NEEDSIM Life simulation

Generated by Doxygen 1.8.10

Fri Nov 27 2015 13:08:24

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 NEEDSIM Namespace Reference

Classes

- class Action

  We hope we wrote our example actions in a way that they can be integrated into your Finite State Machine, Behavior Tree, or Planner. We provide a sample use of our sample actions in the PlanDemo.cs class.
- class Blackboard

  This is used to store some values, and make some methods available in a place where they can be edited without affecting the other classes.
- class ChaseSlot

  This is an example for how a chasing behavior could be implemented in NEEDSIM Life simulation. For a specific game better solutions might be desirable.
- class DecideGoal

  For goal oriented behaviors: Get a goal from the simulation, and try to get a slot where the goal can be satisfied.
- class DecideValue

  For Value-oriented behaviors: Try to get a slot based on utility of all available slots (relative to the current satisfaction level of the needs of the agent).
- class InterruptionFuchsalarm

  This action demonstrates how interruption of typical NEEDSIM behaviors could look like.
- class MoveToSlot

  Moving to a slot. The best implementation for such a behavior might be different in your project, but this script offers a starting point.
- class NEEDSIMManager

  This class stores the values that the NEEDSIMROOT will use for running the simulation
- class NEEDSIMNode

  Every object and agent in NEEDSIM Life simulation has a NEEDSIMNode: This is the essential component for using NEEDSIM Life simulation.
- class NEEDSIMRoot

  Every scene should have one root node for the AFFORDANCE TREE. This uses the settings of the NEEDSIM Manager and controls the simulation.
- class PlanDemo

  A simple behavior control solution. We tried to write this in a way that makes it easy to use our code samples in Finite State Machines, Behavior Trees and Goal-oriented Action Planning. The idea is that you can run our simulation from within a different solution, for example in case you want to have agents with fighting capabilites.
- class SatisfyGoal

  Participate in a slot to satisfy a goal.
- class SatisfyUrgentNeed

  Participating a slot. The respective behavior for value/urgency oriented behaviors.

4.2 NEEDSIM3rdParty Namespace Reference

Classes

- class Singleton

  Be aware this will not prevent a non singleton constructor such as `T myT = new T();` *To prevent that, add* `protected T () {}` *to your singleton class.*

4.3 NEEDSIMEditor Namespace Reference

4.4 NEEDSIMSampleSceneScripts Namespace Reference

Classes

- class Animal

  This example suggests an idea for playing animations based on the states of NEEDSIMNodes.
- class Bunny

  This example script shows how the bunny can deal with the 'EatBunny' interaction, that can be performed by a fox at the slot provided by a bunny.
- class Deer

  The deer currently has no special features. Rather check out the fox and the bunny for now ;)
- class Fox

  Sets the fox to either run or walk
- class FuchsalarmDemoScript

  This script shows how the behaviors of the bunnies are interrupted when the fox is spawned.
- class InputFieldRuntimeEditing

  Helper to change need satisfaction rates of interactions and satisfaction change rates of needs at runtime in a UI. It provides a method to react to the user finishing his input to the InputField.
- class Lake

  This example suggests an idea for how a variety of animations can be played at an interactive object.
- class NeedsUI

  This class shows bars for need satisfaction. A full bar equals full satisfaction, an empty bar means the need is not satisfied. If the need is currently being satisfied an outline will be added.
- class SampleCameraControl

  A very simple scrolling camera for NEEDSIM Life simulation example scenes.
- class SceneSwitcher

  public methods to switch scenes via a button click. You have to add the scenes to your build settings to use the prefab that uses this script.
- class SimpleSpawn

  This is a spawning example script to maintain populations
- class SpawnBedsManager

  This example shows how you could spawn all the objects and agents procedurally.
- class SpawnUIRuntimeEditing

  Spawns a UI Element for each need and each satisfaction rate of an interaction.
- class TimeSystem

  This example uses arrays with 24 values each to modify how behaviors are evaluated at a specific time of day. This class works with Value Oriented behaviors, not with Goal Oriented behaviors, because it relies on the fact that all opportunities to satisfy needs are evaluated, not only the opportunities that can satisfy the need of the current goal.

4.5 Simulation Namespace Reference

Namespaces

- namespace Tests

Classes

- class Affordance

  An affordance is the opportunity granted by an object to an agent to perform an action.
- class AffordanceTreeNode

  Each game object managed by the simulation has an Affordance Tree Node. It manages scope, affordances, slots, and, if the affordance tree is an active part of the simulation, its goals and levels of satisfaction.
- class DatabaseAsset

  Stores the data used by the NEEDSIM Life simulation
- class GeneralSettings

  A number of values used as configuration of the simulation and editor.
- class Goal

  A goal is a satisfaction level of a need that an agent wants to achieve.
- class Interaction

  An interaction at runtime
- class InteractionData

  From these data items used in the editor the interactions for the runtime will be generated.
- class Manager

  A singleton for managing the simulation at runtime
- class NeedItem

  A need, one of the core building blocks of NEEDSIMLifeSimulation
- class Needs

  The Needs class provides methods for interacting with the satisfaction levels of agents.
- class SimulationData

  The data loaded in the simulation at runtime
- class Slot

  A slot is a position in the world where an agent can run the interactions provided by the object that offers the slot.
- class Species

  A species is a set of needs. For example zombies might only have the 'Hunger' need, whereas humans furthermore have a 'Social' need.
- class StringFloatPair

  A class that helps creating key value pairs.
- class Strings

  A centralized place for many of the strings used by the NEEDSIM Life simulation.

4.6 Simulation.Tests Namespace Reference

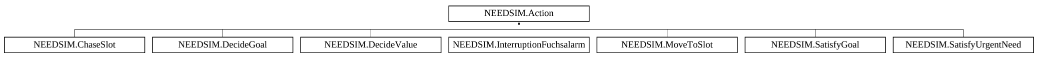
Chapter 5

Class Documentation

5.1 NEEDSIM.Action Class Reference

We hope we wrote our example actions in a way that they can be integrated into your Finite State Machine, Behavior Tree, or Planner. We provide a sample use of our sample actions in the PlanDemo.cs class.

Inheritance diagram for NEEDSIM.Action:



5.1.1 Detailed Description

We hope we wrote our example actions in a way that they can be integrated into your Finite State Machine, Behavior Tree, or Planner. We provide a sample use of our sample actions in the PlanDemo.cs class.

The documentation for this class was generated from the following file:

- Action.cs

5.2 Simulation.Affordance Class Reference

An affordance is the opportunity granted by an object to an agent to perform an action.

Public Member Functions

- Slot.Result ProlongLastInteraction ()

  Prolong the last interaction. Only available if interaction already finished, will then restart without setting the frame the animation started in.
- Slot.Result StartInteraction (string name)

  Start Interaction by name.
- Slot.Result StartRandomInteraction ()

  From all interactions registered at this affordance, pick one randomly
- bool AddInteraction (Interaction interaction)

  Make an interaction available at this affordance

Properties

- bool InteractionStartedThisFrame `[get]`

  Whether the interaction was started in the current frame
- Interaction CurrentInteraction `[get]`

  This will return null if the remaining duration is 0.
- bool HasInteraction `[get]`

  Whether an actual Interaction is available behind this affordance.

5.2.1 Detailed Description

An affordance is the opportunity granted by an object to an agent to perform an action.

5.2.2 Member Function Documentation

5.2.2.1 bool Simulation.Affordance.AddInteraction (Interaction *interaction*)

Make an interaction available at this affordance

Parameters

interaction	The interaction you want to add.

5.2.2.2 Slot.Result Simulation.Affordance.ProlongLastInteraction ( )

Prolong the last interaction. Only available if interaction already finished, will then restart without setting the frame the animation started in.

Returns

NoProlongableInteraction if none is found. The result of StartInteraction otherwise.

5.2.2.3 Slot.Result Simulation.Affordance.StartInteraction (string *name*)

Start Interaction by name.

Parameters

name	

Returns

5.2.2.4 Slot.Result Simulation.Affordance.StartRandomInteraction ( )

From all interactions registered at this affordance, pick one randomly

Returns

5.2.3 Property Documentation

5.2.3.1 Interaction Simulation.Affordance.CurrentInteraction [get]

This will return null if the remaining duration is 0.

5.2.3.2 bool Simulation.Affordance.HasInteraction [get]

Whether an actual Interaction is available behind this affordance.

5.2.3.3 bool Simulation.Affordance.InteractionStartedThisFrame [get]

Whether the interaction was started in the current frame

The documentation for this class was generated from the following file:

- Affordance.cs

5.3 Simulation.AffordanceTreeNode Class Reference

Each game object managed by the simulation has an Affordance Tree Node. It manages scope, affordances, slots, and, if the affordance tree is an active part of the simulation, its goals and levels of satisfaction.

Public Member Functions

- AffordanceTreeNode (AffordanceTreeNode parent, string name, string speciesName, Vector3 position)

  Create a new AffordanceTreeNode and attach it to the parent node
- Slot AvailableSlot (bool consumeSlot)

  Returns a slot that has been won in an auction. Can be consumed in the process of asking for it.
- bool AcceptAuctionVictory (Slot slot)

  Accepting a slot (for which a bid was previously placed).
- bool AddSlot (Vector3 worldPosition, Vector3 localPosition)

  Adds a slot (a location to run interactions) to this Affordance Tree Node.
- bool AddSlot (Vector3 worldPosition, Vector3 localPosition, Vector3 lookAtTarget)

  Adds a slot (a location to run interactions) to this Affordance Tree Node.
- bool AddSlot (Vector3 worldPosition, Vector3 localPosition, Vector3 lookAtTarget, bool isAuctionable)

  Adds a slot (a location to run interactions) to this Affordance Tree Node.
- void ApplyParentInteraction ()

  Apply the effects of the interaction that is running at the parent node to this node.
- bool setSpecies (string name)

  Set the species of this node to a specific kind.
- void Remove ()

  Remove/Delete this affordance tree node at an appropriate time
- string Printer (bool reportAffordances)

  Returns a string with debug information. Recursive (reports this node and its children).
- int CountOfChildren ()

  How many children (Affordance Tree nodes) this node has.

Properties

- Affordance Affordance `[get]`

  The opportunities for interaction at this node, and an abstraction of the opportunities to interact with children nodes.
- Species Species `[get]`

  A species determines which needs an agent has. Only required for agents.
- Goal Goal `[get, set]`

  Goals are optional, and can be used to make decisions.
- Needs SatisfactionLevels `[get]`

  Each need has a value between its min and max value: its satisfaction level.
- string MostUrgentNeed `[get]`

  Which need is most urgent, taking weights as well as current satisfaction level into account.
- bool CurrentInteractionSatisfiesMostUrgentNeed `[get]`

  Whether the interaction this node is currently participating in (if any) can satisfy his most urgent need (taking weights as well as current satisfaction level into account)
- List< Slot > Slots `[get]`

  All the slots that belong to this Affordance Tree Node
- AffordanceTreeNode Parent `[get, set]`

  The parent of this node
- AffordanceTreeNode Root `[get]`

  The root of the Affordance Tree

5.3.1 Detailed Description

Each game object managed by the simulation has an Affordance Tree Node. It manages scope, affordances, slots, and, if the affordance tree is an active part of the simulation, its goals and levels of satisfaction.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Simulation.AffordanceTreeNode.AffordanceTreeNode (AffordanceTreeNode *parent*, string *name*, string *speciesName*, Vector3 *position*)

Create a new AffordanceTreeNode and attach it to the parent node

Parameters

parent	The parent in the Affordance Tree
name	The name of the node, useful for debugging
speciesName	If this is an agent, which species it is
position	World position

5.3.3 Member Function Documentation

5.3.3.1 bool Simulation.AffordanceTreeNode.AcceptAuctionVictory (Slot *slot*)

Accepting a slot (for which a bid was previously placed).

