09 AI in Games

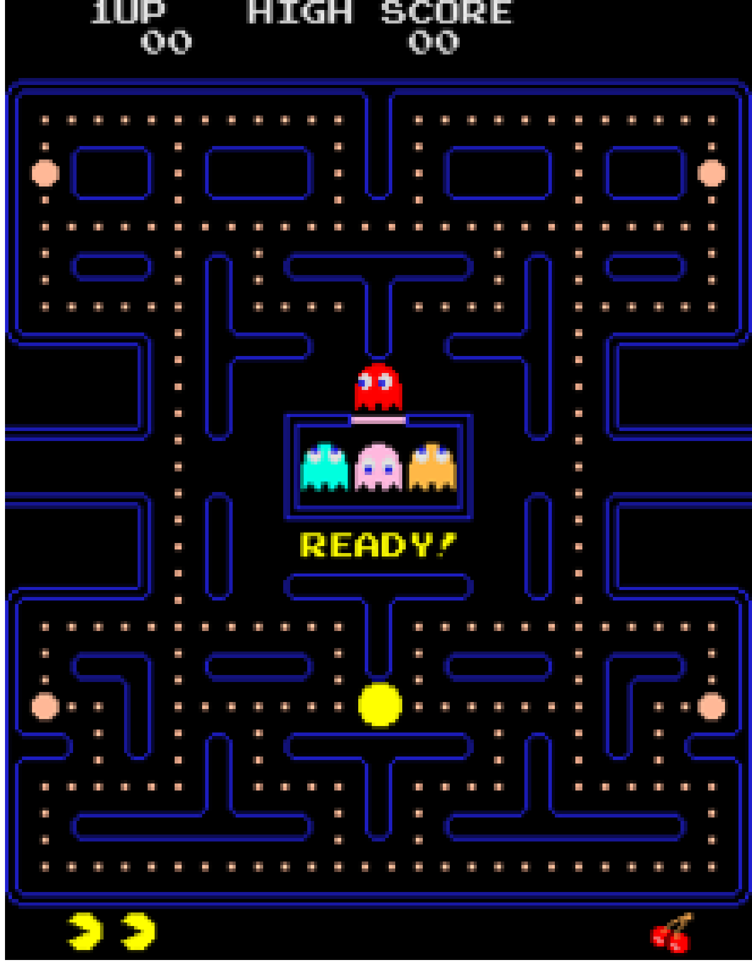
Tvorba a dizajn počítačových hier
Návrh a vývoj počítačových hier
Michal Ferko
21. 11. 2024

Motivation

- We require opponents/teammates in games
- Non-playing characters are usually required to perform some tasks that require AI
  - Following the player, combat, strategic thinking…
- We require something that responds to user actions and imitates the behavior of human players
- Ideally, an AI should fool the players into thinking it is an actual human
  - To keep the immersion level high
  - Turing test

A little history

- Pac-man (1979) was one of the first games with character AI
- Follow the player, run from them, or take a random road
  - Used different states – scatter, chase…
- Ghosts had different personalities
  - New target tiles are determined by personality at every crossroad
- Randomness added a necessary factor – **unpredictable behavior**
- A completely predictable AI is usually easy to beat
  - After a few tries, the player has a detailed model of AI behavior



The Kind of AI in games

- Hacks
  - Games use a lot of hacks, not only in AI
  - Ad hoc solutions to specific problems
- Heuristics – predictions that work most of the time, without guarantees
- Algorithms – the “true” AI
  - Techniques that simulate behavior
    - Usually derived from how real people or animals make decisions and perform actions
- Machine Learning
  - Observe thousands of examples of player behavior
  - Derives its own algorithm on what to do

Hacks - “Game AI is not AI”

- Is the Pac-man example AI?
  - It’s just generating random numbers and performing one of three actions based on the result
  - It’s not an AI technique
- In Sims, a lot of actions are just pre-defined animation sequences
  - There is no actual AI going on
- **More complex AI ≠ better AI**

Heuristics

- Approximate solutions to existing problems
- This is usually how the human mind solves problems
  - I lost my keys ⇒ remember when I last had them and go step by step
  - Simple heuristic – enemy aim is **90%** effective - the chance they will hit you
- Common heuristics
  - **Most constrained**
    - If we have two groups fighting, and one character in one group has a unique weapon that pierces through some unique armor, it should attack a character wearing that armor
  - **Most difficult first** – if you can buy a strong unit, do it instead of getting a few weak ones
  - **Most promising first** – perform the action that will improve your chances the most
    - E.g. Chess AI

Algorithms

- Some AI actions still require other algorithms
  - Movement of characters
  - Decision making
  - Tactics or strategy
  - Analysis of game state and future game state

Academic AI vs. Game AI

- Academic AI – be as smart as possible
  - Solve the problem as efficiently and precisely as possible
    - E.g. 99.99% guarantee required that a traffic camera identifies license plates correctly
- Game AI - **make the player have fun**
    - Provide interesting challenges for the player
    - React to the player
    - Be predictable enough for the player
    - Be believable enough to keep the illusion of a real being in control

Game State Analysis

- Process input data (game state) to simplify the decision process
- This is usually called **sensing** - you create senses for the AI
  - Vision
  - Hearing
  - Touch
  - Smell?
  - …

Gameplay AI is a 3-step process

1. **Sense** – what can I see/hear/feel
    - Some senses can cheat!

