Índice

[1. Nota de Entrega 2](#_Toc423264344)

[1.1 Estado 2](#_Toc423264345)

[1.2. Pass/Fail Ratio del sistema. 2](#_Toc423264346)

[1.3. Bugs conocidos. 2](#_Toc423264347)

[2. Manejo de las Configuraciones 2](#_Toc423264348)

[2.1. Herramienta de control de versiones. 2](#_Toc423264349)

[2.2. Esquema de directorios 3](#_Toc423264350)

[2.3. Plan de esquemas de ramas y políticas de mergeo. 3](#_Toc423264351)

[2.4. Realeases. 3](#_Toc423264352)

[2.5. Herramienta de seguimiento de Bugs. 4](#_Toc423264353)

[3. Requerimientos 4](#_Toc423264354)

[3.1. Diagrama de Casos de Uso. 4](#_Toc423264355)

[3.2. Requerimientos Funcionales 4](#_Toc423264356)

[3.2.1 HeartModel 4](#_Toc423264357)

[3.2.2 CarRaceModel 5](#_Toc423264358)

[3.2.3. Generales 5](#_Toc423264359)

[3.3. Requerimientos No Funcionales. 5](#_Toc423264360)

[3.4. Diagrama de arquitectura preliminar. 5](#_Toc423264361)

[3.5. Matriz de trazabilidad: entre casos de uso y requerimientos. 6](#_Toc423264362)

[4. Arquitectura. 7](#_Toc423264363)

[5. Diseño e Implementación. 7](#_Toc423264364)

[5.1. Diagrama de Estructura de Paquetes. 7](#_Toc423264365)

[5.2. Diagramas de Clases. 8](#_Toc423264366)

[5.2.1. Beat 8](#_Toc423264367)

[5.2.2. CarRace 8](#_Toc423264368)

[5.2.3. Heart 8](#_Toc423264369)

[5.2.4. Strategy 8](#_Toc423264370)

[6. Testing. 8](#_Toc423264371)

[6.1 Unit Tests 8](#_Toc423264372)

[7. Datos Históricos. 8](#_Toc423264373)

[8. Información adicional. 9](#_Toc423264374)

[8.1 Lecciones aprendidas. 9](#_Toc423264375)

# 1. Nota de Entrega

## 1.1 Estado

La presente nota pretende explicar el estado del proyecto al momento del realease. Se proporciona un archivo comprimido en formato “.zip” conteniendo la documentación del trabajo, el código fuente con su respectivo esquema de directorio y un archivo ejecutable. Para abrir el programa, entonces, solo basta con ejecutar dicho ejecutable proporcionado. La funcionalidad proporcionada por el sistema, en la versión a la fecha de entrega, puede resumirse en 6 opciones presentadas en el menú inicial del mismo:

* **Beat**: modelo del ritmo de una batería. Permite ajustar los pulsos por minuto en los que se emitirá el sonido.
* **Heart**: modelo de los latidos de un corazón, haciendo observable gráficamente la evolución del pulso del mismo (sin sonido).
* **Race**: modelo de una pista de autos. Permite ajustar la distancia que se sitúa el auto seleccionado al carril de carga de combustible, lo que hará que varíe el nivel de combustible y por ende también los BPM de acuerdo al nivel mencionado.
* **Todos**: permite ejecutar los tres simuladores al mismo tiempo.
* **CmbBox**: permite desde una misma ventana elegir el modelo que se quiere ejecutar.
* **Juego**: Juego CarRace. Donde se debe esquivar a los autos que vienen en contra sin dejar de lado el nivel de combustible para no perder, para lo cual se emitirá un sonido haciéndole saber que debe volver al carril de carga.

## 1.2. Pass/Fail Ratio del sistema.

## 1.3. Bugs conocidos.

Los bugs reconocidos a la fecha son:

# 2. Manejo de las Configuraciones

## 2.1. Herramienta de control de versiones.

Para el presente trabajo se decidió utilizar la herramienta Github, donde se configuró un repositorio para que cada integrante pudiera acceder a los elementos que en este se almacenan y poder así tener un seguimiento del trabajo.

