Decision Making for Games

Subject: Artificial Intelligence

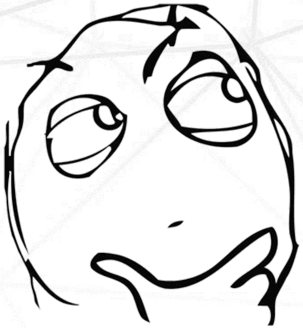
Professors: Edison jair Bejarano Sepulveda & Ramon Mateo Navarro

Course: 2024 / 2025

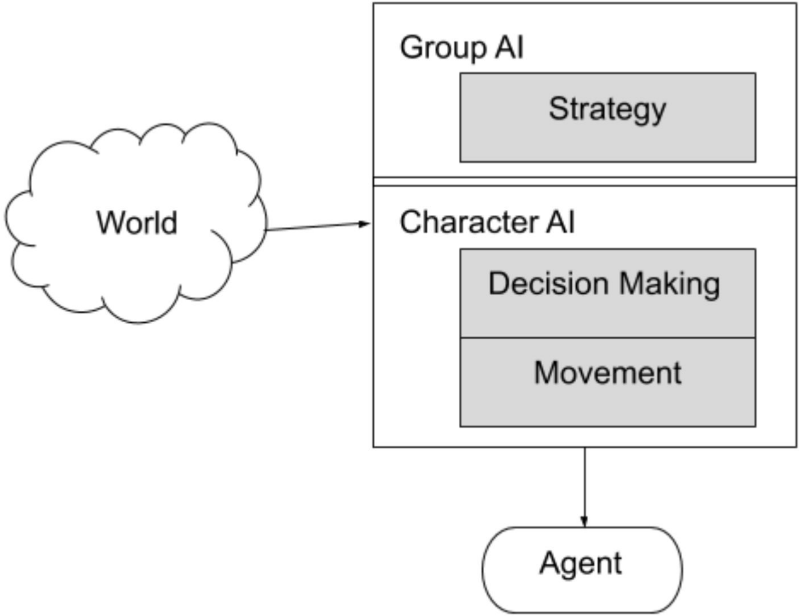
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Overview

- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
- Planning Systems
- References



GameAI: the Model

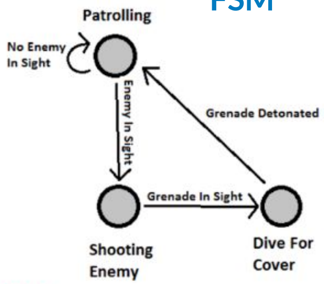


Decision Making

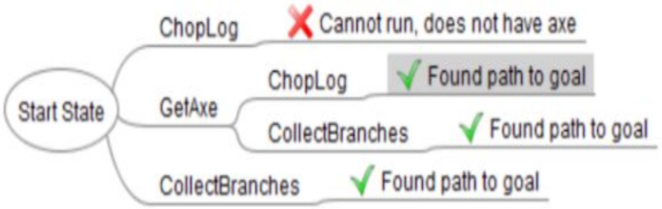
- Input: World Knowledge
- Output: Action
- Important rule:

  Decision Making should **NOT** execute every frame!
- Main algorithms:
  - Finite State Machines
  - Behaviour Trees
  - Goal Oriented Action Planning

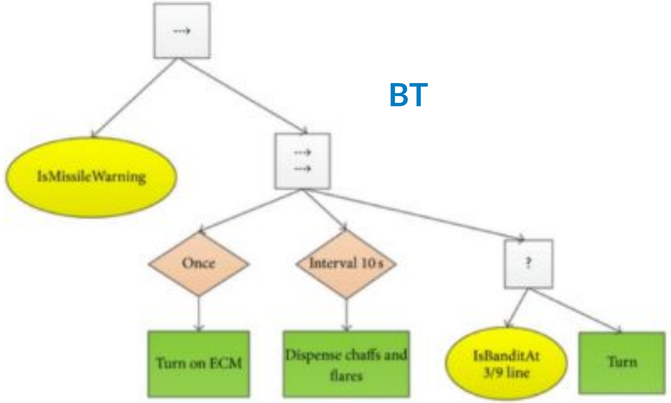
FSM



GOAP

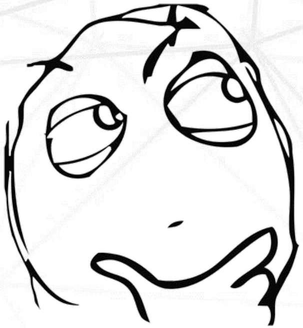


BT



Overview

- Introduction
- **Finite State Machines**
  - **Code(delegates)**
  - Visual Scripting
- Decision Trees
- Behaviour Trees
- Planning Systems
- References



C# Coroutine Example

```
IEnumerator<int> fibonacci()
{
        int a = 0;              // Output:
        int b = 1;              // 0,1,1,2,3,5,8,13,21,34...
        yield return a;
        while (true)
        {
            yield return b;
            int c = b;
            b = a + b;
            a = c;
        }
}
void Start()
{
        IEnumerator<int> f = fibonacci();
        for(int i = 0; i < 10; i++)
        {
            f.MoveNext();
            Debug.Log(f.Current);
        }
}
```

C # coroutines

Example:

```
using UnityEngine;
using System.Collections;
public class WaitForSecondsExample : MonoBehaviour {
  void Start() {
     StartCoroutine("Example");
  }
  IEnumerator Example() {
     Debug.Log(Time.time);
     yield return new WaitForSeconds(5);
     Debug.Log(Time.time);
  }
}
```

- `StartCoroutine`: type of asynchronous "functions"
- `IEnumerator`: returning type
- `yield`: stops execution until something happens
  - `yield return null`: until next frame
  - `yield break`: finish the coroutine

C# delegates

Assigning functions to variables

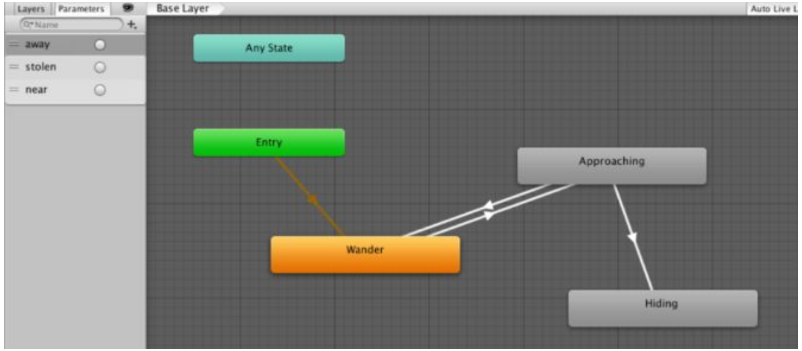
Example:

```
public class DelegateScript : MonoBehaviour {
   delegate void MyDelegate(int num);
   MyDelegate myDelegate;
   void Start () {
      myDelegate = PrintNum;
      myDelegate(50);
      myDelegate = DoubleNum;
      myDelegate(50);
   }
   void PrintNum(int num) {
      Debug.Log(num);
   }
   void DoubleNum(int num) {
      Debug.Log(num * 2);
   }
}
```

FSMs IN Unity

Scene(link unity package) :