2. **Think** – consider what I do next based on what I am sensing
    - Process data from senses and decide what to do

3. **Act** – perform actions I have decided to do
    - Walk to a destination
    - Attack someone
    - Use an item
    - …

AI Difficulty

- For some games, creating skilled AI is simple
  - Counter-strike: aim & shoot to kill instantly
- Difficult ≠ Fun
  - “Dumb down the AI” – sometimes misses, isn’t perfectly efficient
  - Rubber-banding – adjusting to the player to always offer a reasonable challenge
    - Trying waiting for 30 seconds in a racing game
- For more complex games, creating challenging AI is a complex task
  - StarCraft 2 – there are hundreds of options of what to do at any moment
    - Most difficult AI bots cheat, since the AI cannot compete with more skilled players
    - **Sense** & **Act** is easy, **Think** is the difficult part
- Chess has a much smaller possibility space
  - Can simulate 15-20 moves into the future

An example: Sims

- When does a sim become hungry?
- What will the sim do if they really have to go to the bathroom?
- These are several competing systems that are weighted
  - Changing these weights alters the behavior of the sim
- Final decisions about the next are strongly affected by the weights
- When a sim is about to pass out of hunger
  - Getting food becomes top priority
  - AI navigation will help move them to the nearest food source
  - The weight of the hunger system represents the desire to get food

AI types in games

- **Hard-coded** – deterministic behavior
  - If player health above 80, fire gun at player
- **Randomization** – randomized behavior
  - If player health above 70-80, fire gun
  - Less predictable, more realistic
- **Weighted randoms** – every possible next step is given a weight
  - Some of the possibilities will happen more often than others
  - Weights control what happens in the end

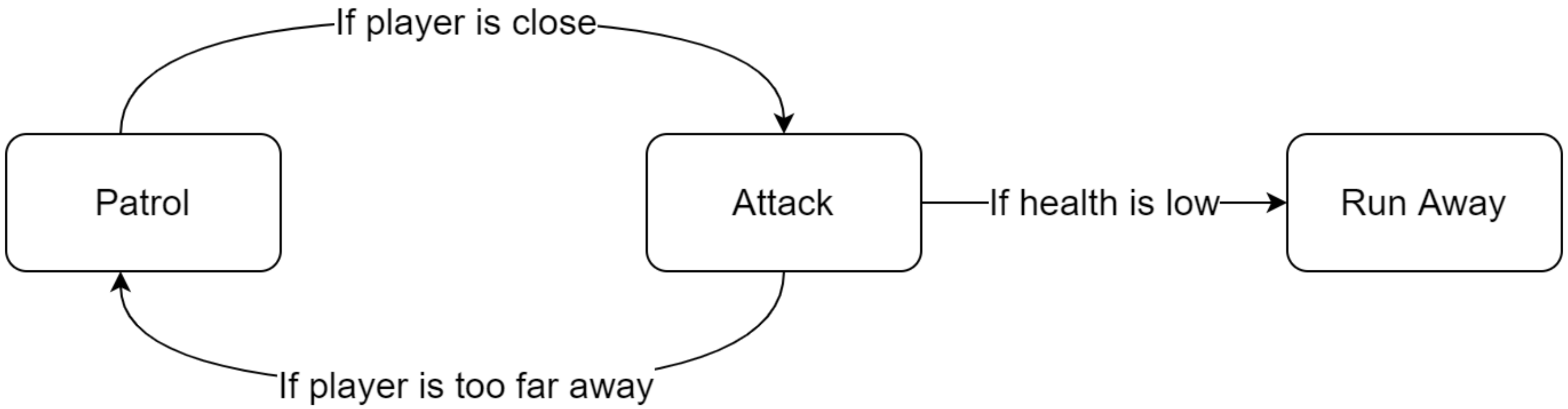
Weighted randoms example

- We have a creature that can perform 3 actions
- We assign weight to these actions
  - Attack 60%, Cast spell 30%, Run away 10%
  - Very similar to the Sims example, but those weights change over time

```
X = RandomFromRange(0, 99);
if (X < 60)
    Attack();
else if (X < 90)
    CastSpell();
else RunAway();
```

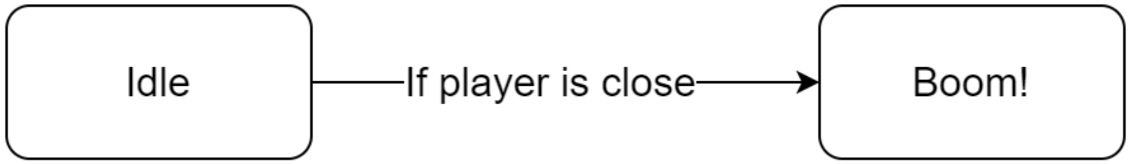
Finite State Machines (FSM)

- Several states an entity can be in (sleeping, wary, attacking, running away…)
- We define rules to transition from one state to another
- There does not have to be a transition from each state into every other state
  - The transition **running away** ⇒ **sleeping** does not make sense
- We define when transitions from one state to another should happen



Finite State Machines (2)

- Based on the current state of the entity, perform an action
  - **Patrol** ⇒ walk through corridors along a pre-defined path
  - **Attack** ⇒ move continuously towards the player while shooting
- A proximity mine can use the same “proximity” check as the guard
  - If near, move to state X

