VERSION 1.0

AgentAI

User Manual

horizontal line

# 

[**1. Introduction 2**](#_ye5geyp42t5b)

[1.1 Overview 2](#_qvh8c78g7x4f)

[1.2 Key Features 2](#_9q9j2qqp4yti)

[1.3 System Requirements 3](#_2bpc6ixnbjqh)

[**2. Getting Started 3**](#_9kjfzlskofvv)

[2.1 Installation 3](#_xtl4nq2h8fbl)

[2.2 Setting up AgentAI in Unity 3](#_c2vsjdnf5620)

[**3. Understanding AgentAI 3**](#_vg5mea42f05x)

[3.1 Basic concepts and definitions 3](#_y0ezzmfgfvse)

[3.2 Task-Oriented Approach 4](#_bxv88kqxng88)

[3.3 Utility-Based System 5](#_j2kdk594nszt)

[3.4 Task Completion and Interruption 5](#_t3hwv2ah60py)

[3.5 The concept of resources 6](#_1pmsto31ejxx)

[3.6 Extensibility and Customization 6](#_25pb1potxnf9)

[**4. Built-in activities 6**](#_h11l3ula75ks)

[4.1 Activity 6](#_kqmu1cltebp0)

[4.2 Idle Activity 7](#_pags92902tqi)

[4.3 Work Activity 7](#_1tqog3t2a1dc)

[4.4 Recovery Activity 8](#_2qyjjr22oitg)

[4.5 Bring Object Activity 9](#_dzfh5jkv7tp)

[4.6 Attention Activity 11](#_ju3pymkr6m55)

[**5. Scene objects 12**](#_yq08a9rgo6jc)

[5.1 Agent 12](#_y8mawxlkuuxb)

[5.2 Portable Object 12](#_lwpy94ijqo1)

[5.3 Warehouse Object 12](#_skevyxuauxfd)

[**6. How to setup scene 13**](#_qpchljqvofgg)

[**7. Technical details 14**](#_bge8b4tnx2rj)

[7.1 How does the system work? 14](#_gucizva4tssl)

[7.2 Creating custom activities 15](#_sdzf65x10b4y)

[7.3 How to create activity with multiple agents? 15](#_oeauxij2q3oe)

[7.4 Ways to start task 16](#_wdsdw4izuxj2)

[7.5 How to stop a task? 17](#_sibdivg8unza)

[7.6 How to change the set of captured resources? 17](#_h1hjdlqye1ax)

# 

# 1. Introduction

## 1.1 Overview

**AgentAI** is a **task-oriented**, **utility-based** artificial intelligence system specifically designed for the Unity game engine. It empowers game developers to create intelligent agents that can make decisions based on their assigned tasks and overall goals.

AgentAI is well suited for simulating the **non-deterministic behavior** of a large number of agents with a large number of different tasks. AgentAI algorithms allow you to effectively distribute tasks between agents, minimizing useless actions, and optimizing the speed and efficiency of task completion.

## **1.2 Key Features**

* Task-Oriented Approach: Agents in AgentAI focus on completing tasks, allowing for organized and goal-driven behavior.
* A smart system for distributing resources and tasks between agents.
* Utility-Based System: The AI system employs utility functions to evaluate task priorities and make optimal decisions.
* Simplicity. Typically, to create a new activity, you only need to assign a script and set up a few fields in it. Nothing more needs to be done. And the activity "just works".
* Extensibility: Developers can expand AgentAI's functionality through custom tasks.
* Seamless Integration: AgentAI seamlessly integrates into the Unity engine, simplifying the development process.
* Efficient Performance: The system is designed to offer high performance, even in complex scenarios.

## 1.3 System Requirements

Unity Engine (Version 2022.3.5 or later)

Platform: Any (Windows, macOS, WebGL, Android or Linux)

# 2. Getting Started

## 2.1 Installation

AgentAI does not require installation - just move the AgentAI folder into your project. AgentAI does not require any third-party libraries.

Inside you can find the Demo folder which contains the demo scene.

After examining the demo scene, you can delete the Demo folder.

## 2.2 Setting up AgentAI in Unity

The library does not contain any configuration files. Once the AgentAI package is imported, you're ready to create intelligent agents for your game. The next sections will guide you through the process.