Link del repositorio: <https://github.com/santivaderster/carRace2015>

Los integrantes del equipo son colaboradores, y Santiago Prados a la vez es el dueño del repositorio. Las operaciones relacionadas al control de versiones “commit”, “pull”, “push”, etc. son realizadas a través del IDE NetBeans con la extensión Git.

## 2.2. Esquema de directorios

El esquema de directorios que contiene el repositorio siguiente:

* CarRace2015:
  + bin:
  + src:
    - Adapters
    - Class
    - Controller
    - Imagenes
    - Liberíia
    - Model
    - Observer
    - TestDrive
    - View

La descripción de cada carpeta es:

* **bin:** donde se alojan los archivos compilados ejecutables.
* **src:** los archivos se encuentran segregados en los paquetes:
  + Adapters
  + Class
  + Controller
  + Imagenes
  + Liberíia
  + Model
  + Observer
  + TestDrive
  + View

## 2.3. Plan de esquemas de ramas y políticas de mergeo.

En un comienzo se optó por generar ramas por cada desarrollador, y luego hacer mergeos a medida que se necesitara. Luego de varios intentos fallidos, muchos relacionados con el IDE NetBeans, se decidio trabajar sobre una misma rama, y hacer commits y push poniendose de acuerdo sobre que clase desarrollaba cada uno, de esa manera se evitaban conflictos entre las revisiones propias y las alojadas en el repositorio.

## 2.4. Realeases.

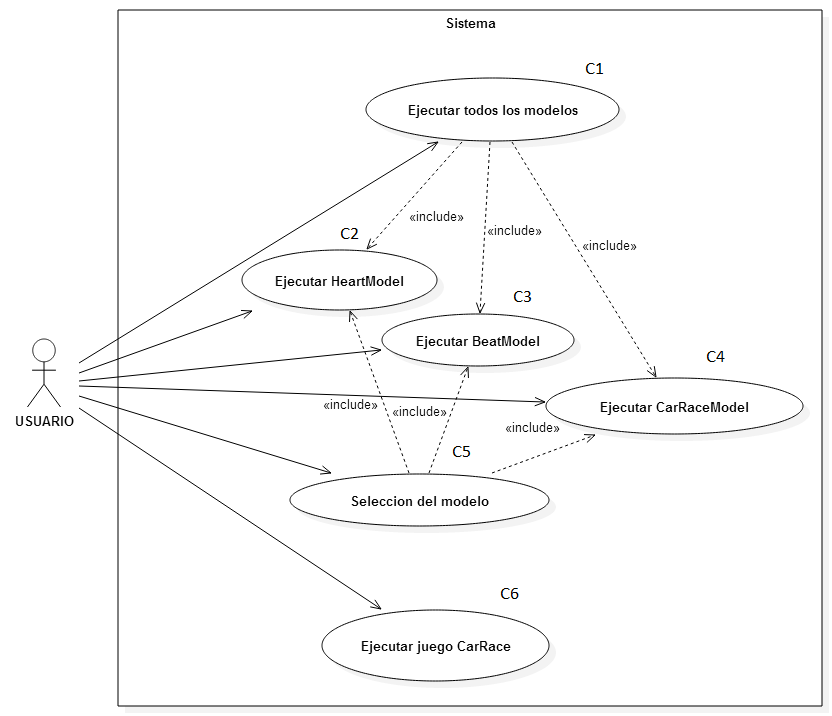
La última versión conseguir fue la considerada como realease, debido a ser la primera potencialmente entregable al final. Considerando a esta la versión 1.0 se espera generar nuevas versiones del proyecto para agregar nuevas funcionalidades, como la de un momento de ganador o una tabla de puntuaciones.

## 2.5. Herramienta de seguimiento de Bugs.

El seguimiento de Bugs se realizó de manera manual. Cada vez que un desarrollador encuentra una falla, intentará resolverlo por su cuenta. En caso de no ser posible, se deberá reportar el bug al grupo para lograr conseguir eliminarlo y proceder con la evolución del proyecto.

