Reference Documentation

SIMON

AI Learning Framework

Seo Jaesuk, Thomas Park,
Oh Hyeseong, Jack Kim

Version 1.0

User Manual

June 20, 2014

SIMON Framework v1.0 Reference Documentation

Overview of SIMON Framework

1. Introduction

The SIMON Framework is an Open-source framework for Game AI modeling and implementation. It initially implemented by project team named SIMON. SIMON allows you to make Non Player Character having unexpected actions and properties, and it could make the chances to have high quality of the degree of freedom.

SIMON is designed to make the learning routine easily. The framework has the core routine to link between framework and user defined environment. After being aware of it, the framework will be able to use the games users made easily.

1.1 Background

Many kinds of Game AI are designed by using 'Rule-based model' which consists of 'if' and 'else if' clauses. If you look on the bright side, it is deterministic and causes predictable result. However it could not make high degree of freedom easily and is able to make uniform actions. Instead of making an AI model roughly, there are several ways to implement it. So many kinds of AI are implemented by using combinations of various ways.

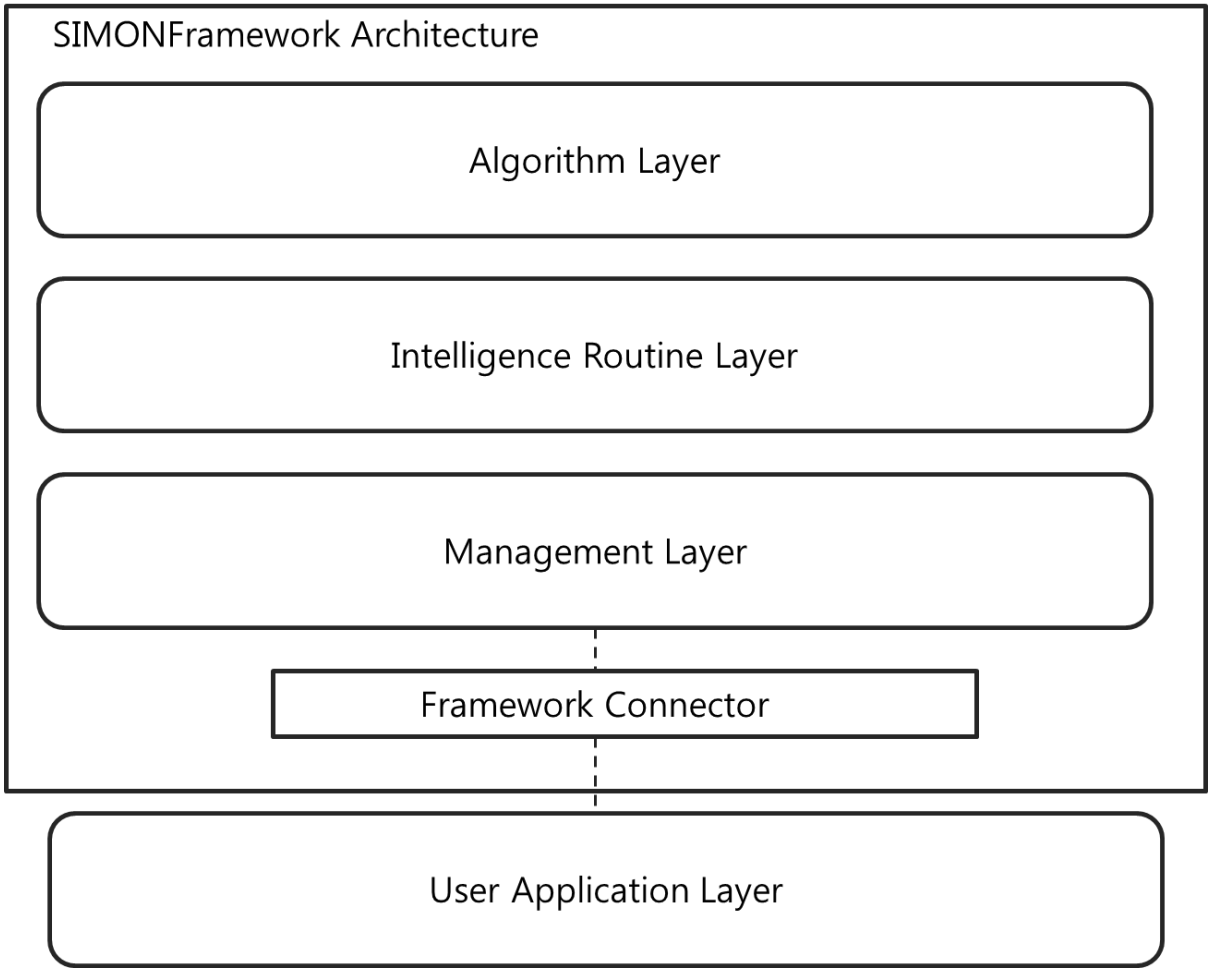
1.2 Purpose

SIMON Framework has been developed to make freewheeling environment by using non-deterministic algorithm.

SIMON Framework is designed to run every routine inside the framework. If developers knew and constructed AI model well, then SIMON could perform it well.

1.3 Framework Architecture

SIMON Framework has many independent layers, and SIMON Manager do the routines using its various modules on each layer. Its structures are described below.



Framework Connector

Framework Connector module is used to linking between the SIMON framework and user level applications or external developing tool. Framework has been implemented by C# based on .Net, and it will serve usable connectors to use it in other tools.

Connector module should have been served to suit target application's environment. Also it has to keep the integrity of the framework. In order to satisfy some conditions, it has been constructed to use manager routine as a singleton.

According to the environment which user application uses, the implementations of the connectors are different, however the interface which manager uses is same.