Task : FSM for the robber



https://learn.unity.com/tutorial/finite-state-machines-1#

FSM with coroutines & delegate

Code template:

```
public class FSM : MonoBehaviour
{
   ...
   private WaitForSeconds wait = new WaitForSeconds(0.05f);   // 1 / 20
   delegate IEnumerator State();
   private State state;
   IEnumerator Start()
   {
      ...
      state = Wander;
      while (enabled)
         yield return StartCoroutine(state());
   }
   IEnumerator Wander()
   {
      Debug.Log("Wander state");
      ...
   }
}
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

TODO

1. Coroutine that executes 20 times per second and goes forever.
2. Explicit every state change with `Debub.Log.`
3. First behaviour is slowly wander.
4. When the *cop* walks away from the treasure he has to approach quickly to steal it.
5. If the *cop* comes back he returns to wander slowly and so on.
6. If the robbery is successful (the treasure must disappear), he begins to permanently hide in the obstacle closest.

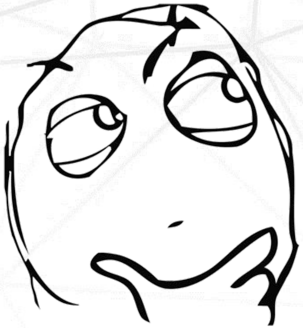
solution: view* / download

Homework

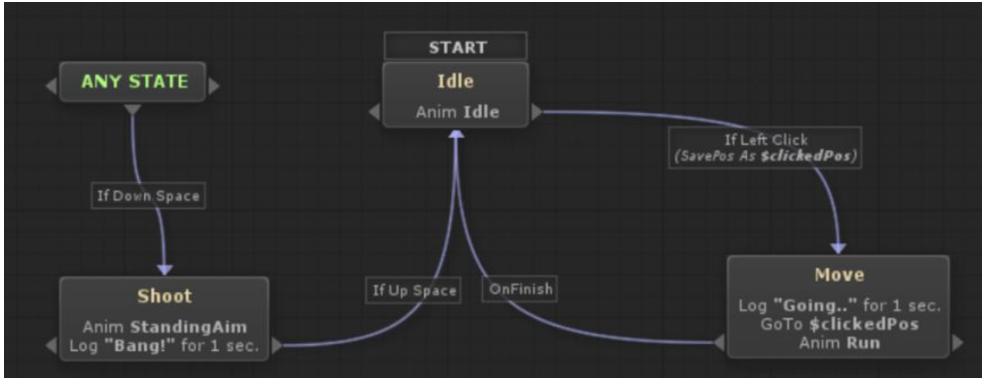
- Watch the videos (5mn): Killzone 2 Review about AI & F.E.A.R. 2 - A.I.

Overview

- Introduction
- **Finite State Machines**
  - Code(delegates)
  - **Visual Scripting**
- Decision Trees
- Behaviour Trees
- Planning Systems
- References

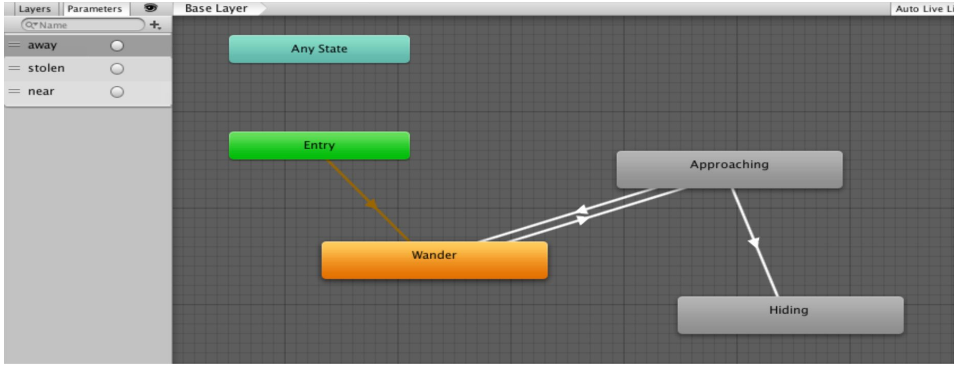


Visual Scripting



- Visual editors helps handling complex behaviours
- Separates coders from game designers
- Many options:
  - CryEngine's flowgraph
  - Unreal Kismet / Blueprint
  - Unity PlayMaker

FSM with Unity's Animator

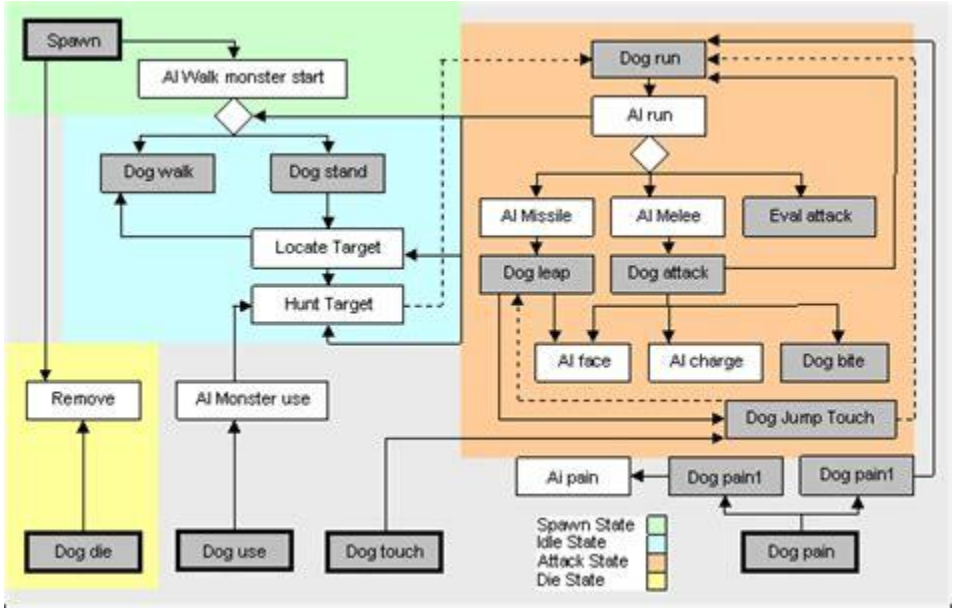


- Wander State: view* / download
- Approaching State: view* / download
- Hiding State: view* / download
- BlackBoard: view* / download

https://github.com/EjbejaranosAI/AI4VJ/tree/main/Lecture%20material/T3

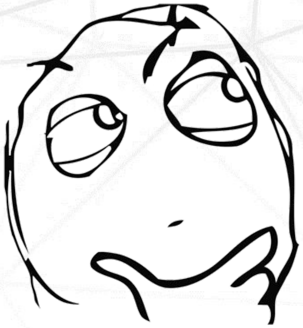
Hierarchical FSM

Complex Behaviours:



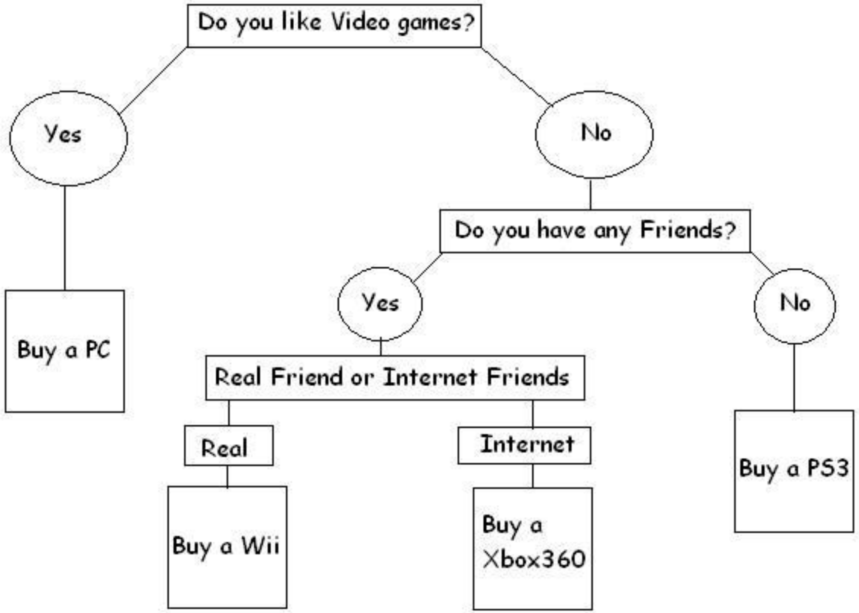
Overview

- Introduction
- Finite State Machines