# 3. Requerimientos

## 3.1. Diagrama de Casos de Uso.



## 3.2. Requerimientos Funcionales

Esta sección lista los requerimientos funcionales de alto nivel. Los requerimientos se dividen en 3 categorias, correspondientes al HeartModel, al CarRaceModel y por último generales:

### 3.2.1 HeartModel

1. El HeartModel debe poder instanciarse sólo una vez. Esto se aplica utilizando el patrón de diseño Singleton.
2. La ventana de control del DJView debe ser extendida para “tratar” de generar nuevas instancias cada vez que se clickea en el botón “>>”.
3. La ventana de la BeatBar debe mostrar el número de intentos de creación de un nuevo HeartModel en lugar de los BPM originales.

### 3.2.2 CarRaceModel

1. El sistema debe poder utilizar la vista DJView con la ventana BeatBar, para representar el CarRaceModel.
2. Debe existir una vista que represente el escenario típico del CarRaceModel, esto es: una pista con 5 carriles, el auto seleccionado y autos viniendo en contra.
3. Se debe poder conducir el auto con las teclas izquierda y derecha.
4. En la esquina superior derecha deben visualizarse el nivel de combustible restante.
5. El combustible debe recargarse ubicándose en el carril izquierdo.
6. Se debe poder reiniciar el juego con la tecla F2, y oprimiento el boton reiniciar en la vista
7. Los autos contrarios deben crearse de forma aleatoria.
8. Al terminarse el combustible debe indicarse que se perdió el juego.
9. El chocar con un auto contrario debe indicarse que se perdió el juego.
10. El juego acaba tras conducir sin chocar por 2 minutos.
11. La velocidad de conducción debe aumentar cada 30 segundos.

### 3.2.3. Generales

1. Debe ser posible mostrar en paralelo los 3 modelos funcionando bajo la interfaz original.
2. El sistema debe brindar la posibilidad de cambiar en tiempo de ejecución el modelo utilizado a través de una extensión de funcionalidad en la ventana BeatBar.

## 3.3. Requerimientos No Funcionales.

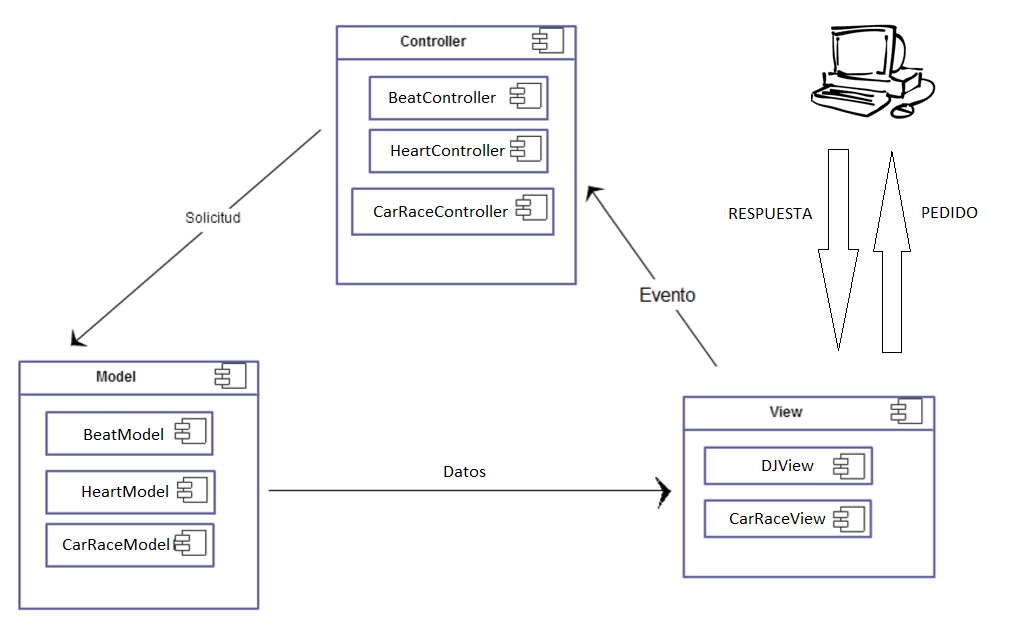
Un proyecto como el nuestro, no resulta ser uno por el cual se deban definir muchos requerimientos no funcionales.