# 3. Understanding AgentAI

## 3.1 Basic concepts and definitions

* **Activity** The work to be done by the agents. Activities generate tasks for agents. For Example: WorkActivity, IdleActivity.
* **Agent** An active character that moves around the scene and can perform activities. The agent has a list of **parameters** indicating its current state.
* **Resource** A scene object that can be captured by exactly one task. Resources are agents, activities, and other objects such as PortableObject. A resource cannot be owned by multiple tasks at the same time.
* **Task** A task is an activity associated with an agent. A task is created when an agent is assigned to an activity and executes it. At one point in time, an agent can have only one task, or not have at all. Also, the task captures resources while the task is active, the captured resources cannot be used by other tasks.  
  Activity can have one or several tasks associated with it - one task for each agent assigned to the activity.  
  A task is an intrasystem object and is not bound to any game objects on the scene.
* **Portable Object** A scene object that can be moved by an agent.
* **Warehouse** A scene object that can generate new Portable Objects.

## 3.2 Task-Oriented Approach

AgentAI focuses on completing tasks assigned to each agent. Each task represents a specific goal or objective the agent aims to achieve.

In AgentAI, the agent is only the executor of activities, and does not have his own desires and actions.

Any actions of the agent are determined by activities on the scene.

Activities, in turn, compete for agents and strive to find agents to own execution.

For example, if there is a control panel for which you need to work, then an activity is attached to the control panel, which itself will look for agents for its execution. The agent itself does not have any actions of its own.

Another example. If you need the agent to sleep, then you need to put a bed object, attach an activity script to it. And then the bed itself will look for agents with low stamina, assign them to itself and thus make the agent sleep.

The priority of activities is determined by the importance parameter and also has a special calculation algorithm.

## 3.3 Utility-Based System

AgentAI employs a utility-based decision-making system. It calculates utility values for activities based on **importance** parameters (defined by user) and various factors. AgentAI selects the most suitable task for the agent to execute.

The utility calculation algorithm depends on the specific implementation of the activity. But often the importance of the activity depends on the parameters of the agent or the parameters of other objects.

For example, if an agent's stamina drops, then the importance of sleep activity increases.

Also, the importance depends on the state of the game object. If it is disabled, or the activity script is disabled, its importance drops to zero and the task is canceled.

Similarly, the importance of a task drops to zero if the agent becomes inactive. This causes the activity to be detached from the agent.

## 3.4 Task Completion and Interruption

Most tasks do not end automatically. They work until a higher priority task appears, or until the current task becomes a lower priority.

How then are tasks completed? Why don't they run indefinitely?

Usually this is achieved by the fact that during execution, the task changes the parameters of the agent or the parameters of other objects. This causes the importance of the current task to decrease and thus, sooner or later, the current task will be replaced by another task. In the worst case - will be changed to Idle activity, which is usually the lowest priority task (but greater than zero).

Tasks are executed automatically according to their priority. This means that any task can be stopped at any moment and the agent can switch to a higher priority task.

For example, if an agent performs the task of carrying a box from one point to another. And if at that moment a higher priority task is assigned to it, then the current task will be interrupted, the box will be put on the ground and another task will begin. This can happen while the agent is walking towards the box and while he is already carrying the box.

## 3.5 The concept of resources

An important concept of the system is resources.

A resource is an object that can only be used by one task at a time. Or it can be free, and not attached to any task.

Agents, scene’s objects like PortableObject, some activities - they are all resources.

This means that, for example, an agent, as a resource, can only be attached to one task at a time. The same goes for activities like WorkActivities, which can also only be used by one task at a time.

The system automatically monitors resource usage. If a resource is captured by a task, then other tasks will no longer be able to capture this resource while the task is executing.

And vice versa, if the task ends, then all the resources that it used are released.

## 3.6 Extensibility and Customization

AgentAI is designed to be extensible, allowing developers to create custom activities and introduce additional functionality tailored to their game's requirements. To implement your own type of activity you need to inherit the Activity class.

# 4. Built-in activities

The built-in implementations of various activities are described below. Despite their small number, they cover a larger number of practical needs.

## 4.1 Activity

Activity is the base abstract class for all activities. You cannot create it on the scene, but its properties are inherited by all activities.

The class contains the following properties:

**float Importance** - The importance of the activity. The higher this number, the more likely that the agent will be assigned to this task. A value between 0 and 1 is recommended. A value of 0 actually means that the activity will never run.

**float AddImportance** - Additional importance if the agent is already running this activity. This is necessary so that the agent strives to complete the current task to the end and there are no abrupt transitions from one task to another.

For example, if an agent is sleeping, then his stamina increases. Accordingly, the importance of the task of sleep is falling. If there is no additional importance, then immediately after the start of sleep, the importance of sleep will decrease and the agent will immediately switch to other tasks.