- **Decision Trees**
- Behaviour Trees
- Planning Systems
- References



What Video Game System Shoulds I own

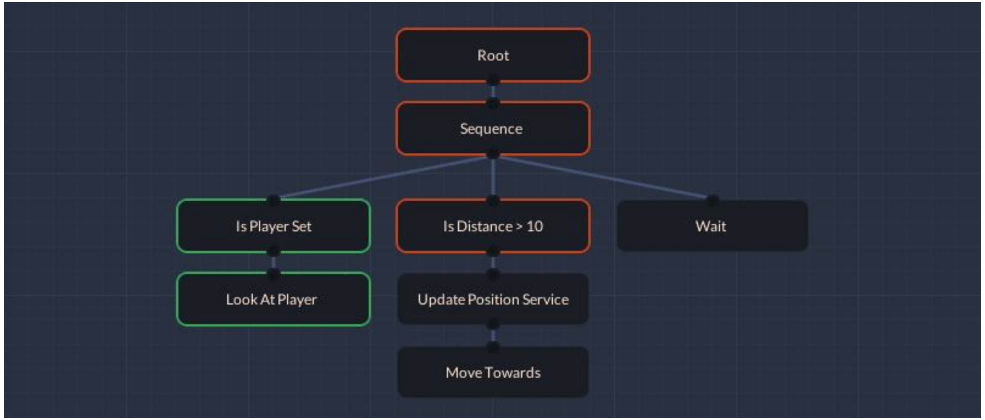
A very simple yet accurate guide on what Video Game System is right for you



source

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

A decision tree is a tool for making decisions by breaking them into a series of questions, where each answer leads to a new question or final outcome. It works by starting at a main question (root), and based on the answer, it follows a branch to the next step until it reaches a decision (leaf).

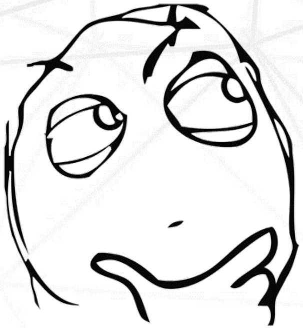


FSM vs DTs

- FSM: States (with Actions) & Transitions (with conditions)
- DTs: Conditions (tree nodes) & Actions (leafs).
- It has no notion of state; we have to go through the whole tree every time we run it.
- How could we use decision trees in games?
  - NPCs Dialogs
  - Bosses that switch state every % HP
  - Bosses that makes different abilities depending on climates conditions
- Decision trees can be generated automatically.
  We will see this in the topic of machine learning.

Overview

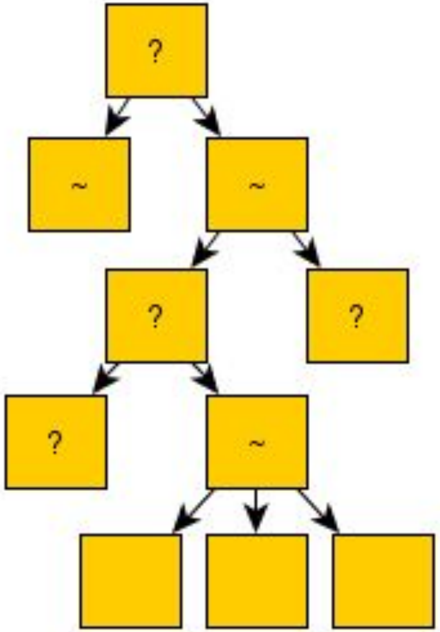
- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
  - Design
  - Behaviour Brincks
- Planning Systems
- References



Behaviour Trees

- Sort of visual programming for AI behaviour (Isla, 2005)
  - Reusability & modularity
  - Major engines: unreal, cryengine, unity
- Behavior Tree combine both:
  - *Decision trees*: execute all at once
  - *State machines*: current state implicit
  - **the execution stays in one of the nodes**
- Designing Trees is a hard task!

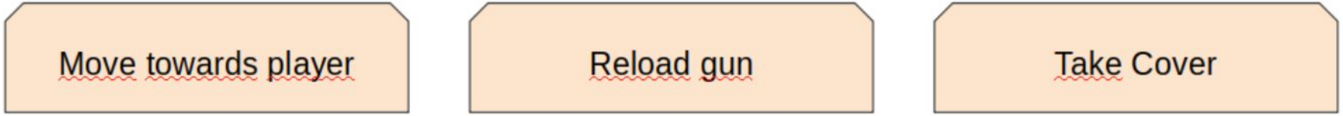
  Reference: *Behavior trees for AI: How they work*



Node Types I

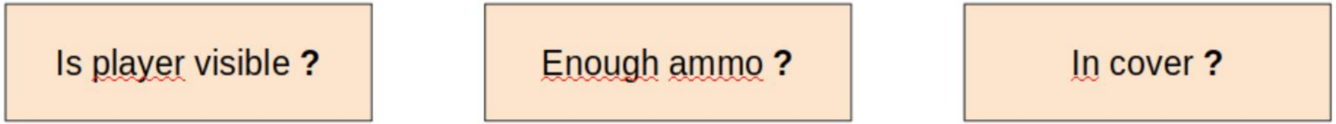
Actions

- All should return **Running, Success** or **Failure**
- They can take a while!
- Most of the time they will be leaf nodes



Conditions

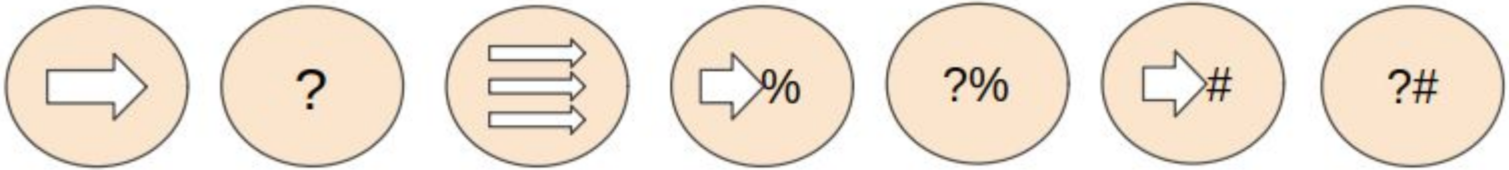
- All should return **True** or **False**
- Conditions normally refer to the **blackboard** for questioning the world state



