ActressMAS

Florin Leon

Version 2.1
3 January 2020

Table of Contents

Namespace Index

Packages

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

Hierarchical Index

Class Hierarchy

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

Class Index

Class List

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

Namespace Documentation

ActressMas Namespace Reference

Classes

- class **Agent**
  *An abstract base class for agents. You must define your own agent classes derived from **ConcurrentAgent** or **TurnBasedAgent**.*
- class **AgentState**
  *The class that stores the serializable state of the agent when it moves. It is the Memento in the Memento design pattern, while the specific **Agent** class whose state is saved and restored is the Originator. This class should be inherited to add all the serializable fields specific to a particular agent. For example, a concurrent agent cannot be serialized directly because MailboxProcessor is not serializable*
- class **ConcurrentAgent**
  The base class for an agent that runs concurrently in its environment. You must create your own agent classes derived from this abstract class.
- class **ConcurrentEnvironment**
  A concurrent environment, where the agents run in parallel.
- class **Container**
  A container contains an environment and is connected to a server. It facilitates the move of agents in a distributed system.
- class **Environment**
  *An abstract base class for environments. You must use **ConcurrentEnvironment** or **TurnBasedEnvironment**.*
- class **Info**
  *Information about **ActressMas** version*
- class **Message**
  A message that the agents use to communicate. In an agent-based system, the communication between the agents is exclusively performed by exchanging messages.
- class **NewTextEventArgs**
  The class that defines a message from a server or a container.
- class **RunnableMas**
  An abstract class which should be derived in order to specify the multiagent system with mobile agents that will be run in the environment of a container.
- class **Server**
  A server that ensures the communication of containers, e.g. for the movement of agents, in a distributed system.
- class **TurnBasedAgent**
  The base class for an agent that runs on a turn-based manner in its environment. You must create your own agent classes derived from this abstract class.
- class **TurnBasedEnvironment**
  A turn-based environment, where all the agents are executed sequentially or in a random order during a turn.

Class Documentation

ActressMas.Agent Class Reference

An abstract base class for agents. You must define your own agent classes derived from **ConcurrentAgent** or **TurnBasedAgent**.
Inherited by **ActressMas.ConcurrentAgent**, and **ActressMas.TurnBasedAgent**.

Detailed Description

An abstract base class for agents. You must define your own agent classes derived from **ConcurrentAgent** or **TurnBasedAgent**.

ActressMas.AgentState Class Reference

The class that stores the serializable state of the agent when it moves. It is the Memento in the Memento design pattern, while the specific **Agent** class whose state is saved and restored is the Originator. This class should be inherited to add all the serializable fields specific to a particular agent. For example, a concurrent agent cannot be serialized directly because MailboxProcessor is not serializable

Public Attributes

- Type **AgentType**
  The agent class needed in order to instantiate the agent object after a move

- string **Name**
  The agent name

Detailed Description

The class that stores the serializable state of the agent when it moves. It is the Memento in the Memento design pattern, while the specific **Agent** class whose state is saved and restored is the Originator. This class should be inherited to add all the serializable fields specific to a particular agent. For example, a concurrent agent cannot be serialized directly because MailboxProcessor is not serializable

---

Member Data Documentation

Type ActressMas.AgentState.AgentType

The agent class needed in order to instantiate the agent object after a move

string ActressMas.AgentState.Name

The agent name

ActressMas.ConcurrentAgent Class Reference

The base class for an agent that runs concurrently in its environment. You must create your own agent classes derived from this abstract class.
Inherits **ActressMas.Agent**.

Public Member Functions

- virtual void **Act** (**Message** message)
  This is the method that is called when the agent receives a message and is activated. This is where the main logic of the agent should be placed.

- void **Broadcast** (string content, bool includeSender=false, string conversationId="")
  Sends a message to all the agents in the environment.

- bool **CanMove** (string destination)
  Tests whether the agent can move to a certain remote container.

- virtual void **LoadState** (**AgentState** state)
- void **Move** (string destination)
  The method that should be called when the agent wants to move to a different container.

- virtual **AgentState SaveState** ()
  Exports the state of the agent, so it can be serialized when moving to another container.

- void **Send** (string receiver, string content, string conversationId="")
  Sends a message to a specific agent, identified by name.

