**Introducción**

***Sobre este documento***

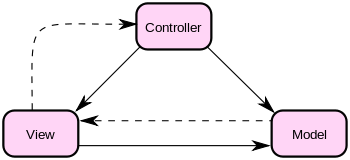
En esta sección se describe la arquitectura del sistema “Walkers of the city”, juego multijugador en línea desarrollado como trabajo práctico final de las materias Programación Concurrente e Ingeniería del Software. Se provee una descripción de alto nivel del sistema, mostrando los subsistemas que lo integran y los componentes de cada subsistema mediante diferentes diagramas. Se define un patrón de organización para la estructura del sistema y a partir de allí se continúa con el diseño de la arquitectura basado en esa estructura.

**Patrón de diseño**

***Descripción del patrón elegido***

El patrón de diseño elegido es el MVC (Model View Controller), un modelo organizacional compuesto por 3 capas que se relacionan entre sí. Cada capa tiene una responsabilidad bien definida, y el objetivo de las mismas es separar la parte la lógica, los datos y la presentación.

El modelo general respeta el siguiente esquema:



Se distinguen 3 subsistemas en el diagrama anterior. Cada uno, como se aclaro arriba, cumple con una función bien definida.

* Model (modelo): Este subsistema mantiene el estado de la aplicación. Contiene los datos principales del sistema con sus valores actualizados.
* Controller (controlador): Este subsistema responde a los eventos disparados desde la vista y se ocupa de toda la lógica que se le aplica a los datos. Es el encargado de cambiar el estado del modelo. También puede hacerle peticiones a la vista.
* View (vista): Este subsistema responde a los eventos disparados por el usuario. Se ocupa de la presentación del modelo en un formato adecuado para interactuar con el usuario.

***Justificación de la elección***

Dados los requerimientos obtenidos en la sección B de este proyecto, necesitamos un sistema en donde el estado (los datos y sus valores) de una partida específica esté compartido y sea accesible por todos los jugadores participantes en ella, ya que los mismos se mueven sobre el mismo mapa, todos deben poder ver el resultado de los demás, etc. Esto implica que la información necesariamente debe estar centralizada y todos los jugadores deben poder acceder a ella en el momento que lo requieran. A su vez existe toda una lógica de control, que define el desarrollo del juego interactuando con el estado de la partida, conoce el formato de los datos y sabe tratar con ellos. Esta lógica es la misma para todos los jugadores.

Por otra parte el estado de una partida debe cambiar cuando los jugadores interactúan con el juego a través de una interfaz, pero toda interacción de los jugadores con los datos debe respetar la lógica del juego.

Es factible pensar que se podría centralizar toda la lógica de control del juego como un subsistema aparte, ya que su existencia es claramente necesaria, todos los jugadores utilizan la misma lógica y no forma parte de los datos, solo se remite a interactuar con los mismos. Luego, bajo este esquema, toda interacción jugador/interfaz debe disparar ciertos eventos en la lógica de control que cambien de manera apropiada los datos de la partida.

Por último, sabemos por lo dicho arriba (y por los requerimientos) que todos los jugadores deben poder acceder al estado de la partida en la cual participan, en el momento que lo requieran. Esto trae aparejado dos consecuencias: una es que debe existir una forma de mostrar este estado al usuario de manera apropiada, es decir debe existir una interfaz con la cual el jugador pueda interactuar; la otra consecuencia es que para acceder al estado no es necesario ninguna lógica intermedia, ya que solo se requiere información y no se modifica nada, por ende el acceso a los datos puede ser directo.

Al elegir un patrón de diseño para la arquitectura de nuestro sistema, vemos que se adapta a la perfección con un MCV. Los requerimientos que definen nuestro sistema nos han llevado a concluir que una estructura organizacional del mismo que respete un patrón MCV

es la indicada para encarar este desarrollo.

