Needs-Based AI Simulation

This is a type of simulation where agents determine their actions based on a set of competing needs. The code package can be used to create a wide range of games such as theme park management simulations and city builder simulations.

As a performant CPU driven simulation, the number of active agents on a map can be throttled to maximized effect on a users machine.

Core Architecture Overview

Map Element

Anything that appears on the map is a map element. All map elements have a set of quantified traits associated with them.

Agent

A map element that has state and seeks to fulfill a set of desired traits it finds in other map elements. It also broadcasts its own set of traits to other agents listening within its broadcast radius.

Item

A map element that has state and broadcasts its set of traits to listening agents within its broadcast radius.

Traits

A quantifiable object that has an id. Similar to a tag with a quantity property.

Command

A modular class used by command queues that start and complete a set of code.

Commandable State

A modular state that can have commands added to it. The added commands enter a queue that, when played, iterate in sequence through each command. The commands can also be added on optional layers for a multi-queue effect. Conceptually, this is very similar to an animation timeline with multiple layers and instead of frames, there are commands.

Advertisement

An object received by agents that informs them about a list of traits at a given location.

Advertisement Broadcaster

Used by map elements to broadcast advertisements about their traits. Only map elements that share a broadcaster can communicate with each other.

Battle Demo

An example of how to use the AI Simulation package. In this example we have two types of agents, knights and miners. Along with the agents, we have one item, gold. The knights primarily seek out enemy agents to attack and secondarily seek out gold. The miners only seek out gold, but will defend themselves if attacked.

How To Overview

The following are the higher level steps taken to create the battle demo.

1. Traits were defined by going to the create menu and selecting:*Create > Indie Dev Tools > Trait*
2. Agent datas with their assigned traits were defined for the knight and miner types by going to the create menu and selecting: *Create > Indie Dev Tools > Agent Data*
3. Item data with its assigned traits was defined for the gold by going to the create menu and selecting: *Create > Indie Dev Tools > Item Data*
4. An Advertising Broadcaster was created for agents and items to share by going to the create menu and selecting: *Create > Indie Dev Tools > Advertisement Broadcaster*
5. A Soldier component sub class of AbstractAgent was created and various commandable states were defined.
6. A scene was created and a game object was created with the Grid Map component alongside a Unity Grid component.
7. Two groups of knights and a miner group were created in the map using the Soldier component and assigning their relevant Agent Data to the data property.
8. A collection of gold items were created in the map using the Item component.
9. A canvas was created along with a UI game object that uses the Map Info Displayer component.

Map Generation

One way to optimize CPU performance is to throttle the amount of active agents in a simulation. In the Generated Map scene, an example of this is shown by use of the Map Generator component found on the Map.

The fill percentage property is used to fill x percentage of the map with map elements defined in the datas property list. An expanded implementation could expose this type of property to users so that the simulation effects can be maximized for their individual machine.