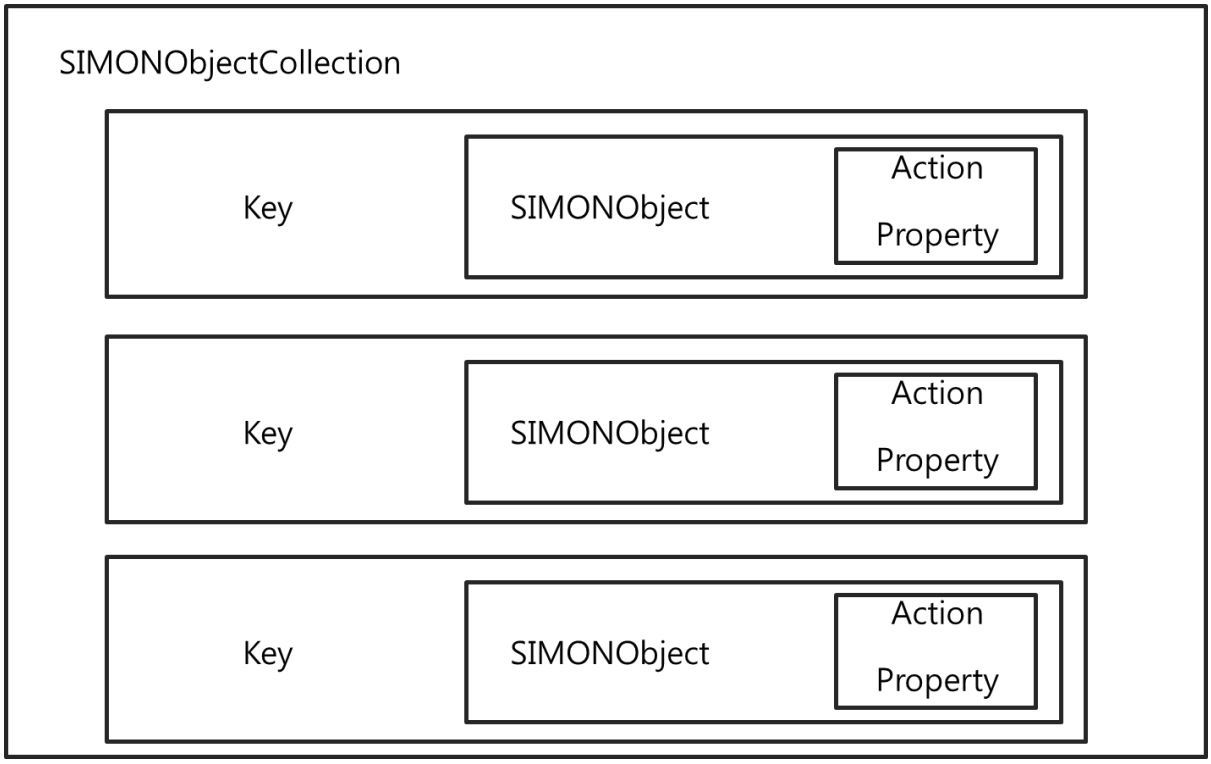
Management Layer

Management Layer included the routine that manages and controls the modules SIMONFramework provides. Although the implementation which would be applied to the user application is included in the SIMON Connector, the function it refers uses modules of the management layer. There are three modules included in the management layer.

- SIMON Object management module
- Constructing learning environment module
- Performing intelligence routine module

SIMONObject management module

To define AI model of the objects is accomplished by SIMONObject class. Each of 'SIMONObject' objects are managed by management layer as a dictionary entry. Describe this relation below.



The key values used to implement SIMONCollection basically consist of object's identical number. The major elements of the SIMONObject are 'action' and 'property'.

- Action

'Action' means the act of the SIMONObject. The class names SIMONAction is supported to realize this notion. There are members that SIMONAction class has.

```
public class SIMONAction
{
    public String ActionName { get; set; }
    public String ActionFunctionName { get; set; }
    public String ExecutionFunctionName { get; set; }
    public String FitnessFunctionName { get; set; }
    public List<SIMONGene> ActionDNA { get; set; }

    .
    .
    .
}
```

The class has the name of the action, the name of the action function, the name of the execution function, the name of the fitness function and the list of the weights as an action's DNA.

```
public class SIMONGene
{
    [XmlElement("ElementName")]
    public String ElementName { get; set; }
    [XmlElement("ElementWeight")]
    public Double ElementWeight { get; set; }

    .
    .
    .

}
```

SIMONGene is the class that consists of DNA of SIMON objects. It has the name of the weight and the value of the weight. Also it is used to the element of SIMONAction 's DNA list.

- Property

It is the unit which describes the SIMONObject 's user-defined traits. The class named SIMONProperty serves about this function. SIMONProperty class has the members below.

```
public class SIMONProperty
{
    [XmlElement("PropertyName")]
    public String PropertyName { get; set; }
    [XmlElement("PropertyValue")]
    public Double PropertyValue { get; set; }
    [XmlElement("Inherit")]
    public Boolean Inherit { get; set; }

.
.
.

}
```

The class includes three members the name of itself, the value it has and whether it would be inherited or not.

Constructing learning environment module

SIMONFramework supports some functions to run learning routine.

- SIMONFunction List

SIMONFramework performs linking between user application and framework by using the method of invoking functions dynamically. In order to do this, delegates of the Simon Functions are saved as a dictionary abstraction data type. Each of

element's default key is the name of the function and value is the delegate of the function which implements SIMONFunction interface. The structure of it describes below.

```
public delegate Object SIMONFunction(SIMONObject One, SIMONObject[] TheOthers);
```

The method has two parameters. One means object itself and the other is the list of another members in the group.

