Implementation – Actions

## Introduction

As we already went through what actions can be used this section will keep that to a minimum and instead focus on our idea behind, and how we implemented them. It will furthermore cover the entire life span of an action object.

## Explanation

An action or as called in our engine XmasAction is a class which provides an API for performing state changes inside the engine, while also ensuring that only one action at a time is being executed.

UML Sequence diagram

As we can see it starts with the XmasModel running an endless loop that tells the ActionManager to execute all newly queued actions. The ActionManager then takes all the actions from a threadsafe list and places them in a local list. After which each action is executed individually, putting the action that is being executed in a running state, the state will not change before the actions “completed” method is called. Once an action has been properly executed it will be changed to a completed state will be properly disposed. When the last action has been executed by the ActionManager, the call to the ActionManager is returned and XmasModel will put the thread in a waiting state. The XmasModel will remain in wait state until a new action has been placed on the queue; this is preventing it from busy waiting when no actions are to be executed.

This might seem tedious having to call a special method on each action however it is quite necessary as the completion of the execute method call does not guarantee that a method is completed.