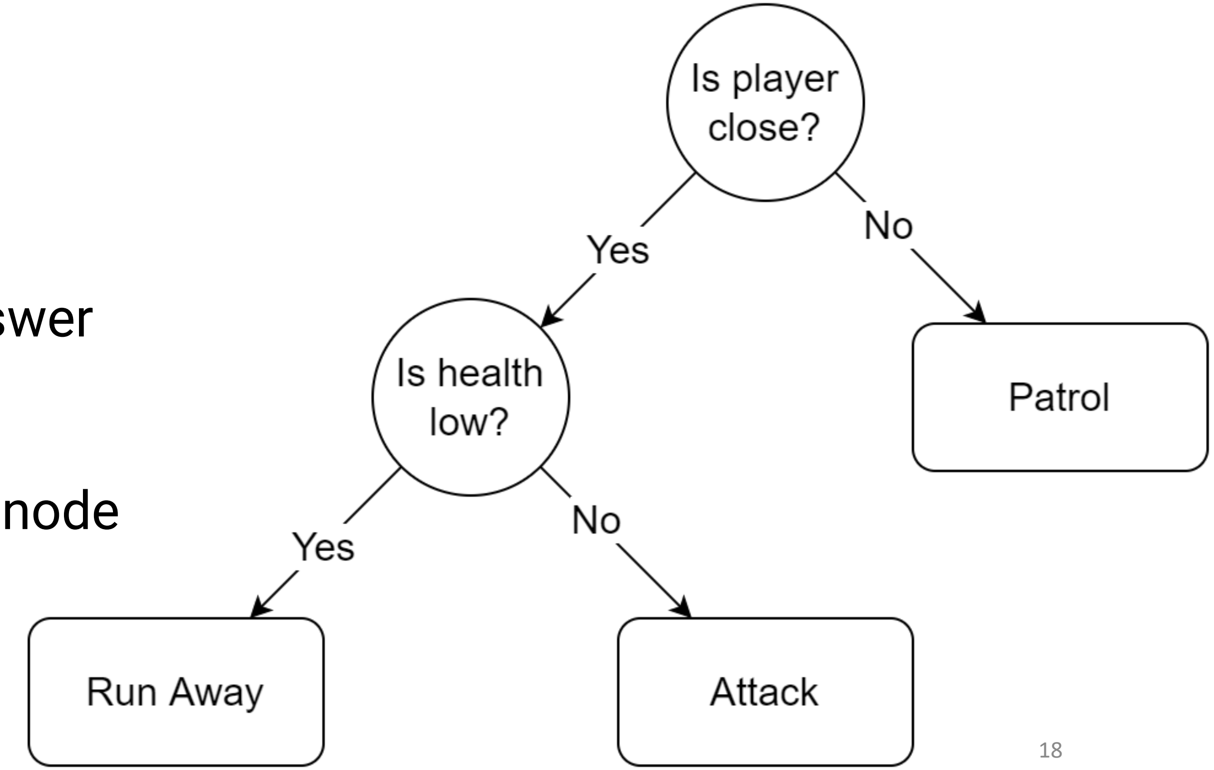


Finite State Machines (3)

- Decision making is encapsulated in the transition rules
- Transition rules can incorporate a certain degree of randomization
  - Such as an enemy running away at less than 15-25% health
- This is called **reactive AI** – always react to a game event
- The other type is **active AI** – constantly look for the best option
  - A sim in Sims
  - An AI opponent in Starcraft 2

Decision trees

- Decision trees are a simple way of representing decision making
- The inner nodes of a decision tree are decisions with only two possible answers: **Yes** or **No**
- Leaf nodes are action nodes
- Other nodes have two children
  - one for the Yes answer, one for the No answer
- We traverse the tree from root
  1. Evaluate conditions till you get to a leaf node
  2. Perform the action of the leaf node

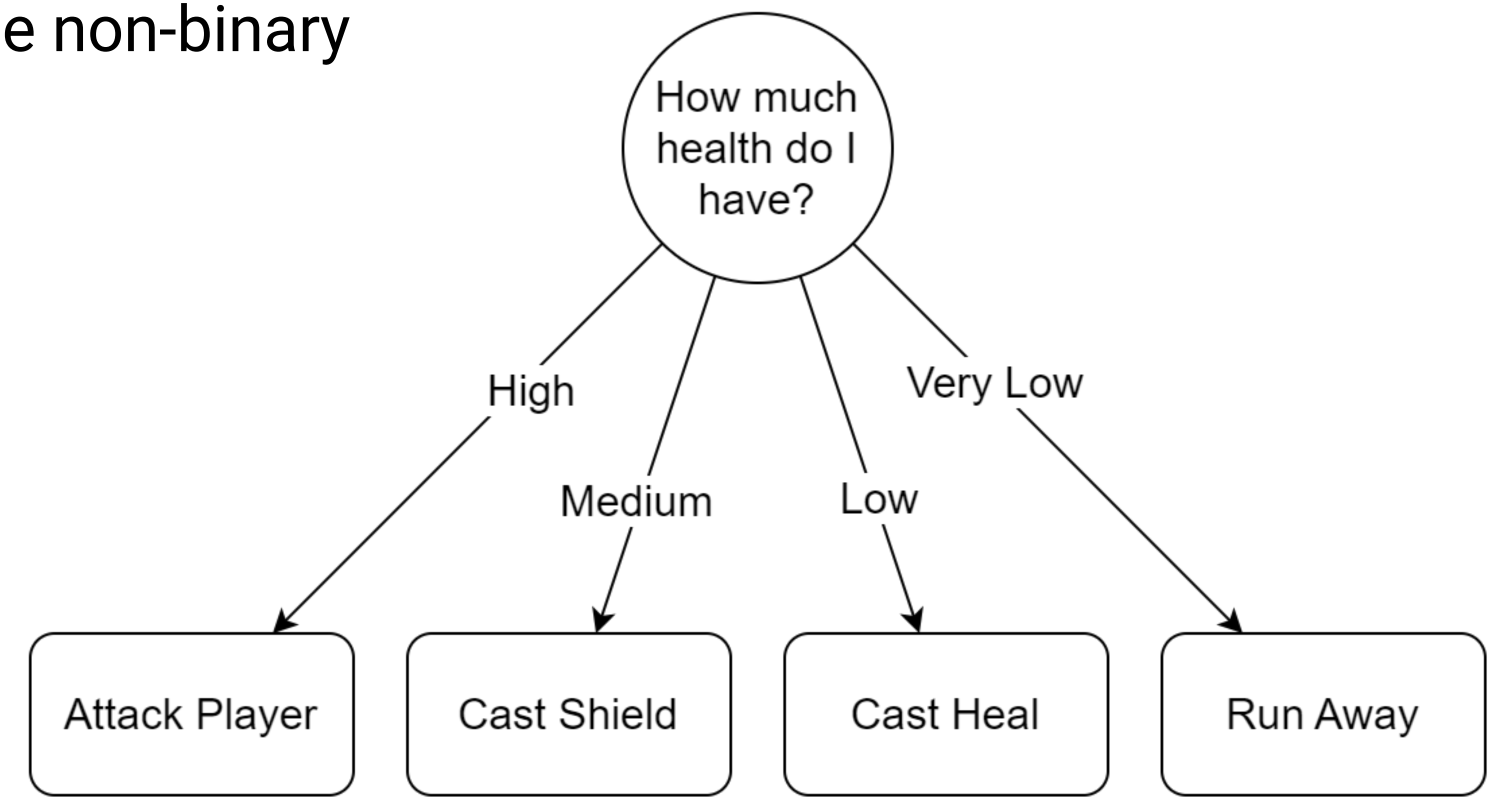


Decision trees (2)

- Apply an action every time a decision needs to be made
  - Decision trees can be shared, as can be individual nodes
  - Decision trees are sometimes built-in visual tools
    - Programmers write code for decision nodes and action nodes
    - (AI) designers connect these to build an actual tree – the AI “mind”
- With each decision we ignore a whole sub-tree
  - This is relatively efficient even for hundreds of nodes
- The decision might take several frames to decide
  - Save the node in which decision making is paused
  - Resume tree traversal in the next frame (decisions might be delayed)