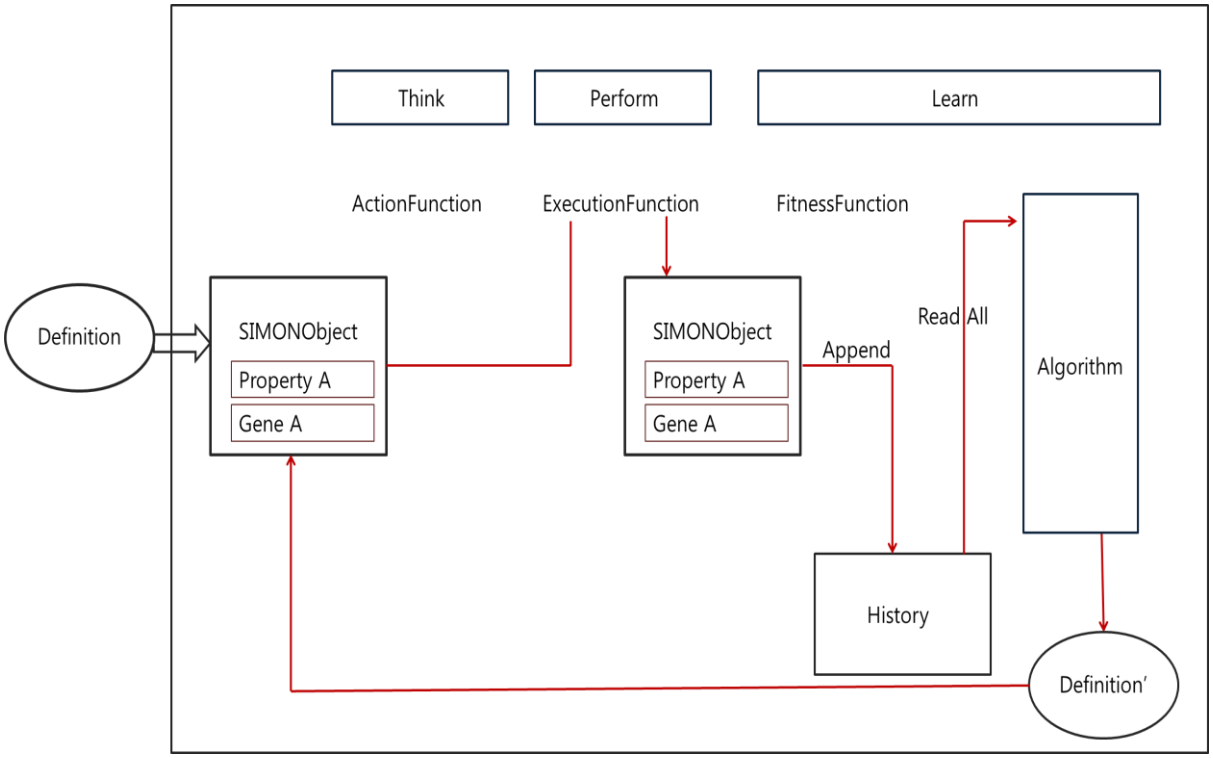
- SIMON Definition file

SIMONFramework uses XML serialize / deserialize for saving the constitution of SIMONObjects in the local storage. The implementation of definition files expresses as a form of hierarchical tree, and default encoding set of it is UTF-8.

Framework supports implementation of serialization and deserialization by using SIMONUtility class. SIMONUtility class is made for using singleton in the management layer.

Performing intelligence routine module

Implementation of learning algorithms is not included in the management routine. However there is a module that makes intelligence routine run, and this performs procedures in the defined workspace. The main sequential process which SIMON Framework performs is described below.

