Architecture

The section will cover how all the components of the engine interacts with one another, it will detail how flow of information is transferred through the engine and into the components connected to it. The component diagram of the Xmas Engine can be found on appendix **[INSERT APPENDIX NUMBER].**

The components of the engine is

* Model
* World Creation
* View
* Controller

# Model Component

**Requires: ##XmasWorld, ##XmasAction and ##Trigger**

**Provides: ##Percept**

The model component is responsible for handling internal interactions of the engine. These interactions are based on which ##XmasAction it is given.

The model component has three requirements, these requirements are necessary for the model component to properly execute the environment requested by the user.

The first requirement of the model component is the ##XmasWorld, the model component uses the ##XmasWorld by giving it to ##ActionManager, the ##ActionManager then gives the ##XmasWorld to all ##XmasActions as they are about to be executed. Thus all ##XmasAction executed in the engine use the same ##XmasWorld.

The second requirement is the ##XmasActions, all ##XmasActions queued on the model component are executed by the ##ActionManager. An ##XmasAction are not executed immediately however as they wait until all prior ##XmasAction’s executions has been completed. Once queued to the ##ActionManager, they are provided with all necessary dependencies such as the ##XmasWorld and the ##EventManager.

##XmasAction is designed to allow other threads the ability to interact with the engine. The reason is that we did not wish for multiple threads to change the state of the model component at once, to ensure this was never necessary we provide the abiltity to inject code into the thread used by the ModelComponent, this code is transferred in the form of an ##XmasAction.

The last requirement of the model component is ##Trigger, the model component takes any number of triggers and inserts them in the ##EventManager. When an ##XmasAction raise’s an ##XmasEvent on the ##EventManager, the ##Triggers that are registered to that ##XmasEvent are all triggered.

The only thing that the model component provides is the ##Percept, each ##Percept is something that an agent can sense. An ##AgentController connected to the model component can receive these ##Percepts which it is meant to use for analyzing the agent’s next move.

The model component is made of many classes however the three ##XmasModel, ##EventManager and ##ActionManager are what provide the core features of the model component and as such is the only ones shown in the diagram. When going into details of the exact design of the engine it will be evident that the class ##XmasEntity also provides some of the features of both ##EventManager and ##ActionManager, however it only does this to make the feel of using the engine more natural. For instance when moving an entity we thought that it would make sense that the code for this was ##Entity.QueueAction(new Move()), instead of ##ActionManager.QueueAction(new Move(EntityToBeMoved)). In actuality the code does the same thing since in the first case: All the ##Entity does is to call the ##ActionManager in the same way we just showed and then use itself in place of the ##EntityToBeMoved. This is the reason why the ##Entity is not shown in the model component as it has no relevance when understanding the component itself.

# World Creation Component

***Provides: ##XmasWorld***

The world creation component is responsible for making a world for the engine’s entities to inhabit. The world is created when the engine starts to execute, as such its internal class ##WorldBuilder only contains a blue print for which entities it should construct and not the actual entities themselves. It does this by storing a function for each entity, those functions contains the information on how each of the entities should be constructed.

The user of the engine is meant to implement his own ##WorldBuilder class, that implementation should contain knowledge on how the world he wishes to construct is created. That means if for instance wants to use a Tile based world then his implementation of ##WorldBuilder should construct a tile based world.

# View Component

**Provides: ##Trigger**

The view component is meant as the component that visualizes the model of the engine to the user, it does not enforce how the visualization is done or in which way the visualization occurs. It only provides the tools necessary to perform this task.

The view is meant to register ##Triggers on the model component, these ##Triggers contains ##XmasEvents when an ##XmasEvent is raised, the ##Triggers with those ##XmasEvents are triggered. The idea is that when a ##Trigger is triggered that means the current state of the model component has changed, the view uses these ##Triggers to be informed about such changes, and are thus able to change its own state in responds correctly making it able to visualize the new model state.

# Controller Component

**Requires: ##Percept  
Provides: ##XmasAction**

The controller component’s responsibility is to command ##Agents to perform actions inside the world. The controller component does this by making the ##AgentController send ##XmasAction objects to a specific ##Agent in the model component. Where upon that ##Agent will perform said ##XmasAction, once the model component has executed all prior actions.

The controller component also has ability to receive ##Percept objects back from the engine, these ##Percept contain data about what the ##Agent it is controlling has sensed. These ##Percepts are meant to be analyzed by the controller component to determine what its next ##XmasAction should be.

The controller component is made up of abstract classes which the user of the engine must first implement; these implementations could be setup to act as an interface between a single APL and our engine. This means that for each APL one must make a new implementation of the controller component. To reduce the burden of the user we will in our extensions provide the ability to interface with EIS supported APLs.

Furthermore the controller component is not only designed to make interfacing with different APLs easier, it is also meant to be used when making an interface between the user and the model component. For instance if one wished to control an agent with the keyboard, then an Keyboard implementation of the AgentController and AgentManager should be made, where it would be possible to bind the queuing of move actions to specific buttons on the keyboard.

# Summary

The architecture of the engine shows the connectivity between each of the components. The Model component which job it is to ensure proper interactions occur inside the world. The world which is constructed by the World Creation Component, meant to be designed along with the world itself.

The interactions of the model component are provides by the controller component which task it is to command the agents inside the engine, and make it so they are given intelligence. And lastly the view component which only task is to visualize the state of the engine.