Node Types II

Composites

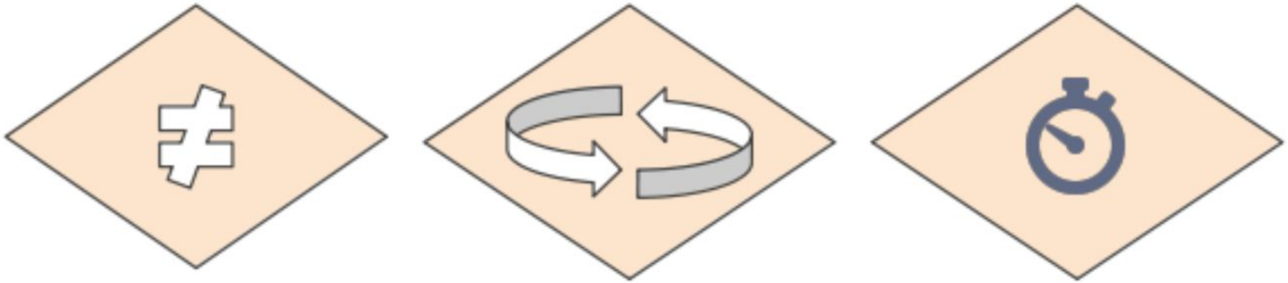
- All should return **True** or **False**
- They iterate all childs from left to right in a specific fashion:
    1. Sequence (AND): A node that executes all its children until one fails
    2. Selector (OR): A node that executes all its children until one succeeds
    3. Parallel (Concurrent AND): Execute all its children at the same time until one fails
    4. Random Sequence or Selector (with %?): Same as sequence or selector but randomly
    5. Priority Sequence or Selector (with %#): Same as sequence or selector but follow a mutable priority



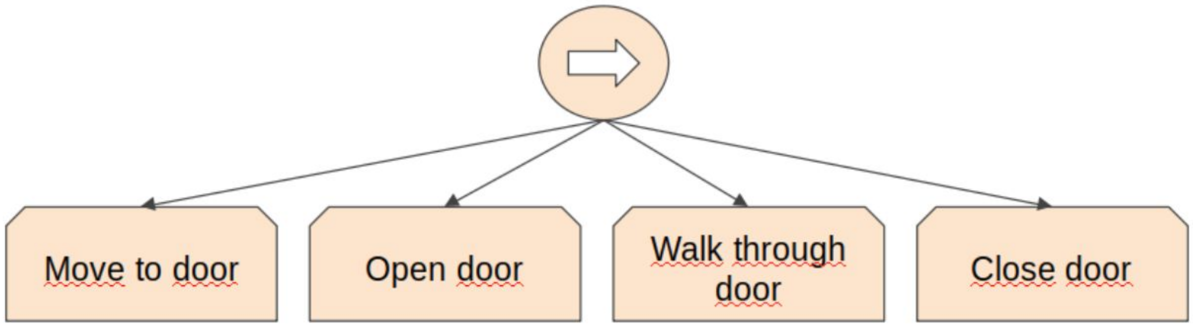
Node Types III

Decorators

- All should return **Running, Success** or **Failure**
- Add enormous flexibility and power to the tree execution flow
- They modify one specific child in some fashion:
  1. Inverter (NOT): invert the result of the child node
  2. Repeater (until fail, N or infinite): basically repeat the child node until fail or N times
  3. Wait until (seconds, condition, etc.): basically a generic delay

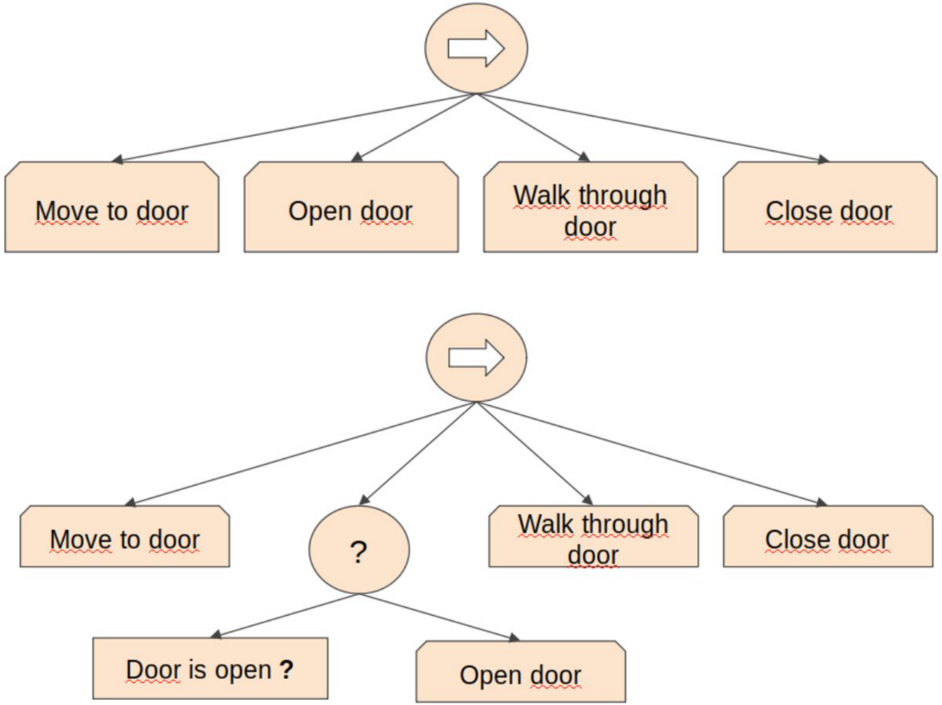


Example I



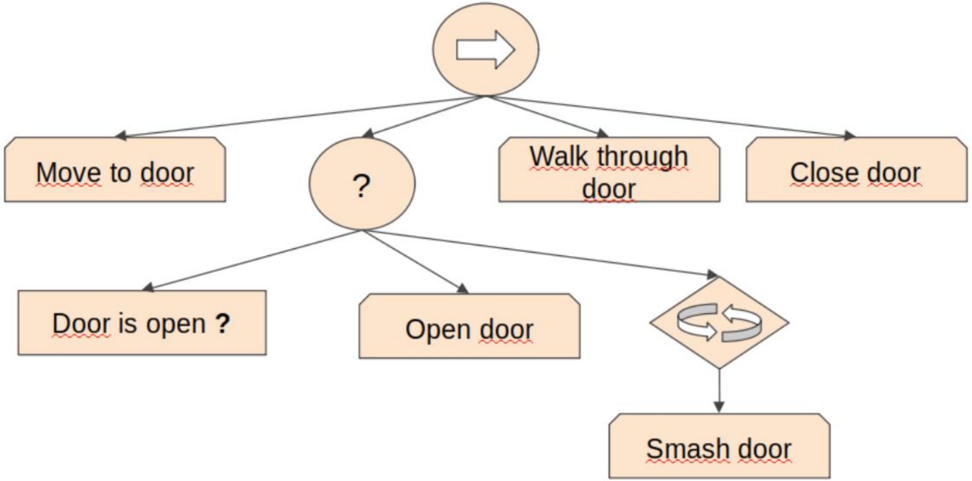
Behavior trees for AI: How they work

Example I



Behavior trees for AI: How they work

Example I



Behavior trees for AI: How they work

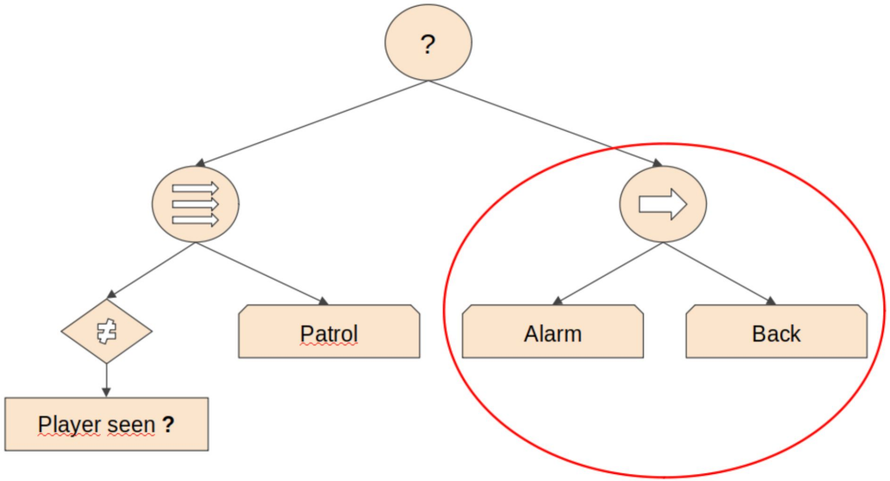
Exercise

What about the Robber?

Template link for design BTs

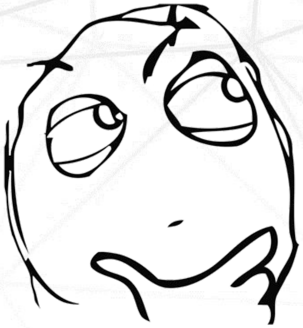
Behavior trees for AI: How they work

Nested Behaviour Trees



Overview

- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
  - Design
  - Behaviour Brincks
- Planning Systems
- References



Behaviour Bricks

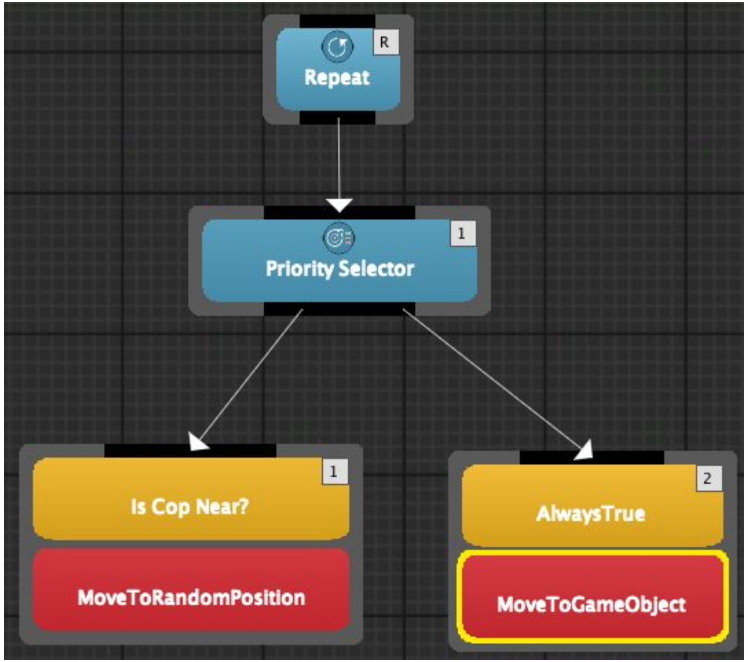
ToSteal behaviour tree:

Starting:

- Handout
- Editor:
- Window Behavior Bricks - Editor
- Robber:
- Add Component Behavior executor component

BlackBoard/properties:

- MoveToRandomPosition: Floor
- MoveToGameObject: Treasure



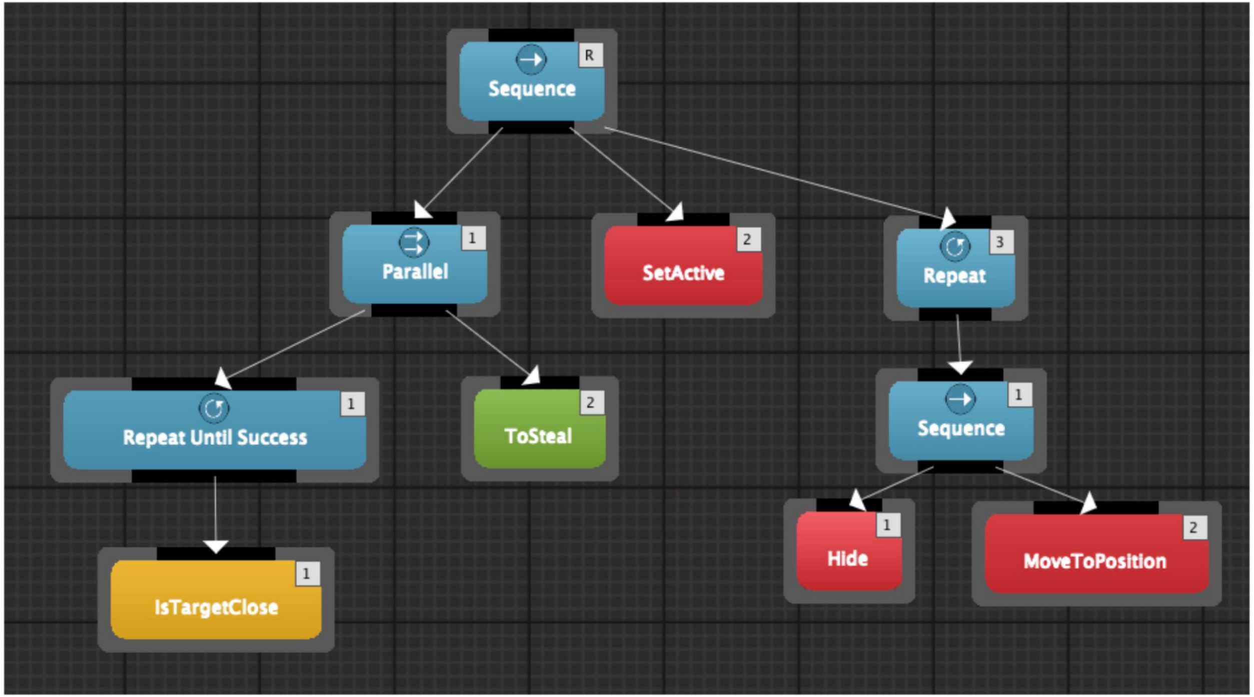
```csharp
using UnityEngine;

