra è l'arco.



le navigation mesh vanno definire da un designer, che definisce i pavimenti calpestabili, marca le mesh e da dei pesi.



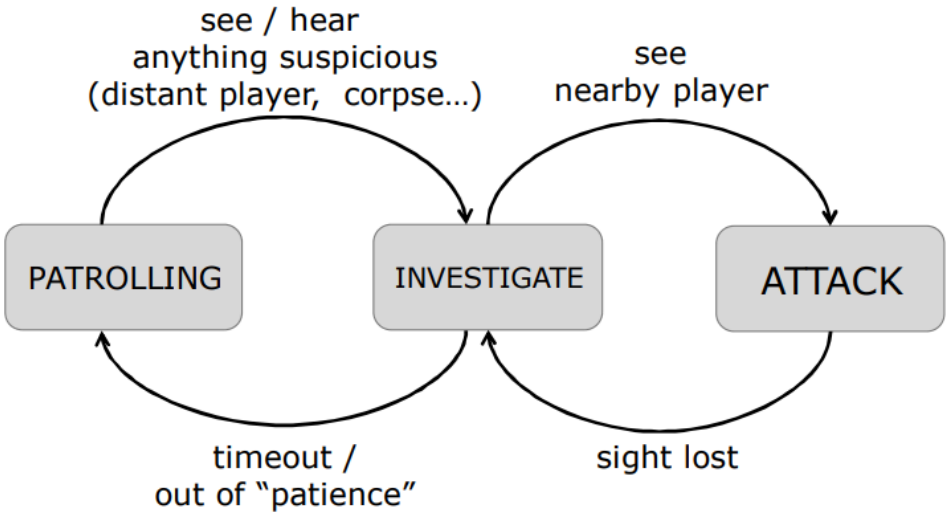
Oppure esistono degli approcci semi-automatici. Sono algoritmi che creano (**bake**) le mesh.

Il costo per spostarsi da una mesh all'altra può anche essere definito in base al tipo di terreno (acqua, sabbia...) oppure anche l'inclinazione (salita, discesa...)

Automi a stati finiti

Un'altra tecnica usata per modellare il comportamento di un agente sono gli **automi a stati finiti**, ci permettono di tenere traccia del cambiamento di stato.

Per esempio una sentinella può avere questi stati:



Tipicamente gli stati sono una serie di condizioni, controlli, verifiche (if).

```
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;
}
```

Vedremo questo di più nella lezione di software engineering.

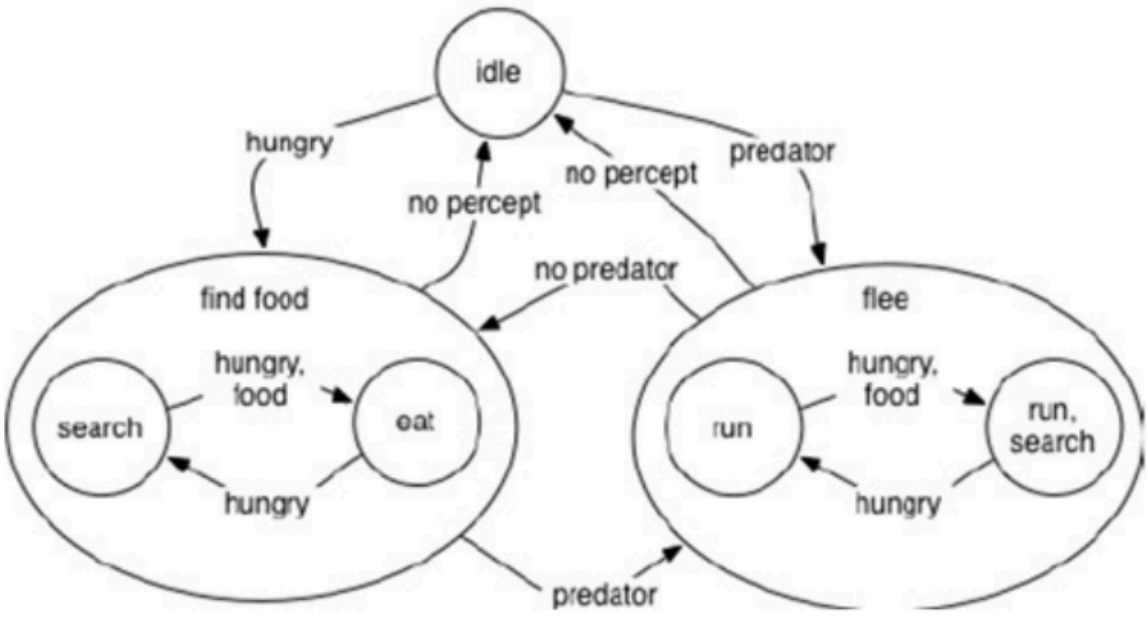
Il problema di questo metodo è che quando abbiamo troppi stati diventa difficilissimo gestire tutte le transizioni da uno stato all'altro.