- void **SendToMany** (List< string > receivers, string content, string conversationId="")
  Sends a message to a specific set of agents, identified by name.

- virtual void **Setup** ()
  This method is called right after Start, before any messages have been received. It is similar to the constructor of the class, but it should be used for agent-related logic, e.g. for sending initial message(s).

- void **Start** ()
  Starts the agent execution, after it has been created. In a concurrent environment, the agent that sends the first message(s) and thus initiates the execution of the whole protocol should be started last, after all the agents have been added to the environment. First, the Setup method is called, and then the Act method is automatically called when the agent receives a message.

- void **Stop** ()
  Stops the execution of the agent and removes it from the environment. Use the Stop method instead of Environment.Remove when the decision to be stopped belongs to the agent itself.

Properties

- string **Name** `[get, set]`

The name of the agent. Each agent must have a unique name in its environment. Most operations are performed using agent names rather than agent objects.

- **ConcurrentEnvironment Environment** `[get, set]`
  The environment in which the agent runs. A concurrent agent can only run in a concurrent environment.

Detailed Description

The base class for an agent that runs concurrently in its environment. You must create your own agent classes derived from this abstract class.

Member Function Documentation

virtual void ActressMas.ConcurrentAgent.Act (Message *message*) `[virtual]`

This is the method that is called when the agent receives a message and is activated. This is where the main logic of the agent should be placed.

Parameters:

message	The message that the agent has received and should respond to

void ActressMas.ConcurrentAgent.Broadcast (string *content*, bool *includeSender* = `false`, string *conversationId* = `""`)

Sends a message to all the agents in the environment.

Parameters:

content	The content of the message
includeSender	Whether the sender itself receives the message or not
conversationId	A conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

bool ActressMas.ConcurrentAgent.CanMove (string *destination*)

Tests whether the agent can move to a certain remote container.

Parameters:

destination	The name of the container that the agent wants to move to

Returns:

virtual void ActressMas.ConcurrentAgent.LoadState (AgentState *state*) `[virtual]`

Imports the state of the agent, after it has moved from another container.

Parameters:

state	The state of the agent

void ActressMas.ConcurrentAgent.Move (string *destination*)

The method that should be called when the agent wants to move to a different container.

Parameters:

destination	The name of the container that the agent wants to move to

virtual AgentState ActressMas.ConcurrentAgent.SaveState () `[virtual]`

Exports the state of the agent, so it can be serialized when moving to another container.

Returns:

void ActressMas.ConcurrentAgent.Send (string *receiver*, string *content*, string *conversationId* = "")

Sends a message to a specific agent, identified by name.

Parameters:

receiver	The agent that will receive the message
content	The content of the message
conversationId	A conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

void ActressMas.ConcurrentAgent.SendToMany (List< string > *receivers*, string *content*, string *conversationId* = "")

Sends a message to a specific set of agents, identified by name.

Parameters:

receivers	The list of agents that will receive the message
content	The content of the message
conversationId	A conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

virtual void ActressMas.ConcurrentAgent.Setup () `[virtual]`

This method is called right after Start, before any messages have been received. It is similar to the constructor of the class, but it should be used for agent-related logic, e.g. for sending initial message(s).

void ActressMas.ConcurrentAgent.Start ()

Starts the agent execution, after it has been created. In a concurrent environment, the agent that sends the first message(s) and thus initiates the execution of the whole protocol should be started last, after all the agents have been added to the environment. First, the Setup method is called, and then the Act method is automatically called when the agent receives a message.

void ActressMas.ConcurrentAgent.Stop ()

Stops the execution of the agent and removes it from the environment. Use the Stop method instead of Environment.Remove when the decision to be stopped belongs to the agent itself.

Property Documentation

string ActressMas.ConcurrentAgent.Name `[get], [set]`

The name of the agent. Each agent must have a unique name in its environment. Most operations are performed using agent names rather than agent objects.

ConcurrentEnvironment ActressMas.ConcurrentAgent.Environment `[get], [set]`

The environment in which the agent runs. A concurrent agent can only run in a concurrent environment.

ActressMas.ConcurrentEnvironment Class Reference

A concurrent environment, where the agents run in parallel.
Inherits **ActressMas.Environment**.