Parameters

slot	The slot this affordance tree node should consume (usually resulting in going to the slot and running an interaction)

Returns

Whether a slot was made available for consumption

5.3.3.2 bool Simulation.AffordanceTreeNode.AddSlot (Vector3 *worldPosition*, Vector3 *localPosition*)

Adds a slot (a location to run interactions) to this Affordance Tree Node.

Parameters

worldPosition	The absolute position in the game world
localPosition	The position relative to the parent transform

Returns

Whether the slot was successfully added

5.3.3.3 bool Simulation.AffordanceTreeNode.AddSlot (Vector3 *worldPosition*, Vector3 *localPosition*, Vector3 *lookAtTarget*)

Adds a slot (a location to run interactions) to this Affordance Tree Node.

Parameters

worldPosition	The absolute position in the game world
localPosition	The position relative to the parent transform
lookAtTarget	A position that characters can be oriented towards for better animation results

Returns

Whether the slot was successfully added

5.3.3.4 bool Simulation.AffordanceTreeNode.AddSlot (Vector3 *worldPosition*, Vector3 *localPosition*, Vector3 *lookAtTarget*, bool *isAuctionable*)

Adds a slot (a location to run interactions) to this Affordance Tree Node.

Parameters

worldPosition	The absolute position in the game world
localPosition	The position relative to the parent transform
lookAtTarget	A position that characters can be oriented towards for better animation results
isAuctionable	If the slot is not auctionable it will not be offered to agents.

Returns

Whether the slot was successfully added

5.3.3.5 void Simulation.AffordanceTreeNode.ApplyParentInteraction ( )

Apply the effects of the interaction that is running at the parent node to this node.

5.3.3.6 Slot Simulation.AffordanceTreeNode.AvailableSlot (bool *consumeSlot*)

Returns a slot that has been won in an auction. Can be consumed in the process of asking for it.

Parameters

consumeSlot	Whether or not the slot should be consumed

Returns

The slot that can be consumed

5.3.3.7 int Simulation.AffordanceTreeNode.CountOfChildren ( )

How many children (Affordance Tree nodes) this node has.

Returns

5.3.3.8 string Simulation.AffordanceTreeNode.Printer (bool *reportAffordances*)

Returns a string with debug information. Recursive (reports this node and its children).

Parameters

report↩ Affordances	Adds detailed information about the affordances to the debug info.

Returns

String with debug information

5.3.3.9 void Simulation.AffordanceTreeNode.Remove ( )

Remove/Delete this affordance tree node at an appropriate time

5.3.3.10 bool Simulation.AffordanceTreeNode.setSpecies (string *name*)

Set the species of this node to a specific kind.

Parameters

name	The name (identifier) of the species

Returns

Whether the species was successfully set.

5.3.4 Property Documentation

5.3.4.1 Affordance Simulation.AffordanceTreeNode.Affordance [get]

The opportunities for interaction at this node, and an abstraction of the opportunities to interact with children nodes.

5.3.4.2 bool Simulation.AffordanceTreeNode.CurrentInteractionSatisfiesMostUrgentNeed [get]

Whether the interaction this node is currently participating in (if any) can satisfy his most urgent need (taking weights as well as current satisfaction level into account)

5.3.4.3 Goal Simulation.AffordanceTreeNode.Goal [get], [set]

Goals are optional, and can be used to make decisions.

5.3.4.4 string Simulation.AffordanceTreeNode.MostUrgentNeed [get]

Which need is most urgent, taking weights as well as current satisfaction level into account.

5.3.4.5 AffordanceTreeNode Simulation.AffordanceTreeNode.Parent [get], [set]

The parent of this node

5.3.4.6 AffordanceTreeNode Simulation.AffordanceTreeNode.Root [get]

The root of the Affordance Tree

5.3.4.7 Needs Simulation.AffordanceTreeNode.SatisfactionLevels [get]

Each need has a value between its min and max value: its satisfaction level.

5.3.4.8 List<Slot> Simulation.AffordanceTreeNode.Slots [get]

All the slots that belong to this Affordance Tree Node

5.3.4.9 Species Simulation.AffordanceTreeNode.Species [get]

A species determines which needs an agent has. Only required for agents.

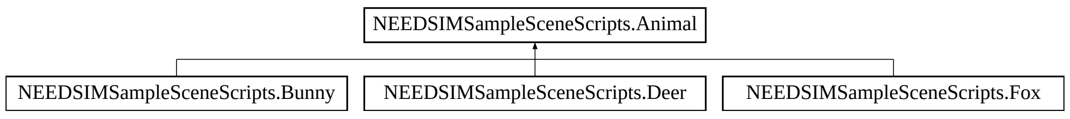
The documentation for this class was generated from the following file:

- AffordanceTreeNode.cs

5.4 NEEDSIMSampleSceneScripts.Animal Class Reference

This example suggests an idea for playing animations based on the states of NEEDSIMNodes.

Inheritance diagram for NEEDSIMSampleSceneScripts.Animal:


NEEDSIMSampleSceneScripts.Animal
NEEDSIMSampleSceneScripts.Bunny
NEEDSIMSampleSceneScripts.Deer
NEEDSIMSampleSceneScripts.Fox

5.4.1 Detailed Description

This example suggests an idea for playing animations based on the states of NEEDSIMNodes.

The documentation for this class was generated from the following file:

- Animal.cs

5.5 NEEDSIM.Blackboard Class Reference

This is used to store some values, and make some methods available in a place where they can be edited without affecting the other classes.

Public Member Functions

- bool HasArrivedAtSlot ()

  Depending on your game and your NavMesh you might have to change the conditions here.
- bool slotToAgentDistanceSmall (Vector3 agentPosition)

  Whether the slot and the agent are as close as defined in the smallDistance value
- bool AcceptSlot (Simulation.Slot slot)

  Do the necessary stept to follow up on accepting a slot. Set destination on nav mesh, and put agent into the correct state

5.5.1 Detailed Description

This is used to store some values, and make some methods available in a place where they can be edited without affecting the other classes.

5.5.2 Member Function Documentation

5.5.2.1 bool NEEDSIM.Blackboard.AcceptSlot (Simulation.Slot *slot*)

Do the necessary stept to follow up on accepting a slot. Set destination on nav mesh, and put agent into the correct state

Parameters

slot	

Returns

5.5.2.2 bool NEEDSIM.Blackboard.HasArrivedAtSlot ( )

Depending on your game and your NavMesh you might have to change the conditions here.

Returns

Whether the NavMeshAgent has arrived at the slot.

5.5.2.3 bool NEEDSIM.Blackboard.slotToAgentDistanceSmall (Vector3 *agentPosition*)

Whether the slot and the agent are as close as defined in the smallDistance value

Parameters

agentPosition	

Returns

true if agent is closer than small distance to target

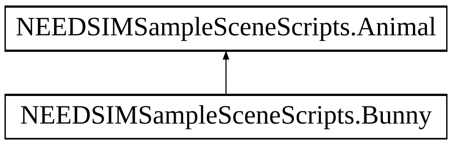
The documentation for this class was generated from the following file:

- Blackboard.cs

5.6 NEEDSIMSampleSceneScripts.Bunny Class Reference

This example script shows how the bunny can deal with the 'EatBunny' interaction, that can be performed by a fox at the slot provided by a bunny.

Inheritance diagram for NEEDSIMSampleSceneScripts.Bunny:


NEEDSIMSampleSceneScripts.Animal
NEEDSIMSampleSceneScripts.Bunny

5.6.1 Detailed Description

This example script shows how the bunny can deal with the 'EatBunny' interaction, that can be performed by a fox at the slot provided by a bunny.

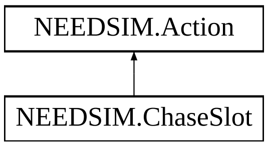
The documentation for this class was generated from the following file:

- Bunny.cs

5.7 NEEDSIM.ChaseSlot Class Reference

This is an example for how a chasing behavior could be implemented in NEEDSIM Life simulation. For a specific game better solutions might be desirable.

Inheritance diagram for NEEDSIM.ChaseSlot:



Public Member Functions

- override Action.Result Run ()

  Go to the slot that has been given to the agent.

5.7.1 Detailed Description

This is an example for how a chasing behavior could be implemented in NEEDSIM Life simulation. For a specific game better solutions might be desirable.

5.7.2 Member Function Documentation

5.7.2.1 override Action.Result NEEDSIM.ChaseSlot.Run ( ) [virtual]

Go to the slot that has been given to the agent.

Returns

Running as long as on the way. Success upon arrival.

Implements NEEDSIM.Action.

The documentation for this class was generated from the following file:

- ChaseSlot.cs

5.8 Simulation.DatabaseAsset Class Reference

Stores the data used by the NEEDSIM Life simulation

Inherits ScriptableObject.

Public Member Functions

- void Init (string databaseName, List< NeedItem > needs, List< Species > species, List< InteractionData > interactions, bool isDefaultDatabase)

  Initializes the database asset with values
- string[ ] GetNeedNames ()

  Constructs a new array with the names of all needs.
- string[ ] GetSpeciesNames ()

  Constructs a new array with the names of all species.
- string[ ] GetInteractionNames ()

  Constructs a new array with the names of all interactions.

Public Attributes

- string DatabaseName

  The name of the database
- bool isDefault

  Whether this is the one default database.
- List< NeedItem > NeedsList

  All the needs in the database
- List< Species > Species

  All the species in the database
- List< InteractionData > Interactions

  All the interactions in the database.

5.8.1 Detailed Description

Stores the data used by the NEEDSIM Life simulation

5.8.2 Member Function Documentation

5.8.2.1 string [ ] Simulation.DatabaseAsset.GetInteractionNames ( )

Constructs a new array with the names of all interactions.

Returns

A new array with the name of all interactions.

5.8.2.2 string [ ] Simulation.DatabaseAsset.GetNeedNames ( )

Constructs a new array with the names of all needs.

Returns

A new array with the name of all needs.

5.8.2.3 string [ ] Simulation.DatabaseAsset.GetSpeciesNames ( )

Constructs a new array with the names of all species.

Returns

A new array with the name of all species.

5.8.2.4 void Simulation.DatabaseAsset.Init (string *databaseName*, List< **NeedItem** > *needs*, List< **Species** > *species*, List< **InteractionData** > *interactions*, bool *isDefaultDatabase*)

Initializes the database asset with values

Parameters

databaseName	The name of the database
needs	The needs in the database
species	The species in the databse
interactions	The interactions in the database
isDefault←Database	Whether this is the one default database.

5.8.3 Member Data Documentation

5.8.3.1 string Simulation.DatabaseAsset.DatabaseName

The name of the database

5.8.3.2 List<**InteractionData**> Simulation.DatabaseAsset.Interactions

All the interactions in the database.

5.8.3.3 bool Simulation.DatabaseAsset.isDefault

Whether this is the one default database.