using Pada1.BBCore;
using Pada1.BBCore.Framework;

[Condition("MyConditions/Is Cop Near?")]
[Help("Checks whether Cop is near the Treasure.")]
public class IsCopNear : ConditionBase
{
    public override bool Check()
    {
        GameObject cop = GameObject.Find("Cop");
        GameObject treasure = GameObject.Find("Treasure");
        return Vector3.Distance(cop.transform.position, treasure.transform.position) < 10f;
    }
}
```

Behaviour Bricks II

ToSteal behaviour tree:



Behaviour Bricks III

ToSteal behaviour tree:

BlackBoard / properties:

- IsTargetClose: Treasure, 2
- ToSteal: Floor, Treasure
- SetActive: false, Tresure
- MoveToPosition: hide

Actions

```
using UnityEngine;
using Pada1.BBCore;          // Code attributes
using Pada1.BBCore.Tasks;    // TaskStatus
using Pada1.BBCore.Framework; // BasePrimitiveAction

[Action("MyActions/Hide")]
[Help("Get the Vector3 for hiding.")]
public class HideBB : BasePrimitiveAction
{
   [InParam("game object")]
   [Help("Game object to add the component, if no assigned the component is added to the game object of this behavior")]
   public GameObject targetGameobject;

   [OutParam("hide")]
   [Help("Vector3 for higing.")]
   public Vector3 hide;

   public override TaskStatus OnUpdate()
   {
      Moves moves = targetGameobject.GetComponent<Moves>();
      hide = moves.HideValue();
      return TaskStatus.COMPLETED;
   }
}
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Behavior Bricks: Links of interest

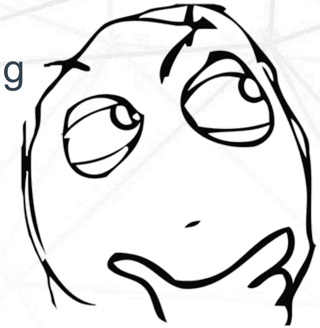
Link: Store

Homework:

1. Watch this tutorials:
    a. https://www.youtube.com/watch?v=CZvfuNfdc1M&t=533s
    b. https://www.youtube.com/watch?v=qBCGSxlXOFY&ab_channel=Sephtis
2. Read the documentation:
    a. https://bb.padaonegames.com/doku.php?id=quick%3Adesign

Overview

- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
- Planning Systems
  - Goal Oriented Behaviour
  - Goal Oriented Action Planning
  - AI Planner
- References



AI Paradigms

Reactive AI:

- How to achieve goals → AI
- Exs: FSM, DT, BT

Deliberative AI:

- World behaviour + goals → AI
- AI decides how to achieve its goals
- Ex: Planners

Games using dynamic planning:

- FEAR, Fallout 3, Total War, Deus Ex: Human Revolution, Shadow of Mordor, Tomb Raider

Goal Oriented Behaviour

Goals

- each agent can have many active, and they could change
- try to fulfill its goals or reduce its insistence (importance or priority as a number)
- examples: eat, drink, kill enemy, regenerate health, etc.

Actions

- atomic behaviours that fulfill a requirement
- combination of positive and negative effects
- Ex: "play game console" increases happiness but decreases energy
- environment can generate or activate new available actions (smart objects)

People simulation example

Goal: Eat = 4
Goal: **Sleep** = 3
Action: Get-Raw-Food (Eat − 3)
Action: Get-Snack (Eat − 2)
Action: **Sleep-In**-Bed (**Sleep** − 4)
Action: **Sleep**-On-Sofa (**Sleep** − 2)

- heuristic needed: most pressing goal, random...
- \+ fast, simple
- − side effects, no timing information

Goal: Eat = 4
Goal: Bathroom = 3
Action: Drink-Soda (Eat − 2; Bathroom + 3)
Action: Visit-Bathroom (Bathroom − 4)

GOB: Discontent

It is an energy metric to minimize:

- Sum of insistence values of all goals
- Sum of square values: it accentuates high values

Example:

Goal: Eat = 4
Goal: **Bathroom** = 3
Action: Drink-Soda (Eat − 2; Bathroom + 2)
after: Eat = 2, **Bathroom** = 5: **Discontentment** = 29
Action: Visit-**Bathroom** (**Bathroom** − 4)
after: Eat = 4, **Bathroom** = 0: **Discontentment** = 16