- ◆ Maintainability
  - Adding and removing states require changing the conditions of all other states that have transition to the new or old one
- ◆ Scalability
  - FSMs with many states become a nightmare of boxes and arrows
- ◆ Reusability
  - Difficult to reuse the same behavior in multiple projects / actors

Sono quindi state inventate alternative

Hirearchical Finite State Machines (HFSM)

L'idea è di dividere un automa a stati finiti complesso in sotto-automi, e quindi potenzialmente riusare parte di questi automi in sistemi diversi. è quindi una scomposizione di un problema in sotto-problemi.



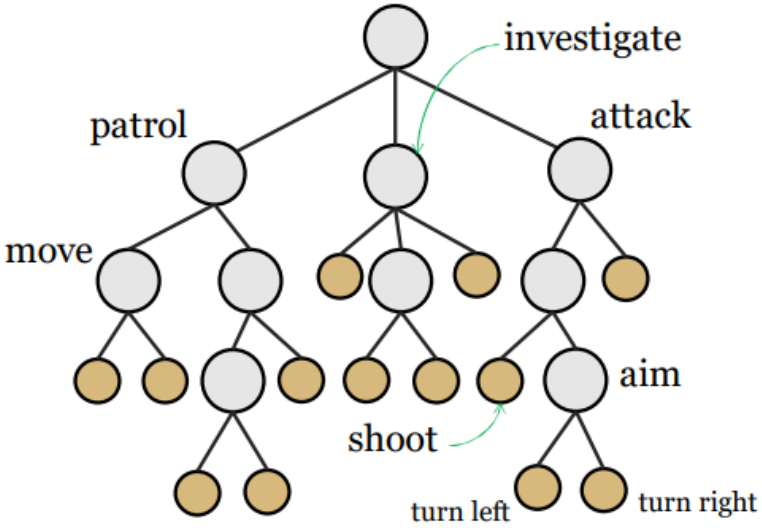
Behavior Trees

Gli automi a stati finiti, quando diventano complessi, sono sostituiti dagli alberi di comportamento.

Bisogna capire qual è il cammino in questo albero, per portare a casa l'obiettivo finale che è quello della radice.

Esempio:

abbiamo un obiettivo iniziale, il personaggio deve fare delle azioni, a seconda del suo stato interno. Per far un attacco deve fare però delle azioni...



L'albero definisce tutte le azioni che potrebbe compiere il personaggio in una certa situazione, e l'obiettivo finale è quale di queste azioni sono percorribili dato lo stato attuale del personaggio e della situazione di gioco?

è una descrizione del comportamento (behavior) del personaggio.

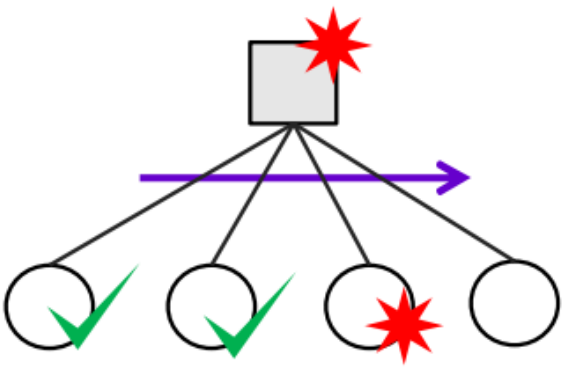
La root è il soggetto. I nodi foglia sono delle azioni concrete, come "devo spostarmi di 10 pixel" o "devo vedere se c'è il nemico" o "devo sparargli".

