****

**Ing. De Software**

**Trabajo Practico N°2**

**Profesores:**  Miceli, Martin M.

Nonino, Julián

**Alumnos:** Ramírez, Nicolás Fabián

Tolay, Antonio Facundo

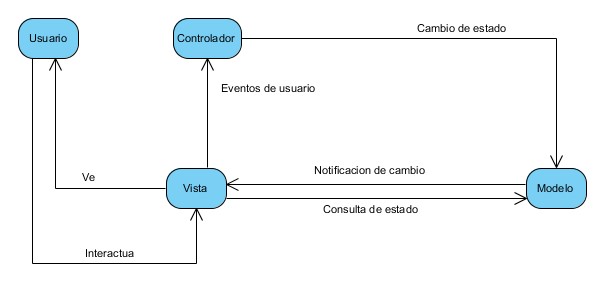
Vaira, Franco Gabriel

**Grupo:** Regedit. Exe

**Año Lectivo:** 2018

**Diagrama de arquitectura preliminar(genérico)**

Este diagrama informal muestra de manera muy general la estructura y las relaciones que existirán entre los distintos módulos que se identificaron como necesarios de acuerdo a los requerimientos y casos de usos identificados.



La arquitectura preliminar se basa en el patrón de arquitectura que especificaremos más adelante (MVC)

**Patrón de Arquitectura**

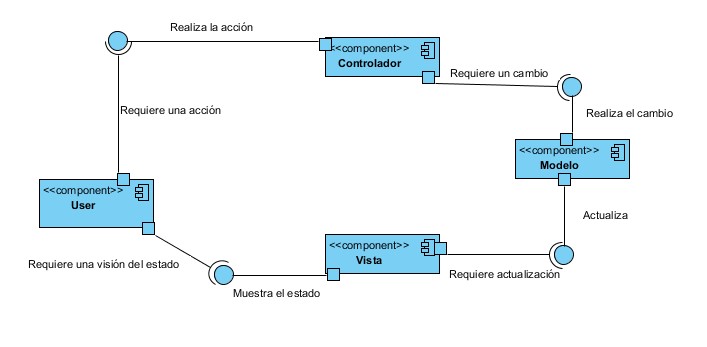
La elección del patrón de arquitectura utilizado es MVC (Model View Controller). Este se estructura en tres componentes lógicos que interactúan entre sí. El componente Modelo maneja los datos del sistema y las operaciones asociadas a esos datos. El componente Vista define y gestiona como se presentan los datos al usuario. El componente Controlador dirige la interacción del usuario (por ejemplo, teclas, oprimidas) y pasa estas interacciones a Vista y Modelo.

Las ventajas son que se permite que los datos cambien de manera independiente de su representación y viceversa. Soporta en diferentes formas la presentación de los mismos datos, y los cambios en una representación se muestran en todos ellos, pudiéndolo reutilizar en distintas circunstancias de diseño.

En nuestro caso utilizamos esta arquitectura porque el modo de interactuar con la aplicación es actuando sobre los botones que nos dicta el programa mediante el uso del mouse, es decir el controlador implementado en patrón de diseño Strategy con la vista, lo que significa, que sin importar el tipo de controlador que usemos o si incluso no existiese una instancia de este la vista seguirá mostrando el estado del modelo. Este de adapta a las acciones de usuario (reflejadas en la pantalla del dispositivo) manipulando al modelo.

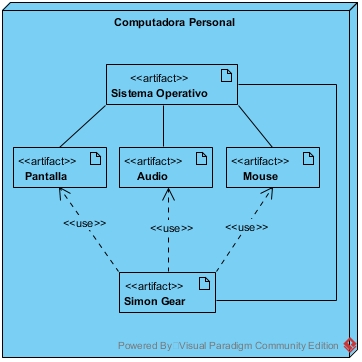
El modelo, el cual es el núcleo de la aplicación en sí, lo consideramos un módulo del cual tenemos dos versiones diferentes, un modo clásico y otro modo desafío, o sea implementando el patrón Strategy, de los cuales tendremos una sola instancia de cada uno y hacemos uso de la misma dependiendo la decisión del usuario (cambiando las instancias en tiempo de ejecución). Y a su vez, la vista es Observer de este modificándose en concordancia con los cambios del modelo.