Public Member Functions

- **ConcurrentEnvironment** ()
  *Initializes a new instance of the **ConcurrentEnvironment** class.*

- void **Add** (**ConcurrentAgent** agent)
  Adds an agent to the environment. The agent should already have a name and its name should be unique.

- void **Add** (**ConcurrentAgent** agent, string name)
  Adds an agent to the environment. Its name should be unique.

- List< string > **AllAgents** ()
  Returns a list with the names of all the agents.

- List< string > **AllContainers** ()
  Returns a list with the names of all the containers in the distributed system. This list may change over time, as some new containers may get connected and existing ones may disconnect.

- List< string > **FilteredAgents** (string nameFragment)
  Returns a list with the names of all the agents that contain a certain string.

- string **RandomAgent** ()
  Returns the name of a randomly selected agent from the environment

- string **RandomAgent** (Random rand)
  Returns the name of a randomly selected agent from the environment using a predefined random number generator. This is useful for experiments involving non-determinism, but which should be repeatable for analysis and debugging.

- void **Remove** (**ConcurrentAgent** agent)
  Stops the execution of the agent and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

- void **Remove** (string agentName)
  Stops the execution of the agent identified by name and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

- void **Send** (**Message** message)
  Sends a message from the outside of the multiagent system. Whenever possible, the agents should use the Send method of their own class, not the Send method of the environment. This method can also be used to simulate a forwarding behavior.

- void **WaitAll** ()
  Prevents the program from closing by waiting as long as some agents still run in the environment. This method should be used at the end of the main program, after all the agents have been added to the environment and started.

Properties

- int **NoAgents** `[get]`
  The number of agents in the environment

- string **ContainerName** `[get]`
  The name of the container that contains the environment. If the container is not set or not connected to the server, this method will return the empty string.

- Dictionary< string, dynamic > **Memory** `[get, set]`
  An object that can be used as a shared memory by the agents.

Detailed Description

A concurrent environment, where the agents run in parallel.

Constructor & Destructor Documentation

ActressMas.ConcurrentEnvironment.ConcurrentEnvironment ()

Initializes a new instance of the **ConcurrentEnvironment** class.

Member Function Documentation

void ActressMas.ConcurrentEnvironment.Add (ConcurrentAgent *agent*)

Adds an agent to the environment. The agent should already have a name and its name should be unique.

Parameters:

agent	The concurrent agent that will be added

void ActressMas.ConcurrentEnvironment.Add (ConcurrentAgent *agent*, string *name*)

Adds an agent to the environment. Its name should be unique.

Parameters:

agent	The concurrent agent that will be added
name	The name of the agent

List<string> ActressMas.ConcurrentEnvironment.AllAgents ()

Returns a list with the names of all the agents.

Returns:

List<string> ActressMas.ConcurrentEnvironment.AllContainers ()

Returns a list with the names of all the containers in the distributed system. This list may change over time, as some new containers may get connected and existing ones may disconnect.

Returns:

List<string> ActressMas.ConcurrentEnvironment.FilteredAgents (string *nameFragment*)

Returns a list with the names of all the agents that contain a certain string.

Returns:

The name fragment that the agent names should contain

string ActressMas.ConcurrentEnvironment.RandomAgent ()

Returns the name of a randomly selected agent from the environment

Returns:

string ActressMas.ConcurrentEnvironment.RandomAgent (Random *rand*)

Returns the name of a randomly selected agent from the environment using a predefined random number generator. This is useful for experiments involving non-determinism, but which should be repeatable for analysis and debugging.

Parameters:

rand	The random number generator which should be non-null and instantiated using a seed

Returns:

void ActressMas.ConcurrentEnvironment.Remove (ConcurrentAgent *agent*)

Stops the execution of the agent and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

Parameters:

agent	The agent to be removed

void ActressMas.ConcurrentEnvironment.Remove (string *agentName*)

Stops the execution of the agent identified by name and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

Parameters:

agentName	The name of the agent to be removed

void ActressMas.ConcurrentEnvironment.Send (Message *message*)

Sends a message from the outside of the multiagent system. Whenever possible, the agents should use the Send method of their own class, not the Send method of the environment. This method can also be used to simulate a forwarding behavior.

