## 

**Memoria del proyecto práctico (2ª convocatoria)**

**Diseño y Mantenimiento del Software**

Motor de ficción interactiva

Alumno:

Alberto Uriarte Martínez

Contenido

[Diagrama de clases 3](#_Toc504521208)

[Patrones utilizados 4](#_Toc504521209)

[Singleton 4](#_Toc504521210)

[Intención 4](#_Toc504521211)

[Participantes 4](#_Toc504521212)

[Comando 4](#_Toc504521213)

[Intención 4](#_Toc504521214)

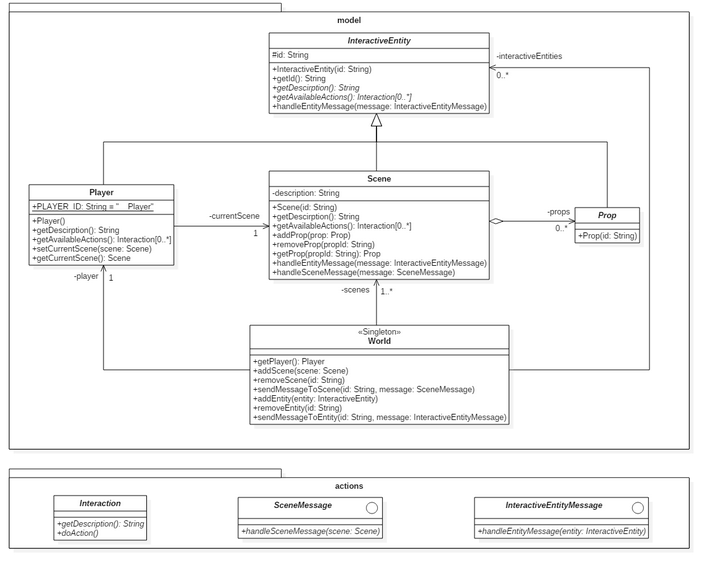
[Participantes 4](#_Toc504521215)

[Mediador 5](#_Toc504521216)

[Intención 5](#_Toc504521217)

[Participantes 5](#_Toc504521218)

# Diagrama de clases



# Patrones utilizados

En el desarrollo de este proyecto se han utilizado los siguientes patrones: Comando, Mediador y Singleton.

## Singleton

### Intención

Se ha utilizado este patrón para garantizar que la clase World tenga solo una instancia, y para que cada acción pueda tener un acceso fácil a él si es necesario enviar mensajes a otras entidades.

### Participantes

La clase World define una operación getInstance(), estática (accesible globalmente y sin necesidad de una instancia), para obtener la única instancia de la clase.

## Comando

### Intención

Usamos como comando abstracto la interfaz Interaction.

Esta interfaz declara un método doAction() abstracto que servirá para ejecutar la acción que encapsulan. Más tarde cada comando concreto implementará este método según convenga.

### Participantes

#### Comando

La interfaz Interaction, que declara la operación a realizar.

#### Comando Concreto

Las clases DescribirEscena, ExaminarCaja, Open y SwitchSceneAction. Todas ellas implementan la interfaz Interaction y le dan una implementación propia al método doAction().

#### Cliente

Interactúa con el subsistema a través de la clase Aplicación.

#### Invocador

La clase World es la encargada de transmitir los eventos cuando se produce una petición.

#### Receptor

Los diferentes concreteprops como Caja, Door, Lock y también SceneGame, son los encargados de llevar a cabo las diferentes operaciones.

## Mediador

### Intención

Coordinará los diferentes eventos en la historia. Esto permite que interactuar con un elemento pueda afectar a otro en una escena distinta. Encapsula cómo interactúan varios objetos reduciendo su acoplamiento.

### Participantes

#### Mediador Concreto

Será nuestra clase World.

Mantiene internamente una referencia a las entidades interactivas que vayan a poder recibir mensajes.

Coordina los eventos mandándolos a los diferentes colegas a través de los métodos sendMessageToEntity() o sendMessageToScene(), dependiendo si se trata de una escena o un elemento interactivo.

#### Colega Concreto

Serán nuestras clases pertenecientes a concreteprops(Caja, Lock, Door) y SceneGame. Que en nuestro caso serán las clases que pueden ser modificadas dinámicamente.

Estas clases reciben los eventos del Mediador mediante los métodos handleSceneMessage() y handleEntityMessage(), dependiendo si se trata de una escena o una entidad concreta.