5.8.3.4 List< NeedItem > Simulation.DatabaseAsset.NeedsList

All the needs in the database

5.8.3.5 List< Species > Simulation.DatabaseAsset.Species

All the species in the database

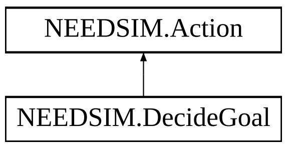
The documentation for this class was generated from the following file:

- DatabaseAsset.cs

5.9 NEEDSIM.DecideGoal Class Reference

For goal oriented behaviors: Get a goal from the simulation, and try to get a slot where the goal can be satisfied.

Inheritance diagram for NEEDSIM.DecideGoal:



Public Member Functions

- override Action.Result Run ()

  Get a goal from the simulation, and try to get a slot where the goal can be satisfied.

5.9.1 Detailed Description

For goal oriented behaviors: Get a goal from the simulation, and try to get a slot where the goal can be satisfied.

5.9.2 Member Function Documentation

5.9.2.1 override Action.Result NEEDSIM.DecideGoal.Run ( ) [virtual]

Get a goal from the simulation, and try to get a slot where the goal can be satisfied.

Returns

Success if a slot has been distributed to the agent.

Implements NEEDSIM.Action.

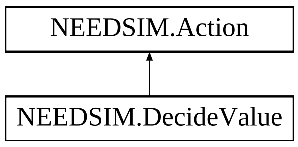
The documentation for this class was generated from the following file:

- DecideGoal.cs

5.10 NEEDSIM.DecideValue Class Reference

For Value-oriented behaviors: Try to get a slot based on utility of all available slots (relative to the current satisfaction level of the needs of the agent).

Inheritance diagram for NEEDSIM.DecideValue:



Public Member Functions

- override Action.Result Run ()

  Try to get a slot based on utility of all available slots (relative to the current satisfaction level of the needs of the agent).

5.10.1 Detailed Description

For Value-oriented behaviors: Try to get a slot based on utility of all available slots (relative to the current satisfaction level of the needs of the agent).

5.10.2 Member Function Documentation

5.10.2.1 override Action.Result NEEDSIM.DecideValue.Run ( ) [virtual]

Try to get a slot based on utility of all available slots (relative to the current satisfaction level of the needs of the agent).

Returns

Success if a slot has been distributed to the agent.

Implements NEEDSIM.Action.

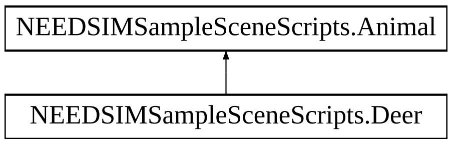
The documentation for this class was generated from the following file:

- DecideValue.cs

5.11 NEEDSIMSampleSceneScripts.Deer Class Reference

The deer currently has no special features. Rather check out the fox and the bunny for now ;)

Inheritance diagram for NEEDSIMSampleSceneScripts.Deer:


NEEDSIMSampleSceneScripts.Animal
NEEDSIMSampleSceneScripts.Deer

5.11.1 Detailed Description

The deer currently has no special features. Rather check out the fox and the bunny for now ;)

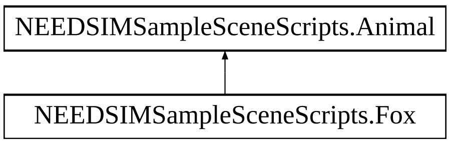
The documentation for this class was generated from the following file:

- Deer.cs

5.12 NEEDSIMSampleSceneScripts.Fox Class Reference

Sets the fox to either run or walk

Inheritance diagram for NEEDSIMSampleSceneScripts.Fox:


NEEDSIMSampleSceneScripts.Animal
NEEDSIMSampleSceneScripts.Fox

5.12.1 Detailed Description

Sets the fox to either run or walk

The documentation for this class was generated from the following file:

- Fox.cs

5.13 NEEDSIMSampleSceneScripts.FuchsalarmDemoScript Class Reference

This script shows how the behaviors of the bunnies are interrupted when the fox is spawned.

Inherits MonoBehaviour.

Public Member Functions

- void SpawnTheFox ()

  Call this to create a fox and interrupt all the bunnies.

5.13.1 Detailed Description

This script shows how the behaviors of the bunnies are interrupted when the fox is spawned.

5.13.2 Member Function Documentation

5.13.2.1 void NEEDSIMSampleSceneScripts.FuchsalarmDemoScript.SpawnTheFox ( )

Call this to create a fox and interrupt all the bunnies.

The documentation for this class was generated from the following file:

- FuchsalarmDemoScript.cs

5.14 Simulation.GeneralSettings Class Reference

A number of values used as configuration of the simulation and editor.

Properties

- static Color SlotColor `[get]`

  The standard color of a slot being drawn in the scene view.
- static Color AuctionableSlotColor `[get]`

  If a slot can be auctioned, that is offered to agents, this color is available.
- static Color BlockedSlotColor `[get]`

  If the slot is blocked it is drawn in this color.
- static Color ReservedSlotColor `[get]`

  The color of a currently reserved slot.
- static Color ReadyCharacterSlotColor `[get]`

  Whether a character is at the slot and ready to participate.
- static float SlotRepresentationRadius `[get]`

  The radius of the circle or sphere around a slot postion.
- static float SlotRepresentationHandleRadius `[get]`

  The radius/size of the handle of a slot.
- static string DefaultNeedName `[get]`

  Default value for newly created needs. The name of the need.
- static float DefaultNeedMinValue `[get]`

  Default value for newly created needs. Need satisfaction can not be below this value.
- static float DefaultNeedMaxValue `[get]`

  Default value for newly created needs. Need satisfaction can not be above this value.
- static float DefaultNeedCriticalState `[get]`

  Default value for newly created needs. Below this value the need is considered to be in a critical state.
- static float DefaultNeedSatisfiedState `[get]`

  Default value for newly created needs. Above this value the need is considered to be in a satisfied state.
- static float DefaultNeedChangePerSecondRate `[get]`

  Default value for newly created needs. At which rate needs become unsatisfied over time

5.14.1 Detailed Description

A number of values used as configuration of the simulation and editor.

5.14.2 Property Documentation

5.14.2.1 Color Simulation.GeneralSettings.AuctionableSlotColor `[static]`, `[get]`

If a slot can be auctioned, that is offered to agents, this color is available.

5.14.2.2 Color Simulation.GeneralSettings.BlockedSlotColor `[static]`, `[get]`

If the slot is blocked it is drawn in this color.

5.14.2.3 float Simulation.GeneralSettings.DefaultNeedChangePerSecondRate `[static]`, `[get]`

Default value for newly created needs. At which rate needs become unsatisfied over time

5.14.2.4 float Simulation.GeneralSettings.DefaultNeedCriticalState [static], [get]

Default value for newly created needs. Below this value the need is considered to be in a critical state.

5.14.2.5 float Simulation.GeneralSettings.DefaultNeedMaxValue [static], [get]

Default value for newly created needs. Need satisfaction can not be above this value.

5.14.2.6 float Simulation.GeneralSettings.DefaultNeedMinValue [static], [get]

Default value for newly created needs. Need satisfaction can not be below this value.

5.14.2.7 string Simulation.GeneralSettings.DefaultNeedName [static], [get]

Default value for newly created needs. The name of the need.

5.14.2.8 float Simulation.GeneralSettings.DefaultNeedSatisfiedState [static], [get]

Default value for newly created needs. Above this value the need is considered to be in a satisfied state.

5.14.2.9 Color Simulation.GeneralSettings.ReadyCharacterSlotColor [static], [get]

Whether a character is at the slot and ready to participate.

5.14.2.10 Color Simulation.GeneralSettings.ReservedSlotColor [static], [get]

The color of a currently reserved slot.

5.14.2.11 Color Simulation.GeneralSettings.SlotColor [static], [get]

The standard color of a slot being drawn in the scene view.

5.14.2.12 float Simulation.GeneralSettings.SlotRepresentationHandleRadius [static], [get]

The radius/size of the handle of a slot.

5.14.2.13 float Simulation.GeneralSettings.SlotRepresentationRadius [static], [get]

The radius of the circle or sphere around a slot postion.

The documentation for this class was generated from the following file:

- GeneralSettings.cs

5.15 Simulation.Goal Class Reference

A goal is a satisfaction level of a need that an agent wants to achieve.

Public Member Functions

- Goal (string needToSatisfy, Needs.NeedSatisfactions satisfactionState)

  Create a new goal to achieve a specific Needs.NeedSatisfactions state for a specific need.
- bool GoalAchieved (float satisfactionValue)

  True, if the specific NeedSatisfaction Goal is achieved. False otherwise.

Properties

- string NeedToSatisfy `[get]`

  Which need should be satisfied by this goal.
- Needs.NeedSatisfactions SatisfactionState `[get]`

  What is the current Needs.NeedSatisfactions state of the need this goal is concerned with.
- bool HasBeenAchieved `[get, set]`

  Whether the goal has been achieved.

5.15.1 Detailed Description

A goal is a satisfaction level of a need that an agent wants to achieve.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Simulation.Goal.Goal (string *needToSatisfy,* Needs.NeedSatisfactions *satisfactionState*)

Create a new goal to achieve a specific Needs.NeedSatisfactions state for a specific need.

Parameters

needToSatisfy	The name of the need
satisfactionState	The state that is desired

5.15.3 Member Function Documentation

5.15.3.1 bool Simulation.Goal.GoalAchieved (float *satisfactionValue*)

True, if the specific NeedSatisfaction Goal is achieved. False otherwise.

Parameters

satisfactionValue	

Returns

5.15.4 Property Documentation

5.15.4.1 bool Simulation.Goal.HasBeenAchieved `[get]`, `[set]`

Whether the goal has been achieved.

5.15.4.2 string Simulation.Goal.NeedToSatisfy `[get]`

Which need should be satisfied by this goal.

5.15.4.3 Needs.NeedSatisfactions Simulation.Goal.SatisfactionState [get]

What is the current Needs.NeedSatisfactions state of the need this goal is concerned with.

The documentation for this class was generated from the following file:

- Goal.cs

5.16 NEEDSIMSampleSceneScripts.InputFieldRuntimeEditing Class Reference

Helper to change need satisfaction rates of interactions and satisfaction change rates of needs at runtime in a UI. It provides a method to react to the user finishing his input to the InputField.

Inherits MonoBehaviour.

Public Member Functions

- void EndInput (string newValue)

  Update the value if the user has ended his input.

5.16.1 Detailed Description

Helper to change need satisfaction rates of interactions and satisfaction change rates of needs at runtime in a UI. It provides a method to react to the user finishing his input to the InputField.

5.16.2 Member Function Documentation

5.16.2.1 void NEEDSIMSampleSceneScripts.InputFieldRuntimeEditing.EndInput (string *newValue*)

Update the value if the user has ended his input.

Parameters

newValue	

The documentation for this class was generated from the following file:

- InputFieldRuntimeEditing.cs

5.17 Simulation.Interaction Class Reference

An interaction at runtime

Public Member Functions

- Interaction (string Name, Dictionary< string, float > SatisfactionRates, float Duration, bool hasPreconditions, Dictionary< string, bool > AllowedSpecies, Dictionary< Needs.NeedSatisfactions, bool > Allowed↩ Satisfactions)

  Create a new interaction instance that will be available at simulation runtime.
- bool CheckPreconditions (AffordanceTreeNode affordanceTreeNode)

  For some interactions to be used conditions need to be fullfilled.