Parameters:

message	The message to be sent

void ActressMas.ConcurrentEnvironment.WaitAll ()

Prevents the program from closing by waiting as long as some agents still run in the environment. This method should be used at the end of the main program, after all the agents have been added to the environment and started.

Property Documentation

int ActressMas.ConcurrentEnvironment.NoAgents `[get]`

The number of agents in the environment

string ActressMas.ConcurrentEnvironment.ContainerName `[get]`

The name of the container that contains the environment. If the container is not set or not connected to the server, this method will return the empty string.

Dictionary<string, dynamic> ActressMas.ConcurrentEnvironment.Memory `[get]`, `[set]`

An object that can be used as a shared memory by the agents.

ActressMas.Container Class Reference

A container contains an environment and is connected to a server. It facilitates the move of agents in a distributed system.

Public Member Functions

- **Container** (string serverIP, int serverPort, string name)
  *Initializes a new instance of the **Container** class.*

- List< string > **AllContainers** ()
  Returns a list with the names of all the containers in the distributed system. This list may change over time, as some new containers may get connected and existing ones may disconnect.

- void **RunConcurrentMas** (**ConcurrentEnvironment** environment, **RunnableMas** mas)
  Starts the execution of the concurrent multiagent system defined in the environment.

- void **RunTurnBasedMas** (**TurnBasedEnvironment** environment, **RunnableMas** mas)
  Starts the execution of the turn-based multiagent system defined in the environment.

- void **Start** ()
  Tries to connect to the server and activates the container.

- void **Stop** ()
  Disconnects from the server and deactivates the container.

Properties

- string **Name** `[get]`
  The name of the container. If the container is not connected to the server, this method will return the empty string.

Events

- NewTextEventHandler **NewText**
  An event handler for the ongoing messages provided by the container.

Detailed Description

A container contains an environment and is connected to a server. It facilitates the move of agents in a distributed system.

Constructor & Destructor Documentation

ActressMas.Container.Container (string *serverIP*, int *serverPort*, string *name*)

Initializes a new instance of the **Container** class.

Parameters:

serverIP	The IP address of the server
serverPort	The port number of the server
name	The name of the container. The name of the container should be unique and cannot contain spaces.

Member Function Documentation

List<string> ActressMas.Container.AllContainers ()

Returns a list with the names of all the containers in the distributed system. This list may change over time, as some new containers may get connected and existing ones may disconnect.

Returns:

void ActressMas.Container.RunConcurrentMas (ConcurrentEnvironment *environment*, RunnableMas *mas*)

Starts the execution of the concurrent multiagent system defined in the environment.

Parameters:

environment	The concurrent environment
mas	The multiagent system to be executed

void ActressMas.Container.RunTurnBasedMas (TurnBasedEnvironment *environment*, RunnableMas *mas*)

Starts the execution of the turn-based multiagent system defined in the environment.

Parameters:

environment	The turn-based environment
mas	The multiagent system to be executed

void ActressMas.Container.Start ()

Tries to connect to the server and activates the container.

void ActressMas.Container.Stop ()

Disconnects from the server and deactivates the container.

Property Documentation

string ActressMas.Container.Name `[get]`

The name of the container. If the container is not connected to the server, this method will return the empty string.

Event Documentation

NewTextEventHandler ActressMas.Container.NewText

An event handler for the ongoing messages provided by the container.

ActressMas.Environment Class Reference

An abstract base class for environments. You must use **ConcurrentEnvironment** or **TurnBasedEnvironment**.
Inherited by **ActressMas.ConcurrentEnvironment**, and **ActressMas.TurnBasedEnvironment**.

Detailed Description

An abstract base class for environments. You must use **ConcurrentEnvironment** or **TurnBasedEnvironment**.

ActressMas.Info Class Reference

Information about **ActressMas** version

Static Public Attributes

- static readonly string **Version** = "ActressMas Version 2.1"
  ActressMas *current version*

Detailed Description

Information about **ActressMas** version

Member Data Documentation

readonly string ActressMas.Info.Version = "ActressMas Version 2.1"`[static]`

ActressMas current version

ActressMas.Message Class Reference

A message that the agents use to communicate. In an agent-based system, the communication between the agents is exclusively performed by exchanging messages.