**Diagrama de Componentes**



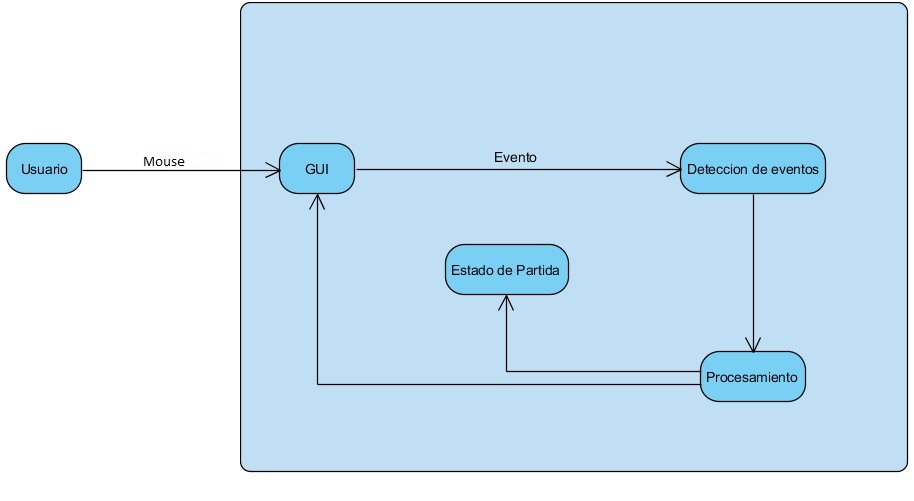
Tenemos 4 componentes principales, 3 son internos y el usuario interactúa con ellos a traces de interfaces. Al principio la vista muestra el estado inicial del modelo a través de la pantalla, con esto se espera que el usuario requiera realizar una acción sobre la pantalla la cual es tomada por el controlador, que a su vez produce un cambio en el modelo. El modelo actualiza su estado y este es consultado constantemente por la vista. Luego se repite el proceso.

**Diagrama de Despliegue**

****

Como este diagrama refleja la disposición física de los componentes, claro está que en nuestro caso solo contamos con la computadora personal donde corre la app. Dentro de este nodo interactúan 5 componentes principales, que son el Sistema Operativo de la computadora (Windows), la pantalla, el audio y el mouse de la computadora y la aplicación que hace uso de las anteriores.

**Diagrama de Contexto**



El usuario interactúa con la interfaz gráfica mediante el mouse de la computadora.