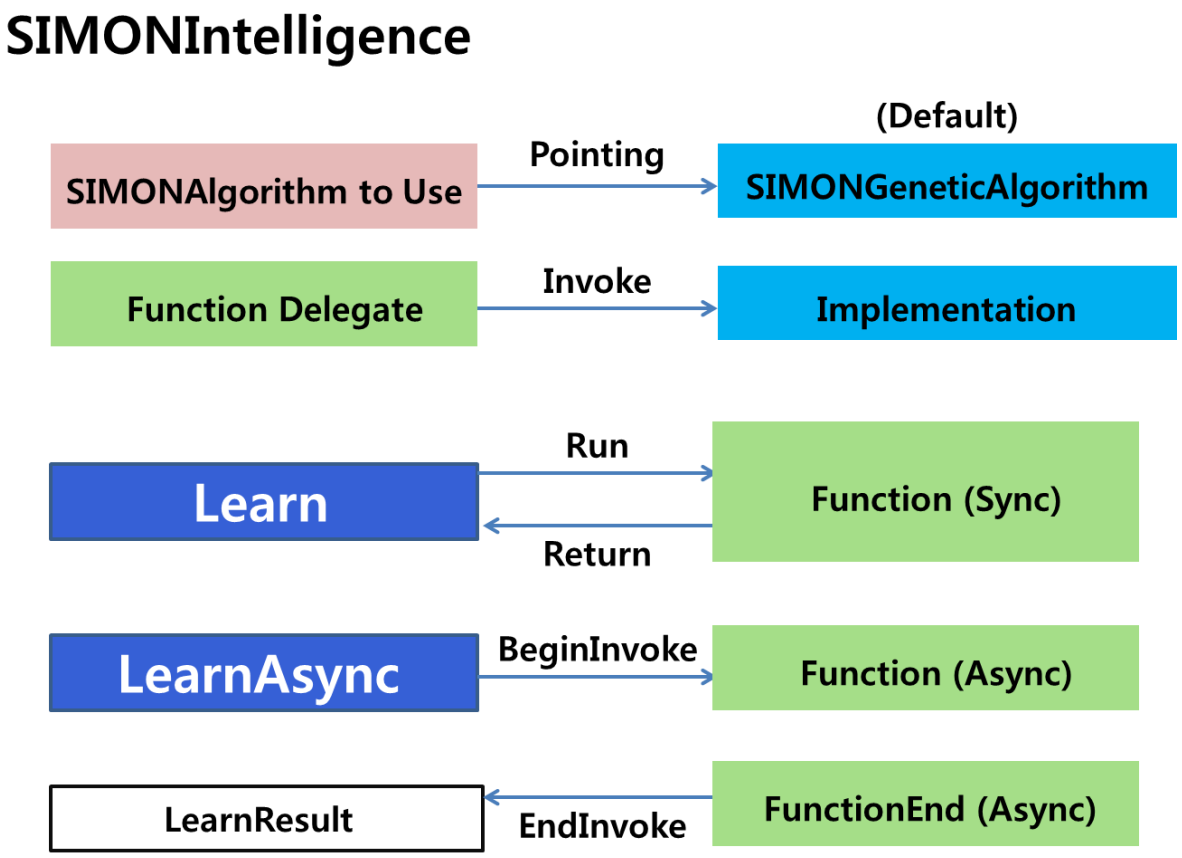


It is the sequence flow that management routine layer does. Detailed movements are performed inside each layer.

Intelligence Routine Layer

Intelligence routine layer controls the implementation of the algorithm layer which is located deeply inside the SIMON Framework by using delegates. It supports synchronous / asynchronous method to manage and control delegates. Also it has the callback method to run asynchronous work.

In this layer, it constraints algorithm to use for learning by designating **Enum** type value. Also it implements a bridge which is able to approach algorithm class for AI.



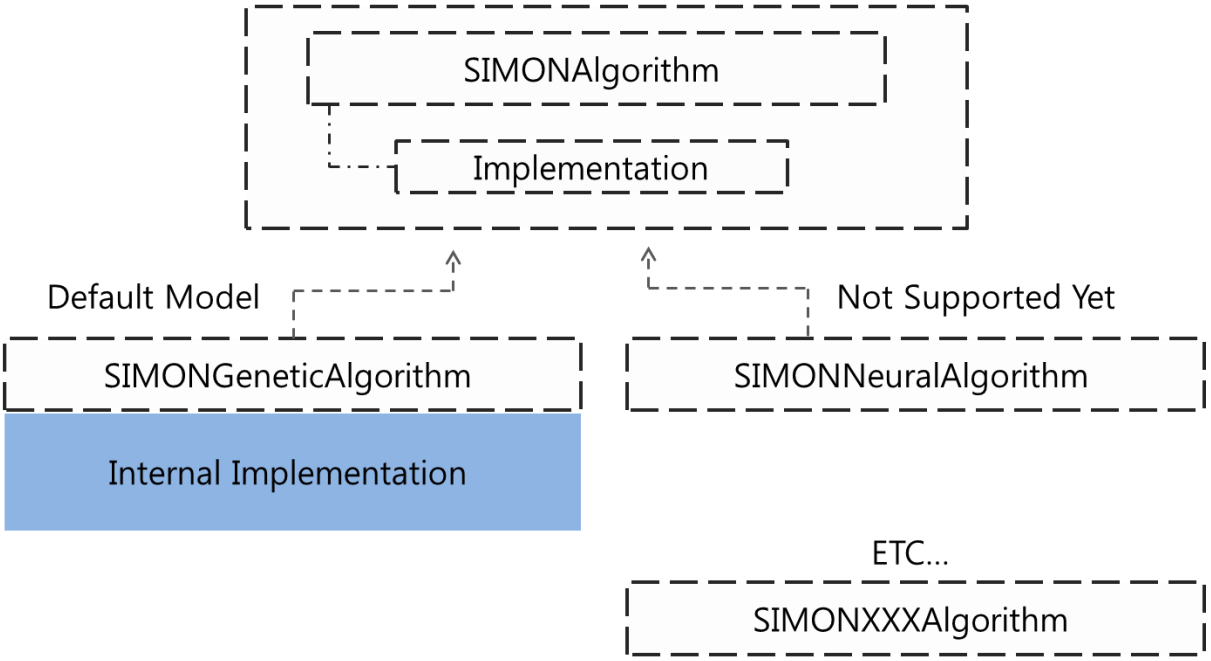
Intelligence Routine Layer consists of learning module, and supported the functions above.

Algorithm Layer

Algorithm layer implements and runs a learning algorithm that is performed the most deeply inside the SIMON Framework.

In this version, SIMON Framework basically uses 'Genetic Algorithm' as an algorithm class. The implementation of the class is indicated by a delegate from the intelligence routine layer, and it controls by invocation synchronous and asynchronous.

SIMONAlgorithm Interface



SIMONGeneticAlgorithm class which implements SIMONAlgorithm interface is used to teach SIMONObjects by using non-deterministic genetic algorithm. Learning method using neural network will also be supported next version. Any class implements SIMONAlgorithm interface could be run by Intelligence routine.