Public Member Functions

- **Message** ()
  *Initializes a new instance of the **Message** class with an empty message.*

- **Message** (string sender, string receiver, string content)
  *Initializes a new instance of the **Message** class.*

- **Message** (string sender, string receiver, string content, string conversationId)
  *Initializes a new instance of the **Message** class.*

- void **Parse** (out string action, out List< string > parameters)
  Parses the content of a message and identifies the action (similar, e.g., to a performative) and the list of parameters.

- void **Parse** (out string action, out string parameters)
  Parses the content of a message and identifies the action (similar, e.g., to a performative) and the parameters concatenated in a string.

- void **Parse1P** (out string action, out string parameter)
  Parses the content of a message and identifies the action (similar, e.g., to a performative) and the single parameter.

- string **Format** ()
  Returns a string of the form "[Sender -> Receiver]: Content"

Properties

- string **Content** `[get, set]`
  The content of the message.

- string **ConversationId** `[get, set]`
  The conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

- string **Receiver** `[get, set]`
  The name of the agent that needs to receive the message

- string **Sender** `[get, set]`
  The name of the agent that sends the message

Detailed Description

A message that the agents use to communicate. In an agent-based system, the communication between the agents is exclusively performed by exchanging messages.

Constructor & Destructor Documentation

ActressMas.Message.Message ()

Initializes a new instance of the **Message** class with an empty message.

ActressMas.Message.Message (string *sender*, string *receiver*, string *content*)

Initializes a new instance of the **Message** class.

Parameters:

sender	The name of the agent that sends the message
receiver	The name of the agent that needs to receive the message
content	The content of the message

ActressMas.Message.Message (string *sender*, string *receiver*, string *content*, string *conversationId*)

Initializes a new instance of the **Message** class.

Parameters:

sender	The name of the agent that sends the message
receiver	The name of the agent that needs to receive the message
content	The content of the message
conversationId	The conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

---

Member Function Documentation

void ActressMas.Message.Parse (out string *action*, out List< string > *parameters*)

Parses the content of a message and identifies the action (similar, e.g., to a performative) and the list of parameters.

void ActressMas.Message.Parse (out string *action*, out string *parameters*)

Parses the content of a message and identifies the action (similar, e.g., to a performative) and the parameters concatenated in a string.

void ActressMas.Message.Parse1P (out string *action*, out string *parameter*)

Parses the content of a message and identifies the action (similar, e.g., to a performative) and the single parameter.

string ActressMas.Message.Format ()

Returns a string of the form "[Sender -> Receiver]: Content"

Returns:

Property Documentation

string ActressMas.Message.Content `[get], [set]`

The content of the message.

string ActressMas.Message.ConversationId `[get], [set]`

The conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

string ActressMas.Message.Receiver `[get], [set]`

The name of the agent that needs to receive the message

string ActressMas.Message.Sender `[get], [set]`

The name of the agent that sends the message

ActressMas.NewTextEventArgs Class Reference

The class that defines a message from a server or a container.
Inherits EventArgs.

Properties

- string **Text** `[get]`
  The text of the message

Detailed Description

The class that defines a message from a server or a container.

Property Documentation

string ActressMas.NewTextEventArgs.Text `[get]`

The text of the message

ActressMas.RunnableMas Class Reference

An abstract class which should be derived in order to specify the multiagent system with mobile agents that will be run in the environment of a container.

Public Member Functions

- virtual void **RunConcurrentMas** (**ConcurrentEnvironment** env)
  Starts the execution of a concurrent environment within a container

- virtual void **RunTurnBasedMas** (**TurnBasedEnvironment** env)
  Starts the execution of a turn-based environment within a container

---

Detailed Description

An abstract class which should be derived in order to specify the multiagent system with mobile agents that will be run in the environment of a container.

Member Function Documentation

virtual void ActressMas.RunnableMas.RunConcurrentMas (ConcurrentEnvironment *env*) `[virtual]`

Starts the execution of a concurrent environment within a container

Parameters:

env	The concurrent environment

virtual void ActressMas.RunnableMas.RunTurnBasedMas (TurnBasedEnvironment *env*) `[virtual]`

Starts the execution of a turn-based environment within a container

Parameters:

env	The turn-based environment

ActressMas.Server Class Reference

A server that ensures the communication of containers, e.g. for the movement of agents, in a distributed system.