Solution: Visit-**Bathroom**

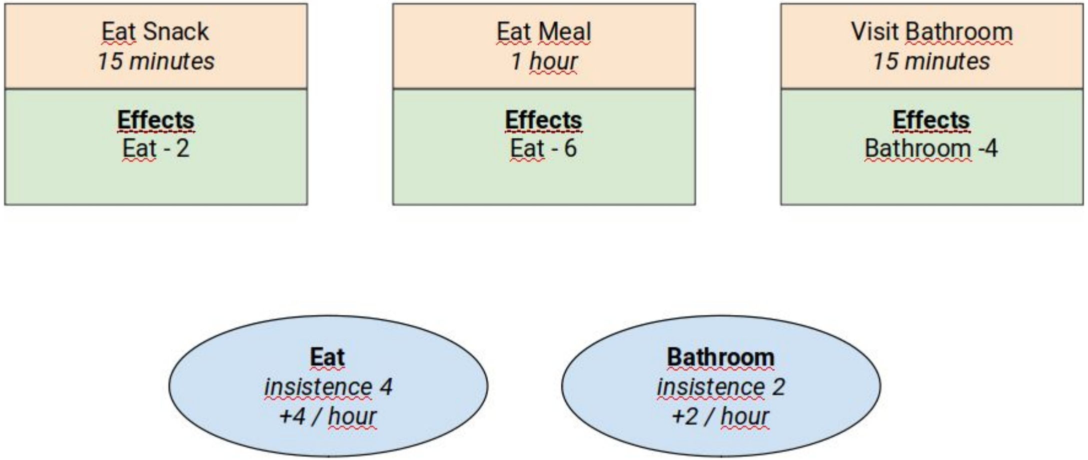
GOB: Timing

Goal: Eat = 4 changing at + 4 per hour
Goal: **Bathroom** = 3 changing at + 2 per hour
Action: Eat-Snack (Eat − 2) 15 minutes
after: Eat = 2, **Bathroom** = 3.5: **Discontentment** = 16.25
Action: Eat-Main-Meal (Eat − 4) 1 hour
after: Eat = 0, **Bathroom** = 5: **Discontentment** = 25
Action: Visit-**Bathroom** (**Bathroom** − 4) 15 minutes
after: Eat = 5, **Bathroom** = 0: **Discontentment** = 25

Solution: Eat-Snack

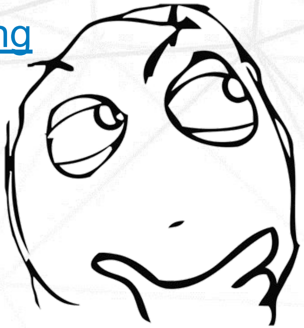
GOB Design

Goal: Eat = 4 changing at + 4 per hour
Goal: Bathroom = 3 changing at + 2 per hour
Action: Eat-Snack (Eat − 2) 15 minutes
Action: Eat-Main-Meal (Eat − 4) 1 hour
Action: Visit-Bathroom (Bathroom − 4) 15 minutes



Overview

- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
- Planning Systems
  - Goal Oriented Behaviour
  - Goal Oriented Action Planning
  - AI Planner
- References



The need for planning

Example:

- Mage character
  - 5 charges in its wand
  - need for healing
  - an ogre approaching him aggressively
- plan

Goal: Heal = 4
Goal: Kill-Ogre = 3
Action: Fireball (Kill-Ogre − 2) 3 charges
Action: Lesser-Healing (Heal − 2) 2 charges
Action: Greater-Healing (Heal − 4) 3 charges

Best combination: Lesser-Healing + Fireball
GOB solution: Greate-Healing

GOB is limited in its prediction, the situation needs to go some steps ahead!

Goal Oriented Action Planning

Chaining actions

- preconditions for chaining actions
- states for satisfying preconditions
- search algorithm for selecting "best" branches (each goal is the root of a tree)

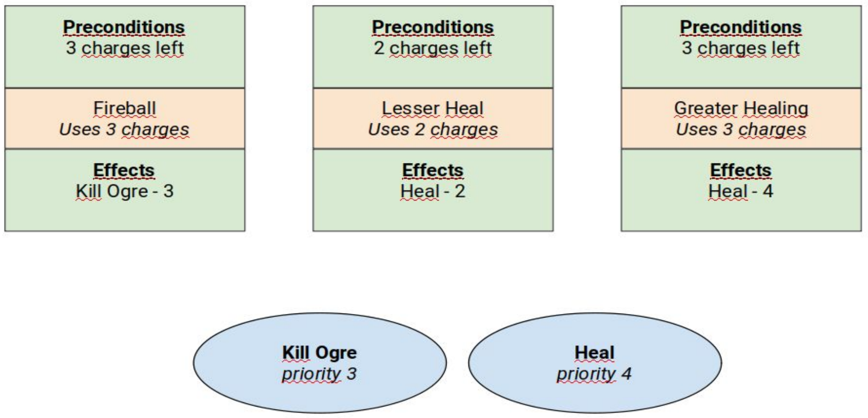
Searching

- BFS increasing the number of actions and goals it becomes quickly inefficient
- A* perhaps distance heuristic cannot be formulated
- Dijkstra: usual solution

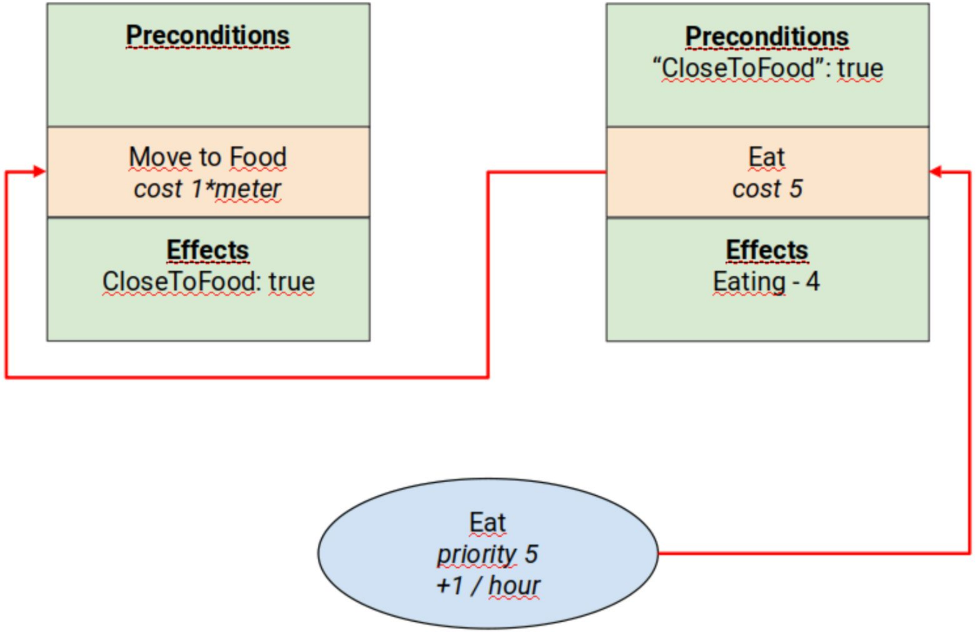
GOAP Design

Goal: Heal = 4
Goal: Kill-Ogre = 3
Action: Fireball (Kill-Ogre 2) 3 charges
Action: Lesser-Healing (Heal - 2) 2 charges
Action: Greater-Healing (Heal 4) 3 charges

Google Canvas Template



GOAP: Time



Preconditions
Move to Food
cost 1*meter
Effects
CloseToFood: true
Preconditions
"CloseToFood": true
Eat
cost 5
Effects
Eating - 4
Eat
priority 5
+1 / hour

Robber behaviour

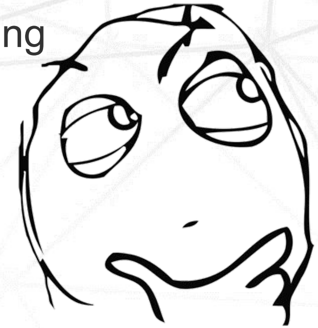
First approach

Preconditions CopNear	**Preconditions** ! CopNear	**Preconditions** ! CopNear Near
Wander	**Approach**	**Steal**
Effects	**Effects** Near	**Effects** Stolen Treasure.disabled