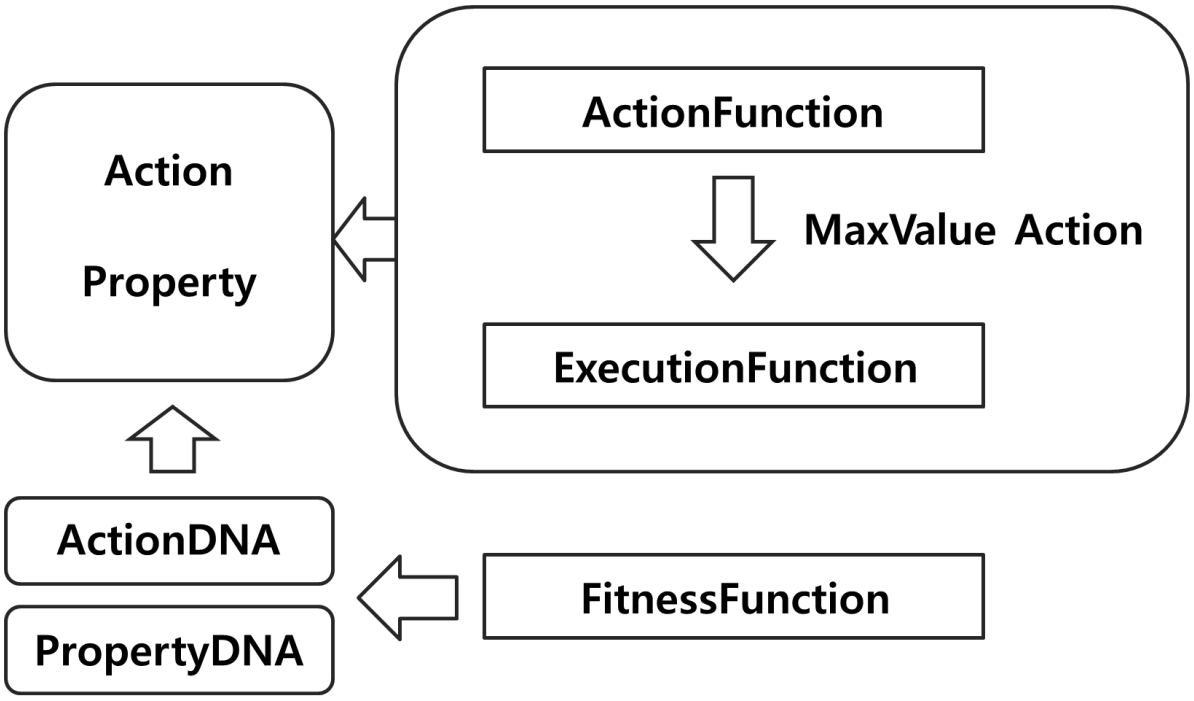
There are two option properties to configure internal action of the SIMONGeneticAlgorithm, the chance of mutation and the rate of mutation. These values are directly used to run default genetic algorithm.

2. Core technology of SIMON Framework

2.1 SIMON Routine

SIMON Framework implements decision and learning routine for SIMON Collection as a main sequence routine. Action function is used to judge actions of the elements in the SIMON Collection, and it uses maximum output value of the Action functions for decision value. Action would be performed in the Execution function. This kind of decision – action pattern compose main sequence of the SIMON Routine. Learning routine is performed independent, and Fitness functions are used to do it.

Decision – Action – Learning process makes SIMON Framework's main routine, and it is described below.



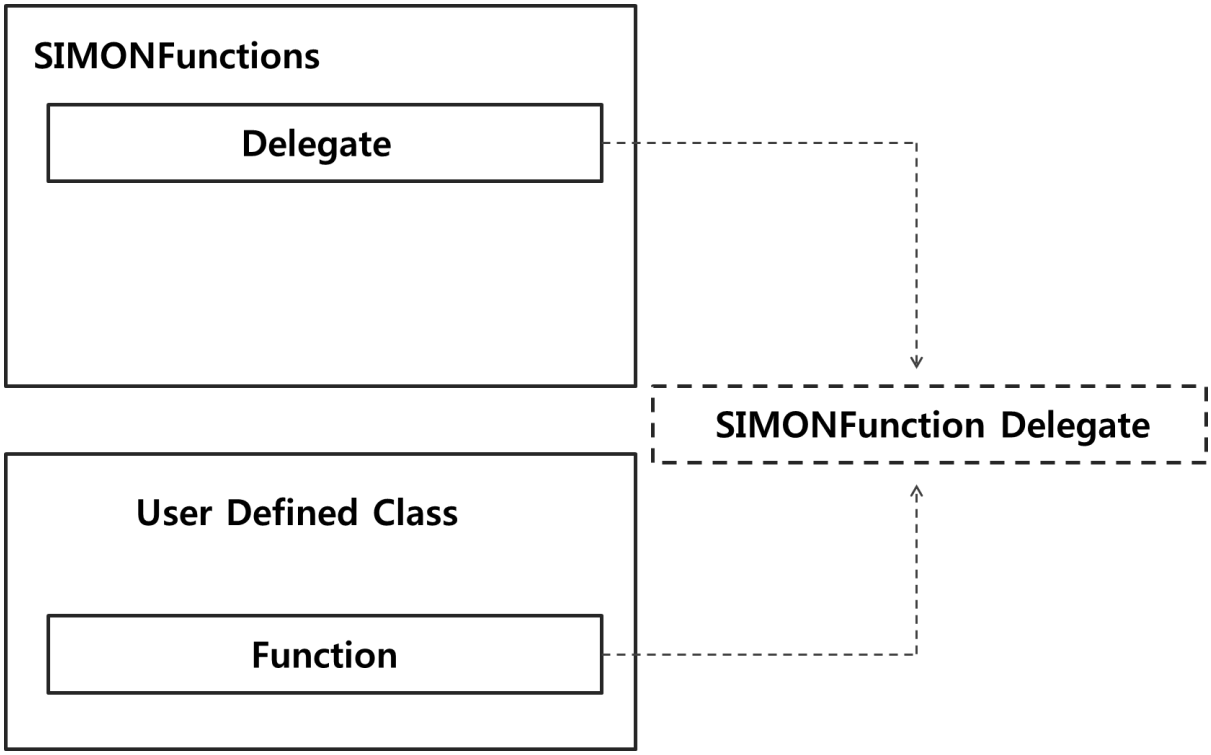
Every function in the routine has to satisfy the form of the SIMON Function's delegates, and it has to be controlled by SIMONManager by registering the method to the manager. The class which is referred basically is 'SIMONFunction', but it could be set by designating the delegates directly.