Public Member Functions

- **Server** (int port, int ping)
  *Initializes a new instance of the **Server** class.*

- void **Start** ()
  Tries to start the server

- void **Stop** ()
  Stops the server

Events

- NewTextEventHandler **NewText**
  An event handler for the ongoing messages provided by the server.

Detailed Description

A server that ensures the communication of containers, e.g. for the movement of agents, in a distributed system.

Constructor & Destructor Documentation

ActressMas.Server.Server (int *port*, int *ping*)

Initializes a new instance of the **Server** class.

Parameters:

port	The port number of the server
ping	The time interval (in miliseconds) for the ping messages, needed to check if the containers are still alive

---

Member Function Documentation

void ActressMas.Server.Start ()

Tries to start the server

void ActressMas.Server.Stop ()

Stops the server

Event Documentation

NewTextEventHandler ActressMas.Server.NewText

An event handler for the ongoing messages provided by the server.

ActressMas.TurnBasedAgent Class Reference

The base class for an agent that runs on a turn-based manner in its environment. You must create your own agent classes derived from this abstract class.
Inherits **ActressMas.Agent**.

Public Member Functions

- virtual void **Act** (Queue< **Message** > messages)
  This is the method that is called once a turn. This is where the main logic of the agent should be placed. Once a message has been handled, it should be removed from the queue, using e.g. the Dequeue method.

- void **Broadcast** (string content, bool includeSender=false, string conversationId="")
  Sends a message to all the agents in the environment.

- bool **CanMove** (string destination)
  Tests whether the agent can move to a certain remote container.

- virtual void **LoadState** (**AgentState** state)
- void **Move** (string destination)
  The method that should be called when the agent wants to move to a different container.

- virtual **AgentState SaveState** ()
  Exports the state of the agent, so it can be serialized when moving to another container.

- void **Send** (string receiver, string content, string conversationId="")
  Sends a message to a specific agent, identified by name.

- void **SendToMany** (List< string > receivers, string content, string conversationId="")
  Sends a message to a specific set of agents, identified by name.

- virtual void **Setup** ()
  This method is called as the first turn or right after an agent has moved to a new container. It is similar to the constructor of the class, but it may be used for agent-related logic, e.g. for sending initial message(s).

- void **Stop** ()
  Stops the execution of the agent and removes it from the environment. Use the Stop method instead of Environment.Remove when the decision to be stopped belongs to the agent itself.

Properties

- string **Name** `[get, set]`
  The name of the agent. Each agent must have a unique name in its environment. Most operations are performed using agent names rather than agent objects.

- **TurnBasedEnvironment Environment** `[get, set]`

The environment in which the agent runs. A turn-based agent can only run in a turn-based environment.

Detailed Description

The base class for an agent that runs on a turn-based manner in its environment. You must create your own agent classes derived from this abstract class.

Member Function Documentation

virtual void ActressMas.TurnBasedAgent.Act (Queue< Message > *messages*) `[virtual]`

This is the method that is called once a turn. This is where the main logic of the agent should be placed. Once a message has been handled, it should be removed from the queue, using e.g. the Dequeue method.

Parameters:

messages	The messages that the agent has received during the previous turn(s) and should respond to

void ActressMas.TurnBasedAgent.Broadcast (string *content*, bool *includeSender* = `false`, string *conversationId* = `""`)

Sends a message to all the agents in the environment.

Parameters:

content	The content of the message
includeSender	Whether the sender itself receives the message or not
conversationId	A conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

bool ActressMas.TurnBasedAgent.CanMove (string *destination*)

Tests whether the agent can move to a certain remote container.

Parameters:

destination	The name of the container that the agent wants to move to

Returns:

virtual void ActressMas.TurnBasedAgent.LoadState (AgentState *state*) `[virtual]`

Imports the state of the agent, after it has moved from another container.

Parameters:

state	The state of the agent

void ActressMas.TurnBasedAgent.Move (string *destination*)

The method that should be called when the agent wants to move to a different container.