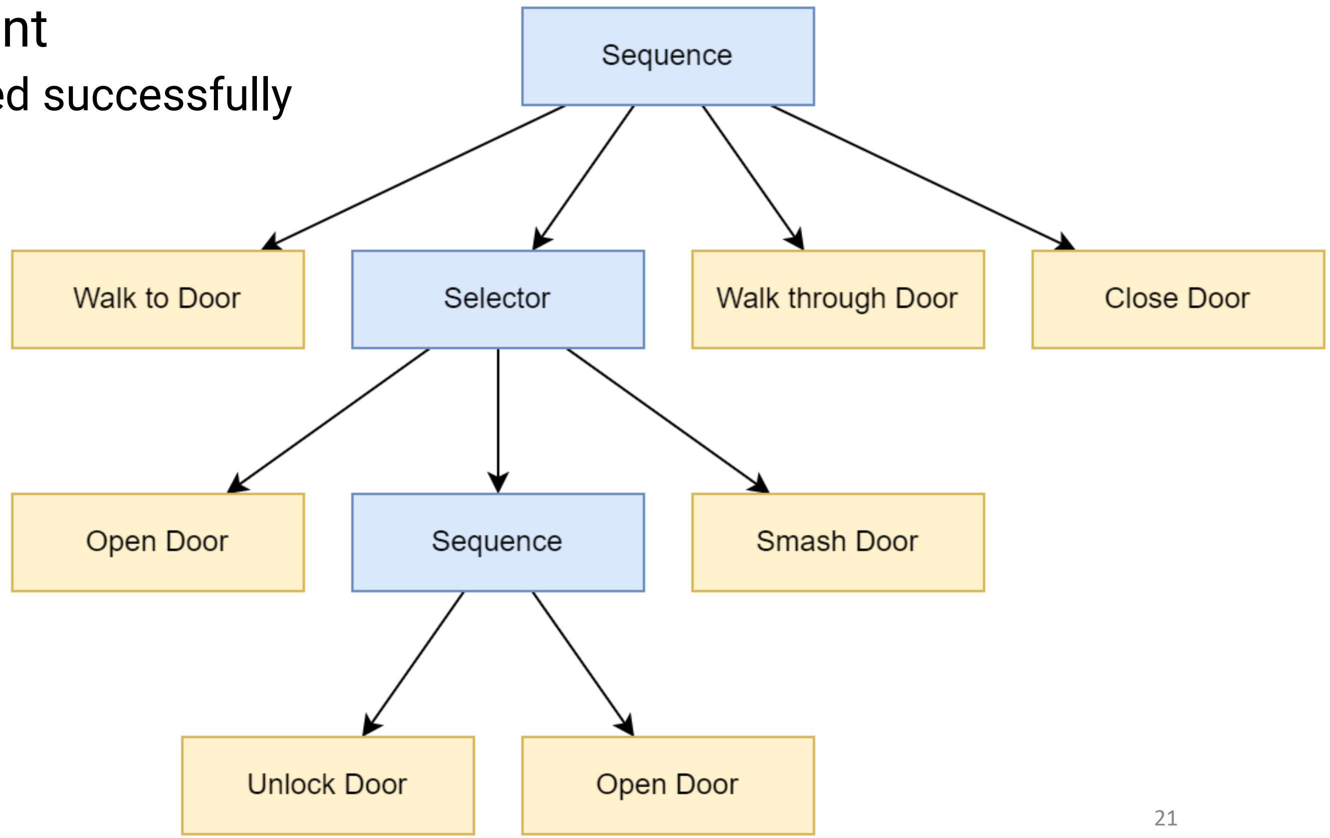
Decision trees (3)

- Can also be non-binary



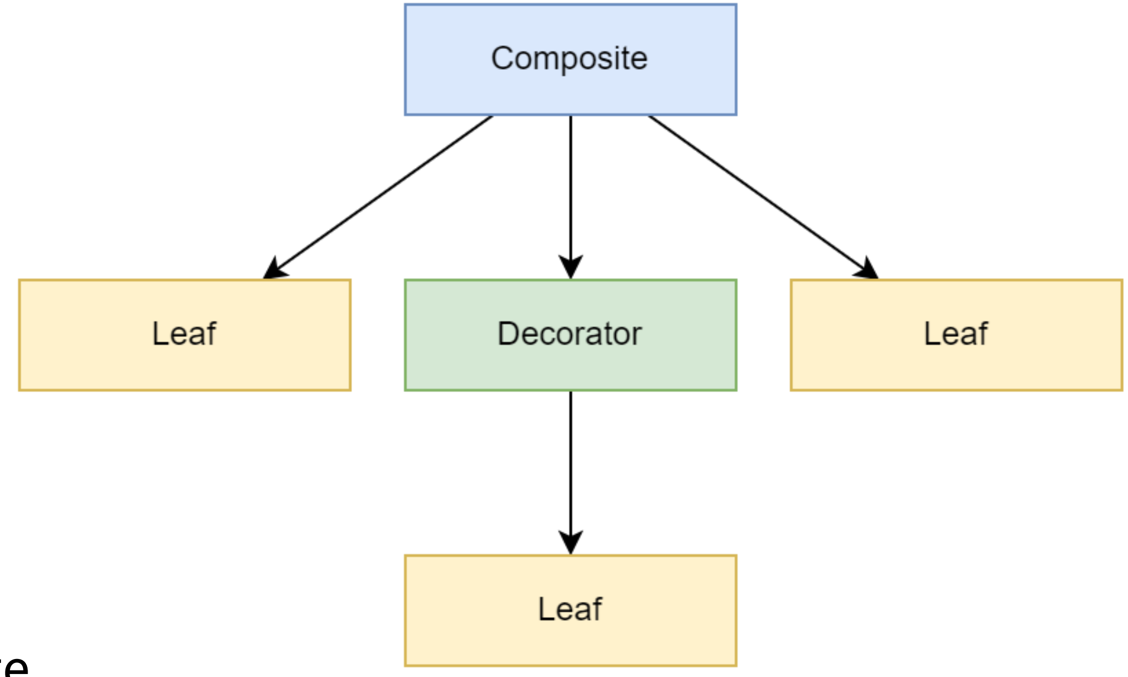
Behavior Trees

- Widely used in games
- It's a tree composed of nodes
- Each node can return a status to its parent
  - **Success** – the operation of this node finished successfully
  - **Failure** – the operation failed
  - **Running** – the operation is still running
- Nodes can have parameters
- Nodes can respond to context
  - Game state
- Available in Unity as a package
  - Unity Behavior
- Built-in support in Unreal



Behavior Trees (2)

- Leaf nodes represent actions
  - State: Running, Success or Failure
  - E.g. **Open a door**, **Run towards the player**
- Composite nodes (**Sequence**, **Selector…**)
  - Encapsulate multiple children
  - Execute children in some order
  - Returns what is returned from children
- Decorator nodes (**Inverter**, **Repeater**…)
  - Have a single child node
  - Transform result from the child, repeat, terminate
  - E.g.