2.2 Dynamic linking and dependency injection of the SIMON Framework

In order to support AI constructing service for users, SIMON Framework uses delegates and dynamic references using reflection.

Delegates are used to run the indicated method instead of calling the function directly, and reflection searches the class and types of the specific method in runtime.

In the side of designating delegates which users defined at the user application, the control of program's functions moves from user level to framework level. The information of the SIMONObjects is managed by manager, and the user-defined functions routine is controlled by manager, therefore each component runs independently in the framework. Also the framework supports Setter- functions to include user-defined functions for the elements which satisfy the interface.



In this respect, SIMON Framework has the property of dependency injection.

3. How to use SIMON Framework

3.1 User Scenario

SIMON is AI supporting framework based on .Net framework. In this version of the framework, it is required at least 3.5 version of .Net framework project.

Install framework

The newest version of SIMON Framework is able to download from Github address : https://github.com/ParkJinSang/SIMONFramework.

- To use SIMON Framework in .Net project, download SIMON Framework
- To use SIMON Framework in Unity project, download SIMON Framework Export package for Unity.

Link SIMON

To use SIMON Framework for .Net project, firstly you should add SIMON Framework to the project. And then include the namespace using "**using**" keyword like below.

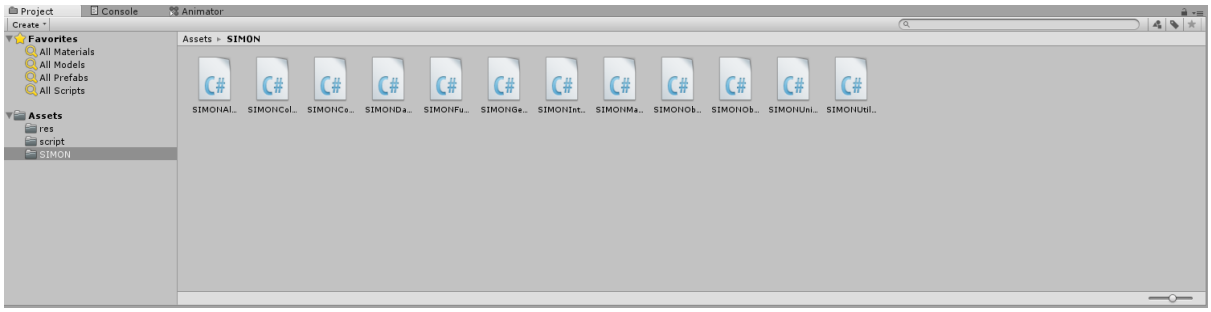
```
1
2  using System;
3  using System.IO;
4  using SIMONFramework;
5
```

The elements of the SIMON Framework have the prefix 'SIMON' , so you could easily use the elements.

SIMON can be used in the external developing tool, and there is information about the way to install and how to use. There are ways of use.

- Usage of SIMON Framework in the Unity

To use SIMON in the Unity, download the export package and import it. And then, there would be an installed package inside the SIMON directory.



Installing the framework, you can use the SIMON from the namespace inside the script. However we recommend to using it from global class.

```
1
2 using UnityEngine;
3 using System.Collections;
4 using System.Collections.Generic;
5 using SIMONFramework;
6
7 public static class SIMON{
8     public static SIMONUnity GlobalSIMON = new SIMONUnity();
9 }
10
```

Framework supports connector script "SIMONUnity.cs" in order to be used in the Unity studio. You can use all the functions of the SIMON and Unity properly from this connector. Also it has been implemented by singleton, integrity of the structure could be assured.

Create SIMONObject

SIMONObject is the smallest unit to apply AI model in the SIMON Framework.

SIMONObject consists of properties, actions and DNA for each of them, and it can be defined everywhere if you can use framework inside. How to define

SIMONObject describes below.

```
public Human()
{
    SIMONObject smartOne = new SIMONObject();
    smartOne.ObjectID = "Tom";
    smartOne.Properties.Add(new SIMONProperty("health", 100, true));
    smartOne.Properties.Add(new SIMONProperty("joy", 10, false));
    smartOne.Properties.Add(new SIMONProperty("wealth", 50, true));
    smartOne.Actions.Add(new SIMONAction("Sleep", "SleepAction", "SleepExecution", "SleepFitness", null));
    smartOne.Actions.Add(new SIMONAction("Play", "PlayAction", "PlayExecution", "PlayFitness", null));
    smartOne.Actions.Add(new SIMONAction("Work", "WorkAction", "WorkExecution", "WorkFitness", null));

}
```

The SIMONObject 'smartOne' has the name 'Tom', and 'health, 'joy' and wealth as properties. In addition this man has three defined behaviors 'sleep', 'play' and 'work'.