Parameters:

destination	The name of the container that the agent wants to move to

virtual AgentState ActressMas.TurnBasedAgent.SaveState () `[virtual]`

Exports the state of the agent, so it can be serialized when moving to another container.

Returns:

void ActressMas.TurnBasedAgent.Send (string *receiver*, string *content*, string *conversationId* = "")

Sends a message to a specific agent, identified by name.

Parameters:

receiver	The agent that will receive the message
content	The content of the message
conversationId	A conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

void ActressMas.TurnBasedAgent.SendToMany (List< string > *receivers*, string *content*, string *conversationId* = "")

Sends a message to a specific set of agents, identified by name.

Parameters:

receivers	The list of agents that will receive the message
content	The content of the message
conversationId	A conversation identifier, for the cases when a conversation involves multiple messages that refer to the same topic

virtual void ActressMas.TurnBasedAgent.Setup () `[virtual]`

This method is called as the first turn or right after an agent has moved to a new container. It is similar to the constructor of the class, but it may be used for agent-related logic, e.g. for sending initial message(s).

void ActressMas.TurnBasedAgent.Stop ()

Stops the execution of the agent and removes it from the environment. Use the Stop method instead of Environment.Remove when the decision to be stopped belongs to the agent itself.

Property Documentation

string ActressMas.TurnBasedAgent.Name `[get], [set]`

The name of the agent. Each agent must have a unique name in its environment. Most operations are performed using agent names rather than agent objects.

TurnBasedEnvironment ActressMas.TurnBasedAgent.Environment `[get], [set]`

The environment in which the agent runs. A turn-based agent can only run in a turn-based environment.

ActressMas.TurnBasedEnvironment Class Reference

A turn-based environment, where all the agents are executed sequentially or in a random order during a turn.
Inherits **ActressMas.Environment**.

Public Member Functions

- **TurnBasedEnvironment** (int numberOfTurns=0, int delayAfterTurn=0, bool randomOrder=true, Random rand=null)
  *Initializes a new instance of the **TurnBasedEnvironment** class.*

- void **Add** (**TurnBasedAgent** agent)
  Adds an agent to the environment. The agent should already have a name and its name should be unique.

- void **Add** (**TurnBasedAgent** agent, string name)
  Adds an agent to the environment. Its name should be unique.

- List< string > **AllAgents** ()
  Returns a list with the names of all the agents.

- List< string > **AllContainers** ()
  Returns a list with the names of all the containers in the distributed system. This list may change over time, as some new containers may get connected and existing ones may disconnect.

- void **Continue** (int noTurns=0)
  Continues the simulation for an additional number of turns, after a simulation has finished.

- List< string > **FilteredAgents** (string nameFragment)
  Returns a list with the names of all the agents that contain a certain string.

- string **RandomAgent** ()
  Returns the name of a randomly selected agent from the environment

- string **RandomAgent** (Random rand)
  Returns the name of a randomly selected agent from the environment using a predefined random number generator. This is useful for experiments involving non-determinism, but which should be repeatable for analysis and debugging.

- void **Remove** (**TurnBasedAgent** agent)
  Stops the execution of the agent and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

- void **Remove** (string agentName)
  Stops the execution of the agent identified by name and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

- void **Send** (**Message** message)
  Sends a message from the outside of the multiagent system. Whenever possible, the agents should use the Send method of their own class, not the Send method of the environment. This method can also be used to simulate a forwarding behavior.

- virtual void **SimulationFinished** ()
  A method that may be optionally overriden to perform additional processing after the simulation has finished.

- void **Start** ()
  Starts the simulation.

- virtual void **TurnFinished** (int turn)
  A method that may be optionally overriden to perform additional processing after a turn of the the simulation has finished.

Properties

- int **NoAgents** `[get]`
  The number of agents in the environment

- string **ContainerName** `[get]`
  The name of the container that contains the environment. If the container is not set or not connected to the server, this method will return the empty string.

- Dictionary< string, dynamic > **Memory** `[get, set]`
  An object that can be used as a shared memory by the agents.

Detailed Description

A turn-based environment, where all the agents are executed sequentially or in a random order during a turn.