**Transform Seat** - Indicates the exact position and rotation of the agent when it arrived to execute the task. If not specified, the transform of the object to which the activity is attached is used.

**bool CanBeAttached** - If this activity is being used as a resource, then this flag enables or disables the use of the activity by tasks.

## 4.2 Idle Activity

The simplest type of behavior. The assigned agents simply walk a random route around the activity object.

Idle usually has the lowest priority. The recommended importance value is 0.001, the additional importance is 0.001.

IdleActivity is one of activities that can have multiple assigned agents.

## 4.3 Work Activity

The most common type of activity, which consists in the fact that the agent must arrive at the point of activity and stay motionless in it, simulating work.

Work behind the machine, installing equipment, extracting resources, working at the control panel are examples of WorkActivity.

WorkActivity does not allow multiple agents. Only one agent can be attached to the WorkActivity.

**float Workload = 1** - "Amount of work" to be done. This parameter determines how long the job will take to run. Typically, this parameter decreases while the agent is working and increases if no one is working on the activity. Usually this value is between 0 and 1.

**float SpeedDown = -0.1f** - Reducing of Workload (per second) when an agent performs an activity. It must be less than zero.

**float SpeedUp = 0.1f** - Increasing of Workload (per second) if no one is working on the activity. Set it to zero if you want one-time work. Set a greater zero if you want periodic work.

**float WorkloadToActivate = 1f** - Minimal Workload value when activity starts to find agents to hire. The WorkActivity does not hire agents if Workload is less than this value.

**ParamCondition[] AgentParamConditions** - A list of minimum agent parameter values so that an agent can be hired for this activity. This allows you to filter agents by their type, params, and status.

**ParamChanger[] AgentParamAffects** - List of agent parameters that will change during the task execution. It only changes when the agent is already in the workplace.

## 4.4 Recovery Activity

This activity is aimed at restoring the agent's parameters. If WorkActivity is aimed at performing work, then RecoveryActivity is aimed at restoring the parameters of the agent itself.

Changing agent parameters is done through the **AgentParamAffects** property of the base class.

For example: sleep, treatment, rest, communication - are implemented with the help of RecoveryActivity.

RecoveryActivity does not allow multiple agents. Only one agent can be attached to the RecoveryActivity.

**float MinUtilityToStart** - The minimum utility value for agent parameters (calculated via AgentParamScores) at which the activity tries to hire an agent for execution.

**ParamScore[] AgentParamScores** - A set of agent parameters on the basis of which the utility (that is, importance) for the activity is calculated.

**ParamCondition[] AgentParamConditions** - A list of minimum agent parameter values so that an agent can be hired for this activity. This allows you to filter agents by their type, params, and status.

**ParamChanger[] AgentParamAffects** - List of agent parameters that will change during the task execution. It only changes when the agent is already in the workplace.

## 4.5 Bring Object Activity

This action is that the agent needs to take the object (PortableObject) and transfer it to another location. The PortableObject may already be in the scene, or alternatively the agent may create a new object in the Warehouse Object.

In addition to the function of transferring objects, this activity can also be considered as an activity similar to WorkActivity, but "working with a tool". That is, this activity is useful for work that requires a tool or resource.

Note, the BringObjectActivity can use PortableObject with Charge value greater than zero only.

Activity usage examples: buying items in a warehouse and installing them in a specified place, blowing up objects with dynamite, collecting garbage on a scene and throwing it in a trash can.

BringObjectActivity can have one or many assigned agents (it is specified in property).

**string ObjectType = "Box"** - The type of object to bring. This string type is specified in the PortableObject. The ObjectType can be empty. It means that the activity allows bringing a PortableObject of any type.

**ObjectSource ObjectSource** - Source of getting the object. This can be an existing object in the scene, or a warehouse, or both sources are valid.

If Any is selected, then you can bring both objects on the scene and generated objects from the Warehouse object. First of all, objects that are already on the scene will be brought.

**bool AllowMultipleAgents = false** - Can I assign multiple agents to a task? If true, then many agents will be able to bring objects at the same time.

**float ChangeCharge = -1f** - By how much will the Charge in the PortableObject change after the object is brought?

The combination of the Charge values from PortableObject and ChangeCharge allows you to create objects that can be used multiple times.

For example, if Charge = 5 and ChangeCharge = -1, this means that the object can be used 5 times before its Charge becomes zero and can no longer be used.