I nodi intermedi possono essere in 3 stati diversi:

- in progress: per portare a casa quel sotto-problema, c'è ancora qualcosa dopo che sta cercando di risolvere i suoi sotto-problemi.
- succeded: tutta la parte sotto è stata risolta, è stato possibile scegliere certe azioni, certi comportamenti, per trovare la soluzione

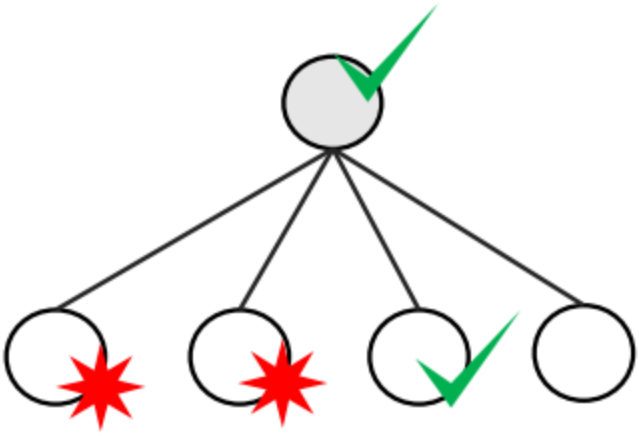
- failed: non è stato possibile trovare una soluzione

I nodi intermedi sono **sequenze**, ho una sequenza di azioni per raggiungere l'obiettivo.



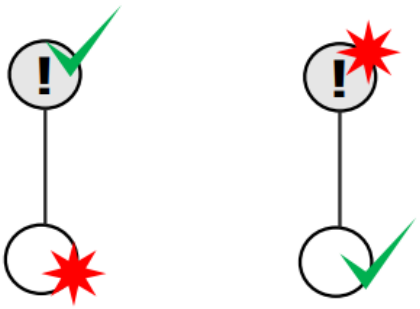
Questo nodo sequenza sarà succeded se tutti i sotto nodi hanno avuto successo, se solo una fallisce allora fallisce anche il nodo intermedio.

Un altro tipo di nodo intermedio è il **selettore**. Significa che ho diverse scelte, scelgo la prima che ha successo.



Se tutte falliscono, allora fallisce anche il nodo selettore.

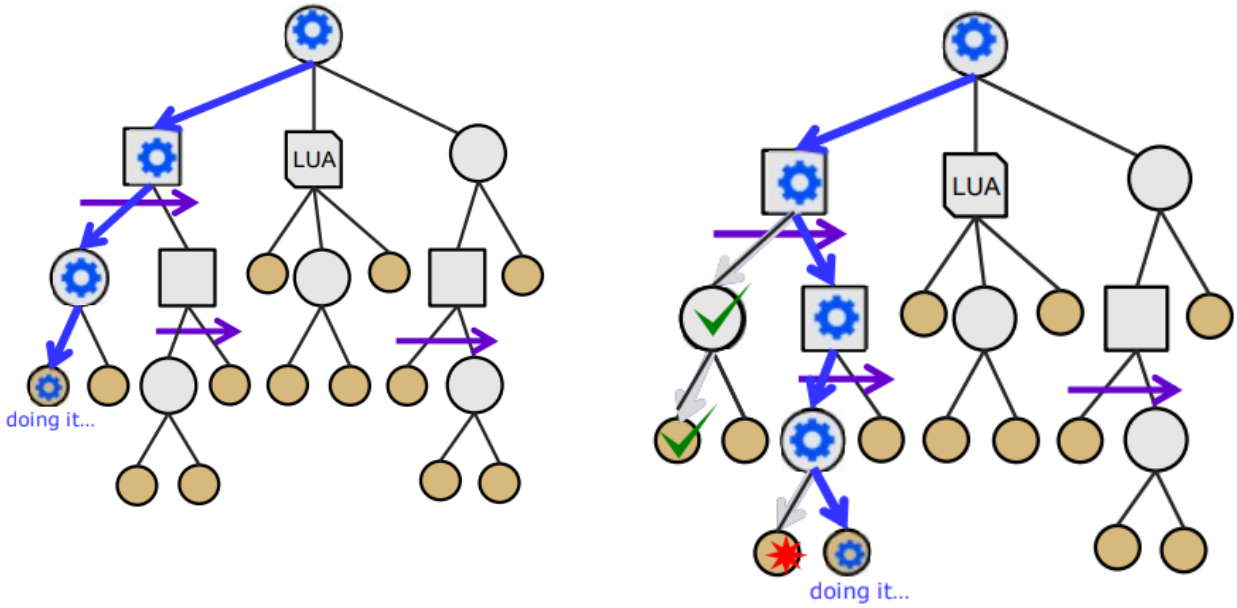
Un altro nodo intermedio è l'**inverter**.