Constructor & Destructor Documentation

ActressMas.TurnBasedEnvironment.TurnBasedEnvironment (int *numberOfTurns* = `0`, int *delayAfterTurn* = `0`, bool *randomOrder* = `true`, Random *rand* = `null`)

Initializes a new instance of the **TurnBasedEnvironment** class.

Parameters:

numberOfTurns	The maximum number of turns of the simulation. The simulation may stop earlier if there are no more agents in the environment. If the number of turns is 0, the simulation runs indefinitely, or until there are no more agents in the environment.

delayAfterTurn	A delay (in miliseconds) after each turn
randomOrder	Whether the agents should be run in a random order (different each turn) or sequentially
rand	A random number generator for non-deterministic but repeatable experiments. It should instantiated using a seed. If it is null, a new Random object is created and used.

Member Function Documentation

void ActressMas.TurnBasedEnvironment.Add (TurnBasedAgent *agent*)

Adds an agent to the environment. The agent should already have a name and its name should be unique.

Parameters:

agent	The concurrent agent that will be added

void ActressMas.TurnBasedEnvironment.Add (TurnBasedAgent *agent*, string *name*)

Adds an agent to the environment. Its name should be unique.

Parameters:

agent	The concurrent agent that will be added
name	The name of the agent

List<string> ActressMas.TurnBasedEnvironment.AllAgents ()

Returns a list with the names of all the agents.

Returns:

List<string> ActressMas.TurnBasedEnvironment.AllContainers ()

Returns a list with the names of all the containers in the distributed system. This list may change over time, as some new containers may get connected and existing ones may disconnect.

Returns:

void ActressMas.TurnBasedEnvironment.Continue (int *noTurns* = 0)

Continues the simulation for an additional number of turns, after a simulation has finished.

Parameters:

noTurns	The maximum number of turns of the continued simulation. The simulation may stop earlier if there are no more agents in the environment. If the number of turns is 0, the simulation runs indefinitely, or until there are no more agents

	in the environment.

List<string> ActressMas.TurnBasedEnvironment.FilteredAgents (string *nameFragment*)

Returns a list with the names of all the agents that contain a certain string.

Returns:

The name fragment that the agent names should contain

string ActressMas.TurnBasedEnvironment.RandomAgent ()

Returns the name of a randomly selected agent from the environment

Returns:

string ActressMas.TurnBasedEnvironment.RandomAgent (Random *rand*)

Returns the name of a randomly selected agent from the environment using a predefined random number generator. This is useful for experiments involving non-determinism, but which should be repeatable for analysis and debugging.

Parameters:

rand	The random number generator which should be non-null and instantiated using a seed

Returns:

void ActressMas.TurnBasedEnvironment.Remove (TurnBasedAgent *agent*)

Stops the execution of the agent and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

Parameters:

agent	The agent to be removed

void ActressMas.TurnBasedEnvironment.Remove (string *agentName*)

Stops the execution of the agent identified by name and removes it from the environment. Use the Remove method instead of Agent.Stop when the decision to stop an agent does not belong to the agent itself, but to some other agent or to an external factor.

Parameters:

agentName	The name of the agent to be removed

void ActressMas.TurnBasedEnvironment.Send (Message *message*)

Sends a message from the outside of the multiagent system. Whenever possible, the agents should use the Send method of their own class, not the Send method of the environment. This method can also be used to simulate a forwarding behavior.

Parameters:

message	The message to be sent

virtual void ActressMas.TurnBasedEnvironment.SimulationFinished () `[virtual]`

A method that may be optionally overriden to perform additional processing after the simulation has finished.

void ActressMas.TurnBasedEnvironment.Start ()

Starts the simulation.

virtual void ActressMas.TurnBasedEnvironment.TurnFinished (int *turn*) `[virtual]`

A method that may be optionally overriden to perform additional processing after a turn of the the simulation has finished.

Parameters:

turn	The turn that has just finished

Property Documentation

int ActressMas.TurnBasedEnvironment.NoAgents `[get]`

The number of agents in the environment

string ActressMas.TurnBasedEnvironment.ContainerName `[get]`

The name of the container that contains the environment. If the container is not set or not connected to the server, this method will return the empty string.

Dictionary<string, dynamic> ActressMas.TurnBasedEnvironment.Memory `[get]`, `[set]`

An object that can be used as a shared memory by the agents.

Index