Next, let's make this man act through SIMON. Add the code in order to define the function and link between a user definition and the SIMON manager.

```
smartOne.ObjectFitnessFunctionName = "TomFitness";
smartOne.UpdatePropertyDNA();

SimonManager.AddMethod("TomFitness", new SIMONFunction(TomFitness));
SimonManager.AddMethod("SleepAction", new SIMONFunction(SleepAction));
SimonManager.AddMethod("SleepExecution", new SIMONFunction(SleepExecution));
SimonManager.AddMethod("SleepFitness", new SIMONFunction(SleepFitness));
SimonManager.AddMethod("PlayAction", new SIMONFunction(PlayAction));
SimonManager.AddMethod("PlayExecution", new SIMONFunction(PlayExecution));
SimonManager.AddMethod("PlayFitness", new SIMONFunction(PlayFitness));
SimonManager.AddMethod("WorkAction", new SIMONFunction(WorkAction));
SimonManager.AddMethod("WorkExecution", new SIMONFunction(WorkExecution));
SimonManager.AddMethod("WorkFitness", new SIMONFunction(WorkFitness));
SimonManager.RegisterSIMONObject(smartOne);
```

Above code is the part matching SIMONObject's action with the defined function with delegates. After matching, you should register the functions and

object itself to the manager object.

```
public object TomFitness(SIMONObject one, SIMONObject[] theOthers)
{
    return one.GetPropertyElement("health") + one.GetPropertyElement("joy") + one.GetPropertyElement("wealth");
}
public object SleepAction(SIMONObject one, SIMONObject[] theOthers)
{
    return -one.GetPropertyElement("health");
}
public object SleepExecution(SIMONObject one, SIMONObject[] theOthers)
{
    one.SetPropertyElement("health", 150);
    return (double)1;
}
public object SleepFitness(SIMONObject one, SIMONObject[] theOthers)
{
    return one.GetPropertyElement("health") + one.GetPropertyElement("joy");
}
public object PlayAction(SIMONObject one, SIMONObject[] theOthers)
{
    return -one.GetPropertyElement("joy") + one.GetPropertyElement("wealth");
}
public object PlayExecution(SIMONObject one, SIMONObject[] theOthers)
{
    one.SetPropertyElement("joy", 150);
    return (double)1;
}
public object PlayFitness(SIMONObject one, SIMONObject[] theOthers)
{
    return one.GetPropertyElement("joy") - one.GetPropertyElement("wealth");
}
public object WorkAction(SIMONObject one, SIMONObject[] theOthers)
{
    return one.GetPropertyElement("wealth") + one.GetPropertyElement("health");
}
public object WorkExecution(SIMONObject one, SIMONObject[] theOthers)
{
    one.SetPropertyElement("wealth", 150);
    return (double)1;
}
public object WorkFitness(SIMONObject one, SIMONObject[] theOthers)
{
    return one.GetPropertyElement("health") + one.GetPropertyElement("joy");
}
```

If you implement the functions which are mapped and registered to the manager, then it is finished to make the SIMONObject move through the framework.

Call SIMONManager

In order to make the object learn using SIMON, you should make the group to

be trained. An object is able to be the group itself, and the routine in the framework is working by the group as a unit. For making the group, use the SIMONCollection class like below.

```
SIMONCollection HumanGroup = SimonManager.CreateSIMONGroup();
HumanGroup.Add(smartOne.ObjectID, smartOne);
```

If the Group is managed well, it is possible to add and remove dynamically. Because elements of the group are specified by key value, we recommend using the object ID as a key and making the object ID as a primary key. SIMONManager is used to learn through group.

```
while (true)
{
    SimonManager.LearnRoutine(HumanGroup);
    System.Threading.Thread.Sleep(1000);
}
```

Running the routine inside the **while** loop, the group named 'HumanGroup' would be learned per a second in above picture.

Using the method **LearnSimulate(),** learning routine could be run asynchronously, and you can set learning rate. If asynchronous scheduling goes well, inner routine performs well.

Configure Learning

In SIMON, developers configure the way of learning properly. In detail, developers could change coefficients inside the algorithm which framework serves. As a result, developers could adjust the effect of the algorithm.

```
SimonManager.ConfigureGeneticLearn(new SIMONGeneticAlgorithm.GeneOption(100, 100));
```

For example, the function above makes developers change the probability of the mutation and the rate of the mutation of the genetic algorithm SIMON Framework basically supports. It assures integrity of the program with external layer.

Use Definition Tool

Making SIMONObject can be easy when you use 'Definition Making Tool'. With this tool, it is easy and convenient to define, publish and reuse the definition of the SIMONObject.

Adding, removing and editing the properties and actions are supported in definition making tool. Besides it is possible to manage many SIMONObjects properly by separating it by the project.

The newest version of definition making tool can be downloaded https://github.com/ParkJinSang/SIMONFramework.

3.2 Design AI Model

Design ActionFunction

ActionFunction represents when the action should be executed and how much weights the action should have. To use SIMON Framework well, we recommend that the method be implemented as a functional formula or a recurrence formula.

The function uses a SIMONObject typed object and SIMONObject array as parameters. There is the method example below.

```
public object SleepAction(SIMONObject one, SIMONObject[] theOthers)
{
    return -one.GetPropertyElement("health");
}
```

The function **SleepAction** returns an inverse of "health" property value. This means the lower the health value of the SIMONObject is the bigger this function returns. In other words, it means

"If one has low health value, then it is highly possible that he would sleep."