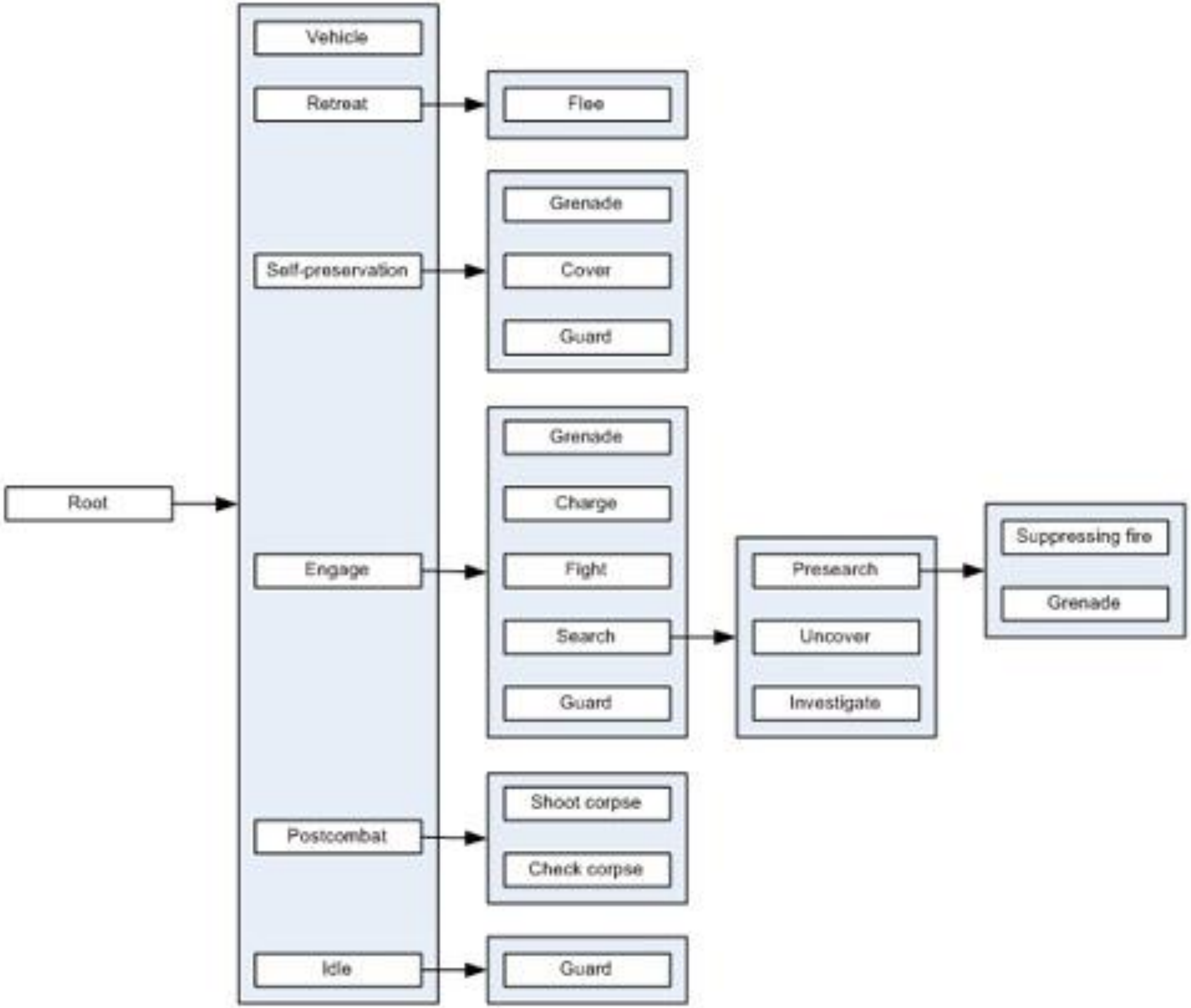


Behavior Trees – Actions

- Action **Walk**
  - Parameters
    - Character
    - Destination – location or another character

  - **Running** On the way
  - **Success** Reached destination
  - **Failure** Failed to reach destination (blocked/died/stunned…)
- **Init** – called the first time the node is visited
- **Process/Update** – called every tick while the node is “running”