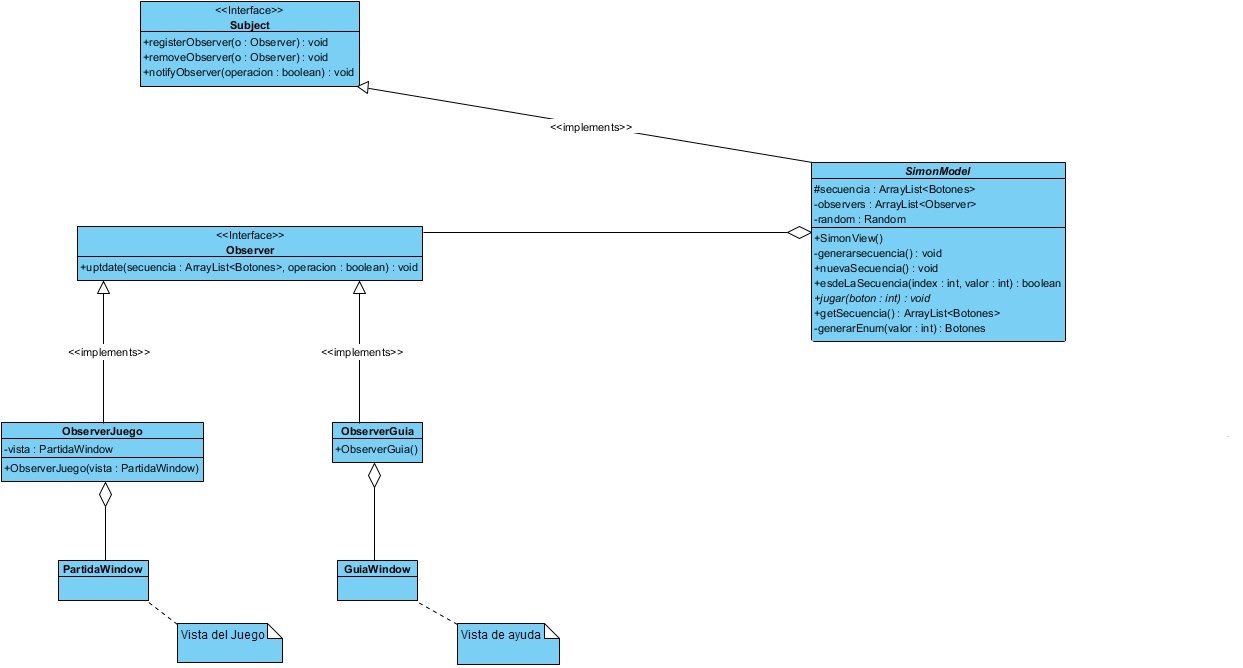
Los eventos de interacción son detectados y enviados a un sistema de procesamiento para actualizar los datos del sistema según la acción realizada.

El estado de la partida son todos los datos actuales del juego, y se actualiza cada vez que se detecta un evento y es procesado.

La interfaz gráfica es actualizada junto con el estado de la partida para reflejar los cambios.

**Diagrama de clases**

**Patron de diseño: Observer**



Este patrón sirve para representar de forma diferente el mismo conjunto de datos.

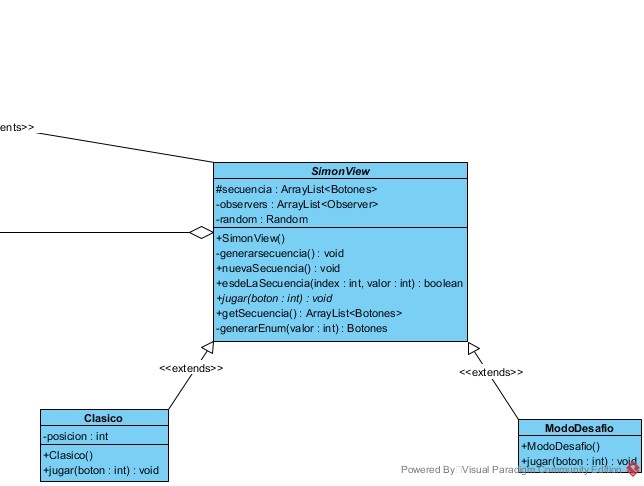
En la aplicación Simon Gear fue aplicado entre la Clase SimonModel (Modelo) y las clases PartidaWindow y GuiaWindow (vistas que representaran la secuencia de botones a seguir de formas diferentes).

A través del del método jugar, la clase SimonModel notificara a los observers que tenga registrados (una instancia de ObserverJuego y otra de ObserverGuia), podrán ocurrir dos cosas:

Si el usuario ingresó toda la secuencia de manera correcta, se mostrara en pantalla toda la secuencia más un elemento nuevo.

Si el usuario se equivoca, automáticamente aparecerá una pantalla indicando si quiere volver a jugar o prefiere volver al menú principal.

**Patrón de diseño: Strategy**



Este patrón sirve para realizar cambios de algoritmos en tiempo de ejecución.

En la aplicación Simon Gear se aplicó en la clase SimonModel, la cual cuenta con dos variantes diferente: la clase Clasico y la clase ModoDesafio.

El algoritmo cambiara según el modo de juego que el usuario elija en la vista principal al comienzo del juego.