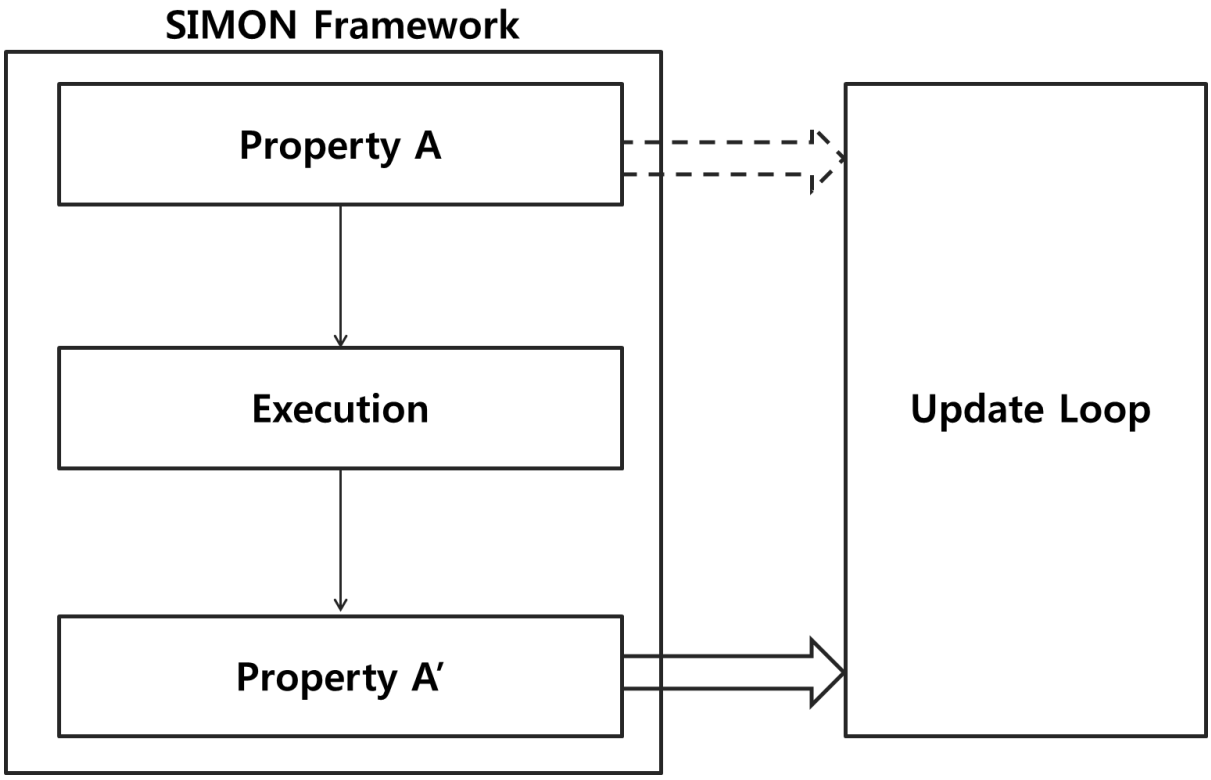
The point is that the sentence does not assure he will sleep, but it just suggest its possibility.

Implementing the functional formula with property values, you can make SIMON Framework handle the multi-dimensional model.

Design ExecutionFunction

ExecutionFunction is the running function as a result of decision. It is related to

the environment with changed property values through actions. SIMON does not interfere in variables or memory of the external layer; however it just affects the properties SIMONObjects have.



The description above mainly appears in the implementation of game AI model with using SIMON Framework. In the update loop, the function to update or draw is fulfilled.

Design FitnessFunction

Although ActionFunction and ExecutionFunction are related to decision routine, FitnessFunction is the function affects learning routine. It makes the patterns on learning and converging fitness values. If FitnessFunction for the action which

Tom sleeps could be defined like below.

```
public object SleepFitness(SIMONObject one, SIMONObject[] theOthers)
{
    return one.GetPropertyElement("health") * one.GetPropertyElement("joy");
}
```

The meaning of the function is that weight of the behavior named 'sleep' is related to the multiplication of health and joy. Therefore the action sleep's learning pattern is affected by coefficients of 'health' and 'joy'.

Design PropertyFitnessFunction

Although the functions ActionFunction, ExecutionFunction and FitnessFunction are modeling function regarding actions, PropertyFitnessFunction is the fitness function about properties. If the AI model of the program is so simple that it could only be described in properties, not actions, it is easy to implement the program with using PropertyFitnessFunction. We recommend using PropertyFitnessFunction and Property DNA for making a simple AI model.

3.3 Extended Implements

Use Dynamic Invoke

SIMON implements the functions by mapping the method and its name runtime. Dynamic binding of the methods is the basic principle. This dynamic binding environment is set, and this could be applied variously.

```
public class SIMONUserFunction : SIMONFunctionInterface
{
    public string[] GetFunctionList()
    {
        string[] arr = { "Move" };
        return arr;
    }
    public int GetFunctionCount()
    {
        return 1;
    }
    public object Move(SIMONObject one, SIMONObject[] theOthers)
    {
        Console.WriteLine("Move Calls");
        return null;
    }
}
```

Given picture describes the class **SIMONUserFunction** which implements the interface SIMONFramework supports. Like this, define the functions into a class to manage source code easily, or it is possible to implements each function in each of classes. If you would like to use some patterns, then you could define a new interface and using it as a container. There are many way to use it.

Use Definition Factory

If you have definitions as elements of the SIMON AI project and you want to make the whole workspace of it, you are able to use the factory which is modeled in SIMONManager. The factory produces copied objects from definition files in the workspace.

```
SIMONObject Jack = SimonManager.CopyDefinitionObject("Tom");
```

With above sentence, you could use the object 'Jack' which has the same properties and actions as the object "Tom". Using this pattern, you are easily able to produce, manage and reproduce the AI model.

3.4 Constraints

Fitness Value Boundary

In SIMON Framework, there are upper and lower boundaries for fitness values. This is for algorithmic implementation and the memory exception which external developing tools may occur, and it is defined +9999.9999 from -9999.9999. We recommend making the fitness value properly by using the traits of **Double** type.

Async Result Handling

SIMON supports the functions about asynchronous work; however the result handling of the objects themselves should be handled by developers. Therefore we recommend managing the result objects separately.