Halo 2

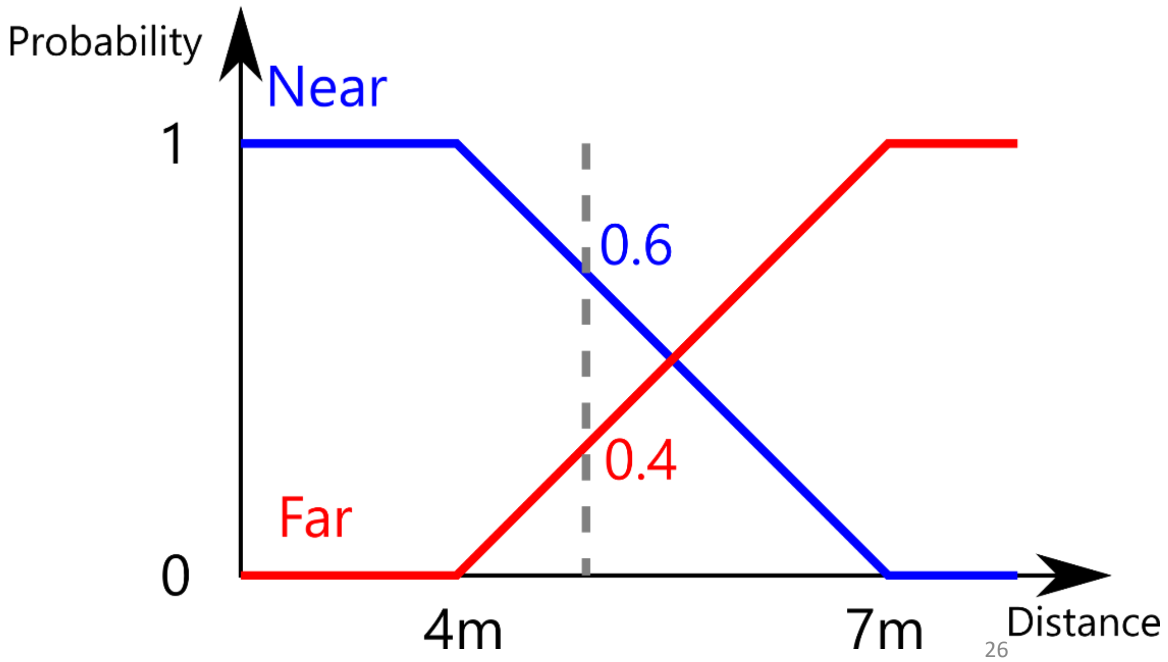

Vehicle
Retreat
Flee
Grenade
Self-preservation
Cover
Guard
Grenade
Root
Charge
Suppressing fire
Engage
Fight
Presearch
Grenade
Search
Uncover
Guard
Investigate
Shoot corpse
Postcombat
Check corpse
Idle
Guard

Fuzzy logic

- Decision trees work quite well, but it’s not realistic enough
  - Using absolute threshold values to decide
- There should be a range of values that allow for both decisions to happen
  - Proximity test
    - 5 meters is still too far sometimes
    - 7 meters is close enough sometimes
- The idea of **fuzzy logic** is that objects belong to multiple fuzzy sets by different amounts
  - A player partially behind cover can be in sets “in cover” as well as “exposed”, however we assign percentages for each set ⇒ 60% in cover, 40% exposed

Fuzzy logic (2)

- The process of assigning the degrees of membership is called *fuzzification*
- In order to decide, we might have to *defuzzify* the membership degrees and give an exact result to which set we fully belong
- Simple fuzzification:
  - Cutoff values for fully belonging to a set
  - Proximity ⇒ 2 sets “near” and “far”
  - 4 meters = near, 7 meters = far
  - between 4 and 7 meters
    - weighted randomness decides



Fuzzy logic (3)

- Defuzzification is much harder
  - From several degrees, we must choose the correct one
  - Just generating a random number and considering which set is more likely to occur can work is some situations
- We cannot just take the set with the highest degree
  - fuzziness provides a chance for something unlikely to happen
- If the result is just a number, it is much easier to defuzzify
  - An AI might be cautious, when combined with the fact that the player is behind cover, we generate a number that says how long the AI will take to aim
- For boolean values, we determine a cutoff and then compare it to the degree

Fuzzy logic (4)

- The real power comes from rapid AI prototyping
  - If (distance < 20 AND health > 1) then Attack()
  - If (player is close AND I am healthy) then Attack()
- We are using two fuzzy sets in the example
- We need to redefine the AND, OR and NOT operators for fuzzy sets
  - It’s no longer Boolean logic

$$\text{AND} \rightarrow P = \min(A, B)$$

$$\text{OR} \rightarrow P = \max(A, B)$$

$$A, B \ - \text{degrees of membership}$$

$$P \ - \text{final probability}$$

Utility theory

- *"Utility theory says that every state has a degree of usefulness, or utility, to an agent and that the agent will prefer states with higher utility."*
- We take the current world state, think of what would happen if we performed some action
- What changes in the world state can be used to derive how much that agent improved its "happiness"
- Actions with the highest utility value are chosen and performed

Utility theory – examples

- Chess is perfect for executing Utility theory
  - If one action causes me to lose an important piece in the next move ⇒ most likely low utility value
    - There are exceptions of course
  - Predict all possible outcomes in the next few steps
    - Choose the step maximizing the utility value
- A strategy game considers multiple things
  - Troop strength
  - Base/worker safety
  - Estimated enemy strength
  - Research level, amount of resources

Utility theory in practice

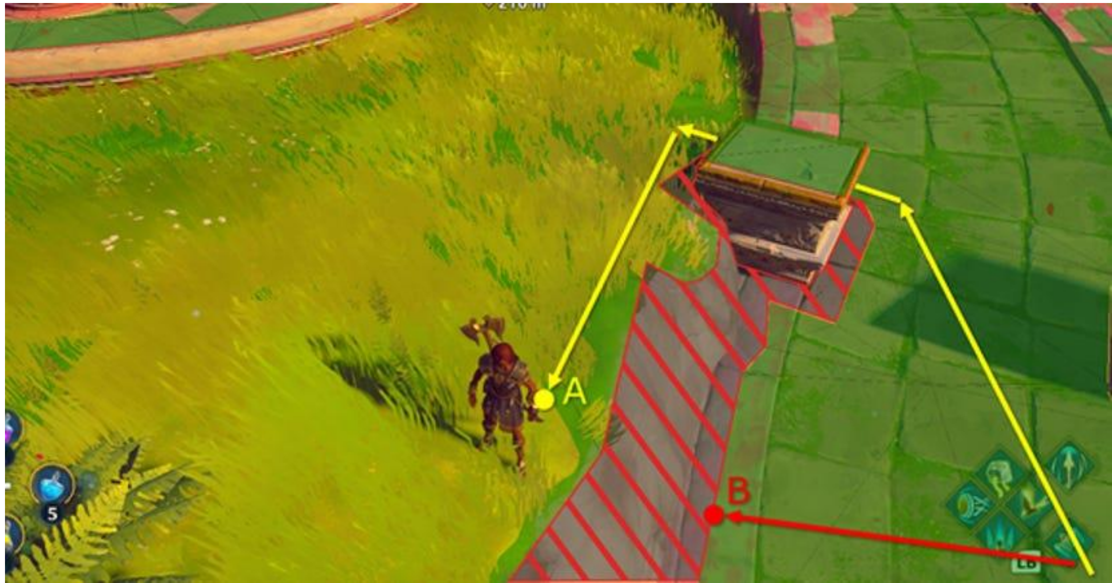
1. Make a copy of the game state
2. Perform the action (can take several seconds)
3. Evaluate what happened – how did utility change
    - Might require player prediction