**Facilidad de uso:** es cuán fácil o difícil es utilizar el sistema para el usuario al cual se destina el producto final. Por lo cual debemos realizar una vista amigable y fácil de comprender, y que el programa brinde diferentes opciones de control, para que la persona que lo utilice no se encuentre en duda si al presionar un botón, o hacer clic en uno, el programa no responda como se esperaría.

**Eficiencia:** Lo vemos importante, por el hecho que el nuestro modelo es una simulación muy simplificada de una carretera, por lo que debería poder ejecutarse incluso en PCs bastante básicas.

## 3.4. Diagrama de arquitectura preliminar.

En la se puede observar claramente que el sistema utilizaría el patrón de diseño Model View Controller. Debido a que el propósito final del sistema es servir al aprendizaje de dicho patrón y demás patrones de diseño, es que se elije implementar esta forma de arquitectura. De esta manera, el usuario interactuaría con la interfaz. El controlador interpreta los eventos solicitando al modelo algún cambio, que luego éste notificará nuevamente a la vista para su exposición al usuario.



## 3.5. Matriz de trazabilidad: entre casos de uso y requerimientos.

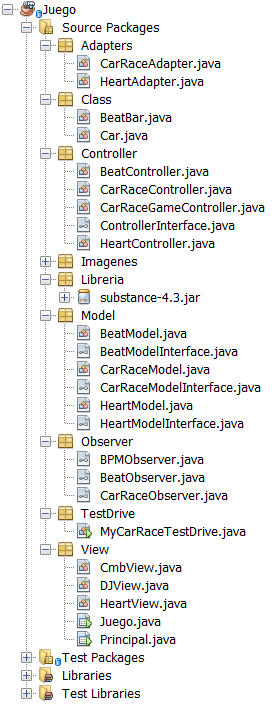
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| C1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## 4. Arquitectura.

Se utilizó el patrón de Arquitectura MVC, cuyo entendimiento constituye el principal objetivo de este proyecto. A través de la utilización de este patrón es posible separar el manejo de datos de un Modelo, de la visualización de estos en una Vista, entendiendo esta última como la manera en que los datos son mostrados. Esto permitió reutilizar una vista con varios modelos, cambiar la vista de un modelo, y así entender el vínculo entre estos dos componentes. Este vínculo es por definición el tercer componente, denominado Controlador. En resumen, la utilización de este patrón de arquitectura facilita la extensibilidad del software, lo cual es importante en este proyecto por ser este, la primera versión de lo que pretende ser una herramienta para el aprendizaje. La extensibilidad implica, en este caso, la posibilidad de reutilizar grandes bloques de código y hacer el código de fácil mantenimiento y depuración. Un diagrama genérico de esta arquitectura es el que sigue:

# 5. Diseño e Implementación.

## 5.1. Diagrama de Estructura de Paquetes.



## 5.2. Diagramas de Clases.

### 5.2.1. Beat

### 5.2.2. CarRace

### 5.2.3. Heart

### 5.2.4. Strategy

# 6. Testing.

## 6.1 Unit Tests

# 7. Datos Históricos.

Se implementó una tabla con la carga horaria aportada por los integrantes del equipo y el total de horas al momento de finalizar.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Integrantes |  |  |  |  |  | Total |
| Santiago |  |  |  |  |  |  |
| Ariel |  |  |  |  |  |  |
| Patricio |  |  |  |  |  |  |

# 8. Información adicional.

## 8.1 Lecciones aprendidas.

Durante el desarrollo de este Trabajo Final, se lograron comprender de mejor manera numerosas cuestiones sobre prácticas de Ingenieria de Software, y muchas otras sin llevar a la práctica.

También podemos rescatar como aprendizaje fundamental que nos dejó la materia, y es que no existe un procedimiento totalmente perfeccionado, que lleve a un software sin problemas, pero que las buenas prácticas y herramientas de seguimiento de proyectos sin duda abaratan costos y optimizan tiempos.

Puntos más importantes:

* Se aprendió a utilizar repositorios, aunque no se lograran superar los problemas que generaba el IDE NetBeans con el mergeo.
* Se obtuvo una buena práctica de programación, lo que ayuda a estimar de una mejor manera próximos proyectos y organizarse mejor en grupos.
* Afianzamos los conocimientos en diagramas UML.