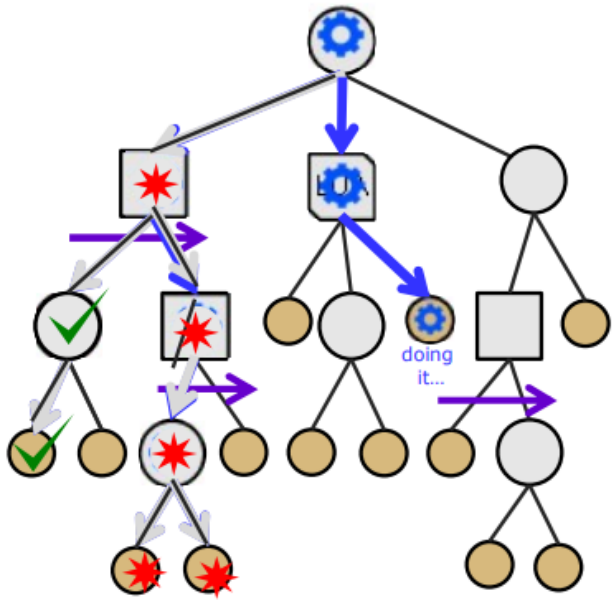


Un altro nodo intermedio è il **repeater**. Significa che la parte sotto deve essere eseguita un tot numero di volte, avendo sempre successo.

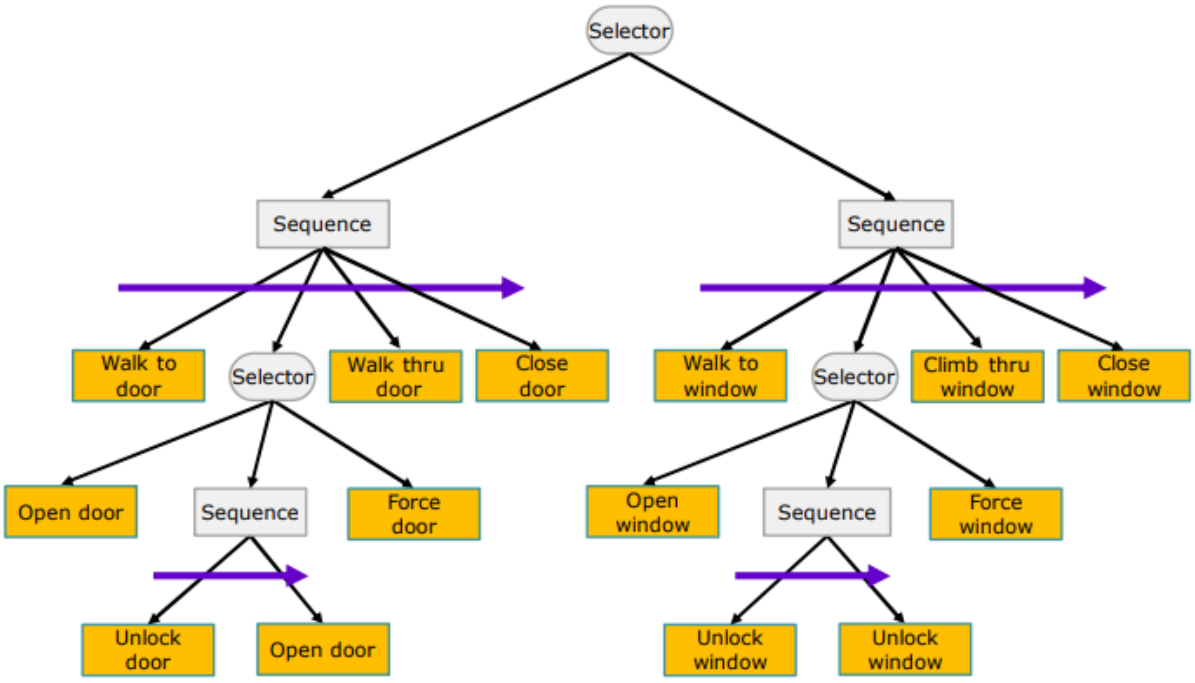
Questi behavioral trees possono anche essere definiti in script, o parte di questi.

root to leaf visit:





Vado avanti così finchè trovo un cammino fattibile, e allora quello sarà quello che deve fare il personaggio. Esempio 2:



Quindi questi alberi servono per prendere decisioni, che poi vengono applicate al personaggio.

Un problema che hanno in comune con gli automi a stati finiti è che dobbiamo scegliere in anticipo tutti i path, tutte le possibilità.

Per questo motivo è stato inventato un altro formalismo: il **GOAP (Goal Oriented Action Plan)**.