- Usually localized decisions – the entire game state is not needed
    - In Sims, a sim usually cares only about themselves
        - If the sim is hungry, eating will improve his happiness the most
        - So they go to the kitchen
- In FPS games, the agents have simple utility preferences
    - Agents will be preferring states where they continue to live
    - And prefer when the player will have low health as a result of their actions

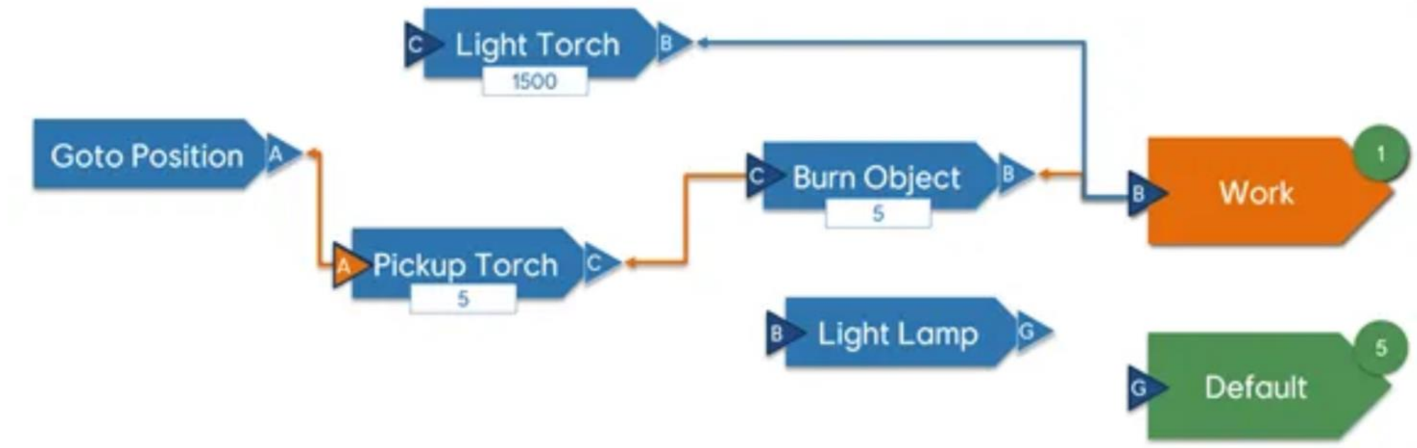
Goal-oriented action planning (GOAP)

- Utility theory decides what an agent wants to do, not HOW to do it
- GOAP is working with **goals** – desirable world states
  - the agent wants to achieve these states by performing actions
- Simple goal: kill the player
  - Attacking the player is one action that achieves this
- An agent has multiple goals, but usually only one active at a time
- Two-stage process:
  - Goal selection – pick the most relevant goal
  - Execution – solve the goal by executing actions
- Goal selection is solved using other methods
  - Decision trees, utility theory, …
- The second stage needs a special solution

https://www.gamedeveloper.com/programming/postmortem-AI-action-planning-on-Assassins-Creed-Odyssey-and-Immortals-Fenyx-Rising-



GOAP (2)



- Say a character is hungry
  - You have no food, so you need to create a plan to obtain food
  - Could be going into the woods to hunt animals, then extract the meat, cook it, and finally eat it
- Each action has a set of conditions it can satisfy, as well as a set of prerequisites that need to be satisfied
  - Eating food requires cooking food
  - Cooking food requires having raw food
  - Having raw food requires buying raw food
  - Buying food requires money
  - Money requires a job
- The algorithm walks back through these preconditions and identifies which actions need to be executed

GOAP (3)

- A sequence of goals might not exist
- There are lots of problems with world representation (not only for GOAP)
  - I desire a world state in which I am not hungry
  - I desire a world state in which the player is dead
  - We need to generate this world state with preconditions and effects
- Search for the shortest (or least difficult) path in a graph of actions
  - There are many ways to solve a goal
- We always walk back from the desired state to the current state
  - Trying to find a way that could work
- Quite advanced, but allows for very “intelligent” AI

Path-finding

- Not really an AI technique, more of a support technique for other AI
- Simply searching for the shortest path from A to B
  - We have nodes and edges
  - Nodes describe points that the agent must be able to reach
  - Nodes are connected by edges – straight lines
  - An agent moves along an edge to get to another node
- To get to a neighboring node, you just rotate the agent and move them along the corresponding edge

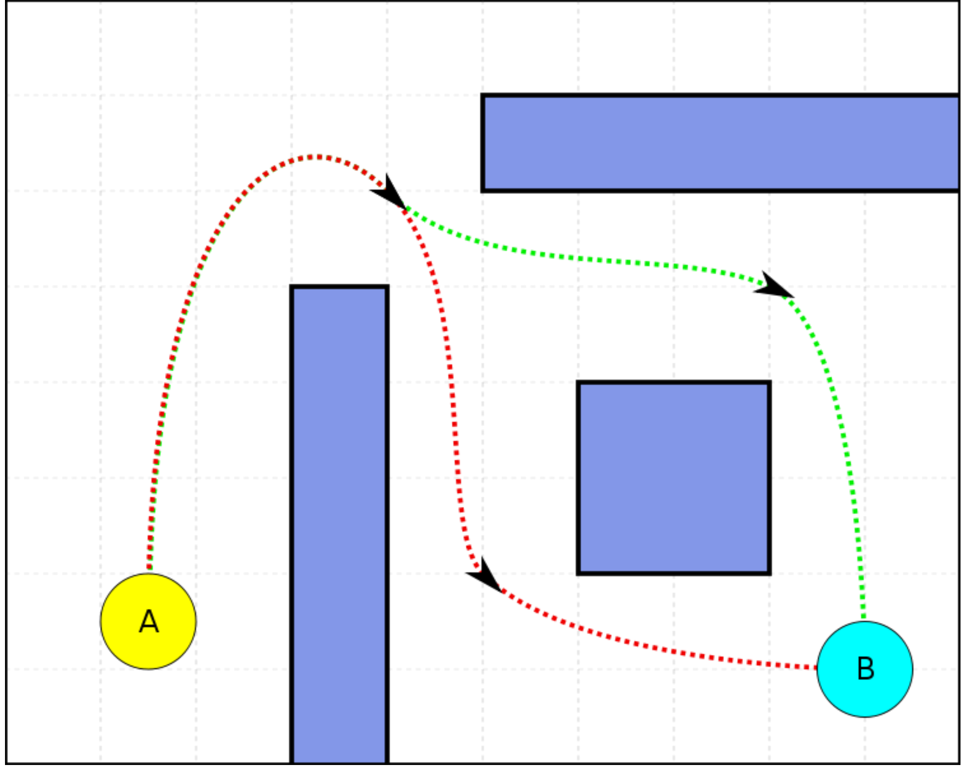


Image from https://en.wikipedia.org/wiki/Pathfinding

Path-finding (2)