Properties

- string Name [get]

  The unique name of the need. Identifier.
- Dictionary< string, float > SatisfactionRates [get]

  The rate at which each need is decayed or satisfied per second whilst the interaction is performed.
- float Duration [get]

  The duration in seconds of the interaction.
- bool Prolongable [get]

  Whether the interaction can be prolonged for another duration intervall
- bool HasPreconditions [get]

  Whether this interaction has preconditions. Setting this to false can speed up the simulation at the cost of not evaluating preconditions.
- Dictionary< string, bool > SpeciesAllowed [get]

  Which species are allowed to participate this interaction (precondition)

5.17.1 Detailed Description

An interaction at runtime

5.17.2 Constructor & Destructor Documentation

5.17.2.1 Simulation.Interaction.Interaction (string *Name*, Dictionary< string, float > *SatisfactionRates*, float *Duration*, bool *hasPreconditions*, Dictionary< string, bool > *AllowedSpecies*, Dictionary< **Needs.NeedSatisfactions**, bool > *AllowedSatisfactions*)

Create a new interaction instance that will be available at simulation runtime.

Parameters

Name	
Satisfaction←↩ Rates	
Duration	
has←↩ Preconditions	
AllowedSpecies	
Allowed←↩ Satisfactions	

5.17.3 Member Function Documentation

5.17.3.1 bool Simulation.Interaction.CheckPreconditions (**AffordanceTreeNode** *affordanceTreeNode*)

For some interactions to be used conditions need to be fullfilled.

Parameters

affordance←↩ TreeNode	The AffordanceTreeNode that wants to use the Interaction

Returns

Whether all preconditions are fullfilled.

5.17.4 Property Documentation

5.17.4.1 float Simulation.Interaction.Duration [get]

The duration in seconds of the interaction.

5.17.4.2 bool Simulation.Interaction.HasPreconditions [get]

Whether this interaction has preconditions. Setting this to false can speed up the simulation at the cost of not evaluating preconditions.

5.17.4.3 string Simulation.Interaction.Name [get]

The unique name of the need. Identifier.

5.17.4.4 bool Simulation.Interaction.Prolongable [get]

Whether the interaction can be prolonged for another duration intervall

5.17.4.5 Dictionary< string, float > Simulation.Interaction.SatisfactionRates [get]

The rate at which each need is decayed or satisfied per second whilst the interaction is performed.

5.17.4.6 Dictionary< string, bool > Simulation.Interaction.SpeciesAllowed [get]

Which species are allowed to participate this interaction (precondition)

The documentation for this class was generated from the following file:

- Interaction.cs

5.18 Simulation.InteractionData Class Reference

From these data items used in the editor the interactions for the runtime will be generated.

Public Member Functions

- void Init (string name, List< StringFloatPair > satisfactionRates)

  Create the data for a new interaction.

Public Attributes

- string interactionName

  The unique name of the need. Identifier.
- List< StringFloatPair > satisfactions

  The rate at which each need is decayed or satisfied per second whilst the interaction is performed.
- float duration

  The duration in seconds of the interaction.
- bool doesHavePreconditions

Whether this interaction has preconditions. Setting this to false can speed up the simulation at the cost of not evaluating preconditions.

- List< string > SpeciesAllowed

  Which species are allowed to participate this interaction (precondition)
- List< string > AtSatisfactionLevels

  NOT YET IMPLEMENTED

5.18.1 Detailed Description

From these data items used in the editor the interactions for the runtime will be generated.

5.18.2 Member Function Documentation

5.18.2.1 void Simulation.InteractionData.Init (string *name*, List< StringFloatPair > *satisfactionRates*)

Create the data for a new interaction.

Parameters

name	The unique name of the need. Identifier.
satisfactionRates	The rate at which the interaction satisfies or decays each need per second.

5.18.3 Member Data Documentation

5.18.3.1 List<string> Simulation.InteractionData.AtSatisfactionLevels

NOT YET IMPLEMENTED

5.18.3.2 bool Simulation.InteractionData.doesHavePreconditions

Whether this interaction has preconditions. Setting this to false can speed up the simulation at the cost of not evaluating preconditions.

5.18.3.3 float Simulation.InteractionData.duration

The duration in seconds of the interaction.

5.18.3.4 string Simulation.InteractionData.interactionName

The unique name of the need. Identifier.

5.18.3.5 List<StringFloatPair> Simulation.InteractionData.satisfactions

The rate at which each need is decayed or satisfied per second whilst the interaction is performed.

5.18.3.6 List<string> Simulation.InteractionData.SpeciesAllowed

Which species are allowed to participate this interaction (precondition)

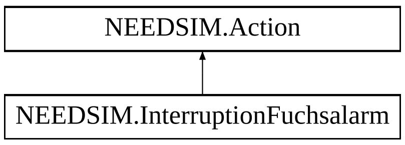
The documentation for this class was generated from the following file:

- InteractionData.cs

5.19 NEEDSIM.InterruptionFuchsalarm Class Reference

This action demonstrates how interruption of typical NEEDSIM behaviors could look like.

Inheritance diagram for NEEDSIM.InterruptionFuchsalarm:



Public Member Functions

- override Action.Result Run ()

  Satisfying a goal at an AffordanceTree node.

5.19.1 Detailed Description

This action demonstrates how interruption of typical NEEDSIM behaviors could look like.

5.19.2 Member Function Documentation

5.19.2.1 override Action.Result NEEDSIM.InterruptionFuchsalarm.Run ( ) `[virtual]`

Satisfying a goal at an AffordanceTree node.

Returns

Success if need satsifaction goal was achieved, running whilst it is being satisfied.

Implements NEEDSIM.Action.

The documentation for this class was generated from the following file:

- InterruptionFuchsalarm.cs

5.20 NEEDSIMSampleSceneScripts.Lake Class Reference

This example suggests an idea for how a variety of animations can be played at an interactive object.

Inherits MonoBehaviour.

5.20.1 Detailed Description

This example suggests an idea for how a variety of animations can be played at an interactive object.

The documentation for this class was generated from the following file:

- Lake.cs

5.21 Simulation.Manager Class Reference

A singleton for managing the simulation at runtime

Public Member Functions

- Dictionary< string, float > ZeroValuedNeeds (List< string > needNames)

  Get a new Dictionary with the respecitve needs values at 0.0f
- bool UpdateAffordanceTree ()

  Update this Affordance Tree Node, and its children (recursive). Necessary to call for the simulation to work
- bool SetAllNeedSatisfactionWeightsToOne ()

  This essentially turns of the WeightsForNeedSatisfaction, as if they are all one it means when they are multiplied the original values are not changed.

Properties

- static Manager Instance `[get]`

  This is a singleton
- SimulationData Data `[get, set]`

  The data loaded into the simulation.

5.21.1 Detailed Description

A singleton for managing the simulation at runtime

5.21.2 Member Function Documentation

5.21.2.1 bool Simulation.Manager.SetAllNeedSatisfactionWeightsToOne ( )

This essentially turns of the WeightsForNeedSatisfaction, as if they are all one it means when they are multiplied the original values are not changed.

Returns

true, if there was no issue.

5.21.2.2 bool Simulation.Manager.UpdateAffordanceTree ( )

Update this Affordance Tree Node, and its children (recursive). Necessary to call for the simulation to work

5.21.2.3 Dictionary<string, float> Simulation.Manager.ZeroValuedNeeds (List< string > *needNames*)

Get a new Dictionary with the respecitve needs values at 0.0f

Parameters

needNames	The needs you want to construct the dictionary for

Returns

a dictionary with the value 0.0f for each need.

5.21.3 Property Documentation

5.21.3.1 SimulationData Simulation.Manager.Data `[get]`, `[set]`

The data loaded into the simulation.

5.21.3.2 Manager Simulation.Manager.Instance [static], [get]

This is a singleton

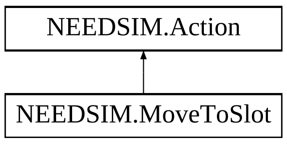
The documentation for this class was generated from the following file:

- Manager.cs

5.22 NEEDSIM.MoveToSlot Class Reference

Moving to a slot. The best implementation for such a behavior might be different in your project, but this script offers a starting point.

Inheritance diagram for NEEDSIM.MoveToSlot:



Public Member Functions

- override Action.Result Run ()

  Go to the slot that has been given to the agent.

5.22.1 Detailed Description

Moving to a slot. The best implementation for such a behavior might be different in your project, but this script offers a starting point.

5.22.2 Member Function Documentation

5.22.2.1 override Action.Result NEEDSIM.MoveToSlot.Run ( ) [virtual]

Go to the slot that has been given to the agent.

Returns

Running as long as on the way. Success upon arrival.

Implements NEEDSIM.Action.

The documentation for this class was generated from the following file:

- MoveToSlot.cs

5.23 Simulation.NeedItem Class Reference

A need, one of the core building blocks of NEEDSIMLifeSimulation

Public Member Functions

- NeedItem (string name, float changeRate, float critical, float satisfied)

  Create a new need.

Public Attributes

- string needName

  The unique name of the need. Identifier.
- float minValue

  The limit below which the need satisfaction level is capped.
- float maxValue

  The limit above which the the need satisfaction level is capped.
- float changePerSecond

  How much the need changes per second. For example how hungry a character gets over time
- float criticalState

  The limit below which the state of the need satisfcation is considered critical.
- float satisfiedState

  The limit above which the state of the need satisfaction is considered satisfied.

5.23.1 Detailed Description

A need, one of the core building blocks of NEEDSIMLifeSimulation

5.23.2 Constructor & Destructor Documentation

5.23.2.1 Simulation.NeedItem.NeedItem (string *name*, float *changeRate*, float *critical*, float *satisfied*)

Create a new need.

Parameters

name	The unique name of the need. Identifier.
changeRate	How much the need changes per second. For example how hungry a character gets over time
critical	The limit below which the state of the need satisfcation is considered critical.
satisfied	The limit above which the state of the need satisfaction is considered satisfied.

5.23.3 Member Data Documentation

5.23.3.1 float Simulation.NeedItem.changePerSecond

How much the need changes per second. For example how hungry a character gets over time

5.23.3.2 float Simulation.NeedItem.criticalState

The limit below which the state of the need satisfcation is considered critical.

5.23.3.3 float Simulation.NeedItem.maxValue

The limit above which the the need satisfaction level is capped.

5.23.3.4 float Simulation.NeedItem.minValue

The limit below which the need satisfaction level is capped.

5.23.3.5 string Simulation.NeedItem.needName

The unique name of the need. Identifier.

5.23.3.6 float Simulation.NeedItem.satisfiedState

The limit above which the state of the need satisfaction is considered satisfied.

The documentation for this class was generated from the following file:

- NeedItem.cs

5.24 Simulation.Needs Class Reference

The Needs class provides methods for interacting with the satisfaction levels of agents.

Public Types

- enum NeedSatisfactions {
  NeedSatisfactions.Unvalued = 0, NeedSatisfactions.Maximized, NeedSatisfactions.Satisfied, Need↩
  Satisfactions.Uncritical,
  NeedSatisfactions.Critical }

  Discrete states of satisfaction levels.

Public Member Functions

- Needs (Species species)

  Constructor
- void SetSpecficSatisfactionLevels (Dictionary< string, float > needLevels)

  Set the level of each need satisfaction to a particular value
- void ApplyChangeRates (Dictionary< string, float > input)

  Apply a specific set of change rates, for example the change rates provided by an interaction.
- void ApplyChangePerSecond ()

  Apply the change per second rates, that is decay (or increase) each need with its default global change per second rate
- Goal GoalToSatisfyLowestNeed ()

  Compute a goal to satisfy the numerically lowest need.
- void RandomizeValues ()

  Set a random value for each need satisfaction
- Dictionary< string, Needs.NeedSatisfactions > Satisfactions ()

  The discrete satisfaction states for each need
- float GetValue (string key)

5.24.1 Detailed Description

The Needs class provides methods for interacting with the satisfaction levels of agents.