Steal
priority 25

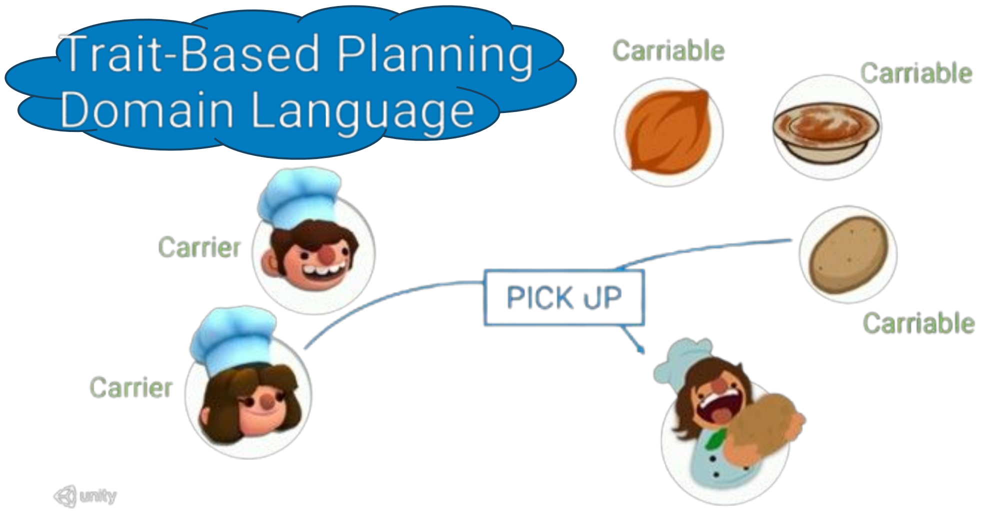
Overview

- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
- Planning Systems
  - Goal Oriented Behaviour
  - Goal Oriented Action Planning
  - AI Planner
- References



AI Planner

The AI Planner package can generate optimal plans for use in angente AI



- [Reference]()
- it contains a plan visualizer

Robber: Traits

Traits

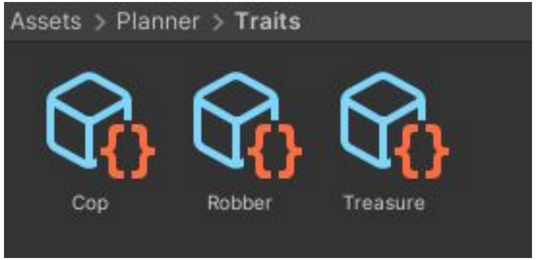
- Create - Semantic - Trait Definition
- fundamental data (game state)
- quality of objects (components)
- contains attributes

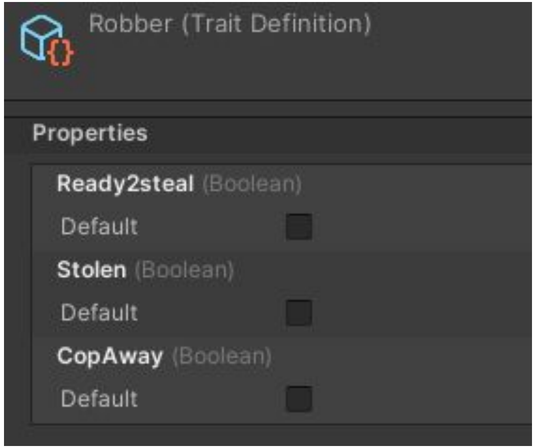
Robber

- Treasure
- Cop
- Robber: CopAway (false), Ready2steal (false), Stolen (false)

Building the traits

Menu - Semantic - Traits - Build





UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Robber: Actions I

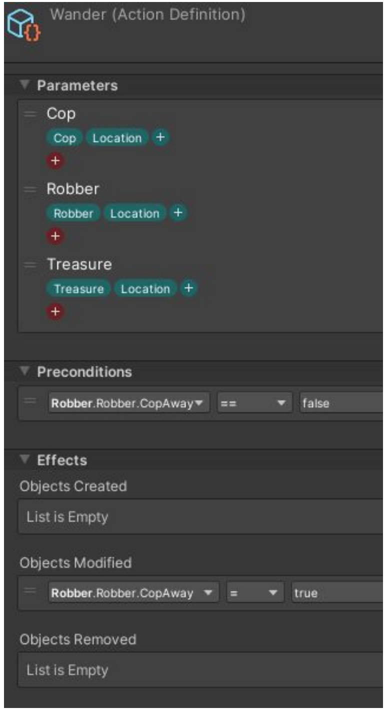
Actions

- Create - AI - Planner - Action - Definition
- planner potential decisions
- executes nothing
- Properties:
  name, parameters,
  preconditions, effects,
  cost/reward

Robber

Wander

- parameters: cop, robber, treasure
- precondition: CopAway == false
- effects: CopAway = true



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Robber: Actions II

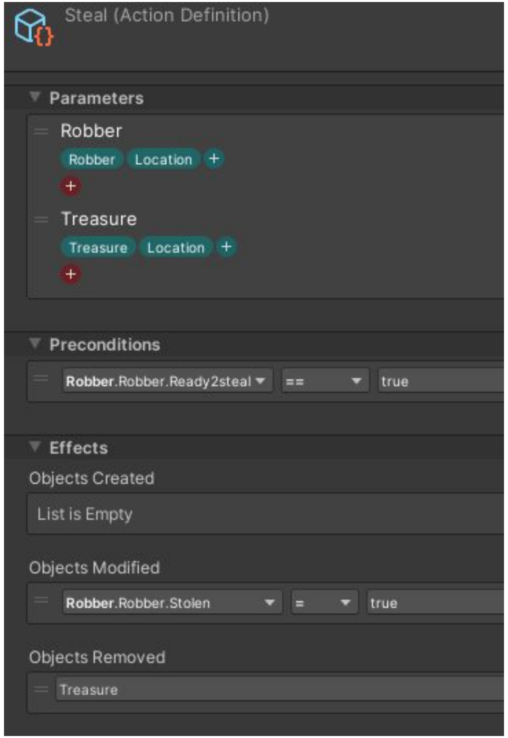
Robber

Approach

- parameters: cop, robber, treasure
- precondition: CopAway == true, Ready2steal false
- effect: Ready2steal = true

Steal

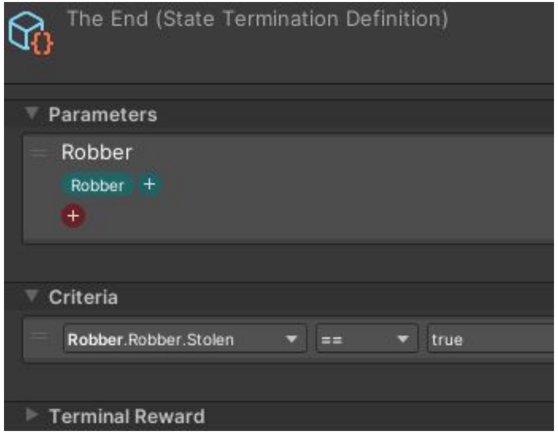
- parameters: robber, treasure
- precondition: Ready2steal == true
- effect: Stolen = true, treasure removed



Robber : Plan

Plan
create - AI - Planner - Problem Definition

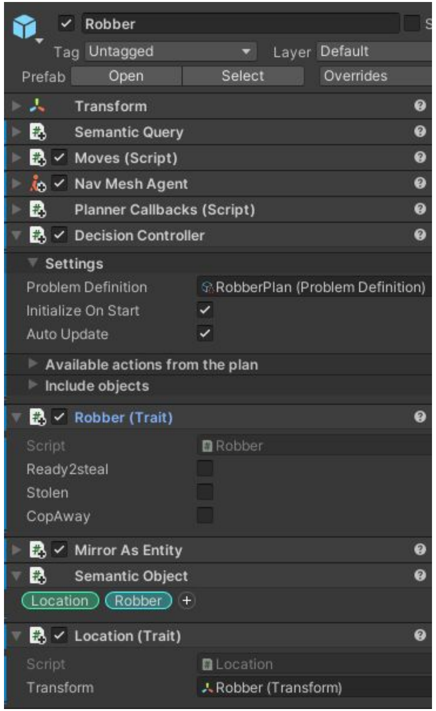
Termination criteria
create - AI - Planner - Termination Definition



Building the Plan
Menu - AI - Planner - Build

Robber: Configuring the Scene

1. Add Component - Semantic Object
   to the GameObjects
2. Add Component -DecisionController
   to the AI agent GameObject
   - Add the plan definition
   - Add the world objects
     with traits
3. Create and link the callbacks...