- Moving along straight lines is highly unnatural
  - Except for robots maybe
- Nodes may be in a grid, resulting in not very smooth motion
- A few possibilities to avoid this:
  - Irregularly placed nodes
  - Allow each node to have a tolerance as to how close the agent must be to consider that they visited the node
  - Placing an interpolation curve (e.g. piecewise bezier curve) through the nodes

Path-finding (3)

- Edges may be unidirectional, bidirectional, even weighted
- Higher weight means a harder to pass route
- Weights could even be different for different types of agents
  - Flying units versus ground units, or units that can walk up cliffs
  - Results in different paths taken by different agents
- Weights can be dynamic
  - Building something on top of existing nodes sets the weight to infinity
  - Flying units might try to avoid guard towers, so the guard towers increase the weight of nearby edges

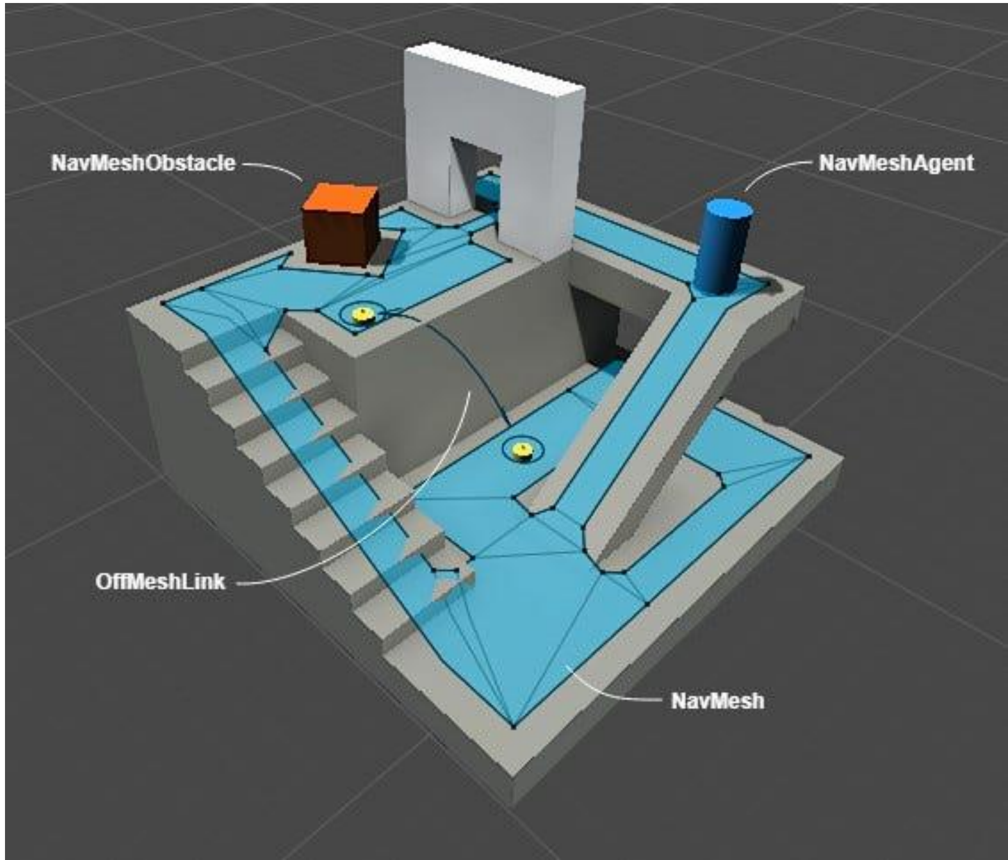
A* path-finding (aka. A-star)

- There are lots of algorithms that solve the path-finding problem
- A* is the most used one
- Relatively fast to compute
- Has lots of modifications

https://en.wikipedia.org/wiki/A*_search_algorithm

Path-finding – taking it a step further

- Another common technique is called a **navigation mesh (navmesh)**
- It is a simple mesh that describes **all** walkable terrain in the level
  - Can be artist generated
  - Much better is when it's generated automatically
    - Might require some tweaking by artists or designers
- Triangles are nodes, edges are between neighboring triangles
- A* can be used, we just have to set the tolerance values based on the triangles



AI in Unity

- Limited AI support without plugins
  - Can use Unity Behavior for Behavior Trees
  - Can use Unity's Animator for Finite State Machines
  - Can use Visual Scripting for Finite State Machines
- Has ML Agents package for reinforced learning
- Making your own is not that hard for simple games
- Other free/paid plugins: Behavior Designer, NodeCanvas, Apex Utility AI…
- Writing it yourself (no visual representation) is also OK
  - But think about configurability & the potential to modify it

AI in Unity

- Unity has built-in support for NavMesh Path-finding
  - Static NavMeshes
  - Dynamic obstacles and priorities
  - Rebuild NavMesh dynamically
  - Only for 3D
- For 2D
  - Use NavMeshPlus – https://github.com/h8man/NavMeshPlus
    - Built on top of Unity's 3D NavMesh
  - Use A* Pathfinding Project - has a free/paid version – https://arongranberg.com/astar/

References

- McShaffry, M., and Graham, D. *Game Coding Complete, Fourth Edition*. Course Technology PTR, 2012.
- Millington, Ian, and John D. Funge. *Artificial intelligence for games*. Burlington, MA: Morgan Kaufmann/Elsevier, 2009. Print.
- https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work
- https://www.gamedeveloper.com/programming/postmortem-AI-action-planning-on-Assassins-Creed-Odyssey-and-Immortals-Fenyx-Rising-