5.24.2 Member Enumeration Documentation

5.24.2.1 enum Simulation.Needs.NeedSatisfactions [strong]

Discrete states of satisfaction levels.

Enumerator

Unvalued	Unfortunately no discrete value of the need satisfaction levels is available.
Maximized	The satisfaction level of the need is maximized.
Satisfied	The need is satisfied, but not yet maximized.
Uncritical	The need is below the satisfied limit, but not yet critical
Critical	The need satisfaction level is below or equal to the target value for critical satisfaction.

5.24.3 Constructor & Destructor Documentation

5.24.3.1 Simulation.Needs.Needs (Species *species*)

Constructor

Parameters

species	The species that the agent owning this set of needs belongs to

5.24.4 Member Function Documentation

5.24.4.1 void Simulation.Needs.ApplyChangePerSecond ( )

Apply the change per second rates, that is decay (or increase) each need with its default global change per second rate

5.24.4.2 void Simulation.Needs.ApplyChangeRates (Dictionary< string, float > *input*)

Apply a specific set of change rates, for example the change rates provided by an interaction.

Parameters

input	

5.24.4.3 float Simulation.Needs.GetValue (string *key*)

Parameters

key	The name of the need.

Returns

The satisfaction level for the specified need. NaN if the key is invalid.

5.24.4.4 Goal Simulation.Needs.GoalToSatisfyLowestNeed ( )

Compute a goal to satisfy the numerically lowest need.

Returns

5.24.4.5 void Simulation.Needs.RandomizeValues ( )

Set a random value for each need satisfation

5.24.4.6 Dictionary< string, Needs.NeedSatisfactions > Simulation.Needs.Satisfactions ( )

The discrete satisfaction states for each need

Returns

A dictionary which has the discrete states as values for each need.

5.24.4.7 void Simulation.Needs.SetSpecficSatisfactionLevels (Dictionary< string, float > *needLevels*)

Set the level of each need satisfaction to a particular value

Parameters

needLevels	The new satisfaction levels

The documentation for this class was generated from the following file:

- Needs.cs

5.25 NEEDSIM.NEEDSIMManager Class Reference

This class stores the values that the NEEDSIMROOT will use for running the simulation

Inherits MonoBehaviour.

5.25.1 Detailed Description

This class stores the values that the NEEDSIMROOT will use for running the simulation

The documentation for this class was generated from the following file:

- NEEDSIMManager.cs

5.26 NEEDSIM.NEEDSIMNode Class Reference

Every object and agent in NEEDSIM Life simulation has a NEEDSIMNode: This is the essential component for using NEEDSIM Life simulation.

Inherits MonoBehaviour.

Public Member Functions

- void BuildTreeBasedOnSceneHierarchy ()

  Recursively build an Affordance Tree from the scene hierarchy. This method will not work for intermediate objects in the hierarchy - there is only deeper search if a direct ancestor is a NEEDSIMNode.
- bool AcceptSlot ()

  If the simulation distributed a slot to this agent try to accept it.
- bool ArrivalAtSlot (Slot slot)

This method tries to call the AgentArrivalEasy() method at the slot it is passed to.

- bool TryConsumingAnimationOrder (Animator animator)

  This method assumes that in the animator a trigger named "Movement" exists, and that for each interaction a trigger with the same name exists in the animator, e.g. that for the interaction "Eat" a trigger named "Eat" is in the animation and is used to transition into a state or sub-state-machine that plays animation(s) for eating.

5.26.1 Detailed Description

Every object and agent in NEEDSIM Life simulation has a NEEDSIMNode: This is the essential component for using NEEDSIM Life simulation.

5.26.2 Member Function Documentation

5.26.2.1 bool NEEDSIM.NEEDSIMNode.AcceptSlot ( )

If the simulation distributed a slot to this agent try to accept it.

Returns

Whether the slot was accepted

5.26.2.2 bool NEEDSIM.NEEDSIMNode.ArrivalAtSlot (Slot *slot*)

This method tries to call the AgentArrivalEasy() method at the slot it is passed to.

Parameters

slot	The slot this agent arrives to

Returns

Whether the arrival at a slot by this agent was successful

5.26.2.3 void NEEDSIM.NEEDSIMNode.BuildTreeBasedOnSceneHierarchy ( )

Recursively build an Affordance Tree from the scene hierarchy. This method will not work for intermediate objects in the hierarchy - there is only deeper search if a direct ancestor is a NEEDSIMNode.

5.26.2.4 bool NEEDSIM.NEEDSIMNode.TryConsumingAnimationOrder (Animator *animator*)

This method assumes that in the animator a trigger named "Movement" exists, and that for each interaction a trigger with the same name exists in the animator, e.g. that for the interaction "Eat" a trigger named "Eat" is in the animation and is used to transition into a state or sub-state-machine that plays animation(s) for eating.

Parameters

animator	

Returns

Whether the most recent animation order was consumed.

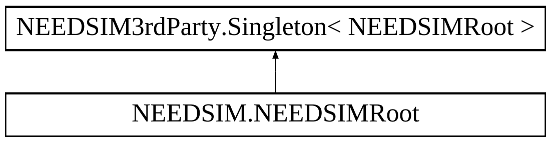
The documentation for this class was generated from the following file:

- NEEDSIMNode.cs

5.27 NEEDSIM.NEEDSIMRoot Class Reference

Every scene should have one root node for the AFFORDANCE TREE. This uses the settings of the NEEDSIM Manager and controls the simulation.

Inheritance diagram for NEEDSIM.NEEDSIMRoot:



Public Member Functions

- void BuildFlatAffordanceTreeFromScene ()

  Goes through the scene and adds all NEEDSIMNodes as children of the root. This is best if you don't have complexity issues to deal with. Otherwise building the affordance tree with more depth is adviced.
- void BuildAffordanceTreeFromNode (NEEDSIMNode node)

  To use this please pass the node you want to use as root node to this method. It is assumed that in the scene hierarchy this node is the root node to all other NEEDSIMNodes. So if you have a village with houses and objects in each house you should, in the scene view, make the village parent of the houses, and the houses object of the village, and each game object should have a NEEDSIMNode. You can leave the interactions set to none for the village and the houses if they are just used to calculate the abstraction for the Affordance Tree. The method BuildTreeBased↩ OnSceneHierarchy() that is called here will not work for intermediate objects in the hierarchy - there is only deeper search if a direct ancestor is a NEEDSIMNode.

Protected Member Functions

- NEEDSIMRoot ()

  guarantee this will be always a singleton only - can't use the constructor!

5.27.1 Detailed Description

Every scene should have one root node for the AFFORDANCE TREE. This uses the settings of the NEEDSIM Manager and controls the simulation.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 NEEDSIM.NEEDSIMRoot.NEEDSIMRoot ( ) [protected]

guarantee this will be always a singleton only - can't use the constructor!

5.27.3 Member Function Documentation

5.27.3.1 void NEEDSIM.NEEDSIMRoot.BuildAffordanceTreeFromNode (NEEDSIMNode *node*)

To use this please pass the node you want to use as root node to this method. It is assumed that in the scene hierarchy this node is the root node to all other NEEDSIMNodes. So if you have a village with houses and objects in each house you should, in the scene view, make the village parent of the houses, and the houses object of the village, and each game object should have a NEEDSIMNode. You can leave the interactions set to none for the village and the houses if they are just used to calculate the abstraction for the Affordance Tree. The method

BuildTreeBasedOnSceneHierarchy() that is called here will not work for intermediate objects in the hierarchy - there is only deeper search if a direct ancestor is a NEEDSIMNode.

Parameters

node	

5.27.3.2 void NEEDSIM.NEEDSIMRoot.BuildFlatAffordanceTreeFromScene ( )

Goes through the scene and adds all NEEDSIMNodes as children of the root. This is best if you don't have complexity issues to deal with. Otherwise building the affordance tree with more depth is adviced.

The documentation for this class was generated from the following file:

- NEEDSIMRoot.cs

5.28 NEEDSIMSampleSceneScripts.NeedsUI Class Reference

This class shows bars for need satisfaction. A full bar equals full satisfaction, an empty bar means the need is not satisfied. If the need is currently being satisfied an outline will be added.

Inherits MonoBehaviour.

5.28.1 Detailed Description

This class shows bars for need satisfaction. A full bar equals full satisfaction, an empty bar means the need is not satisfied. If the need is currently being satisfied an outline will be added.

The documentation for this class was generated from the following file:

- NeedsUI.cs

5.29 NEEDSIM.PlanDemo Class Reference

A simple behavior control solution. We tried to write this in a way that makes it easy to use our code samples in Finite State Machines, Behavior Trees and Goal-oriented Action Planning. The idea is that you can run our simulation from within a different solution, for example in case you want to have agents with fighting capabilites.

Public Member Functions

- void Update ()

  Update runs the plan.

5.29.1 Detailed Description

A simple behavior control solution. We tried to write this in a way that makes it easy to use our code samples in Finite State Machines, Behavior Trees and Goal-oriented Action Planning. The idea is that you can run our simulation from within a different solution, for example in case you want to have agents with fighting capabilites.

5.29.2 Member Function Documentation

5.29.2.1 void NEEDSIM.PlanDemo.Update ( )

Update runs the plan.

If the currently running action returns, upon evaluation, Result.Running, we keep on running that action. If Result.↩
Failure is returend we start a new sequence. If Result.Success is returned we go to the next step in the current sequence, or, if at the last step, start a new sequence.

The documentation for this class was generated from the following file:

- PlanDemo.cs

5.30 NEEDSIMSampleSceneScripts.SampleCameraControl Class Reference

A very simple scrolling camera for NEEDSIM Life simulation example scenes.

Inherits MonoBehaviour.

5.30.1 Detailed Description

A very simple scrolling camera for NEEDSIM Life simulation example scenes.

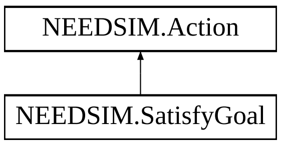
The documentation for this class was generated from the following file:

- SampleCameraControl.cs

5.31 NEEDSIM.SatisfyGoal Class Reference

Participate in a slot to satisfy a goal.

Inheritance diagram for NEEDSIM.SatisfyGoal:



Public Member Functions

- override Action.Result Run ()

  Satisfying a goal at an AffordanceTree node.

5.31.1 Detailed Description

Participate in a slot to satisfy a goal.

5.31.2 Member Function Documentation

5.31.2.1 override Action.Result NEEDSIM.SatisfyGoal.Run ( ) [virtual]

Satisfying a goal at an AffordanceTree node.

Returns

Success if need satsifaction goal was achieved, running whilst it is being satisfied.

Implements NEEDSIM.Action.

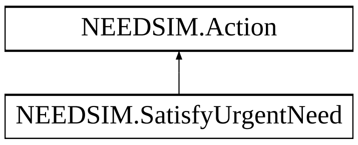
The documentation for this class was generated from the following file:

- SatisfyGoal.cs

5.32 NEEDSIM.SatisfyUrgentNeed Class Reference

Participating a slot. The respective behavior for value/urgency oriented behaviors.