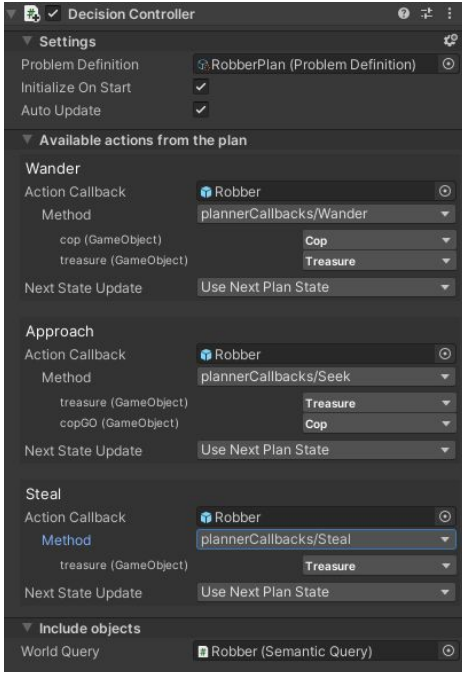
Robber: Action Callbacks

- ActionDefinitions components are not applied to the scene
- It is the Action Callbacks goal
- Coroutine is the choice for

  actions that execute over multiple frames

Robber

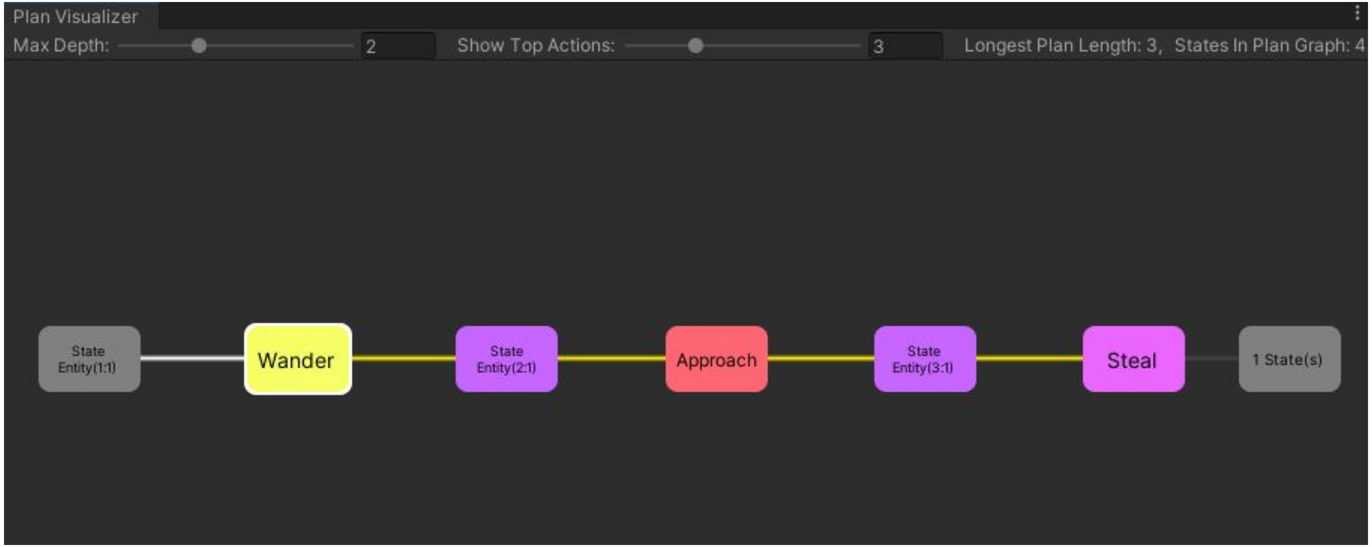
- C# view*



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

AI Planner:Debugging

Window - AI -Plan Visualizer



Source & documentation

AI Planner: dynamic planning

Example:

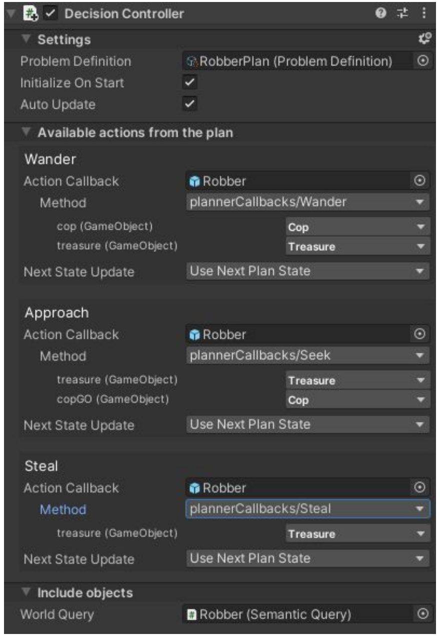
Non linear behaviour of Robber

Traits in scripts

Approach: Next State Update

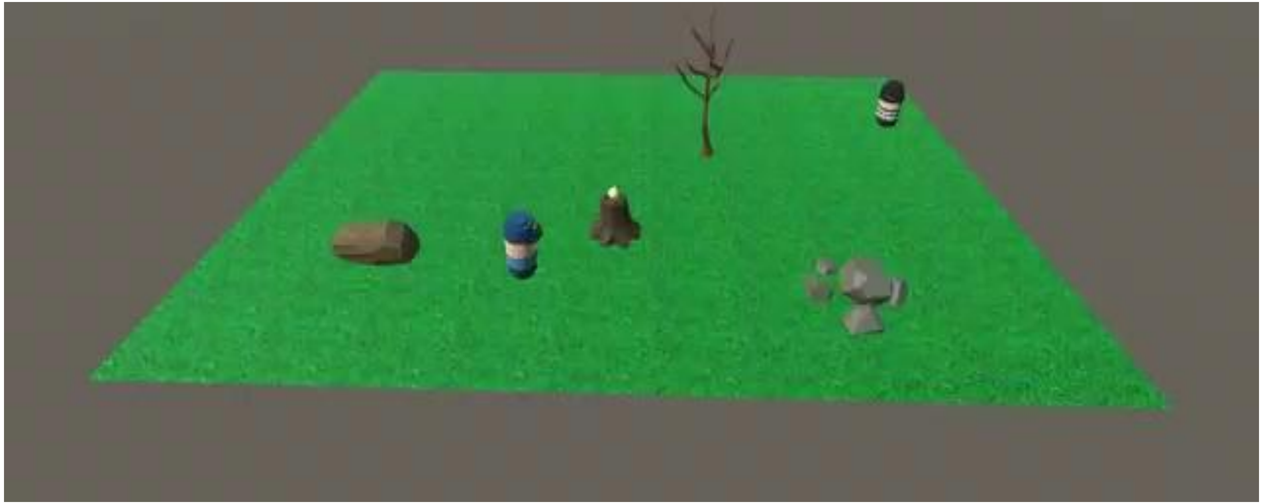
C# code

- view* /cs
-



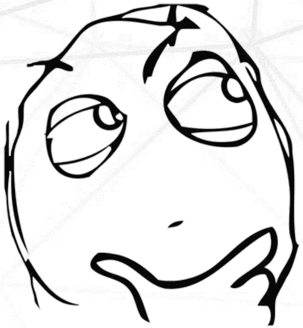
The result

- Robber video



Overview

- Introduction
- Finite State Machines
- Decision Trees
- Behaviour Trees
- Planning Systems
- References



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

References

- Ian Millington. *AI for Games* (3rd edition). CRC Press, 2019.
- Penny de Byl. *Artificial Intelligence for Beginners*. Unity Course.
- Chris Simpson. *Behavior trees for AI: How they work*. Gamasutra, 2014.
- Unity Technologies. AI Planner. 2020.
- Damian Isla. *Handling Complexity in the Halo 2 AI*. GDC, 2005.
- Ricard Pillosu. *Previous year slides of the AI course*, 2019.