# Actions

XmasActions in the UML Domain model diagram refers to a the object type that performs actions inside the engine, the reasoning behind these actions being its own class is to ensure only one action at a time is being performed this is because there are many separate threads operating on the model code at once and as such there must be a way to active only one action at a time.

For the task of executing the actions we have the ActionManager its job is to take in one action at a time and place them in a queue.



The idea is that a controller, such as keyboard input or an Agent language queues an action such as moving an entity on the action manager, the action manager will then execute the action as soon as it is ready.

## Action Types

The engine is featured with two different action types, one is an Environment Action, and this action type is meant to reflect actions that do changes to the entire environment. The can be actions such as closing this engine or adding/removing entities from the world.

The other action type is an Entity Action this action type is meant as an action that a single entity performs; ideally the actions should be as atomic as possible. In our reference implementation we have given some ideas how these actions work such as grab which is an action that grabs a package from below an agent.

## Example

### Entity Action – Move entity

To give an idea of how an entity action is made by inheriting the EntityAction class. As shown in this uml we can see we have created a MoveEntityAction class with one field containing the direction of the move.



To make so the action performs something it is required that an abstract method Execute is implemented this execute method is the method that is executed by the action manager.

The implementation of the execute action could then look something like this

#### Pseudo code

method Execute returns nothing

NewPosition = GetPositionOf(World, This.getSource()) + Direction

Wait(MOVE\_TIME)

SetPositionOf(World,This.getSource(), NewPosition)

Endmethod

As one can see the idea would be that you find the position of the source of the EntityAction and use that to generate a new position which is its old position incremented by its direction vector. The wait is there to give the move a speed otherwise it would be an instant movement.

## Summerize

The idea of actions is fairly simple to use and it is there to hide the user of the engine from a lot of hazel that happens behind the scene. It is meant to ensure thread safety and allow multiple threads working with the engine at once. The reasoning why we chose this design was for these exact reasons as we had to deal with the problem of interference from multiple concurrent threads. Furthermore it also is meant to reduce code redundancy as generic actions can be reused by other actions. The problem with this design is that it in a sense remakes what is already implemented in a programming language. As running procedural code is what programming languages are meant to do. However in return it provides a lot of utility and in the sense that it makes it possible to make tools for simplifying the process of making actions. As well it gives the ability to differentiate between different action types and even create new action type if one wishes to.