Inheritance diagram for NEEDSIM.SatisfyUrgentNeed:



Public Member Functions

- override Action.Result Run ()

  Participating a slot. The respective behavior for value/urgency oriented behaviors.

5.32.1 Detailed Description

Participating a slot. The respective behavior for value/urgency oriented behaviors.

5.32.2 Member Function Documentation

5.32.2.1 override Action.Result NEEDSIM.SatisfyUrgentNeed.Run ( ) [virtual]

Participating a slot. The respective behavior for value/urgency oriented behaviors.

Returns

TODO

Implements NEEDSIM.Action.

The documentation for this class was generated from the following file:

- SatisfyUrgentNeed.cs

5.33 NEEDSIMSampleSceneScripts.SceneSwitcher Class Reference

public methods to switch scenes via a button click. You have to add the scenes to your build settings to use the prefab that uses this script.

Inherits MonoBehaviour.

5.33.1 Detailed Description

public methods to switch scenes via a button click. You have to add the scenes to your build settings to use the prefab that uses this script.

The documentation for this class was generated from the following file:

- SceneSwitcher.cs

5.34 NEEDSIMSampleSceneScripts.SimpleSpawn Class Reference

This is a spawning example script to maintain populations

Inherits MonoBehaviour.

Public Member Functions

- void fillPopulation ()

    Spawn an instance of the prefab for each spawning point at which the previously (if any) spawned instance is dead (null)
- void killAll ()

    Remove all agents from the slots they currently paticipate in and delete them.

5.34.1 Detailed Description

This is a spawning example script to maintain populations

5.34.2 Member Function Documentation

5.34.2.1 void NEEDSIMSampleSceneScripts.SimpleSpawn.fillPopulation ( )

Spawn an instance of the prefab for each spawning point at which the previously (if any) spawned instance is dead (null)

5.34.2.2 void NEEDSIMSampleSceneScripts.SimpleSpawn.killAll ( )

Remove all agents from the slots they currently paticipate in and delete them.

The documentation for this class was generated from the following file:

- SimpleSpawn.cs

5.35 Simulation.SimulationData Class Reference

The data loaded in the simulation at runtime

Public Member Functions

- SimulationData (Dictionary< string, float > minValueDictionary, Dictionary< string, float > maxValue↩ Dictionary, Dictionary< string, float > changePerSecondDictionary, Dictionary< string, float > criticalState↩ Dictionary, Dictionary< string, float > satisfiedStateDictionary, List< string > needNames, List< string >

speciesNames, List< Species > species, List< Interaction > interactionList, Dictionary< string, Species > speciesByName)

Constructs new simulation data.

Public Attributes

- readonly Dictionary< string, float > CriticalStateDictionary

  For each need, what is the value below or equal to which it is considered critical.
- readonly Dictionary< string, float > SatisfiedStateDictionary

  For each need, what is the value above which it is considered satisfied.
- readonly List< string > NeedNames

  The names of all the needs.
- readonly Dictionary< string, Interaction > InteractionByNameDictionary

  An instance of each interaction by its name

Properties

- Dictionary< string, float > ChangePerSecond `[get]`

  How much each need changes over time, for example at which rate a character becomes more hungry.
- AffordanceTreeNode Root `[get, set]`

  The root of the AFFORDANCE TREE. A lot of management of the simulation can be done from this point.
- Dictionary< string, float > WeightsForNeed `[get, set]`

  A general and global weight that affects the utility agent see in each need. Best to use values between 0.0 and 1.0.
- Dictionary< Needs.NeedSatisfactions, float > WeightsForNeedSatisfaction `[get, set]`

  How each of the Needs.NeedSatisfactions states is valued - use this for example to prioritize critical needs. Best to use values between 0.0 and 1.0

5.35.1 Detailed Description

The data loaded in the simulation at runtime

5.35.2 Constructor & Destructor Documentation

5.35.2.1 Simulation.SimulationData.SimulationData (Dictionary< string, float > *minValueDictionary,* Dictionary< string, float > *maxValueDictionary,* Dictionary< string, float > *changePerSecondDictionary,* Dictionary< string, float > *criticalStateDictionary,* Dictionary< string, float > *satisfiedStateDictionary,* List< string > *needNames,* List< string > *speciesNames,* List< Species > *species,* List< Interaction > *interactionList,* Dictionary< string, Species > *speciesByName*)

Constructs new simulation data.

Parameters

minValue↩ Dictionary	The lowest value a need can have at runtime.
maxValue↩ Dictionary	The highest value a need can have at runtime.
changePer↩ Second↩ Dictionary	For each need, how much it changes over time, for example a decay of the need 'Hunger' would mean characters get hungry over time

criticalState↩ Dictionary	The values below or equal to which a need is considered to be in critical state.
satisfiedState↩ Dictionary	The value above which a need is considered to be in satisfied state.
needNames	The names of all needs
speciesNames	The names of all species
species	All species
interactionList	All interactions
speciesByName	

5.35.3 Member Data Documentation

5.35.3.1 readonly Dictionary<string, float> Simulation.SimulationData.CriticalStateDictionary

For each need, what is the value below or equal to which it is considered critical.

5.35.3.2 readonly Dictionary<string, Interaction> Simulation.SimulationData.InteractionByNameDictionary

An instance of each interaction by its name

5.35.3.3 readonly List<string> Simulation.SimulationData.NeedNames

The names of all the needs.

5.35.3.4 readonly Dictionary<string, float> Simulation.SimulationData.SatisfiedStateDictionary

For each need, what is the value above which it is considered satisfied.

5.35.4 Property Documentation

5.35.4.1 Dictionary<string, float> Simulation.SimulationData.ChangePerSecond [get]

How much each need changes over time, for example at which rate a character becomes more hungry.

5.35.4.2 AffordanceTreeNode Simulation.SimulationData.Root [get], [set]

The root of the AFFORDANCE TREE. A lot of management of the simulation can be done from this point.

5.35.4.3 Dictionary<string, float> Simulation.SimulationData.WeightsForNeed [get], [set]

A general and global weight that affects the utility agent see in each need. Best to use values between 0.0 and 1.0.

5.35.4.4 Dictionary<Needs.NeedSatisfactions, float> Simulation.SimulationData.WeightsForNeedSatisfaction [get], [set]

How each of the Needs.NeedSatisfactions states is valued - use this for example to prioritize critical needs. Best to use values between 0.0 and 1.0

The documentation for this class was generated from the following file:

- SimulationData.cs

5.36 NEEDSIM3rdParty.Singleton< T > Class Template Reference

Be aware this will not prevent a non singleton constructor such as `T myT = new T();` To prevent that, add `protected T () {}` to your singleton class.

Inherits MonoBehaviour.

Public Member Functions

- void OnDestroy ()

  When Unity quits, it destroys objects in a random order. In principle, a Singleton is only destroyed when application quits. If any script calls Instance after it have been destroyed, it will create a buggy ghost object that will stay on the Editor scene even after stopping playing the Application. Really bad! So, this was made to be sure we're not creating that buggy ghost object.

5.36.1 Detailed Description

Be aware this will not prevent a non singleton constructor such as `T myT = new T();` To prevent that, add `protected T () {}` to your singleton class.

As a note, this is made as MonoBehaviour because we need Coroutines.

Type Constraints

T* : *MonoBehaviour

5.36.2 Member Function Documentation

5.36.2.1 void NEEDSIM3rdParty.Singleton< T >.OnDestroy ( )

When Unity quits, it destroys objects in a random order. In principle, a Singleton is only destroyed when application quits. If any script calls Instance after it have been destroyed, it will create a buggy ghost object that will stay on the Editor scene even after stopping playing the Application. Really bad! So, this was made to be sure we're not creating that buggy ghost object.

The documentation for this class was generated from the following file:

- Singleton.cs

5.37 Simulation.Slot Class Reference

A slot is a position in the world where an agent can run the interactions provided by the object that offers the slot.

Public Types

- enum SlotStates {
  SlotStates.Blocked, SlotStates.Reserved, SlotStates.ReadyCharacter, SlotStates.ReadyForAuction, SlotStates.CurrentlyOnAuction }

  A slot is in one of these states at any time.
- enum Result {
  Result.Success, Result.NoAffordance, Result.NoInteraction, Result.UnclearFailure, Result.InteractionAlreadyRunning, Result.NoProlongableInteraction }

  The result of an agent trying to use this slot

Public Member Functions

- Slot (Affordance affordance)

    Create a new slot
- bool InterruptInteraction ()

    Interrupt the interaction currently running at this slot
- float CurrentInteractionDuration ()

    Get the remaining duration of the currently running interaction.
- bool ReserveSlot ()

    Set the state of the slot to reserved.
- Result AgentArrivalEasy (AffordanceTreeNode participant)

    This tries to start a random interaction if no interaction is running and sets slot state to ReadyCharacter.
- bool AgentDeparture ()

    Frees the slot for being offered to other agents.
- bool OfferSlot ()

    Sets the slot state to currently being on auction
- bool SetSlotBlocked ()

    Set the slot state to being blocked.

Properties

- SlotStates SlotState `[get]`

    The current state of the slot.
- Vector3 Position `[get, set]`

    World space position of the slot.
- Vector3 LocalPosition `[get, set]`

    Please set the position relative to the slot's parent transform here
- Vector3 LookAt `[get, set]`

    In world coordinates, where the agent should orient him/herself to
- Vector3 LocalLookAt `[get, set]`

    The local position of he look at, where the agent should orient him/herself to, relative to the parents transform.
- bool IsAuctionable `[get, set]`

    Whether the auction system is allowed to offer this slot to agents.
- Interaction currentInteraction `[get]`

    Which interaction is currently running at this slot.

5.37.1 Detailed Description

A slot is a position in the world where an agent can run the interactions provided by the object that offers the slot.

5.37.2 Member Enumeration Documentation

5.37.2.1 enum Simulation.Slot.Result `[strong]`

The result of an agent trying to use this slot

Enumerator

Success	No problem was determined as is slot used.
NoAffordance	There as a problem with initialization and the affordance this slot belongs to is null.
NoInteraction	There is no interaction available at the affordace this slot belongs to.

UnclearFailure There is some issue that lacks proper description

InteractionAlreadyRunning There is already an interaction running. This is not necessarily a problem, and can in many circumstances be considered a success.

NoProlongableInteraction When trying to prolong an interaction there was no such option.

5.37.2.2 enum Simulation.Slot.SlotStates `[strong]`

A slot is in one of these states at any time.

Enumerator

Blocked A slot that is blocked for whatever reason can not be used by agents.

Reserved An agent reserved this slot for her/himself, and it is assumed that the agent is on the way to slot.

ReadyCharacter An agent is ready to interact with the slot, or interacting with the slot.

ReadyForAuction This slot can be auctioned, but is not currently auctioned

CurrentlyOnAuction Currently this slot is available for auctions.

5.37.3 Constructor & Destructor Documentation

5.37.3.1 Simulation.Slot.Slot (Affordance *affordance*)

Create a new slot

Parameters

affordance	The affordance this slot will belong to.

5.37.4 Member Function Documentation

5.37.4.1 Result Simulation.Slot.AgentArrivalEasy (AffordanceTreeNode *participant*)

This tries to start a random interaction if no interaction is running and sets slot state to ReadyCharacter.

Parameters

participant	The agent that will be participating in the interactions of this slot.

Returns

The result of trying to start an interaction

5.37.4.2 bool Simulation.Slot.AgentDeparture ( )

Frees the slot for being offered to other agents.