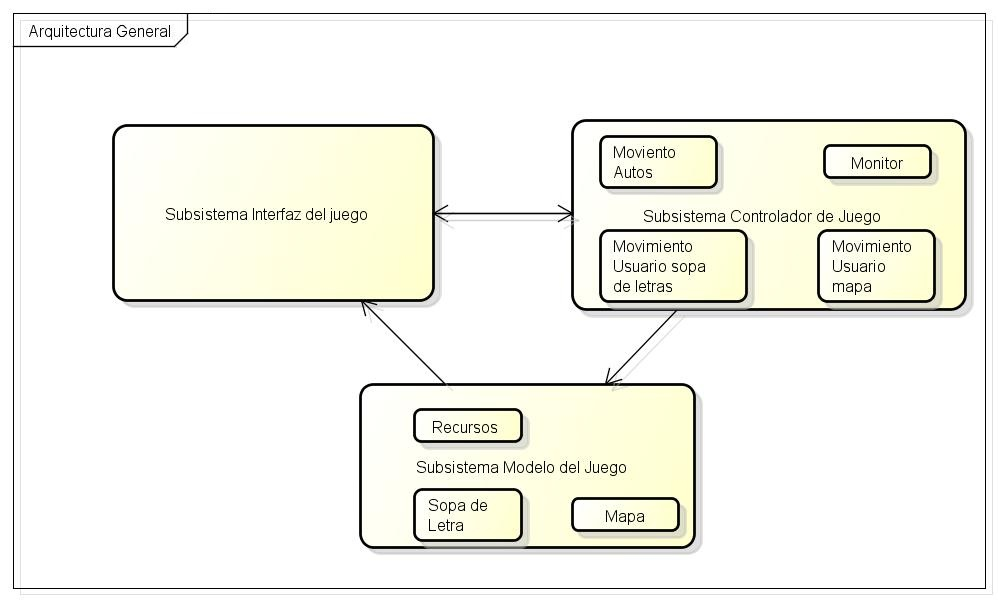
En la elección del modelo también se debatió la posibilidad de utilizar el modelo de capas ( con 3 capas - capa de presentación, control y modelo), con el cual también era posible abordar el problema. Hemos seleccionado el MCV en consecuencia de que el modelo de capas, señala que una capa presta servicios a las consiguientes, cosa que para nuestro caso, pensando en lo que se dijo anteriormente, no es necesario. Esto se ve claramente en el caso en que para mostrar la interfaz no es necesario ninguna lógica, por lo que, si se realizara con un modelo de capas, el control debería intervenir en estas situaciones, no siendo necesario. Por otro lado, si se analiza con más detenimiento, en la capa de control, se realizan diferentes procesos, pudiendo aplicar un modelo multicapa de esta misma, pero no es conveniente ni lógico ya que estos subsistemas de control, no se prestan servicios entre ellos.

El modelo elegido (MVC), además, nos permite facilitar la resolución de los requerimientos no funcionales de la siguiente forma:

* Para el requerimiento 2.1.1 (Duración máxima de una partida) al tener el controlador como una unidad aparte nos permite llevar una cuenta de todo el juego separada de cada jugador, y así podemos llevar una cuenta unificada.
* Para los requerimientos 2.1.2 (inanición) y 2.1.3 (interbloqueos), es mucho más fácil realizar esto si tenemos los componentes necesarios (como los monitores) agrupados en el mismo subsistema, en este caso en el controlador.
* Para el 2.1.6 (responsabilidad del cliente) vemos que tener el subsistema vista separado e independiente nos facilita mucho la tarea de que el cliente puede mostrar el estado del modelo actualizado y, además, la forma en que el jugador actúa sobre el sistema es más clara y ordenada al tener por un lado la interfaz o vista y por otro el controlador que efectúe la acción en el modelo.
* Para el 2.1.5 (ejecución alojada en el servidor), al tener los datos y la lógica de control independiente, es más simple alojarlo en el servidor, y que los clientes hagan solicitudes al mismo, de este modo se centraliza el funcionamiento del juego.

***Modelo de Arquitectura***

Este es el gráfico de arquitectura general en el cual nos basaremos para la realizacion del software solicitado.