**DestroyObjectStrategy DestroyObject** - Manner of destroying after the PortableObject was brought.

Possible values: **DoNotDestroy** - do not destroy PortableObject, just drop it on the ground. **IfZeroCharge** - Destroy the object if Charge becomes zero. **IfCompleted** - destroy object after it was brought, do not take to attention Charge value.

**ParamCondition[] AgentParamConditions** - A list of minimum agent parameter values so that an agent can be hired for this activity. This allows you to filter agents by their type, params, and status.

**ParamChanger[] AgentParamAffects** - List of agent parameters that will change when the task is completed (once).

**UnityEvent OnTaskCompleted** - The event occurs after PortableObject was brought.

## 4.6 Attention Activity

This activity allows agents to stop and approach an activity point on the scene, look at objects some time, and then move on.

This activity uses OnTriggerEnter event, so gameobject of the activity must have a collider in trigger mode. Rigidbody is also needed either for agents or for activity.

The activity fires as soon as the agent enters the trigger area.

**float MinUtilityToStart = 0.5f** - The minimum utility value for agent parameters (calculated via AgentParamScores) at which the activity tries to hire an agent for execution.

**float Duration = 2f** - The time during which the agent will stand near the point of activity (seconds).

**ParamScore[] AgentParamScores** - A set of agent parameters on the basis of which the utility for the activity is calculated.

**ParamCondition[] AgentParamConditions** - A list of minimum agent parameter values so that an agent can be hired for this activity. This allows you to filter agents by their type, params, and status.

**ParamChanger[] AgentParamAffects** - List of agent parameters that will change when the task is completed (once).

# 5. Scene objects

## 5.1 Agent

Agent is a scene object that performs activities.

**bool CanBeAttached** - Enable or disable the ability to hire yourself for tasks.

**List<Param> Params** - A set of parameters that characterize the current state, status or type of the agent.

**ParamChanger[] AutoAffectedParams** - List of agent parameters that will change during agent life automatically.

## 5.2 Portable Object

PortableObject is an object that can be carried by an agent from place to place. The PortableObject may already exist on the stage, or it may be generated by a special WarehouseObject.

Usually it is used in pairs with BringObjectActivity.

**bool CanBeAttached** - Enable or disable the ability to use for tasks.

**string Type = "Box"** - Type of the object.

**float Charge = 1** - The "Amount" of the object. For example, if it is cartridges, then this can be the number of cartridges. If it's gas, it could be volume, and so on.

## 5.3 Warehouse Object

The warehouse is a scene object that can generate PortableObjects.

**bool CanBeAttached** - Enable or disable the ability to use for tasks.

**float GeneratingTime = 1** - Time to generate an object (seconds).

**PortableObj[] ProvidedObjects** - Prefabs of objects that the warehouse can generate.

**ParamChanger[] AgentParamAffects** - List of agent parameters that will change when the object is created (once).

**Transform Seat** - Indicates the exact position and rotation of the agent when it arrived in place. If not specified, the transform of the object is used.

# 

# 6. How to setup scene

You can watch the demo scene to see how it all works.

Also you can use prefabs from folder Demo/Prefabs where presented examples of scene objects with different activities.

The most typical scenario of how to create a new scene from scratch is described below.

* Place an empty object on the stage, attach a **TaskManager** to it. This is a necessary object that manages all tasks on the stage.
* Create (or use a prefab) an agent object. Attach the **Agent** script to it.
* Generate a **NavMeshSurface** for your scene. This will allow the agents to move around the scene.
* Create an object (or use a prefab) with the **IdleActivity** script.

After that, you can run the scene and see that all agents perform the IdleActivity activity - they move to arbitrary points around the scene.

You probably want more complex behavior. For example, you might want an agent to do some work. For this:

* Create an object on the stage and attach the **WorkActivity** script to it.

Run the scene. You will see that the agent from time to time comes to the place of work and is there, “working”.

Более сложные типы поведения можете увидеть в демо сцене.

Note:

The system automatically keeps track of all agents, activities and other resource objects on the scene.

Therefore, you can freely create activities, agents and resources on the scene. The system will track their occurrences and use the provided objects.

You can also remove or disable objects in the scene. The system will stop using them automatically. If a task is associated with the object, the task will be automatically stopped before the object is disabled.

# 

# 7. Technical details

## 7.1 How does the system work?

Singleton object **TaskManager** сontains lists of agents, resources and activities. It also orders tasks by priority, starts and stops tasks, and deals with the distribution of resources between tasks.