Returns

5.37.4.3 float Simulation.Slot.CurrentInteractionDuration ( )

Get the remaining duration of the currently running interaction.

Returns

5.37.4.4 bool Simulation.Slot.InterruptInteraction ( )

Interrupt the interaction currently running at this slot

Returns

5.37.4.5 bool Simulation.Slot.OfferSlot ( )

Sets the slot state to currently being on auction

Returns

5.37.4.6 bool Simulation.Slot.ReserveSlot ( )

Set the state of the slot to reserved.

Returns

Will fail if the slot is blocked.

5.37.4.7 bool Simulation.Slot.SetSlotBlocked ( )

Set the slot state to being blocked.

Returns

5.37.5 Property Documentation

5.37.5.1 Interaction Simulation.Slot.currentInteraction `[get]`

Which interaction is currently running at this slot.

5.37.5.2 bool Simulation.Slot.IsAuctionable `[get]`, `[set]`

Whether the auction system is allowed to offer this slot to agents.

5.37.5.3 Vector3 Simulation.Slot.LocalLookAt `[get]`, `[set]`

The local position of he look at, where the agent should orient him/herself to, relative to the parents transform.

5.37.5.4 Vector3 Simulation.Slot.LocalPosition `[get], [set]`

Please set the position relative to the slot's parent transform here

5.37.5.5 Vector3 Simulation.Slot.LookAt `[get], [set]`

In world coordinates, where the agent should orient him/herself to

5.37.5.6 Vector3 Simulation.Slot.Position `[get], [set]`

World space position of the slot.

5.37.5.7 SlotStates Simulation.Slot.SlotState `[get]`

The current state of the slot.

The documentation for this class was generated from the following file:

- Slot.cs

5.38 NEEDSIMSampleSceneScripts.SpawnBedsManager Class Reference

This example shows how you could spawn all the objects and agents procedurally.

Inherits MonoBehaviour.

Public Member Functions

- void SpawnBed ()

  Create an instance of a bed, add it to the simulation, and translate it to the right position.
- void DestroyBed ()

  Destroy an instance of a bed.

5.38.1 Detailed Description

This example shows how you could spawn all the objects and agents procedurally.

5.38.2 Member Function Documentation

5.38.2.1 void NEEDSIMSampleSceneScripts.SpawnBedsManager.DestroyBed ( )

Destroy an instance of a bed.

5.38.2.2 void NEEDSIMSampleSceneScripts.SpawnBedsManager.SpawnBed ( )

Create an instance of a bed, add it to the simulation, and translate it to the right position.

The documentation for this class was generated from the following file:

- SpawnBedsManager.cs

5.39 NEEDSIMSampleSceneScripts.SpawnUIRuntimeEditing Class Reference

Spawns a UI Element for each need and each satisfaction rate of an interaction.

Inherits MonoBehaviour.

5.39.1 Detailed Description

Spawns a UI Element for each need and each satisfaction rate of an interaction.

The documentation for this class was generated from the following file:

- SpawnUIRuntimeEditing.cs

5.40 Simulation.Species Class Reference

A species is a set of needs. For example zombies might only have the 'Hunger' need, whereas humans furthermore have a 'Social' need.

Public Member Functions

- Species (string name, List< string > needs)

  Construct a new species

Public Attributes

- string speciesName

  The unique name of the need. Identifier.
- List< string > needs

  The set of needs that defines this species.

5.40.1 Detailed Description

A species is a set of needs. For example zombies might only have the 'Hunger' need, whereas humans furthermore have a 'Social' need.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Simulation.Species.Species (string *name,* List< string > *needs*)

Construct a new species

Parameters

name	The unique name of the need. Identifier.
needs	The set of needs that defines this species

5.40.3 Member Data Documentation

5.40.3.1 List<string> Simulation.Species.needs

The set of needs that defines this species.

5.40.3.2 string Simulation.Species.speciesName

The unique name of the need. Identifier.

The documentation for this class was generated from the following file:

- Species.cs

5.41 Simulation.StringFloatPair Class Reference

A class that helps creating key value pairs.

Public Member Functions

- StringFloatPair (string key, float value)

  Create a new helper to later on create key value pairs

Public Attributes

- string stringValue

  The key/name/identifier
- float floatValue

  The value

5.41.1 Detailed Description

A class that helps creating key value pairs.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 Simulation.StringFloatPair.StringFloatPair (string *key,* float *value*)

Create a new helper to later on create key value pairs

Parameters

key	This will be the key
value	This will be the value

5.41.3 Member Data Documentation

5.41.3.1 float Simulation.StringFloatPair.floatValue

The value

5.41.3.2 string Simulation.StringFloatPair.stringValue

The key/name/identifier

The documentation for this class was generated from the following file:

- StringFloatPair.cs

5.42 Simulation.Strings Class Reference

A centralized place for many of the strings used by the NEEDSIM Life simulation.

Static Public Member Functions

- static string NewHasBeenSet (string name)
- static string DefaultInteractionRelativeNeedName (string needName)
- static string InteractionLabelToExtend (int extension)
- static string SlotNumberLabel (int i)

Public Attributes

- const string EditorExtensionPosition = "Window/" + ProductName + " - " + ProductDescription
- const string ProductName = "NEEDSIM"
- const string ProductDescription = "Life Simulation"

Static Public Attributes

- static GUIContent LabelNeedName
- static GUIContent LabelNeedChangeRate
- static GUIContent LabelCriticalState
- static GUIContent LabelSatisfiedState
- static GUIContent InteractionDurationLabel
- static GUIContent NeedAffectedLabel
- static GUIContent NumberOfNeedsAffectedLabel
- static GUIContent SatisfactionRateLabel

Properties

- static string SimulationManagerConstructed `[get]`
- static string FolderName `[get]`
- static string EditorWindowTitle `[get]`
- static string NoDefaultDataFound `[get]`
- static string MoreThanOneDefaultData `[get]`
- static string Advice `[get]`
- static string NoDatabaseFound `[get]`
- static string BuildAffordanceTreeFromSceneLabel `[get]`
- static string DialogTitleExitPlay `[get]`
- static string DialogMessageExitPlay `[get]`
- static string DialogButtonExitPlay `[get]`
- static string Welcome `[get]`
- static string SupportMail `[get]`
- static string WebsiteURL `[get]`
- static string WebsiteURLButtonText `[get]`
- static string VisitUsAt `[get]`
- static string Contact `[get]`
- static string AdvancedInstructionLine1 `[get]`
- static string AdvancedInstructionLine2 `[get]`
- static string AdvancedInstructionLine3 `[get]`
- static string AdvancedInstructionLine4 `[get]`
- static string AdvancedInstructionLine5 `[get]`

- static string MarkCurrentDatabaseAsDefault `[get]`
- static string UnableToLoadDatabase `[get]`
- static string GenericDatabaseName `[get]`
- static string defaultString `[get]`
- static string currentDatabaseLoadedLabel `[get]`
- static string DefaultDatabaseName `[get]`
- static string AssetsPath `[get]`
- static string NeedsDataBasePath `[get]`
- static string DataFolder `[get]`
- static string UserDataResources `[get]`
- static string Asterisk `[get]`
- static string AssetAppendix `[get]`
- static string SceneAppendix `[get]`
- static string DefaultNeedName `[get]`
- static string DefaultSpecies `[get]`
- static string SpeciesNameLabel `[get]`
- static string NeedsInSpeciesLabel `[get]`
- static string AddNeedButtonLabel `[get]`
- static string AddSpeciesButtonLabel `[get]`
- static string AddInteractionButtonLabel `[get]`
- static string RemoveNeedButtonLabel `[get]`
- static string DefaultDatabaseHeadline `[get]`
- static string SpeciesViewHeadline `[get]`
- static string AdvancedDatabaseViewHeadline `[get]`
- static string DataBasesAvailableLabel `[get]`
- static string ButtonAddDatabaseLabel `[get]`
- static string ButtonRemoveDatabaseLabel `[get]`
- static string[ ] WindowSelectionNames `[get]`
- static string InteractionViewHeadline `[get]`
- static string None `[get]`
- static string DefaultInteractionName `[get]`
- static string AssignNeedLabel `[get]`
- static string InteractionNameLabel `[get]`
- static string PreconditionsHeadline `[get]`
- static string PreconditionsSatisfactionLevelLabel `[get]`
- static string[ ] PreconditionsStatisfactionLevelArray `[get]`
- static string PreconditionsSpecies `[get]`
- static string PreconditionDefaultAny `[get]`
- static string PreconditionsUnsatisfiedLabel `[get]`
- static string IsAgent `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string DrawGizmosInGame `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string O_Space `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string TerritoryCenter `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string SpeciesName `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string ShowDebugInGame `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string ShowDebugInInspector `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

- static string InteractionData `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string SlotPositionsArray `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string AuctionableBoolArray `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string InteractionDataArrayAccess `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string SlotPositionsArraySize `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string SlotPositionsArrayAccess `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string IsAuctionableArraySize `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string IsAuctionableArrayAccess `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string IsAgentLabel `[get]`
- static string ModifyLookAt `[get]`

  This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs
- static string RandomStartLevelsLabel `[get]`
- static string ShowDebugGizmosLabel `[get]`
- static string TerritoryControlHeadline `[get]`
- static string NumberOfSlotsLabel `[get]`
- static string SlotPositionLabel `[get]`
- static string IsAuctionableLabel `[get]`
- static string AssignSpeciesOptionToAgentLabel `[get]`
- static string DebugIngameLabel `[get]`
- static string DebugInSpectorLabel `[get]`
- static string DeleteDatabaseTitle `[get]`
- static string DeleteDatabaseMessage `[get]`
- static string DialogYes `[get]`
- static string DialogCancel `[get]`
- static string YourDatabaseName `[get]`
- static string GeneralSettings `[get]`
- static string AdvancedSettings `[get]`
- static string LogSimulationLabel `[get]`
- static string PrintSimDebugLogLabel `[get]`
- static string AttachSpecificDBLabel `[get]`

5.42.1 Detailed Description

A centralized place for many of the strings used by the NEEDSIM Life simulation.