***Diagrama de componentes***

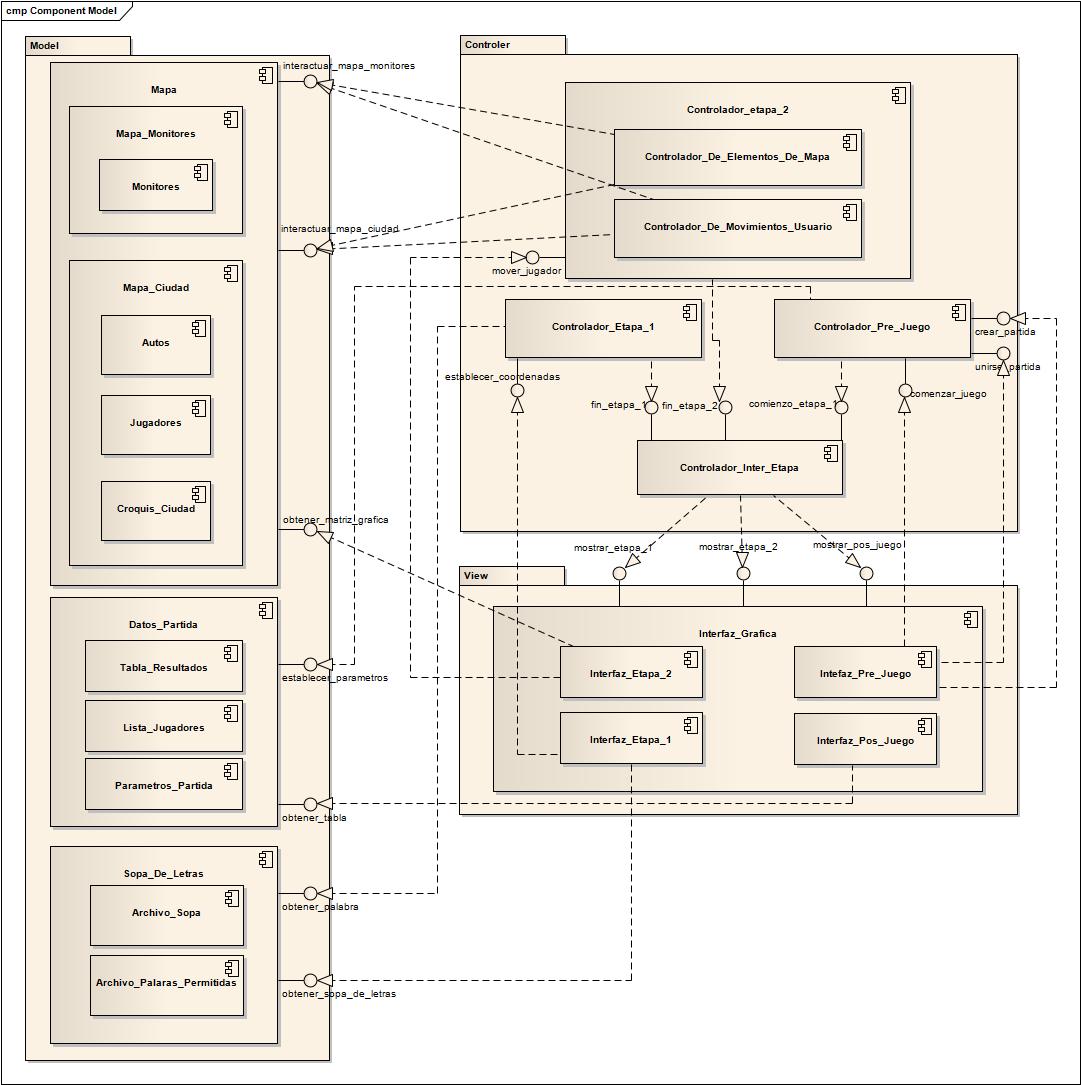


Diagrama de Desplazamiento