The main purpose of activities is to generate tasks. To do this, they implement method **IEnumerable<Task> GetTaskCandidates()**. This method is sometimes called by TaskManager and should return tasks that the activity would like to run. Typically, the activity returns a tasks of all possible combinations of activity and agents. The list of agents is taken from TaskManager.Agents.

Usually, an activity returns tasks of the appropriate type (but this is not required). For example, the WorkActivity generates tasks of the WorkTask type.

The TaskManager considers the declared candidate tasks, calculates their importance, the employment of the withered resources, free agents, and decides whether to launch the tasks or not.

The task calls the base constructor, which passes a list of resources that this task would like to capture. For example, a WorkTask may want to use two resources - an WorkActivity (because this script attached to workplace) and an Agent to work with.

When a task actually starts, these resources are attached to the task and cannot be captured by other tasks.

The task class contains two important methods you should override - **float CalcImportance()** and **IEnumerator OnUpdate()**.

* **CalcImportance()** calculates current importance of the task. Typical range of importance - from 0 to 1.
* **IEnumerator OnUpdate()** - This method contains code for managing the agent during the execution of the task. This is a lazy IEnumerator method, so it can run indefinitely. If the method ends, the task is automatically stopped too.

## 7.2 Creating custom activities

To create your activity, you need to create a class that inherits from activity and a class that inherits from task. For example FooActivity and FooTask.

FooActivity should generate tasks of type FooTask.

The FooTask contains method OnUpdate() that will control the Agent and task execution overall.

You can see examples of implementation in built-in activities. For example see WorkActivity as a typical case.

## 7.3 How to create activity with multiple agents?

Some activities use many agents for their execution. For example IdleActivity.

In order for your activity to be able to use multiple agents, you need the task to be unable to capture the activity as a resource. Because if the activity is captured, the system will no longer allow you to create another task for this activity.

To prevent capture you can do:

* Do not pass activity as resource to Task base constructor. It is the best approach. For example, IdleActivity does this.
* Pass activity to Task constructor, but at same time implement interface IResource and implement method IResource.AttachTask. Assign null to AttachedTask there:   
  void IResource.AttachTask(Task task) => AttachedTask = null;

It is a more flexible approach, because you can allow users to adjust to make the activity multiple or not in activity properties. For example, BringObjectActivity does this.

## 7.4 Ways to start task

In general, the system works in such a way that tasks start and stop automatically.

But sometimes you might want to force the task to start, or the standard approach won't work for you.

Generally there are three ways to start a task:

* Usage of method **GetTaskCandidates()** of Activity. This method is called automatically from TaskManager when it wants to get task candidates from your activity. It is a standard approach.
* Use the method **StartTask(Task task)** of TaskManager. This method starts a task, captures resources for the task, stops tasks that lost resources.

This method forcibly starts a task, regardless of the priorities of other tasks. But if the priority of a given task is low, it can be stopped and replaced by higher priority tasks in the future automatically.

* Use the method **ProposeTask(Task task)** of TaskManager. This method proposes a task to execute. This request will be considered in the next frames and the task will be started or will be rejected. You can use this method to gracefully run a task once, respecting the priority of other tasks. This method does not guarantee task execution. This approach is used in AttentionActivity for example.

## 7.5 How to stop a task?

You can stop a task at any moment by calling the method **StopTask(Task task)** of TaskManager. It will stop the task forcibly, attached to the task resources will be released.

This can be useful to immediately release the resources occupied by a task before starting another task.

Note also that the termination of the OnUpdate() method and the destruction or disabling of any resource associated with the task causes the task to stop.

Also the Task will stop if any unhandled exceptions appear in any task methods, for example in CalcImportance() or OnUpdate().

## 7**.6 How to change the set of captured resources?**

Sometimes it becomes necessary to change the resources captured by a task during its execution.

For example, in the BringObjectActivity activity, you first need to go to the warehouse, then take the object and take it to the activity location.

And there are two troubles:

1. The object is created by the warehouse and it needs to be captured (as resource) after the task has started.

2. After the object is received in the warehouse, it is advisable to free the warehouse resource so that other agents can use it while we carry the box.

To replace the resources of an already created and active task, you can use the **AttachResource(IResource resource)** and **DetachResource(IResource resource)** methods in the Task class.

When calling AttachResource(IResource resource), the task associated with this resource will be stopped (if it was attached to any task), and the resource will be captured by the current task.

Congratulations! You are now equipped with the essential knowledge to start using AgentAI a powerful task-oriented, utility-based AI system for Unity. Happy game development!