5.42.2 Member Function Documentation

5.42.2.1 static string Simulation.Strings.DefaultInteractionRelativeNeedName (string *needName*) `[static]`

Parameters

needName	

Returns

5.42.2.2 static string Simulation.Strings.InteractionLabelToExtend (int *extension*) [static]

Parameters

extension	

Returns

5.42.2.3 static string Simulation.Strings.NewHasBeenSet (string *name*) [static]

Parameters

name	

Returns

5.42.2.4 static string Simulation.Strings.SlotNumberLabel (int *i*) [static]

Parameters

i	

Returns

5.42.3 Member Data Documentation

5.42.3.1 const string Simulation.Strings.EditorExtensionPosition = "Window/" + ProductName + " - " + ProductDescription

5.42.3.2 GUIContent Simulation.Strings.InteractionDurationLabel [static]

Initial value:

```
= new GUIContent(
                "Duration",
                "The duration of the interaction in seconds. Often a good idea is to use the duration of
       the respective animations.")
```

5.42.3.3 GUIContent Simulation.Strings.LabelCriticalState [static]

Initial value:

```
= new GUIContent(
            "Critical below:",
            "A need does not only have a numeric value, but we define some numeric ranges to be more
       abstract states of how satisfied a need is - Critical, Uncritical, Unsatisfied, Satisfied and Maximized. This can
       be used in behavior authored on top of the core NEEDSIM Life simulation."
            )
```

5.42.3.4 GUIContent Simulation.Strings.LabelNeedChangeRate [static]

Initial value:

```
= new GUIContent(
            "+/- per second",
            "How satisfied a need is changes over time - for example we used a negative rate to model how
       characters get hungry in a simulation where we assume a value of 100 for 'Hunger' means the agent has a full
       belly, and zero means he is starving."
            )
```

5.42.3.5 GUIContent Simulation.Strings.LabelNeedName [static]

Initial value:

```
= new GUIContent(
            "Need name",
            "Please chose a unique name for this need. Creating a model of needs and interactions to satisy
       them is the core idea of NEEDSIM Life simulation.")
```

5.42.3.6 GUIContent Simulation.Strings.LabelSatisfiedState [static]

Initial value:

```
= new GUIContent(
            "Satisfied above:",
            "A need does not only have a numeric value, but we define some numeric ranges to be more
       abstract states of how satisfied a need is - Critical, Uncritical, Unsatisfied, Satisfied and Maximized. This can
       be used in behavior authored on top of the core NEEDSIM Life simulation."
            )
```

5.42.3.7 GUIContent Simulation.Strings.NeedAffectedLabel [static]

Initial value:

```
= new GUIContent(
            "Need affected:",
            "The interaction changes the satisfaction level of the need selected from the drop down menu."
            )
```

5.42.3.8 GUIContent Simulation.Strings.NumberOfNeedsAffectedLabel [static]

Initial value:

```
= new GUIContent(
            "No. of Needs affected:",
            "One ore more needs can be affected by the same interaction. For example the interaction
       sunbathing might reduce the need to relax, but could increase thirst."
            )
```

5.42.3.9 const string Simulation.Strings.ProductDescription = "Life Simulation"

5.42.3.10 const string Simulation.Strings.ProductName = "NEEDSIM"

5.42.3.11 GUIContent Simulation.Strings.SatisfactionRateLabel [static]

Initial value:

```
= new GUIContent(
            "Satisfaction +/-",
            "The rate at which the specified need is decayed or satisfied per second. Standard ranges for
       needs are from 0 to 100, so values should be in between -100 to 100."
            )
```

5.42.4 Property Documentation

5.42.4.1 string Simulation.Strings.AddInteractionButtonLabel [static], [get]

5.42.4.2 string Simulation.Strings.AddNeedButtonLabel [static], [get]

5.42.4.3 string Simulation.Strings.AddSpeciesButtonLabel [static], [get]

5.42.4.4 string Simulation.Strings.AdvancedDatabaseViewHeadline [static], [get]

5.42.4.5 string Simulation.Strings.AdvancedInstructionLine1 [static], [get]

5.42.4.6 string Simulation.Strings.AdvancedInstructionLine2 [static], [get]

5.42.4.7 string Simulation.Strings.AdvancedInstructionLine3 [static], [get]

5.42.4.8 string Simulation.Strings.AdvancedInstructionLine4 [static], [get]

5.42.4.9 string Simulation.Strings.AdvancedInstructionLine5 [static], [get]

5.42.4.10 string Simulation.Strings.AdvancedSettings [static], [get]

5.42.4.11 string Simulation.Strings.Advice [static], [get]

5.42.4.12 string Simulation.Strings.AssetAppendix [static], [get]

5.42.4.13 string Simulation.Strings.AssetsPath [static], [get]

5.42.4.14 string Simulation.Strings.AssignNeedLabel [static], [get]

5.42.4.15 string Simulation.Strings.AssignSpeciesOptionToAgentLabel [static], [get]

5.42.4.16 string Simulation.Strings.Asterisk [static], [get]

5.42.4.17 string Simulation.Strings.AttachSpecificDBLabel [static], [get]

5.42.4.18 string Simulation.Strings.AuctionableBoolArray [static], [get]

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.19 string Simulation.Strings.BuildAffordanceTreeFromSceneLabel [static], [get]

5.42.4.20 string Simulation.Strings.ButtonAddDatabaseLabel `[static], [get]`

5.42.4.21 string Simulation.Strings.ButtonRemoveDatabaseLabel `[static], [get]`

5.42.4.22 string Simulation.Strings.Contact `[static], [get]`

5.42.4.23 string Simulation.Strings.currentDatabaseLoadedLabel `[static], [get]`

5.42.4.24 string Simulation.Strings.DataBasesAvailableLabel `[static], [get]`

5.42.4.25 string Simulation.Strings.DataFolder `[static], [get]`

5.42.4.26 string Simulation.Strings.DebugIngameLabel `[static], [get]`

5.42.4.27 string Simulation.Strings.DebugInSpectorLabel `[static], [get]`

5.42.4.28 string Simulation.Strings.DefaultDatabaseHeadline `[static], [get]`

5.42.4.29 string Simulation.Strings.DefaultDatabaseName `[static], [get]`

5.42.4.30 string Simulation.Strings.DefaultInteractionName `[static], [get]`

5.42.4.31 string Simulation.Strings.DefaultNeedName `[static], [get]`

5.42.4.32 string Simulation.Strings.DefaultSpecies `[static], [get]`

5.42.4.33 string Simulation.Strings.defaultString `[static], [get]`

5.42.4.34 string Simulation.Strings.DeleteDatabaseMessage `[static], [get]`

5.42.4.35 string Simulation.Strings.DeleteDatabaseTitle `[static], [get]`

5.42.4.36 string Simulation.Strings.DialogButtonExitPlay `[static], [get]`

5.42.4.37 string Simulation.Strings.DialogCancel `[static], [get]`

5.42.4.38 string Simulation.Strings.DialogMessageExitPlay `[static], [get]`

5.42.4.39 string Simulation.Strings.DialogTitleExitPlay `[static], [get]`

5.42.4.40 string Simulation.Strings.DialogYes `[static], [get]`

5.42.4.41 string Simulation.Strings.DrawGizmosInGame `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.42 string Simulation.Strings.EditorWindowTitle `[static], [get]`

5.42.4.43 string Simulation.Strings.FolderName `[static], [get]`

5.42.4.44 string Simulation.Strings.GeneralSettings `[static], [get]`

5.42.4.45 string Simulation.Strings.GenericDatabaseName `[static], [get]`

5.42.4.46 string Simulation.Strings.InteractionData `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.47 string Simulation.Strings.InteractionDataArrayAccess `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.48 string Simulation.Strings.InteractionNameLabel `[static]`, `[get]`

5.42.4.49 string Simulation.Strings.InteractionViewHeadline `[static]`, `[get]`

5.42.4.50 string Simulation.Strings.IsAgent `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.51 string Simulation.Strings.IsAgentLabel `[static]`, `[get]`

5.42.4.52 string Simulation.Strings.IsAuctionableArrayAccess `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.53 string Simulation.Strings.IsAuctionableArraySize `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.54 string Simulation.Strings.IsAuctionableLabel `[static]`, `[get]`

5.42.4.55 string Simulation.Strings.LogSimulationLabel `[static]`, `[get]`

5.42.4.56 string Simulation.Strings.MarkCurrentDatabaseAsDefault `[static]`, `[get]`

5.42.4.57 string Simulation.Strings.ModifyLookAt `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.58 string Simulation.Strings.MoreThanOneDefaultData `[static]`, `[get]`

5.42.4.59 string Simulation.Strings.NeedsDataBasePath `[static]`, `[get]`

5.42.4.60 string Simulation.Strings.NeedsInSpeciesLabel `[static]`, `[get]`

5.42.4.61 string Simulation.Strings.NoDatabaseFound `[static]`, `[get]`

5.42.4.62 string Simulation.Strings.NoDefaultDataFound `[static]`, `[get]`

5.42.4.63 string Simulation.Strings.None `[static]`, `[get]`

5.42.4.64 string Simulation.Strings.NumberOfSlotsLabel `[static]`, `[get]`

5.42.4.65 string Simulation.Strings.O_Space `[static]`, `[get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.66 string Simulation.Strings.PreconditionDefaultAny `[static], [get]`

5.42.4.67 string Simulation.Strings.PreconditionsHeadline `[static], [get]`

5.42.4.68 string Simulation.Strings.PreconditionsSatisfactionLevelLabel `[static], [get]`

5.42.4.69 string Simulation.Strings.PreconditionsSpecies `[static], [get]`

5.42.4.70 string [ ] Simulation.Strings.PreconditionsStatisfactionLevelArray `[static], [get]`

5.42.4.71 string Simulation.Strings.PreconditionsUnsatisfiedLabel `[static], [get]`

5.42.4.72 string Simulation.Strings.PrintSimDebugLogLabel `[static], [get]`

5.42.4.73 string Simulation.Strings.RandomStartLevelsLabel `[static], [get]`

5.42.4.74 string Simulation.Strings.RemoveNeedButtonLabel `[static], [get]`

5.42.4.75 string Simulation.Strings.SceneAppendix `[static], [get]`

5.42.4.76 string Simulation.Strings.ShowDebugGizmosLabel `[static], [get]`

5.42.4.77 string Simulation.Strings.ShowDebugInGame `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.78 string Simulation.Strings.ShowDebugInInspector `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.79 string Simulation.Strings.SimulationManagerConstructed `[static], [get]`

5.42.4.80 string Simulation.Strings.SlotPositionLabel `[static], [get]`

5.42.4.81 string Simulation.Strings.SlotPositionsArray `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.82 string Simulation.Strings.SlotPositionsArrayAccess `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.83 string Simulation.Strings.SlotPositionsArraySize `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.84 string Simulation.Strings.SpeciesName `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.85 string Simulation.Strings.SpeciesNameLabel `[static], [get]`

5.42.4.86 string Simulation.Strings.SpeciesViewHeadline `[static], [get]`

5.42.4.87 string Simulation.Strings.SupportMail `[static], [get]`

5.42.4.88 string Simulation.Strings.TerritoryCenter `[static], [get]`

This has to be changed if changes to the respective field name are made in NEEDSIMNode.cs

5.42.4.89 string Simulation.Strings.TerritoryControlHeadline `[static], [get]`

5.42.4.90 string Simulation.Strings.UnableToLoadDatabase `[static], [get]`

5.42.4.91 string Simulation.Strings.UserDataResources `[static], [get]`

5.42.4.92 string Simulation.Strings.VisitUsAt `[static], [get]`

5.42.4.93 string Simulation.Strings.WebsiteURL `[static], [get]`

5.42.4.94 string Simulation.Strings.WebsiteURLButtonText `[static], [get]`

5.42.4.95 string Simulation.Strings.Welcome `[static], [get]`

5.42.4.96 string [ ] Simulation.Strings.WindowSelectionNames `[static], [get]`

5.42.4.97 string Simulation.Strings.YourDatabaseName `[static], [get]`

The documentation for this class was generated from the following file:

- Strings.cs

5.43 NEEDSIMSampleSceneScripts.TimeSystem Class Reference

This example uses arrays with 24 values each to modify how behaviors are evaluated at a specific time of day. This class works with Value Oriented behaviors, not with Goal Oriented behaviors, because it relies on the fact that all opportunities to satisfy needs are evaluated, not only the opportunities that can satisfy the need of the current goal.

Inherits MonoBehaviour.

5.43.1 Detailed Description

This example uses arrays with 24 values each to modify how behaviors are evaluated at a specific time of day. This class works with Value Oriented behaviors, not with Goal Oriented behaviors, because it relies on the fact that all opportunities to satisfy needs are evaluated, not only the opportunities that can satisfy the need of the current goal.

The documentation for this class was generated from the following file:

- TimeSystem.cs

Index